UNIVERSIDAD DE VALLADOLID

E.T.S.I. TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS ESPECÍFICAS DE TELECOMUNICACIÓN

Análisis de la respuesta emocional en redes sociales usando Técnicas de Aprendizaje Automático

Autor:
D. Miguel Ángel Arias Navarro
Tutor:
Dña. Noemí Merayo Álvarez

TÍTULO: **Análisis de la respuesta emocional en redes sociales usando Técnicas de Aprendizaje Automático**
AUTOR: **D. Miguel Ángel Arias Navarro**
TUTOR: **Dña. Noemí Merayo Álvarez**
DEPARTAMENTO: **Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL
PRESIDENTE:
SECRETARIO:
VOCAL:
SUPLENTE:
SUPLENTE:

FECHA:
CALIFICACIÓN:

Resumen de TFG

En el trabajo realizado se aborda la problemática de analizar las respuestas emocionales acaecidas en las redes sociales mediante técnicas de aprendizaje automático. El crecimiento exponencial sufrido por las plataformas de redes sociales ha vuelto crucial la necesidad de comprender las emociones que evocan los usuarios al escribir mensajes en la misma.

El objetivo de este TFG es realizar un estudio mediante el cual se analicen los sentimientos reflejados en mensajes de Twitter y de Twitch, ya que son dos de las redes sociales más utilizadas, y también puede servir para medir el nivel de *engagement* de los creadores de contenido de las mismas.

En primer lugar, se analizan los corpus con los que se va a trabajar, en este caso uno para Twitter y otro para Twitch, cada uno de ellos con un mensaje y sus correspondientes etiquetas de polaridades y sentimientos. Posteriormente, a los mensajes extraídos de los corpus se les realiza un procesado para eliminar información redundante y se normaliza la información. A continuación, dichos mensajes se pasan por un modelo de clasificación basado en Deep Learning, y en este punto se evalúa el rendimiento que presenta el modelo utilizando diferentes métricas de precisión.

Palabras clave

Inteligencia Artificial, Aprendizaje Automático, Procesamiento de Lenguaje Natural, Deep Learning, redes neuronales, redes sociales, respuesta emocional

Abstract

The work addresses the issue of analyzing emotional responses occurring on social media platforms using machine learning techniques. The exponential growth experienced by social media platforms has made it crucial to understand the emotions evoked by users when writing messages on these platforms.

The objective of this bachelor's thesis is to conduct a study in which the sentiments reflected in messages from Twitter and Twitch are analyzed, as they are two of the most widely used social media platforms. It can also serve to measure the level of engagement of content creators on these platforms.

First, the corpora to be worked with are analyzed, in this case, one for Twitter and another for Twitch, each of them containing a message and their corresponding polarity and sentiment labels. Subsequently, the messages extracted from the corpora undergo processing to eliminate redundant information, and the information is normalized. Next, these messages are passed through a classification model based on Deep Learning, and at this point, the performance is evaluated.

Keywords

AI (Artificial Intelligence), ML (Machine Learning), NLP (Natural Language Processing), DP (Deep Learning), NN (Neuronal Network), ER (Emotional Response)

Agradecimientos

Primero me gustaría agradecer a mis padres por todo el apoyo y por la educación que me dieron desde pequeño.

Me gustaría agradecer también a Noemí, que desde el principio me ha ayudado para que el proyecto saliese adelante.

Índice

Índice de figuras

Índice de tablas

Índice de ecuaciones

1

Introducción

1.1 Motivación

La razón principal que motivó la realización de este trabajo es la exploración sobre el potencial que tiene la inteligencia artificial aplicada al procesamiento de lenguaje natural.

A esto hay que sumarle el gran interés que ha despertado el mundo de las redes sociales, las cuales cada día toman más fuerza como medio de comunicación, y las convierte en las plataformas perfectas para analizar las respuestas emocionales de las personas ya que de ellos se obtiene una cantidad valiosa de datos.

Otro factor que motivó la realización de este proyecto es la repercusión de la inteligencia artificial para los creadores de contenidos en redes sociales, lo cual puede resultar de gran utilidad un estudio de estas características, puesto que les permite estudiar hasta cierto punto el nivel de *engagement* y conexión emocional con su audiencia.

Finalmente, hecho todo esto, se procederá a hacer una investigación sobre el análisis de sentimientos en las redes sociales Twitter y Twitch con el fin de modelar un algoritmo preciso y efectivo.

1.2 Objetivos

1.2.1 Objetivo Principal

El objetivo principal de este Trabajo de Fin de Grado es utilizar técnicas de Deep Learning para analizar la respuesta emocional, en concreto el nivel de polaridad y las emociones expresadas en mensajes de redes sociales, en concreto en Twitter y Twitch.

Se toma como punto de partida un algoritmo desarrollado previamente [1] en un TFG previo basado en técnicas de Deep Learning para aplicarlo a diferentes conjuntos de datos que pertenecen a diferentes redes sociales y temáticas distintas, para así analizar el comportamiento del algoritmo en diferentes contextos.

El objetivo final del trabajo será obtener un modelo de Deep Learning versátil y flexible que garantice alcanzar la mayor precisión posible a la hora de clasificar los mensajes según la polaridad o la emoción que expresa.

1.2.2 Objetivos Específicos

Durante este Trabajo de Fin de Grado se han desarrollado los objetivos específicos que se enumeran a continuación:

1. Análisis del estado del arte en técnicas de Deep Learning y redes neuronales.

2. Aplicación y optimización de un modelo de Deep Learning aplicado a un corpus de Twitter, en el cual se busca realizar un análisis para una correcta clasificación de emociones.

3. Aplicación y optimización de un modelo de Deep Learning aplicado a un corpus de Twitch centrado en el mundo de los videojuegos, en el cual se busca realizar un análisis para una correcta clasificación de polaridades y emociones.

4. Aplicación y optimización de un modelo de Deep Learning aplicado a un corpus de salud mental en redes sociales, en el cual se busca realizar un análisis para una correcta clasificación de polaridades y emociones.

1.3 Metodología del trabajo

La metodología seguida para el desarrollo de los objetivos ha constado fundamentalmente de las fases que se explicarán a continuación.

1.3.1 Fase de Documentación

Esta fase ha sido de una gran importancia en varios aspectos, entre ellos destacamos en primer lugar fundamentar el problema, es decir, ayudar a comprender la importancia

del análisis de emociones en redes sociales y, por otro lado, en la elección de las técnicas o enfoques que se pueden aplicar al realizar trabajos de aprendizaje automático.

1.3.2 Fase de análisis

Esta fase es crucial, puesto que ayuda a comprender en profundidad los datos y a extraer la información relevante. En esta fase se analizan los algoritmos previamente creados con su correspondiente documentación, y a su vez también se realiza un estudio sobre los hiperparámetros que confeccionan los modelos desarrollados. Estos modelos se entrenan y evalúan utilizando técnicas de validación cruzada.

Para la realización de esta fase se ha partido de los siguientes trabajos:

- TFG de Jesús Herrero Llanos [1].
- TFM de José Carlos Sobrino [2].
- TFM de Julia Isabel Medrano Sanz [3].
- TFM de Iván Arévalo Nuñez [4].

1.3.3 Fase de Pruebas

Fase en la cual se realizaron las pruebas con el objetivo de evaluar el rendimiento y la efectividad del modelo desarrollado. Durante la misma se realizaron a su vez, los ajustes de los hiperparámetros para optimizar el rendimiento en diferentes corpus (datasets) pertenecientes a diferentes temáticas y en diferentes redes sociales.

1.3.4 Fase de escritura del informe

Fase en la cual se ha redactado el Trabajo de Fin de Grado presente, y donde se plasman los resultados obtenidos durante todo el proceso.

1.4 Estructura de la Memoria

La memoria del proyecto está conformada por distintos capítulos en los cuales se recogen diversos objetivos, que se explicarán a continuación.

Capítulo 1: se introduce el problema a tratar, los objetivos que se busca conseguir y se explica la metodología que se va a utilizar para la resolución del problema.

Capítulo 2: se proporciona un estado del arte sobre la inteligencia artificial con el fin de poner en contexto al lector y que se facilite la comprensión del documento.

Capítulo 3: se exponen en el mismo las diversas herramientas software utilizadas en la realización del estudio y análisis.

Capítulo 4: se realiza el estudio de clasificación de polaridades para los corpus de Twitch y salud mental. También se comentan y analizan los resultados.

Capítulo 5: se realiza el estudio de clasificación de emociones para los corpus de Twitch, salud mental y Twitter. También se comentan y analizan los resultados de manera global, incluyendo también los de polaridades.

Capítulo 6: se exponen las conclusiones finales del estudio y se proponen líneas futuras de investigación.

2

Estado del arte

2.1 Inteligencia Artificial

Se entiende por inteligencia Artificial [5] un campo de la informática el cual busca crear y desarrollar sistemas y máquinas las cuales requerirían de inteligencia humana. Para esto se utilizarán algoritmos y técnicas de aprendizaje automático, consiguiendo así llevar a cabo varias tareas, entre ellas: procesamiento de gran cantidad de datos, reconocimiento de patrones o la toma de decisiones. El objetivo final es el desarrollo de sistemas que realicen las tareas encomendadas de manera eficiente y precisa incluso para situaciones complejas.

Desde sus inicios en la década de 1950 la IA ha mantenido un constante desarrollo impulsado por el crecimiento exponencial del poder de cómputo y el acceso a volúmenes de datos mayores [6].

Dentro de la Inteligencia Artificial se puede destacar la rama de Machine Learning [7], para el cual el aprendizaje se centra en el desarrollo de modelos y algoritmos que habiliten a la máquina para aprender mediante experiencia y con los datos proporcionados. Utiliza técnicas como pueden ser aprendizaje supervisado, no supervisado y por refuerzo.

La evolución de Machine Learning es lo que hoy en día se conoce como Deep Learning, el cual es un subconjunto del Machine Learning cuyo objetivo es una simulación del pensamiento humano más precisa utilizando redes neuronales. El proceso de estas técnicas es generar una configuración de los parámetros básicos en relación con los datos que se tienen, para posteriormente entrenar a la máquina con el fin de que, utilizando distintas capas de procesamiento, sea capaz de reconocer patrones. Para implementar Deep Learning los modelos de redes neuronales artificiales son comúnmente utilizados.

En resumen, el Machine Learning crea algoritmos que aprenden de los datos y toman decisiones basadas en patrones observados, por otro lado, Deep Learning utiliza modelos de redes neuronales con el fin de resolver los problemas planteados sin necesidad de la intervención humana. Por tanto, Deep Learning es una parte especializada de Machine Learning, la cual a su vez es un subconjunto de la Inteligencia Artificial [8].

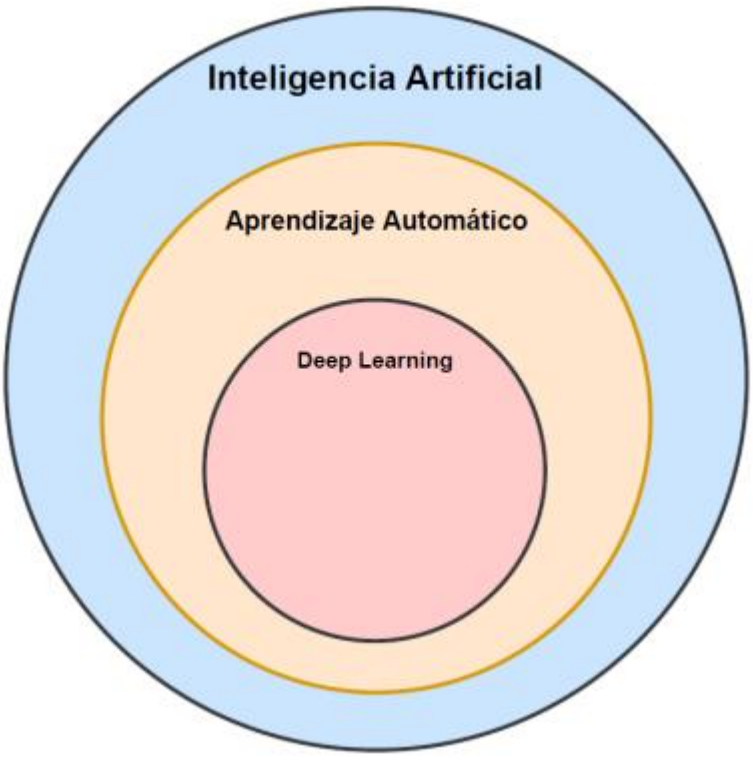


Figura 1: Explicación categorías

2.2 Comparativa entre Deep Learning y Machine Learning

Como bien se ha explicado antes, tanto Deep Learning como Machine Learning son dos subcampos de la Inteligencia Artificial, los cuales se centran en el procesamiento de datos y el aprendizaje automático, aunque tienen una diferencia principal, el hecho de que Machine Learning no requiere de una intervención humana. A continuación, se muestra una comparativa más exhaustiva [9]:

- Definición: Machine Learning por definición es la capacitación de algoritmos para distinguir y aprender patrones a partir de los datos dados y haciendo uso de estos datos tomar decisiones sin una programación explícita. Mientras, Deep Learning se basa en el uso de redes neuronales artificiales para la extracción de características y la toma de decisiones.

- Representación de los datos: en Machine learning las características de los datos se seleccionan de manera manual y se utilizan como entrada para los algoritmos de aprendizaje, mientras que en Deep Learning los datos se utilizan directamente como entrada en las redes neuronales profundas.

- Capacidad de aprendizaje: Machine Learning requiere de una selección manual de las características relevantes, mientras que Deep Learning realiza dicha selección de manera automática.

- Entrenamiento: Machine Learning se entrena mediante la CPU, a diferencia de Deep Learning, la cual se entrena mediante la GPU, necesitando este último una mayor cantidad de datos para procesar para aprovechar al máximo sus capacidades.

- Escala y complejidad: Machine Learning ha demostrado funcionar bien para casos en los que el tamaño de datos y la complejidad del problema son moderados, mientras que Deep Learning destaca para casos en los que tanto el volumen de datos como la complejidad del problema a resolver son elevados, como es el caso del procesamiento de lenguaje natural.

En la Figura 2 se muestra un ejemplo de lo que se comentaba previamente.

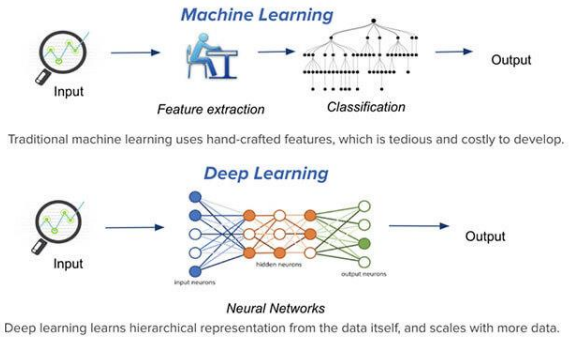


Figura 2: Machine Learning vs Deep Learning

2.3 Origen del Deep Learning

A medida que se investigaban formas de replicar la inteligencia humana mediante ordenadores iban surgiendo las primeras ideas y conceptos que formarían los cimientos de lo que hoy en día se conoce como Deep Learning. Entre los acontecimientos más destacables podemos encontrar los siguientes [10] [11]:

- En 1943 Warren McCulloch y Walter Pitts desarrollan la "neurona de McCulloh-Pitts", con el fin de simular una neurona biológica, la cual, si bien no poseía un mecanismo de aprendizaje, iba a sentar las bases para que posteriormente se estudiasen las redes neuronales.

- En 1957 Frank Rosenblatt crearía el perceptrón, un modelo de aprendizaje capaz de realizar clasificación binaria y, además, poseía un mecanismo de aprendizaje. Este invento potenció la revolución de las redes neuronales, aunque esta misma no tardaría en estancarse debido a las limitaciones propias de la época. Este periodo se conoce como el "invierno de las redes neuronales".

- En 1965 Alexey Ivakhenko creó el primer modelo de perceptrón multicapa, considera el precursor del Deep Learning. Sin embargo, en la década de los 70 se centraría en la investigación, dando lugar al algoritmo de propagación hacia atrás (backpropagation) el cual se implementa en código ordenador.

- En 1980 Kunihiko Fukushima creó el Neocognitrón, el cual se basaba en una arquitectura de red neuronal convolucional, algo que hasta la fecha no se había implementado, y con el que se podía reconocer patrones visuales, lo cual resultaría de gran ayuda para aplicar Deep Learning a problemas de visión por computadora.

- En 1982 John Hopfield creó la primera red neuronal recurrente, la cual resultaría fundamental para comprensión de la memoria a largo plazo y sentó las bases para modelos de Deep Learning posteriores.

- En 1986 se da inicio al Deep Learning propiamente dicho. Sejnowski desarrolla NeTalk, la cual es una red neuronal que puede pronunciar

palabras en inglés a través de transcripciones telefónicas. A su vez, el algoritmo backpropagation se implementó con éxito en redes neuronales, haciendo que el entrenamiento de redes neuronales complejas se facilitase.

- En 1989 Yann LeCun creó LeNet, una red convolucional que utilizaba el algoritmo backpropagation, con la cual lograría reconocer dígitos escritos a mano de manera precisa. Esto supuso un avance significativo en el uso de Deep Learning para tareas de visión por computadora.

- En 1997 Hochreiter revolucionaría los modelos de Deep Learning con la introducción de la red neuronal recurrente LSTM (Memoria a corto y largo plazo). Estas redes, en conjunto con las redes neuronales recurrentes bidireccionales, permitieron abordar problemas de series temporales y secuencias más complejos.

- En 2008 se entrenan modelos de redes neuronales con la GPU por primera vez, consiguiendo una reducción significativa de los tiempos de entrenamiento y permitiendo un procesamiento masivo en paralelo, el cual es necesario para los modelos de aprendizaje profundo.

- En 2011 Andrew Ng y Jeff Dean crean el grupo de investigación Google Brain, el cual desarrollaría y popularizaría TensorFlow, el cual es un marco de aprendizaje profundo.

- En 2012 Alex Krizhevsky desarrolló una red neuronal convolucional profunda entrenada en la GPU la cual se llamó AlexNet, y que en comparación con los modelos existentes de la época mejoraba significativamente la precisión en clasificación de imágenes aumentando la popularidad de Deep Learning.

- En 2014 Goodfellow creó las Redes Neuronales de Creencia Profunda (DBN), también conocidas como Redes Generativas Adversariales (GAN) y se utilizarían en la creación de imágenes realistas y la generación de contenido.

- En 2016 DeepMind creó el programa de AlphaGo, el cual derrotaría al campeón mundial de Go, un juego estratégico de gran complejidad, demostrando así la habilidad de Deep Learning tanto en el proceso de investigación como luego en la parte práctica.

Los eventos mencionados previamente son algunos de los hitos clave en la historia del Deep Learning.

2.4 Aprendizaje supervisado vs aprendizaje no supervisado

Tanto el aprendizaje supervisado como el no supervisado son dos posibles enfoques para Deep Learning [12] a continuación se dará una explicación sobre cada uno.

El aprendizaje supervisado se caracteriza porque su conjunto de datos de entrenamiento ha sido etiquetado previamente, es decir, se conocen las respuestas correctas para cada ejemplo. Se utilizan algoritmos de regresión, los cuales buscan generar una predicción en forma de dato numérico, y algoritmos de clasificación, que generan una predicción en forma de dato categórico, todo esto minimizando las diferencias entre las salidas predichas por el modelo y las salidas reales conocidas.

Como gran ventaja se puede destacar su capacidad para predecir resultados sobre datos nuevos basándose en ejemplos anteriores los cuales estén correctamente etiquetados. Su precisión depende en gran medida de la calidad de los datos de entrenamiento.

Por otro lado, el aprendizaje no supervisado se basa en descubrir patrones, o relaciones intrínsecas en un conjunto de datos no etiquetados. En contraposición con el aprendizaje supervisado, en el aprendizaje no supervisado no se produce un etiquetado previo y, por tanto, no hay respuestas conocidas. Este enfoque resulta útil en conjuntos de datos sin etiquetar y para los cuales se desconoce la existencia de una estructura inherente de los datos.

Una ventaja clave es su capacidad a la hora de revelar información oculta, patrones o características de los datos que puedan existir.

Este tipo de algoritmos se clasifican en algoritmos jerárquicos los cuales posibilitan la formación de distintos niveles de agrupaciones y algoritmos no jerárquicos, los cuales siguen una metodología de particionamiento.

Cabe mencionar la existencia de técnicas híbridas, las cuales combinan elementos de ambos enfoques y se utilizan para la resolución de problemas cuya complejidad sea elevada. El tipo de datos que se estén tratando o los objetivos que se buscan conseguir también afectarán a la hora de elegir un método u otro.

2.5 Redes neuronales

Entendemos por redes neuronales algoritmos de Machine Learning a los cuales se les puede aplicar Deep Learning. Estos algoritmos consisten en, a una entrada numérica realizarle un conjunto de operaciones y obtener a la salida un resultado numérico.

2.5.1 Anatomía de una red neuronal

Las redes neuronales están formadas por los siguientes elementos [13]:

- Datos de entrada: información que se proporciona a la red neuronal para su procesamiento, pueden ser características o atributos de los ejemplos utilizados en el entrenamiento. Es fundamental para la obtención de resultados precisos que los datos de entrada sean muy relevantes y representativos.

- Capas: las redes neuronales están estructuradas en capas, entendiendo por las mismas un conjunto de neuronas dispuestas secuencialmente. La primera capa se denomina capa de entrada y es la que recibe los datos de entrada. Las capas intermedias se denominan capas ocultas y son las encargadas de realizar todo el procesamiento de la información. Por último, la capa final se conoce como capa de salida y es la cual proporciona los resultados.

- Función de activación: se utilizan para propagar la salida de una capa hacia la siguiente. Estas funciones introducen no linealidades y permiten que aprenda y modele relaciones más complejas. Un ejemplo de función de activación sería la función ReLu.

- Optimizador: algoritmo que se encarga de ajustar los valores de los parámetros de la red neuronal durante el entrenamiento con el objetivo de reducir los errores cometidos. Un ejemplo de optimizador sería el optimizador Adam.

- Función de pérdida: también conocida como la función de coste, evalúa lo que difieren las predicciones realizadas por la red neuronal en comparación con los valores reales del conjunto de datos empleados. El objetivo principal del entrenamiento es minimizar el valor de esta función ya que representa cuán ineficiente es la misma, para lograrlo se ajustan los pesos de la red neuronal.

El funcionamiento de la red neuronal es el siguiente: cada neurona en una capa recibe la salida de las neuronas de la capa anterior y las procesa utilizando una combinación lineal entre las entradas y los pesos sinápticos (los cuales se ajustarán utilizando el optimizador) que tiene asociada cada conexión. Acto seguido se aplica la función de activación a la suma ponderada y se obtiene así la salida de la neurona. Este proceso se repite hasta llegar a la capa de salida en la cual se evalúa el rendimiento utilizando la función de pérdidas. En la Figura 3 se muestra la estructura de una neurona [14]:

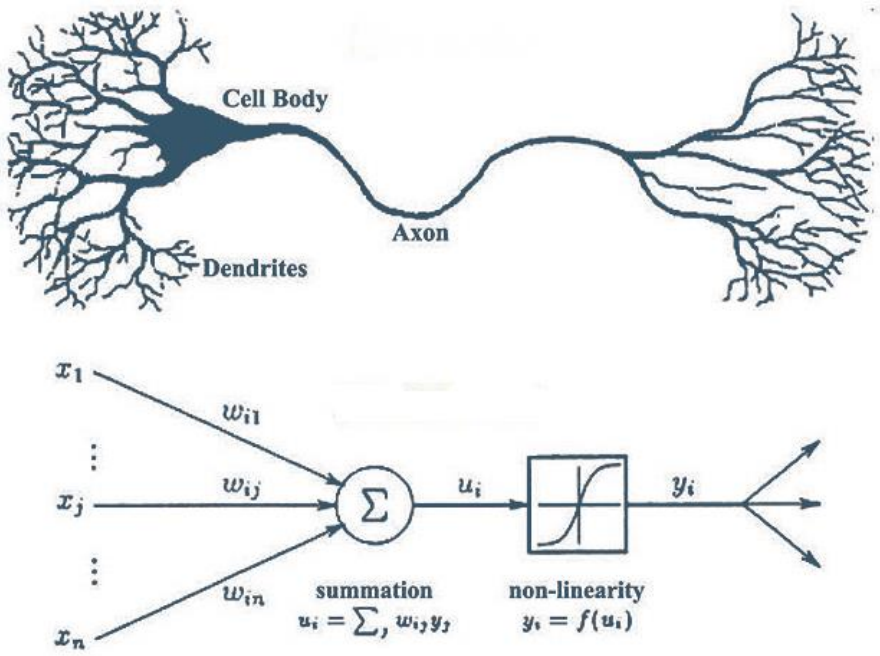


Figura 3: Neurona artificial

La salida de la misma se calcula de la siguiente manera (Ecuación 1):

$$y = FA\sum_{i=1}^{5} x_i w_i$$

Ecuación 1: salida neurona artificial

Entendiendo por FA la función de activación de la capa en la que se encuentra la neurona la cual introduce no linealidades y determina la salida de cada neurona.

El peso de una conexión entre dos neuronas representará la importancia de esa unión. Cada conexión tiene asociado un peso sináptico el cual determina qué tanto influye la neurona origen en la neurona destino. Durante el entrenamiento los pesos sinápticos se ajustan de manera iterativa con la finalidad de obtener las salidas deseadas para el conjunto de entrenamiento. El ajuste de los mismos es fundamental para el aprendizaje, de modo que cuantos más ejemplos se posean más preciso será el ajusto de los mismos y por tanto mejor será la precisión obtenida.

La función de activación es un elemento clave durante el proceso de la red neuronal puesto que afecta directamente a la capacidad de aprender y modelar relaciones complejas. Algunas de las más utilizadas son las siguientes [15]:

- Función de activación de Sigmoid: mapea los valores de entrada en un rango entre 0 y 1, por lo que, encuentra su mayor utilidad en problemas de clasificación binaria. Esta función tiene una curva en forma de “S” y se representa en la Ecuación 2.

$$f(x) = \frac{1}{1 - e^{-x}}$$

Ecuación 2: FA de Sigmoid

- Función de activación de Tanh: transforma los valores de entrada en una escala entre -1 y 1. Igual que para la función de Sigmoid, esta también presenta una curva en forma de “S”, con la diferencia de que esta tiene su centro en 0 y se representa en la Ecuación 3.

$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$

Ecuación 3: FA de Tanh

- Función de activación ReLu (*Rectified Linear Unit*): es una función no lineal ampliamente utilizada en redes neuronales. Convierte los valores negativos en 0 y los positivos los deja como están. Si bien es una función simple, ha demostrado ser muy eficaz y su no linealidad permite la resolución de problemáticas complejas y se representa en la Ecuación 4.

$$f(x) = \begin{cases} 0 & si \quad x < 0 \\ \\ x & si \quad x \geq 0 \end{cases}$$

Ecuación 4: FA de ReLu

- Función de activación Softmax: comúnmente utilizada en la capa de salida al abordar un problema multiclase. Transforma la salida en una representación de probabilidades, donde la suma total es igual a 1. Se utiliza para asignar una probabilidad a cada clase de salida y se representa en la Ecuación 5.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{para } j = 1, \ldots, K.$$

Ecuación 5: FA softmax

2.5.2 Entrenamiento de una red neuronal

Entrenar una red neuronal [16] consiste en ajustar los pesos y sesgos de la red de manera iterativa utilizando el conjunto de datos de entrenamiento. El objetivo final que se busca tras el entrenamiento es que la red neuronal sea capaz de realizar predicciones precisas partiendo de los datos que se le han proporcionado.

Durante el entrenamiento, a la red neuronal se le introducen unos datos de partida, los cuales, al pasar por las distintas capas ocultas, sufren una serie de operaciones matemáticas hasta generar una salida, esta salida se compara con el valor esperado para

calcular el error cometido, el entrenamiento busca hacer que este sea lo menor posible. Posteriormente, se utilizan algoritmos de optimización, como el descenso de gradiente, el cual consiste en moverse en la dirección de máximo decrecimiento, con el objetivo de encontrar los pesos y sesgos que minimicen la función de pérdida.

El proceso se repite de manera iterativa sobre los datos de entrenamiento, los cuales pueden estar divididos en lotes o de manera individual, hasta alcanzar lo que se conoce como criterio de convergencia o bien se agoten las iteraciones. Al finalizar todo el proceso los pesos y sesgos se habrán ajustado para poder realizar predicciones más precisas.

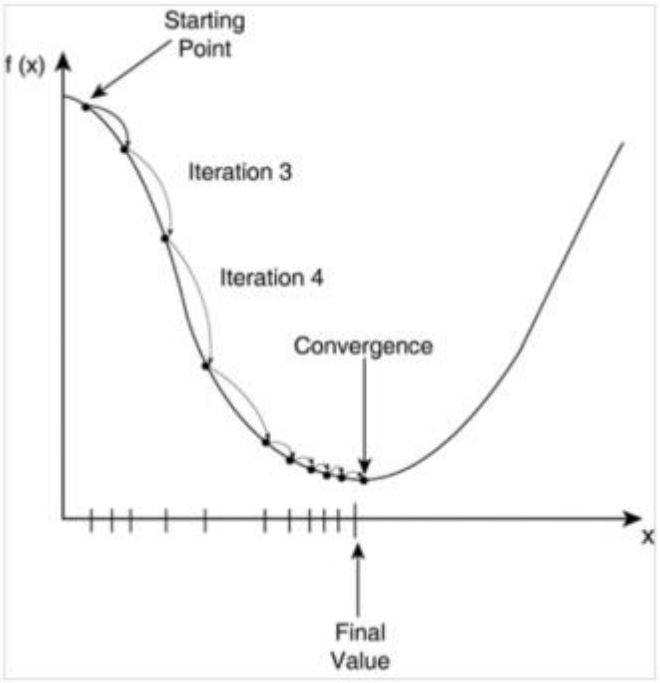


Figura 4: método del gradiente descendiente [17]

En la Figura 4 se muestra representado el método del gradiente descendiente, el cual busca reducir el error en cada iteración ajustando los pesos y en función de la magnitud y la dirección del error hasta que la red converja.

2.5.3 Tipos de redes neuronales

Si bien a la hora de crear redes neuronales hay muchas posibilidades, hay unas opciones que se utilizan en mayor medida y son las siguientes [18]:

- Redes neuronales feedforward (FNN): reciben el nombre de feedforward porque el flujo de información no se produce de manera cíclica, si no que,

por el contrario, fluye en una única dirección desde la capa de entrada, pasando por las capas ocultas hasta llegar a la capa de salida, sin retroalimentaciones. Existen dos tipos de FNN: perceptrón de una capa y perceptrón multicapa.

El perceptrón es la forma más básica de una FNN, estando formada únicamente por la capa de entrada y una capa de salida, y es utilizado principalmente en problemas de clasificación binaria linealmente separables. Se basa en una función de activación escalón, que toma un valor de entrada y produce uno de salida en función del umbral marcado. Si bien su capacidad para resolver problemas complejos es limitada debido a su simplicidad, es la base fundamental para las redes neuronales y ayuda a comprender el funcionamiento inicial de las FNN.

Las redes de perceptrón multicapa, también conocidas como MLP son FNN las cuales contienen una o más capas ocultas. Mediante el uso de capas ocultas la red puede aprender representaciones no lineales abordando así problemas de mayor complejidad. Pero tienen el inconveniente de que son computacionalmente muy costosas de entrenar para modelos con muchas entradas.

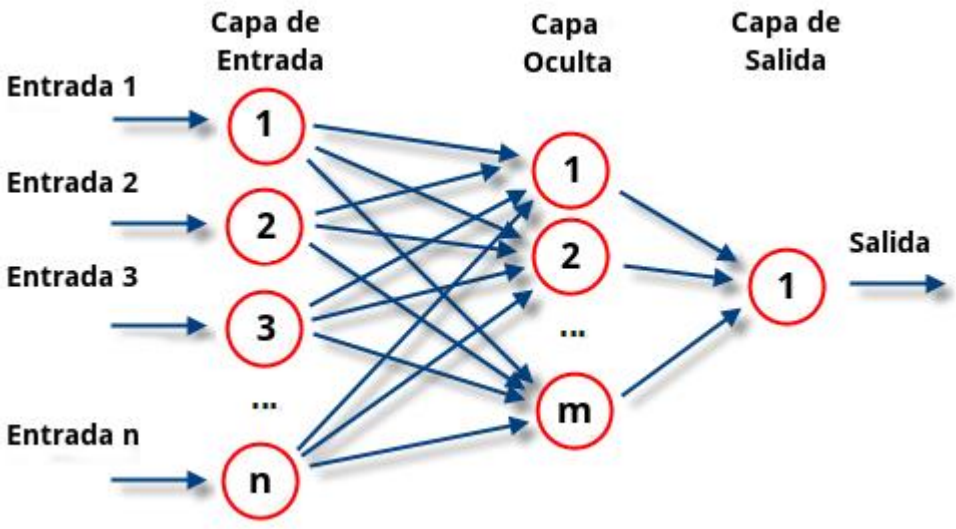


Figura 5: Perceptrón Multicapa [14]

En la Figura 5 se aprecia como cada neurona de una capa está conectada con todas las neuronas de la siguiente capa, esto es lo que se conoce como

una red con conectividad total. En caso de existir dos o más capas ocultas, se forma lo que se conoce como una red neuronal densa (DNN).

- Redes neuronales convolucionales (CNN): un tipo de red neuronal muy utilizada sobre todo para casos de procesamiento de imagen y tareas de visión por computadora. Son un tipo de red que ha demostrado ser muy eficaz a la hora de reconocer y clasificar patrones visuales.

  La capa convolucional es la piedra angular de este tipo de redes, ya que se encarga de extraer información local de las imágenes. A continuación, en la Figura 6 se muestra cómo se extraen patrones locales de texto, dentro de las secuencias correspondientes al texto. Estos patrones pueden volver a utilizarse facilitando así la correcta clasificación de los textos.

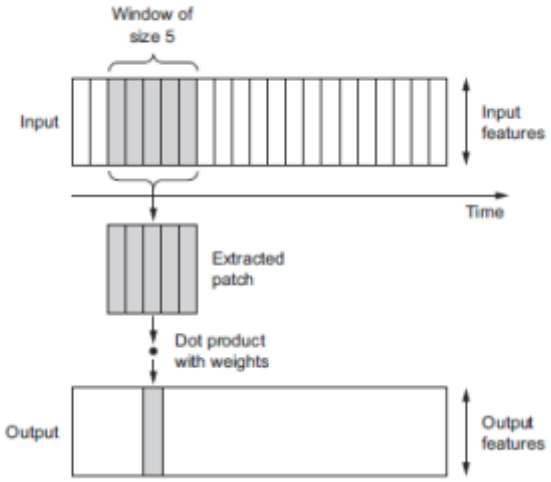


Figura 6: red convolucional 1D [19]

  La capa de agrupación (pooling) es la encargada de reducir la dimensionalidad espacial de los mapas de características y extraer las de mayor relevancia, lo más normal es realizar un max pooling o muestreo máximo, el cual consiste en tomar el valor máximo de cada región.

- Redes neuronales recurrentes (RNN): son redes neuronales utilizadas para modelar datos con dependencias temporales, es decir, son redes neuronales con memoria. Tienen conexiones recurrentes que les permiten mantener y volver a utilizar información previa en caso de ser necesaria en el

procesamiento de nuevas secuencias, es decir, tienen una estructura en forma de bucle. Este tipo de redes son muy útiles para tratar problemas como puede ser el procesamiento de lenguaje natural.

En la Figura 7 se aprecia su arquitectura, en la cual se pueden destacar las capas ocultas recurrentes, que son unidades que toman tanto la entrada actual como una recurrente, la cual es la salida de la unidad en el instante de tiempo anterior. Combina ambas entradas con una función de activación y produce una nueva salida recurrente.

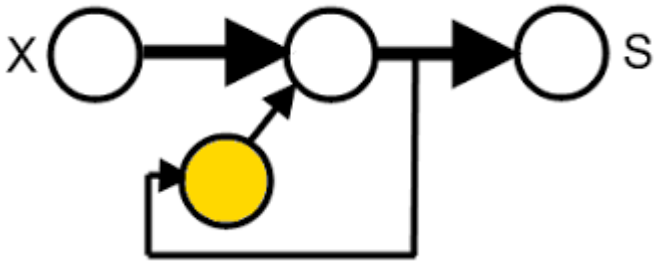


Figura 7: Red neuronal recurrente

2.6 Métricas de rendimiento

Para evaluar un modelo, es necesario utilizar métricas objetivas, las cuales ayuden a elegir el que mejor funciona. Durante el estudio nos vamos a encontrar distintos casos los cuales se pueden resumir de la siguiente manera:

- Positivo Verdadero (TP): se refiere a los casos en que el clasificador predice correctamente la etiqueta. Un ejemplo de esto es que clasifique una palabra como sustantivo y sí sea un sustantivo.

- Positivo Falso (FP): se refiere a los casos en que el clasificador predice "falsamente" la etiqueta. Un ejemplo de esto es que clasifique una palabra como un sustantivo y sea en realidad un adverbio.

- Falso Negativo (FN): se refiere a los casos en que el clasificador "falla" en reconocer la etiqueta. Un ejemplo de esto es que no clasifique una palabra como sustantivo cuando la palabra es de hecho un sustantivo.

- Verdadero Negativo (VN): se refiere a los casos en que el clasificador etiqueta correctamente la muestra como perteneciente a la clase negativa. Un ejemplo de esto es que no clasifique una palabra como sustantivo y efectivamente esa palabra no sea un sustantivo.

Estas cuatro métricas van a ser de vital importancia para evaluar el rendimiento del modelo utilizando aciertos y errores a la hora de clasificar.

Una vez se han explicado los distintos casos que se pueden presentar, hay que decidir las métricas que se van a utilizar con el fin de decidir si los modelos diseñados son mejores o peores. Cabe destacar que la elección de las métricas varía en función de la casuística a resolver. La métrica tiene como objetivo brindar una representación del funcionamiento de la red neuronal, utilizando un conjunto de valores los cuales resultan de la realización de una serie de procesos, las métricas que se van a usar en este TFG serán:

- Precisión (Accuracy): es la medida utilizada para evaluar los modelos, puesto que es la métrica que se encarga de representar el porcentaje de predicciones positivas correctas frente al total de predicciones positivas que realiza el modelo. Esto es, la exactitud con la que el modelo está prediciendo las clases. Está representada por fórmula de la Ecuación 6.

$$Exactitud = \frac{TP + TN}{TP + TN + FP + FN}$$

Ecuación 6: accuracy

- Exhaustividad (Recall): métrica que se encarga de representar el porcentaje de verdaderos positivos frente al total de muestras positivas, representando así la capacidad del modelo de predecir todas las muestras positivas. Está representada por la fórmula representada en la Ecuación 7.

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

Ecuación 7: Recall

- F1-Score: combinación entre las métricas de precisión y recall, se calcula realizando la media armónica de ambas, es útil a la hora de trabajar con desequilibrios entre las clases. Está representada por la siguiente fórmula de la Ecuación 8.

$$F_{beta} = (1 + beta^2) * \frac{Precision * Exhaustividad}{(beta^2 * Precision) + Exhaustividad}$$

Ecuación 8: F1-Score

- Support: métrica que representa el número de muestras de cada clase.

3 Herramientas utilizadas

3.1 Introducción

Para llevar a cabo el proyecto ha sido necesario utilizar las herramientas y plataformas software que a continuación se describen.

3.1.1 Python

Python [20] es un lenguaje de programación de alto nivel, interpretado y de propósito general. Fue creado por Guido von Rossum en el año 1991 y posee una amplia variedad de bibliotecas y frameworks especializados en desarrollo de Inteligencia Artificial. A continuación, se enumeran una serie de características que lo hacen una buena elección para este proyecto:

- Bibliotecas: especializadas en desarrollo de Inteligencia Artificial, las cuales tienen una gran cantidad de algoritmos y herramientas para tareas como la clasificación.

- Sintaxis sencilla y legible: esto es especialmente importante a la hora de desarrollar aprendizaje automático en el que se busca una implementación y análisis rápidos de los modelos y algoritmos.

- Fusión con otras tecnologías: funciona bien a la hora de hacer uso de herramientas desarrolladas para otras tecnologías, como puede ser la biblioteca de visualización de datos Matplotlib.

- Flexibilidad y extensibilidad: esto permite unificar de manera sencilla el aprendizaje automático con otros aspectos de la IA, en el caso de este estudio, el procesado de lenguaje natural.

Por todo lo comentado previamente, Python es un lenguaje muy utilizado a la hora de trabajar para resolver problemas con Inteligencia Artificial y en particular con técnicas

de aprendizaje automático, lo cual ha llevado a la conclusión de que este trabajo debía realizarse utilizando Python.

3.1.2 Tensorflow y Keras

Tanto Tensorflow como Keras son bibliotecas de código abierto que se utilizan para trabajar con redes neuronales en Python.

Tensorflow [21] es una biblioteca de software de aprendizaje automático desarrollada por Google, utilizado para construir y entrenar modelos de aprendizaje automático. Tensorflow si bien es un framework de bajo nivel, proporciona una plataforma flexible con un rendimiento elevado a la hora de construir y entrenar modelos de aprendizaje automático. Algunas características relevantes que posee son las siguientes:

- Grafos de cómputo: permiten un alto nivel de paralelismo y optimiza el procesado de grandes cantidades de datos en forma de texto.

- Soporte para redes neuronales: ofrece un amplio repertorio de capas que se utilizarán para crear redes neuronales, así como optimizadores y métricas para el entrenamiento de modelos de procesamiento de lenguaje natural.

Keras [22] es una biblioteca de alto nivel, la cual se ejecuta sobre Tensorflow. Permite la combinación de capas de diferentes tipos, para así, crear redes neuronales más complejas y a su vez ofrece funciones como pueden ser la tokenizacion o One-Hot, las cuales se usarán en el proyecto. En definitiva, Keras simplifica la creación de redes neuronales.

3.1.3 NLTK

NLTK [23] es un kit de herramientas de lenguaje natural y un conjunto de bibliotecas de Python el cual se utiliza para asistir en la resolución de problemas de Procesamiento de Lenguaje Natural (PNL).

3.1.4 Google Colab

Google Colab [24] es la plataforma en la cual se ha desarrollado el código de Python para crear las redes neuronales y el modelo de Deep Learning. Google creó este entorno de ejecución en la nube con el fin de que se utilizase en proyectos de investigación y creación de algoritmos de Machine Learning.

3.1.5 Twitter

Plataforma de microblogging [25] donde los mensajes no pueden superar los 280 caracteres, y donde estos mensajes reciben el nombre de tweets. Es una red social ampliamente utilizada para compartir noticias, opiniones o pensamientos, lo cual la hace ideal a la hora de realizar un estudio sobre emociones para mensajes. Los usuarios pueden interactuar con los tweets mediante “me gusta” o retweets.

3.1.6 Twitch

Plataforma de retransmisión en vivo [26] centrada en videojuegos y retransmisión de eventos en directo. En la actualidad se ha expandido a otros ámbitos sociales como pueden ser la música o el arte, entre otros. En Twitch se permite interacciones en tiempo real entre usuarios mediante el uso de los chats en los cuales los espectadores pueden enviar mensajes. Estos mensajes se pueden utilizar, una vez se les realiza un estudio para discernir la respuesta emocional que reflejan, así como para analizar el nivel de *engagement* de los creadores de contenido de la plataforma o medir la respuesta emocional que éstos generan a partir de sus transmisiones en directo.

4

Análisis de la respuesta emocional en redes sociales: polaridad

4.1 Introducción

Este apartado se centrará en realizar una optimización de los hiperparámetros utilizados para analizar el nivel de polaridad de los mensajes en redes sociales a partir del entrenamiento con diferentes corpus de datos pertenecientes a diferentes contextos y redes sociales. El objetivo final será detectar y clasificar cada mensaje siguiendo la polaridad que presente el mismo, pudiendo ser esta de 3 tipos distintos, en concreto, positivo, negativo e indeterminado.

4.2 Definición del modelo de Deep Learning

El modelo con el cual se va a trabajar es un modelo híbrido basado en Deep learning. El modelo combina los beneficios de una Red Neuronal Recurrente (RNN) y una Red Neuronal Convolucional (CNN) con el fin de mejorar la capacidad de resolver problemas específicos.

Para el procesado de lenguaje natural, que es lo que se busca hacer con los mensajes de Twitch y Twitter, las CNN se utilizan para obtener características locales y espaciales tanto de palabras individuales como de secuencias de las mismas, capturando así los patrones relevantes. Las capas convolucionales aplicarán filtros de convolución al texto generando así características de nivel superior, las cuales servirán para realizar el análisis y clasificación de los contenidos.

A su vez, las RNN se utilizan para obtener información secuencial y relaciones a largo plazo entre datos. En el caso de los mensajes, el orden de las palabras es un factor

importante ya que existe correlación entre palabras adyacentes, por lo que las RNN se pueden utilizar para modelar la estructura temporal y las dependencias de palabras.

El modelo híbrido aprovecha el hecho de que las capas convolucionales extraigan características locales y estructurales del texto, haciendo así que los datos de partida se transformen en una representación de características más compacta, la cual se pasará por las capas recurrentes donde se obtendrán las dependencias a largo plazo y la información temporal de la secuencia. Este proceso se muestra en la Figura 8.

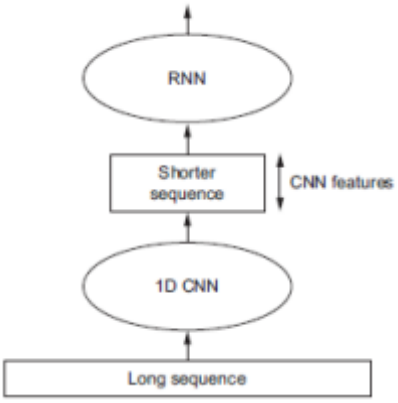


Figura 8: Representación Modelo Híbrido [16]

Esta combinación presenta la ventaja de que las capas convolucionales van a reducir la carga de trabajo de las capas recurrentes, ya que, previamente se han extraído las características relevantes y, por tanto, se ha disminuido la dimensión del problema. Esta estructura en sí mejora tanto la eficacia como la precisión del modelo.

A su vez, combinar características locales con dependencias temporales producirá una mejora en la capacidad de la red para capturar información, lo cual se traduce en que se posibilita abordar problemas de mayor complejidad.

El modelo con el que vamos a trabajar parte de los siguientes componentes base: una capa de Embedding, una capa convolucional de una dimensión (Conv1D), una capa de MaxPooling, una capa LSTM y, por último, una capa densa, la cual clasificará los mensajes en 2 o 3 clases (2 o 3 polaridades) dependiendo del problema que estemos estudiando en cada momento, tal como se observa en la representación de la Figura 9 y 10.

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 33, 200)	9080400
conv1d (Conv1D)	(None, 27, 128)	179328
max_pooling1d (MaxPooling1D)	(None, 2, 128)	0
lstm (LSTM)	(None, 90)	78840
dense (Dense)	(None, 3)	273

Figura 9: Modelo Híbrido

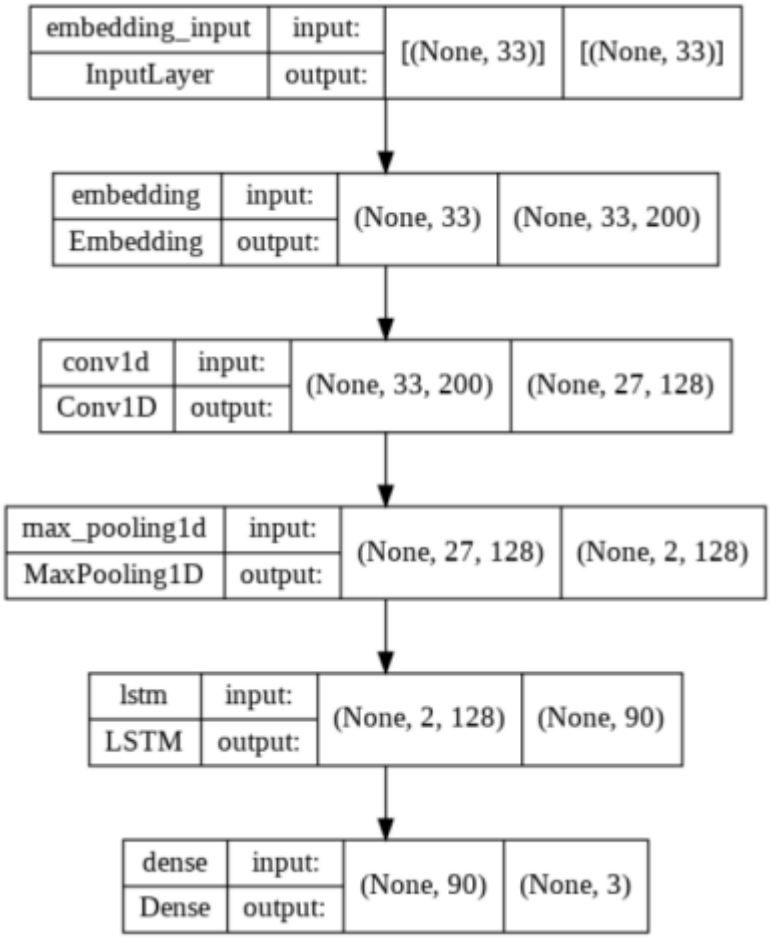


Figura 10: Modelo Híbrido

4.3 Análisis de la polaridad en Twitch y videojuegos

4.3.1 Definición del dataset de Twitch

El corpus de mensajes de Twitch consta de 2007 mensajes, los cuales están clasificados con su correspondiente polaridad, que puede ser de 3 tipos diferentes: Positiva, Negativa e Indeterminada. Del corpus se obtienen los datos de partida, esto es, los mensajes y su polaridad, que posteriormente se usarán para el entrenamiento del modelo de Deep Learning.

En este caso, la distribución de los mensajes es la siguiente: 967 positivos, 822 negativos y 218 indeterminados, y su distribución se muestra en la Figura 11.

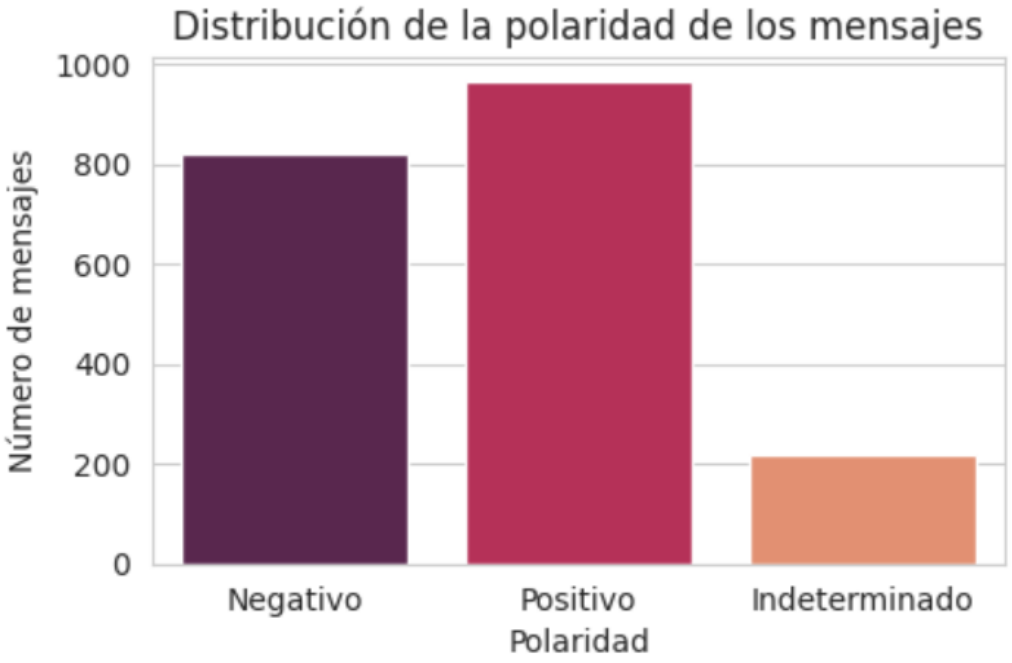


Figura 11: distribución del corpus por polaridad

Dado que cada mensaje solo puede pertenecer a una polaridad, estamos ante un problema de clasificación de etiqueta única y multiclase.

Se puede observar en la Figura 11 que las polaridades negativa y positiva tienen una mayor cantidad de muestras en comparación con la indeterminada, lo que podría ocasionar un problema de desbalance, haciendo así que el modelo prediga con mayor probabilidad que el mensaje pertenece a las clases con más muestras.

En la Figura 12, se presenta una nube de palabras, la cual representa de manera gráfica las palabras de mayor relevancia para una polaridad específica, en este caso la polaridad positiva.

Figura 12: nube de palabras de la clase Positiva

4.3.2 Resultados de polaridad para el dataset de Twitch

4.3.2.1 Análisis con dos polaridades

Entrenamiento y evaluación del modelo de Deep Learning

Para entrenar y evaluar el funcionamiento del modelo el conjunto de datos fue dividido en dos subconjuntos: entrenamiento (70%) y prueba (30%). Para el entrenamiento se han realizado distintos tipos de pruebas con el objetivo de ver las variaciones en la métrica de precisión que conllevaban. Estas pruebas son: ajuste del número de neuronas y los filtros LSTM, ajuste de las tasas de dropout y acto seguido el de la tasa de aprendizaje para el optimizador Adam. A continuación, se hicieron pruebas con la reducción del número total de palabras únicas escogidas de entre todas las del corpus y por último un

ajuste del tamaño de batch size utilizado. Todo esto con la finalidad de ajustar los hiperparámetros y evitar el sobreajuste en el modelo desarrollado.

Durante la fase de entrenamiento se siguieron varios pasos:

- Preparación de los datos: se obtienen dos conjuntos de datos, primero el conjunto 'X', resultado de concatenar 'X_train' y 'X_test' y posteriormente el conjunto 'y' resultado de concatenar 'y_train' e 'y_test'.

- Se realiza la validación cruzada: para la validación cruzada se utiliza un objeto 'KFold' con 5 divisiones y a su vez se definen dos listas, 'acc_per_fold' y 'loss_per_fold' las cuales almacenan los resultados de precisión y pérdidas respectivamente.

- Bucle de validación cruzada: el cual va a iterar en relación a los índices de entrenamiento generados por el 'KFold'.

A su vez, la fase de prueba se realizó teniendo en cuenta varios ejemplos de métricas de rendimiento: precisión, recall y F1-score. Estas métricas proporcionan una medida cuantitativa de la capacidad del modelo para clasificar de manera acertada los mensajes dentro de cada una de las diferentes categorías que se barajan.

Ajuste del número de neuronas y filtros

Como primera etapa de la fase de entrenamiento se va a repetir el proceso de entrenamiento variando el número de neuronas y filtros LSTM dentro de un rango de valores con el fin de encontrar la combinación que proporcione mejores resultados, y también va a servir para obtener una idea general de cómo afecta la arquitectura del modelo a su rendimiento.

Se muestra a continuación la Tabla 1 a modo de resumen los resultados obtenidos con la variación de neuronas y filtros LSTM. A su vez se resalta en gris la configuración para la cual se obtienen los mejores resultados.

Número de neuronas en capa convolucional	Filtros para capa LSTM	Dropout para capa convolucional	Dropout para capa LSTM	Tamaño del Kernel	Accuracy Validación cruzada
192	256	0.2	0.3	8	68.72%
192	128	0.2	0.3	8	67.97%
192	96	0.2	0.3	8	67.41%
192	64	0.2	0.3	8	69.09%
180	256	0.2	0.3	8	70.76%
180	96	0.2	0.3	8	68.53%
160	128	0.2	0.3	8	70.76%
160	64	0.2	0.3	8	67.78%
150	256	0.2	0.3	8	68.34%
128	128	0.2	0.3	8	69.09%
128	64	0.2	0.3	8	70.20%
96	64	0.2	0.3	8	68.16%
180	512	0.2	0.3	8	70.58%

Tabla 1: resultados obtenidos para variación de neuronas y filtros LSTM

Observamos en la Tabla 1 que los parámetros muestran un nivel de ajuste bastante bueno, no existiendo una gran diferencia en los resultados obtenidos modificando el valor de dichos hiperparámetros. A continuación, en la Tabla 2 se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, número de neuronas igual a 180 y número de filtros LSTM igual a 256.

Polaridad	Precisión	Recall	F1-score	Support
Negativo	70%	63%	66%	243
Positivo	71%	78%	74%	294
Global (weighted avg)	71 %	71%	71%	537

Tabla 2: valores de métricas para optimización de neuronas y filtros LSTM

En la Tabla 2 observamos las métricas previamente mencionadas y que se busca optimizar, obteniendo unas precisiones del 70% para la polaridad negativa y un 71% para la polaridad positiva.

Ajuste de las tasas de dropout

El segundo paso del entrenamiento del modelo va a ser, partiendo de la mejor configuración previamente obtenida, variar las tasas de dropout en un intervalo que recorra

valores desde los 0.2 a los 0.8 para ambos casos. En concreto, se han obtenido los valores de precisión representados en la Tabla 3 a modo de resumen.

Número de neuronas en capa convolucional	Filtros para capa LSTM	Dropout para capa convolucional	Dropout para capa LSTM	Tamaño del Kernel	Accuracy Validación cruzada
180	256	0.2	0.3	8	70.76%
180	256	0.3	0.3	8	70.20%
180	256	0.4	0.3	8	69.27%
180	256	0.5	0.3	8	71.51%
180	256	0.6	0.3	8	71.14%
180	256	0.7	0.3	8	71.14%
180	256	0.8	0.3	8	71.69%
180	256	0.2	0.4	8	71.14%
180	256	0.2	0.5	8	67.04%
180	256	0.2	0.6	8	**71.88%**
180	256	0.2	0.7	8	69.83%
180	256	0.2	0.8	8	70.02%

Tabla 3: resultados obtenidos para variación de tasas de dropout

Analizando los resultados obtenidos destacamos el hecho de que realizar una variación de los valores de las tasas de dropout no genera en sí un cambio significativo en la precisión del modelo, en este caso no llegando ni a suponer una fuerte mejora del mismo. A continuación, en la Tabla 4 se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, valor del dropout de la capa convolucional igual a 0.2 y dropout de la capa LSTM igual a 0.6.

Polaridad	Precisión	Recall	F1-score	Support
Negativo	70%	66%	68%	243
Positivo	73%	77%	75%	294
Global (weighted avg)	72%	72%	72%	537

Tabla 4: valores de métricas para optimización de tasas de dropout

En la Tabla 4 se observa como la polaridad negativa se mantiene en una precisión 70% mientras que la polaridad positiva ha aumentado en un 2%.

Ajuste de tasa de aprendizaje del optimizador Adam

El siguiente paso a seguir en el modelo, es un barrido para distintos valores de la tasa de aprendizaje. Es un factor fundamental a la hora de entrenar modelos de aprendizaje automáticos, puesto que determinará el grado de magnitud con el cual se realizarán los ajustes a los distintos parámetros del modelo, lo cual afecta a su vez a la convergencia del mismo. Hay dos motivos principales por los cuales resulta interesante controlar la tasa de aprendizaje: controlar la velocidad de convergencia y superar mínimos locales que se

puedan producir. Cabe destacar que encontrar la tasa de convergencia óptima es un proceso empírico, es decir, que requiere de prueba y error para encontrar el resultado que mejor se adecue al problema en particular que se intenta resolver.

Tasa de aprendizaje	Accuracy Validación cruzada
0.0008	70.02%
0.001	71.88%
0.003	68.34%
0.005	70.39%
0.007	72.07%
0.009	68.53%
0.01	69.46%

Tabla 5: resultados obtenidos para variación de tasa de aprendizaje

Como resultado del estudio de la tasa de aprendizaje (Tabla 5) se puede observar que dependiendo del valor que se le asigne, se produce una diferencia de casi un 4% en los resultados obtenidos. A continuación, en la Tabla 6 se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, una tasa de aprendizaje igual a 0.007.

Polaridad	Precisión	Recall	F1-score	Support
Negativo	71%	65%	68%	243
Positivo	73%	78%	75%	294
Global (weighted avg)	72%	72%	72%	537

Tabla 6: valores de métricas para optimización de tasa de aprendizaje

En la Tabla 6 se observa como la polaridad negativa aumenta su precisión en un 1% mientras que la polaridad positiva se mantiene en una precisión del 73%.

Reducción del número total de palabras del corpus

Previa realización de la reducción de palabras es importante explicar el concepto de Ganancia de Información (IG), ya que es este mismo el que nos permitirá darle un sentido a reducir el número de palabras y que aumente la precisión. La IG [27] es un concepto de gran importancia en el procesamiento de lenguaje natural cuyo objetivo es determinar la importancia de una característica o término en relación a una tarea específica, en este caso, la clasificación de textos. En este caso se utilizará para determinar los términos que aportan una mayor información para distinguir entre clases. Aplicando este concepto se consigue reducir el número de palabras totales, de tal forma que se utilice un

menor número, pero con el mayor valor de IG posible, es decir, solamente teniendo en cuenta las palabras de más peso.

De este modo, el siguiente paso del entrenamiento es realizar una reducción del número total de palabras únicas del corpus utilizadas en el modelo final, con el fin de aumentar la eficiencia computacional. De este modo se descartan las palabras de menor relevancia, las cuales se clasificaron de esta manera teniendo en cuenta tanto la frecuencia con que se repetían, así como la ganancia de información que aportaban.

La reducción del vocabulario utilizado puede llevar a una mayor eficiencia computacional y un aumento en la velocidad de entrenamiento del modelo. Existe a su vez un compromiso entre la reducción del vocabulario utilizado y la posibilidad de perder información importante, por lo que es necesario realizar las reducciones de manera escalonada (en este caso en saltos de 100 palabras), como se representa en la Tabla 7.

Tamaño vocabulario	**Accuracy Validación cruzada**
2700	72.07%
1653	79.14%
1453	78.77%
1253	77.84%
1053	79.33%
1553	77.09%
1153	80.45%

Tabla 7: resultados obtenidos para variación de tamaño del vocabulario

Es apreciable cómo la utilización de un menor número de palabras, descartando aquellas de menor relevancia nos conduce en un principio a un aumento en la precisión del modelo, pero que al seguir disminuyendo el número ocurre lo que se había comentado previamente y la precisión empieza a disminuir, ya que llegará un momento en el que se comenzarán a descartar palabras importantes del corpus.

En la Tabla 8, se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, tamaño del vocabulario igual a 1153 palabras. En la Tabla 8 se observa como la polaridad negativa aumenta su precisión en un 13% mientras que la polaridad positiva aumenta su precisión en un 5%.

Polaridad	Precision	Recall	F1-score	Support
Negativo	84%	70%	76%	243
Positivo	78%	89%	83%	294
Global (weighted avg)	81%	80%	80%	537

Tabla 8: valores de métricas para optimización de tamaño del vocabulario

Ajuste del tamaño del batch_size

Como último paso en la fase de entrenamiento del modelo se realiza una variación del parámetro batch_size. Si bien un valor elevado de batch_size lleva a una convergencia más rápida y un rendimiento mejorado, hay que tener en cuenta también que si es demasiado grande se puede producir un exceso de demanda para la memoria y los recursos computacionales. Por el contrario, si se utiliza un valor pequeño, se obtiene una mayor variabilidad para los datos de entrenamiento, lo que es interesante ya que ayudaría a solventar problemas como pueden ser los estancamientos de óptimos locales. Cabe mencionar también que es una alternativa muy recomendada si se tienen recursos computacionales limitados. Se observa en la Tabla 9 que el equilibrio entre eficiencia y rendimiento se encuentra para un batch_size de valor 256, que es el valor con el que veníamos realizando las pruebas hasta este momento.

batch size	Accuracy Validación cruzada
32	79.70%
64	79.89%
128	79.33%
256	80.45%
512	79.33%

Tabla 9: resultados obtenidos para variación del batch_size

Resumen de hiperparámetros del modelo final y resultados

En la Figura 13 se muestra el resultado final del modelo una vez se han realizado todos los ajustes de hiperparámetros que se han ido mencionando paso a paso. En el se aprecia la concatenación de los subconjuntos train y test, la configuración del bucle, los valores finales para el número de neuronas de la capa convolucional, en este caso 180, el número de filtros LSTM, en este caso 256, las tasas de dropout que son 0.2 y 0.6 respectivamente, el optimizador Adam con un learning rate de 0.007 y el valor del batch_size de 256.

La configuración de parámetros que ha permitido optimizar el modelo de Deep learning se representa a modo de resumen en la Tabla 10.

```
from keras.models import Sequential
from sklearn.utils.multiclass import type_of_target
from tensorflow.keras.callbacks import EarlyStopping
import numpy as np
from keras.layers import Embedding, Flatten, Dense, LSTM, Conv1D, GlobalMaxPooling1D, MaxPooling1D, Bidirectional, GRU
from sklearn.model_selection import KFold
import statistics
from tensorflow.keras.optimizers import Adam

X = np.concatenate((X_train, X_test), axis=0)
y = np.concatenate((y_train, y_test), axis=0)

#y=np.argmax(y,axis=1)
print(type_of_target(y))
acc_per_fold=[]
loss_per_fold=[]
print(type_of_target(y))
kf=KFold(n_splits=5, shuffle=True, random_state=999)
cvscores=[]
for train, test in kf.split(X_train, y_train):
  model = Sequential()
  embedding_layer = Embedding(vocab_size, 200, input_length=maxlen)
  #embedding_layer = Embedding(vocab_size, 300, input_length=maxlen, weights=[embedding_matrix],trainable=False)

  model.add(embedding_layer)

  model.add(Conv1D(180, 8, activation='relu'))

  model.add(MaxPooling1D(10))

  model.add(LSTM(256, dropout=0.2, recurrent_dropout=0.6))


  model.add(Dense(2, activation='softmax'))
  model.compile(optimizer=Adam(learning_rate=0.007), loss='categorical_crossentropy', metrics=['accuracy'])
  model.summary()


  early_stop = EarlyStopping(monitor = 'accuracy', mode = 'max', verbose = 1, patience = 5)
  model.fit(X[train], y[train], epochs=100, batch_size=256, verbose=1,validation_data = (X[test], y[test]), callbacks=[early_stop])

  scores = model.evaluate(X[test], y[test], verbose=1)
  print(f'Score for fold : {model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')
  acc_per_fold.append(scores[1] * 100)
  loss_per_fold.append(scores[0])
print(acc_per_fold)
print(statistics.mean(acc_per_fold))
```

Figura 13: modelo final tras optimizaciones

Parámetros	Valores
Número Neuronas	180
Número filtros	256
Tasa de Dropout	0.2
Tasa Recurrent_dropout	0.6
Tasa de aprendizaje	0.007
Tamaño vobaculario	1153
Batch_size	256

Tabla 10: parámetros optimizados

En la Tabla 11 se observan los siguientes valores en las diferentes métricas:

- Para polaridad negativa: precisión del 84%, lo cual nos dice el porcentaje de las predicciones negativas que en realidad lo eran. Recall del 70% nos indica que de todos los mensajes que en realidad eran negativos solo se escogieron como tal el 70%.

- Para polaridad positiva: precisión del 78%, lo cual nos dice el porcentaje de las predicciones negativas que en realidad lo eran. Recall del 89% nos indica que de todos los mensajes que en realidad eran negativos solo se escogieron como tal el 89%.

Polaridad	Precisión	Recall	F1-score	Support
Negativo	84%	70%	76%	243
Positivo	78%	89%	83%	294
Global (weighted avg)	81%	80%	80%	537

Tabla 11: valores de métricas para configuración óptima

Por otro lado, la matriz de confusión tiene como finalidad aportar una representación visual de las predicciones realizadas por el modelo y compararlas con las clases a las que verdaderamente pertenecen los datos tratados. La matriz de confusión tiene la siguiente configuración: las filas corresponden a las clases reales y las columnas a las clases predichas por el modelo. De ese comportamiento se puede deducir que en la diagonal de la matriz se encuentran las muestras que han sido correctamente predichas, a este conjunto se le conoce como verdaderos positivos. En este caso se observa en la Figura 14 como el modelo predice más veces, de manera errónea, mensajes de polaridad positiva, cuando en realidad son negativos. Esto se puede deber a la representación de características, falta de información discriminatoria o a ruido en los datos. Por otro lado, los mensajes de polaridad positiva real los predice con una precisión elevada.

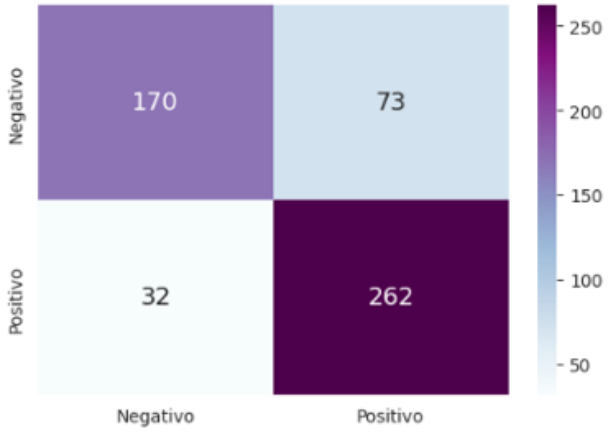


Figura 14: matriz de confusión para configuración óptima

Figura 15: resultados para configuración óptima

En la Figura 15 observamos una representación de los valores finales obtenidos una vez se han optimizado todos los hiperparámetros.

4.3.2.2 Análisis con tres polaridades

Las capas ocultas [28] tienen como finalidad mejorar la capacidad de extracción de características importantes del modelo, consiguiendo así una clasificación más precisa. Para realizar las pruebas, se utilizaron 3 capas de tamaño 256, 128 y 64 neuronas simultáneamente. La posición elegida para situarlas en el código (Figura 16) se debe a una representación jerárquica en la cual la capa de convolución se encarga de capturar características locales y de bajo nivel, mientras que las capas ocultas se encargan de

capturar características globales y de alto nivel, al colocarlas de esta manera se permite al modelo aprender representaciones más abstractas, lo cual es una gran ventaja en problemas de procesado de lenguaje natural. En este caso concreto se realizó un estudio simultáneo de los resultados tanto sin utilizar capas ocultas (Tabla 12) como haciendo uso de las mismas (Tabla 13).

Ajuste del número de neuronas y filtros

Como se puede observar de los resultados obtenidos tanto en la Tabla 12 como en la Tabla 13, a la hora de realizar una clasificación con 3 polaridades añadir 3 capas ocultas supone una mejora de casi un 10% en los resultados.

Número de neuronas en capa convolucional	Filtros para capa LSTM	Dropout para capa convolucional	Dropout para capa LSTM	Tamaño del Kernel	Accuracy Validación cruzada
192	256	0.2	0.3	8	55.06%
192	128	0.2	0.3	8	**55.72%**
192	96	0.2	0.3	8	**50.25%**
192	64	0.2	0.3	8	54.73%
180	256	0.2	0.3	8	54.56%
180	96	0.2	0.3	8	54.39%
160	128	0.2	0.3	8	54.06%
160	64	0.2	0.3	8	53.07%
150	256	0.2	0.3	8	50.41%
128	128	0.2	0.3	8	52.07%
128	64	0.2	0.3	8	53.07%
96	64	0.2	0.3	8	52.90%
192	150	0.2	0.3	8	51.74%

Tabla 12: resultados obtenidos para variación de neuronas y filtros LSTM sin capas ocultas

Número de neuronas en capa convolucional	Filtros para capa LSTM	Neuronas capas ocultas	Dropout para capa convolucional	Dropout para capa LSTM	Tamaño del Kernel	Accuracy Validación cruzada
192	256	256,128,64	0.2	0.3	8	55.06%
192	128	256,128,64	0.2	0.3	8	65.01%
192	96	256.128.64	0.2	0.3	8	63.85%
192	64	256,128,64	0.2	0.3	8	55.06%
180	256	256,128,64	0.2	0.3	8	64.51%
180	96	256,128,64	0.2	0.3	8	52.40%
160	128	256,128,64	0.2	0.3	8	52.74%
160	64	256,128,64	0.2	0.3	8	59.87%
150	256	256,128,64	0.2	0.3	8	51.41%

128	128	256,128,64	0.2	0.3	8	62.69%
128	64	256,128,64	0.2	0.3	8	54.89%
96	64	256,128,64	0.2	0.3	8	53.90%
192	150	256,128,64	0.2	0.3	8	53.90%

Tabla 13: resultados obtenidos para variación de neuronas y filtros LSTM con capas ocultas

A continuación, en la Tabla 14 se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, número de neuronas igual a 192 y número de filtros LSTM igual a 128. En la Tabla 14 observamos las métricas previamente mencionadas y que se busca optimizar, partiendo de unas precisiones del 64% para la polaridad negativa y un 68% para la polaridad positiva.

Polaridad	Precisión	Recall	F1-score	Support
Negativo	64%	66%	65%	243
Positivo	68%	73%	70%	297
Indeterminado	43%	25%	32%	63
Global (weighted avg)	64%	65%	64%	603

Tabla 14: valores de métricas para optimización de neuronas y filtros LSTM

Ajuste de las tasas de dropout

Como se demostró también para las neuronas y filtros en el apartado anterior, para el estudio con tres polaridades los hiperparámetros afectan en mayor medida a los resultados obtenidos, tal y como se observa en los resultados de la Tabla 15.

Número de neuronas en capa convolucional	Filtros para capa LSTM	Neuronas capas ocultas	Dropout para capa convolucional	Dropout para capa LSTM	Tamaño del Kernel	Accuracy Validación cruzada
192	128	256,128,64	0.2	0.3	8	65.01%
192	128	256,128,64	0.3	0.3	8	53.07%
192	128	256.128.64	0.4	0.3	8	55.22%
192	128	256,128,64	0.5	0.3	8	55.39%
192	128	256,128,64	0.6	0.3	8	56.88%
192	128	256,128,64	0.7	0.3	8	54.73%
192	128	256,128,64	0.8	0.3	8	58.21%
192	128	256,128,64	0.2	0.4	8	52.74%
192	128	256,128,64	0.2	0.5	8	64.84%
192	128	256,128,64	0.2	0.6	8	54.39%
192	128	256,128,64	0.2	0.7	8	53.23%
192	128	256,128,64	0.2	0.8	8	53.90%

Tabla 15: resultados obtenidos para variación de tasas de dropout

En la Tabla 16 se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, Dropout de la capa convolucional igual a 0.2 y Dropout de la capa LSTM igual a 0.3.

Polaridad	Precisión	Recall	F1-score	Support
Negativo	64%	66%	65%	243
Positivo	68%	73%	70%	297
Indeterminado	43%	25%	32%	63
Global (weighted avg)	64%	65%	64%	603

Tabla 16: valores de métricas para optimización de tasas de dropout

En la Tabla 4 observamos las métricas previamente mencionadas y, se observa como las tres polaridades mantienen el nivel de precisión previa a la variación de las tasas de dropout lo cual nos indica que ya estaban optimizadas.

Ajuste de tasa de aprendizaje del optimizador Adam

Tal y como se observa en la Tabla 17 para la tasa de aprendizaje se observa que, como para el caso de dos polaridades, no se produce una variación considerable en la métrica de precisión del modelo al modificar el valor de dicho parámetro.

Tasa de aprendizaje	Accuracy Validación cruzada
0.001	65.01%
0.003	65.01%
0.005	65.51%
0.007	61.69%
0.009	64.34%
0.01	64.84%

Tabla 17: resultados obtenidos para variación de tasa de aprendizaje

Como resultado del estudio de la tasa de aprendizaje (Tabla 17) se puede observar que, dependiendo del valor que se le asigne, se produce una diferencia de casi un 4% en los resultados obtenidos. A continuación, se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, una tasa de aprendizaje igual a 0.005.

Polaridad	Precision	Recall	F1-score	Support
Negativo	64%	69%	67%	243
Positivo	73%	70%	71%	297
Indeterminado	36%	33%	34%	63
Global (weighted avg)	65%	66%	65%	603

Tabla 18: valores de métricas para optimización de tasa de aprendizaje

En la Tabla 18 observamos las métricas previamente mencionadas y que se busca optimizar, se observa como la polaridad negativa ha mantenido su precisión en un 64% mientras que la precisión para la polaridad positiva ha aumentado en un 5%.

Reducción del número total de palabras únicas del corpus.

Como se mostraba en el caso de dos polaridades, al disminuir el número de palabras se obtienen mejores resultados, teniendo cuidado de no descartar demasiadas palabras como para que se produzca una pérdida de información relevante para el modelo, tal y como se muestra en la Tabla 19.

Tamaño vocabulario	Accuracy Validación cruzada
todo	65.51%
1653	68.16%
1553	70.32%
1453	70.81%
1253	72.14%
1153	69.98%

Tabla 19: resultados obtenidos para variación de tamaño del vocabulario

Es apreciable cómo la utilización de un menor número de palabras, descartando aquellas de menor relevancia nos conduce en un principio a un aumento en la precisión del modelo, pero que al seguir disminuyendo el número ocurre que precisión empieza a disminuir, ya que llegará un momento en el que se comenzarán a descartar palabras importantes del corpus. En la Tabla 20 se muestran los valores de las métricas para la configuración con una mayor tasa de precisión, esto es, tamaño del vocabulario igual a 1253 palabras.

Polaridad	Precision	Recall	F1-score	Support
Negativo	75%	65%	70%	243
Positivo	73%	85%	78%	297
Indeterminado	54%	41%	47%	63
Global (weighted avg)	72%	72%	72%	603

Tabla 20: valores de métricas para optimización de tamaño del vocabulario

En la Tabla 20 se observa como la polaridad negativa aumenta su precisión en un 11% mientras que la precisión para la polaridad positiva se mantiene constante.

Ajuste del tamaño del batch size

En la Tabla 21 se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, un batch_size igual a 512.

batch size	Accuracy Validación cruzada
32	72.80%
64	72.14%
128	73.13%
256	72.14%
512	73.63%

Tabla 21: resultados obtenidos para variación de batch_size

Resumen de hiperparámetros del modelo final y resultados

En la Figura 16 se muestra el resultado final del modelo una vez se han realizado todos los ajustes de hiperparámetros que se han ido mencionando paso a paso.

```
from keras.models import Sequential
from sklearn.utils.multiclass import type_of_target
from tensorflow.keras.callbacks import EarlyStopping
import numpy as np
from keras.layers import Embedding, Flatten, Dense, LSTM, Conv1D, GlobalMaxPooling1D, MaxPooling1D, Bidirectional, GRU
from sklearn.model_selection import KFold
import statistics
from tensorflow.keras.optimizers import Adam

X = np.concatenate((X_train, X_test), axis=0)
y = np.concatenate((y_train, y_test), axis=0)

#y=np.argmax(y,axis=1)
print(type_of_target(y))
acc_per_fold=[]
loss_per_fold=[]
print(type_of_target(y))
kf=KFold(n_splits=5, shuffle=True, random_state=999)
cvscores=[]
for train, test in kf.split(X_train, y_train):
  model = Sequential()
  embedding_layer = Embedding(vocab_size, 200, input_length=maxlen)
  #embedding_layer = Embedding(vocab_size, 300, input_length=maxlen, weights=[embedding_matrix],trainable=False)

  model.add(embedding_layer)

  model.add(Conv1D(192, 8, activation='relu'))

  model.add(MaxPooling1D(10))

  model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.3))

  model.add(Dense(256, activation='relu'))

  # Agregar más capas ocultas
  model.add(Dense(128, activation='relu'))
  model.add(Dense(64, activation='relu'))

  #Si queremos 3 clases usaremos 3 neuronas en la última capa densa, por el contrario usaremos 4
  model.add(Dense(3, activation='softmax'))

  model.compile(optimizer=Adam(learning_rate=0.005), loss='categorical_crossentropy', metrics=['accuracy'])
  model.summary()


  early_stop = EarlyStopping(monitor = 'accuracy', mode = 'max', verbose = 1, patience = 5)
  model.fit(X[train], y[train], epochs=100, batch_size=512, verbose=1,validation_data = (X[test], y[test]), callbacks=[early_stop])

  scores = model.evaluate(X[test], y[test], verbose=1)
  print(f'Score for fold : {model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')
  acc_per_fold.append(scores[1] * 100)
  loss_per_fold.append(scores[0])
print(acc_per_fold)
print(statistics.mean(acc_per_fold))
```

Figura 16: modelo tras optimización 3 polaridades

En dicha figura se aprecia la concatenación de los subconjuntos train y test, la configuración del bucle, los valores finales para el número de neuronas de la capa

convolucional, en este caso 192, el número de filtros LSTM, en este caso 128, las tasas de dropout que son 0.2 y 0.3 respectivamente, el optimizador Adam con un learning rate de 0.005 y el valor del batch_size de 512. La configuración de parámetros que ha permitido optimizar el modelo se representa en la Tabla 22 a modo de resumen, con dichos parámetros y valores alcanzados.

Parámetros	Valores
Número Neuronas	192
Número filtros	128
Capas ocultas	256,128,64
Tasa dropout	0.2
Tasa recurrent_dropout	0.3
Tasa de aprendizaje	0.005
Vocab_size	1253
Batch_size	512

Tabla 22: parámetros optimizados

En la Tabla 23 se observan los siguientes valores:

- Para polaridad negativa: precisión del 73%, lo cual nos dice el porcentaje de las predicciones negativas que en realidad lo eran. Recall del 71% nos indica que de todos los mensajes que en realidad eran negativos solo se escogieron como tal el 71%.

- Para polaridad positiva: precisión del 75%, lo cual nos dice el porcentaje de las predicciones negativas que en realidad lo eran. Recall del 84% nos indica que de todos los mensajes que en realidad eran negativos solo se escogieron como tal el 84%.

Polaridad	Precision	Recall	F1-score	Support
Negativo	73%	71%	72%	243
Positivo	75%	84%	79%	297
Indeterminado	63%	35%	45%	63
Global (weighted avg)	73%	74%	73%	603

Tabla 23: valores de métricas para configuración óptima 3 polaridades

Por otro lado, la matriz de confusión aporta una representación visual de las predicciones realizadas por el modelo y compararlas con las clases a las que verdaderamente pertenecen los datos tratados. En este caso en concreto (Figura 17), se aprecia como el modelo experimenta el mayor grado de dificultad a la hora de clasificar los mensajes de polaridad Indeterminada, debido a la posible ambigüedad que representan los mismos, ya que normalmente los mensajes suelen tener inherente una polaridad y no

son en muchas ocasiones neutros totalmente. También se puede deber al desbalance existente entre las clases de datos, como se puede observar en la Figura 11, puesto que el número de mensajes de polaridad indeterminada es bastante inferior al resto de clases. Otro aspecto destacable es el hecho de que para los mensajes cuya polaridad es verdaderamente negativa, se está produciendo un error considerable al confundirlos con mensajes de polaridad positiva, esto se ve representado en la Figura 17 con un valor de 62 el cual es un 25.5% de la totalidad de los mensajes de polaridad verdaderamente negativa. Una manera de solucionarlo podría ser investigar la existencia de patrones específicos o características que pudiesen estar causando la confusión.

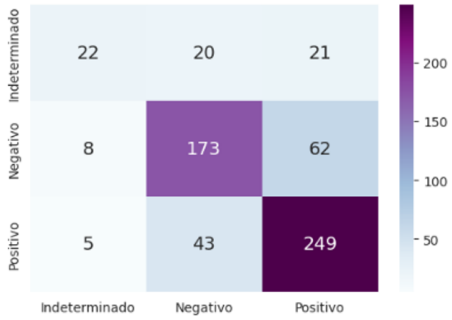


Figura 17: matriz de confusión para configuración optima 3 polaridades

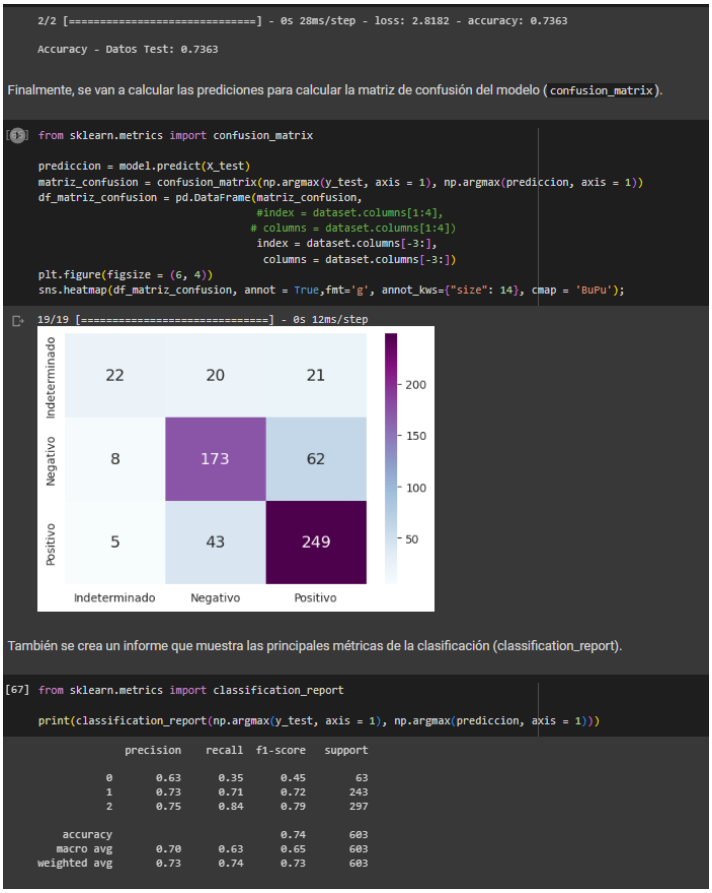


Figura 18: resultados para configuración óptima 3 polaridades

En la Figura 18 se observa una representación de los valores finales obtenidos una vez se han optimizado todos los hiperparámetros.

4.4 Análisis de la polaridad en dataset de salud mental

4.4.1 Definición del dataset de salud mental

El corpus de mensajes de salud mental consta de 2286 mensajes, los cuales están clasificados con su correspondiente polaridad, que puede ser de 3 tipos diferentes: Positiva, Negativa e Indeterminada. Del corpus se obtienen los datos de partida, esto es, los mensajes y su polaridad, que posteriormente se usarán para el entrenamiento del modelo de Deep learning.

En este caso, la distribución de los mensajes es la siguiente: 1526 positivos, 587 negativos y 173 indeterminados, y su distribución se muestra en la Figura 19.

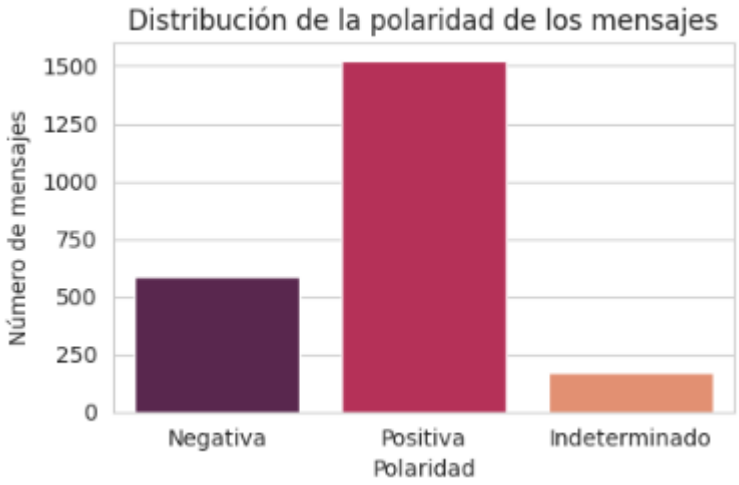


Figura 19: distribución corpus salud mental por polaridad

Dado que cada mensaje solo puede pertenecer a una polaridad, estamos ante un problema de clasificación de etiqueta única y multiclase.

Se puede observar en la Figura 19 que la polaridad positiva tiene una mayor cantidad de muestras en comparación con la negativa e indeterminada, lo que podría ocasionar un problema de desbalance, haciendo así que el modelo prediga con mayor probabilidad que el mensaje pertenece a la clase con más muestras.

En la Figura 20, se presenta una nube de palabras, la cual representa de manera gráfica las palabras de mayor relevancia para una polaridad específica, en este caso la polaridad positiva.

Figura 20: nube de palabras de la clase positiva

4.4.2 Resultados de polaridad para el dataset de salud mental

4.4.2.1 Análisis con dos polaridades

Entrenamiento y evaluación del modelo de Deep learning

Para entrenar y evaluar el funcionamiento del modelo el conjunto de datos fue dividido en dos subconjuntos: entrenamiento (70%) y prueba (30%). Para el entrenamiento se han realizado distintos tipos de pruebas con el objetivo de ver las variaciones en la métrica de precisión que conllevaban. Estas pruebas son: ajuste del número de neuronas y los filtros LSTM, ajuste de las tasas de dropout y acto seguido el de la tasa de aprendizaje para el optimizador Adam. A continuación, se hicieron pruebas con la reducción del número total de palabras únicas escogidas de entre todas las del corpus y por último un

ajuste del tamaño de batch size utilizado. Todo esto con la finalidad de ajustar los hiperparámetros y evitar el sobreajuste en el modelo desarrollado.

Durante la fase de entrenamiento se siguieron varios pasos:

- Preparación de los datos: se obtienen dos conjuntos de datos, primero el conjunto 'X', resultado de concatenar 'X_train' y 'X_test' y posteriormente el conjunto 'y' resultado de concatenar 'y_train' e 'y_test'.

- Se realiza la validación cruzada: para la validación cruzada se utiliza un objeto 'KFold' con 5 divisiones y a su vez se definen dos listas, 'acc_per_fold' y 'loss_per_fold' las cuales almacenan los resultados de precisión y pérdidas respectivamente.

- Bucle de validación cruzada: el cual va a iterar en relación a los índices de entrenamiento generados por el 'KFold'.

A su vez, la fase de prueba se realizó teniendo en cuenta varios ejemplos de métricas de rendimiento: precisión, recall y F1-score. Estas métricas permiten obtener una medida cuantitativa de la capacidad del modelo para clasificar de manera acertada los mensajes dentro de cada una de las diferentes categorías que se barajan.

Ajuste del número de neuronas y filtros

Como primera etapa de la fase de entrenamiento se va a repetir el proceso de entrenamiento variando el número de neuronas y filtros LSTM dentro de un rango de valores con el fin de encontrar la combinación que proporcione mejores resultados, también va a servir para obtener una idea general de cómo afecta la arquitectura del modelo al rendimiento del mismo.

Se muestra a continuación la Tabla 24 a modo de resumen para los resultados obtenidos con la variación de neuronas y filtros LSTM. A su vez se resalta en gris la configuración para la cual se obtienen los mejores resultados.

Número de neuronas en capa convolucional	Filtros para capa LSTM	Dropout para capa convolucional	Dropout para capa LSTM	Tamaño del Kernel	Accuracy Validación cruzada
192	256	0.2	0.3	8	82.65%
192	128	0.2	0.3	8	84.07%
192	96	0.2	0.3	8	83.91%
192	64	0.2	0.3	8	84.23%
180	256	0.2	0.3	8	83.75%
180	96	0.2	0.3	8	83.28%
160	128	0.2	0.3	8	82.97%
160	64	0.2	0.3	8	84.54%
150	256	0.2	0.3	8	82.33%
128	128	0.2	0.3	8	71.29%
128	64	0.2	0.3	8	84.07%
96	64	0.2	0.3	8	84.38%
192	150	0.2	0.3	8	82.97%

Tabla 24: resultados obtenidos para variación de neuronas y filtros LSTM

Observamos en la Tabla 24 que los parámetros muestran un nivel de ajuste bastante bueno, no existiendo una gran diferencia en los resultados obtenidos modificando el valor de dichos hiperparámetros exceptuando un caso en el que se produce un sobreentrenamiento y la precisión baja en más de un 10% del optimo. A continuación, en la Tabla 25 se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, número de neuronas igual a 160 y número de filtros LSTM igual a 64. En la Tabla 25 se muestran las métricas previamente mencionadas, partiendo de unas precisiones del 77% para la polaridad negativa y un 87% para la polaridad positiva.

Polaridad	Precision	Recall	F1-score	Support
Negativo	77%	66%	71%	182
Positivo	87%	92%	89%	452
Global (weighted avg)	84%	85%	84%	634

Tabla 25: valores de métricas para optimización de neuronas y filtros LSTM

Ajuste de las tasas de dropout

El segundo paso del entrenamiento del modelo va a ser, partiendo de la mejor configuración previamente obtenida, variar las tasas de dropout en un intervalo que recorra valores desde los 0.2 a los 0.8 para ambos casos. En concreto se han obtenido los valores de precisión representados en la Tabla 26 a modo de resumen.

Número de neuronas en capa convolucional	Filtros para capa LSTM	Dropout para capa convolucional	Dropout para capa LSTM	Tamaño del Kernel	Accuray Validación cruzada
160	64	0.2	0.3	8	84.54%
160	64	0.3	0.3	8	84.38%
160	64	0.4	0.3	8	83.60%
160	64	0.5	0.3	8	83.91%
160	64	0.6	0.3	8	84.86%
160	64	0.7	0.3	8	84.23%
160	64	0.8	0.3	8	71.29%
160	64	0.2	0.4	8	83.60%
160	64	0.2	0.5	8	82.97%
160	64	0.2	0.6	8	85.02%
160	64	0.2	0.7	8	82.81%
160	64	0.2	0.8	8	82.65%

Tabla 26: resultados obtenidos para variación de tasas de dropout

Analizando los resultados obtenidos destacamos el hecho de que realizar una variación de los valores de las tasas de dropout no genera en sí un cambio significativo en la precisión del modelo, en este caso no llegando ni a suponer una fuerte mejora, salvo en un caso donde se produce una bajada considerable debido a un sobreentrenamiento. A continuación, en la Tabla 27 se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, dropout de la capa convolucional igual a 0.2 y dropout de la capa LSTM igual a 0.6. En la Tabla 27 se observa como la precisión para la polaridad negativa ha aumentado un 1%, mientras que la polaridad positiva se mantiene en un 87%.

Polaridad	Precision	Recall	F1-score	Support
Negativo	78%	66%	72%	182
Positivo	87%	93%	90%	452
Global (weighted avg)	85%	85%	85%	634

Tabla 27: valores de métricas para optimización de tasas de dropout

Ajuste de tasa de aprendizaje del optimizador Adam

El siguiente paso a seguir en el modelo, es un barrido para distintos valores de la tasa de aprendizaje. Es un factor fundamental a la hora de entrenar modelos de aprendizaje automáticos, puesto que determinará el grado de magnitud con el cual se realizarán los ajustes a los distintos parámetros del modelo, lo cual afecta a su vez a la convergencia del mismo. Hay dos motivos principales por los cuales resulta interesante controlar la tasa de aprendizaje: controlar la velocidad de convergencia y superar mínimos locales que se puedan producir. Cabe destacar que encontrar la tasa de convergencia óptima es un proceso empírico, es decir, que requiere de prueba y error para encontrar el resultado que mejor se adecue al problema en particular que se intenta resolver. Como resultado del estudio de la tasa de aprendizaje (Tabla 28) se puede observar en la Tabla 28 que dependiendo del valor que se le asigne, se produce una diferencia de casi un 4% en los resultados obtenidos.

Tasa de aprendizaje	Accuracy Validación cruzada
0.001	85.02%
0.003	83.28%
0.005	81.70%
0.007	83.44%
0.009	81.86%
0.01	81.07%

Tabla 28: resultados obtenidos para variación de tasa de aprendizaje

A continuación, en la Tabla 29 se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, una tasa de aprendizaje igual a 0.001. En la Tabla 29 se observa como ambas polaridades mantienen el nivel de precisión previa variación de la tasa de aprendizaje, lo cual nos indica que ya estaba optimizada.

Polaridad	Precison	Recall	F1-score	Support
Negativo	78%	66%	72%	182
Positivo	87%	93%	90%	452
Global (weighted avg)	85%	85%	85%	634

Tabla 29: valores de métricas para optimización de tasas de aprendizaje

Reducción del número total de palabras del corpus

El siguiente paso del entrenamiento es realizar una reducción del número total de palabras únicas del corpus utilizadas en el modelo final, con el fin de aumentar la eficiencia computacional. De este modo se descartan las palabras de menor relevancia, las cuales se clasificaron teniendo en cuenta tanto la frecuencia con que se repetían, así como la ganancia de información que aportaban.

La reducción del vocabulario utilizado puede llevar a una mayor eficiencia computacional y un aumento en la velocidad de entrenamiento del modelo. Existe a su vez un compromiso entre la reducción del vocabulario utilizado y la posibilidad de perder información importante, por lo que es necesario realizar las reducciones de manera escalonada (en este caso en saltos de 100 palabras), como se representa en la Tabla 30.

Tamaño vocabulario	Accuracy Validación cruzada
todas	85.02%
2421	86.28%
2321	88.33%
2221	87.54%
2121	86.28%
2021	87.22%
1921	85.49%

Tabla 30: resultados obtenidos para variación de tamaño del vocabulario

Es apreciable cómo la utilización de un menor número de palabras, descartando aquellas de menor relevancia nos conduce en un principio a un aumento en la precisión del modelo, pero que al seguir disminuyendo el número ocurre que la precisión empieza a disminuir, ya que llegará un momento en el que se comenzarán a descartar palabras importantes del corpus. En la Tabla 31, se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, tamaño del vocabulario igual a 2321 palabras. En la Tabla 31 se observa como la polaridad negativa aumenta su precisión en un 3% mientras que la polaridad positiva aumenta su precisión en un 4%.

Polaridad	Precision	Recall	F1-score	Support
Negativo	81%	77%	79%	182
Positivo	91%	93%	92%	452
Global (weighted avg)	88%	88%	88%	634

Tabla 31: valores de métricas para optimización de tamaño del vocabulario

Ajuste del tamaño del batch_size.

Como último paso en la fase de entrenamiento del modelo se realiza una variación del parámetro batch_size. Si bien un valor elevado de batch_size lleva a una convergencia más rápida y un rendimiento mejorado, hay que tener en cuenta también que si es demasiado grande se puede producir un exceso de demanda para la memoria y los recursos computacionales. Por el contrario, si se utiliza un valor pequeño, se obtiene una mayor variabilidad para los datos de entrenamiento, lo que es interesante ya que ayudaría a solventar problemas como pueden ser los estancamientos de óptimos locales. Cabe mencionar también que es una alternativa muy recomendada si se tienen recursos computacionales limitados. Se observa en la Tabla 32 que el equilibrio entre eficiencia y rendimiento se encuentra para un batch_size de valor 128.

batch size	Accuracy Validación cruzada
32	90.06%
64	89.43%
128	90.22%
256	88.33%
512	88.96%

Tabla 32: resultados obtenidos para variación del batch_size

Resumen de hiperparámetros del modelo final y resultados

En la Figura 21 se muestra el resultado final del modelo una vez se han realizado todos los ajustes de hiperparámetros que se han ido mencionando paso a paso. En el se aprecia la concatenación de los subconjuntos train y test, la configuración del bucle, los valores finales para el número de neuronas de la capa convolucional, en este caso 160, el número de filtros LSTM, en este caso 64, las tasas de dropout que son 0.2 y 0.6 respectivamente, el optimizador Adam con un learning rate de 0.001 y el valor del batch_size de 128.

La configuración de parámetros que ha permitido optimizar el modelo de Deep learning se representa a modo de resumen en la Tabla 33.

```
from keras.models import Sequential
from sklearn.utils.multiclass import type_of_target
from tensorflow.keras.callbacks import EarlyStopping
import numpy as np
from keras.layers import Embedding, Flatten, Dense, LSTM, Conv1D, GlobalMaxPooling1D, MaxPooling1D, Bidirectional, GRU
from sklearn.model_selection import KFold
import statistics
from tensorflow.keras.optimizers import Adam

X = np.concatenate((X_train, X_test), axis=0)
y = np.concatenate((y_train, y_test), axis=0)

#y=np.argmax(y,axis=1)
print(type_of_target(y))
acc_per_fold=[]
loss_per_fold=[]
print(type_of_target(y))
kf=KFold(n_splits=5, shuffle=True, random_state=999)
cvscores=[]
for train, test in kf.split(X_train, y_train):
  model = Sequential()
  embedding_layer = Embedding(vocab_size, 200, input_length=maxlen)
  #embedding_layer = Embedding(vocab_size, 300, input_length=maxlen, weights=[embedding_matrix],trainable=False)

  model.add(embedding_layer)

  model.add(Conv1D(160, 8, activation='relu'))

  model.add(MaxPooling1D(10))

  model.add(LSTM(64, dropout=0.2, recurrent_dropout=0.6))


  #Si queremos 3 clases usaremos 3 neuronas en la última capa densa, por el contrario usaremos 4
  #model.add(Dense(3, activation='softmax'))
  #Si queremos 2 clases usaremos 3 neuronas en la última capa densa, por el contrario usaremos 4
  model.add(Dense(2, activation='softmax'))
  model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
  model.summary()


  early_stop = EarlyStopping(monitor = 'accuracy', mode = 'max', verbose = 1, patience = 5)
  model.fit(X[train], y[train], epochs=100, batch_size=128, verbose=1,validation_data = (X[test], y[test]), callbacks=[early_stop])

  scores = model.evaluate(X[test], y[test], verbose=1)
  print(f'Score for fold : {model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')
  acc_per_fold.append(scores[1] * 100)
  loss_per_fold.append(scores[0])
print(acc_per_fold)
print(statistics.mean(acc_per_fold))
```

Figura 21: modelo final tras optimizaciones

Parámetros	Valores
Número Neuronas	160
Número filtros	64
Tasa de Dropout	0.2
Tasa Recurrent_dropout	0.6
Tasa de aprendizaje	0.001
Tamaño vocabulario	2321
Batch_size	128

Tabla 33: parámetros optimizados

Finalmente, en la Tabla 34 se observan los siguientes valores para las diferentes métricas de evaluación de nuestro modelo optimizado:

- Para polaridad negativa: precisión del 84%, lo cual nos dice el porcentaje de las predicciones negativas que en realidad lo eran. Recall del 81% nos indica que de todos los mensajes que en realidad eran negativos solo se escogieron como tal el 81%.

- Para polaridad positiva: precisión del 93%, lo cual nos dice el porcentaje de las predicciones negativas que en realidad lo eran. Recall del 90% nos indica que de todos los mensajes que en realidad eran negativos solo se escogieron como tal el 90%.

Polaridad	Precision	Recall	F1-score	Support
Negativo	84%	81%	83%	182
Positivo	93%	94%	93%	452
Global (weighted avg)	90%	90%	90%	634

Tabla 34: valores de métricas para configuración óptima

Por otro lado, la matriz de confusión indica en la Figura 22 como el modelo predice más veces, de manera errónea, mensajes de polaridad positiva, cuando en realidad son negativos. Esto se puede deber a la representación de características, falta de información discriminatoria o al desbalance en el tamaño de los datos. Por otro lado, los mensajes de polaridad positiva real los predice con una precisión elevada.

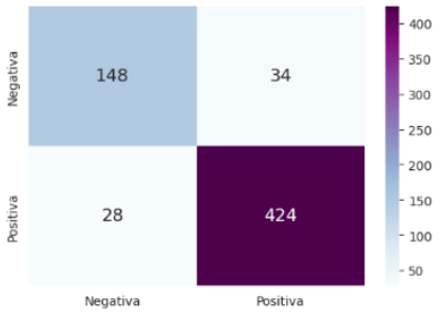


Figura 22: matriz de confusión para configuración óptima

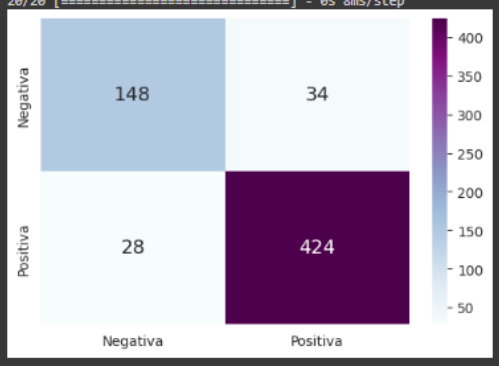
Finalmente, en la Figura 23 se muestra una representación de los valores finales obtenidos una vez se han optimizado todos los hiperparámetros.

```
5/5 [==============================] - 0s 32ms/step - loss: 0.4712 - accuracy: 0.9022

Accuracy - Datos Test: 0.9022
```

Finalmente, se van a calcular las prediciones para calcular la matriz de confusión del modelo (`confusion_matrix`).

```
[51] from sklearn.metrics import confusion_matrix

     prediccion = model.predict(X_test)
     matriz_confusion = confusion_matrix(np.argmax(y_test, axis = 1), np.argmax(prediccion, axis =
     1))
     df_matriz_confusion = pd.DataFrame(matriz_confusion,
                                        #index = dataset.columns[1:4],
                                       # columns = dataset.columns[1:4])
                                        index = dataset.columns[-2:],
                                         columns = dataset.columns[-2:])
     plt.figure(figsize = (6, 4))
     sns.heatmap(df_matriz_confusion, annot = True,fmt='g', annot_kws={"size": 14}, cmap = 'BuPu');
```

```
20/20 [==============================] - 0s 8ms/step
```



También se crea un informe que muestra las principales métricas de la clasificación (classification_report).

```
[52] from sklearn.metrics import classification_report

     print(classification_report(np.argmax(y_test, axis = 1), np.argmax(prediccion, axis = 1)))
```

	precision	recall	f1-score	support
0	0.84	0.81	0.83	182
1	0.93	0.94	0.93	452
accuracy			0.90	634
macro avg	0.88	0.88	0.88	634
weighted avg	0.90	0.90	0.90	634

Figura 23: resultados para configuración óptima

4.4.2.2 Análisis con tres polaridades

Ajuste del número de neuronas y filtros

El primer paso es realizar un estudio sobre el efecto que tiene variar el número de neuronas y filtros LSTM, esto viene representado en la Tabla 35.

Número de neuronas en capa convolucional	Filtros para capa LSTM	Dropout para capa convolucional	Dropout para capa LSTM	Tamaño del Kernel	Accuracy Validación cruzada
192	256	0.2	0.3	8	76.97%
192	128	0.2	0.3	8	65.74%
192	96	0.2	0.3	8	76.82%
192	64	0.2	0.3	8	77.11%
180	256	0.2	0.3	8	79.15%
180	96	0.2	0.3	8	65.74%
160	128	0.2	0.3	8	77.84%
160	64	0.2	0.3	8	77.55%
150	256	0.2	0.3	8	76.24%
128	128	0.2	0.3	8	65.74%
128	64	0.2	0.3	8	65.74%
96	64	0.2	0.3	8	65.74%
192	150	0.2	0.3	8	76.82%

Tabla 35: resultados obtenidos para variación de neuronas y filtros LSTM

Como se puede observar de los resultados obtenidos tanto en la Tabla 35 a la hora de realizar una clasificación con 3 clases, el modelo demuestra ser bastante sensible a variaciones en los hiperparámetros.

A continuación, en la Tabla 36 se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, número de neuronas igual a 180 y número de filtros LSTM igual a 256. La Tabla 36 muestra las métricas previamente mencionadas, partiendo de unas precisiones del 72% para la polaridad negativa y un 86% para la polaridad positiva.

Polaridad	Precision	Recall	F1-score	Support
Indeterminado	36%	33%	34%	49
Negativo	72%	69%	71%	186
Positivo	86%	88%	87%	451
Global (weighted avg)	79%	79%	79%	686

Tabla 36: valores de métricas para optimización de neuronas y filtros LSTM

Ajuste de las tasas de dropout

Como se demostró también para las neuronas y filtros en el apartado anterior, para el estudio con tres polaridades los hiperparámetros afectan en mayor medida a los resultados obtenidos, tal y como se observa en los resultados de la Tabla 37.

Número de neuronas en capa convolucional	Filtros para capa LSTM	Dropout para capa convolucional	Dropout para capa LSTM	Tamaño del Kernel	Accuracy Validación cruzada
180	256	0.2	0.3	8	79.15%
180	256	0.3	0.3	8	79.01%
180	256	0.4	0.3	8	77.11%
180	256	0.5	0.3	8	65.74%
180	256	0.6	0.3	8	65.74%
180	256	0.7	0.3	8	65.74%
180	256	0.8	0.3	8	65.745
180	256	0.2	0.4	8	75.955
180	256	0.2	0.5	8	76.245
180	256	0.2	0.6	8	77.705
180	256	0.2	0.7	8	78.575
180	256	0.2	0.8	8	77.705

Tabla 37: resultados obtenidos para variación de tasas de dropout

En la Tabla 38 se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, dropout de la capa convolucional igual a 0.2 y dropout de la capa LSTM igual a 0.3. En la Tabla 38 se observan las distintas métricas, y como las tres polaridades mantienen el nivel de precisión previa variación de los hiperparámetros, lo cual nos indica que ya estaban optimizados.

Polaridad	Precision	Recall	F1-score	Support
Indeterminado	36%	33%	34%	49
Negativo	72%	69%	71%	186
Positivo	86%	88%	87%	451
Global (weighted avg)	79%	79%	79%	686

Tabla 38: valores de métricas para optimización de tasas de dropout

Ajuste de tasa de aprendizaje del optimizador Adam

Tal y como se observa en la Tabla 39 para la tasa de aprendizaje se observa que, como para el caso de dos polaridades, no se produce una variación considerable en la métrica de precisión del modelo al modificar el valor de dicho parámetro.

Tasa de aprendizaje	Accuracy Validación cruzada
0.001	79.15%
0.003	77.70%
0.005	78.43%
0.007	77.84%
0.009	79.01%
0.01	79.01%

Tabla 39: resultados obtenidos para variación de tasa de aprendizaje

Como resultado del estudio de la tasa de aprendizaje (Tabla 39) se puede observar que, dependiendo del valor, se produce una diferencia de casi un 2% en los resultados. A continuación, se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, una tasa de aprendizaje igual a 0.001.

Polaridad	Precision	Recall	F1-score	Support
Indeterminado	36%	33%	34%	49
Negativo	72%	69%	71%	186
Positivo	86%	88%	87%	451
Global (weighted avg)	79%	79%	79%	686

Tabla 40: valores de métricas para optimización de tasas de aprendizaje

En la Tabla 40 se observa como las tres polaridades mantienen el nivel de precisión previo a la variación de la tasa de aprendizaje, lo cual nos indica que ya estaba optimizada.

Reducción del número total de palabras únicas del corpus

Como se mostraba en el caso de dos polaridades, al disminuir el número de palabras se obtienen mejores resultados, teniendo cuidado de no descartar demasiadas palabras como para que se produzca una pérdida de información relevante para el modelo, tal y como se muestra en la Tabla 41. Es apreciable cómo la utilización de un menor número de palabras, descartando aquellas de menor relevancia nos conduce en un principio a un aumento en la precisión del modelo, pero que al seguir disminuyendo el número la precisión empieza a disminuir, ya que llegará un momento en el que se comenzarán a descartar palabras importantes del corpus.

Tamaño vocabulario	Accuracy Validación cruzada
Todas	79.15%
2421	78.28%
2321	80.61%
2221	81.05%
2121	77.70%
2021	78.72%

Tabla 41: resultados obtenidos para variación de tamaño del vocabulario

En la Tabla 42 se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, tamaño del vocabulario igual a 2221 palabras. En la Tabla 42 se observa como la polaridad negativa aumenta su precisión en un 1% mientras que la precisión para la polaridad positiva aumenta en un 2%.

Polaridad	Precision	Recall	F1-score	Support
Indeterminado	40%	33%	36%	49
Negativo	73%	74%	73%	186
Positivo	88%	89%	89%	451
Global (weighted avg)	81%	81%	81%	686

Tabla 42: valores de métricas para optimización de tamaño de vocabulario

Ajuste del tamaño del batch size

En la Tabla 43 se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, un batch_size igual a 512.

batch size	Accuracy Validación cruzada
32	81.34%
64	80.32%
128	80.61%
256	81.05%
512	65.74%

Tabla 43: resultados obtenidos para variación de batch_size

Resumen de hiperparámetros del modelo final y resultados

En la Figura 24 se muestra el resultado final del modelo una vez se han realizado todos los ajustes de hiperparámetros que se han ido mencionando paso a paso. Se aprecia la concatenación de los subconjuntos train y test, la configuración del bucle, los valores finales para el número de neuronas de la capa convolucional, en este caso 180, el número de filtros LSTM, en este caso 256, las tasas de dropout que son 0.2 y 0.3 respectivamente, el optimizador Adam con un learning rate de 0.001 y el valor del batch_size de 32.

```
from keras.models import Sequential
from sklearn.utils.multiclass import type_of_target
from tensorflow.keras.callbacks import EarlyStopping
import numpy as np
from keras.layers import Embedding, Flatten, Dense, LSTM, Conv1D, GlobalMaxPooling1D, MaxPooling1D, Bidirectional, GRU
from sklearn.model_selection import KFold
import statistics
from tensorflow.keras.optimizers import Adam

X = np.concatenate((X_train, X_test), axis=0)
y = np.concatenate((y_train, y_test), axis=0)

#y=np.argmax(y,axis=1)
print(type_of_target(y))
acc_per_fold=[]
loss_per_fold=[]
print(type_of_target(y))
kf=KFold(n_splits=5, shuffle=True, random_state=999)
cvscores=[]
for train, test in kf.split(X_train, y_train):
  model = Sequential()
  embedding_layer = Embedding(vocab_size, 200, input_length=maxlen)
  #embedding_layer = Embedding(vocab_size, 300, input_length=maxlen, weights=[embedding_matrix],trainable=False)

  model.add(embedding_layer)

  model.add(Conv1D(180, 8, activation='relu'))

  model.add(MaxPooling1D(10))

  model.add(LSTM(256, dropout=0.2, recurrent_dropout=0.3))


  #Si queremos 3 clases usaremos 3 neuronas en la última capa densa, por el contrario usaremos 4
  model.add(Dense(3, activation='softmax'))
  #Si queremos 2 clases usaremos 3 neuronas en la última capa densa, por el contrario usaremos 4
  #model.add(Dense(2, activation='softmax'))
  model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
  model.summary()


  early_stop = EarlyStopping(monitor = 'accuracy', mode = 'max', verbose = 1, patience = 5)
  model.fit(X[train], y[train], epochs=100, batch_size=32, verbose=1,validation_data = (X[test], y[test]), callbacks=[early_stop])

  scores = model.evaluate(X[test], y[test], verbose=1)
  print(f'Score for fold : {model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')
  acc_per_fold.append(scores[1] * 100)
  loss_per_fold.append(scores[0])
print(acc_per_fold)
print(statistics.mean(acc_per_fold))
```

Figura 24: modelo tras optimización 3 polaridades

La configuración de parámetros que ha permitido optimizar el modelo se representa en la Tabla 44 a modo de resumen, con dichos parámetros y valores alcanzados.

Parámetros	Valores
Número Neuronas	192
Número filtros	128
Capas ocultas	256,128,64
Tasa dropout	0.2
Tasa recurrent_dropout	0.3
Tasa de aprendizaje	0.005
Vocab_size	1253
Batch_size	512

Tabla 44: parámetros optimizados

Finalmente, en la Tabla 45 se observan los siguientes valores:

- Para polaridad negativa: precisión del 78%, lo cual nos dice el porcentaje de las predicciones negativas que en realidad lo eran. Recall del 67% nos indica que de todos los mensajes que en realidad eran negativos solo se escogieron como tal el 67%.

- Para polaridad positiva: precisión del 85%, lo cual nos dice el porcentaje de las predicciones negativas que en realidad lo eran. Recall del 94% nos

indica que de todos los mensajes que en realidad eran negativos solo se escogieron como tal el 94%.

Polaridad	Precision	Recall	F1-score	Support
Indeterminado	42%	22%	29%	49
Negativo	78%	67%	72%	186
Positivo	85%	94%	89%	451
Global (weighted avg)	80%	81%	80%	686

Tabla 45: valores de métricas para configuración óptima 3 polaridades

Por otro lado, en la matriz de confusión (Figura 25), se aprecia como el modelo experimenta el mayor grado de dificultad a la hora de clasificar los mensajes de polaridad Indeterminada, debido a la posible ambigüedad que representan los mismos, ya que normalmente los mensajes suelen tener inherente una polaridad y no son en muchas ocasiones neutros totalmente. También se puede deber al desbalance existente entre las clases de datos, como se puede observar en la Figura 19, puesto que el número de mensajes de polaridad indeterminada y negativa son bastante inferiores a los de polaridad positiva. Otro aspecto destacable es el hecho de que para los mensajes cuya polaridad es verdaderamente negativa, se está produciendo un error considerable al confundirlos con mensajes de polaridad positiva, esto se ve representado en la Figura 25 con un valor de 50 el cual es un 26.88% de la totalidad de los mensajes de polaridad verdaderamente negativa. Una manera de solucionarlo podría ser investigar la existencia de patrones específicos o características que pudiesen estar causando la confusión.

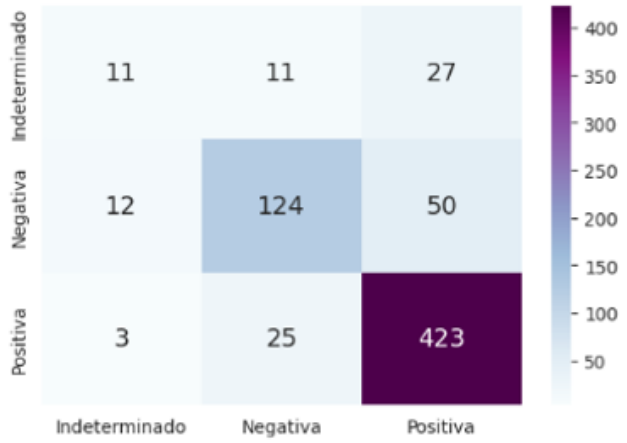


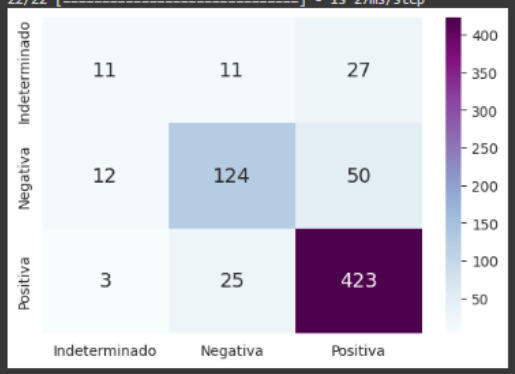
Figura 25: matriz de confusión para configuración óptima con 3 polaridades

```
22/22 [==============================] - 2s 27ms/step - loss: 1.6840 - accuracy: 0.8134

Accuracy - Datos Test: 0.8134
```

Finalmente, se van a calcular las prediciones para calcular la matriz de confusión del modelo (`confusion_matrix`).

```
[77] from sklearn.metrics import confusion_matrix

     prediccion = model.predict(X_test)
     matriz_confusion = confusion_matrix(np.argmax(y_test, axis = 1), np.argmax(prediccion, axis = 1))
     df_matriz_confusion = pd.DataFrame(matriz_confusion,
                                        #index = dataset.columns[1:4],
                                       # columns = dataset.columns[1:4])
                                        index = dataset.columns[-3:],
                                         columns = dataset.columns[-3:])
     plt.figure(figsize = (6, 4))
     sns.heatmap(df_matriz_confusion, annot = True,fmt='g', annot_kws={"size": 14}, cmap = 'BuPu');
```

```
22/22 [==============================] - 1s 27ms/step
```



También se crea un informe que muestra las principales métricas de la clasificación (classification_report).

```
[78] from sklearn.metrics import classification_report

     print(classification_report(np.argmax(y_test, axis = 1), np.argmax(prediccion, axis = 1)))
```

	precision	recall	f1-score	support
0	0.42	0.22	0.29	49
1	0.78	0.67	0.72	186
2	0.85	0.94	0.89	451
accuracy			0.81	686
macro avg	0.68	0.61	0.63	686
weighted avg	0.80	0.81	0.80	686

Figura 26: resultados obtenidos para configuración óptima 3 polaridades

En la Figura 26 se observa una representación de los valores finales obtenidos una vez se han optimizado todos los hiperparámetro.

5

Análisis de la respuesta emocional en redes sociales: Emociones

5.1 Introducción

Este capítulo de la memoria se centrará en realizar una optimización de los hiperparámetros utilizados para analizar las respuestas emocionales presentes en los corpus del apartado anterior, pero en este caso estará enfocado en detectar y clasificar las emociones expresadas en cada mensaje.

5.2 Análisis de emociones en Twitch, salud mental y Twitter

5.2.1 Definición del dataset de Twitch

El corpus de mensajes de Twitch consta de 2007 mensajes, los cuales están clasificados con su correspondiente emoción, que puede ser de 7 tipos diferentes: aprobación, desinterés, decepción, desaprobación, enfado, interés e indeterminado. Del corpus se obtienen los datos de partida, esto es, los mensajes y la emoción que evoca, y estos datos posteriormente se usarán para el entrenamiento del modelo de Deep Learning.

En este caso, la distribución de los mensajes es la siguiente: 711 de aprobación, 171 de desinterés, 182 de decepción, 246 de desaprobación, 168 de enfado, 268 de interés y 261 de indeterminado. Su distribución se muestra en la Figura 27.

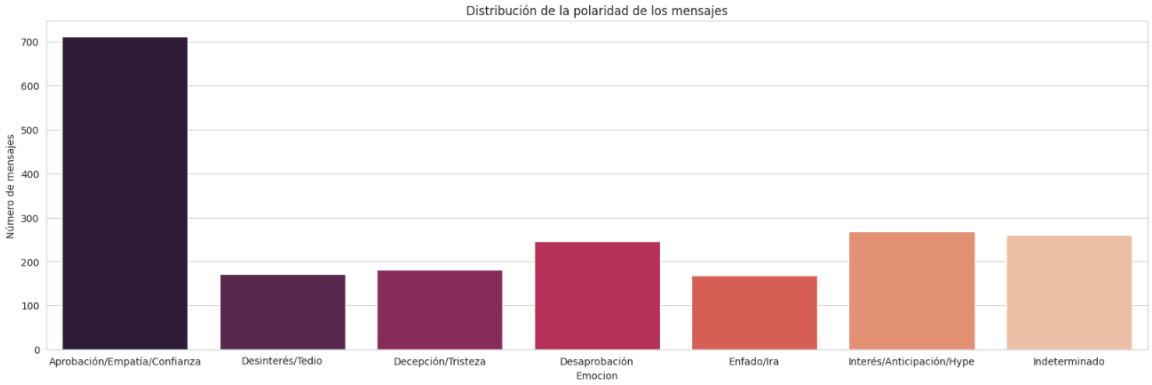


Figura 27: distribución del corpus por emociones

Dado que cada mensaje solo puede pertenecer a una polaridad, estamos ante un problema de clasificación de etiqueta única y multiclase. Se puede observar en la Figura 27 que la emoción de aprobación tiene una mayor cantidad de muestras en comparación con el resto, lo que podría ocasionar un problema de desbalance, haciendo así que el modelo prediga con mayor probabilidad que el mensaje pertenece a la clase con más muestras. En la Figura 28, se presenta una nube de palabras, la cual representa de manera gráfica las palabras de mayor relevancia para una emoción específica, en este caso la emoción de desaprobación.

Figura 28: nube de palabras de la emoción de desaprobación

5.2.2 Resultados de emociones para el dataset de Twitch

5.2.2.1 Análisis con 7 emociones

Entrenamiento y evaluación del modelo de Deep learning

Para entrenar y evaluar el funcionamiento del modelo el conjunto de datos fue dividido en dos subconjuntos: entrenamiento (70%) y prueba (30%). Para el entrenamiento se han realizado distintos tipos de pruebas con el objetivo de ver las variaciones en la métrica de precisión que conllevaban. Estas pruebas son: ajuste del número de neuronas y los filtros LSTM, ajuste de las tasas de dropout y acto seguido el de la tasa de aprendizaje para el optimizador Adam. A continuación, se hicieron pruebas con la reducción del número total de palabras únicas escogidas de entre todas las del corpus y por último un ajuste del tamaño de batch size utilizado. Todo esto con la finalidad de ajustar los hiperparámetros y evitar el sobreajuste en el modelo desarrollado.

Durante la fase de entrenamiento se siguieron varios pasos:

- Preparación de los datos: se obtienen dos conjuntos de datos, primero el conjunto ‘X’, resultado de concatenar ‘X_train’ y ‘X_test’ y posteriormente el conjunto ‘y’ resultado de concatenar ‘y_train’ e ‘y_test’.

- Se realiza la validación cruzada: para la validación cruzada se utiliza un objeto ‘KFold’ con 5 divisiones y a su vez se definen dos listas, ‘acc_per_fold’ y ‘loss_per_fold’ las cuales almacenan los resultados de precisión y pérdidas respectivamente.

- Bucle de validación cruzada: el cual va a iterar en relación a los índices de entrenamiento generados por el ‘KFold’.

A su vez, la fase de prueba se realizó teniendo en cuenta varios ejemplos de métricas de rendimiento: precisión, recall y F1-score. Estas métricas permiten obtener una medida cuantitativa de la capacidad del modelo para clasificar de manera acertada los mensajes dentro de cada una de las diferentes categorías que se barajan.

Ajuste del número de neuronas y filtros

Como primera etapa de la fase de entrenamiento se va a repetir el proceso de entrenamiento variando el número de neuronas y filtros LSTM dentro de un rango de

valores con el fin de encontrar la combinación que proporcione mejores resultados. También va a servir para obtener una idea general de cómo afecta la arquitectura del modelo al rendimiento del mismo. Para este caso, se realizó el estudio simultaneo de los resultados sin utilizar capas ocultas (Tabla 46) y utilizando las mismas (Tabla 47).

Número de neuronas en capa convolucional	Filtros para capa LSTM	Dropout para capa convolucional	Dropout para capa LSTM	Tamaño del Kernel	Accuracy Validación cruzada
192	256	0.2	0.3	8	46.43%
192	128	0.2	0.3	8	44.44%
192	96	0.2	0.3	8	42.62%
192	64	0.2	0.3	8	43.62%
180	256	0.2	0.3	8	41.96%
180	96	0.2	0.3	8	40.80%
160	128	0.2	0.3	8	43.45%
160	64	0.2	0.3	8	44.28%
150	256	0.2	0.3	8	39.30%
128	128	0.2	0.3	8	42.45%
128	64	0.2	0.3	8	43.28%
96	64	0.2	0.3	8	43.45%
192	112	0.2	0.3	8	45.11%

Tabla 46: resultados obtenidos para variación de neuronas y filtros LSTM sin capas ocultas

Número de neuronas en capa convolucional	Filtros para capa LSTM	Neuronas capas ocultas	Dropout para capa convolucional	Dropout para capa LSTM	Tamaño del Kernel	Accuracy Validación cruzada
192	256	256,128,64	0.2	0.3	8	37.81%
192	128	256,128,64	0.2	0.3	8	39.64%
192	96	256.128.64	0.2	0.3	8	41.63%
192	64	256,128,64	0.2	0.3	8	38.47%
180	256	256,128,64	0.2	0.3	8	42.95%
180	96	256,128,64	0.2	0.3	8	41.63%
160	128	256,128,64	0.2	0.3	8	42.95%
160	64	256,128,64	0.2	0.3	8	36.15%
150	256	256,128,64	0.2	0.3	8	39.145
128	128	256,128,64	0.2	0.3	8	37.15%
128	64	256,128,64	0.2	0.3	8	39.80%
96	64	256,128,64	0.2	0.3	8	35.66%
192	112	256,128,64	0.2	0.3	8	36.55%

Tabla 47: resultados obtenidos para variación de neuronas y filtros LSTM con capas ocultas

Como se puede observar de los resultados obtenidos tanto en la Tabla 46 como en la Tabla 47, a la hora de realizar una clasificación con 7 emociones añadir 3 capas ocultas no supone una mejora. Otro aspecto a destacar es la sensibilidad del modelo a los cambios de los hiperparámetros, lo cual se puede comprobar en la diferencia de aproximadamente un 7% de precisión entre el valor más alto y el más bajo de la Tabla 46.

A continuación, en la Tabla 48 se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, número de neuronas igual a 192 y número de filtros LSTM igual a 256. En dicha tabla se muestran los valores de las métricas previamente mencionadas, a su vez se observan las precisiones de partida para cada emoción, destacando las emociones de polaridad positiva: aprobación e interés. Esto puede explicarse viendo la Figura 27 donde aprobación tiene más muestras que el resto de clases y relacionándolo con la Figura 11 donde se nos muestra que la polaridad positiva es la que más muestras acumula, las cuales solo se dividen luego en dos emociones. Extrapolando esto a las clases negativas, las cuales, al tener menos muestras divididas en más emociones, se clasificarán de manera menos precisa

Emoción	Precision	Recall	F1-score	Support
Aprobación	56%	69%	62%	220
Decepción	22%	15%	18%	52
Desaprobación	29%	28%	28%	69
Desinterés	32%	28%	30%	50
Enfado	32%	30%	31%	53
Indeterminado	29%	24%	26%	72
Interés	69%	62%	65%	87
Global (weighted avg)	45%	46%	45%	603

Tabla 48: valores de métricas para optimización de neuronas y filtros LSTM

Ajuste de las tasas de dropout

Como se demostró también para las neuronas y filtros en el apartado anterior, para el estudio con 7 emociones los hiperparámetros afectan en mayor medida a los resultados obtenidos, tal y como se observa en los resultados de la Tabla 49.

Número de neuronas en capa convolucional	Filtros para capa LSTM	Dropout para capa convolucional	Dropout para capa LSTM	Tamaño del Kernel	Accuracy Validación cruzada
192	256	0.2	0.3	8	46.43%
192	256	0.3	0.3	8	45.61%
192	256	0.4	0.3	8	44.94%
192	256	0.5	0.3	8	37.15%
192	256	0.6	0.3	8	44.28%
192	256	0.7	0.3	8	44.94%
192	256	0.8	0.3	8	43.62%
192	256	0.2	0.4	8	43.62%
192	256	0.2	0.5	8	41.46%
192	256	0.2	0.6	8	41.79%
192	256	0.2	0.7	8	45.27%
192	256	0.2	0.8	8	43.62%
192	256	0.3	0.3	8	45.61%

Tabla 49: resultados obtenidos para variación de tasas de dropout

En la Tabla 50 se muestran los valores de las métricas para la configuración con una mayor tasa de precisión, esto es, dropout de la capa convolucional igual a 0.2 y dropout de la capa LSTM igual a 0.3. En la Tabla 50 se observa que las 7 emociones mantienen el nivel de precisión previa a la variación de las tasas de dropout, lo cual nos indica que ya estaban optimizadas.

Emoción	Precision	Recall	F1-score	Support
Aprobación	56%	69%	62%	220
Decepción	22%	15%	18%	52
Desaprobación	29%	28%	28%	69
Desinterés	32%	28%	30%	50
Enfado	32%	30%	31%	53
Indeterminado	29%	24%	26%	72
Interés	69%	62%	65%	87
Global (weighted avg)	45%	46%	45%	603

Tabla 50: valores de métricas para optimización de tasas de dropout

Ajuste de tasa de aprendizaje del optimizador Adam

Como resultado del estudio de la tasa de aprendizaje (Tabla 51) se puede observar que, dependiendo del valor que se le asigne, se produce una diferencia de aproximadamente un 7% en los resultados obtenidos. A continuación, se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión,

esto es, una tasa de aprendizaje igual a 0.01. En la Tabla 52 se muestra el valor de las métricas, u se observa que las 7 emociones mantienen el nivel de precisión previa a la variación de la tasa de aprendizaje, lo cual nos indica que ya estaba optimizada.

Tasa de aprendizaje	**Accuracy Validación cruzada**
0.006	40.30%
0.008	39.97%
0.01	46.43%
0.012	43.45%
0.014	44.44%
0.02	39.14%

Tabla 51: resultados obtenidos para variación de tasa de aprendizaje

Emoción	Precision	Recall	F1-score	Support
Aprobación	56%	69%	62%	220
Decepción	22%	15%	18%	52
Desaprobación	29%	28%	28%	69
Desinterés	32%	28%	30%	50
Enfado	32%	30%	31%	53
Indeterminado	29%	24%	26%	72
Interés	69%	62%	65%	87
Global (weighted avg)	45%	46%	45%	603

Tabla 52: valores de métricas para optimización de tasa de aprendizaje

Reducción del número total de palabras únicas del corpus

Como se mostraba en el estudio de polaridades, al disminuir el número de palabras se obtienen mejores resultados, teniendo cuidado de no descartar demasiadas palabras como para que se produzca una pérdida de información relevante para el modelo, tal y como se muestra en la Tabla 53.

Tamaño vocabulario	**Accuracy Validación cruzada**
Todas	46.43%
1653	47.76%
1553	50.75%
1453	42.62%
1253	49.25%
1153	45.44%

Tabla 53: resultados obtenidos para variación de tamaño del vocabulario

Es apreciable cómo la utilización de un menor número de palabras nos conduce en un principio a un aumento en la precisión del modelo, pero que al seguir disminuyendo el número la precisión empieza a disminuir, ya que llegará un momento en el que se comenzarán a descartar palabras importantes del corpus. En la Tabla 54 se muestran los

valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, tamaño del vocabulario igual a 1553 palabras. En la Tabla 54 observamos como algunas emociones han sufrido bajadas en su precisión (aprobación de un 1% e interés de un 7%) mientras que el resto han subido considerablemente, consiguiendo así una mayor precisión global.

Emoción	Precision	Recall	F1-score	Support
Aprobación	55%	80%	65%	220
Decepción	40%	19%	26%	52
Desaprobación	35%	26%	30%	69
Desinterés	50%	30%	37%	50
Enfado	35%	26%	30%	53
Indeterminado	34%	21%	26%	72
Interés	62%	68%	65%	87
Global (weighted avg)	48%	51%	48%	603

Tabla 54: valores de métricas para optimización de tamaño del vocabulario

Ajuste del tamaño del batch size

En la Tabla 55 se muestran los resultados obtenidos para el proceso de optimización del batch_size, donde el mejor resultado se observa para un valor de 256.

batch size	Accuracy Validación cruzada
32	48.09%
64	44.11%
128	47.26%
256	50.75%
512	46.93%

Tabla 55: resultados obtenidos para variación del batch_size

Resumen de hiperparámetros del modelo final y resultados

En la Figura 29 se muestra el resultado final del modelo una vez se han realizado todos los ajustes de hiperparámetros que se han ido mencionando paso a paso. Se aprecia la concatenación de los subconjuntos train y test, la configuración del bucle, los valores finales para el número de neuronas de la capa convolucional, en este caso 192, el número de filtros LSTM, en este caso 256, las tasas de dropout que son 0.2 y 0.3 respectivamente, el optimizador Adam con un learning rate de 0.01 y el valor del batch_size de 256. La configuración de parámetros que ha permitido optimizar el modelo se representa en la Tabla 56 a modo de resumen, con dichos parámetros y valores alcanzados.

```python
from keras.models import Sequential
from sklearn.utils.multiclass import type_of_target
from tensorflow.keras.callbacks import EarlyStopping
import numpy as np
from keras.layers import Embedding, Flatten, Dense, LSTM, Conv1D, GlobalMaxPooling1D, MaxPooling1D, Bidirectional, GRU
from sklearn.model_selection import KFold
import statistics
from tensorflow.keras.optimizers import Adam


X = np.concatenate((X_train, X_test), axis=0)
y = np.concatenate((y_train, y_test), axis=0)
filters = 192
units = 256
#y=np.argmax(y,axis=1)
print(type_of_target(y))
acc_per_fold=[]
loss_per_fold=[]
kf=KFold(n_splits=5, shuffle=True, random_state=999)
#ACUERDATE
cvscores=[]
for train, test in kf.split(X_train,y_train):
  model = Sequential()
  embedding_layer = Embedding(vocab_size, 200, input_length=maxlen)
  #embedding_layer = Embedding(vocab_size, 300, input_length=maxlen, weights=[embedding_matrix],trainable=False)

  model.add(embedding_layer)

  model.add(Conv1D(filters, 8, activation='relu'))

  model.add(MaxPooling1D(10))

  model.add(LSTM(units, dropout=0.2, recurrent_dropout=0.3))
  #model.add(Dense(256, activation='relu'))

  # Agregar más capas ocultas
  #model.add(Dense(128, activation='relu'))
 # model.add(Dense(64, activation='relu'))

  #Si queremos 7 clases usaremos 7 neuronas en la última capa densa
  model.add(Dense(7, activation='softmax')), #hay que poner la , para que entienda que no son la misma linea esta y la de abajo luego vuelve a cambiar

  model.compile(optimizer=Adam(learning_rate=0.01), loss='categorical_crossentropy', metrics=['accuracy'])


  model.summary()


  early_stop = EarlyStopping(monitor = 'accuracy', mode = 'max', verbose = 1, patience = 5) #cambio un poco el patience para que se entrene más el sis
  model.fit(X[train], y[train], epochs=100, batch_size=256, verbose=1,validation_data = (X[test], y[test]), callbacks=[early_stop])

  scores = model.evaluate(X[test], y[test], verbose=1)
  print(f'Score for fold : {model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')
  acc_per_fold.append(scores[1] * 100)
  loss_per_fold.append(scores[0])
print(acc_per_fold)
print(statistics.mean(acc_per_fold))
```

Figura 29: modelo tras optimización de 7 emociones

Parámetros	Valores
Número Neuronas	192
Número filtros	256
Tasa dropout	0.2
Tasa recurrent_dropout	0.3
Tasa de aprendizaje	0.01
Vocab_size	1553
Batch_size	256

Tabla 56: parámetros optimizados

En la Tabla 57 se muestran los valores para todas las métricas. Se observa que las dos emociones que mejor reconoce son aprobación e interés, mientras que las emociones que más confunde son desaprobación, desinterés y enfado, las cuales pueden guardar características en común que lleven a confusión a la hora de que el modelo las clasifique.

Emoción	Precision	Recall	F1-score	Support
Aprobación	55%	80%	65%	220
Decepción	40%	19%	26%	52
Desaprobación	35%	26%	30%	69
Desinterés	50%	30%	37%	50
Enfado	35%	26%	30%	53
Indeterminado	34%	21%	26%	72
Interés	62%	68%	65%	87
Global (weighted avg)	48%	51%	48%	603

Tabla 57: valores de métricas para configuración optima 7 emociones

Por otro lado, la matriz de confusión de la Figura 30, muestra como el modelo experimenta el mayor grado de dificultad a la hora de clasificar los mensajes las emociones desaprobación y enfado, debido a la posible ambigüedad que representan los mismos. También se puede deber al desbalance existente entre las clases de datos, como se puede observar en la Figura 27. Se puede destacar también el hecho de que se predicen muchos mensajes como emoción de aprobación, aunque no pertenezcan a la misma, esto se aprecia en la primera columna de la matriz de confusión, suceso el cual puede deberse al desbalance entre el tamaño de esta clase y el resto.

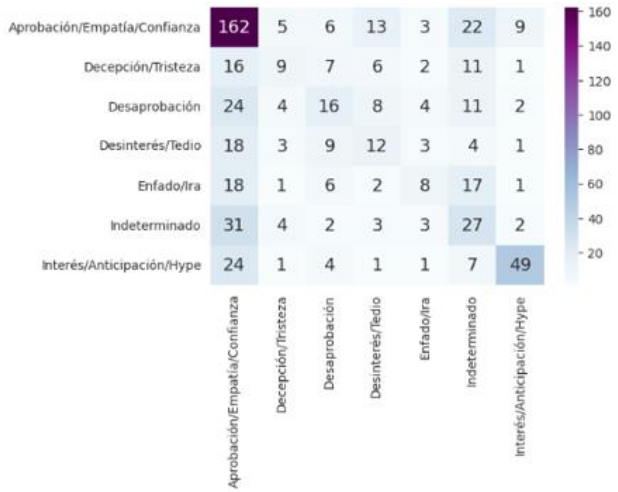


Figura 30: matriz de confusión para configuración óptima

5.2.3 Definición del dataset de salud mental

El corpus de mensajes de salud mental consta de 2287 mensajes, los cuales están clasificados con su correspondiente emoción, que puede ser de 6 tipos diferentes: amor, gratitud, comprensión, tristeza, enfado e indeterminado. Del corpus se obtienen los datos de partida, esto es, los mensajes y la emoción que evoca, estos datos posteriormente se usarán para el entrenamiento del modelo de Deep Learning.

En este caso, la distribución de los mensajes es la siguiente: 640 de amor, 227 de gratitud, 659 de comprensión, 122 de tristeza, 466 de enfado y 173 de indeterminado. Su distribución se muestra en la Figura 27.

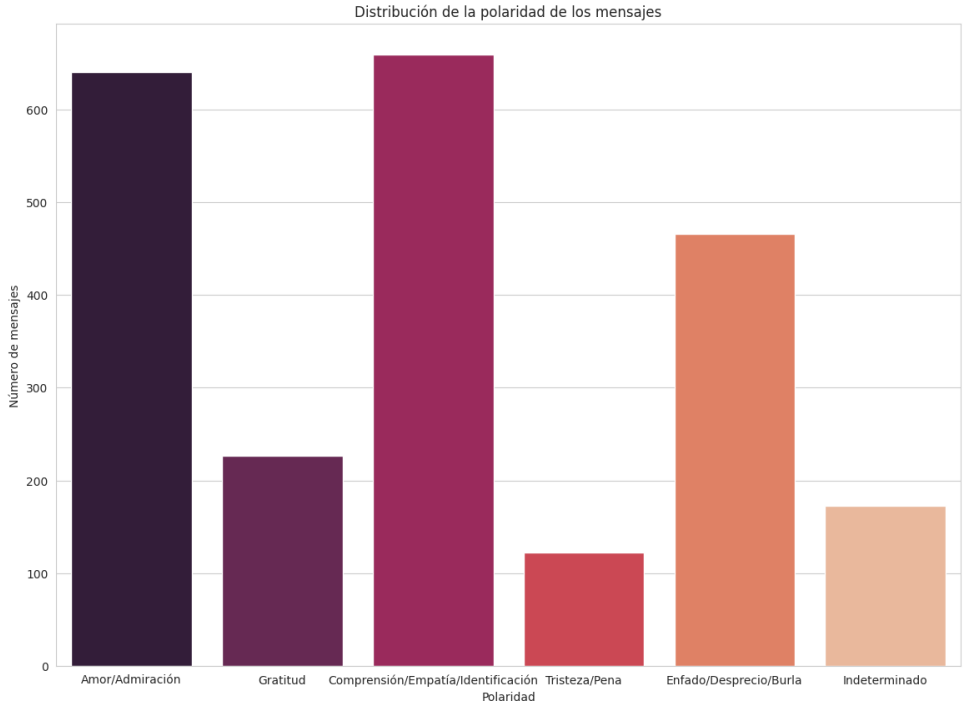


Figura 31:distribución del corpus por emociones

Dado que cada mensaje solo puede pertenecer a una polaridad, estamos ante un problema de clasificación de etiqueta única y multiclase.

Se puede observar en la Figura 31 que las emociones de amor y comprensión tienen una mayor cantidad de muestras en comparación con el resto, lo que podría ocasionar un problema de desbalance, haciendo así que el modelo prediga con mayor probabilidad que el mensaje pertenece a las clases con más muestras.

En la Figura 32, se presenta una nube de palabras, la cual representa de manera gráfica las palabras de mayor relevancia para una emoción específica, en este caso la emoción de tristeza.

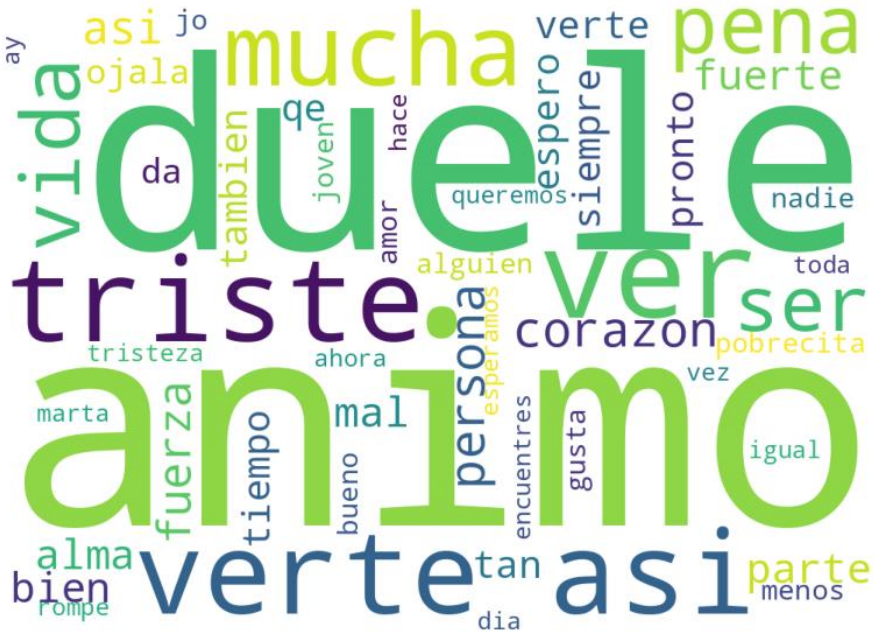


Figura 32:nube de palabras de la emoción de tristeza

5.2.4 Resultados de emociones para el dataset de salud mental

5.2.4.1 Análisis con 6 emociones

Entrenamiento y evaluación del modelo de Deep learning

Para entrenar y evaluar el funcionamiento del modelo el conjunto de datos fue dividido en dos subconjuntos: entrenamiento (70%) y prueba (30%). Para el entrenamiento se han realizado distintos tipos de pruebas con el objetivo de ver las variaciones en la métrica de precisión que conllevaban. Estas pruebas son: ajuste del número de neuronas y los filtros LSTM, ajuste de las tasas de dropout y acto seguido el de la tasa de aprendizaje para el optimizador Adam. A continuación, se hicieron pruebas con la reducción del número total de palabras únicas escogidas de entre todas las del corpus y por último un ajuste del tamaño de batch size utilizado. Todo esto con la finalidad de ajustar los hiperparámetros y evitar el sobreajuste en el modelo desarrollado.

Durante la fase de entrenamiento se siguieron varios pasos:

- Preparación de los datos: se obtienen dos conjuntos de datos, primero el conjunto ‘X’, resultado de concatenar ‘X_train’ y ‘X_test’ y posteriormente el conjunto ‘y’ resultado de concatenar ‘y_train’ e ‘y_test’.

- Se realiza la validación cruzada: para la validación cruzada se utiliza un objeto ‘KFold’ con 5 divisiones y a su vez se definen dos listas, ‘acc_per_fold’ y ‘loss_per_fold’ las cuales almacenan los resultados de precisión y pérdidas respectivamente.

- Bucle de validación cruzada: el cual va a iterar en relación a los índices de entrenamiento generados por el ‘KFold’.

A su vez, la fase de prueba se realizó teniendo en cuenta varios ejemplos de métricas de rendimiento: precisión, recall y F1-score. Permitiendo así obtener una medida cuantitativa de la capacidad del modelo para clasificar de manera acertada los mensajes dentro de cada una de las diferentes categorías que se barajan.

Ajuste del número de neuronas y filtros

Como primera etapa de la fase de entrenamiento se va a repetir el proceso de entrenamiento variando el número de neuronas y filtros LSTM dentro de un rango de valores con el fin de encontrar la combinación que proporcione mejores resultados, también va a servir para obtener una idea general de cómo afecta la arquitectura del modelo al rendimiento del mismo. En la Tabla 58 vienen recogidos los resultados obtenidos de variar el número de neuronas y filtros LSTM.

Número de neuronas en capa convolucional	Filtros para capa LSTM	Dropout para capa convolucional	Dropout para capa LSTM	Tamaño del Kernel	Accuracy Validación cruzada
192	256	0.2	0.3	8	67.83%
192	128	0.2	0.3	8	68.56%
192	96	0.2	0.3	8	68.41%
192	64	0.2	0.3	8	69.14%
180	256	0.2	0.3	8	69.14%
180	96	0.2	0.3	8	72.05%
160	128	0.2	0.3	8	67.25%
160	64	0.2	0.3	8	67.25%
150	256	0.2	0.3	8	66.67%
128	128	0.2	0.3	8	68.41%
128	64	0.2	0.3	8	69.29%
96	64	0.2	0.3	8	68.41%
180	128	0.2	0.3	8	69.14%

Tabla 58: resultados obtenidos para variación de neuronas y filtros LSTM

Un aspecto a destacar es la sensibilidad del modelo a los cambios de los hiperparámetros, lo cual se puede comprobar en la diferencia de aproximadamente un 7% de precisión entre el valor más alto y el más bajo de la Tabla 58. A continuación, en la Tabla 59 se muestran los valores de las métricas para la configuración con una mayor tasa de precisión, esto es, número de neuronas 180 y número de filtros LSTM 96.

Emoción	Precision	Recall	F1-score	Support
Amor	82%	79%	81%	207
Comprensión	64%	75%	69%	190
Enfado	86%	69%	76%	147
Gratitud	87%	89%	88%	61
Indeterminado	38%	34%	36%	41
Tristeza	43%	49%	45%	41
Global (weighted avg)	73%	72%	72%	687

Tabla 59: valores de métricas para optimización de neuronas y filtros LSTM

En la Tabla 59 observamos las métricas y a su vez las precisiones de partida para cada emoción. Las polaridades positivas tienen una mayor cantidad de muestras, lo cual se observa en la Figura 19, esto da sentido al hecho de que 2 de las 3 emociones que mejor está clasificando sean de polaridad positiva (amor y gratitud), mientras que enfado se puede razonar que tenga un nivel de precisión tan alto debido a que es la tercera emoción con más muestras de todo el corpus.

Ajuste de las tasas de dropout

Como se demostró también para las neuronas y filtros en el apartado anterior, para el estudio con 6 emociones los hiperparámetros afectan en mayor medida a los resultados obtenidos, tal y como se observa en los resultados de la Tabla 60.

Número de neuronas en capa convolucional	Filtros para capa LSTM	Dropout para capa convolucional	Dropout para capa LSTM	Tamaño del Kernel	Accuracy Validación cruzada
192	96	0.2	0.3	8	72.05%
192	96	0.3	0.3	8	65.50%
192	96	0.4	0.3	8	70.16%
192	96	0.5	0.3	8	69.87%
192	96	0.6	0.3	8	69.29%
192	96	0.7	0.3	8	69.14%
192	96	0.8	0.3	8	65.07%
192	96	0.2	0.4	8	67.69%
192	96	0.2	0.5	8	68.27%
192	96	0.2	0.6	8	68.41%
192	96	0.2	0.7	8	69.29%
192	96	0.2	0.8	8	68.85%
192	96	0.2	0.3	8	72.05%

Tabla 60: resultados obtenidos para variación de tasas de dropout

En la Tabla 61 se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, Dropout de la capa convolucional igual a 0.2 y Dropout de la capa LSTM igual a 0.3. En la Tabla 61 se observa que las 6 emociones mantienen el nivel de precisión previa variación de las tasas de dropout, lo cual nos indica que ya estaban optimizadas.

Emoción	Precision	Recall	F1-score	Support
Amor	82%	79%	81%	207
Comprensión	64%	75%	69%	190
Enfado	86%	69%	76%	147
Gratitud	87%	89%	88%	61
Indeterminado	38%	34%	36%	41
Tristeza	43%	49%	45%	41
Global (weighted avg)	73%	72%	72%	687

Tabla 61: valores de métricas para optimización de tasas de dropout

Ajuste de tasa de aprendizaje del optimizador Adam

Tal y como se observa en la Tabla 62 para la tasa de aprendizaje no se produce una variación considerable en la métrica de precisión del modelo al modificar el valor de dicho parámetro. Como resultado del estudio de la tasa de aprendizaje (Tabla 62) se puede observar que, dependiendo del valor que se le asigne, se produce una diferencia de aproximadamente un 7% en los resultados obtenidos. A continuación, se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, una tasa de aprendizaje igual a 0.01 (Tabla 63).

Tasa de aprendizaje	**Accuracy Validación cruzada**
0.006	66.81%
0.008	70.31%
0.01	72.05%
0.02	67.83%
0.04	38.43%

Tabla 62: resultados obtenidos para variación de tasa de aprendizaje

Emoción	Precision	Recall	F1-score	Support
Amor	82%	79%	81%	207
Comprensión	64%	75%	69%	190
Enfado	86%	69%	76%	147
Gratitud	87%	89%	88%	61
Indeterminado	38%	34%	36%	41
Tristeza	43%	49%	45%	41
Global (weighted avg)	73%	72%	72%	687

Tabla 63: valores de métricas para optimización de tasa de aprendizaje

En la Tabla 63, se observa que las 6 emociones mantienen el nivel de precisión previa variación de los hiperparámetros, lo cual nos indica que ya estaban optimizados.

Reducción del número total de palabras únicas del corpus

Como se mostraba en los estudios previos, al disminuir el número de palabras se obtienen mejores resultados, teniendo cuidado de no descartar demasiadas palabras como para que se produzca una pérdida de información relevante para el modelo, tal y como se muestra en la Tabla 64. En la Tabla 65 se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, tamaño del vocabulario igual a 1957 palabras.

Tamaño vocabulario	Accuracy Validación cruzada
Todas	72.05%
2457	69.14%
2357	68.56%
2257	69.58%
2157	70.60%
1957	72.49%

Tabla 64: resultados obtenidos para variación de tamaño del vocabulario

Emoción	Precision	Recall	F1-score	Support
Amor	77%	81%	79%	207
Comprensión	66%	77%	71%	190
Enfado	84%	68%	75%	147
Gratitud	85%	85%	65%	61
Indeterminado	34%	27%	30%	41
Tristeza	60%	51%	55%	41
Global (weighted avg)	73%	72%	72%	687

Tabla 65: valores de métricas para optimización de tamaño del vocabulario

En la Tabla 65 se observa como algunas emociones han sufrido bajadas en su precisión, amor de un 5%, enfado de un 2%, gratitud de un 2% e indeterminado de un 4%. Mientras que comprensión ha subido un 2% y tristeza un 17%, consiguiendo así una mayor precisión global.

Ajuste del tamaño del batch size

En la Tabla 66 se muestran los resultados obtenidos para el proceso de optimización del batch_size, donde el valor de 256 obtiene los mejores resultados.

batch size	Accuracy Validación cruzada
32	70.01%
64	70.89%
128	71.91%
256	72.49%
512	70.16%

Tabla 66: resultados obtenidos para variación del batch_size

Resumen de hiperparámetros del modelo final y resultados

En la Figura 33 se muestra el resultado final del modelo una vez se han realizado todos los ajustes de hiperparámetros que se han ido mencionando paso a paso. Se aprecia

la concatenación de los subconjuntos train y test, la configuración del bucle, los valores finales para el número de neuronas de la capa convolucional, en este caso 180, el número de filtros LSTM, en este caso 96, las tasas de dropout que son 0.2 y 0.3 respectivamente, el optimizador Adam con un learning rate de 0.01 y el valor del batch_size de 256.

```
from keras.models import Sequential
from sklearn.utils.multiclass import type_of_target
from tensorflow.keras.callbacks import EarlyStopping
import numpy as np
from keras.layers import Embedding, Flatten, Dense, LSTM, Conv1D, GlobalMaxPooling1D, MaxPooling1D, Bidirectional, GRU
from sklearn.model_selection import KFold
import statistics
from tensorflow.keras.optimizers import Adam


X = np.concatenate((X_train, X_test), axis=0)
y = np.concatenate((y_train, y_test), axis=0)
filters = 180
units = 96
#y=np.argmax(y,axis=1)
print(type_of_target(y))
acc_per_fold=[]
loss_per_fold=[]
kf=KFold(n_splits=5, shuffle=True, random_state=999)
#ACUERDATE
cvscores=[]
for train, test in kf.split(X_train, y_train):
  model = Sequential()
  embedding_layer = Embedding(vocab_size, 200, input_length=maxlen)
  #embedding_layer = Embedding(vocab_size, 300, input_length=maxlen, weights=[embedding_matrix],trainable=False)

  model.add(embedding_layer)

  model.add(Conv1D(filters, 8, activation='relu'))

  model.add(MaxPooling1D(10))

  model.add(LSTM(units, dropout=0.2, recurrent_dropout=0.3))
  model.add(Dense(256, activation='relu'))

  # Agregar más capas ocultas
  model.add(Dense(128, activation='relu'))
  model.add(Dense(64, activation='relu'))


  #Si queremos 6 clases (número de emociones) usaremos 6 neuronas en la última capa densa
  model.add(Dense(6, activation='softmax')), #hay que poner la , para que entienda que no son la misma linea esta y la de abajo luego v

  model.compile(optimizer=Adam(learning_rate=0.01), loss='categorical_crossentropy', metrics=['accuracy'])


  model.summary()


  early_stop = EarlyStopping(monitor = 'accuracy', mode = 'max', verbose = 1, patience = 5) #cambio un poco el patience para que se ent
  model.fit(X[train], y[train], epochs=100, batch_size=256, verbose=1,validation_data = (X[test], y[test]), callbacks=[early_stop])

  scores = model.evaluate(X[test], y[test], verbose=1)
  print(f'Score for fold : {model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')
  acc_per_fold.append(scores[1] * 100)
  loss_per_fold.append(scores[0])
print(acc_per_fold)
print(statistics.mean(acc_per_fold))
```

Figura 33: modelo tras optimización de 6 emociones

La configuración de parámetros que ha permitido optimizar el modelo se representa en la Tabla 67 a modo de resumen, con dichos parámetros y valores alcanzados. Finalmente, en la Tabla 68, se observa que las 6 emociones mantienen el nivel de precisión previa variación del valor de batch_size, lo cual nos indica que ya estábamos trabajando con el valor óptimo del mismo, que para este caso es 256.

Parámetros	Valores
Número Neuronas	180
Número filtros	96
Tasa dropout	0.2
Tasa recurrent_dropout	0.3
Tasa de aprendizaje	0.01
Vocab_size	1957
Batch_size	256

Tabla 67: parámetros optimizados

Emoción	Precision	Recall	F1-score	Support
Amor	77%	81%	79%	207
Comprensión	66%	77%	71%	190
Enfado	84%	68%	75%	147
Gratitud	85%	85%	65%	61
Indeterminado	34%	27%	30%	41
Tristeza	60%	51%	55%	41
Global (weighted avg)	73%	72%	72%	687

Tabla 68: valores de métricas para configuración optima 6 emociones

Por otro lado, la matriz de confusión para este caso concreto, Figura 34, indica como el modelo experimenta el mayor grado de dificultad a la hora de clasificar los mensajes de la emoción tristeza. También se puede deber al desbalance existente entre las clases de datos, como se puede observar en la Figura 31. Se puede destacar también el hecho de que se predicen muchos mensajes como emoción de comprensión, aunque no pertenezcan a la misma, esto se aprecia en la segunda columna de la matriz de confusión, suceso el cual puede deberse al desbalance entre el tamaño de esta clase y el resto.

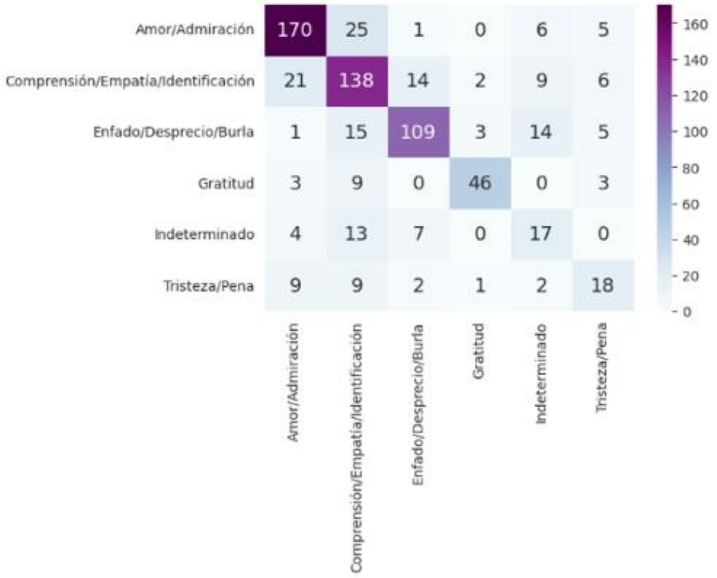


Figura 34: matriz de confusión para configuración óptima

Inclusión de capas ocultas

A modo de experimento se probó a añadir 3 capas ocultas siguiendo varias combinaciones de las mismas buscando una mejora en los resultados que ya se habían optimizado (Tabla 69). En la Tabla 69 al introducir 3 capas ocultas combinándolas de distintas maneras, no se produce una mejora en la precisión en ninguno de los casos.

Número de capas ocultas	Valores	Accuracy Validación cruzada
Una capa oculta	256	67.83%
Una capa oculta	128	69.29%
Una capa oculta	64	67.69%
Dos capas ocultas	256 y 128	66.67%
Dos capas ocultas	256 y 64	66.96%
Dos capas ocultas	128 y 64	67.69%
3 capas ocultas	256, 128 y 64	69.43%

Tabla 69: resultados obtenidos para variación de tasa de aprendizaje

5.2.5 Definición del dataset de Twitter

El corpus de mensajes de Twitter consta de 8409 mensajes, los cuales están clasificados con su correspondiente emoción, que puede ser de 7 tipos diferentes: enfado, asco, miedo, alegría, tristeza, sorpresa e indeterminado. Del corpus se obtienen los datos de partida, esto es, los mensajes y la emoción que evoca, estos datos posteriormente se usarán para el entrenamiento del modelo de Deep learning.

En este caso, la distribución de los mensajes es la siguiente: 857 de enfado, 161 de asco, 96 de miedo, 1815 de felicidad, 1009 de tristeza, 344 de sorpresa y 4127 de indeterminado. Su distribución se muestra en la Figura 27.

Dado que cada mensaje solo puede pertenecer a una polaridad, estamos ante un problema de clasificación de etiqueta única y multiclase.

Se puede observar en la Figura 35 que la emoción de alegría tiene una mayor cantidad de muestras en comparación con el resto, exceptuando la clase indeterminado, lo que podría ocasionar un problema de desbalance, haciendo así que el modelo prediga con mayor probabilidad que el mensaje pertenece a la clase con más muestras.

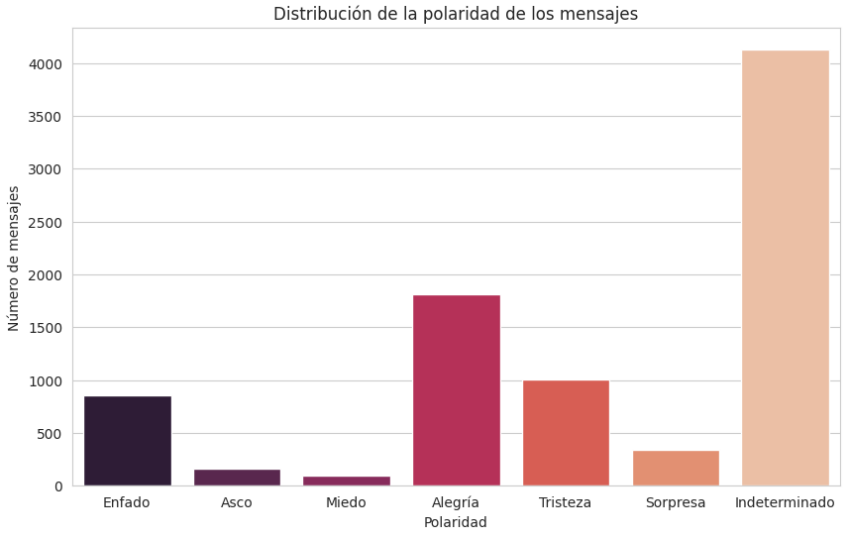


Figura 35: distribución del corpus por emociones

En la Figura 36, se presenta una nube de palabras, la cual representa de manera gráfica las palabras de mayor relevancia para cada emoción, en este caso alegría.

Figura 36: nube de palabras de la emoción de alegría

5.2.6 Resultados de emociones para el dataset de Twitter

5.2.6.1 Análisis con 7 emociones polaridades

Entrenamiento y evaluación del modelo de Deep learning

Para entrenar y evaluar el funcionamiento del modelo el conjunto de datos fue dividido en dos subconjuntos: entrenamiento (70%) y prueba (30%). Para el entrenamiento

se han realizado distintos tipos de pruebas con el objetivo de ver las variaciones en la métrica de precisión que conllevaban. Estas pruebas son: ajuste del número de neuronas y los filtros LSTM, ajuste de las tasas de dropout y acto seguido el de la tasa de aprendizaje para el optimizador Adam. A continuación, se hicieron pruebas con la reducción del número total de palabras únicas escogidas de entre todas las del corpus y por último un ajuste del tamaño de batch size utilizado. Todo esto con la finalidad de ajustar los hiperparámetros y evitar el sobreajuste en el modelo desarrollado.

Durante la fase de entrenamiento se siguieron varios pasos:

- Preparación de los datos: se obtienen dos conjuntos de datos, primero el conjunto 'X', resultado de concatenar 'X_train' y 'X_test' y posteriormente el conjunto 'y' resultado de concatenar 'y_train' e 'y_test'.

- Se realiza la validación cruzada: para la validación cruzada se utiliza un objeto 'KFold' con 5 divisiones y a su vez se definen dos listas, 'acc_per_fold' y 'loss_per_fold' las cuales almacenan los resultados de precisión y pérdidas respectivamente.

- Bucle de validación cruzada: el cual va a iterar en relación a los índices de entrenamiento generados por el 'KFold'.

A su vez, la fase de prueba se realizó teniendo en cuenta varios ejemplos de métricas de rendimiento: precisión, recall y F1-score. Permitiendo así obtener una medida cuantitativa de la capacidad del modelo para clasificar de manera acertada los mensajes dentro de cada una de las diferentes categorías que se barajan.

Ajuste del número de neuronas y filtros

Como primera etapa de la fase de entrenamiento se va a repetir el proceso de entrenamiento variando el número de neuronas y filtros LSTM dentro de un rango de valores con el fin de encontrar la combinación que proporcione mejores resultados.

En la Tabla 70 se recogen los resultados obtenidos de variar el número de neuronas y filtros LSTM. La primera conclusión a la que se puede llegar es que los parámetros muestran un nivel de ajuste bastante bueno, no existiendo una gran diferencia en los resultados obtenidos.

Número de neuronas en capa convolucional	Filtros para capa LSTM	Dropout para capa convolucional	Dropout para capa LSTM	Tamaño del Kernel	Accuracy Validación cruzada
192	256	0.2	0.3	8	54.14%
192	128	0.2	0.3	8	54.78%
192	96	0.2	0.3	8	55.89%
192	64	0.2	0.3	8	54.18%
180	256	0.2	0.3	8	56.64%
180	96	0.2	0.3	8	56.56%
160	128	0.2	0.3	8	55.89%
160	64	0.2	0.3	8	55.33%
150	256	0.2	0.3	8	55.65%
128	128	0.2	0.3	8	55.33%
128	64	0.2	0.3	8	56.56%
96	64	0.2	0.3	8	56.36%
160	100	0.2	0.3	8	56.24%

Tabla 70: resultados obtenidos para variación de neuronas y filtros LSTM

A continuación, en la Tabla 71 se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, número de neuronas igual a 180 y número de filtros LSTM igual a 256. En la Tabla 71 se representan las métricas previamente mencionadas y que se busca optimizar, a su vez se observan las precisiones de partida para cada emoción. Se observa que las precisiones para las emociones de enfado, alegría y tristeza son las mayores, con un 43%, 53% y 64% respetivamente, esto se puede deber a que son también las emociones que más ejemplos tienen. Mientras que asco, miedo y sorpresa tienen unas precisiones de 20%, 17% y 10% respectivamente, siendo también las 3 emociones de las que menos ejemplos se tienen.

Polaridad	Precision	Recall	F1-score	Support
Enfado	43%	32%	37%	276
Asco	20%	14%	16%	37
Miedo	17%	3%	5%	37
Alegría	53%	43%	48%	552
Indeterminado	64%	73%	68%	1237
Tristeza	62%	65%	63%	277
Sorpresa	10%	13%	11%	107
Global (weighted avg)	55%	57%	56%	2523

Tabla 71: valores de métricas para optimización de neuronas y filtros LSTM

Ajuste de las tasas de dropout

En la Tabla 72 se aprecia como para el caso de las tasas de dropout el modelo presenta un nivel de ajuste bastante bueno, no existiendo una gran diferencia de precisión en los resultados obtenidos.

Número de neuronas en capa convolucional	Filtros para capa LSTM	Dropout para capa convolucional	Dropout para capa LSTM	Tamaño del Kernel	Accuracy Validación cruzada
180	256	0.2	0.3	8	56.64%
180	256	0.3	0.3	8	55.89%
180	256	0.4	0.3	8	56.84%
180	256	0.5	0.3	8	55.17%
180	256	0.6	0.3	8	55.69%
180	256	0.7	0.3	8	56.20%
180	256	0.8	0.3	8	55.57%
180	256	0.2	0.4	8	55.85%
180	256	0.2	0.5	8	54.54%
180	256	0.2	0.6	8	56.40%
180	256	0.2	0.7	8	56.40%
180	256	0.2	0.8	8	55.33%
180	256	0.2	0.3	8	56.64%

Tabla 72: resultados obtenidos para variación de tasas de dropout

En la Tabla 73 se muestran los valores de las métricas para la configuración con una mayor tasa de precisión, esto es, Dropout de la capa convolucional igual a 0.4 y Dropout de la capa LSTM igual a 0.3. En la Tabla 73 se observa que las precisiones para algunas emociones aumentan, siendo miedo la que más con un aumento de un 14%, enfado con un 3% y sorpresa con un 2%; mientras que el resto han sufrido bajadas en la precisión, aunque ninguna de ellas tan significativa como el aumento de miedo, con lo que se consigue un aumento en la precisión global.

Polaridad	Precision	Recall	F1-score	Support
Enfado	46%	37%	41%	276
Asco	15%	11%	12%	37
Miedo	31%	11%	16%	37
Alegría	48%	43%	46%	552
Indeterminado	64%	71%	68%	1237
Tristeza	61%	69%	65%	277
Sorpresa	14%	11%	13%	107
Global (weighted avg)	55%	57%	56%	2523

Tabla 73: valores de métricas para optimización de tasas de dropout

Ajuste de tasa de aprendizaje del optimizador Adam

Tal y como se observa en la Tabla 74 para la tasa de aprendizaje se observa que, como para el caso de dos polaridades, no se produce una variación considerable en la métrica de precisión del modelo al modificar el valor de dicho parámetro. Como resultado del estudio de la tasa de aprendizaje se observar que, dependiendo del valor que se le asigne, se produce una diferencia de aproximadamente un 2% en los resultados, lo cual refuerza nuestra afirmación de que los parámetros estaban bastante bien ajustados.

Tasa de aprendizaje	**Accuracy Validación cruzada**
0.001	56.84%
0.002	54.46%
0.006	56.72%
0.008	55.53%
0.01	55.13%
0.014	53.47%

Tabla 74: resultados obtenidos para variación de tasa de aprendizaje

En la Tabla 75 se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, una tasa de aprendizaje igual a 0.01. En la Tabla 75 observamos las métricas previamente mencionadas, se observa que las 7 emociones mantienen los niveles de precisión previos a la variación de la tasa de aprendizaje, lo cual nos indica que ya estaba optimizada.

Polaridad	Precision	Recall	F1-score	Support
Enfado	46%	37%	41%	276
Asco	15%	11%	12%	37
Miedo	31%	11%	16%	37
Alegría	48%	43%	46%	552
Indeterminado	64%	71%	68%	1237
Tristeza	61%	69%	65%	277
Sorpresa	14%	11%	13%	107
Global (weighted avg)	55%	57%	56%	2523

Tabla 75: valores de métricas para optimización de tasa de aprendizaje

Reducción del número total de palabras únicas del corpus

Como se mostraba en el estudio de polaridades, al disminuir el número de palabras se obtienen mejores resultados, teniendo cuidado de no descartar palabras que produzcan una pérdida de información relevante para el modelo, Tabla 76. Así, en la Tabla 77 se muestran los valores de las métricas obtenidos para la configuración con una mayor tasa de precisión, esto es, tamaño del vocabulario igual a 6233 palabras.

Tamaño vocabulario	**Accuracy Validación cruzada**
Todas (7033)	56.84%
6833	57.47%
6633	57.55%
6433	57.71%
6233	58.66%
6033	56.28%

Tabla 76: resultados obtenidos para variación de tamaño del vocabulario

En la Tabla 77 se muestra que las precisiones para algunas emociones aumentan, asco tiene un aumento de un 6%, alegría de un 4% y tristeza de un 6% y sorpresa de un 2%; mientras que el resto han sufrido bajadas en la precisión, aunque ninguna de ellas tan significativa como el aumento de las anteriores, con lo que se consigue un aumento en la precisión global.

Polaridad	Precisión	Recall	F1-score	Support
Enfado	43%	41%	42%	276
Asco	21%	14%	16%	37
Miedo	25%	3%	5%	37
Alegría	52%	45%	48%	552
Indeterminado	66%	74%	70%	1237
Tristeza	67%	64%	65%	277
Sorpresa	16%	12%	14%	107
Global (weighted avg)	57%	59%	57%	2523

Tabla 77: valores de métricas para optimización de tamaño del vocabulario

Ajuste del tamaño del batch size

En la Tabla 78 se muestran los resultados obtenidos para el proceso de optimización del batch_size, donde el valor de 32 es donde se alcanzan los mejores valores.

batch size	Accuracy Validación cruzada
32	60.76%
64	58.38%
128	58.22%
256	58.66%
512	55.73%

Tabla 78: resultados obtenidos para variación del batch_size

Resumen de hiperparámetros del modelo final y resultados

En la Figura 37 se muestra el resultado final del modelo una vez se han realizado todos los ajustes de hiperparámetros que se han ido mencionando paso a paso. Se aprecia la concatenación de los subconjuntos train y test, la configuración del bucle, los valores finales para el número de neuronas de la capa convolucional, en este caso 180, el número de filtros LSTM, en este caso 256, las tasas de dropout que son 0.4 y 0.3 respectivamente, el optimizador Adam con un learning rate de 0.001 y el valor del batch_size de 32. La configuración de parámetros que ha permitido optimizar el modelo se representa en la Tabla 56 a modo de resumen, con dichos parámetros y valores alcanzados.

```
from keras.models import Sequential
from sklearn.utils.multiclass import type_of_target
from tensorflow.keras.callbacks import EarlyStopping
import numpy as np
from keras.layers import Embedding, Flatten, Dense, LSTM, Conv1D, GlobalMaxPooling1D, MaxPooling1D, Bidirectional, GRU
from sklearn.model_selection import KFold
import statistics
from tensorflow.keras.optimizers import Adam

X = np.concatenate((X_train, X_test), axis=0)
y = np.concatenate((y_train, y_test), axis=0)
filters = 180
units = 256
#y=np.argmax(y,axis=1)
print(type_of_target(y))
acc_per_fold=[]
loss_per_fold=[]
kf=KFold(n_splits=5, shuffle=True, random_state=999) #ACUERDATE DE CAMBIARLO LUEGO A 10 EEEEEEEEH
#ACUERDATE
cvscores=[]
for train, test in kf.split(X_train,y_train):
  model = Sequential()
  embedding_layer = Embedding(vocab_size, 200, input_length=maxlen)
  #embedding_layer = Embedding(vocab_size, 300, input_length=maxlen, weights=[embedding_matrix],trainable=False)

  model.add(embedding_layer)

  model.add(Conv1D(filters, 8, activation='relu'))

  model.add(MaxPooling1D(10))

  model.add(LSTM(units, dropout=0.4, recurrent_dropout=0.3))



  #Si queremos 7 clases usaremos 7 neuronas en la última capa densa
  model.add(Dense(7, activation='softmax')), #hay que poner la , para que entienda que no son la misma linea esta y la de abajo

  model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

  model.summary()


  early_stop = EarlyStopping(monitor = 'accuracy', mode = 'max', verbose = 1, patience = 3) #cambio un poco el patience para que se e
  model.fit(X[train], y[train], epochs=100, batch_size=32, verbose=1,validation_data = (X[test], y[test]), callbacks=[early_stop])

  scores = model.evaluate(X[test], y[test], verbose=1)
  print(f'Score for fold : {model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')
  acc_per_fold.append(scores[1] * 100)
  loss_per_fold.append(scores[0])
print(acc_per_fold)
print(statistics.mean(acc_per_fold))
```

Figura 37:modelo tras optimización de 7 emociones

Parámetros	Valores
Número Neuronas	180
Número filtros	256
Tasa dropout	0.4
Tasa recurrent_dropout	0.3
Tasa de aprendizaje	0.001
Vocab_size	6233
Batch_size	32

Tabla 79: parámetros optimizados

En la Tabla 80 se muestran los valores para todas las métricas, donde se observan fuertes mejoras tras la optimización del modelo inicial, y donde algunas sufren un aumento en la precisión muy elevado, tales como enfado de un 7%, miedo de un 32% y alegría de un 13%, mientras que el resto de clases bajan su precisión, aunque en menor medida, por lo que se consigue un aumento en la precisión global. Se observa también que las dos emociones que mejor reconoce son alegría y tristeza, las cuales son también las dos clases que menos muestras tienen excluyendo a la clase indeterminado.

Polaridad	Precision	Recall	F1-score	Support
Enfado	50%	25%	34%	276
Asco	11%	5%	7%	37
Miedo	57%	11%	18%	37
Alegría	65%	34%	45%	552
Indeterminado	62%	88%	72%	1237
Tristeza	66%	63%	64%	277
Sorpresa	11%	4%	6%	107
Global (weighted avg)	59%	61%	57%	2523

Tabla 80: valores de métricas para configuración optima 7 emociones

Por otro lado, en la matriz de confusión Figura 38, se puede destacar el hecho de que se predicen muchos mensajes como emoción de indeterminado, aunque no pertenezcan a la misma, esto se aprecia en la quinta columna de la matriz de confusión, suceso el cual puede deberse al desbalance entre el tamaño de esta clase y el resto.

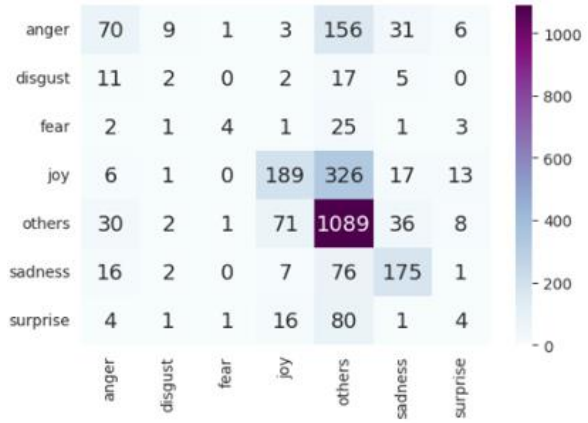


Figura 38:matriz de confusión para configuración óptima

5.3 Conclusiones

Como primera conclusión se puede decir, una vez realizadas todas las pruebas para cada uno de los casos, que el hiperparámetro que mayor repercusión generó al ser modificado fue el tamaño del vocabulario, al descartar palabras con un bajo IG las precisiones aumentaban en gran medida.

El estudio para medir la respuesta emocional respecto a la polaridad ha dado como resultado para Twitch una precisión equiparable tanto para positivo y negativo, mientras que para salud mental la polaridad que mejor reconoce es la positiva. Estos resultados se pueden explicar teniendo en consideración que el corpus de Twitch está formado en su mayor parte por mensajes de polaridades positiva y negativa, a su vez, el corpus de salud mental está conformado en su mayoría por mensajes de polaridad positiva. Quedando con esto demostrada la importancia de contar con una base de datos extensa, pues las clases mayoritarias se han clasificado mejor.

A nivel global, Twitch obtuvo un 81% para el caso de dos polaridades y un 73% para el de tres polaridades. En salud mental los resultados fueron de un 90% para dos polaridades y de un 80% en el caso de tres polaridades. En vista de estos resultados se puede concluir un mayor grado de adaptación del modelo utilizado al trabajar con el corpus de salud mental.

Para la clasificación por emociones se han obtenido los siguientes resultados:

- Twitch: las dos emociones cuya clasificación ha dado mejores resultados han sido aprobación e interés, las cuales tienen polaridad positiva, siendo además

la emoción de aprobación aquella de la cual se tienen más muestras. Como segundo aspecto a destacar se observa cómo las emociones de decepción, desaprobación, desinterés y enfado se están confundiendo en gran medida entre ellas, esto puede deberse al reducido número de muestras que tienen.

- Salud mental: en este experimento los resultados obtenidos fueron muy buenos, con emociones como gratitud o enfado superando el 80% y amor con un 77% de precisión. También se observa como la emoción de comprensión a pesar de ser la segunda con más muestras, presenta una precisión del 66%, debido a que algunos mensajes que expresan otras emociones se están clasificando como comprensión de manera errónea, quedando en claro que si bien el número de muestras que se tengan de una clase es un factor decisivo no es el único que afecta.
- Twitter: el experimento con el corpus de Twitter si bien era el que tenía más ejemplos, al estudiarlo más a fondo se ve que la mayoría pertenecen a la emoción de indeterminado, la cual no aporta información. Las emociones que mejor clasifica son alegría y tristeza. Alegría presenta un nivel alto de precisión porque, aunque a la hora de clasificar un mensaje como alegría, esté seleccionando solo un 34% de los que en realidad lo son (Tabla 80) tampoco está seleccionando muchos mensajes que no sean de alegría y clasificándolos como tal. Por otro lado, tristeza tiene una precisión alta debido a que selecciona un 63% de los mensajes que sí son de tristeza y los clasifica como tal, haciendo así que, aunque luego clasifique erróneamente como tristeza varios mensajes la precisión sigue siendo elevada. En este estudio se puede observar como una precisión elevada se puede obtener de varios escenarios distintos.

Una vez se estudian los resultados obtenidos, es posible concluir que el modelo desarrollado tiene un mayor nivel de adaptación para la red social de Twitter, teniendo una precisión global de un 59%, mientras que para Twitch solo se alcanzó un 48%, es decir, un 11% menos de precisión, el cual se puede deber a factores externos como un correcto etiquetado del corpus, o bien, a factores internos del modelo como puede ser un menor grado de adaptación al tratar la información obtenida.

6

Conclusiones y Líneas futuras

6.1 Conclusiones

Este Trabajo de Fin de Grado se ha centrado en la investigación y desarrollo de un modelo de clasificación de mensajes utilizando técnicas de procesado de lenguaje natural y aprendizaje automático. Para ello se ha utilizado un modelo de Deep Learning basado en redes neuronales, combinación de diferentes tipos de capas, basado en redes neuronales convolucionales y memorias a corto y largo plazo con el fin de capturar las características más relevantes y poder realizar una correcta clasificación.

Los resultados que se han obtenido se pueden analizar por partes. En primer lugar, respecto a los estudios de polaridades, el modelo demostró un alto rendimiento, llegando a obtener unos resultados de 80.45% y 73.63% para 2 y 3 polaridades respectivamente en Twitch, valores los cuales ya son elevados de por sí. Pero incluso estos valores son mejorados para el estudio de salud mental, donde alcanzan niveles de 90.22% para 2 polaridades y 81.34% para 3 polaridades. Observando estos resultados es posible afirmar que la idea de utilizar un enfoque basado en embeddings junto con arquitecturas de redes neuronales resulta efectivo para la resolución del problema planteado.

Acto seguido, al analizar los resultados para clasificación de emociones observamos que los resultados no mantienen una uniformidad de valores elevados como era el caso para polaridades. En concreto, para el corpus de videojuegos en Twitch el modelo presenta una precisión del 50.75%, en Twitter un rendimiento de 60.76% y por último para el corpus de salud mental consigue unos valores más elevados del 72.49%. Al estudiar los resultados, y teniendo en consideración que el modelo base es el mismo en todos los estudios, se concluye que, la precisión con la cual se realicen las clasificaciones va a depender de los corpus utilizados, los cuales necesitan cumplir dos objetivos fundamentales: un equilibrio entre el número de muestras de cada clase y una correcta

clasificación de las clases en las categorías estipuladas. También es destacable el hecho de que la clase indeterminada, como se muestra en el estudio de Twitter es una clase que no aporta información y, por lo tanto, que sea aquella con el mayor número de muestras no ayuda a una correcta clasificación. Dicho esto, para los 3 dataset se observa un patrón: siempre hay al menos una emoción cuya polaridad sea positiva entre aquellas que mejor se clasifican. Esto puede deberse a otros factores, tales como que hay un mayor número de muestras para las emociones positivas y el hecho de que las emociones positivas se reconocen mejor que las negativas, ya que estas últimas tienen un mayor nivel de similitud entre ellas. Por último. es importante destacar el aumento en la complejidad que supone realizar un estudio a nivel de emociones en comparación con las polaridades, lo cual también justifica en cierta medida la reducción en niveles precisión.

6.2 Líneas futuras

Si bien los resultados obtenidos, sobre todo para polaridades, han prometedores, hay aspectos que se pueden mejorar para futuras investigaciones. Algunas de las líneas futuras a seguir serían las siguientes:

- Técnicas de transferencia de aprendizaje: esto es, utilizar modelos pre-entrenados los cuales poseen conocimientos adquiridos al realizar estudios relacionados con este, en los cuales san capturado patrones, características o conocimientos sobre lingüística que pueden ser de utilidad a la hora de clasificar textos. Con estos modelos se podrían obtener beneficios como un ahorro en el tiempo y los recursos necesarios para realizar el estudio, una mejora en el rendimiento o tener un mayor grado de generalización.
- Realizar una búsqueda más exhaustiva de los hiperparámetros: si bien se ha realizado una búsqueda inicial de los hiperparámetros, hay otras técnicas que se podría utilizar, como podría ser una optimización bayesiana, la cual sería de utilidad a la hora de encontrar una combinación óptima de hiperparámetros.
- Ampliación de los corpus: se ha explicado a lo largo del trabajo la importancia que tienen los datos a tratar en el estudio. Sería interesante por lo tanto conseguir una ampliación de los mismos, es decir tener un mayor número de datos de entrenamiento supondría un mejor aprendizaje del modelo.
- Mejora en el preprocesamiento de los datos: es una etapa fundamental a la hora de realizar estudios sobre procesamiento de lenguaje natural, en este proyecto se

tomaron unos por defecto. Sería interesante buscar mejores combinaciones de los mismos, o maneras de modificarlos para que estén más enfocados al tipo de textos que se está buscando clasificar, consiguiendo así eliminar el ruido y una mejora en la calidad de los datos.

- Nuevas líneas de categorización: realizar otro tipo de categorizaciones e integrarlas en los corpus, teniendo en cuenta por ejemplo el impacto de los discursos de odio o los estigmas en redes sociales, que pueden ser clasificaciones interesantes de realizar en estos entornos virtuales.

Con este trabajo se han sentado las bases para realizar futuras investigaciones sobre la clasificación de textos utilizando procesado de lenguaje natural, por lo que, las líneas futuras de investigación en este caso son bastante claras, se buscará seguir desarrollando el modelo para conseguir un aumento en los niveles de precisión que presenta.

7

Bibliografía

[1] TFG Jesús Herrero Llanos. URL: https://uvadoc.uva.es/handle/10324/57316

[2] TFM José Carlos Sobrino Sande. URL: https://github.com/jcsobrino/TFM-Analisis_sentimientos_Twitter-UOC.

[3] TFM de Julia Isabel Medrano Sanz. URL: https://oa.upm.es/65589/1/TFM%5C_Carlos_Hernandez%5C_P.pdf.

[4] TFM de Ivan Arévalo Nuñez. URL: https://oa.upm.es/65589/1/TFM%5C_Carlos_Hernandez%5C_P.pdf.

[5] Russell, S. J., & Norvig, P. (2016). Artificial Intelligence: A Modern Approach. Pearson.

[6] Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press.

[7] Machine Learning vs Deep Learning. URL: https://www.xataka.com/robotica-e-ia/machine-learning-y-deep-learning-como-entender-las-claves-del-presente-y-futuro-de-la-inteligencia-artificial.

[8] Carlos Hernández. Diferencias entre Deep Learning y Machine Learning. URL: https://levity.ai/blog/difference-machine-learning-deep-learning.

[9] Autor: Goodfellow, I., Bengio, Y., & Courville, A. Título: Deep Learning Año: 2016 Editorial: MIT Press

[10] Historia del Deep Learning. URL: https://machinelearningknowledge.ai/brief-history-of-deep-learning/%5C#Deep%5C_Learning%5C_History%5C_Timeline.

[11] López-Molina, C., Luna-Ramírez, E., Cerrada, J.A., & López, V. (2020). Aprendizaje profundo para clasificación y reconocimiento. Universidad Politécnica de Madrid, España. Disponible en: http://oa.upm.es/63436/1/INVE_MEM_2020_307753.pdf.

[12] Batch Size. URL: https://healthdataminer.com/data-mining/aprendizaje-supervisado-y-no-supervisado/#:~:text=El%5C%20aprendizaje%5C%20supervisado%5C%20supone%5C%20que,de%5C%20datos%5C%20no%5C%20etiquetados%5C%20previamente.

[13] Anatomía de una red neuronal. URL: https://vincentblog.xyz/posts/conceptos-basicos-sobre-redes-neuronales.

[14] Funcionamiento neurona. URL: https://oa.upm.es/51557/1/PFC_LUIS_MIGUEL_NUNEZ_VIVERO_2018.pdf.

[15] Funciones de activación. URL: https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/.

[16] Proceso de Entrenamiento red neuronal. URL: https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i.

[17] TFG Carlos Hernández. URL: https://oa.upm.es/65589/1/TFM%5C_Carlos_Hernandez%5C_P.pdf.

[18] Clasificación de redes neuronales. URL: https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales.

[19] François Chollet. Deep Learning with Python. URL: https://tanthiamhuat.files.wordpress.com/2018/03/deeplearningwithpython.pdf.

[20] Web oficial de Python. URL: https://www.python.org/.

[21] Web oficial de Tensorflow. URL: https://www.tensorflow.org/?hl=es-419.

[22] Web oficial de Keras. URL: https://keras.io/.

[23] Web oficial de NLTK. URL: https://www.nltk.org/.

[24] Web oficial de Google Colab. URL: https://colab.research.google.com/.

[25] Web oficial de Twitter. URL: https://twitter.com/.

[26] Web oficial de Twitch. URL: https://twitch.com/.

[27] Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press.

[28] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.