



**Universidad de Huelva**
Escuela Técnica Superior de Ingeniería

Trabajo de Fin de Grado

# Detección de Noticias Falsas Mediante Técnicas de Deep Learning

Autor: **Álvaro Esteban Muñoz**

Tutores: **Jacinto Mata Vázquez**

**Victoria Pachón Álvarez**

**Huelva, Julio 2022**

**Resumen**

Vivimos en una época en la que estamos continuamente recibiendo información de muchísimas fuentes diferentes, el problema yace en que no todas esas fuentes de información son del todo fiables y muchas noticias son generadas solo para producir miedo, agitación o simplemente llamar la atención. Las noticias falsas han resultado ser un gran problema y su detección automática se ha convertido en una necesidad para la sociedad.

Con el auge de la inteligencia artificial, y en especial, de las redes neuronales, el campo del procesamiento del lenguaje natural ha crecido mucho, en gran parte, gracias al reciente desarrollo de unas arquitecturas de aprendizaje profundo conocidas como "transformers".

Este proyecto tiene, como objetivo principal, estudiar algunas de las técnicas de aprendizaje profundo más utilizadas hoy en día para la detección automática de noticias falsas, así como proponer nuevas estrategias para aumentar el rendimiento de éstas.

**Abstract**

We live in an era in which we are continuously receiving information from many different sources, the problem lies in the fact that not all of these sources are completely reliable and many news are generated only to produce fear, agitation or simply to attract attention. Fake news has proven to be a major problem problem and its automatic detection has become a necessity for they society.

With the rise of artificial intelligence, and in particular, neural networks, the field of natural language processing has grown significantly, thanks in large part to the recent development of deep learning architectures known as "transformers".

The project aims to study some of the deep learning techniques most widely used today for automatic detection of fake news, as well as to propose new strategies to improve the their performance.

# Índice

# Índice de tablas

# Índice de figuras

# Índice de códigos

# 1 Introducción

## 1.1 Definición de noticias falsas

Cuando hablamos de **noticias falsas**, también conocidas como *'fake news'* en inglés, nos referimos a aquella información engañosa que intenta aparentar ser una noticia. Estas suelen difundirse con propósitos fraudulentos, pero no siempre es así, muchas noticias falsas se difunden por el simple hecho de llamar la atención, provocar miedo, o difamar a personas o entidades del campo político. Si bien es verdad que las noticias falsas se originaron hace muchos años, es ahora, con el auge de las redes sociales en internet, que se está haciendo mucho más frecuente su difusión, convirtiendo lo que antiguamente era leer el periódico sentado tranquilamente en el sofá, en una tarea detectivesca para identificar si lo que estamos leyendo es realmente verdad o una tomadura de pelo.

## 1.2 Motivación

La elección del tema para este Trabajo de Fin de Grado, surge principalmente de dos factores: El primero, por el interés personal en el campo del *Machine Learning* y el *Natural Language Processing*. El segundo, por la propia realidad del problema. La inmensa cantidad de noticias falsas que se generan, sumado a la dificultad para distinguirlas del resto, ha hecho necesario el desarrollo de técnicas para su detección automática.

Las redes sociales en línea han adquirido una gran importancia en el ámbito de la información en los últimos años. Todos hemos visto como muchos periódicos han tenido que adaptar su formato, multitud de televisiones y diarios se han creado cuentas en diferentes redes sociales como **Twitter**, **Facebook** o **Instagram**. Las redes sociales en línea nos mantienen conectados entre nosotros todo el tiempo, creando un ambiente perfecto para la difusión de información, ya sea para bien o para mal. Las noticias falsas han encontrado en las redes sociales un terreno perfecto para expandirse.

El campo del *Machine Learning* y, concretamente, el del *Deep Learning*, están actualmente en auge, es por eso, que este tema se presentó como una oportunidad perfecta para mejorar, tanto a nivel académico, como profesional respecto a estos campos.

## 1.3 Objetivos

El siguiente proyecto tiene como objetivo plantear una serie de técnicas basadas en *Deep Learning*, que permitan automatizar su detección, así como proponer alguna mejora para las mismas. No obstante, para la consecución de dicho objetivo, es necesario plantear primero una serie de metas más concretas:

- Estudiar las técnicas más utilizadas hasta la fecha en el campo del *Deep Learning*, para la clasificación de noticias falsas en Internet.
- Comparar los modelos más utilizados para dicha clasificación.
- Elegir, de entre todos los modelos, aquellos que realicen la tarea de la forma más eficaz.
- Proponer y desarrollar algunas mejoras para incrementar el rendimiento de dichos modelos.
- Realizar un estudio en profundidad sobre la mejoría de las técnicas propuestas.

## 1.4 Estructura de la memoria

La memoria está organizada de la siguiente forma:

- El **Capítulo 1**, Introducción, presenta el tema de este TFG, así como qué motivó a su elección y realización. También se plantean los objetivos que se pretende conseguir y la forma en la que está estructurada el documento.
- En el **Capítulo 2**, Marco Teórico, se pretende contextualizar de forma teórica, todos los temas tratados en nuestro trabajo, para que cualquier persona que lea este documento, sea capaz de entenderlo de una manera más sencilla.
- En el **Capítulo 3** se realiza una descripción del problema, describiendo el dataset proporcionado y estableciendo la *baseline*.
- Los experimentos y resultados se exponen en el **Capítulo 4**. Se describe la *pipeline* seguida y el desarrollo de las mejoras realizadas para incrementar los resultados.
- Para finalizar, en el **Capítulo 5**, Conclusiones, se expone la consecución de objetivos y se plantea en qué medida son satisfactorios los resultados del trabajo, así como una opinión personal respecto al TFG y al proceso de su realización. Se concluye el documento describiendo posibilidades de trabajos futuros.

# 2 Marco Teórico

En este punto se introducen los conceptos teóricos necesarios para entender en qué campos nos hemos centrado a la hora de enfocar el proyecto. Estos han sido el aprendizaje automático, las redes neuronales y el aprendizaje profundo.

## 2.1 Aprendizaje automático

El aprendizaje automático es una rama de la inteligencia artificial que consiste en desarrollar algoritmos que permitan a las máquinas "aprender". Decimos que una máquina puede aprender cuando su desempeño para una tarea, mejora con la experiencia y mediante el uso de datos. En el campo del aprendizaje automático distinguimos tres familias de algoritmos: aprendizaje supervisado, no supervisado y reforzado [1].

**Aprendizaje Supervisado:** El aprendizaje supervisado hace uso de datos de los que conoce la salida esperada para cada entrada, a partir de los cuales puede crear un modelo capaz de predecir una salida dada una entrada. En particular, si se quisiera predecir la velocidad de vuelo de un ave dadas las medidas de sus alas, necesitaríamos un conjunto de datos de entrenamiento conformado por las medidas del ala de diferentes aves (entrada), así como su velocidad de vuelo (salida).

**Aprendizaje no supervisado:** El aprendizaje no supervisado, en cambio, no conoce la salida esperada, en su lugar, buscará patrones entre los datos para poder clasificar o separarlos entre ellos. Al igual que el ejemplo anterior, no podríamos predecir la velocidad de vuelo, pero si que podríamos clasificar el tamaño de las alas, obteniendo lo que podría ser una predicción del tipo de ave según su tamaño de ala.

**Aprendizaje reforzado:** El aprendizaje reforzado, al igual que el no supervisado, no conoce la salida esperada, sin embargo, sí que recibe retroalimentación. El modelo aprenderá a resolver la tarea mediante un sistema de **recompensas**, que le indicaran aquellas acciones necesarias para obtener la solución.

## 2.2 Redes neuronales

Probablemente una de las técnicas de clasificación más populares de los últimos años sean las redes neuronales artificiales. Estas proporcionan un modelo de computación que permite aproximar funciones reales. El motivo de su reciente éxito se debe a su

increíble eficacia en los campos del procesamiento del lenguaje natural y la visión por computador. Dicha eficacia viene dada por su capacidad para introducir no linealidades en las funciones, permitiendo así aproximar cualquier función real que nos permita clasificar con bastante precisión.[2]

Las redes neuronales artificiales están formadas por un conjunto de unidades de procesamiento denominadas **neuronas** (Figura 1), estas reciben una señal de entrada, sobre la que aplican una función, normalmente no lineal, y generan una señal de salida. Los enlaces que conectan las neuronas tienen un valor de peso, por el que se multiplica la señal que pasa a través de ellos. [3]

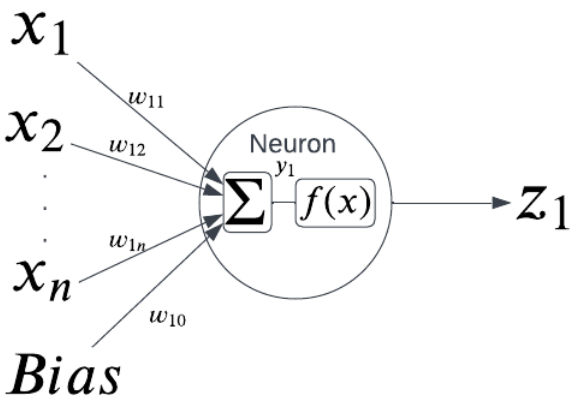


Figura 1. Diagrama de la neurona de una red neuronal

La eficacia de la red es evaluada mediante una función de error que se intenta minimizar aplicando cambios a los diferentes pesos que se encuentran en los enlaces de la red, este proceso se denomina "propagación hacia atrás"(Backpropagation).

### 2.2.1 Aprendizaje profundo

Cuando una red neuronal artificial está formada por más de tres capas, se cataloga como **profunda**. Las redes neuronales profundas han demostrado superar con creces a las redes neuronales superficiales (tres capas o menos), esto es debido a que una mayor cantidad de capas en la red, permite un mayor nivel de abstracción a la hora de representar los datos, pudiendo representar estructuras más complejas. Tomando

como ejemplo las famosas redes neuronales convolucionales, estas redes son capaces de aprender sobre el contenido de una imagen mediante el valor de los píxeles que se encuentran en ella, unas pocas capas en nuestra red nos permitirían diferenciar la paleta de colores que se usa para representar gatos y perros, no obstante, una mayor profundidad en la red nos permitiría llegar a detectar estructuras más complejas, tales como la forma de los ojos o el hocico. [4]

### 2.2.2 Aprendizaje por transferencia

El aprendizaje por transferencia es un método de aprendizaje automático que consiste en la reutilización de un modelo que ya ha sido entrenado para una tarea previa, como punto de comienzo para el desarrollo de un nuevo modelo en otra tarea diferente. Se usa el conocimiento almacenado para una tarea distinta, pero relacionada [5]. En resumen, usamos un modelo pre-entrenado para reducir el tiempo de entrenamiento en la tarea objetivo. [6]. En la Figura 2 podemos observar cómo se reutiliza el mismo modelo que ha sido previamente entrenado para reconocer imágenes de animales, para una tarea más específica, como reconocer imágenes de gatos.

### 2.2.3 Overfitting

Cuando un modelo es muy complejo, puede llegar a aprender demasiado bien resolver una tarea con un determinado conjunto de datos. Esto se convierte en un problema cuando queremos extrapolar el conocimiento aprendido a nuevos datos. Los resultados serán peores de los experados, ya que el modelo habrá aprendido muy bien a clasificar el conjunto de entrenamiento y no será capaz de generalizar. Existen algunas técnicas para evitar este problema. [4]

**Cross-validation.** Esta es una técnica que consiste en dividir el conjunto de entrenamiento en $k$ subconjuntos, luego entrenaremos el modelo $k$ veces usando $k-1$ conjuntos de entrenamiento y el que queda como conjunto de test. Tras ejecutar las $k$ iteraciones, se realiza la media aritmética de los resultados. Este método, aunque realmente eficaz, es bastante lento. [3]

**Early Stopping.** Entrenar modelos de forma iterativa nos permite saber a partir de que punto se empieza a producir el *overfitting*, una técnica para evitarlo consiste en parar el proceso cuando se alcanza dicho punto (ver Figura 3). [3]

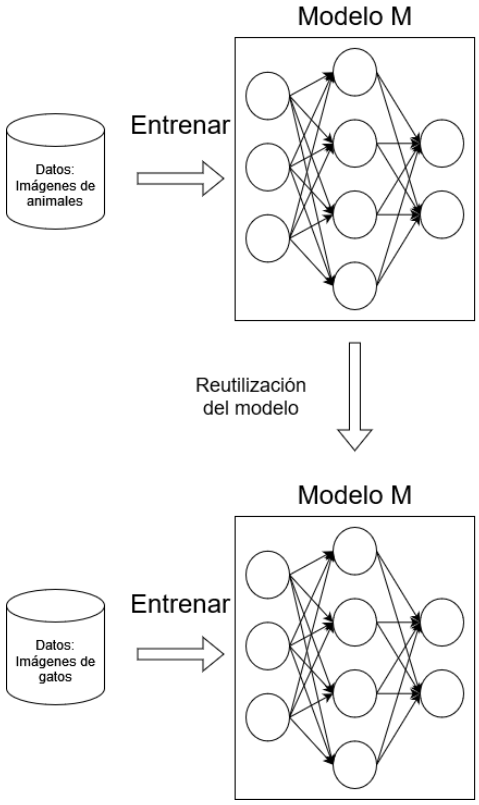


Figura 2. Diagrama del aprendizaje por transferencia

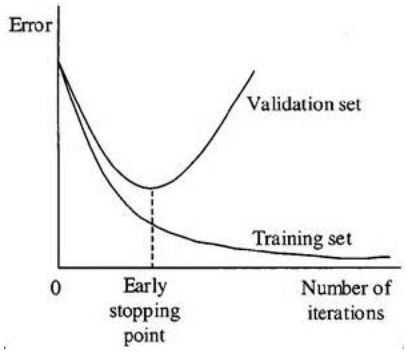


Figura 3. Aplicación de Early Stopping a un problema de overfitting

**Bagging.** El *bagging* es una técnica basada en *ensembles* de modelos, se entrenan varios modelos sin ningún tipo de restricciones y luego se juntan a todos para que sus predicciones se suavicen entre si. [7]

## 2.3 Procesamiento del lenguaje Natural

El procesamiento del lenguaje natural es un campo de la inteligencia artificial cuyo objetivo es hacer que las maquinas entiendan el contenido de documentos, así como su connotaciones contextuales [8]. Esta tarea ha sido enfrentada utilizando dos enfoques principales:

- **Programación de reglas formales:** Al igual que en el lenguaje formal, se busca definir una serie de reglas que permitan representar cómo se construye el lenguaje. Este enfoque está muy basado en el campo de la lingüística, pues utiliza reglas léxicas, sintácticas y semánticas para programar las reglas formales del lenguaje. [9]
- **Aprendizaje automático:** Gracias al aprendizaje automático, podemos abstraernos de la parte lingüística y centrarnos en programar una máquina que aprenda las reglas que definen el lenguaje de forma autónoma. Esta aproximación, no solo da muy buenos resultados, sino que además, se está volviendo cada vez más popular y exitosa gracias al auge de las redes neuronales profundas. [10]

### 2.3.1 Word Embeddings

El principal problema de utilizar redes neuronales para el procesamiento del lenguaje natural, es la necesidad de buscar una representación numérica para los datos. Para encontrar una representación que se ajuste a lo que buscamos empezaremos por definir la unidad a procesar. A esta unidad se la conoce como 'token'. Estos tokens pueden ser caracteres, palabras o n-gramas (conjuntos de caracteres). Para que nuestra red pueda procesar los tokens, lo lógico sería pensar en asignar una etiqueta numérica a cada uno. Esto presenta un problema, pues la red asociaría una distancia geométrica a cada token, entendiendo por ejemplo que un token con la etiqueta '5' está a medio camino entre un token con la etiqueta '0' y otro con la etiqueta '10'. Una representación que soluciona este problema es la denominada 'one-hot encoding', que se basa en representar cada token como un vector con tantas componentes como tokens exista en el vocabulario, donde el elemento que represente a nuestro token estará marcado con un '1' y el resto de elementos estarán marcados con un '0', aportando así independencia lineal a todos los tokens de nuestro vocabulario (Figura 4) [11, 12].

El uso de esta representación significa que si nuestro vocabulario contiene 10.000 tokens, nuestros vectores tendrán 10.000 elementos. A efectos prácticos estamos des-

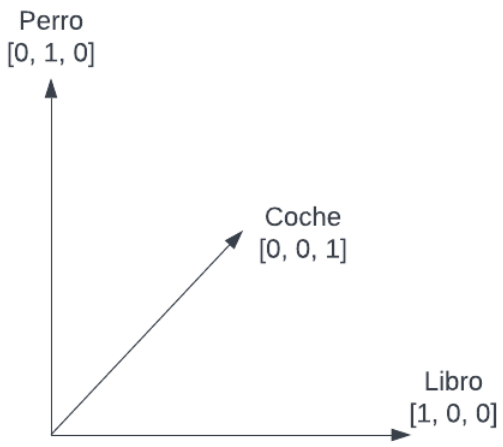


Figura 4. Espacio vectorial formado por one-hot encoding vectors

aprovechando mucho espacio puesto que 9.999 elementos serán 0 y solo un elemento será 1. Pero la compactación de información es una tarea que realizan por defecto las redes neuronales cuando procesan datos, precisamente por eso hacemos uso del aprendizaje por transferencia para usar modelos pre-entrenados en transformar vectores de 'one-hot encoding' en vectores compactados que representan de forma óptima todos los token de nuestro vocabulario. A estás representaciones compactas se las conoce como *'Word Embeddings'*

**Word2Vec.** Word2vec, publicado en 2013, es un algoritmo de NLP que utiliza redes neuronales para producir *word embeddings*. Utiliza dos tipos de arquitecturas diferentes [13]:

- CBOW (*Continuous Bag Of Words*): Usa una ventana de un tamaño determinado para predecir la palabra siguiente, es decir, predice en base al contexto.
- Skip Gram: Predice la ventana de palabras circundantes según una palabra dada.

Podemos observar la diferencia entre estas dos arquitecturas en la Figura 5.

**GloVe.** GloVe, lanzado un año después, es un algoritmo no supervisado con el mismo objetivo que Word2Vec. GloVe construye una matriz de probabilidades respecto

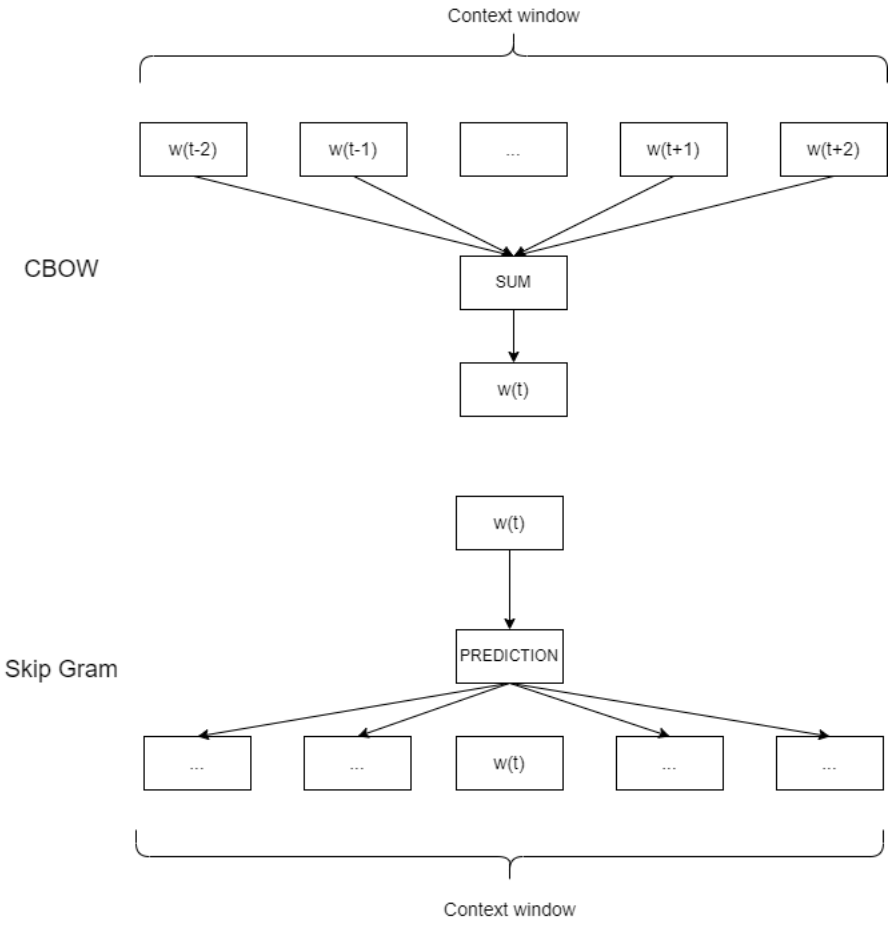


Figura 5. Arquitectura CBOW frente a arquitectura Skip Gram

a la relación semántica de una palabra con el resto (Figura 6). Esta probabilidad es indirectamente proporcional a la distancia lexicográfica entre ambas palabras [14]. Estos modelos son muy importantes ya que fueron los primeros en su campo, adquiriendo unos resultados significativos hasta el momento. Este estudio daría paso a los *Embeddings from Language Models*, más conocidos como **ELMo**

**ELMo** Los *Embeddings from Language Models* o *ELMo* son los más usado a día de hoy para la producción de embeddings. Estos modelos, a diferencia de *Word2Vec* o *GloVe*, son capaces de generar una representación diferente para palabras homónimas [15]. Para ello, se toman tokens a nivel de caracteres como entrada para una red LSTM bi-direccional, que se encarga de producir los *Word Embeddings*.

| Probabilities | | | |
|---|---|---|---|
| | k='gusta' | k='mucho' | k='el' |
| P(k \| gusta) | 0.5 | 0.75 | 0.02 |
| P(k \| mucho) | 0.75 | 0.5 | 0.02 |
| P(k \| el) | 0.23 | 0.03 | 0.5 |

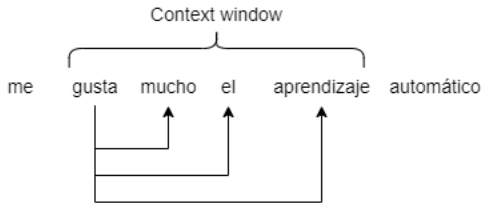


Figura 6. Funcionamietno de GloVe

### 2.3.2 Redes Neuronales Recurrentes

El procesamiento secuencial implica la necesidad de recordar resultados anteriores. Estos deberán ser aplicados como entrada en los siguientes procesamientos. De este concepto surgen las Redes Neuronales Recurrentes, también conocidas como RNN por sus siglas en inglés (*Recurrent Neural Networks*). Las RNNs procesan secuencias concatenando el resultado de la salida anterior a la entrada del siguiente elemento (Figura 7). Esto aporta al modelo una especie de "memoria".

Las RNNs ven reducida su eficacia cuando la secuencia es larga, pues comienzan a olvidar los estados procesados con mucha anterioridad. Este problema es conocido con **desvanecimiento de gradiente**. [16]

Las redes **LSTM** (Long-Short Term Memory), surgen como solución al problema del desvanecimiento de gradiente. Estas añaden otro vector de estado qué cada celda de procesamiento será capaz de leer, modificar o reiniciar, controlando así en qué medida olvidar o recordar ciertos estados. [17]

### 2.3.3 Transformadores (Transformers)

Aunque a primera instancia puede parecer que las redes neuronales recurrentes son el modelo perfecto para el procesamiento del lenguaje natural, éstas se enfrentan a un problema muy importante. Cuando la secuencia de texto es muy larga, los tokens

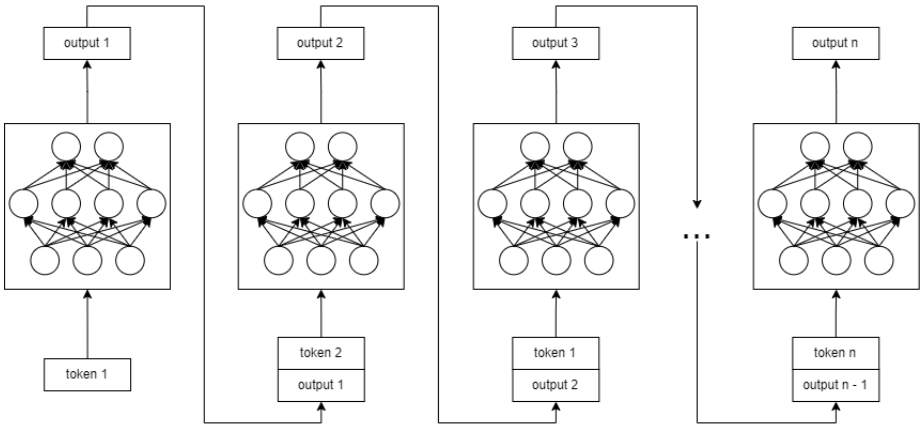


Figura 7. Esquema general de una red neuronal recurrente.

que se procesaron al principio de la secuencia pierden mucha importancia, es decir, las redes neuronales recurrentes tienen un problema de falta de memoria. La solución a la que se terminó llegando fue usar 'mecanismos de atención' para dar importancia a aquellos tokens realmente importantes. Estos mecanismos se pueden observar ya en los modelos NMT (Neural Machine Translation) [18].

**Mecanismos de atención.** Los mecanismos de atención se pueden definir como funciones que mapean una consulta y un conjunto de pares clave-valor con una salida, entendiendo todos estos elementos como vectores. Para obtener la salida se deberá realizar una suma ponderada de los vectores valor, donde el peso de cada vector es determinado mediante una función de compatibilidad entre la consulta y la clave del valor correspondiente. Cabe decir que para obtener el vector valor procesaremos el embedding del token por nuestra red. La función de compatibilidad no es nada más que el producto escalar entre el vector clave y el vector consulta (ver Figura 8) [18, 19].

Siendo un poco más concretos, los vectores consulta y clave no son más que dos representaciones vectoriales diferentes de cada token de nuestro texto, una representación obtenida mediante un red que genera vectores consulta y otra que genera vectores clave, es decir, estamos aplicando la función de atención para cada token con el resto de tokens del texto, obteniendo así, un vector conocido como **vector de atención**, que nos indica la importancia que tiene cada token, para el resto de tokens (ver Figura 9).

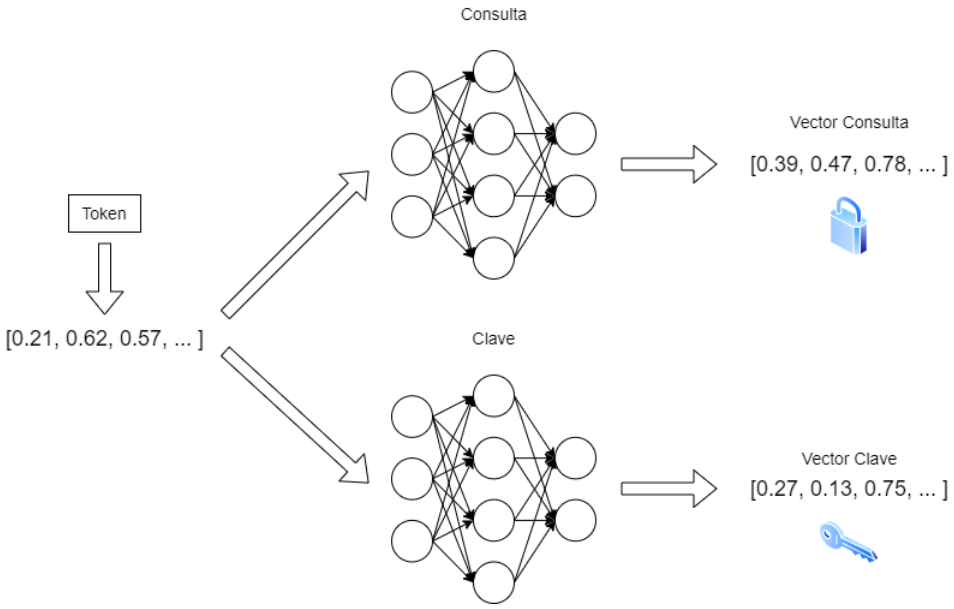


Figura 8. Generación de los vectores consulta y clave

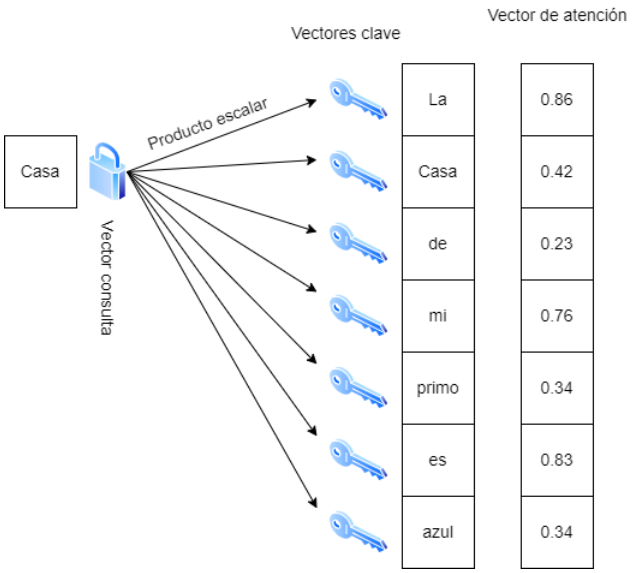


Figura 9. Generación del vector de atención para el token 'Casa'

**Codificación posicional.** Los mecanismos de atención y las RNNs persiguen el mismo objetivo, encontrar las relaciones latentes entre los token de la secuencia.

Conservar ambos mecanismos aportarían demasiada complejidad al modelo, por eso se prescinde de las RNNs. Sin embargo, la omisión de la recurrencia tiene una contra parte, perdemos la posición de los token en el texto. Para solucionar esto, se presenta una forma de codificar la posición de cada token en el texto. [20]. En la Figura 10 podemos observar tanto la codificación posicional, como el mecanismo de atención.

La codificación posicional se basa en fundamentos matemáticos, puesto que para cada token usaremos un número entero representado en binario. Así, el token número 3 será el 011 con 3 bits. Esta codificación se puede representar fácilmente mediante una ecuación matemática basada en frecuencias de onda:

$$PE = sin(pos/i) \tag{1}$$

Aunque en el artículo oficial la fórmula es un poco más compleja, al final se resume en este concepto. [21]

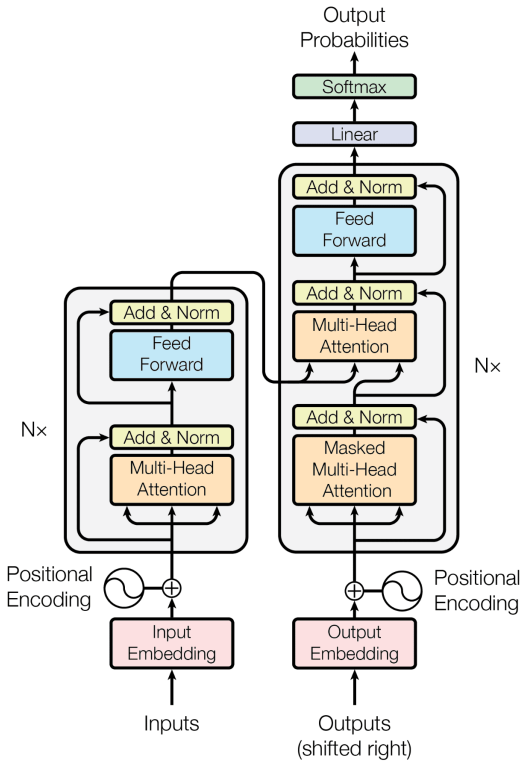


Figura 10. Arquitectura de un transformador

## 2.4 Ajuste fino (fine-tuning)

Una técnica muy utilizada en *deep learning* es el conocido como ajuste fino o *fine tuning*, en inglés. Esta técnica utiliza las propiedades del aprendizaje por transferencia, para acelerar el entrenamiento de grandes modelos en tareas concretas. Lo que se hace es entrenar modelos en tareas generales, como pueden ser, en el campo del NLP, el rellenado de máscara (*Fill Mask*) o predicción de la siguiente sentencia (*Next Sentence Prediction*). Una vez tenemos un modelo entrenado en una o más de las tareas mencionadas, se procede a re-entrenar el mismo modelo, con un corpus más pequeño, sobre una tarea más concreta, por ejemplo resumir o clasificar texto.

El ajuste fino se ha convertido en una tarea, no solo muy popular, sino necesaria. Entrenar un modelo de *deep learning* desde cero es una tarea que consume muchos recursos, se necesitan cantidades masivas de datos a las que no todo el mundo tiene acceso, y el entrenamiento tarda tiempos poco asumibles. Entrenar un modelo muy grande para una tarea concreta puede llegar a tardar meses, un tiempo poco razonable. Se convierte en algo mucho más eficiente el entrenar modelos grandes en tareas generales, para luego solo tener que concretar mediante ajuste fino, convirtiendo así, una tarea de meses, en horas.

## 2.5 Tecnologías utilizadas

Para el desarrollo del TFG se han utilizado las siguientes tecnologías:

### 2.5.1 Python

Hay bastantes lenguajes de programación preparados para el aprendizaje automático. Sin embargo, el más popular, y a su vez, el elegido para este trabajo, ha sido Python[1]. Existen gran cantidad de librerías y frameworks de aprendizaje automático para este lenguaje, por no hablar de que al ser tan popular, se dispone de mucha información online.

### 2.5.2 Google Colab

El entorno de desarrollo utilizado ha sido Google Colab[2]. Es un entorno online, pensado para el lenguaje Python y proporcionado por Google. La elección de este entorno ha venido motivada por su posibilidad para acelerar la ejecución del código,

[1] https://www.python.org/
[2] https://colab.research.google.com/

gracias a GPUs proporcionadas por la misma empresa. Además, nos permite tener todo en la nube, pudiendo continuar el mismo trabajo desde sitios diferentes, así como compartirlo con otros desarrolladores.

### 2.5.3 PyTorch

PyTorch[3] es una librería de aprendizaje automático de código abierto para Python y está basada en la orginalmente conocida como *Torch*. El punto fuerte de esta librería es su capacidad para operar con tensores de forma muy rápida, a través de GPUs, con lo que permite desarrollar de forma más sencilla, modelos de *deep learning* pensados para campos como el de la visión arificial o el procesamiento del lenguaje natural.

### 2.5.4 HuggingFace

HuggingFace[4] es una comunidad de programadores que ha desarrollado una serie de APIs que permiten descargar y entrenar modelos de aprendizaje automático pre-entrenados catalogados como *estado-del-arte*, de forma muy rápida y sencilla.

### 2.5.5 Weight and Biases

Weight & Biases[5] es una compañía que ha desarrollado una serie de herramientas para la visualización y colaboración en proyectos de aprendizaje automático. En el TFG hemos hecho uso de sus herramientas para visualizar el entrenamiento de nuestros modelos y poder entender como funcionan un poco mejor.

[3] https://pytorch.org/
[4] https://huggingface.co/
[5] https://wandb.ai/site

# 3 Planteamiento

Antes de comenzar a explicar los experimentos debemos tener muy claro el problema que estamos tratando y realizar una buena descripción del mismo. Esto nos permitirá determinar si se están cumpliendo los objetivos al finalizar el trabajo.

Estamos abordando el problema planteado en la competición de IberLEF de 2021, concretamente, la tarea de detección de noticias falsas en Español https://sites.google.com/view/fakedes/. Tal y como se cita en la página, *El objetivo planteado es decidir si una noticia es real o falsa analizando su representación textual.*

## 3.1 Descripción del Dataset

El Dataset proporcionado para la tarea es el "Spanish Fake News Corpus"[22], compuesto por un conjunto de 971 noticias extraídas de diversos medios de comunicación de México desde Enero hasta Julio de 2018. La longitud media de las noticias es de 369 palabras. Traduciendo esta medida a tokens obtenemos 546 tokens de media (utilizando el tokenizador de BERT), donde aproximadamente el 40 % son mayores de 512. Como podemos ver en la Tabla 1, el corpus viene dividido en 491 noticias reales y 480 falsas que comprenden 9 categorías diferentes; Ciencia, Deportes, Economía Educación, Entretenimiento, Política, Salud, Seguridad y Sociedad. El 30 % de las noticias se reservaron para crear el dataset de validación mientras que el restante 70 % se utilizó para el dataset de entrenamiento. Para evaluar los resultados de la competición, se utilizó un dataset de testeo compuesto por 572 nuevas noticias (ver Tabla 2), esta vez extraídas también de medios de comunicación fuera de México, como Argentina, Bolivia, Chile, Colombia, Costa Rica, Ecuador, Estados Unidos, Francia, Perú, Uruguay, Inglaterra y Venezuela. Este dataset de testeo también incluía noticias relacionadas con tópicos nuevos como 'COVID-19' o 'Internacional', los cuales no aparecen en el dataset de entrenamiento. Con estas características se pretendía poner a prueba la variabilidad de la temática y del lenguaje.

El contenido de las noticias dentro del dataset se encuentra dividido en las siguientes columnas:

- ID: Identificador de la noticia.
- Category: Etiqueta de la noticia, verdarea o falsa (true or fake)
- Topic: Temática de la noticia.

- Source: Nombre de la fuente de la que se ha extraído la noticia.
- Headline: Cabecera de la noticia.
- Text: Cuerpo de la noticia.
- Link: Enlace de la fuente.

| | Entrenamiento | Validación | |
|---|---|---|---|
| Verdaderas | 338 | 153 | 491 |
| Falsas | 338 | 142 | 480 |
| Total | 676 | 295 | 971 |

Tabla 1. Proporción de noticias del dataset de entrenamiento

| | |
|---|---|
| Verdaderas | 286 |
| Falsas | 286 |
| Total | 572 |

Tabla 2. Proporción de noticias del dataset de test

## 3.2 Métricas

Para la evaluación de modelos de aprendizaje automático se usa un variado conjunto de métricas. Claramente cuanta más información tenemos del funcionamiento de un modelo, más fácil es determinar si satisface los objetivos propuestos. Para definir las métricas necesitamos introducir varios conceptos primero:

**Matriz de confusión.** La matriz de confusión (Tabla 3) es una matriz en las que se nos indica el número de predicciones de una clase, frente al número de instancias reales de la misma. Los elementos de la matriz se pueden definir de la siguiente forma:

- *True positive:* Predicciones positivas, cuya instancia real coincide.
- *True negative:* Predicciones negativas, cuya instancia real coincide.
- *False positive:* Predicciones positivas, cuya instancia real es negativa.
- *False negative:* Predicciones negativas, cuya instancia real es positiva.

| | Prediction | |
|---|---|---|
| | Positives | Negatives |
| Positives | True positvies (TP) | False negatives (FN) |
| Negatives | False positives (FP) | True negatives (TN) |

Tabla 3. Matriz de confusión

Para la competición de Iberlef de 2021, los modelos se evaluaron usando la métrica *F1-Score*. Para calcular esta métrica necesitamos introducir primero las métricas *Precisión* y *Recall*. Para evaluar nuestro modelo usaremos también la *Accuracy*.

**Precision** La *precision* mide la cantidad de predicciones positivas, que son a su vez instancias positivas, frente al total de predicciones.

$$precision = \frac{TP}{TP + FP} \tag{2}$$

**Recall** La *recall* mide la cantidad de predicciones positivas, que son a su vez instancias positivas, frente al total de instancias positivas.

$$recall = \frac{TP}{TP + FN} \tag{3}$$

**F1-Score** La *f1-Score* es una medida que combina las dos métricas anteriores.

$$f1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{4}$$

**Accuracy** La *accuracy* mide la fracción de elementos que han sido correctamente clasifciados del total de elementos.

$$acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

## 3.3 Establecimiento de la linea base

Otro punto muy importante que debemos tener en cuenta a la hora de plantear el problema es el correcto establecimiento de una linea base, también conocida como *baseline*. Una *baseline* bien planteada nos permitirá evaluar correctamente los resultados de los experimentos, pudiendo determinar de forma eficaz la mejoría de los mismos respecto a un punto de partida.

Puesto que ya conocemos los resultados de la competición del año 2021 (Tabla 4), podemos establecer los resultados del ganador de dicha competición como *baseline* [23]. Todas las mejoras que se consigan a partir de esos resultados indicarán que las estrategias planteadas y desarrolladas son eficaces.

| | Validation Set | | Test Set |
|---|---|---|---|
| Model | Accuracy | F1-Score | F1-Score |
| bert-base-spanish-wwm-cased | 86.44 | 85.07 | 76.66 |

Tabla 4. Resultados obtenidos por el ganador de la competición 2021

# 4 Experimentación y resultados

Una vez tenemos bien claro el problema al que nos estamos enfrentando, es hora de mostrar en qué van a consistir nuestros experimentos, es decir, como hemos abordado el problema, así como los resultados obtenidos.

## 4.1 Descripción de los modelos

A continuación se describen los modelos usados para la experimentación del proyecto. Todos son modelos pre-entrenados sobre los que hemos aplicado la técnica de ajuste fino, es decir, han sido re-entrenados utilizando nuestro corpus.

**RoBERTa-large-bne:** RoBERTa-large-bne es un modelo de lenguaje basado en la arquitectura de los transformadores. Fue pre-entrenado usando el corpus en español más grande conocido hasta la fecha, con un total de 570GB de texto extraído por la Biblioteca Nacional de España desde 2009 hasta 2019 [24].

**bertin-roberta-base-spanish:** bertin-roberta-base-spanish es un modelo basado en BERT, entrenado desde cero en una porción de mC4 usando Flax [25, 26].

**bert-base-spanish-wwm-cased:** bert-base-spanish-wwm-cased es una versión de **BERT** entrenada en español. La versión *cased* tiene en cuenta las mayúsculas. Este modelo es además, el usado en la **baseline** [23, 27].

**bert-base-spanish-wwm-uncased:** bert-base-spanish-wwm-uncased es el mismo modelo que el anterior en su verisón *uncased*, es decir, no tiene en cuenta las mayúsculas [27].

**RuPERTa-base:** RuPERTa-base[6] es una versión de RoBERTa entrenada en el **big Spanish corpus** en su versión *uncased*, es decir, el corpus fue modificado para cambiar todos los caracteres en mayúscula a minúsculas.

## 4.2 Pipeline de experimentación

Para los experimentos, se ha diseñado una *pipeline*, presente en la Figura 11. Cada modelo se verá sometido a las diferentes fases que la conforman. Así determinaremos,

[6] https://huggingface.co/mrm8488/RuPERTa-base

mediante una serie de rankings, qué modelos son óptimos y con qué preprocesamiento aprende mejor a detectar noticias falsas.

1. **Preprocesamiento básico**: Se preprocesan los datos de forma básica y se entrena a los modelos con dichos datos.
2. **Ranking básico**: Se establece un ranking de los modelos y se elige a los tres mejores que serán los que pasaran a la siguiente fase.
3. **Preprocesamientos alternativos**: Se preprocesan los datos de tres formas diferentes a la primera y se entrena a los modelos de la fase anterior con todos los preprocesamientos.
4. **Ranking múltiple**: Se establece un ranking de los modelos por cada preprocesamiento, de esta forma, tendremos una lista de que los mejores pares modelo-preprocesamiento.
5. **Mejoras**: Se intenta realizar una serie de modificaciones o mejoras que puedan aumentar la puntuación de los modelos óptimos de la fase anterior.

## 4.3 Preprocesamiento básico

Para el preprocesado básico del corpus se ha optado por realizar una extracción de una representación textual de la noticia mediante la concatenación de los campos más fundamentales de la misma. Los campos fundamentales de una noticia son los elementos más básicos de la misma, la fuente, el titular y el texto. Como resultado, la representación textual de la noticia queda como la concatenación de los campos **'Source'**, **'Healine'** y **'Text'**.

Trás la obtención de la representación textual se suelen aplicar algunos procesos para limpiar el texto. En nuestro caso, hemos pensado que toda la información del texto puede ser importante, por lo tanto, no realizamos ni *lowercasing* ni *stopword removal*. En la detección de noticias falsas es muy importante conservar todos los elementos ortográficos, por eso se evita modificar el texto lo máximo posible [10].

Con el texto limpio, es necesario tokenizarlo para obtener las unidades mínimas a procesar por la red. Para la tokenización se ha usado el *tokenizer* por defecto de cada modelo, realizando un truncamiento a la derecha o un *padding* cuando la longitud de la representación sea mayor o menor de 256 tokens. Los *tokenizer* también se encargan de obtener los *embeddings* de las noticias (ver Código 1).

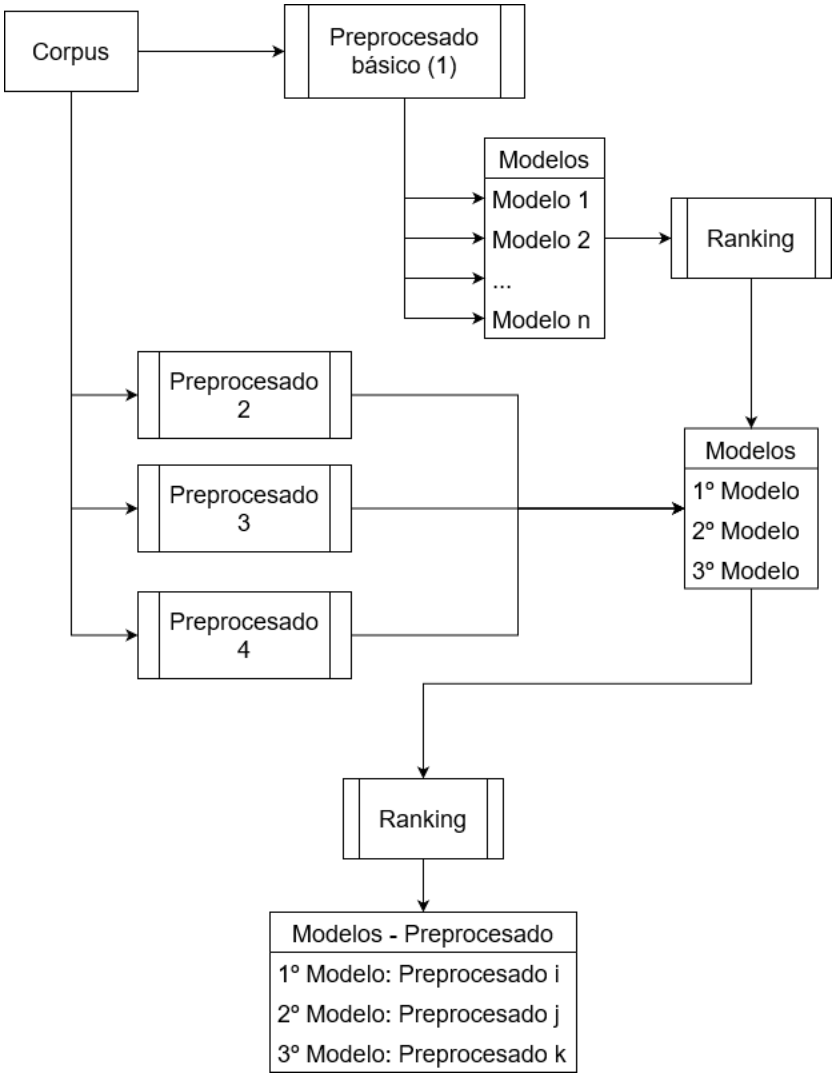


Figura 11. Pipeline de experimentación

Para entrenar los modelos, es necesario hacer una buena elección de los hiperparámetros, los cuales han variado relativamente poco. Se ha optado por un learning rate de $2e-5$, $1{,}75e-5$ y $1e-5$. Aunque hemos mantenido el número de epochs a 10, hemos hecho uso de la técnica de *early stopping* para evitar el *overfitting*. Las APIs de **HuggigFace** proporcionan una interfaz intuitiva, facilitando una compacta implementación del proceso de *fine-tuning*. La prueba de esto se puede observar en el bloque de Código 2.

```
# Función para concatenar los campos de la noticia
def concat_data(records):
    return {'Data': str(records['Source']) + '. ' +
                    str(records['Headline']) + '. ' +
                    str(records['Text'])
            }

# Tokenizar la representación textual de la noticia | TRUNCATION_LEN = 256
def tokenize_data(records):
    return tokenizer(records['Data'],
                         padding=True,
                         truncation=True,
                         max_length=TRUNCATION_LEN
                     )

# Mapea las funciones a cada fila del dataset
dataset = dataset.map(concat_data)
encoded_dataset = dataset.map(tokenize_data, batched=True)
```

Código 1. Código para el preprocesamiento básico de los datos

Para evaluar los resultados, la misma API permite realizar las predicciones (ver Código 3).

## 4.4 Ranking básico

Con los resultados de haber entrenado cada modelo, estos fueron ordenados en una tabla de mayor a menor *f1-score*. Podemos observar en la Tabla 5 que la mayoría de nuestros modelos quedan por encima de la *baseline* con una diferencia bastante alta.

En la Figura 12 podemos observar los problemas de *overfitting* resultantes de entrenar a **bertin-roberta-based-spanish**.

## 4.5 Preprocesamientos alternativos

La representación textual de las noticias aportada en el dataset es bastante más completa que la que hemos usado para el preprocesamiento básico. En este apartado

```
trainer = Trainer(
    # Modelo a entrenar
    model=model,
    # Hiperparámetros del modelo
    args=training_args,
    # Función para calcular las métricas
    compute_metrics=compute_metric,
    callbacks=[
        EarlyStoppingCallback(early_stopping_patience=3)
    ],
    train_dataset=dataset['train'],
    eval_dataset=dataset['valid'],
    tokenizer=tokenizer
)

trainer.train()
```

Código 2. Código para entrenar un modelo.

```
predictions = trainer.predict(test_dataset=dataset['test'])
print(predictions.metrics)
```

Código 3. Código para predecir a partir de un modelo entrenado.

| Ranking | Model | Accuracy | F1-Score |
|---|---|---|---|
| 1º | Roberta-large-bne | 0.84615 | 0.84452 |
| 2º | bertin-roberta-based-spanish | 0.85139 | 0.84403 |
| 3º | bert-base-spanish-wwm-uncased | 0.83391 | 0.82632 |
| 4º | BASELINE | - | 0.76660 |
| 5º | bert-base-spanish-wwm-cased | 0.74475 | 0.74475 |
| 6º | RuPERTa-base | 0.60489 | 0.64687 |

Tabla 5. Resultados del primer ranking

se propone añadir alguno de esos campos para construir una representación textual más rica que la utilizada en el apartado 4.3.

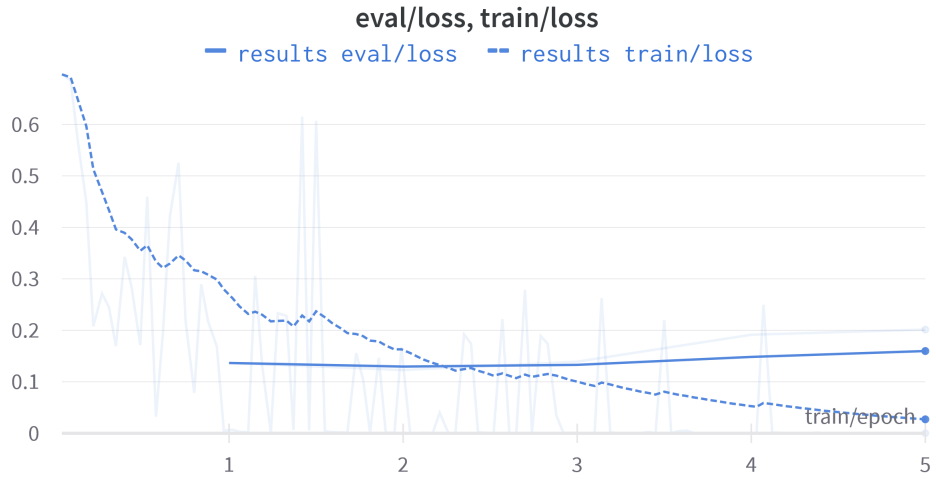


Figura 12. Overfitting en el 2º modelo

Analizando el corpus, llegamos a la conclusión de que el titular (*Headline*) y las dos primeras líneas del cuerpo (*Text*) son muy similares y quizá estén aportando información redundante. Por ello, hemos pensado que eliminarlo de algún preprocesamiento sería una buena idea. Otros campos interesantes que se nos proporcionan en el dataset son el enlace o *Link* y la temática de la noticia o *Topic*. Algún preprocesamiento tendrá también uno o los dos campos. Para resumir, las representaciones textuales alternativas que hemos diseñado son las siguientes:

- **Prepreocesamiento 2**: Source + Topic + Link + Text
- **Prepreocesamiento 3**: Source + Link + Text
- **Prepreocesamiento 4**: Source + Link + Headline + Text

Con los datos generados por cada preprocesamiento, procedimos a entrenar cada modelo pre-entrenado desde el principio, es decir, antes de ser entrenado por el preprocesamiento básico.

## 4.6 Ranking múltiple

Una vez entrenados nuestros modelos, obtuvimos un ranking por cada preprocesamiento. De esta forma, pudimos elegir el mejor preprocesamiento para cada modelo. En la Tabla 6 podemos ver los resultados de aplicar el preprocesamiento 2.

| Ranking | Model | Accuracy | F1-Score |
|---|---|---|---|
| 1º | bertin-roberta-based-spanish | 0.86188 | 0.86260 |
| 2º | Roberta-large-bne | 0.84615 | 0.85620 |
| 3º | bert-base-spanish-wwm-uncased | 0.80419 | 0.81935 |

Tabla 6. Ranking del preprocesamiento 2

Los resultados de aplicar el preprocesamiento 3 se muestran en la Tabla 7.

| Ranking | Model | Accuracy | F1-Score |
|---|---|---|---|
| 1º | Roberta-large-bne | 0.86713 | 0.86330 |
| 2º | bert-base-spanish-wwm-uncased | 0.79895 | 0.81717 |
| 3º | bertin-roberta-based-spanish | 0.82867 | 0.81081 |

Tabla 7. Ranking del preprocesamiento 3

Finalmente, en la Tabla 8 se pueden observar los resultados de haber entrenado con el corpus preprocesado con el preprocesamiento 4.

| Ranking | Model | Accuracy | F1-Score |
|---|---|---|---|
| 1º | bertin-roberta-based-spanish | 0.85489 | 0.85714 |
| 2º | bert-base-spanish-wwm-uncased | 0.80419 | 0.82389 |
| 3º | Roberta-large-bne | 0.80594 | 0.77755 |

Tabla 8. Ranking del preprocesamiento 4

Podemos ver que el mejor modelo de todos es **RoBERTa** con el corpus preprocesado con el preprocesamiento 3. Sin embargo, si realizamos un promedio de los resultados, podemos notar que el procesamiento que mejor funciona es el 2. También es importante destacar que, a pesar de ser el modelo mejor clasificado, **RoBERTa** se ve superado por **BERTIN** en los otros dos preprocesamientos con una puntuación bastante significativa.

En la gráfica que se muestra en la Figura 13 se puede observar cómo **BETO** siempre se encuentra por debajo de el resto de modelos, llegando a obtener un máximo de casi 0,95 de *f1-score* en el set de validación. Además, podemos notar que los resultados de BETO y RoBERTA se encuentran muy igualados en todo momento.

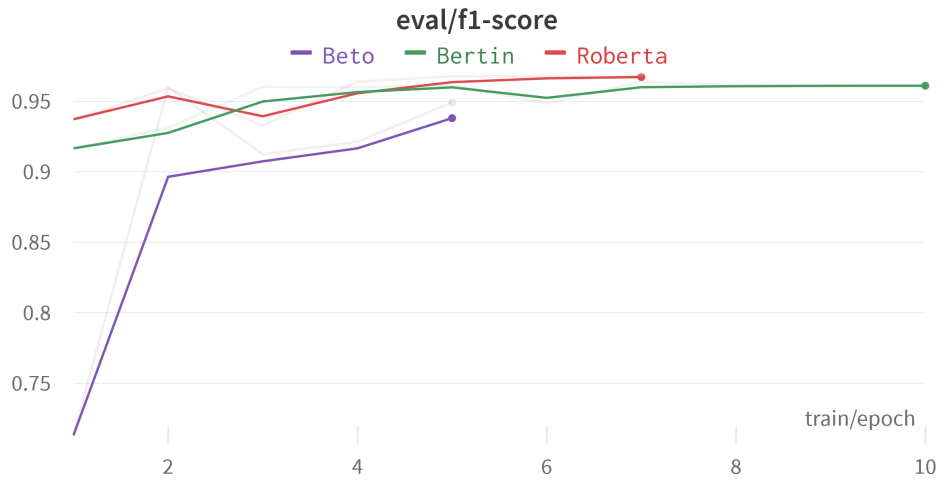


Figura 13. Evolución de la f1-score de los tres mejores modelos

## 4.7 Propuestas de mejoras

En esta sección se plantea el diseño y desarrollo de algunas mejoras que puedan incrementear el rendimiento de alguno o algunos de los modelos.

### 4.7.1 Sintetización del texto

Muchas veces las noticias se vuelven redundantes y cuentan de forma muy extensa cosas muy simples. Es por eso, que pensamos que podría ser una buena idea resumir el texto de las noticias, de forma que contuvieran la información más relevante de la misma. Para generar el resumen de las noticias se usó otro de los modelos de **HuggingFace**, Narrativa/bsc_roberta2roberta_shared-spanish-finetuned-mlsum-summarization. Para construir nuestra representación textual usamos el texto resumido en lugar del original, el resto del preprocesamiento no fue alterado. En la Tabla 9 se pueden observar los resultados del entrenamiento con el corpus una vez se aplicó su mejor preprocesamiento y el campo **"Text"** de las noticias resumido. Cabe destacar que los resultados, a pesar de ser bastante significativos, no son mejores que los obtenidos con anterioridad. Este hecho puede venir causado por una perdida de información al resumir las noticias.

| Ranking | Model (Preprocessing) | Accuracy | F1-Score |
|---|---|---|---|
| 1º | bertin-roberta-based-spanish (2) | 0.84440 | 0.84467 |
| 2º | Roberta-large-bne (3) | 0.81468 | 0.81849 |
| 3º | bert-base-spanish-wwm-uncased (1) | 0.82517 | 0.81684 |

Tabla 9. Ranking de los modelos con sintetización de texto

### 4.7.2 Head & Tail embeddings

Como se relata en el artículo ganador de la competición de *IberLeF* de 2021, entrenar usando los tokens tanto del principio como del final de la noticia, puede proporcionar mejores resultados [28, 29]. Es por eso que hemos decidido poner a prueba esta técnica, simplificándola un poco. Como podemos observar en la Figura 14, para la construcción del embedding que se le pasará al modelo, se ha tokenizado la noticia de dos formas; truncando a la derecha (para obtener el 'Head' embedding) y truncando a la izquierda (para obtener el 'Tail' embedding). Ambos resultados se concatenan entre sí para obtener el embedding final. Cabe destacar que tanto el 'Head' como el 'Tail' embedding tienen una longitud de 256 tokens, es decir, nuestro embedding final será de 512 tokens. Una vez más, el resto del preprocesamiento permanece invariable.

En la Tabla 10 podemos ver que esta técnica aporta mejores resultados que el resumen del texto. No obstante, estos resultados no terminan de proporcionar una gran mejoría respecto a los obtenidos previamente.

| Ranking | Model (Preprocessing) | Accuracy | F1-Score |
|---|---|---|---|
| 1º | bertin-roberta-based-spanish (2) | 0.86188 | 0.86260 |
| 2º | Roberta-large-bne (3) | 0.82867 | 0.83986 |
| 3º | bert-base-spanish-wwm-uncased (1) | 0.75699 | 0.77544 |

Tabla 10. Ranking de los modelos usando Head & Tail embeddings

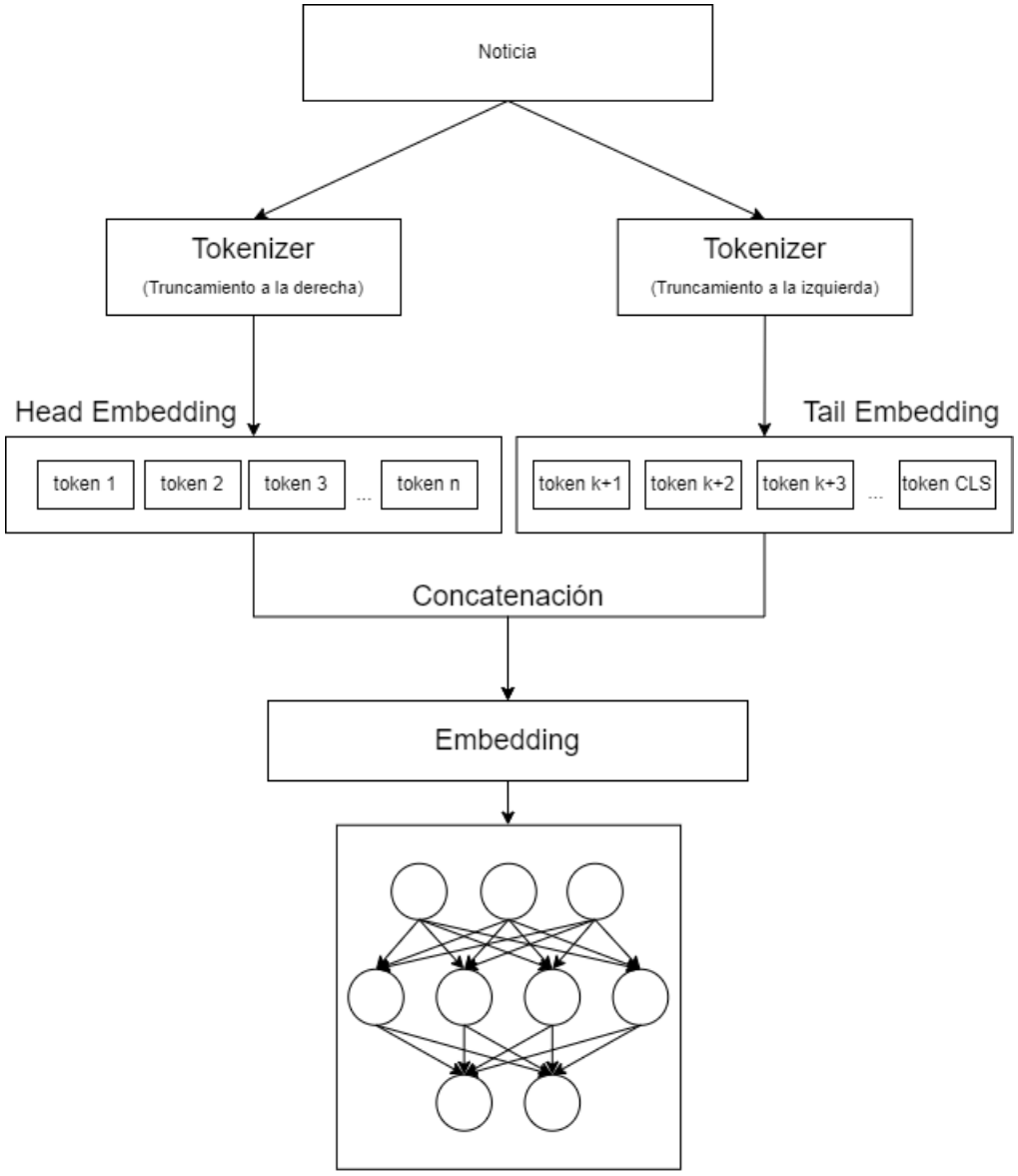


Figura 14. Diagrama de los embeddings 'Head' y 'Tail'

### 4.7.3 Ensembles de modelos

Hasta ahora, se ha observado que hay modelos que son más eficaces que otros. Sin embargo, un punto importante a tener en cuenta, es que dos modelos que clasifiquen

con una puntuación similar, pueden tener diferentes matrices de confusión. En otras palabras, unos modelos detectan mejor un tipo de noticias que otros. Dependiendo de la representación textual que le demos al modelo, este aprenderá unas características u otras.

Para hacer uso de las características detectadas por aquellos modelos que no obtienen buenos resultados medios, pero que aportan información relevante, se propone el diseño de una agregación de modelos que clasifique de forma conjunta. Mediante el uso de varios modelos diferentes, se puede establecer un sistema de sufragio, a través de el cual, los modelos elegidos clasifican la noticia, la clase mayoritaria será la elegida por el **ensemble** de modelos. Este ensemble es catalogado como "no ponderado", puesto que todos los modelos tienen la misma importancia (o peso) en la votación (Figura 15).

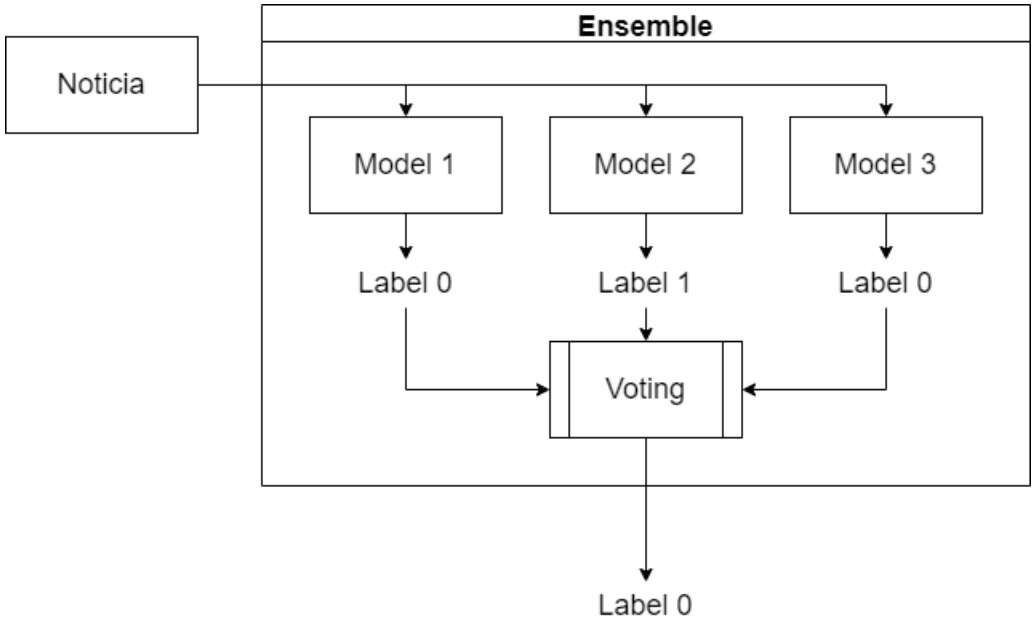


Figura 15. Funcionamiento de un ensemble de modelos

Se ha apostado por dos aproximaciones a la hora de elegir los modelos que forman parte de los ensembles:

1. Elegir los tres mejores modelos de todos los anteriores
2. Elegir el mejor modelo de todos los anteriores junto al mejor modelo para clasificar noticias verdaderas y el mejor modelo para clasificar noticias falsas.

Para saber el modelo que clasifica mejor las noticias verdaderas o falsas, hemos recurrido a la matriz de confusión (ver Tablas 11 y 12).

| | **Positive** | **Negative** |
|---|---|---|
| Positive | 264 | 22 |
| Negative | 76 | 210 |

Tabla 11. Matriz de confusión del modelo 2

| | **Positive** | **Negative** |
|---|---|---|
| Positive | 208 | 78 |
| Negative | 14 | 272 |

Tabla 12. Matriz de confusión del modelo 3

Los ensembles resultantes tras la elección de modelos efectuada siguiendo las estrategias planteadas han sido los siguientes:

- **Ensemble 1:** Aproximación 1
  - Modelo 1: Roberta-large-bne - Preprocesamiento 3
  - Modelo 2: bertin-roberta-based-spanish - Preprocesamiento 2
  - Modelo 3: bertin-roberta-based-spanish - Preprocesamiento 1
- **Ensemble 2:** Aproximación 2
  - Modelo 1: Roberta-large-bne - Preprocesamiento 3
  - Modelo 2: bertin-roberta-based-spanish - Preprocesamiento 3
  - Modelo 3: Roberta-large-bne - Preprocesamiento 1

En la Tabla 13, se pueden observar los resultados de los ensembles propuestos. Es importante remarcar que dichos resultados son los mejores obtenidos hasta el momento, con una puntuación de 0,87931, aproximadamente dos décimas más que el mejor modelo anterior.

| Ranking | Ensemble | Accuracy | F1-Score |
|---|---|---|---|
| 1º | Ensemble 1 | 0.87762 | 0.87931 |
| 2º | Ensemble 2 | 0.87762 | 0.87762 |

Tabla 13. Ranking de los ensembles

El principal problema de estos ensembles es que los tres modelos elegidos pueden no estar al mismo nivel. Si un modelo es bastante mejor que los otros dos, estos pueden generar un efecto contraproducente, reduciendo su efectividad en lugar de aumentarla. Una forma de corregir esto, es aplicar una serie de pesos a los modelos, pudiendo hacer a un modelo más importante que al resto.

El funcionamiento de este sistema es simple, cada modelo predice la etiqueta con una probabilidad que va de 0 a 1, de esta probabilidad se extrae una puntuación que irá de 0 a 1, donde una puntuación más cerca de 0, indicará una mayor probabilidad de la etiqueta 0 y viceversa. Con las puntuaciones de cada etiqueta, se realiza una suma ponderada y se redondea para saber cual será la etiqueta elegida por el ensemble (ver Figura 16).

Los pesos escogidos para cada ensemble han sido los siguientes:

- **Ensemble 1:**
  - $w_1$: 0.5
  - $w_2$: 0.3
  - $w_3$: 0.2
- **Ensemble 2:**
  - $w_1$: 0.5
  - $w_2$: 0.25
  - $w_3$: 0.25

La elección de los pesos se ha realizado con un poco de lógica, atribuyendo más peso a aquel modelo con la mejor puntuación y distribuyendo el resto de forma coherente. En el caso del **Ensemble 2**, apostamos por dar el mismo peso a ambos, para que se clasificaran las noticias de ambas clases de forma justa, no obstante, en los resultados de la Tabla 14, se puede ver que, aunque se hayan mejorado un poco los resultados

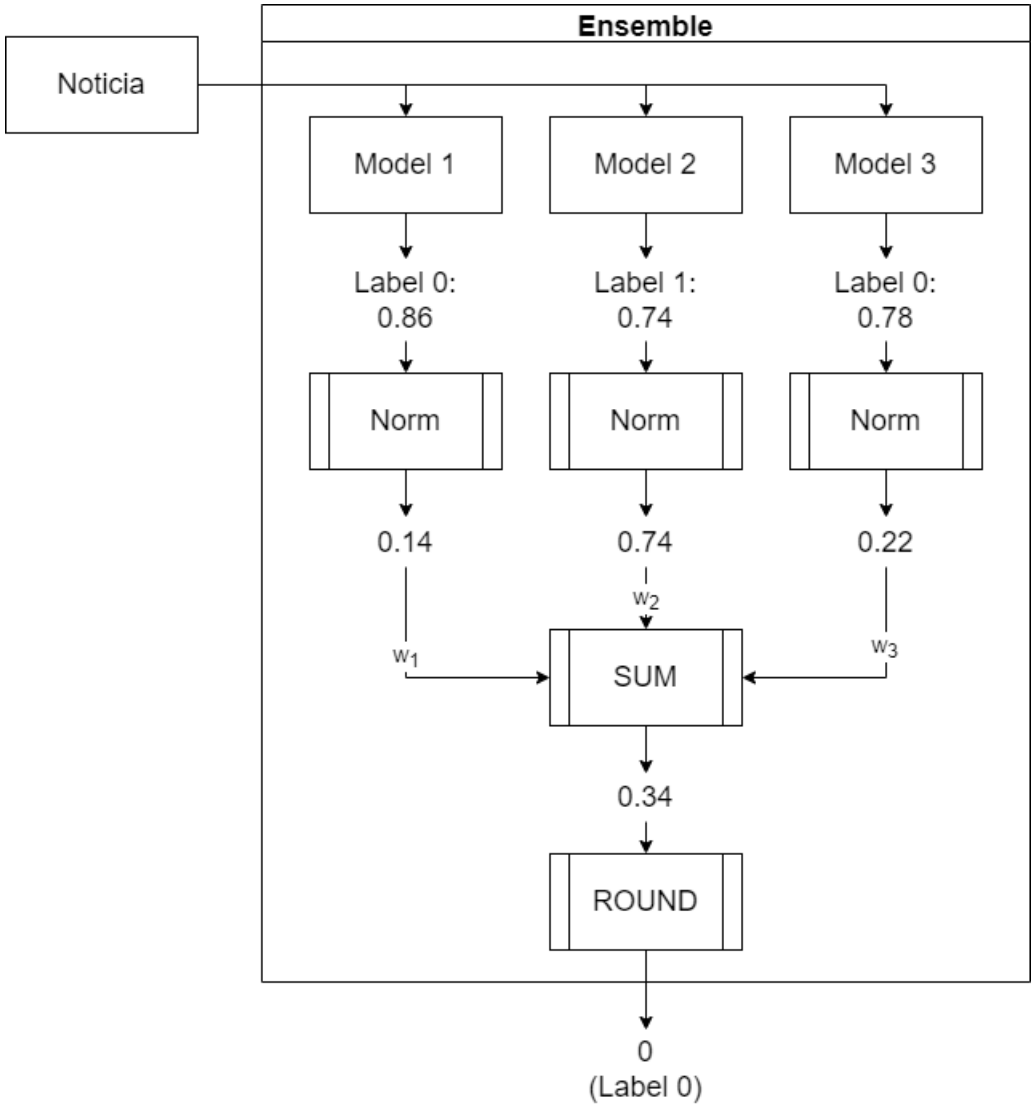


Figura 16. Funcionamiento de un ensemble de modelos ponderado

del mismo ensemble sin ponderar, no se han obtenido mejores resultados que con el **Ensemble 1**.

Como el **Ensemble 2** no dió los resultados esperados, decidimos variar un poco los pesos para ver si podemos aumentar su efectividad. Aunque es verdad que el modelo 2 clasifica mejor un tipo de noticias que el modelo 3, este sigue siendo bastante menos

| **Ranking** | **Ensemble** | **Accuracy** | **F1-Score** |
|---|---|---|---|
| 1º | Ensemble 1 | 0.88461 | 0.88421 |
| 2º | Ensemble 2 | 0.87937 | 0.87830 |

Tabla 14. Resultados de los ensembles ponderados

| **Model** | **Accuracy** | F1-Score |
|---|---|---|
| Ensemble 2 (Nuevo ponderamiento) | 0.88461 | 0.884210 |
| Ensemble 1 | 0.88461 | 0.884210 |

Tabla 15. Resultados del nuevo ponderamiento para el **Ensemble 2**

efectivo a nivel global que el modelo 3, con una f1-score de 0,85 para el modelo 3 frente a 0,81 del modelo 2. Para solucionar este problema, desbalanceamos un poco los pesos para que el modelo 3 recibiera más importancia que el modelo 2 ($w_2 = 0{,}15$, $w_3 = 0{,}35$), obteniendo los mismos resultados que con el **Ensemble 1**.

## 4.8 Resultados finales

Con todos los modelos obtenidos al aplicar las mejoras, se realizó un nuevo ranking para evaluar los resultados. El ranking final se puede observar en la siguiente tabla.

| Ranking | **Model (Preprocessing)** | **Accuracy** | F1-Score |
|---|---|---|---|
| 1º | Ensemble | 0.88461 | 0.88421 |
| 2º | Roberta-large-bne (3) | 0.86713 | 0.86330 |
| 3º | bertin-roberta-based-spanish (Head&Tail) | 0.86188 | 0.86260 |
| 4º | bertin-roberta-based-spanish (2) | 0.86188 | 0.86260 |
| 5º | bertin-roberta-based-spanish (4) | 0.85489 | 0.85714 |
| 6º | Roberta-large-bne (2) | 0.84615 | 0.85620 |
| 7º | bertin-roberta-based-spanish (Sum) | 0.84440 | 0.84467 |
| 8º | Roberta-large-bne (1) | 0.84615 | 0.84452 |
| 9º | bertin-robert-based-spanish (1) | 0.85139 | 0.84403 |
| 10º | Roberta-large-bne (Head&Tail) | 0.82867 | 0.83986 |
| ... | ... | ... | ... |
| - | BASELINE | - | 0.76660 |

Tabla 16. Resultados finales de todos los modelos

Como se puede ver en la tabla final, Tabla 16, nuestros experimentos propor-

cionan unos resultados que superan ampliamente la *baseline* proporcionada por la organización. Concretamente, nuestro mejor modelo fue el ensemble ponderado con una diferencia muy considerable sobre la *baseline*. A pesar de que el embedding *Head & Tail* aplicado a **BERTIN** proporciona notorios resultados, no es capaz de superar a **RoBERTa** con uno de los preprocesamientos alternativos. Es importante destacar que la mayoría de las posiciones del *top 5* son alcanzadas por **BERTIN**, siendo este el modelo más eficiente, pues es bastante más pequeño que RoBERTa y obtiene resultados muy notables. Para concluir, se debe mencionar que la principal mejora con respecto a la *baseline* es la elección de la representación textual. Mientras que para esta se eligió una representación basada solo en el campo **"Text"**, en los experimentos aquí aportados se ha optado por una representación más rica, compuesta por diferentes campos elegidos cuidadosamente.

# 5 Conclusiones

## 5.1 Consecución de objetivos

A lo largo del proyecto se han ido cumpliendo las metas propuestas. Para empezar, se han estudiado las técnicas más utilizadas en el campo del *Deep Learning* para la clasificación de noticias, como son los *Word Embeddings*, las redes neuronales recurrentes y la implementación de mecanismos de atención gracias a la arquitectura de los *Transformers*, sin olvidarnos de una de las técnicas más utilizadas, el *fine tuning*. También, se han descrito los modelos más utilizados para clasificación y se han comparado sus resultados tras la realización del *fine tuning* sobre los mismos. Para finalizar, no solo se han propuesto y desarrollado una serie de mejoras, sino que se han alcanzado resultados muy satisfactorios gracias a las mismas. Es por todo esto, por lo que se dan los objetivos de este trabajo como cumplidos.

## 5.2 Opinión personal

El desarrollo de este Trabajo de Fin de Grado me ha abierto la puerta a una nueva forma de trabajar en el campo del *deep learning* y del *natural language processing*, dado que gracias al mismo, he descubierto una gran cantidad de herramientas que me pueden ayudar en trabajos futuros relacionados con los mismos campos.

Cabe destacar, que aunque ya tuviera conocimientos previos respecto al tema tratado en el trabajo, este TFG me ha brindado la motivación necesaria para adentrarme en el campo del *deep learning* a nivel práctico, el conocimiento obtenido sobre las herramientas utilizadas es algo que, personalmente, valoro en gran medida, puesto que es una experiencia muy beneficiosa a nivel laboral. Por otro lado, la organización de un proyecto de este calibre, es algo que tiene merito, pues es una tarea que denota madurez y responsabilidad, por ello, considero que este trabajo, no solo me ha hecho crecer a nivel académico, sino personal

La detección de noticias falsas es una tarea compleja, a la par que interesante, este es otro de los puntos que han hecho que me sintiera atraído por este trabajo, lo que ha provocado unas ganas incansables de querer progresar. He disfrutado mucho del desarrollo, tanto de la parte de programación como la de investigación y confío en que esta no sea la última vez en la que me vea envuelto en un trabajo relacionado con este campo.

Para finalizar, quería dejar claro que opino que el campo de la detección de noticias falsas es algo muy importante en nuestra sociedad y que es algo que debe atraer más la preocupación de las empresas a cargo de redes sociales y medios de comunicación. Como ya se comentó en la introducción, estamos siendo bombardeados en todas partes con cantidades masivas de información y debemos tener cuidado que esta sea veraz y de calidad.

## 5.3 Trabajo futuro

Uno de los principales problemas actuales de los modelos de *deep learning* es la falta de datos en el entrenamiento. Para obtener un modelo capaz de predecir de forma eficaz, es necesario poseer una cantidad masiva de datos. El dataset proporcionado para este trabajo no es nada comparado con la cantidad de datos que han sido necesarios para entrenar a otros modelos en el mismo campo. La cuestión, ¿De dónde se sacan tantos datos? Ya se está estudiando una solución a este problema, las GANs, o *Generative Adversarial Networks*, donde dos redes neuronales compiten entre si, una genera datos que puedan engañar a otra red que se encarga de clasificar los datos recibidos con la etiqueta correcta, solucionando así, la falta de datos, pues estos son generados. Las GANs podrían ser un buen punto de partida para mejorar el estudio realizado en este TFG. Otro inconveniente de nuestro dataset, es su equilibrio, puesto que posee el mismo número de noticias de una clase que de la otra. Esto, aunque es la situación ideal, es poco frecuente, por lo que no da pie al estudio de técnicas de balanceamiento como el *undersampling*, el *oversampling* o técnicas de *data augmentation*, que podrían ser muy interesantes para trabajos futuros.

Todos los modelos tratados en este trabajo están basados en BERT. Un trabajo interesante podría abarcar también a los modelos recientemente lanzados de OpenAI, los modelos GPT. El único problema de estos modelos, es que no es totalmente gratis hacer uso de ellos, con lo que puede convertirse en un inconveniente para mucha gente.

# Referencias


[1] *Aprendizaje Automático*. Wikimedia Foundatin, Inc. 2022. URL: https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico.

[2] Tom M Mitchell y col. "Machine learning". En: (1997).

[3] Sergios Theodoridis. *Machine Learning: A Bayesian and Optimization Perspective*. 1st. USA: Academic Press, Inc., 2015. ISBN: 0128015225.

[4] Ian Goodfellow. *Deep learning*. eng. Cambridge, Mass, 2017.

[5] *Transfer Learning*. Wikimedia Foundatin, Inc. 2022. URL: https://en.wikipedia.org/wiki/Transfer_learning.

[6] Stevo Bozinovski. *Reminder of the First Paper on Transfer Learning in Neural Networks*. 2019. DOI: 10.31449/INF.v44i3.2828. URL: https://www.informatica.si/index.php/informatica/article/view/2828.

[7] Eric Bauer y Ron Kohavi. *An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants*. 1999. DOI: 10.1023/A:1007515423169. URL: https://doi.org/10.1023/A:1007515423169.

[8] *Natural language processing*. Wikimedia Foundatin, Inc. 2022. URL: https://en.wikipedia.org/wiki/Natural_language_processing.

[9] Roussanka Loukanova. *Natural Language Processing in Artificial Intelligence–NLPinAI 2020*. eng. Vol. 939. Studies in Computational Intelligence. Cham: Springer International Publishing AG, 2021. ISBN: 3030637867.

[10] *Natural Language Processing: Emerging Neural Approaches and Applications*. eng. 2022.

[11] Carlos Santana. *INTRO al Natural Language Processing (NLP) 1 - ¡De PALABRAS a VECTORES!* Youtube. 2020. URL: https://www.youtube.com/watch?v=Tg1MjMIVArc.

[12] Carlos Santana. *INTRO al Natural Language Processing (NLP) 2 - ¿Qué es un EMBEDDING?* Youtube. 2020. URL: https://www.youtube.com/watch?v=RkYuH_K7Fx4.

[13] Tomas Mikolov y col. *Efficient Estimation of Word Representations in Vector Space*. DOI: 10.48550/ARXIV.1301.3781. URL: https://arxiv.org/abs/1301.3781.

[14] Jeffrey Pennington, Richard Socher y Christopher Manning. "GloVe: Global Vectors for Word Representation". En: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, oct. de 2014, págs. 1532-1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162.

[15] Vidur Joshi, Matthew Peters y Mark Hopkins. *Extending a Parser to Distant Domains Using a Few Dozen Partially Annotated Examples*. 2018. DOI: 10.48550/ARXIV.1805.06556. URL: https://arxiv.org/abs/1805.06556.

[16] Ilya Sutskever, Oriol Vinyals y Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". En: *arXiv e-prints*, arXiv:1409.3215 (sep. de 2014), arXiv:1409.3215. arXiv: 1409.3215 [cs.CL].

[17] Sepp Hochreiter y Jürgen Schmidhuber. "Long Short-term Memory". En: *Neural computation* 9 (dic. de 1997), págs. 1735-80. DOI: 10.1162/neco.1997.9.8.1735.

[18] Yoon Kim y col. *Structured Attention Networks*. 2017. DOI: 10.48550/ARXIV.1702.00887. URL: https://arxiv.org/abs/1702.00887.

[19] Carlos Santana. *Las REDES NEURONALES ahora prestan ATENCIÓN! - TRANSFORMERS ¿Cómo funcionan?* Youtube. 2021. URL: https://www.youtube.com/watch?v=aL-EmKuB078&t=545s.

[20] Ashish Vaswani y col. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[21] Carlos Santana. *¿Por qué estas REDES NEURONALES son tan POTENTES? — TRANSFORMERS Parte 2*. Youtube. 2021. URL: https://www.youtube.com/watch?v=xi94v_jl26U.

[22] H. Gómez-Adorno y col. *Overview of FakeDeS at IberLEF 2021: Fake News Detection in Spanish Shared Task*. 2021.

[23] Xixuan Huang, Jieying Xiong y Shengyi Jiang. "GDUF$_D$*MatFakeDeS*2021 : *SpanishFakeNewsDetectionwithBERTandSampleMemory*". En: (2021). URL: http://ceur-ws.org/Vol-2943/fakedes_paper4.pdf.

[24] Asier Gutiérrez-Fandiño y col. "MarIA: Spanish Language Models". En: *Procesamiento del Lenguaje Natural* 68.0 (2022), págs. 39-60. ISSN: 1989-7553. URL: http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/6405.

[25] Javier De la Rosa y Eduardo G. Ponferrada y Manu Romero y Paulo Villegas y Pablo González de Prado Salas y María Grandury. "BERTIN: Efficient Pre-Training of a Spanish Language Model using Perplexity Sampling". En: *Procesamiento del Lenguaje Natural* 68.0 (2022), págs. 13-23. ISSN: 1989-7553. URL: http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/6403.

[26] Colin Raffel y col. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". En: *arXiv e-prints* (2019). arXiv: 1910.10683.

[27] José Cañete y col. "Spanish Pre-Trained BERT Model and Evaluation Data". En: *PML4DC at ICLR 2020*. 2020.

[28] Chi Sun y col. "How to Fine-Tune BERT for Text Classification?" En: *CoRR* abs/1905.05583 (2019). arXiv: 1905.05583. URL: http://arxiv.org/abs/1905.05583.

[29] Xixuan Huang, Jieying Xiong y Shengyi Jiang. "Spanish Fake News Detection with BERT and Sample Memory". En: (2021). URL: http://ceur-ws.org/Vol-2943/fakedes_paper4.pdf.