Universidad de Valladolid

Escuela de Ingeniería Informática

Trabajo Fin De Grado

Grado en Ingeniería Informática
(Mención Tecnologías de la Información)

Análisis de sentimientos en Twitter mediante técnicas de Deep Learning

Autor:
D. Jesús Herrero Llanos
Tutores:
Dr. D. Jesús M. Vegas Hernández
Dra. Dña. Noemí Merayo Álvarez

Resumen

El crecimiento de las redes sociales en los últimos años hace que dichos medios abarquen una gran cantidad de información de la que podemos extraer para diversos fines. Por lo tanto, resulta de gran interés y motivación el trabajo presentado.

En este TFG se pretende hacer un estudio acerca del análisis de sentimientos en Twitter mediante la aplicación de técnicas Deep Learning. El objetivo que se presenta en dicho trabajo es poder conocer el mundo del Deep Learning, y poder crear un clasificador de tweets en español en función del sentimiento que transmite con una precisión de al menos un 75 %.

Para ello, en una primera fase, se ha llevado a cabo la creación de un clasificador utilizando redes neuronales en el que se ha comparado el funcionamiento de diversas combinaciones propias del Deep Learning. Se ha partido de un corpus de aprendizaje y se ha realizado un análisis exploratorio de las diversas opciones que se han podido investigar para poder probar cual de ellas ofrecía mejores resultados.

En una segunda fase, se ha optimizado el mejor modelo obtenido en la comparativa mencionada en el párrafo anterior, para así intentar mejorar el porcentaje de acierto del clasificador. Como resultado se ha obtenido un clasificador con una precisión superior al 75 %.

Palabras clave

IA (Inteligencia Artificial), AP (Aprendizaje Profundo), AU (Aprendizaje Automático), TFG (Trabajo de Fin de Grado) , PNL (Procesamiento del Lenguaje Natural), CPU(Unidad Central de Procesamiento), GPU (Unidad de Procesamiento Gráfico), TASS (Taller de Análisis de Sentimientos).

Abstract

The growth of social networks in recent years means that these media cover a large amount of information from which we can extract for various purposes. Therefore, the presented work is of great interest and motivation.

In this TFG is intended to make a study about sentiment analysis in Twitter by applying Deep Learning techniques. The objective presented in this work is to know the world of Deep Learning, and to be able to create a classifier of tweets in Spanish according to the sentiment transmitted with an accuracy of at least 75 %.

For this, in a first phase, we have carried out the creation of a classifier using neural networks in which we have compared the performance of various combinations of Deep Learning. A learning corpus was used as a starting point and an exploratory analysis of the various options that could be investigated was carried out in order to test which of them offered the best results.

In a second phase, the best model obtained in the comparison mentioned in the previous paragraph was optimized in order to try to improve the classifier's success rate. As a result, a classifier with an accuracy of over 75 % was obtained.

KeyWords

AI (Artificial Intelligence), DL (Deep Learning), ML (Machine Learning), FDP (Final Degree Project) , NLP (Natural Language Processing), CPU(Central Processing Unit), GPU (Graphics Processing Unit), TASS (Sentiment Analysis Workshop).

Agradecimientos

Lo primero de todo, agradecer a mis dos tutores por la ayuda y el seguimiento realizado para poder llevar a cabo dicho trabajo, así como la posibilidad de realizar este TFG. También me gustaría agradecer a mis amigos de la carrera que siempre me han ayudado cuando lo he necesitado.

Después me gustaría agradecer a mi familia por el apoyo que me han mostrado durante los años que he estado en la carrera.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Motivación

Este trabajo ha sido de gran motivación por mi parte debido al profundo desconocimiento que tenía acerca de las principales aplicaciones de la Inteligencia artificial así como del aprendizaje automático, sumado a la fascinación por el mundo de las redes sociales.

Hoy en día la informática tiene un peso fundamental en cualquier aspecto que nos planteemos, pues todo tiene algo de informática de por medio. La pandemia COVID-19 ha hecho que estemos más obligados a estar conectados en red, por lo que resulta de gran interés poder saber que piensan o que sienten las personas ante determinadas situaciones a través de mensajes que residen en Internet.

En este trabajo se va a intentar realizar una investigación sobre el análisis de sentimientos en Twitter mediante redes neuronales y poder obtener un algoritmo lo más preciso posible.

1.2. Objetivos

1.2.1. Objetivo principal

El principal objetivo de este trabajo es la aplicación de técnicas Deep Learning en un problema de clasificación de tweets por medio del sentimiento que trasmite.

Se parte de un algoritmo en el que se pretende poder aplicar técnicas Deep Learning para mejorar sus resultados.

Nuestro objetivo final será la obtención de un sistema que permita predecir con la mayor precisión posible el sentimiento que trasmite un determinado tweet.

1.2.2. Objetivos específicos

Entre los objetivos más específicos podemos destacar los siguientes:

- Estudio del arte referente al mundo del Deep Learning y las redes neuronales.
- Análisis de diferentes técnicas de Deep Learning empleadas en problemas de clasificación de textos.
- Estudio comparativo de diferentes arquitecturas de redes neuronales que aplican técnicas de Deep Learning.
- Explicación de plataformas y sistemas donde poder desarrollar tus proyectos de Deep Learning.
- Obtención de un modelo que supere el 75 % de exactitud utilizando tres clases.

1.3. Metodología del trabajo

La metodología empleada durante el desarrollo del trabajo ha sido la siguiente:

1.3.1. Fase de Documentación

Esta fase ha sido crucial a la hora de hacer frente a un problema que hasta este momento desconocía. He podido hacer familiar conceptos como aprendizaje automático, Deep learning, red neuronal etc. Básicamente tener claros los conceptos teóricos en los que se basa esta rama de la IA.

1.3.2. Fase de análisis

En esta fase se ha analizado con la documentación pertinente los distintos algoritmos ya creados, así como un estudio de los diferentes hiperparámetros que confeccionan una red neuronal y las diferentes alternativas de redes neuronales que se pueden programar. Se ha partido de los siguientes trabajos:

- TFM de Jose Carlos Sobrino.[1]
- TFM de Julia Isabel Medrano Sanz[2].
- TFM de Ivan Arévalo Nuñez[3].

1.3.3. Fase de pruebas

En esta fase se han realizado las diferentes pruebas pertinentes de acuerdo al análisis previo realizado en la fase anterior.

1.3.4. Fase de escritura del informe

En esta fase se ha realizado la redacción del informe presente, la reflexión y explicación de las conclusiones con las que se ha llegado a este trabajo.

1.4. Estructura de la memoria

La memoria de este trabajo consta de diferentes capítulos agrupados por el objetivo que tienen de cara al lector.

En el primero de ellos (el capítulo actual) se pretende dar una introducción de la metodología seguida, así como los motivos por los que ha tenido lugar este trabajo.

En el capítulo 2 se ofrece la planificación llevada a cabo durante el proyecto, la explicación de los riesgos potenciales del proyecto, y una simulación de costes y recursos en el caso de que fuera un proyecto de encargo.

En el capítulo 3 se pretende ofrecer un marco teórico para que el lector sea capaz de ubicarse en el tema que abarca dicho TFG. Se expondrán los conceptos más importantes así como las herramientas utilizadas para el desarrollo del mismo.

En el capítulo 4 entramos en un ambiente más práctico, sin dejar de lado las explicaciones teóricas que ilustran los conceptos que se han implementado. Se expondrán dos fases: la primera para poder elegir el mejor modelo de los planteados, y la segunda para poder mejorar los resultados ofrecidos por el mejor modelo de la fase anterior.

En el capítulo 5 se expondrán las aplicaciones, dificultades, conclusiones obtenidas y las posibles líneas futuras para seguir desarrollando el tema.

En el capítulo 6 se incluyen dos secciones muy útiles en el caso de que se quiera aprender a como crear modelos de Deep Learning, mostrando los requisitos de computación necesarios y las opciones de entorno disponibles para la creación de los algoritmos.

Capítulo 2

Planificación

2.1. Planificación

Con el fin de establecer una secuenciación de los diferentes hitos acontecidos en el proyecto se establece un plan de proyecto para establecer difentes etapas: Se ha establecido una metodología de trabajo en el capítulo anterior, pero vamos a ser un poco más precisos indicando los plazos de dichas tareas.

Se ha establecido un comienzo del proyecto el día 10/01/2022 y una fecha prevista de finalización del día 10/06/2022.

En la figura 2.1, se puede observar el diagrama de Gantt creado con la herramienta Project de la planificación del proyecto, incluyendo las diversas y principales actividades de las que se compone, así como una fecha de inicio y fin para cada una de ellas.

Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
Definir contenidos de trabajo	2 horas	lun 10/01/22	lun 10/01/22		
Fase de Documentación	**30 días**	**lun 10/01/22**	**lun 21/02/22**	**1**	
Documentación Previa	30 días	lun 10/01/22	lun 21/02/22		
Fase de Análisis	**30 días**	**lun 21/02/22**	**lun 04/04/22**		
Cálculo de estrategias	15 días	lun 21/02/22	lun 14/03/22	3	
Estudio de estrategias	15 días	lun 14/03/22	lun 04/04/22	5	
Fase de pruebas	**30 días**	**lun 04/04/22**	**lun 16/05/22**		
Modelo Original	5 días	lun 04/04/22	lun 11/04/22	4	
Modelo Deep Learning	20 días	lun 04/04/22	lun 02/05/22	3;6	
Modelo Avanzado	10 días	lun 02/05/22	lun 16/05/22	9;8	
Fase de escritura de la memoria	**19 días**	**lun 16/05/22**	**vie 10/06/22**	**7**	
Redacción de la memoria	15 días	lun 16/05/22	lun 06/06/22	3;10	
Reflexión y Conclusiones	4 días	lun 06/06/22	vie 10/06/22	12	

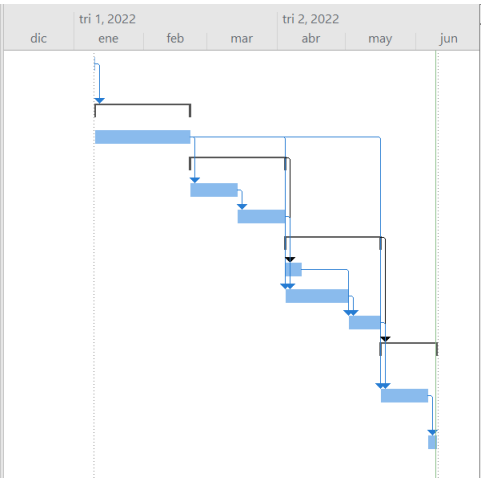

Figura 2.1: Diagrama de Gantt

2.2. Presupuesto

Como se ha mencionado anteriormente, se ha supuesto en la planificación que la duración de dicho trabajo va a ser de 5 meses.

El principal coste del proyecto es el de contratar a un Ingeniero de Datos, ya que tienen grandes conocimientos en el área de la Inteligencia Artificial.

Vamos a suponer que el proyecto tiene lugar durante 300 horas, ya que es el número de horas que vienen establecidas en la duración mínima de un Trabajo de Fin de Grado, y consultando el salario medio de un Ingeniero de Datos, un Ingeniero de Datos promedio cobra entorno a 15,38€/h[4], lo que hace un total de 4614€ de salario al Ingeniero.

Referentes a gastos materiales, se ha comprado un equipo con unas características estándar ya que todas las pruebas se realizarán en servidores en la nube por lo que no se utilizan los recursos de nuestro propio ordenador. Este ordenador ha costado 799€. También hay que sumar la luz que el ordenador pueda llegar a gastar. Se ha estimado que durante las 300 horas se pueda llegar a consumir entorno a 300W y al ser un valor muy fluctuante se ha supuesto el peor de los casos: a 0.60 kw/h, lo que hace un total de unos 50€ aprox.

Por otro lado, los servidores mencionados tienen una serie de recursos limitados, pues si las pruebas realizadas necesitan más potencia de cómputo se podrá adquirir una mejora en CPU y RAM cuyo importe dependerá de la plataforma. Se puede consultar en anexos las distintas opciones que hay en el mercado.

Por lo tanto el coste del proyecto es de 5463 € + posible potencia de cómputo.

2.3. Riesgos

Es muy importante analizar los posibles riesgos que puedan suceder a lo largo del proyecto, ya que pueden retrasar la ejecución del mismo y ser causante de la no satisfacción del resultado obtenido en el mismo proyecto.

Se han planteado los posibles riesgos que se ilustran a continuación:

Referencia	Riesgo	Probabilidad	Impacto	Gestión del Riesgo
R1	Falta de conocimientos	Alta	Alto	Correcta fase de documentación previa.
R2	Escasez de recursos informáticos	Alta	Medio	Planificación de requisitos de computación y previo estudio de viabilidad del proyecto.
R3	Estimaciones de costes y recursos irreales	Media	Medio	Correcta planificación del proyecto
R4	Gold Plating	Media	Bajo	Definir claramente unos objetivos y una terminación de proyecto
R5	Enfermedad	Baja	Alto	Asumir los días de baja

Figura 2.2: Riesgos del proyecto

En la figura podemos observar en la primera columna, la referencia al riesgo y en la segunda un título que sintetize el riesgo.

Para medir la exposición a un determinado riesgo se sigue la siguiente fórmula:

$$ExposiciónRiesgo = ProbabilidadSuceda * Impacto$$

En este caso se ha decidido usar una notación cualitativa teniendo en cuenta las siguientes posibilidades: alto, medio y bajo.

En la última columna tenemos la gestión del riesgo, pues tenemos distintas opciones, podemos aceptar el riesgo, evitar el riesgo, reducir el riesgo, mitigar el riesgo o transferir el riesgo.

Se ha decidido utilizar en gran parte la posibilidad de evitar el riesgo a excepción del riesgo R5 que se acepta ya que es algo que no se puede preever de manera sencilla.

Algunas de las decisiones que se han tomado respecto a los riesgos son las siguientes:

- Se ha creado un plan de contingencia en caso de que se necesiten más recursos de hardware para poder ejecutar los programas. Un plan de contingencia sirve para intentar frenar un riesgo que ya se ha materializado.

Capítulo 3

Estado del arte

3.1. Inteligencia Artificial

La inteligencia artificial se define como[5]: *"Disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico."*

Siempre ha existido un debate que cuestiona si la inteligencia de un ser humano es comparable y alcanzable mediante la inteligencia que puede llegar a tener un ordenador.

Dentro de la inteligencia artificial, existe una rama llamada denominada Machine Learning[6] que consiste en utilizar algoritmos para parsear datos, aprender de dichos datos y posteriormente poder predecir algo.

Durante estos últimos años el Machine Learning ha evolucionado a lo que conocemos hoy en día como Deep Learning. El Deep Learning se entiende como un subconjunto del Machine Learning.

A diferencia del Machine Learning, en el Deep Learning no se requiere de supervisión humana y se intenta simular el pensamiento humano de una manera más exacta a través de las neuronas de las redes neuronales. Se configuran parámetros básicos acerca de los datos y se entrena a la máquina para que aprenda por si misma reconociendo patrones mediante el uso de diversas capas de procesamiento. El Deep Learning se puede implementar de diferentes maneras, pero la más común es mediante la creación de modelos de redes neuronales artificiales que veremos a continuación.

Como se ve en la figura 3.1, la Inteligencia Artificial abarca todo, es decir cualquier algoritmo que lleve a cabo tareas que requieran de IA.

El Machine Learning se encarga de crear algoritmos que aprenden de los datos y pueden tomar decisiones en función de los patrones observados. Algunos algoritmos de Machine Learning

son Naive Bayes, máquina de vectores de soporte, K vecinos más cercanos y los árboles de decisión.

El Deep Learning utiliza modelos de redes neuronales para resolver problemas sin la intervención humana. Por lo que podemos decir que el Deep Learning es una subparte de las redes neuronales.

Se podría resumir todo en la siguiente oración: El Deep Learning es un subconjunto especializado del Machine Learning, que a su vez es un subconjunto de la Inteligencia artificial, por lo que el Deep Learning es Machine Learning[7].

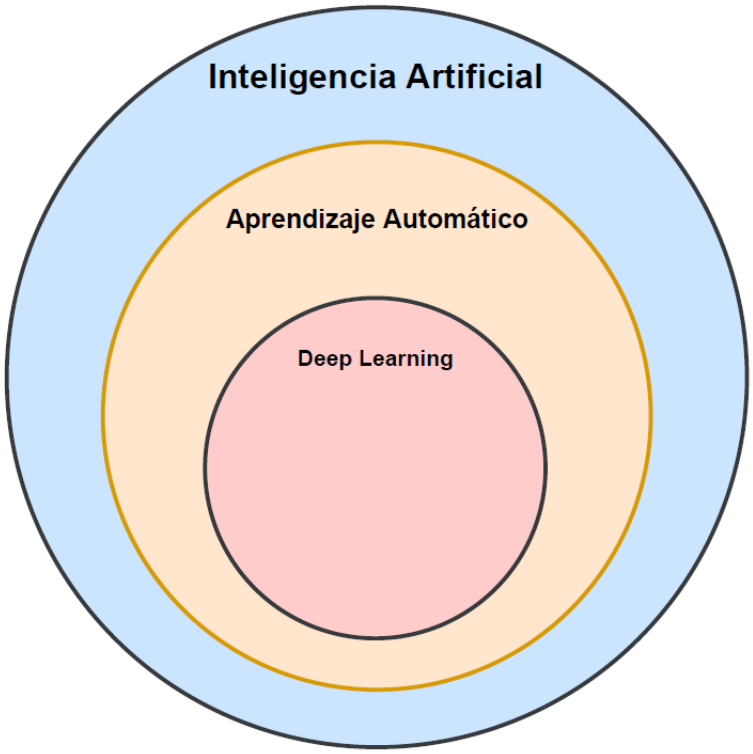

Figura 3.1: Categorias

3.2. Deep learning vs Machine learning

Como ya se ha comentado, la principal diferencia es el hecho de que el Deep Learning es un subconjunto del Machine Learning y no requiere de intervención humana.

Los modelos Deep Learning requieren una gran cantidad de datos de partida, ya que necesitan tener mucha información que procesar para poder ser útiles. Los algoritmos Deep Learning se deben de entrenar mediante el uso de GPU, a diferencia de los de Machine Learning que se entrenan con la CPU.

En la figura 3.1, podemos ver una diferencia en la creación de algoritmos de Machine Learning

y algoritmos de Deep Learning. Como se ve, el proceso de extracción de características es realizado por un humano y es pasado al algoritmo en cuestión para que devuelva algo. Por otro lado, en el caso del Deep Learning la extracción de características la realiza el propio algoritmo.

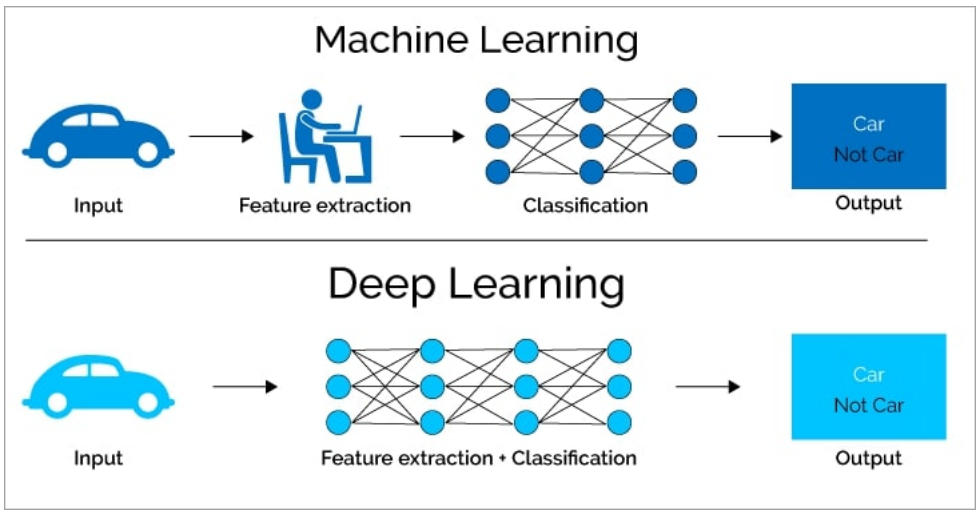

Figura 3.2: Deep Learning vs Machine Learning[7]

3.3. Origen del Deep Learning

Durante estos últimos años ha habido una explosión con temas relacionados con la IA, y una de las causas es debido al Deep Learning. Parece algo muy reciente que se ha desarrollado en los últimos años, pero la historia del Deep Learning comienza en la década de los 40. Algunos de los acontecimientos más importantes han sido los siguientes[8]:

- En 1943 se crea la neurona de McCulloch Pitts capaz de imitar el comportamiento de una neurona biológica aunque no tenía mecanismo de aprendizaje y tenía capacidad limitada.
- En 1957 Frank Rosenblatt creaba el perceptron capaz de realizar clasificación binaria con capacidades de aprendizaje. Este fue un movimiento que inspiró a la revolución de las redes neuronales artificiales en los siguientes años, hasta lo que se conoce como el invierno de las redes neuronales.
- En 1965 Ivakhnenko crea el primer modelo de perceptrón multicapa, considerado como el padre del Deep Learning.
- Durante la década de los 70 supone un período de investigación, en el que se deja de lado el perceptron y se crea el algoritmo de propagación hacia atrás en código de ordenador.
- En 1980 la organización Neocognitron crea la primera arquitectura de red convolucional, la cual era capaz de reconocer patrones visuales como caracteres escritos a mano.

- En 1982 Hopfield crea lo que se conoce como la primera red neuronal recurrente. Era un sistema de memoria direccionable por contenido que serviría como base de futuros modelos de Deep Learning.
- 1986 fue un año clave ya que tuvieron lugar diversos acontecimientos: Sejnowski crea NeTalk, una red capaz de pronunciar palabras escritas en ingles mediante el uso de transcripciones de telefonía. También se implementa el algoritmo de propagación hacia atrás (backpropagation) en redes neuronales, lo que permitió el entrenamiento de redes neuronales muy complejas de manera más sencilla.
- En 1989, LeCunn aplicó el algoritmo de backpropagation a una red convolucional que reconocía digitos escritos.
- En 1997 Hochreiter crea la red neuronal recurrente LSTM (Memoria corta a largo plazo), la cual va a revolucionar los modelos Deep Learning de los siguientes años. Durante los siguientes años se implementaron las redes neuronales recurrentes bidireccionales.
- En 2008 se empezó a utilizar la GPU para entrenar los modelos de redes neuronales ya que según estudios realizados reducían considerablemente el tiempo de entrenamiento.
- En 2011 se crea el grupo de investigación Google Brains que fue el encargado en desarrollar TensorFlow.
- En 2012 Khrizhevsky crea Alexnet, una red convolucional entrenada con GPU, capaz de clasificar imágenes con una precisión del 84 %, la cual mejora hasta casi un 10 % los modelos existentes.
- En 2014 Goodfellow crea las redes neuronales de creencia profunda también conocidas como Generative Adversarial Nets, muy utilizadas en problemas de imágenes.
- En 2016 se produce un hecho insólito, una máquina que implementa Deep Learning es capaz de ganar a un humano en el juego del Go, el cual es mucho más complejo que el ajedrez. Esto da mucho que pensar y se abre un nuevo nivel de aprendizaje de máquinas.
- En 2019 Bengio, Hinton y LeCun ganan el premio Turing por su contribución en investigación en Deep Learning, como dando a entender que el esfuerzo producido tiene un sentido, ya que el Deep Learning se está utilizando mucho en estos tiempos.

3.4. Aprendizaje supervisado vs aprendizaje no supervisado

Los algoritmos se pueden clasificar en dos tipos: supervisados y no supervisados[9]. Los primeros requieren partir de un conjunto de datos etiquetado previamente, es decir, sabemos cual es el dato objetivo para el conjunto de datos, y los segundos son todo lo contrario, no conocemos el

valor del atributo objetivo para el conjunto de datos.

Este conjunto de datos etiquetado previamente se denomina corpus de aprendizaje, que veremos más adelante.

La rama de algoritmos supervisados pertenece a la rama del aprendizaje automático y nos encontramos con los algoritmos de regresión que se basan en producir una predicción en forma de dato numérico, y los algoritmos de clasificación que se basan en producir una predicción en forma de dato categórico. El éxito en utilizar algoritmos supervisados se basa principalmente en dos factores: el algoritmo seleccionado y las características seleccionadas que alimentan al algoritmo y de las que se entrena.

Estos algoritmos se pueden implementar de muchas maneras, ya sea mediante Naive-Bayes, máquina de vectores de soporte, árboles de decisión, k-vecinos más cercanos y redes neuronales que va a ser el algoritmo objeto de estudio de este trabajo.

Por otro lado, la rama de algoritmos no supervisados se basa en no tener ningún tipo de información acerca de los datos, si no que mediante técnicas de agrupamiento, llamadas clustering o segmentación, encontrar grupos que compartan similitudes en el conjunto de datos.

Estos algoritmos de clasifican en algoritmos jerárquicos que son aquellos que posibilitan la creación de distintos niveles de agrupación, y algoritmos no jerárquicos o particionales que siguen otro tipo de métodos de particionamiento.

Algunos ejemplos de este tipo de algoritmos son k-medias, análisis de componentes clave o agrupación jerárquica.

3.5. Redes neuronales

Las redes neuronales son algoritmos que pertenecen a la rama del Machine Learning de la IA a las que podemos aplicar técnicas de Deep Learning. En términos simples el funcionamiento de una red neuronal consiste en tomar una entrada de números, hacer una serie de operaciones y devolver una salida de números.

3.5.1. Anatomía de una red neuronal

Los principales elementos por los que está formado una red neuronal son los siguientes[10]:

- Datos de entrada: Los datos de entrada son los datos que proporcionamos a la red para que pueda aprender de ellos mediante el entrenamiento. Nosotros a la red la tenemos que proporcionar los datos y los resultados que se esperan de esos datos para que nuestra red

pueda hacer predicciones. Esto es un paso muy importante ya que en función de los datos de entrada, nuestra red va a ser mejor o peor.

- Capas: Una capa de una red neuronal es un conjunto de neuronas cuyas entradas vienen de una capa anterior y sus salidas son las entradas de la capa posterior. La primera capa se denomina capa de entrada a la que le proporcionamos ciertos datos elegidos con cierto sentido, y la ultima capa es la capa de salida que nos proporciona los resultados. Las capas intermedias se denominan capas ocultas.

- Función de activación: Se usan para propagar la salida de los nodos de una capa hacia la siguiente capa. Devuelve una salida a partir de un valor de entrada, normalmente el conjunto de valores de salida en un rango determinado como (0,1) o (-1,1).

- Optimizador: Se encarga de optimizar los valores de los parámetros para reducir el error cometido por la red. A este proceso se le denomina propagación hacia atrás. Los más conocidos son adam, rmsprop y sgd.

- Función de pérdidas: Se encarga de evaluar la desviación entre las predicciones realizadas por la red neuronal y los valores reales del conjunto de datos empleado durante el entrenamiento. El objetivo es conseguir disminuir esta desviación ajustando los distintos pesos de la red neuronal. Cuanto menor es el resultado de esta función, más eficiente es la red neuronal.

El funcionamiento de una red es sencillo de entender. Las neuronas que forman una capa recogen las salidas de la capa anterior y pasan a ser sus entradas. Si la capa de entrada es la capa anterior, dichos valores serán los valores de entrada de la capa correspondiente. A estos valores se les aplican unos pesos, conocidos como pesos sinápticos que son característicos de cada conexión de dos neuronas. Todas estas operaciones se suman y se aplica la función de activación de la capa correspondiente. El resultado de dicha operación es la entrada de la siguiente capa o la salida en caso de ser la capa de salida de la red neuronal.

La estructura de una neurona se puede ver en la siguiente figura[11]:

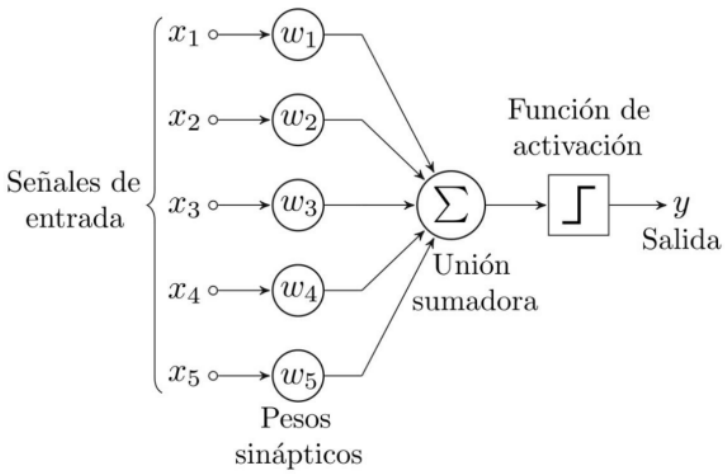

Figura 3.3: Neurona artificial (Perceptrón simple)[11]

La salida y se calcula con la siguiente expresión:

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

,donde FA es la función de activación que tiene la capa en la que se encuentre la neurona, ya que las funciones de activación son características de las capas. En ocasiones se suele sumar un parámetro llamado bias.

El peso de una entrada representa la fuerza de unión entre la neurona que ha proporcionado dicho peso y la neurona actual. La idea es que dichos pesos se vayan ajustando para a partir de las entradas proporcionadas poder obtener las salidas deseadas.

Las funciones de activación son un elemento muy importante durante el proceso, y la selección de la misma puede ser clave para obtener mejores o peores resultados. Algunos de las funciones de activación más utilizadas son las siguientes[12]:

- Función de activación Sigmoid: Transforma los valores en una escala entre (0,1) en la que los valores altos tienden a 1 y los valores pequeños tienden a 0.

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

- Función de activación Tanh: Transforma los valores en una escala entre (-1,1) en la que los valores altos tienden a 1 y los valores pequeños tienden a -1.

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

- Función de activación ReLU: Transforma los valores negativos en 0 y los positivos los deja igual.

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

- Función de activación Softmax: Transforma las salidas a una representación en forma de probabilidades, de tal manera que el sumatorio de todas las probabilidades de las salidas de 1. Se utiliza en la capa de salida normalmente en problemas de clasificación multiclase.

3.5.2. Entrenamiento de una red neuronal

El proceso de entrenamiento de una red neuronal[13] consiste en ajustar cada uno de los pesos que caracterizan las conexiones entre neuronas de las capas de una red neuronal, con el objetivo de obtener las salidas esperadas de acuerdo a los datos que sabemos que son ciertos.

Por lo tanto, el objetivo del entrenamiento es obtener los mejores valores de los pesos posibles a través del algoritmo de propagación hacia atrás. El error que se obtiene en la salida con respecto al valor real tiene que ser ajustado utilizando para ello el método de descenso del gradiente, que consiste en moverse en la dirección de máximo decrecimiento, con el objetivo de minimizar el error que el modelo comete.

Como se puede observar en la figura 3.4, la idea es ir reducciendo el error en cada iteracción hasta obtener un valor aceptable por el que la red pueda converger, es decir a partir de un número finito de pasos, obtener un resultado aceptable.

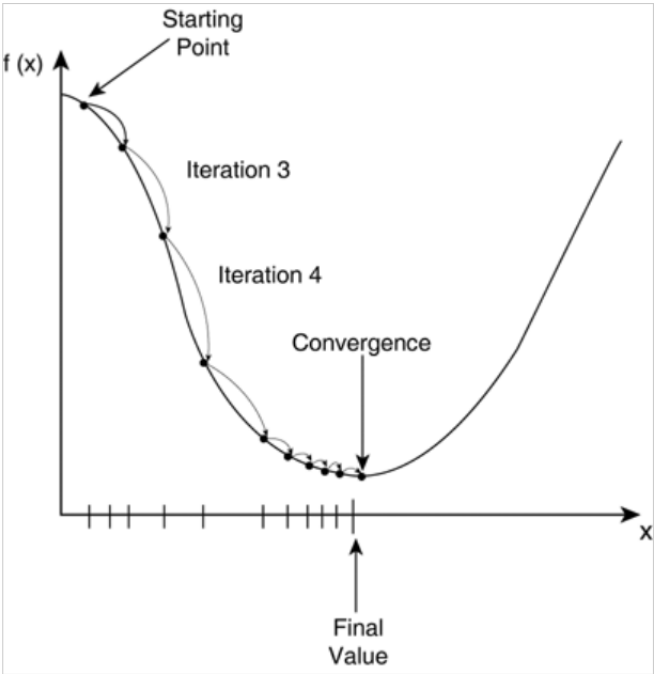

Figura 3.4: Método del gradiente descendiente[14]

Uno de los optimizadores más utilizados es el optimizador adam, el cual varía la tasa de aprendizaje para adaptar los errores del modelo y propicia mejores resultados.

3.5.3. Tipos de redes neuronales

Existen un gran número de posibilidades a la hora de crear una red neuronal, sin embargo, las principales opciones que se utilizan son las siguientes[15]:

- Redes neuronales feedforward (FNN): Son redes neuronales donde las conexiones entre neuronas no forman un ciclo. El flujo de procesamiento de la información se produce únicamente en una dirección pasando en primer lugar por las capas de entrada, las capas ocultas y por último las capas de salida. Existen dos tipos: perceptrón de una capa y perceptrón multicapa.

  Las primeras son el tipo de red neuronal más simple que se conoce, es decir está formada por dos capas: una capa de entrada y una capa de salida, cuya misión es pasar de datos de la capa de entrada directamente a la capa de salida. No es muy utilizada y más en problemas de Deep Learning donde se requiere más complejidad estructural. En la figura 3.3, se explicó el funcionamiento de una neurona del modo más simple posible, pues es básicamente el funcionamiento de este tipo de red, en la que no hay procesamiento entre la entrada y la salida, es decir solo hay dos capas en toda la arquitectura.

  Por otro lado, tenemos las redes de perceptrón multicapa que son aquellas que cuentan con capas ocultas de neuronas. Estas redes son muy versátiles ya que funcionan bien con todo tipo de problemas, pero tienen la gran limitación de ser computacionalmente muy costosas de entrenar en modelos donde haya muchas entradas. En la siguiente imagen podemos ver un ejemplo de este tipo de red:

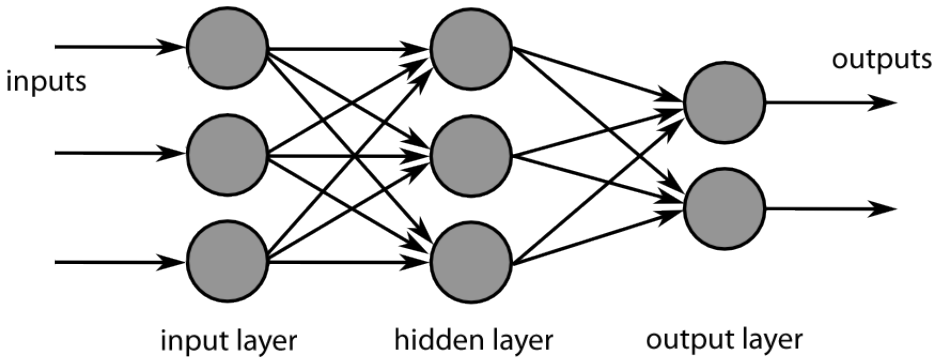

Figura 3.5: Perceptrón Multicapa[11]

Como se ve en la figura 3.5, si las neuronas tienen conexiones con todas las siguientes neuronas, se conoce como una red con conectividad total o full connected. En caso contrario se denomina parcialmente conectada. Otra denominación empleada es en el caso de que existan dos o más capas ocultas formando lo que se conoce como red neuronal densa (DNN). Este tipo de redes entran dentro de la denominación Deep Learning.

- Redes neuronales convolucionales (CNN): Este tipo de redes neuronales tienen un gran uso en distintas áreas de problemas como pueden ser en el campo de la visión artificial con redes convolucionales 2D, y en el campo de la clasificación de textos utilizando redes convolucionales 1D. La idea principal de este modelo es el uso de patrones, en forma de series de palabras, lo que permite que dichos patrones se reconozcan en diferentes posiciones y facilitar el aprendizaje a la red.

  Esta idea se ve reflejada en la figura 3.6, es decir extraer patrones locales de texto (subsecuencias) dentro de las secuencias que corresponden a los textos. Dichos patrones se pueden volver a utilizar y facilitarán la correcta clasificación de los textos.

  Por ejemplo, si tenemos un tamaño de ventana de 5 (kernel size), la red podrá aprender palabras o patrones de palabras de longitud 5 o menos, y será capaz de reconocer dichas palabras en cualquier contexto de cada texto.

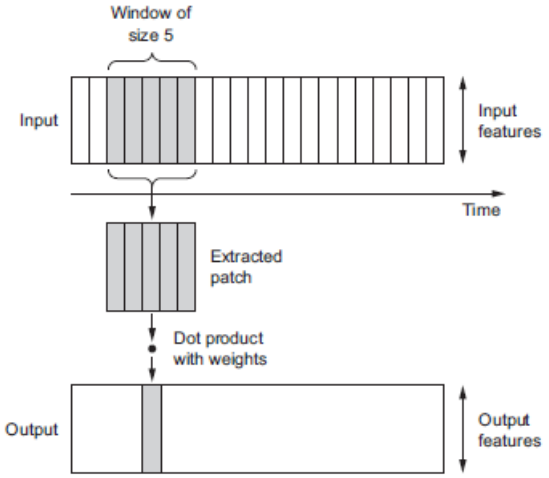

Figura 3.6: Red Convolucional 1D[16]

- Redes neuronales recurrentes (RNN): Las redes neuronales recurrentes (RNN), a diferencia de las DNN y las CNN, tienen memoria, es decir procesan los textos teniendo en cuenta los textos anteriores. Por ejemplo cuando estamos leyendo una página de un libro, tenemos en la mente lo que hemos leído en las páginas anteriores y entendemos dicha página pensando en el contexto en el que se ha leído la página anterior. Este es el principio fundamental de las RNN, procesan información y guardan que información se ha procesado antes, y usarla en el caso de que tenga alguna relación. A priori, dichas redes parece que van a ser las que mejores resultados nos devuelvan, ya que funcionan igual que el cerebro humano. Se pueden implementar de múltiples maneras, aunque la más conocida es utilizar la capa LSTM(Long Short-Term Memory), traducido sería memoria corta a largo plazo. Veremos también el uso de redes neuronales recurrentes GRU.

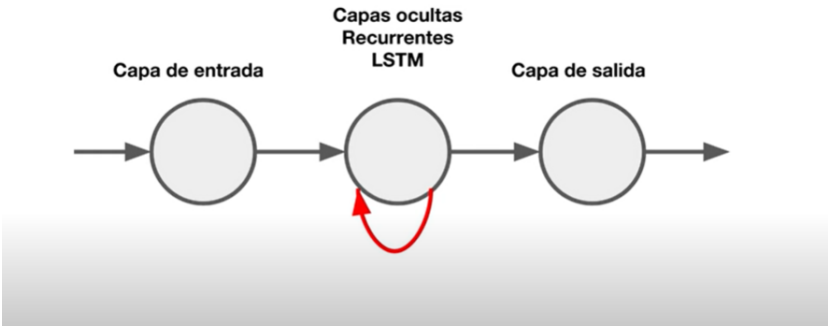

Figura 3.7: Red neuronal recurrente[3]

Como se puede ver en la figura 3.7, la idea de estas capas es que reciben un dato , generan una prediccion, y la salida se alimenta de nuevo como entrada de la capa oculta para la siguiente predicción.

3.6. Métricas de rendimiento

A la hora de evaluar si un modelo es mejor o peor que otro se suelen utilizar una serie de medidas que de forma objetiva nos ayudan a decidir que modelos son mejores que otros. Es importante tener en cuenta que se pueden dar distintas situaciones que se ilustran a continuación.

Como se está creando un clasificador de textos, imaginemos que tenemos un clasificador cuyo objetivo es reconocer si una palabra es o no es un adjetivo. Se pueden dar 4 situaciones:

- El algoritmo clasifica la palabra como adjetivo y si era un adjetivo. A esto se le conoce como Positivo Verdadero (TP).
- El algoritmo clasifica la palabra como adjetivo y no era un adjetivo. A esto se le conoce como Positivo Falso (FP).
- El algoritmo no clasifica la palabra como adjetivo y si era un adjetivo. A esto se le conoce como Falso Negativo (FN).
- El algoritmo no clasifica la palabra como adjetivo y no era un adjetivo. A esto se le conoce como Negativo Verdadero (TN).

Dejando claro esto, se van a definir algunas métricas que se han tenido en cuenta, aunque como veremos más adelante la métrica empleada ha sido la exactitud ya que se ha considerado suficientemente representativa como para decidir si los modelos diseñados son mejores o peores.

El objetivo de una métrica es conocer el funcionamiento de un sistema de computación, en nuestro caso una red neuronal, a través de un valor o conjunto de valores resultados de una serie de procesos. Las métricas que vamos a explicar son la exactitud, la precision, la exhaustividad y el valor-F[1].

- Exactitud (Acuraccy): Es la medida que ha utilizado en la evaluación de los modelos ya que representa la razón entre las predicciones correctas de las predicciones totales, es decir en nuestro caso el número de palabras que ha clasificado correctamente como adjetivos dividido entre el número de palabras totales. Esta medida es útil en situaciones en las que las clases del modelo están balanceadas, algo que se explicará en los siguientes apartados.

  La fórmula es:

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

- Precision (Precision): Es la razón entre el número de documentos clasificados correctamente como pertenecientes a la clase A y el número total de documentos de que han sido clasificados por el modelo como de clase A. Es una métrica que se tiene que calcular para cada una de las clases del problema.

  La fórmula es:

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

- Exhaustividad (Recall): Es la relación entre los documentos clasificados correctamente como pertenecientes a la clase A y la suma de todos los documentos de la clase A. Es una métrica que se tiene que calcular para cada una de las clases del problema.

  La fórmula es:

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

- Valor-F (F-score): La fórmula del valor-F combina las dos medidas anteriores de manera ponderada a través de un parámetro beta lo que permite otorgar una mayor importancia a una que a otra.

  La fórmula es:

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

3.7. Herramientas utilizadas

Las herramientas que se han utilizado durante el desarrollo del proyecto han sido las siguientes:

3.7.1. Python

Python[17] es un lenguaje de alto nivel muy utilizado para resolver problemas mediante Inteligencia Artificial ya que tiene librerías que facilitan el desarrollo de algoritmos de IA como pueden ser las redes neuronales y a su vez es muy sencillo el procesamiento de datos necesario antes de crear una red neuronal. El desarrollo de este trabajo por tanto ha sido mediante dicho lenguaje de programación.

Según varios expertos, afirman que el 57 % de los ingenieros/científicos de datos utilizan Python[18]. Es un lenguaje multiparadigma, ya que soporta orientación a objetos, programación imperativa y

programación funcional.

Por ejemplo grandes multinacionales como la NASA han desarrollado sus modelos de predicción de condiciones meterológicas en sus lanzamientos en Python o Tesla, la cual programó el autopilot de los coches en Python.

3.7.2. Tensorflow y Keras

Tensorflow[19] es un framework a bajo nivel desarrollada por Google en el 2017 y se utiliza para crear aplicaciones de aprendizaje automático. Al ser a más bajo nivel, conlleva más trabajo debido a que no está todo tan triturado como en Keras.

Keras[20] es una biblioteca que está integrada en Tensorflow y se utiliza para programar redes neuronales a más alto nivel, por lo que facilita el proceso. Keras permite la creación de redes neuronales de manera muy sencilla ya que lo único que necesitas es elegir los parámetros e hiperparámetros de la red neuronal que se explicarán en el siguiente capítulo con detalle. Despúes compilar la red creada, entrenarla y obtener resultados.

También se ha utilizado keras-tuner[21], que es una funcionalidad que viene incluida para la búsqueda de los hiperparámetros de las redes neuronales que se verán más adelante.

3.7.3. NLTK

El kit de herramientas de lenguaje natural (NLTK)[22] es un conjunto de bibliotecas desarrolladas en Python que se utilizan para facilitar la resolución de problemas de procesamiento del lenguaje natural (PNL).

3.7.4. Google Colab

Google Colab[23] ha sido la plataforma utilizado para alojar y ejecutar el código de creación de las redes neuronales. Es un entorno de ejecución en la nube creado por Google que te permite cargar y modificar cuadernos Jupyter usado principalmente en proyectos de investigación y creacción de algoritmos de Machine Learning.

3.7.5. Overleaf

Overleaf[24] es un editor colaborativo de LaTeX basado en la nube que se utiliza para escribir, editar y publicar documentos científicos. Tanto la plantilla del TFG como el desarrollo y escritura del informe se han hecho mediante este editor.

3.8. Twitter

Twitter[25] es una red social fundada en 2006 que se autodenomina como un servicio de microblog. Un microblog consta de mensajes de corta duración y de fácil publicación en la que los usuarios expresan opiniones acerca de distintos temas de actualidad. Por lo tanto es una gran oportunidad para realizar un estudio de los sentimientos de estos mensajes que se publican denominados tweets.

Twitter es un medio de comunicación y extracción de información muy potente ya que cuenta con más de 200 millones de usuarios activos, es decir que escriben tweets frecuentemente, con casi 500 millones de tweets publicados diariamente[26].

No solo usuarios particulares expresan sus opiniones, si no que grandes empresas y personas públicas hacen uso de esta red social, lo que hace más interesante su análisis.

Algunos de los conceptos más importantes que son necesarios de entender son los siguientes[27]:

- Un tweet es un mensaje que no puede superar los 280 caracteres.
- Una mención es una manera de que otro usuario pueda ver el mensaje que se ha publicado y va precedido de un @nombreusuario.
- Un retweet es la republicación de un tweet lanzado por otro usuario.
- Un hashtag es una cadena de texto que va precedida de la almohadilla y sirve para facilitar las búsquedas.

Capítulo 4

Análisis de sentimientos en Twitter

4.1. Procesamiento del lenguaje natural

Una definición adecuada del procesamiento del lenguaje natural (PLN) es la siguiente: Es una subdisciplina de la Inteligencia Artificial. El PLN se ocupa de la formulación e investigación de mecanismos eficaces computacionalmente para la comunicación entre personas o entre personas y máquinas por medio de lenguajes naturales.

Por lo tanto es necesario definir una serie de protocolos que hacen más sencilla la comunicación persona-computador. Una de los grandes usos que tiene el PLN es el de la clasificación de textos, que veremos a continuación.

4.2. Problemas de clasificación de textos

Nuestro estudio se centra en el análisis de sentimientos, que se corresponde a problemas de clasificación de textos. La clasificación de textos es un proceso que consiste en categorizar textos en diferentes grupos en función de una serie de parámetros.

Existen otros tipos de problemas que entran dentro de esta categoría y son los siguientes[28]:

- Análisis de sentimientos: este es nuestro problema a resolver y basicamente consiste en clasificar textos en función de la polaridad que representan dichos textos. Estos textos pueden ser tweets, opiniones de productos, opiniones de películas...
- Categorización de noticias: en este tipo de problemas se pretende clasificar las noticias por el tema que trata. Por ejemplo noticias de COVID-19, noticias de Ucrania...
- Análisis de temas: en este tipo de problemas se pretende obtener el tema sobre el que trata un texto. Por ejemplo si una revisión de un producto es sobre facilidad de uso o atención al cliente...

- Respuestas a preguntas: Dada una pregunta y una lista de respuestas intenta clasificar como válidas o no válidas las distintas respuestas a la pregunta.

4.3. Niveles del análisis de sentimientos

Como ya se ha dicho, nuestro problema en concreto se trata de un problema de análisis de sentimientos a partir de la clasificación de textos.

Dentro de este tipo de problemas podemos encontrar tres niveles para resolverlo[1]:

- Análisis a nivel de documento: en este nivel se analiza el sentimiento global de un documento como un todo indivisible clasificándolo como positivo, negativo o usando otro sistema de clasificación.
- Análisis a nivel de oración: en este caso, se divide el documento en oraciones individuales para extraer posteriormente la opinión que contiene cada una de ellas. La opinión de cada oración puede ser, de nuevo, positiva, negativa o bien tomar un valor en base a cualquier otro tipo de medida.
- Análisis a nivel de aspecto y entidad: este es el nivel de análisis con mayor detalle posible, en donde una entidad está formada por distintos elementos o aspectos y sobre cada uno de ellos se expresa una opinión cuya polaridad puede ser distinta en cada caso. Es el nivel más complicado de analizar, pero el más concreto y específico.

En nuestro estudio se ha llevado a cabo un análisis a nivel de documento, tomando en cuenta cada tweet como un documento.

4.4. Proceso de entrenamiento de un clasificador de textos

Nuestro problema se trata de una clasificación de textos, por lo que hay que tener en cuenta cuál es la metodología para resolver este tipo de problemas.

4.4.1. Descripción del corpus de aprendizaje

En primer lugar se parte de un corpus de aprendizaje, el cúal está formado por 72161 tweets con su polaridad correspondiente. Un corpus de aprendizaje es un conjunto de datos de partida que sirven para poder entrenar a nuestra red neuronal. Dicho corpus se ha extraido concatenando varios años del TASS (Taller Análisis Semántico)[29].

Nuestro corpus de entrenamiento está formado por 72161 tweets de los cuales tienen la siguiente distribución: 25836 positivos, 20007 negativos, 3272 neutros y 23046 sin sentimiento.

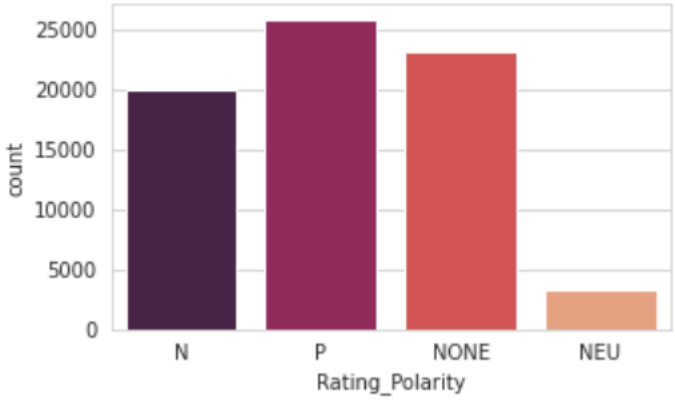

Figura 4.1: Numero de tweets por clase en el corpus

Dentro de las redes neuronales podemos distinguir distintos tipos de problemas de machine-learning en función del propósito que queramos. Los más utilizados son los siguientes:

- Problemas de clasificación binaria: En estos problemas se pretende realizar una clasificación en dos clases. Ej Clasificación de una imagen en función de si es un gato o un perro.
- Problemas de clasificación multiclase: A diferencia de la anterior, se pretende hacer una clasificación en más de dos clases. Ej Clasificación de una imagen en función si es un gato, un perro o un pájaro.
- Problemas de regresión: estos problemas no clasifican, si no que predicen un valor. Ej Predecir la temperatura que va a hacer mañana.

En nuestro caso concreto se trata de un problema de clasificación multiclase ya que presentamos los siguientes posibles sentimientos: Positivo(P), Negativo(N), Neutro(NEU), Ninguno(NONE). Además como cada tweet solo puede pertenecer a un sentimiento se conoce como problema *single-label, multiclass classification*. Aclarar que un tweet sea neutro quiere decir que tiene tanto sentimientos positivos como sentimientos negativos, por lo que se queda en neutro.

Un aspecto a tener muy en cuenta es el balance de clases, ya que si estamos resolviendo un problema de clasificación y tenemos más datos de una clase que de otra, de cierto modo va a ser más probable que la red neuronal prediga un tweet en la clase más abundante.

En las pruebas que se han realizado se ha empezado probando con las 3 clases más representativas, es decir tweets Positivos, Negativos y NONEs. Posteriormente se podrá comprobar cuál es el funcionamiento con un número distinto de clases. Luego vamos a trabajar con las clases Positivo, Negativo y None con 25836 positivos, 20007 negativos y 23046 sin sentimiento.

Si queremos probar con 4 clases, como se ve en la figura 4.1, tendríamos que equilibrar de alguna forma el número de tweets de la clase NEU ya que como se ha dicho el clasificador no funcionaría del todo bien. Se pueden añadir más muestras o generar nosotros muestras artificiales

que equilibren los datos. Esto no se ha aplicado ya que no es nada recomendable generar muestras artificiales ya que no sabemos como se generan estas muestras ni sabemos si reprensentan de la manera en la que queremos los datos, por lo que puede llevar a sobreajuste[30].

En las figuras 4.2 y 4.3 se puede apreciar una *wordcloud*, o nube de palabras que sirve para mostrar las palabras de los tweets que más se repiten de una clase. Se han incluido de las clases positiva y negativa.

Figura 4.2: Nube de palabras de la clase Negativa

Figura 4.3: Nube de palabras de la clase Positiva

4.4.2. Preprocesamiento de datos y codificación

Para tener una referencia a la hora de explicar los pasos se ha tenido en cuenta el TFM de José Carlos Sobrino Sande[1]. El primer paso a la hora de afrontar este tipo de problemas es el

preprocesamiento, es decir de modo básico, tenemos que hacer que la red entienda los textos en las mismas condiciones. En el preprocesamiento se han tenido en cuenta los siguientes apartados:

- Normalización de mayúsculas y minúsculas.
- Eliminación de tildes y vocales diacríticas.
- Reducción de la repetición de caracteres.
- Normalización de la risa.
- Normalización de las jergas.
- Eliminado de menciones, hashtags, retweets y enlaces.
- Eliminación de signos de puntuación.

El siguiente paso es la tokenización que consiste en considerar cada palabra de cada tweet como un token. Tenemos que tener en cuenta que en twitter hay un vocabulario específico luego usaremos un tokenizador particular creado para twitter denominado TweetTokenizer[31].

En la extracción de características, a partir de los tokens obtenidos en el paso anterior lo que se hace es transformar el texto a números, ya que una red neuronal no puede ser alimentada por cadenas de texto. Las entradas de la red neuronal tienen que estar únicamente formada por números y en nuestro caso tenemos que transformar dos partes: La primera serían los tweets tokenizados y normalizados, que normalmente se llama `data` y las etiquetas que corresponden con una letra y que representan al polaridad de cada tweet, también llamadas `target`.

Para transformar los data en números se ha creado un diccionario por el que cada tweet será representado por un vector de índices correspondientes en el diccionario. Esto se explicará con más detalle en la sección de la incrustación de palabras.

Para transformar los target se ha utilizado *One Hot Encoding*, un mecanismo que codifica las distintas clases (P, N y NONE) como una matriz en este caso de 3 columnas, en la que se pone un 1 en la clase que corresponde un tweet y un 0 en las otras.

Esta técnica se puede ver ilustrada en la siguiente figura 4.4

	Tweet	N	NONE	P
2	gonzal altozan tras presentacion de ...	0	0	1
6	buen noch followercet mañan va ...	0	0	1
7	mas de mañan en gacet upyd cont...	0	0	1
8	felic no en grand anhel sin...	0	0	1
11	habi promet respond a tod per ...	0	0	1

Figura 4.4: One Hot Encoding

La reducción de características es un paso opcional, y consiste en reducir el vocabulario que puede manejar la red neuronal. En nuestro caso se han contemplado el utilizar tres técnicas:

- Eliminación de stopwords: existe un conjunto de palabras que, aunque son necesarias para construir oraciones con sentido, carecen de información que ayude a determinar la polaridad de los textos en los que se encuentran. En español, estas palabras son las preposiciones, los pronombres, las conjunciones y las distintas formas del verbo haber, entre otras.

- Stemming: es un proceso de transformación de normalización morfológica por el que una palabra se transforma a su raíz por medio de la supresión de sus sufijos e inflexiones. Siguiendo el ejemplo anterior, la palabra "guapas" se convertiría a su raíz, "guap".

- Lematización: este es un proceso de normalización morfológica que transforma cada palabra en su lema mediante el uso de diccionarios y de un proceso de análisis morfológico. Por ejemplo, la lematización convertiría la palabra "guapas" a su lema "guapo".

Para ver los conceptos de stemming y lematización más claros, en la figura 4.5 podemos apreciar un ejemplo y como se convertiría un texto aplicando dichos procesos de reducción de características.

```
[23] import spacy
     #_nlp = spacy.load('es_core_news_sm')
     text_auxiliar="Hola retratando he retradado al retratado"
     texto=""
     for word in nlp(text_auxiliar):
         texto=texto+word.lemma_+" "
     print(texto)

     Hola retratar haber retradado al retratar
```

```
[22] from nltk.stem import SnowballStemmer
     from nltk import TweetTokenizer
     _stemmer = SnowballStemmer('spanish')
     _tokenizer = TweetTokenizer().tokenize
     text="Hola retratando he retradado al retratado"
     text = ' '.join(_stemmer.stem(w) for w in _tokenizer(text))
     print(text)

     hol retrat he retrad al retrat
```

Figura 4.5: Ejemplo de diferencias entre Stemming y Lematización

El último paso antes de entrenar la red neuronal es el de la ponderación de las características, es decir dar un peso a las características que se han extraído. En este punto tenemos texto transformado en números, pero exactamente que números son los que se tienen que rellenar para representar los tweets, pues bien, en este paso lo que se hace es eso, decidir la manera en la que se asignan dichos pesos y que un tweet se convierta en un array de números ponderados.

Figura 4.6: Proceso de entrenamiento[1]

4.5. Elección de hiperparámetros en modelos

Los hiperparámetros son aquellas variables de una red neuronal de las que escogemos un valor en el proceso anterior al entrenamiento de la red. Algunos ejemplos de hiperparámetros son: el número de capas de la red, el número de neuronas por capa, la función de activación, el optimizador, la tasa de aprendizaje, el batch size... Como es lógico, dependiendo de qué valores escojamos, vamos a obtener unos resultados mejores o peores. No hay unos valores prefijados, ya que dependen en gran medida del problema a resolver y de los recursos computacionales disponibles por lo que es una tarea fundamental.

4.5.1. Técnicas de búsqueda de hiperparámetros

Las dos principales técnicas para la búsqueda de hiperparámetros son:

- Grid Search: Esta estrategia lo que hace es crear una malla de n dimensiones, a la que a cada una de ellas se le asigna un hiperparámetro. Para cada hiperparámetro (dimensión) se define un rango de valores y se comprueban todas las combinaciones posibles para ver con que hiperparámetros la red neuornal se comporta mejor. Si por ejemplo tenemos x hiperparámetros, de los que pueden tomar un rango de valores de y cada uno, necesitariamos entrenar a nuestro modelo:

  $$y^x$$

  por lo que computacionalmente hablando es bastante inviable.

  Se puede considerar su uso en situaciones en las que haya como mucho 4 hiperparámetros para considerar un método efectivo. La principal ventaja de esta aproximación es la efectividad del mismo, es decir por fuerza bruta va a encontrar los mejores hiperparámetros, pero a un alto coste computacional.

  Otro incoveniente que tiene este método es el hecho de que tu eliges los valores que el algoritmo utiliza, es decir no sabemos si existen otros valores que arrojen mejores resultados de precisión por lo que habría que probar con un amplio abanico de valores.

- Random Search: Esta estrategia es similar a la anterior, pero se diferencia fundamentalmente en la idea de elegir los valores de los hiperparámetros aleatoriamente, por lo que se entrenarían los modelos de una forma más eficiente en caso de un gran número de hiperparámetros. La idea es buscar en menores iteraciones cuales son los mejores resultados. La gran pega que tiene esta aproximación es que no te garantiza encontrar la mejor combinación posible, y estadísticamente hablando depende de la suerte.

En la siguiente foto podemos[32] ver una ilustración de como funcionan las dos aproximaciones. Como vemos, dependemos un poco de la suerte ya que en la primera aproximación, sabemos si o si que tenemos que pasar por los valores de los hiperparámetros que den peores resultados, lo que conlleva un gasto significativo de tiempo, y en la segunda vemos que si las pruebas aleatorias son las mejores, se reducirá el gasto en tiempo de la búsqueda de los mejores valores.

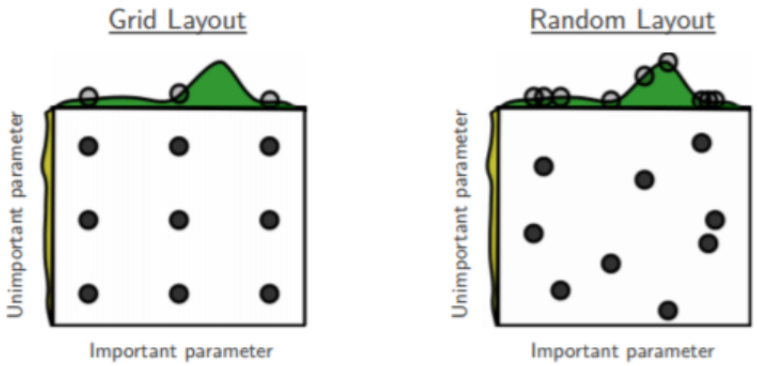

Figura 4.7: Grid Layout vs Random Layout[32]

Existen otros métodos para realizar una búsqueda de hiperparámetros, y una cosa muy a tener en cuenta es el hecho de combinar un buen diseño arquitectónico de la red neuronal con una buena elección de hiperparámetros.

En nuestro caso en un primer instante se ha realizado la técnica conocida como prueba y error en la que mediante un gran número de ejecuciones se iban descartando valores de hiperparámetros que no funcionaban bien, para así recortar el espacio disponible de búsqueda de los mismos. Esta técnica se combinó con distintas referencias de otros algoritmos que resolvían problemas similares en las que ciertos valores de hiperparámetros se sabía que iban mejor que otros.

Después para afinar un poco más la búsqueda se utilizó una biblioteca llamada keras-tuner[21] la cual facilita la programación para la búsqueda de hiperparámetros. Keras-Tuner incorpora distintos tipos de búsqueda, aunque la que se eligió en este caso ha sido el RandomSearch.

Como se ve en la figura 4.8, se define un modelo con un parámetro que es el que va tener las funciones de keras-tuner. Por ejemplo, se ha establecido una capa LSTM cuyas neuronas pueden tener los valores 30, 60, 90 o 120. La tasa dropout es un valor entre 0.2 o 0.3 y la tasa de dropout

recurrente entre 0.2 y 0.3. En este caso se ha utilizado las funciones Int() y Choice() para ajustar los valores. Se pueden experimentar con muchas más combinaciones pero como decía se ha realizado una fase previa de prueba y error para descartar ciertos rangos de valores que no sean típicos.

```
def createModel(hp):

    model = Sequential()
    embedding_layer = Embedding(vocab_size, 200, input_length=maxlen)
    model.add(embedding_layer)
    neurons = hp.Int('units', min_value=30, max_value=120, step=30)
    dropout_N=hp.Choice('dropout_N', values=[0.2,0.3])
    dropout_R=hp.Choice('dropout_R', values=[0.2,0.3])

    model.add(LSTM(neurons , dropout=dropout_N, recurrent_dropout=dropout_R))
    #learning_rate = hp.Choice('learning_rate', values=[0.001,0.01,0.1])

    #loss = hp.Choice('loss', values=['binary_crossentropy'])

    model.add(Dense(3, activation='softmax'))
    model.compile(optimizer="adam", loss='binary_crossentropy', metrics=['accuracy'])

    return model

tuner = RandomSearch(createModel,
                     objective="val_accuracy",
                     max_trials=12,
                     executions_per_trial=2,
                     directory='tuner6',
                     project_name='PruebitasTFG')
```

Figura 4.8: Ejemplo de keras-tuner

Una vez se compila el modelo, se realiza la búsqueda mediante combinaciones aleatorias de los valores seleccionados. En este caso se han realizado 12 combinaciones.

Como se puede ver en la figura 4.9, nos muestra cual ha sido la mayor precisión de validación obtenida y el tiempo total del proceso. Obviamente cuanto más complicado sea el modelo, cuantos más hiperparámetros se quieran ajustar y cuantas más combinaciones se quieran probar, más tiempo tardará el algoritmo en finalizar.

```
early_stop = EarlyStopping(monitor = 'val_loss', mode = 'min', verbose = 1, patience=2)
tuner.search(X_train, y_train, epochs=100, batch_size=256, validation_data=(X_test, y_test), callbacks=[early_stop])
```

```
Trial 12 Complete [00h 09m 48s]
val_accuracy: 0.7213673889636993

Best val_accuracy So Far: 0.7249963581562042
Total elapsed time: 02h 45m 15s
```

Figura 4.9: Ejemplo de keras-tuner

4.5.2. Hiperparámetros más importantes

Como ya se ha comentado antes, existen ciertos tipos de patrones para los que algunos hiperparámetros funcionan mejor o peor con un tipo de problema. Por ejemplo, en el caso de las funciones de activación[33], como se ve en la figura 4.10, para capas densas y capas convolucionales dan mejores resultados la función de activación relu, mientras que en las capas recurrentes se utilizan las funciones de activación sigmoid y tanh.

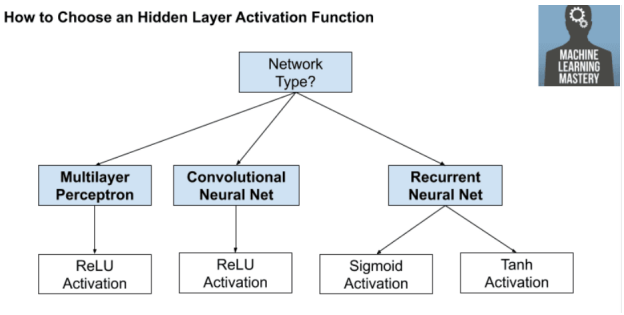

Figura 4.10: Mejores funciones de activación capas ocultas[33]

En la figura 4.11, podemos ver un desglose para elegir la función de activación de la capa de salida. En nuestro caso al ser un problema de clasificación multiclase se utilizará la función softmax.
La función softmax devuelve un array con las probabilidades de cada tweet de pertenecer a una clase eligiendo la clase que mayor probabilidad tenga. La suma de estos valores tiene que dar 1.

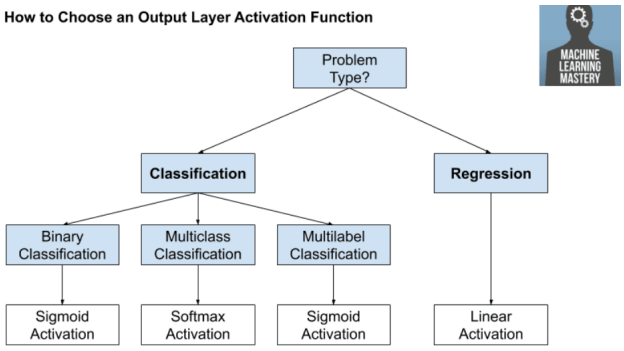

Figura 4.11: Mejores funciones de activación capas de salida[33]

Otros hiperparámetros que se recomiendan utilizar para nuestro tipo de problemas son los siguientes[16]:

- Optimizador rmsprop o adam: Ambos optimizadores son los más usados y su elección se debe al recurrente uso en diversos experimentos consultados. En principio, Adam es mucho más versátil por lo que funciona mejor en más situaciones, pero quería hacer hincapié en Rmsprop debido a su recurrente insistencia en diversos libros. El optimizador Adam es una mezcla de Rmsprop con diversos factores que por lo general hacen que el modelo ofrezca mejores resultados.

- Función de pérdidas binary_crossentropy o categorical_crossentropy: Dichas funciones de pérdidas son las utilizadas en problemas de clasificación, la primera de ellas se utiliza en problemas donde la entrada se puede clasificar en dos etiquetas, y la segunda de ellas se emplea en situaciones en las que la entrada se clasifica en tres o más etiquetas[16].

La tasa de aprendizaje o learning rate es el hiperparámetro más crucial que podemos ajustar en nuestro modelo de red neuronal, razón por la que dedicamos una explicación más amplia.

Este hiperparámetro en términos informales es el encargado de hacer que nuestra red converga rápido o lento, y que por lo tanto pueda hacer que nuestra red se quede en una especie de bucle infinito y no poder ofrecer buenos resultados ya que no es capaz de aprender en un tiempo finito.

La tasa de aprendizaje es un parámetro que tienen los optimizadores y es muy importante elegir adecuadamente su valor. Una tasa de aprendizaje elevada permite a la red neuronal converger hacia un valor apropiado con un número de iteraciones reducido, mientras que una tasa de aprendizaje pequeña permite que el restultado sea más preciso aunque se tendrán que utilizar más iteraciones[14].

En la figura 4.12 se puede apreciar esto, como se puede apreciar se han realizado diferentes pruebas con la misma arquitectura de red y los mismos hiperparámetros a excepción de fluctuar la tasa de aprendizaje en los valores 0.0001, 0.001, 0.01, 0.1 y 0.02. Como se observa es bastante curioso que en algunos casos la precisión disminuye hasta 0.39 con un valor de 0.1. Por defecto en los optimizadores viene el valor 0.001, ya que es el estándar, pero es algo que tenemos que comprobar porque puede ser el causante de que con nuestro modelo no se obtengan los mejores resultados.

```
Results summary
Results in tuner7/PruebitasTsFG
Showing 10 best trials
<keras_tuner.engine.objective.Objective object at 0x7f6301ff9e50>
Trial summary
Hyperparameters:
learning_rate: 0.001
Score: 0.7293753325939178
Trial summary
Hyperparameters:
learning_rate: 0.0001
Score: 0.7233028709888458
Trial summary
Hyperparameters:
learning_rate: 0.0002
Score: 0.7211012840270996
Trial summary
Hyperparameters:
learning_rate: 0.01
Score: 0.7205690443515778
Trial summary
Hyperparameters:
learning_rate: 0.02
Score: 0.5322978645563126
Trial summary
Hyperparameters:
learning_rate: 0.1
Score: 0.39149367809295654
```

Figura 4.12: Hiperparámetro learning rate

4.6. Sobreajuste

Uno de los principales problemas que nos podemos encontrar es el sobreajuste. El sobreajuste[34] consiste en crear modelos tan ajustados a los datos de entrenamiento que no se extrapolen a los datos del test. Es decir, que aprenda muy bien con los datos de entrenamiento, pero que con datos completamente nuevos, no funcione tan bien, y esto es un problema muy serio. Un ejemplo que me gustaría ilustraros para que se entienda aún mejor es el siguiente: Te estás preparando un examen de matemáticas y sabes que van a caer problemas similares a los que haces en clase (entrenamiento), pero luego en el examen te modifican el esquema del problema y no sabes como resolverlo (test), pues esto es producido por aprender memorizando en lugar de entendiendo.

Con este término lo que se quiere ilustrar son los conceptos de optimizar y generalizar, pues el primero busca para los datos propuestos (entrenamiento), buscar el mejor resultado posible, y el segundo lo que busca es que para cualquier entrada (test), arroje buenos resultados. Se tiene que buscar un equilibrio entre dichos conceptos encontrando la mejor combinación posible para el problema en cuestión.

Existen algunas soluciones que intentan regular el sobreajuste para evitar que los modelos pierdan cierto nivel de generalización.

- Dropout: Esta técnica[35] consiste en desactivar ciertas neuronas de forma aleatoria que irán cambiando en cada iteración. Básicamente las neuronas cercanas suelen aprender patrones que se relacionan y estas relaciones pueden llegar a formar un patron muy especifico con los datos de entrenamiento, con dropout esta dependencia entre neuronas es menor en toda la red neuronal, lo que hace que las neuronas necesitan trabajar mejor de forma solitaria y no depender tanto de las relaciones con las neuronas vecinas.

- EarlyStopping: Esta técnica consiste en detener el entrenamiento del modelo en un punto donde se minimize la pérdida de validadicón. Básicamente en cuanto la métrica de pérdida de entrenamiento haya dejado de mejorar, el entrenamiento se detiene. Es el método más usado para regular el sobreajuste ya que es sencillo de implementar y ofrece buenos resultados.

Con la técnica EarlyStopping solucionamos el sobreajuste de una manera rápida y sencilla. En nuestro caso está configurado con un parámetro para que se realizen 2 epoch a mayores por si ha habido un error y no ha sido sobreajuste. En la siguiente figura se muestra un ejemplo de sobreajuste y de como se debería de comportar el algoritmo para evitar el sobreajuste. Como se ve en la figura 4.13, tanto como la precisión de entrenamiento como la de validación se muestran en valores muy cercanos, lo que es una buena señal de que el modelo funciona correctamente.

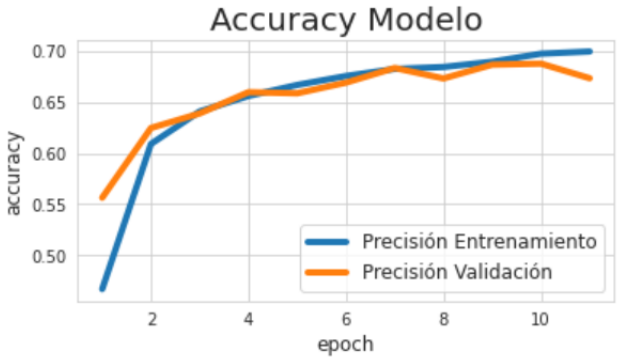

Figura 4.13: Ejemplo sobreajuste

En la epoch 10, la precisión de validación empieza a descender de forma agresiva, pues es en este momento en el que nuestro modelo ha sufrido un sobreajuste. La solución para evitar esto es volver a ejecutar el código ajustando el número como se muestra en la siguiente figura:

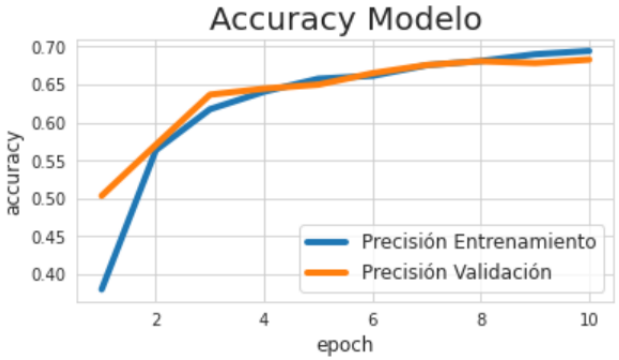

Figura 4.14: Ejemplo sobreajuste 2

4.7. Evaluación de modelos

Un aspecto fundamental a la hora de evaluar los modelos, es el tipo de evaluación realizada ya que la precisión de nuestro modelo se verá claramente influenciada por el método seleccionado. El proceso que se sigue para evaluar un modelo consiste en tener dos sets distintos: el set de entrenamiento y el set de validación. Con el primero de ellos entrenamos nuestro modelo y con el segundo de ellos lo evaluamos. Por cada set se obtiene una precisión, siendo realmente útil la precisión obtenida del conjunto de validación ya que no podemos tener en cuenta la obtenida del conjunto de entrenamiento debido a que el modelo carecería de la propiedad de generalización.

Normalmente se suele dividir el conjunto de datos en un 70-80 % para el set de entrenamiento y un 20-30 % para el set de validación. La función train_test_split de la librería sklearn es la usada para este proceso y en nuestro caso se ha dividido en una proporción 70-30. Para evaluar un modelo existen dos opciones genéricas.

En primer lugar tenemos la validación manual, esta consiste en realizar una evaluación utilizando los conjuntos obtenidos en la aplicación de la función de división. Es la más intuitiva pero tiene un gran problema, y es el siguiente: ¿Qué pasa si los conjuntos que se han seleccionado no son los idóneos para evaluar el modelo?, es decir no podemos saber si el set de entrenamiento y el set de validación que se han seleccionado son los mejores que resulven dicho problema.

Para solucionar esto, se creó la validación cruzada, que intenta encontrar los mejores sets para el modelo que estamos entrenando. Existen distintos tipos de validación cruzada, aunque el más usado y el que vamos a explicar se conoce como validación cruzada k-fold.

La idea que abarca esta aproximación se puede apreciar en la figura 4.15, consiste en realizar k iteracciones, de tal manera que estás entrenando tu modelo y evaluandolo k veces. Esto se aplica sobre el conjunto de entrenamiento, por lo que ahora tenemos 3 sets: el conjunto de entrenamiento, el conjunto de validación sobre el conjunto de entrenamiento, y el conjunto de validación final, es decir el que de verdad comprueba la generalización del modelo. Esto trata de potenciar el concepto de independencia de los sets.

La precisión final será el resultado de la media de la precisión de las k validaciones realizadas en el proceso.

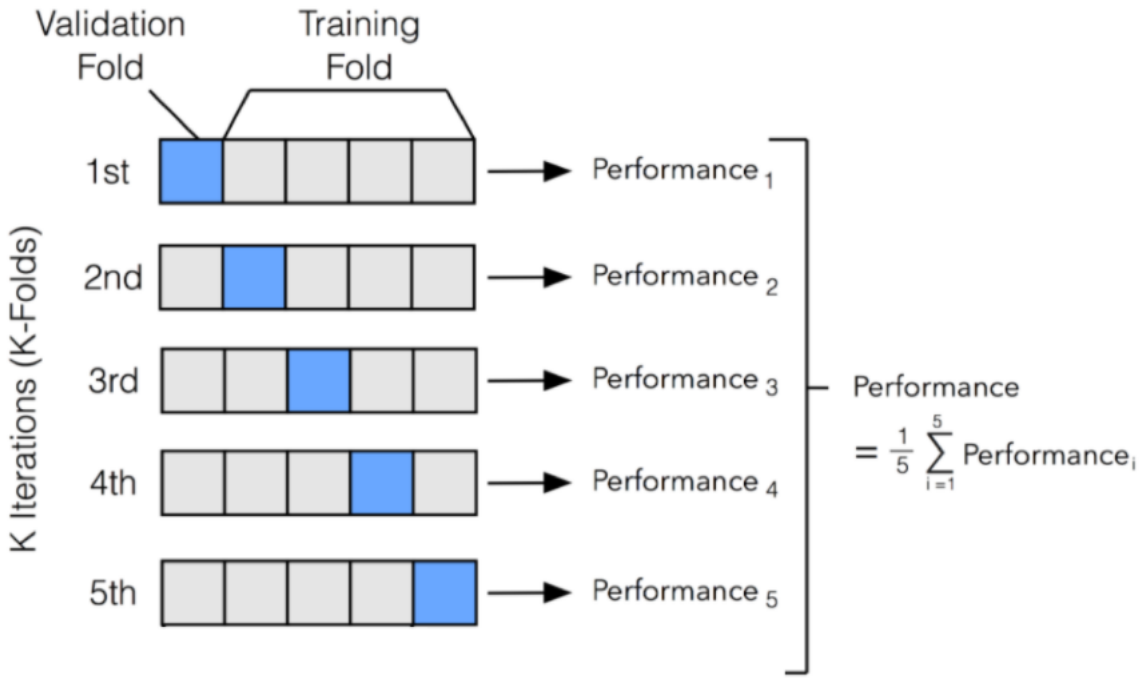

Figura 4.15: Cross Validation[36]

Por lo general, la validación cruzada obtiene ligeros mejores resultados, ya que con la evaluación manual realmente no estamos siendo nada concretos ya que en diversas ejecuciones puede variar la precisión obtenida por el modelo. De ahí nace la validación cruzada, la cual nos permite obtener

ligeras mejoras de precisión debido a que si en la evaluación manual el conjunto de entrenamiento y test es el peor, estaríamos obteniendo un resultado erróneo o de la forma contraria, si cogemos la mejor combinación de entrenamiento y test, estaríamos engañando sobre la precisión de nuestro modelo.

En resumen, si queremos obtener resultados fiables que no dependan de la aleatoriedad del conjunto de entrenamiento y test seleccionado, es necesario utilizar la validación cruzada.

4.8. Modelos Deep Learning

En esta sección se presentarán diversos modelos Deep Learning de los que se han realizado distintas pruebas para su evaluación. Finalmente se ha elegido uno de los modelos para poder seguir optimizándolo.

4.8.1. Incrustaciones de palabras

Una de las técnicas de Deep Learning más usadas se conoce como Word Embedding[37], o traducido, incrustación de palabras. Como ya sabemos, las redes neuronales tienen que ser alimentadas mediante vectores de números o también conocidos como tensores. El procedimiento para transformar los textos de entrada en números se puede hacer de distintas maneras. Por ejemplo, en el modelo básico del que se partía en el TFG se utilizó la técnica de Bow y TF-IDF para la extracción de características y la ponderación de los pesos.

La diferencia fundamental que existe entre usar este tipo de aproximaciones o usar word embedding, se halla en el hecho de que los vectores que representan a las palabras pueden ser enormes. Por ejemplo, si en nuestro corpus tenemos un total de 5 millones de palabras únicas, y queremos representar una frase de 20 palabras, se va a generar un vector de 5 millones de dimensiones de los que sólo 20 índices tendrán un valor distinto de 0. Esto es un gran problema en cuanto a eficiencia ya que requiere un mayor tiempo de entrenamiento y un mayor gasto de recursos.

En word embedding cada palabra se representa como un vector de n dimensiones. La idea fundamental de las incrustraciones de palabras (Word Embedding) se basa en representar a las palabras que tengan cierta relación semántica con valores parecidos.

Hay dos aproximaciones que se pueden seguir:

- Aprender incrustaciones de palabras desde cero. Se empezaría de vectores de palabras aleatorias y se irían ajustando mediante el algoritmo de propagación hacia atrás como lo hace una red neuronal con los pesos.
- Cargar en el modelo incrustraciones de palabras ya creadas específicas para problemas de procesamiento de textos. A esta técnica se la llama incrustación de palabras preentrenadas.

Esta idea pretende consultar un diccionario en el que la palabra serán las claves, y los valores serán listas de valores de las respectivas palabras.

Dentro de este último grupo, se pueden utilizar diferentes modelos creados, que se conocen como bases de datos de palabras incrustadas. Los más usados en este tipo de tareas son los siguientes[38]:

- GloVe[39]
- FasText[40]
- Word2Vec[41]

Para implementar la incrustación de palabras, en Keras disponemos de la capa Embedding, la cual se puede entender como un diccionario que mapea indíces de enteros a vectores de números, es decir cada palabra tiene asociada un índice, pues se utiliza para consultar el vector correspondiente a esa palabra.

Dicha capa recibe como entrada vectores de 2D de la forma (samples, sequence_length) y devuelve vectores de 3D de la forma (samples, sequence_length, embedding_dimensionality).

En la siguientes figuras se pretende explicar los pasos para tener una matriz de palabras incrustadas. Partimos de cadenas de caracteres que en nuestro caso son tweets. Dichas cadenas necesitan ser convertidas en números, pues con el método fit_on_texts lo que hacemos es crear un diccionario interno donde la palabra es la clave y el valor es el índice que le corresponde en el diccionario. El método texts.to_sequences transforma las palabras en el índice que le corresponde en el diccionario. Después tendremos que rellenar con 0 aquellas posiciones de los vectores que no cumplan una longitud ya que todas las cadenas tienen que tener el mismo tamaño a través de la función pad_sequences.

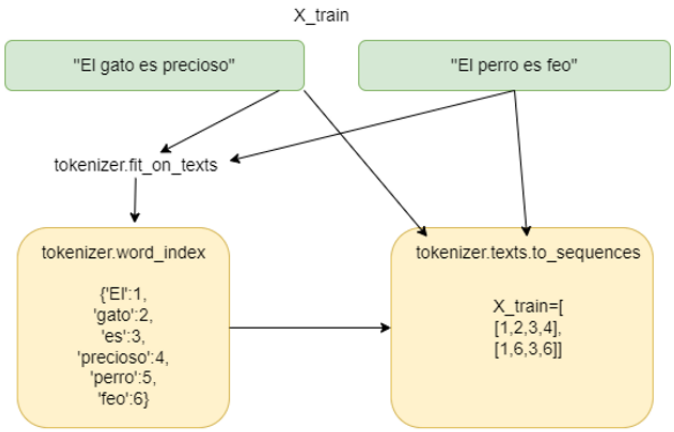

Figura 4.16: Word Embedding

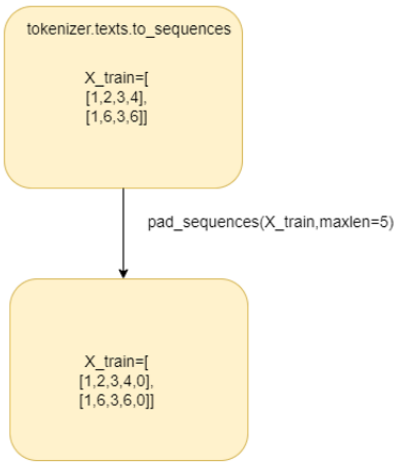

Figura 4.17: Word Embedding

El resultado obtenido tras aplicar esta técnica es una matriz de dos dimensiones donde las filas representan cada una de las palabras tokenizadas que forman los tweets, y las columnas los pesos asignados a cada una de las palabras. Como ya se ha dicho, estos pesos se van ajustado a medida que se va entrenando la red en caso de que no utilizemos ningún mecanismo de palabras incrustadas preentrenadas.

Si utilizamos técnicas de palabras incrustadas preentranadas, cada palabra se va a representar por un vector de n dimensiones que van a ser fijos y dependientes del diccionario que seleccionemos para crear los datos de entrada de la red neuronal.

La idea en mente es en un primer lugar comparar diferentes alternativas de diseño de redes neuronales, es decir comparar diferentes arquitecturas y ver cúal produce mejores resultados para unos parámetros fijos y sin optimizar al máximo. Después nos quedaremos con el mejor modelo e intentaremos mejorarlo hasta donde se pueda.

4.8.2. Red neuronal recurrente LSTM

Como ya se comentó, las redes recurrentes tienen memoria, es decir, guardan información de los textos que ya han procesado, por lo que puede ser muy útil para analizar otros textos futuros. El uso de la capa LSTM mejora esta condición que tienen las redes neuronales recurrentes ya que incorporan la memoria a largo plazo junto con la memoria a corto plazo.

La estructura de esta red está formada por una capa Embedding, una capa LSTM con 90 unidades y una capa Densa formada por 3 unidades que será la que clasifique el sentimiento del tweet.

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 33, 200)	9080400
lstm_2 (LSTM)	(None, 90)	104760
dense_2 (Dense)	(None, 3)	273

Figura 4.18: Modelo LSTM

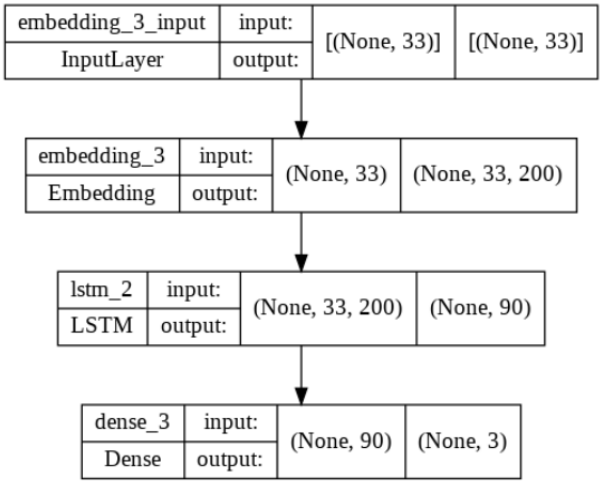

Figura 4.19: Modelo LSTM

4.8.3. Red neuronal recurrente GRU

Existe otro tipo de red neuronal recurrente utilizada para problemas de clasificación de texto, y es mediante el uso de la capa GRU. La principal diferencia que tiene con respecto a la capa LSTM es el coste computacional que tiene, ya que en cuanto a tiempo y recursos es significativamente mejor. Se pierde potencia de poder de representatividad a cambio de un coste computacional menor. Por lo tanto, si estamos trabajando con secuencias de texto excesivamente largas, las redes GRU no son una buena opción.

La estructura de la red está formada por una capa Embedding, una capa GRU con 32 unidades y una capa Densa formada por 3 unidades que será la que clasifique el sentimiento del tweet.

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 33, 200)	9080400
gru_1 (GRU)	(None, 32)	22464
dense_1 (Dense)	(None, 3)	99

Figura 4.20: Modelo GRU

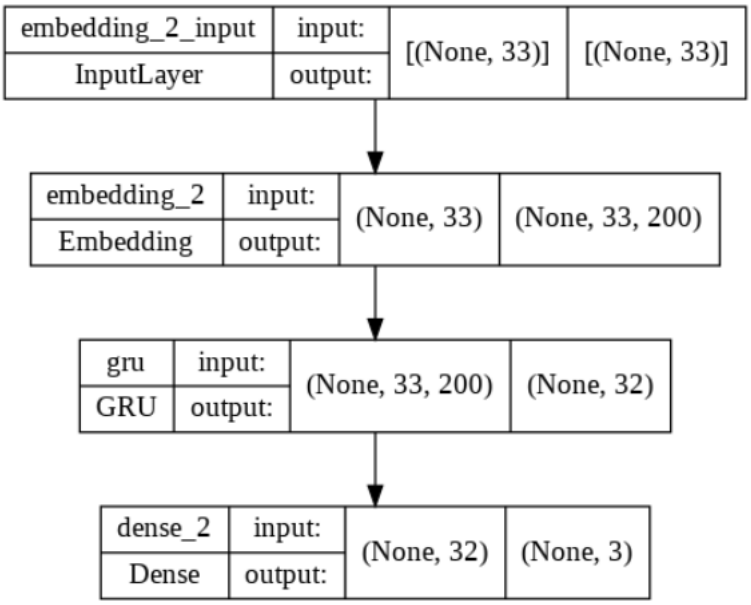

Figura 4.21: Modelo GRU

4.8.4. Red neuronal convolucional 1D

Las redes convoluciones 1D son utilizadas en resolución de problemas de clasificación de texto. Se utilizan de la misma manera que las redes convolucionales 2D, ya que requieren de un conjunto de capas Conv1D y MaxPooling1D terminando en una capa GlobalMaxPooling o Flatten para reducir la dimensionalidad y poder añadir una capa Densa al final que sea la encargada de clasificar los tweets en las distintas clases.

La primera capa es la capa Embedding que como ya se explicó es la encargada de transformar los tweets en numeros para que la red pueda comprenderlos y procesarlos.

La segunda capa es la capa convolucional1D que cuenta con 128 filtros y un tamaño del kernel de 7 y la función de activación relu.

Las siguientes capa son MaxPooling1D y GlobalMaxPooling que se utilizan para reducir la dimensionalidad y poder ser la entrada de la capa densa de clasificación.

En la capa densa tenemos que poner tantas neuronas como clases queremos clasificar en el problema. Como en nuestro caso tenemos 3 tipos de sentimientos, ponemos 3 neuronas.

Layer (type)	Output Shape	Param #
embedding_6 (Embedding)	(None, 33, 200)	9080400
conv1d_6 (Conv1D)	(None, 27, 128)	179328
max_pooling1d_4 (MaxPooling 1D)	(None, 2, 128)	0
dense_3 (Dense)	(None, 2, 3)	387

Figura 4.22: Modelo Conv1D

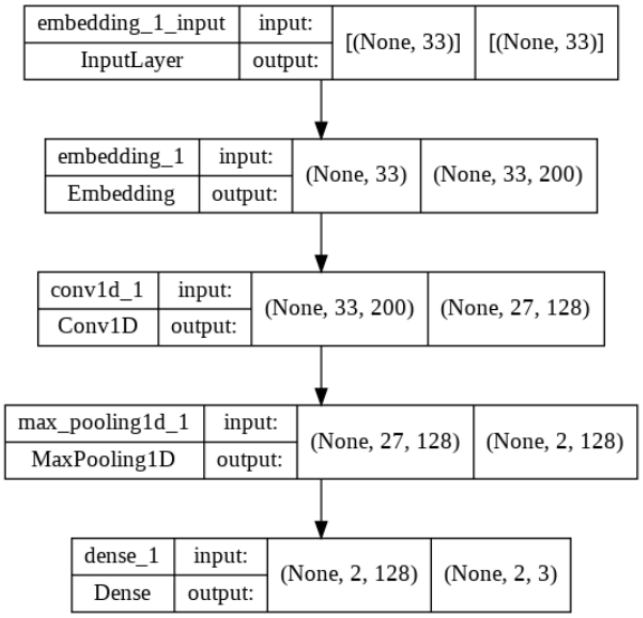

Figura 4.23: Modelo Conv1D

4.8.5. Red neuronal Híbrida

Hemos probado con redes neuronales recurrentes y redes neuronales convolucionales, pero ¿y si las aplicamos simultáneamente?

La idea del modelo híbrido es intentar explotar las ventajas de cada uno de los modelos anteriores, es decir, el objetivo es que la red convolucional elimine trabajo a la red recurrente por lo que esta última podría ser mucho más eficiente.

Una de las principales diferencias entre ambas redes es la importancia del orden en el texto, es decir las redes convolucionales funcionan mejor para resolver problemas en los que el orden en que aparecen los textos no es significante. En cambio, en las redes recurrentes, el orden si es significativo, por lo que a priori se deberían de comportar mejor en nuestro tipo de problema en concreto ya que existe una correlación entre las palabras de un tweet.

Esta idea se muestra en la figura 4.24, y como bien se dice, el principal beneficio es el de ligerar el procesamiento de las redes recurrentes mediante la eliminación de ciertos pasos que se encargan de ello las convoluciones. Por lo tanto, la red convolucional transformará los tensores que representan el tweet en una secuencia de características de alto nivel más representativas. Esta secuencia de características es la que recibirá como entrada la red recurrente.

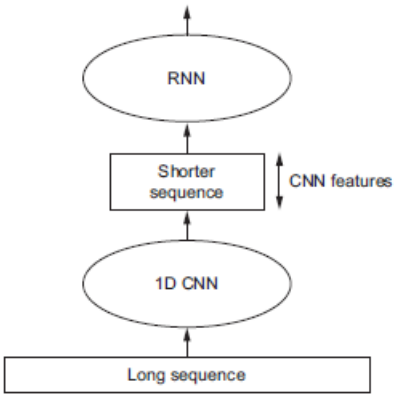

Figura 4.24: Explicación Modelo Híbrido[16]

El modelo que se propone está formado por una capa Embedding, una capa convolucional de 1 dimensión (Conv1D), una capa MaxPooling, una capa LSTM y por último una capa densa que clasifique el tweet correspondiente en las 3 clases.

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 4.25: Modelo Híbrido

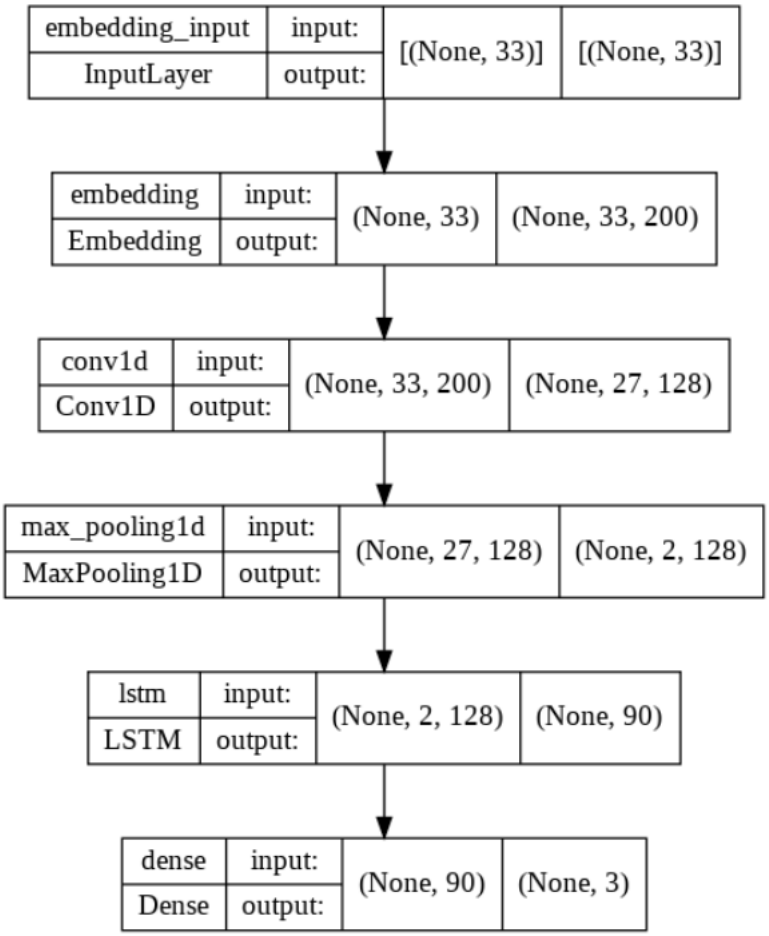

Figura 4.26: Modelo Híbrido

También en el modelo Híbrido se han realizado pruebas utilizando la capa recurrente GRU con los mismos datos que en el modelo GRU ya visto anteriormente.

4.9. Resultados evaluación Modelo Híbrido Básico

Para evaluar los modelos y poder compararlos se ha utilizado la métrica de la exactitud (accuracy), que como se explicó es la más utilizada para obtener el rendimiento de un modelo ya que es el porcentaje de tweets clasificados correctamente.

Es importante diferenciar entre la exactitud de entrenamiento y la exactitud de validación, siendo esta última la que de verdad evalúa nuestro modelo debido a que con la primera estaríamos evaluando el modelo con los datos que ha utilizado la red para aprender.

La idea es elegir de entre los modelos presentados en el apartado anterior, cual de ellos ofrece mejores resultados para poder elegir uno y continuar optimizando y afinando dicho modelo.

Para que todos los modelos estén en las mismas condiciones, se han decidido algunos parámetros

fijos. Estos parámetros elegidos están en el rango de valores típicos en este tipo de estudios, es decir no están elegidos al azar bajo un criterio personal.

- La dimensión de la capa de Embedding es 200.
- La longitud máxima de los tensores de entrada es 33.
- Se ha utilizado un tamaño de vocabulario fijo de 45402 palabras.
- El tamaño de lote (batch size) es de 256.
- Se empleó EarlyStopping para evitar el sobreajuste.

En esta primera fase de las técnicas de reducción de características, únicamente se ha aplicado la eliminación de stopwords, dejando el uso de lematización y stemming para el modelo mejorado.

Estos parámetros son compartidos entre cada uno de los modelos, ahora se va a ir explicando los parámetros concretos de cada modelo y sus respectivos resultados.

Se han utilizado todos los tweets que forman el corpus de las clases P, N y NONE.

4.9.1. Modelo LSTM

Los parámetros que se han seleccionado en la capa LSTM han sido 90 neuronas, una tasa de dropout de 0.2 y una tasa de dropout recurrente de 0.3. Este último valor es recomendable ser usado en modelos recurrentes ya que ayuda a evitar el sobreajuste y su objetivo es el de descartar neuronas por conexiones recurrentes.

La primera columna indica el optimizador utilizado, la segunda es la función de pérdidas, la tercera es una ejecución aleatoria para el mismo set de entrenamiento/validación y la cuarta es una validación cruzada de k=10 para tener una mejor exactitud de los modelos. El valor es la media de las 10 respectivas ejecuciones.

Optimizador	Pérdidas	Val. Acuraccy	Cross Validation
`Adam`	`binary_crossentropy`	**0.7260**	**0.7277**
`Adam`	`categorical_crossentropy`	**0.7330**	**0.7297**
`Rmsprop`	`binary_crossentropy`	**0.7181**	**0.7157**
`Rmsprop`	`categorical_crossentropy`	**0.7212**	**0.7154**

Cuadro 4.1: Resumen de pruebas realizadas para evaluar el modelo LSTM.

Como se puede observar, los mejores resultados se obtienen con la combinación del optimizador Adam y la función de pérdidas categorical_crossentropy.

4.9.2. Modelo GRU

Los parámetros seleccionados en la capa GRU han sido 32 neuronas, tasa de dropout de 0.2 y tasa de dropout recurrente de 0.3.

La primera columna indica el optimizador utilizado, la segunda es la función de pérdidas, la tercera es una ejecución aleatorio para el mismo set de entrenamiento/validación y la cuarta es una validación cruzada de k=10 para tener una mejor exactitud de los modelos.

Optimizador	Pérdidas	Val. Acuraccy	Cross Validation
Adam	binary_crossentropy	**0.3704**	–
Adam	categorical_crossentropy	**0.3704**	–
Rmsprop	binary_crossentropy	**0.3704**	–
Rmsprop	categorical_crossentropy	**0.5138**	–

Cuadro 4.2: Resumen de pruebas realizadas para evaluar el modelo GRU.

Como podeis observar no se ha calculado los valores de la última columna, ya que es obvio que no van a mejorar los valores de LSTM. Es curioso que de el valor 0.3704 y una de las conclusiones a las que he llegado es debido a que la red en los parámetros seleccionados no ha tenido suficiente tiempo para converger y poder obtener buenos resultados.

4.9.3. Modelo Conv1D

Los parámetros seleccionados en la capa Conv1D han sido 128 neuronas, 7 filtros y la función activación relu. En la capa MaxPooling1D el tamaño del pool es de 10.

La primera columna indica el optimizador utilizado, la segunda es la función de pérdidas, la tercera es una ejecución aleatorio para el mismo set de entrenamiento/validación y la cuarta es una validación cruzada de k=10 para tener una mejor exactitud de los modelos. El valor es la media de las 10 respectivas ejecuciones.

Optimizador	Pérdidas	Val. Acuraccy	Cross Validation
Adam	binary_crossentropy	**0.7297**	**0.7301**
Adam	categorical_crossentropy	**0.7304**	**0.7301**
Rmsprop	binary_crossentropy	**0.7257**	**0.7293**
Rmsprop	categorical_crossentropy	**0.7235**	**0.7285**

Cuadro 4.3: Resumen de pruebas realizadas para evaluar el modelo Convolucional.

Como se puede observar son unos resultados bastante buenos y se observa que como sucede con el modelo LSTM, funciona mejor con el optimizador adam y la función de pérdidas categorical_crossentropy.

4.9.4. Modelo Híbrido

En el modelo híbrido se han utilizado los mismos parámetros utilizados en los modelos que implementa para poder tener una referencia y saber si funciona mejor que cada uno de los modelos por separado. Es el mecanismo que se ha supuesto para poder valorar el utilizar el modelo híbrido o no, es decir saber si la combinación de dichos modelos mejora los resultados.

La primera columna indica el optimizador utilizado, la segunda es la función de pérdidas, la tercera es una ejecución aleatorio para el mismo set de entrenamiento/validación y la cuarta es una validación cruzada de k=10 para tener una mejor exactitud de los modelos. El valor es la media de las 10 respectivas ejecuciones.

Se ha probado utilizando primero capa convolucional + capa LSTM y después utilizando capa convolucional + capa GRU.

Optimizador	Pérdidas	Val. Acuraccy	Cross Validation
Adam	binary_crossentropy	**0.7343**	**0.7304**
Adam	categorical_crossentropy	**0.7355**	**0.7314**
Rmsprop	binary_crossentropy	**0.7178**	**0.7157**
Rmsprop	categorical_crossentropy	**0.7298**	**0.7290**

Cuadro 4.4: Resumen de pruebas realizadas para evaluar el modelo Híbrido con LSTM.

Como se puede observar son los mejores resultados obtenidos hasta el momento, más concretamente con adam y categorical_crossentropy.

Optimizador	Pérdidas	Val. Acuraccy	Cross Validation
Adam	binary_crossentropy	**0.7350**	**0.7331**
Adam	categorical_crossentropy	**0.7355**	**0.7314**
Rmsprop	binary_crossentropy	**0.7350**	**0.7310**
Rmsprop	categorical_crossentropy	**0.7242**	**0.7308**

Cuadro 4.5: Resumen de pruebas realizadas para evaluar el modelo Híbrido con GRU.

A diferencia del modelo LSTM, los valores obtenidos con el optimizador rmsprop son bastante mejores que en el modelo anterior, aunque como vemos con adam y categorical_crossentropy obtenemos los mismos resultados.

4.9.5. Comparativa de resultados

En la siguiente tabla se muestran los resultados de los diferentes modelos en los que se han realizado las pruebas.

Modelo	Optimizador	Pérdidas	Cross Validation
Conv1D	Adam	categorical_crossentropy	**0.7304**
GRU	Rmsprop	categorical_crossentropy	**0.5138**
LSTM	Adam	categorical_crossentropy	**0.7297**
Híbrido con LSTM	Adam	categorical_crossentropy	**0.7314**
Híbrido con GRU	Adam	categorical_crossentropy	**0.7314**

Cuadro 4.6: Resumen de las mejores pruebas realizadas por cada uno de los modelos para seleccionar el mejor modelo.

Como se puede observar, el optimizador adam y la función de pérdidas categorical_crossentropy, nos ofrecen los mejores resultados utilizando los modelos híbridos.

La pregunta es: ¿Y ahora con que modelo híbrido nos quedamos?

Para poder responder esta pregunta se ha realizado una comparativa a mayores. Si recordamos los modelos GRU funcionan mejor con textos muy pequeños y los modelos LSTM se adaptan mejor a modelos cuyos textos son más largos.

El parámetro que controlaba esto estaba en 33, pues bien en esta prueba se ha realizado utilizando 200 para poder observar su comportamiento ya que no nos interesa que solo funcione bien en situaciones muy concretas.

El resto de parámetros se mantiene igual y para demostrar este caso no se ha utilizado validación cruzada, únicamente validación manual, ya que queremos evaluar el comportamiento del modelo para poder decidirnos.

Optimizador	Pérdidas	Val. Acuraccy
Adam	binary_crossentropy	**0.6003**
Adam	categorical_crossentropy	**0.5530**
Rmsprop	binary_crossentropy	**0.6979**
Rmsprop	categorical_crossentropy	**0.6463**

Cuadro 4.7: Resumen de pruebas realizadas para evaluar el modelo Híbrido con GRU con longitud 200.

Optimizador	Pérdidas	Val. Acuraccy
Adam	binary_crossentropy	**0.7318**
Adam	categorical_crossentropy	**0.7388**
Rmsprop	binary_crossentropy	**0.7231**
Rmsprop	categorical_crossentropy	**0.7168**

Cuadro 4.8: Resumen de pruebas realizadas para evaluar el modelo Híbrido con LSTM con longitud 200.

Como se puede apreciar en las dos tablas, con mayores longitudes de entrada el uso de GRU ofrece peores resultados, por lo que nos vamos a decantar para seguir optimizando, el modelo híbrido convolucional + LSTM ya que es el que mejores resultados nos ha ofrecido en distintas situaciones.

4.10. Resultados evaluación Modelo Híbrido Mejorado

Partimos del modelo híbrido con capa LSTM con una exactitud obtenida de 0.7314. El objetivo es intentar mejorar el modelo para intentar obtener mejores resultados de precisión.

4.10.1. Evaluación del Modelo usando stemming

El stemming es una técnica de reducción de características que como ya se explicó anteriormente se aplica en problemas de clasificación de textos para reducir el ruido de los tweets en este caso.

Se ha utilizado SnowballStemmer en español para aplicar stemming ya que es el mejor y el más utilizado en este tipo de problemas.

El resultado de cross accuracy obtenido aplicando stemming ha sido de `0.7390%`.

4.10.2. Evaluación del Modelo usando lemmatización

La lematización es otra técnica de reducción de características que es similar al stemming ya que tiene el mismo objetivo para intentar dar mejores resultados.

Se han utilizado dos lematizadores en español: el ofrecido por la librería de spacy, y el de la librería de stanza.

Los resultados obtenidos aplicando la técnica de lematización son los siguientes:

- Con spacy obtenemos una cross accuracy de `0.7360%`.

- Con stanza obtenemos una cross accuracy de `0.7359%`.

Los resultados ofrecidos por ambos lematizadores no mejoran el resultado obtenido utilizando stemming, además hay que decir que utilizando stanza, el tiempo de procesamiento para crear los términos es muy superior con respecto a utilizar spacy o el stemming visto anteriormente.

4.10.3. Evaluación del modelo con ajuste óptimo de hiperparámetros

Se ha realizado un ajuste de las neuronas y del tamaño del kernel de la capa convolucional y de las neuronas, tasa de dropout y tasa de dropout recurrente de la capa LSTM.

Para evaluar los resultados obtenidos se ha seguido el siguiente procedimiento, se han realizado distintas pruebas y nos hemos quedado con las 10 mejores ejecuciones para simplificar el número de pruebas mostradas.

La elección del rango de valores de prueba viene determinada a través de los distintos estudios e informes que se han ido consultando para abordar dicho trabajo.

NeuronasC	NeuronasR	Dropout	R_dropout	T.Kernel	Cross Validation
192	96	0.2	0.3	8	**0.7413**
160	128	0.2	0.2	8	**0.7410**
192	128	0.2	0.4	3	**0.7403**
128	96	0.4	0.4	5	**0.7399**
160	64	0.3	0.2	8	**0.7390**
192	64	0.3	0.3	3	**0.7381**
128	32	0.4	0.2	5	**0.7377**
96	64	0.3	0.2	3	**0.7366**
64	32	0.3	0.3	8	**0.7361**

Cuadro 4.9: Resumen de pruebas realizadas para evaluar los hiperparámetros del número de neuronas de la capa convolucional y de la capa LSTM, el tamaño del kernel de la capa convolucional, y las tasas de dropout de la capa LSTM.

He de decir que ya se partía de un buen primer modelo en el prácticamente estos parámetros estaban bastante ajustados y las diferencias obtenidas son mínimas y también dependen de las ejecuciones realizadas.

Por tanto los mejores valores de los hiperparámetros tras las distintas pruebas son los siguientes:

- 192 neuronas en la capa Conv1D.
- Tamaño de kernel de 8.
- 96 neuronas en la capa LSTM.

- Tasa de 0.2 de dropout en la capa LSTM.
- Tasa de 0.3 de dropout recurrente en la capa LSTM.

4.10.4. Evaluación del usando incrustaciones preentrenadas

Como se comentó, existen dos maneras de cargar las incrustaciones en el proceso de creación de una red neuronal. La primera de ellas es la utilizada y consiste si os acordáis en que las incrustaciones se aprendan a medida que se entrena la red con información del propio corpus. La otra opción, que es la empleada en este apartado, consiste en utilizar incrustaciones de palabras preentrenadas, es decir en lugar de que los pesos se ajusten con el corpus, se ajusten con un diccionario definido que por lo general se ha creado para solventar este tipo de problemas.

Los resultados obtenidos utilizando diccionarios de palabras incrustradas preentrenadas son los siguientes:

- Utilizando GloVe se ha obtenido una cross accuracy de `0.6655%.`
- Utilizando word2vec se ha obtenido una cross accuracy de `0.6779%.`

Como se observa, los resultados empeoran bastante y la conclusión a la que he llegado es básicamente que al ser algo creado de forma general y que se puede aplicar en otros corpus, depende en gran medida de la relación que haya en nuestro corpus con las palabras que estén en el diccionario, por lo que siempre va a ser mejor aprender directamente de algo más específico como es nuestro corpus.

4.10.5. Evaluación del modelo usando capa Bidirectional

Como ya sabemos, las redes recurrentes procesan la información en orden, es decir son dependientes del orden en el que aparecen las palabras de los tweets, pero si cambiamos la manera en la que se procesan las entradas podemos hacer que se detecten patrones que una red unidireccional no hubiera encontrado.

Una red neuronal recurrente bidireccional es una variante de las redes neuronales recurrentes y por lo general suelen ofrecer mejores resultados que en su versión unidireccional, aunque no siempre funcionan mejor.

A diferencia de las redes unidireccionales, las redes bidireccionales procesan la información en una ambas direcciones, es decir hacia atrás (del futuro al pasado) y hacia delante (del pasado al futuro). Esto puede llevar a que la red aprenda patrones y extraiga características que utilizando una red unidireccional se perderían y podrían ser útiles para el modelo.

La idea básicamente es tener dos capas, cada una va en un sentido de procesamiento, y que fusionan las características extraídas por cada una de ellas.

En la siguiente figura se puede observar de manera gráfica la idea que abarca las redes en este caso LSTM bidireccionales.

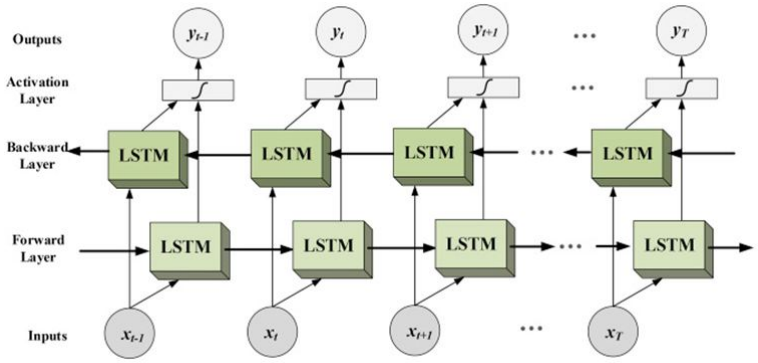

Figura 4.27: Explicación Bidireccional[42]

Se ha realizado una prueba de validación cruzada para evaluar si esta técnica mejora la precisión. El cross accuracy obtenido es de `0.7384%`, por lo que utilizando esta técnica, no mejora los resultados obtenidos.

4.10.6. Evaluacion del modelo aplicando el concepto de Information Gain

La ganancia de información (IG)[43] de una palabra de un tweet mide el número de bits de información obtenidos para la predicción de una clase (c) al conocer la presencia o ausencia de un término (t) en un tweet. Es una medida que resume como de común es una palabra en una clase comparada como de común es en las otras clases.

Para entender que es la ganancia de información vamos a suponer que tenemos dos palabras : palabra1 y palabra2. Si palabra1 tiene un valor de IG más grande que palabra2, palabra1 será más útil en nuestro entrenamiento de la red neuronal debido a que aumenta la información y reduce la incertidumbre.

Para calcular la IG de una palabra, primero necesitamos calcular la entropia[44], cuya fórmula es la siguiente:

$$H = \sum_{i=1}^{n} p_i * log_2(p_i)$$

donde la n es el número de clases posibles, p su i es la probabilidad de que la palabra w su i esté o no esté en la clase c su i.

Después, para calcular la ganancia de información(IG), la fórmula es la siguiente:

$$IG(C, X) = H(C) - H(C, X)$$

donde C es el conjunto de clases, X el subconjunto de textos donde aparece la palabra w su i.

Para conocer la H(C), primero necesitamos calcular las probabilidades de cada clase dentro del corpus. Para conocer la H(C,X), necesitamos calcular las probabilidades de la palabra de aparecer y de no aparecer en el corpus, así como las probabilidades de aparecer y de no aparecer en cada una de las clases correspondientes.

Se ha aplicado este concepto de IG para quedarnos con las mejores palabras del vocabulario para ver como funciona este concepto y comprobar si mejoran los resultados del clasificador.

Tamaño Corpus	Tamaño Vocab	Cross Validation
Corpus original	20000	**0.7482**
Corpus original	15000	**0.7468**
Corpus original	10000	**0.7486**
Corpus original	5000	**0.7477**

Cuadro 4.10: Resumen de pruebas realizadas para evaluar el modelo Híbrido aplicando IG.

Como se aprecia, si mejora la exactitud del modelo y el mejor resultado obtenido es quedándonos con un tamaño de vocabulario de 10000.

4.10.7. Evaluación del Modelo aplicando Reducción de tamaño del corpus + IG

Como ya se ha dicho la elección del corpus es algo crucial para que nuestro modelo obtenga mejores o peores resultados. Las pruebas que se han hecho en este apartado han sido eliminando los últimos 12000 tweets quedándonos con una distribución del corpus con 22021 tweets positivos, 15748 tweets negativos y 21094 tweets sin polaridad. Vamos a aplicar esta idea junto con la anterior a ver si mejora con respecto a los resultados obtenidos.

Tamaño Corpus	Tamaño Vocab	Cross Validation
Quitando últimos 12000	20000	**0.7558**
Quitando últimos 12000	15000	**0.7570**
Quitando últimos 12000	10000	**0.7582**
Quitando últimos 12000	5000	**0.7570**

Cuadro 4.11: Resumen de pruebas realizadas para evaluar el modelo Híbrido aplicando reducción de corpus e IG.

Esta mejoría en los resultados se puede deber al hecho de que los tweets eliminados estaban perjudicando al modelo, ya que como hemos mencionado, se podía dar el caso de que no aportasen información apenas y estuvieran aumentando la incertidumbre por lo que el modelo obtendría peores resultados.

Se han realizado pruebas eliminando los últimos 4000 tweets y los últimos 22000 tweets con tamaño de vocabulario 10000 y los resultados se pueden observar en la siguiente tabla:

Tamaño Corpus	Tamaño Vocab	Cross Validation
Quitando últimos 4000	Vocab 10000	**0.7497**
Quitando últimos 22000	Vocab 10000	**0.7510**

Cuadro 4.12: Resumen de pruebas realizadas para evaluar el modelo Híbrido aplicando reducción de corpus e IG.

Como se observa, hemos realizado una prueba seleccionando un valor por encima y por debajo del umbral de 12000 y no obtenemos mejores resultados.

4.10.8. Evaluación del modelo analizando Emb Dim, maxlen y Batch Size

Se han realizado pruebas para evaluar los hiperparámetros del embedding dim y el maxlen, que indican el tamaño de entrada de la matrix de embedding, indicando el primero de ellos las columnas y el segundo de ellos las filas. Vamos a estudiar como afectan dichos parámetros. Los valores empleados son valores típicos en este tipo de problemas.

Emb Dim	maxlen	Cross Validation
100	22	**0.7532**
100	33	**0.7570**
100	44	**0.7545**
200	22	**0.7519**
200	33	**0.7582**
200	44	**0.7560**
300	22	**0.7522**
300	33	**0.7567**
300	44	**0.7552**

Cuadro 4.13: Resumen de pruebas realizadas para evaluar el modelo Híbrido modificando embedding dim y maxlen.

Como se observa, el mejor resultado es utilizando los valores 200 para embedding dim y 33 para

maxlen.

El batch size (BS)[10] indica la cantidad de datos que el modelo tiene en cada iteración que realiza. Vamos a estudiar como afecta la modificación de este valor. Los valores propuestos son valores típicos que se utilizan como estándares.

Batch Size	Tiempo	Cross Validation
32	14min	**0.7538**
64	7min	**0.7510**
128	6min	**0.7578**
256	3min	**0.7582**
512	2min	**0.7562**

Cuadro 4.14: Resumen de pruebas realizadas para evaluar el modelo Híbrido modificando el Batch size.

Como se observa, no existen grandes diferencias en cuanto a resultados de exactitud, sin embargo en tiempos de ejecución vemos como una relación evidente de que a mayor número, menor tiempo de ejecución y esto es debido al hecho de que en cada iteración el modelo tiene más datos por lo que tarda menos en procesar la información.

4.10.9. Evaluación del modelo balanceando el corpus de aprendizaje

Otra cosa a tener en cuenta, es el concepto ya mencionado de balanceo del corpus, pues bien cuanto más balanceado esté el corpus, mejores resultados vamos a obtener ya que el modelo puede aprender mejor a clasificar los tweets.

Se han hecho dos pruebas, ambas utilizando cross validation para su evaluación. En la primera de ellas se han seleccionado 20000 tweets de cada clase, es decir 20000 tweets positivos, 20000 tweets negativos y 20000 tweets nones. Se ha aplicado reducción del tamaño de vocabulario utilizando IG quedándonos con las 10000 mejores palabras del vocabulario y con el corpus original. El resultado de esta prueba ha sido de `0.7554%`.

La segunda prueba realizada ha sido eliminando del corpus los últimos 12000 tweets, y quedándonos con 15000 positivos, 15000 negativos y 15000 nones. Se ha aplicado reducción del tamaño de vocabulario utilizando IG quedándonos con las 10000 mejores palabras del vocabulario y el resultado ha sido de `0.7562%`.

Por lo que se observa, no conseguimos mejorar el mejor resultado obtenido. Podemos llegar a la conclusión de que es más importante tener más información adecuada lo que se traduce en tener más tweets que aporten más información al modelo, que tener menos tweets y que estén

balanceadas las clases.

4.11. Comparativa antes y después de las mejoras

En las figuras 4.28, y 4.29, se observa los valores de otras métricas como son la precisión, el recall y el f1-score que ya se explicaron anteriormente. El 0 representa la clase negativa, el 1 la clase none y el 2 la clase positiva. Los valores de estas métricas son por cada clase y como se puede observar se ha conseguido mejorar entorno al 3 % de exactitud.

Otros factores a tener en cuenta ha sido la precisión obtenida de cada clase, por ejemplo la clases P y N han mejorado de precisión un 3 %, y la clase NONE ha mejorado un 2 %.

En la métrica recall (exhaustividad) como observamos, la clase que más ha mejorado ha sido la NONE con un 7 %. Esto es debido a que el algoritmo estaba dudando mucho en situaciones en las que un tweet pertenecía a esta clase y el modelo no lo consideraba así. El valor del f1-score era una combinación de ciertos parámetros, la métrica precision y la métrica recall y como se observa, la clase NONE ha mejorado también un 7 %.

	precision	recall	f1-score	support
0	0.71	0.77	0.74	6094
1	0.69	0.64	0.66	6918
2	0.79	0.78	0.79	7655
accuracy			0.73	20667
macro avg	0.73	0.73	0.73	20667
weighted avg	0.73	0.73	0.73	20667

Figura 4.28: Valores de métricas antes de las mejoras

	precision	recall	f1-score	support
0	0.74	0.77	0.75	4677
1	0.71	0.71	0.71	6344
2	0.82	0.79	0.80	6638
accuracy			0.76	17659
macro avg	0.75	0.76	0.76	17659
weighted avg	0.76	0.76	0.76	17659

Figura 4.29: Valores de métricas después de las mejoras

La matriz de confusión[45] es una herramienta que nos sirve para ver los resultados del modelo y mostrar cuándo una clase es confundida con otra. Hay que tener en cuenta una cosa, una de las mejoras propuestas y que más éxito ha tenido, ha sido eliminar ciertos tweets del corpus, por lo que el total de tweets no es el mismo en las dos matrices correspondientes.

En las figuras 4.30, 4.31, se observan ambas matrices. La diagonal representa los valores bien clasificados por cada clase, las columnas representan los valores predichos y las filas los valores reales. Es decir, las columas nos dicen los valores que se han predicho por cada clase, siendo verdaderos o falsos positivos, y las filas son valores que son de la clase e incluyen aquellos que no se han predicho así.

Lo más destacable a concluir es que entre las clases positivo y negativo se equivoca muy poco, es decir hay muy pocos tweets que son positivos y el algoritmo los clasifica como negativos o viceversa. Sin embargo con las clase NONE observamos que hay cierta confusión ya que el modelo es donde más falla.

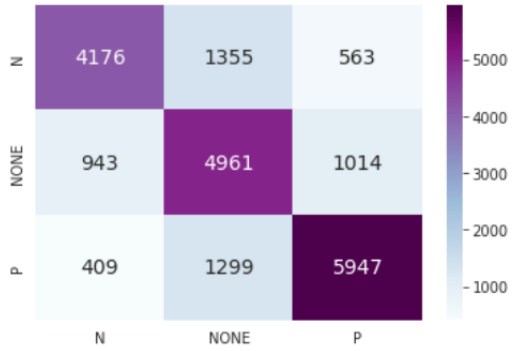

Figura 4.30: Matriz de confusión antes de realizar las mejoras

Ambas matrices presentan el mismo problema con la clase NONE, pero aplicando las mejoras, vemos que se ha mejorado en este aspecto.

Por ejemplo había 1355 tweets que el modelo los había predicho como NONE y eran negativos, y había 1014 tweets predichos como positivos y que eran NONE. Estos dos valores como se ve, son los que más se han visto reducidos, se ha pasado de 1355 a 800 y de 1014 a 755.

Otra cuestión a comentar es el hecho de que ha mejorado en estos dos casos, pero en el caso de tweets predichos como NONE y que eran positivos, y tweets predichos como negativos y que eran NONE, no ha mejorado apenas nada.

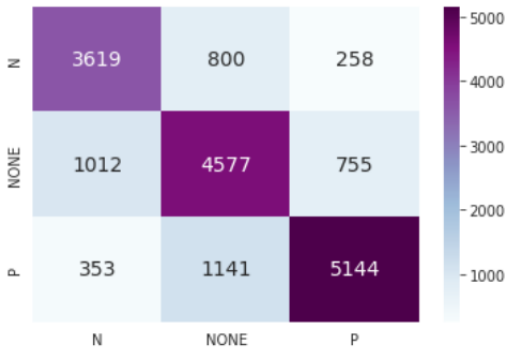

Figura 4.31: Matriz de confusión después de realizar las mejoras

4.12. Evaluación del modelo utilizando 2 clases

Veamos los resultados que obtenemos si en lugar de tener 3 clases (P, N y NONE), tenemos solo dos clases (P y N). Estos resultados realmente son los más importantes debido a que si un tweet no tiene polaridad, realmente no aporta mucha información.

En las pruebas que se han realizado, los únicos parámetros que se han modificado, han sido el tamaño del corpus y el tamaño del vocabulario. Para el resto de parámetros son los valores con los que se han obtenido los mejores resultados.

Tamaño Corpus	Tamaño Vocab	Cross Validation
Corpus original	Vocab original	**0.8829**
Corpus original	Vocab 10000	**0.8932**
Quitando últimos 12000	Vocab original	**0.8956**
Quitando últimos 12000	Vocab 10000	**0.9070**

Cuadro 4.15: Resumen de pruebas realizadas para evaluar el modelo Híbrido utilizando dos clases: P y N.

Como se observa, se obtiene casi un 91 % de exactitud utilizando dos clases.

En la figura 4.32, se puede observar los valores de las métricas precision, recall y f1-score de las dos clases siendo 0 la clase negativa y 1 la clase positiva.

	precision	recall	f1-score	support
0	0.90	0.88	0.89	4713
1	0.91	0.93	0.92	6618
accuracy			0.91	11331
macro avg	0.91	0.90	0.90	11331
weighted avg	0.91	0.91	0.91	11331

Figura 4.32: Valores de las métricas utilizando dos clases P y N

4.13. Resumen de resultados

Partíamos de un modelo básico en el que se había empleado las técnicas bag of words y TF-IDF para la extración de características y ponderación de pesos. Se usó una arquitectura DNN, es decir redes densas conectadas. La exactitud que se obtuvo para 3 clases fue de 0.7006 %.

Aplicando técnicas de Deep Learning se consiguió mejorar el modelo hasta un valor de exactitud de 0.7314 % utilizando el modelo Híbrido.

Aplicando técnicas de mejora sobre el modelo anterior se ha conseguido alcanzar un valor de exactitud de 0.7582 %.

Utilizando 2 clases se ha obtenido un valor de exactitud de 0.9070 %.

4.14. Esquema informativo

En la figura 4.33, se puede observar un esquema del uso de librerías propias de python para desarrollar cada uno de los módulos en los que como se ve, se ha dividio el trabajo.

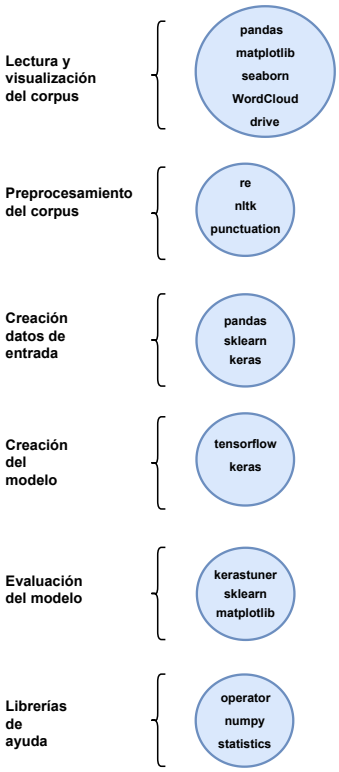

Figura 4.33: Librerías utilizadas para el desarrollo del TFG

Capítulo 5

Conclusiones

5.1. Dificultades del análisis de sentimientos

En esta sección me gustaría hacer referencia a algunas de las dificultades que presenta el desarrollar un clasificador de textos en función del análisis de los sentimientos.

En primer lugar hay que destacar el simple hecho de que es imposible obtener una precisión del 100 %, y esto en parte es debido a que ni los propios seres humanos podemos saber en ciertas situaciones el sentimiento que un tweet transmite.

El significado de una palabra, sobre todo de los adjetivos [46], depende mucho del contexto en el que esté la frase. Por ejemplo el adjetivo perfecto suena a algo positivo, pero en esta frase: "Perfecto mensaje a favor del machismo, os habéis lucido", el adjetivo perfecto claramente está haciendo referencia a algo negativo, por lo que el uso de adjetivos depende en gran parte del contexto de la frase, algo que es muy complicado de resolver.

Otro problema son los sarcamos y las ironías, si ya a los humanos nos cuesta entender en ocasiones cuando hay un sarcasmo en una oración, a un clasificador lo resulta ciertamente difícil.

Otra dificultad es el hecho de detectar las opiniones explicitas e implícitas[47]. Por opinión explicita se entiende una afirmación subjetiva que da una opinión simple o comparativa utilizando una o varias palabras que indican esa opinión. Una opinión implícita es aquella que aparece entre líneas, es decir, únicamente el lector es capaz de detectar el verdadero significado del tweet, y hace muy difícil a un clasificador detectar este tipo de problemas. Por ejemplo la oración "Me compré unos calcetines y ya tienen un tomate", claramente tiene un significado negativo, aunque para la máquina es complicado detectar dicha connotación.

5.2. Aplicaciones análisis de sentimientos

Como algo interesante me gustaría hacer un hincapié al trasfondo del TFG, es decir ¿Qué utilidades puede tener el análisis de sentimientos de Twitter en el mundo real?

Las principales aplicaciones que esto conlleva están relacionadas con monitorizar a tiempo real el sentimiento de los mensajes para tener más control sobre en este caso mensajes de Twitter.

La primera aplicación que se nos viene a la cabeza es intentar replicar las encuestas de opinión de una manera más inmediata y con un menor coste. El problema de esto es el hecho de que en las encuestas se eligen muestras que representan a la población, cosa que en Twitter es más complicado de hacer. Por eso el análisis de sentimientos muestra de una manera rápida y sencilla las tendencias que se siguen en un determinado aspecto.

Tenemos que tener en cuenta que para analizar una tendencia tenemos que filtrar los tweets para por un lado quedarnos con la información adecuada, y por otro eliminar los tweets que sean de spam, es decir que interfieran a propósito en la tendencia objetivo manipulando los resultados obtenidos.

Algunas de las aplicaciones son [48] :

- Aplicaciones de negocio: La reputación en Internet es una de las cosas más preciadas para una marca. Las malas reseñas pueden causar una mala imagen de la marca así como tener consecuencias económicas ya que hoy en día todo lo que leemos en Internet es más creíble. Poder ver a tiempo tendencias negativas acerca de tu empresa o de tu producto puede salvarte y evitar que unas simples reseñas negativas causen mayores consecuencias.
  También te permite saber que aspectos tanto positivos como negativos tiene tu negocio para así poder mejorar. Por ejemplo si tenemos una empresa de compra-venta de coches que se llama VentaCoches, en primer lugar tendríamos que filtrar tweets en donde se mencione a la cuenta, se utilizen hashtags creados por nosotros, o por búsqueda de determinadas palabras que sabemos con cierta precisión que se refieren a nuestra empresa.
  Ahora tendríamos que definir un objetivo de estudio, si por ejemplo queremos saber si las garantías que ofrecemos son de calidad, obtendríamos todos los tweets filtrados con la palabra garantía, que mencionen a nuestra cuenta etc y después realizaríamos la predicción de dichos tweets para obtener cuantos son positivos y cuantos son negativos.

- Aplicaciones de economía: Se ha demostrado que es posible predecir comportamientos de la bolsa en función a la polaridad de los tweets que contengan información relacionada con el activo en cuestión. Los cambios de los valores de un activo pueden estar afectados directamente por la actividad de ciertos usuarios, por lo que si monitorizamos a determinados usuarios, podremos predecir de cierto modo la evolución o el comportamiento de un activo. Un ejemplo de esto podría ser analizar los tweets de Elon Musk, ya que en el mundo de las

criptomonedas tiene mucha influencia y que por ejemplo si los tweets sobre una criptomoneda son positivos, se ha demostrado que el valor de la criptomoneda ha aumentado.

Otra aplicación posible sería relacionada con predecir tendencias como por ejemplo el precio de la gasolina, ya que si analizamos durante un cierto período tweets publicados por el Gobierno o por determinadas personas, se podría analizar los cambios de sentimientos para analizar y predecir la tendencia y cambios del precio.

- Aplicaciones de política: En Twitter, la política es uno de los temas de conversación más recurrentes. Con el análisis de sentimientos podemos analizar las tendencias que tienen alrededor de los políticos que se presentan a unas elecciones, para poder observar la opinión acerca de ellos y poder realizar estadísticas y predicciones sobre el resultado de las elecciones.

  Por ejemplo durante las elecciones de 2016 en US entre Donald Trump y Hillary Clinton se realizó un análisis de los tweets en los que se mencionaban a los dos candidatos y se puede apreciar en las siguientes figuras.

Figura 5.1: Análisis tweets Donald Trump[48]

Figura 5.2: Análisis tweets Hillary Clinton[48]

Como se puede apreciar, el candidato Donald Trump fue mucho más mediático que Hillary Clinton, lo que seguramente fue una de las principales razones por las que ganase dichas elecciones.

También se puede aplicar esto durante el mandato de un gobierno y analizar como de contentos/enfadados están los ciudadanos con las diferentes decisiones que se han tomado y desde el propio gobierno aplicar soluciones de una manera rápida y efectiva. Por ejemplo si buscamos tweets con palabras que contengan "PSOE.º "Pedro Sánchez", podríamos detectar el pensamiento que tienen los ciudadanos acerca del gobierno actual, y esto también servir al propio gobierno a tener contenta a la opinión pública.

Otra aplicación interesante es poder analizar el voto indeciso, es decir como evoluciona el voto de una persona a lo largo de una campaña electoral y poder ver que decisiones han hecho su acercamiento a un determinado partido político. Por ejemplo si tomamos la lista de tweets de un usuario y vemos que filtrando por el nombre de los partidos políticos no hay ninguno que se diferencie del resto en cuanto a valoraciones positivas, podríamos monitorizar una lista de usuarios en una situación similar y ver como evolucionan sus valoraciones a los partidos a lo largo de la campaña analizando que decisiones de campaña han encajado mejor en su forma de pensar.

5.3. Conclusiones

El Deep Learning es el futuro del procesamiento de los datos debido a su reciente y rápido crecimiento en los últimos años. Me ha resultado de gran interés poder investigar acerca de este sector ya que no había oído hablar de ello anteriormente, y debido a mi mención escogida, de IA no he podido apenas estudiar nada. Es un campo que tiene un recorrido muy largo debido al reciente crecimiento e interés acerca de la obtención y el análisis de los datos, ya que si conseguimos que las máquinas se acerquen al pensamiento de los seres humanos, estaríamos hablando de un avance tecnológico importante.

Me gustaría resaltar la satisfacción del trabajo realizado ya que se ha alcanzado el objetivo claro de lograr un modelo con una precisión de al menos un 75 % utilizando tres clases. Se ha conseguido realizar un trabajo exploratorio de diversas opciones para implementar redes neuronales para así poder identificar los mejores modelos.

También se ha conseguido obtener una precisión superior al 90 % cuando el modelo disponía únicamente de dos clases: positiva y negativa.

Por otro lado, como se ha dicho tener un algoritmo que obtenga un 100 % de precisión es

imposible, debido a que ni siquiera un humano es capaz de poder clasificar correctamente todos los tweets, por lo que a partir de un 70 % ya se considera un algoritmo decente, según todos los estudios que he podido observar, por lo que los resultados son bastante satisfactorios.

Además, cabe resaltar el hecho de que el modelo elegido híbrido no ha sido propuesto en trabajos similares en castellano y relacionados con las redes sociales, por lo que resulta de carácter innovador.

5.4. Lineas futuras

Como posibles líneas futuras se plantean las siguientes:

- Analizar posibles mejoras de preprocesamiento del texto.
- Investigar sobre posibles soluciones a la ironía, el sarcasmo y la detección de posibles opiniones spam que únicamente sirven para desvirtuar la opinión general.
- Aumentar el tamaño del corpus con tweets que aporten información e investigar sobre obtener un corpus lo más adecuado posible para la tarea del análisis de sentimientos, ya que como se ha dicho, el corpus es la base de obtener buenos resultados en el modelo.
- Como se ha visto, la clase NONE es la clase que peor precisión obtenía, pues al ser la más problemática se podría investigar acerca de este hecho y poder obtener mejores resultados del modelo optimizando dicha clase.
- Integrar un sistema híbrido combinando varios clasificadores, tanto basados en algoritmos de aprendizaje automático como el modelo presentado, como algorimos basados en recursos léxicos. De esta manera, se podría ver que algoritmo funciona mejor y utilizar dicho algoritmo.
- Combinándolo con lo anterior, se podría implementar una aplicación web para cargar nuestro modelo y poder hacer predicciones a tiempo real de tweets que se realizen mediante Scrapping Web o técnicas similares de recogida de información. Dicha aplicación tendría filtros para buscar por palabras o patrones de texto y así se podrían observar las tendencias sobre diferentes conceptos o temáticas.
- Otra idea futura sería el poder aplicar el mismo modelo híbrido con otros corpus de otras redes sociales, como de Twitch, ya que es una red social que está en pleno crecimiento. Al ser redes sociales, seguramente el modelo híbrido funcione bastante bien.

Capítulo 6

Anexos

6.1. Requisitos de computación

Uno de los principales problemas que me he encontrado durante la realización del TFG, ha sido como solventar los problemas referentes a mi tarjeta gráfica.

Para entrenar una red neuronal podemos utilizar tanto la CPU como la GPU. Normalmente se suelen entrenar con la segunda ya que acelera el proceso significativamente.

Si nos decantasemos por ejecutar los programas en nuestro ordenador, lo primero que deberíamos de saber son los requisitos para utilizar TensorFlow con GPU.

Lo primero a tener en cuenta es que necesitas tener una tarjeta gráfica NVIDIA que soporte CUDA. Se puede consultar la lista que proporciona NVIDIA[49].

Lo siguiente a tener en cuenta es buscar para tu tarjeta gráfica los drivers de instalación y los paquetes compatibles con tu versión de CUDA que admite tu tarjeta gráfica. Para ello se debe consultar la guía que ofrece TensorFlow para ejecutarse con GPU[50].

Por todos los problemas que he tenido, no recomiendo a nadie utilizar los recursos de su propio ordenador para entrenar una red neuronal, a no ser de que tengas un super ordenador específico para realizar dichas tareas. En el siguiente apartado explicaré algunas de las opciones que he barajado que ofrecen computación en la nube.

6.2. Alternativas de entorno

Como ya he dicho, no es nada recomendable utilizar tus propios recursos por lo que puedes recurrir a alguna de las opciones que se plantean.

Google Colab es la alternativa utilizada ya que es gratuita y viene con unos recursos computacionales suficientes: Ofrece una memoria RAM de 12.69 GB y una memoria de disco de 107,72 GB. Además presenta una GPU gratuita NVIDIA Tesla K80 o la T4 dependiendo de la que se

te asigne. Ofrece un cuaderno de Jupyter para ejecutar los programas y es perfecto para escribir código en python.

Google Colab Pro es la versión de pago del programa anterior, cuesta 9.99$ al mes y ofrece mayor RAM, mayor disco y una GPU mejor. Google Colab Pro + es la versión mejorada de la anterior, cuesta 49.99$ al mes y ofrece mejores GPUs, más memoria RAM y espacio en disco.

Kaggle es otra plataforma muy interesante que ofrece un entorno gratuito. Cuenta con una GPU NVIDIA TESLA P100, con RAM de 16GB y espacio de disco de 5GB.

Existen otras opciones[51] como Amazon SageMaker de Amazon Web Services pero no tiene opciones gratuitas por lo que en principio no la recomiendo. Para otro tipo de proyectos más grandes es posible visitar la página de la ACADEMIC RESEARCH COMPUTING[52] y ponerse en contacto con ellos.

Bibliografía

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

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

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

[4] *Salario Ingeniero de Datos.* URL: https://es.talent.com/salary?job=Ingeniero+de+Datos.

[5] *Definición Inteligencia Artificial.* URL: https://dle.rae.es/inteligencia#2DxmhCT.

[6] *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.

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

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

[9] *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..

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

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

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

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

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

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

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

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

[18] *Python e Inteligencia Aritificial.* URL: https://blog.centrodeelearning.com/2020/12/23/aprender-phyton-lenguaje-ia/.

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

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

[21] *Keras Tuner.* URL: https://keras.io/keras_tuner/.

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

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

[24] *Overleaf.* URL: https://es.overleaf.com/.

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

[26] *Estadísticas de uso de twitter.* URL: https://www.websiterating.com/es/research/twitter-statistics/#:~:text=Estad%5C%C3%5C%ADsticas%5C%20y%5C%20hechos%5C%20generales%5C%20de%5C%20Twitter&text=Hay%5C%20un%5C%20total%5C%20de%5C%201.3,publican%5C%20500%5C%20millones%5C%20de%5C%20tweets..

[27] *Conceptos de Twitter.* URL: https://www.webempresa.com/blog/que-es-twitter-como-funciona-2.html.

[28] *Deep Learning Based Text Classification: A Comprehensive Review.* URL: https://arxiv.org/pdf/2004.03705.pdf.

[29] *Taller Análisis Semántico.* URL: http://tass.sepln.org/tass_data/download.php.

[30] *Uso del sobremuestreo.* URL: https://www.quora.com/Is-it-a-good-idea-to-undersample-or-oversample-a-heavily-imbalanced-dataset-from-a-statistical-perspective.

[31] *Tokenizadores de python.* URL: https://towardsdatascience.com/an-introduction-to-tweettokenizer-for-processing-tweets-9879389f8fe7.

[32] *Grid Layout vs Random Layout.* URL: https://analyticsindiamag.com/why-is-random-search-better-than-grid-search-for-machine-learning/.

[33] *Elección de funciones de activación.* URL: https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/.

[34] *Sobreajuste.* URL: https://machinelearningparatodos.com/que-es-el-sobreajuste-u-overfitting-y-por-que-debemos-evitarlo/.

[35] *Regularización de Dropout.* URL: https://vincentblog.xyz/posts/dropout-y-batch-normalization.

[36] *Ejemplo de K-fold cross validation.* URL: https://es.stackoverflow.com/questions/442783/cross-validation-%5C%C3%5C%B3-gridsearchcv.

[37] *Word Embedding.* URL: https://pharos.sh/python-para-nlp-embeddings-de-palabras-para-el-aprendizaje-profundo-en-keras/.

[38] *Descarga de palabras incrustadas preentrenadas.* URL: https://github.com/dccuchile/spanish-word-embeddings.

[39] *GloVe.* URL: https://en.wikipedia.org/wiki/GloVe.

[40] *FastText.* URL: https://en.wikipedia.org/wiki/FastText.

[41] *Word2Vec.* URL: https://en.wikipedia.org/wiki/Word2vec.

[42] *Imagen BiLSTM.* URL: https://www.ibzstore.com/?category_id=4126842.

[43] *Ganancia de Información.* URL: https://stackoverflow.com/questions/41159030/how-information-gain-works-in-text-classification.

[44] *Fórmulas IG y Entropía.* URL: https://mariuszprzydatek.com/2014/10/31/measuring-entropy-data-disorder-and-information-gain/.

[45] *Matriz de confusión.* URL: https://empresas.blogthinkbig.com/como-interpretar-la-matriz-de-confusion-ejemplo-practico/.

[46] *Dificultades análisis de sentimientos.* URL: https://itelligent.es/es/analisis-de-sentimiento/.

[47] *Dificultades análisis de sentimientos 2.* URL: https://www.ucm.es/data/cont/docs/758-2019-01-04-TFG_Panico_Chiara_TFG.pdf.

[48] *Aplicaciones análisis de sentimientos en Twitter.* URL: https://monkeylearn.com/blog/sentiment-analysis-of-twitter/.

[49] *Lista de NVIDIA con soporte de CUDA.* URL: https://developer.nvidia.com/cuda-gpus.

[50] *Compatibilidad con GPU TensorFlow.* URL: https://www.tensorflow.org/install/gpu?hl=es-419.

[51] *Aumento memoria RAM.* URL: https://stackoverflow.com/questions/62745386/is-it-possible-to-increase-the-ram-in-google-colab-with-another-way.

[52] *Web oficial de ACADEMIC RESEARCH COMPUTING.* URL: https://arc.wpi.edu/.