Universidad Zaragoza
1542

TRABAJO DE FIN DE GRADO

MODELOS DE EVALUACIÓN EN INTELIGENCIA ARTIFICIAL Y MACHINE LEARNING

Autor:
Sanz Barreras, Rebeca

Directores:
Íñiguez Dieste, David
Rivero Gracia, Alejandro

UNIVERSIDAD DE ZARAGOZA

FACULTAD DE CIENCIAS

JULIO 2024

A mi padre, por ser el faro de luz que me guía y me inspira
a cada paso que doy, por hacer de mi una valiente navegante.
A mi madre, por ser mi chaleco salvavidas sin importar
la adversidad.

Y a mis amigos, por ser la mejor tripulación
con la que podía haber surcado las aguas de este viaje.

Índice

1. Introducción **1**

2. Estado del arte y fundamentos **4**
2.1. El perceptrón y las redes neuronales 5
2.2. Redes neuronales recurrentes 6
2.3. Redes *Transformer* 7

3. Enfoque de la solución y herramientas utilizadas **10**
3.1. Análisis de los datos 11
3.2. Extracción de características 12
3.3. Algoritmos de clasificación 12
3.4. Métricas de rendimiento 14
3.5. Extracción de características mediante Bolsa de palabras. Clasificador Naive Bayes 15
3.5.1. Corpus de datos limpio 15
3.5.2. Datos de *Kampal Collective Learning* 17
3.6. Extracción de características con Bolsa de palabras. Red neuronal como clasificador. 20

4. Modelo final, análisis y resultados obtenidos **21**

5. Conclusiones **24**

Resumen

A la hora de diseñar un modelo de lenguaje de Inteligencia Artificial capaz de proponer respuestas a una pregunta debe tenerse un *feedback* indicador en su entrenamiento de cuan buena ha sido la respuesta. Este *feedback* puede tener su origen en un modelo de *Machine Learning* entre muchos otros. El desarrollo de este trabajo se fundamenta en el diseño de un algoritmo capaz de predecir la calidad de una respuesta mediante una puntuación. Para ello se exploran varias técnicas, tanto de extracción de características como de clasificación de las mismas, con el objetivo último de converger al modelo con mejor rendimiento.

Esto tiene una aplicación directa con el proyecto de la spin-off *Kampal Data Solutions*, el cual engloba una herramienta *online* generadora de inteligencia colectiva, *Kampal Collective Learning*. Esta prevé la implementación de *bots*, o lo que es lo mismo, modelos de lenguaje generadores de respuestas. Estos simularán usuarios conectados proponiendo buenas ideas a considerar por el resto de individuos. El rendimiento de estos simuladores será evaluado con el modelo desarrollado en este trabajo.

1. Introducción

Definimos inteligencia colectiva, CI, como el tipo de inteligencia que surge cuando un elevado número de individuos trabaja de manera colaborativa en un mismo esfuerzo intelectual [1]. Uno de los paradigmas más importante que se presenta al hablar de Inteligencia Colectiva es la siguiente pregunta: "¿Es más inteligente una comunidad de individuos que el más inteligente de sus miembros?". De acuerdo a numerosos estudios sociológicos la respuesta es que sí, pero siempre y cuando se cree el ambiente propicio para un trabajo colaborativo.

La spin-off de la Universidad de Zaragoza *Kampal Data Solutions* se encuentra inmersa en el proyecto *"La física de la Inteligencia Humana Colectiva y la propagación de opiniones en la red"* [2], donde se propone un marco teórico que explica las grandes interacciones entre humanos tratando de generar inteligencia colectiva, todo ello haciendo uso de la Mecánica Estadística sobre la red. En base a dicha teoría y con el propósito de llevar la cooperación entre individuos a un marco digital, surge el desarrollo de la herramienta *online Kampal Collective Learning*[1]. Esta es capaz de monitorear toda la información acerca de las interacciones en la red de usuarios, teniendo así constancia de cómo las ideas han sido generadas, copiadas, modificadas y propagadas.

Además, trabajar en un contexto digital no altera la posibilidad de que las soluciones cooperativas sigan siendo mejores a las individuales. Si pensamos en ello, quizás concluyamos que es incluso beneficioso, pues no solo evitamos los poderes de influencia que algunos individuos puedan tener debido a su posición social sino que también garantizamos que inicialmente todas las opiniones dentro del proceso deliberativo tendrán el mismo peso, evitando así la creación de peligrosos sesgos sociales. Esto tiene una gran aplicación en el mundo real si pensamos en entornos tales como debates políticos en sociedades democráticas modernas, información generada y propagada en redes sociales, o situaciones en un entorno escolar donde ocurren la mayor parte de los experimentos realizados en el proyecto.

La implementación de dicha herramienta se realiza sometiendo al grupo de individuos conectados de manera *online* a siete fases, estas fases simularán cualquier situación de cooperación entre humanos de la vida cotidiana.

Además, con el fin de paliar sesgos de la interacción humana son introducidas de manera progresiva tres dinámicas:

- Dinámica de permutación: la aplicación cambia el lugar de los participantes en la red con el objetivo de una máxima propagación de ideas. Es así como abordamos el problema de

[1] `https://www.kampal.com/collective-learning/`

la no interacción de un individuo con el entorno de sus vecinos (ya sea por afinidad, por falta de interés...).

- Dinámica de copia: esta dinámica pretende que aquellos usuarios que no han tenido una respuesta globalmente aceptada reflexionen sobre las soluciones de otros individuos. Para ello, dado un usuario, la aplicación toma la respuesta de uno de sus vecinos y la sobrescribe sobre la respuesta de dicho usuario. Este intercambio se realizará teniendo en cuenta la calidad de la respuesta del vecino mencionado. De esta manera, esta dinámica neutraliza la tendencia de no prestar atención a las ideas propuestas por otras personas e impulsa la convergencia de unas pocas soluciones.
- Dinámica de extinción: se eliminan ideas que no han sido ampliamente compartidas, dando la posibilidad al usuario de copiar soluciones más frecuentes de otros individuos o escribir una nueva.

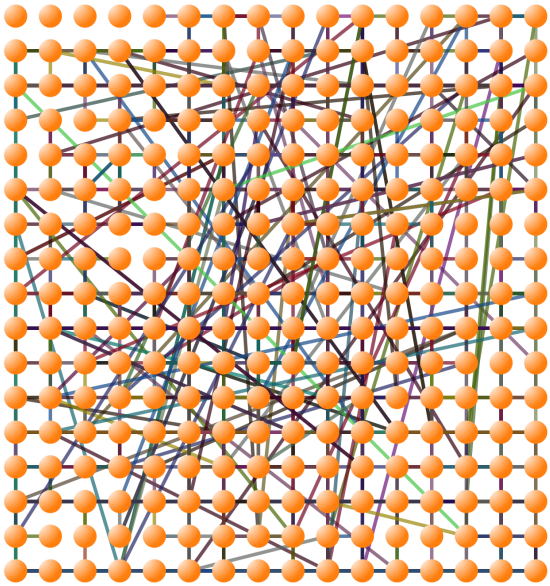

Figura 1.1: [2] Posiciones de los usuarios y copias en el transcurso de las 7 fases.

Una descripción más detallada de las fases por las que pasan los usuarios conectados es la siguiente (véase Figura 1.2 como apoyo):

- Fase 0: Lectura de la situación propuesta y entendimiento del problema.
- Fase 1: Respuesta individual de cada usuario. No hay posibilidad de visualizar respuestas vecinas.
- Fase 2: Cada usuario puede ver las respuestas escritas en la Fase 1 por sus 4 vecinos más cercanos.
- Fase 3: Cada usuario puede visualizar las respuestas de sus 4 vecinos más cercanos a tiempo real. Entra en juego la dinámica de permutación.
- Fase 4: Además de lo implementado en la fase 4, comienza la dinámica de copia.
- Fase 5: Se integra la dinámica de extinción, estando vigente aún lo implementado en la fase 3 y 4.
- Fase 6: Cada usuario visualiza el Top10 de las respuestas y tiene la posibilidad aún de escribir.

- Fase 7: Cada usuario únicamente puede copiar ideas ya existentes en el Top10.

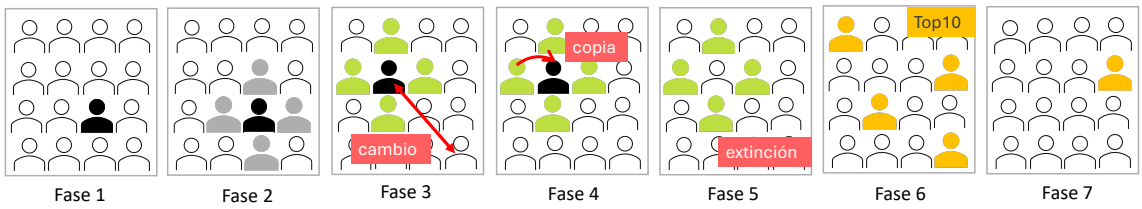

Figura 1.2: Esquema de la evolución de las 7 fases de la herramienta *online Kampal Collective Learning*.

Es así finalmente como se consigue generar inteligencia colectiva en un contexto digital. El siguiente paso previsto en el desarrollo de esta herramienta es la implementación de *bots*, que generen ideas óptimas en relación al problema planteado, simulando así individuos conectados que proponen soluciones interesantes a tener en cuenta por el resto. Todo ello con el mismo objetivo siempre de generar mayor inteligencia colectiva.

A la hora de trabajar en estos simuladores es necesario utilizar herramientas de Inteligencia Artificial (IA) en el campo del Procesamiento de Lenguaje Natural (NLP, *Natural Language Processing*), en particular Grandes Modelos de Lenguaje (LLM, *Large Language Model*).

Estos modelos de lenguaje son entrenados y optimizados para la tarea específica mediante aprendizaje por refuerzo con retroalimentación humana (RLHF, *Reinforcement Learning from Human Feedback*), técnica fundamental para permitir que los modelos produzcan respuestas más útiles, inofensivas y honestas, tal y como se demuestra en aplicaciones tales como ChatGPT, Gémini...

Todo ello es llevado a cabo en dos etapas, véase Figura 1.3 donde se presenta el ”mapa de ruta” de este proyecto.

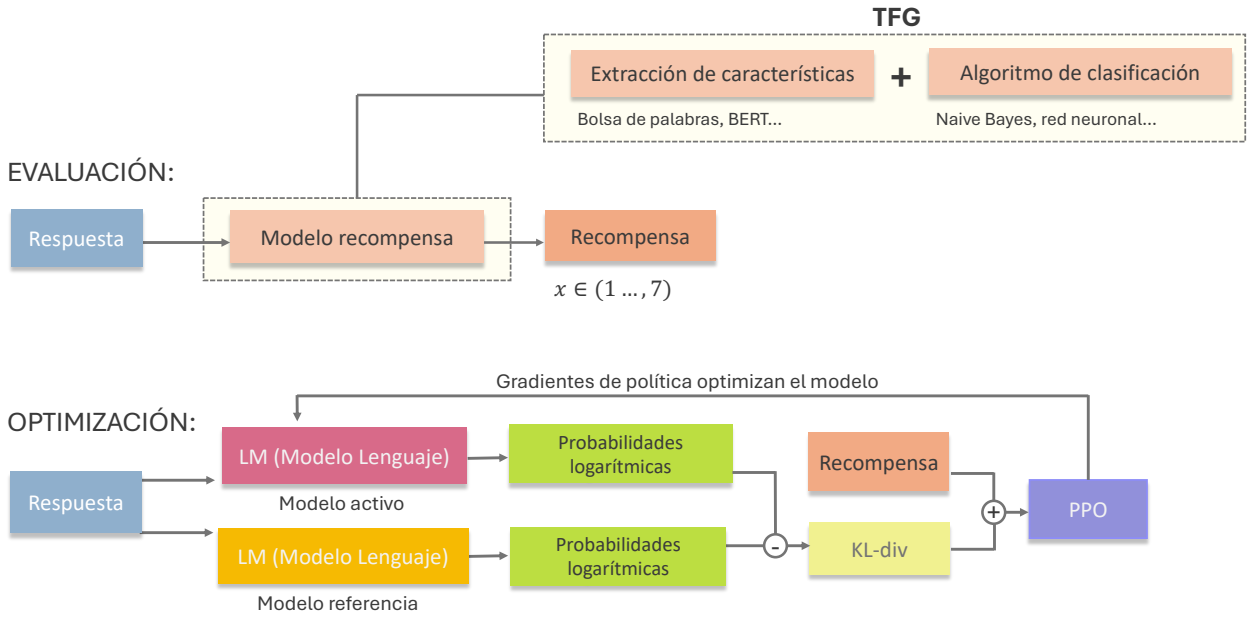

Figura 1.3: Flujo de trabajo del proyecto general. [3] Trabajo de fin de grado enfocado en la parte sombreada y encuadrada. ”Respuesta” hace referencia al conjunto de datos proporcionados por *Kampal Collective Learning*.

El proceso de optimización es el principal y más complejo, y tiene como objetivo el entrenamiento y ajuste fino del modelo ("Modelo activo" en Figura 1.3).

Este comienza con un modelo previamente entrenado y un modelo de referencia, donde ambos generan probabilidades logarítmicas que serán posteriormente comparadas. Tras ello, mediante una "recompensa" que mide la calidad de la respuesta generada y el uso de la divergencia de Kullback-Leibler (KL-div) [2] se evalúa su rendimiento. Finalmente, los gradientes de política optimizan el modelo utilizando el entrenador PPO (Proximal Policy Optimization)[3], el cual considerando tanto la recompensa como la divergencia KL ajusta los parámetros del modelo de lenguaje activo consiguiendo así mejorar su rendimiento [3]. Cabe destacar que este es el caso para clasificación multiclase, puntuación $x \in (1, ..., 7)$, para el caso de clasificación binaria es intercambiado el entrenador PPO por un entrenador KTO [4].

En definitiva, se pretende que mediante un algoritmo de aprendizaje por refuerzo, que tiene como objetivo maximizar una recompensa acumulada, se consiga el ajuste fino de un modelo enfocado a la tarea que nos concierne. Esta recompensa mencionada puede tener su origen en una función, feedback humano o como es nuestro caso, un modelo de IA, etapa de evaluación en la Figura 1.3. Este proporcionará una métrica que evaluará la calidad de la respuesta y será la que llamamos "recompensa" en el ajuste fino del modelo grande (etapa optimización).

El propósito de este Trabajo Fin de Grado es llevar a cabo dicha etapa de evaluación, en particular, diseñando un modelo de clasificación que permita estimar la calidad de una respuesta. Esto conllevará tareas como preprocesamiento de la información y análisis exploratorio de los datos, estudio de posibles algoritmos y finalmente diseño e implementación del modelo.

2. Estado del arte y fundamentos

En la última década la Inteligencia Artificial se ha convertido en el campo líder de tareas de procesamiento y generación de información a través de la aparición del aprendizaje automático (o *Machine Learning*) basado en redes neuronales. Esto ha tenido aplicaciones tales como procesamiento de voz, visión artificial, procesamiento del lenguaje natural... pretendiendo en cierta medida dotar a sistemas computacionales de la capacidad de aprender.

Como bien se ha comentado, el propósito de este trabajo es el desarrollo de un modelo capaz de evaluar ideas generadas por un LLM. Este modelo debe por tanto saber interpretar lenguaje.

El lenguaje es una facultad del ser humano para expresarse y comunicarse a través del sonido articulado o de sistemas de signos [4]. Podemos distinguir dos tipos de lenguaje, por un lado el lenguaje natural, lenguaje humano, este está en constante crecimiento sin tener en cuenta las reglas que lo rigen, es lo que comúnmente llamamos idiomas. Por otro lado, lenguajes formales que se encuentran enmarcados en disciplinas como la matemática, la lógica o la programación, los cuales están ceñidos rigurosamente a reglas establecidas. Además, el lenguaje natural se caracteriza por su flexibilidad, ambigüedad e indeterminación, permitiendo así la variedad en la interpretación dependiendo de la situación, lo cual resulta ventajoso en el momento de efectuar la comunicación humana. Sin embargo, al momento de enfrentarse al procesamiento computacional dichas características se presentan como un problema ya que dificultan la aplicación de procesos de razonamiento, caracterización y formalización. [5]

Se define Procesamiento del Lenguaje Natural (NLP) como el campo de estudio que busca

[2] Esta mide la diferencia entre las distribuciones de probabilidad del modelo activo y del modelo de referencia, básicamente se utiliza como señal de recompensa adicional para garantizar que las respuestas generadas no se desvíen demasiado del modelo de lenguaje de referencia

[3] Para más información visitar `https://openai.com/index/openai-baselines-ppo/`

[4] `https://dle.rae.es/lenguaje`

entender cómo funciona el lenguaje, su construcción, la generación de nuevo lenguaje, así como todas las tareas que tienen relación con el tratamiento del mismo [6]. Entre estas tareas se tiene la generación de nuevo texto, traducciones de un idioma a otro, generar un resumen... o inclusión el diseño de *bots*, como es la cuestión que nos concierne.

Es gracias a la llegada del aprendizaje profundo (o *Deep Learning*) que se han generado avances significativos en el campo del NLP. Las redes neuronales artificiales son las herramientas fundamentales en esta disciplina y de las cuales hablaremos a continuación en más detalle.

2.1. El perceptrón y las redes neuronales

Se define Inteligencia Artificial (IA) como la "ciencia que tiene como objetivo el diseño y construcción de máquinas capaces de imitar el comportamiento inteligente de las personas". [7]

Los algoritmos más usados en esta disciplina son las denominadas Redes Neuronales Artificiales (RNA), que buscan de cierta manera representar el funcionamiento biológico de una red de neuronas en el cerebro humano.

Una neurona biológica recibe señales a través de sus dentritas, procesa estas señales en el soma y produce una salida a través de su axón [8]. De manera similar lo hace el modelo más básico de una RNA, el perceptrón, este recibe entradas, las procesa mediante una combinación lineal de pesos y produce una salida basada en una función de activación, véase Figura 2.1.

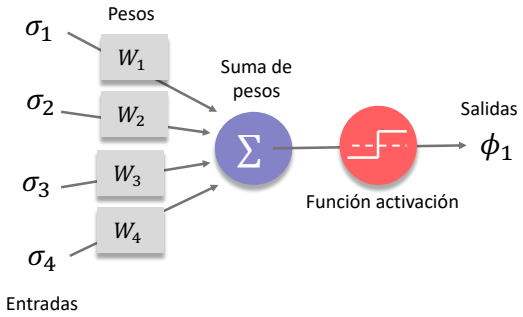

Figura 2.1: Esquema básico del perceptrón

El modelo matemático de la neurona artificial es,

$$\phi_1 = f\left(\sum_{i=1}^{N} W_i\sigma_i - b\right) \tag{2.1}$$

donde f es la función de activación (función no lineal), σ_i las entradas, W_i los pesos asociados y b el sesgo o *bias*.

De manera más detallada, Figura 2.1, los inputs σ_i llegan con un peso asociado W_i, de esta manera el perceptrón recibe por cada conexión un potencial $W_i\sigma_i$, que tras procesar la suma de todos ellos y atravesar la función de activación la neurona devuelve finalmente un único número, la salida. Las funciones de activación más comunes son la función escalón, ReLU (*Rectified Lineal Unit*) y Sigmoide.

Una capa de la red se compone de múltiples perceptrones, de manera que la salida producida por este es utilizada por los perceptrones de la siguiente capa. La conexión entre las capas se realiza mediante la matriz de pesos W_i^j y el vector de sesgos b^j. Para una capa en partícular, el elemento W_{ij} representa la conexión de la i-ésima neurona de la capa anterior y la j-ésima neurona de la capa actual. El vector b^j contendrá los sesgos asociados a cada neurona en la capa actual. Al organizar las neuronas en capas sucesivas, se forma una Red Neuronal Artificial de tipo perceptrón o red neurona *feedforward*, véase Figura 2.2, esto permite al ordenador procesar los datos intentando emular el pensamiento humano. [5]

[5]Debe tenerse en cuenta la escala de los modelos actuales comerciales, estos están dotados de un centenar de capas y más de 12000 neuronas por capa y entrada. Un millón de neuronas por entrada, significa que los modelos de contexto largo (con más de 10 mil entradas) pueden llegar a tener durante la generación más neuronas activas que un cerebro humano.

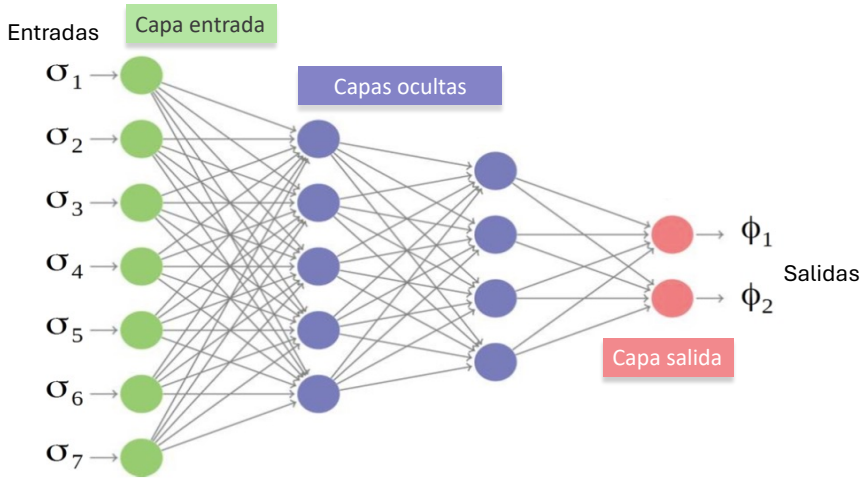

Figura 2.2: Esquema de una red neuronal tipo perceptrón. [8]

Las redes neuronales de este tipo siguen siendo fundamentales en el campo de la IA pero presentan limitaciones significativas en el campo del Procesamiento de Lenguaje Natural, especialmente cuando se trata de aplicaciones generativas [6]. Los perceptrones son modelos de clasificación que pueden aprender a distinguir entre clases utilizando una función de activación simple. Sin embargo, su capacidad para capturar la complejidad y las dependencias temporales inherentes al lenguaje es extremadamente limitada, es así como se impulsó el desarrollo de modelos más avanzados.

2.2. Redes neuronales recurrentes

Fueron las Redes Neuronales Recurrentes (RNN) que revolucionaron el campo del NLP al introducir la capacidad de procesar secuencias de datos de manera efectiva. A diferencia de las redes neuronales de tipo perceptrón, las cuales asumen independencia entre los datos de entrada, las RNN poseen un estado oculto que tiene en cuenta el estado de tiempo anterior [7].

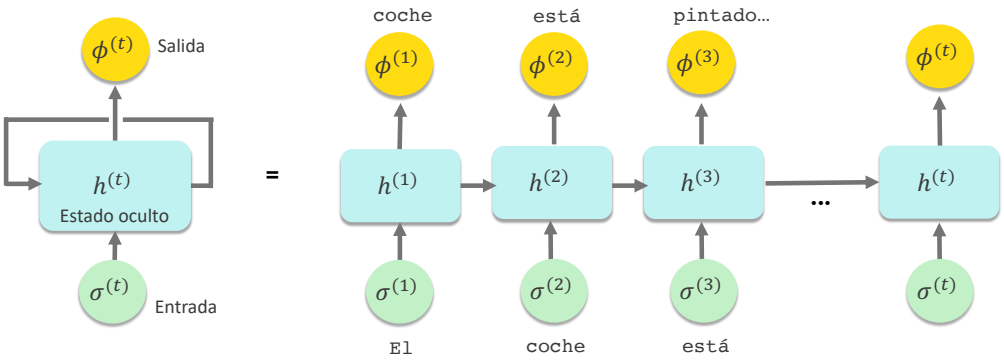

Figura 2.3: Esquema de una red neuronal recurrente. Formato reducido a la izquierda y su despliegue a la derecha. Notar como en el despliegue las entradas para un tiempo t son tanto σ_t como el estado oculto anterior h_{t-1}

[6] Se denominan aplicaciones generativas a modelos que pueden producir contenido nuevo basado en patrones aprendidos a partir de datos existentes. Estos modelos tienen la capacidad de generar texto, traducir idiomas, crear diálogos coherentes...

[7] Se define tiempo en las RNN no como el tiempo cronológico, sino a la posición de la secuencia de datos de entrada (esto puede ser la posición específica de una secuencia de palabras en un texto, posición de las letras en una palabra...)

Una RNN procesa una secuencia de datos paso a paso, manteniendo un estado oculto que se actualiza en cada paso de tiempo (h_t en la Figura 2.3). Podemos representar su funcionamiento matemáticamente como,

$$\text{Estado oculto: } h^{(t)} = f_1(W_h h^{(t-1)} + W_\sigma \sigma^{(t)} + b_h) \tag{2.2}$$

$$\text{Salida: } \phi^{(t)} = f_2(W_\phi h^{(t)} + b_\phi) \tag{2.3}$$

donde f_1 y f_2 son funciones de activación, $h^{(t-1)}$ es el estado oculto en el tiempo $t-1$, $h^{(t)}$ en el tiempo t y $x^{(t)}$ la entrada en el tiempo t, b_h y b_y los sesgos y W_h, W_x y W_y matrices de pesos.

Notar como la salida depende del estado oculto $h^{(t)}$, y esta a su vez del anterior $h^{(t-1)}$. Es con estas conexiones recurrentes que tienen la capacidad de mantener una "memoria" sobre los estados anteriores y una longitud variable. Esto es crucial en el NLP donde la comprensión del contexto y la secuencia de las palabras es esencial para el diseño de un buen modelo [10].

Aunque las RNN han demostrado ser muy efectivas para el campo del NLP, presentan un problema significativo de paralelización. La naturaleza secuencial de las RNN requiere que los estados ocultos se calculen uno tras otro, lo que impide el procesamiento paralelo de las secuencias. Esto resulta en un entrenamiento más lento y menos eficiente. Además, a medida que la red procesa más elementos de la cadena, tiene problemas en recordar información pasada, es por esto que las RNN más sencillas no son capaces de aprender patrones muy extendidos en el tiempo.

Con el fin de mitigar dicho inconveniente, aparece un nuevo tipo de celda de memoria que sí es capaz de trabajar con secuencias de mayor longitud. Se conoce como celdas de Memoria de Corto y Largo Plazo (LSTM, por sus siglas en inglés *Long-Short Term Memory*). Años posteriores, nace el GRU (*Gated Recurrent Unit*), arquitectura más eficiente computacionalmente.

Las RNN y sus variantes han demostrado ser excepcionalmente efectivas en una amplia gama de aplicaciones. Entre ellas la generación de texto, donde son capaces de producir secuencias de textos coherentes y contextualmente relevantes aprendiendo patrones complejos del lenguaje a partir de grandes corpus de datos. Sin embargo, estas redes son notorias por su lentitud y alta complejidad computacional debido a su diseño no paralelizable.

2.3. Redes *Transformer*

Hasta la aparición de una nueva arquitectura en 2017, los modelos de transducción de secuencias se basaban en complejas redes neuronales recurrentes o convolucionales. Numerosos esfuerzos se centraron en ampliar los límites de estos modelos, incluyendo arquitecturas de codificador-decodificador, trucos de factorización... pero siempre dependiendo de la principal restricción, la computación secuencial, y con ello la no paralelización de los algoritmos.

Es en 2017 con la publicación del artículo "Attention Is All You Need" por Vaswani et al. [12], cuando el campo del NLP se revoluciona ante la novedosa y simple arquitectura *Transformer*, que prescinde por completo de las recurrencias y convoluciones. En su lugar, depende completamente de mecanismos de autoatención que representan dependencias globales entre la entrada y la salida. Numerosos experimentos afirmaron que estos modelos eran de calidad superior, a la vez que eran más paralelizables y requerían menos tiempo de entrenamiento [12].

El *Transformer* fue propuesto como un mecanismo de autoatención siguiendo la arquitectura general presentada en la Figura 2.4.

Sin entrar mucho en detalle [8] diferenciamos dos partes principales, el codificador o *encoder* (parte izquierda, Figura 2.4) y el decodificador o *decoder* (parte derecha). Ambos se forman mediante $N = 6$ capas de atención (*attention layers*) apiladas una tras otra. Estas capas contienen principalmente una subcapa de autoatención y una subcapa con una red neuronal de tipo *Feed Forward*.

Por una parte, el codificador asigna una secuencia de entrada de representaciones de símbolos [9] $(x_1, ..., x_n)$ a una secuencia de representaciones continuas $z = (z_1, ..., z_n)$, conocidas como embeddings, son estas últimas las que nos permitirán capturar relaciones semánticas entre símbolos. El codificador lleva a cabo esta transformación de manera paralela, es decir, analiza todos los símbolos de entrada a la vez. Es por esta misma razón que esta parte de la arquitectura presenta inconvenientes en aplicaciones generativas pero funciona realmente bien para capturar el significado semántico de oraciones.

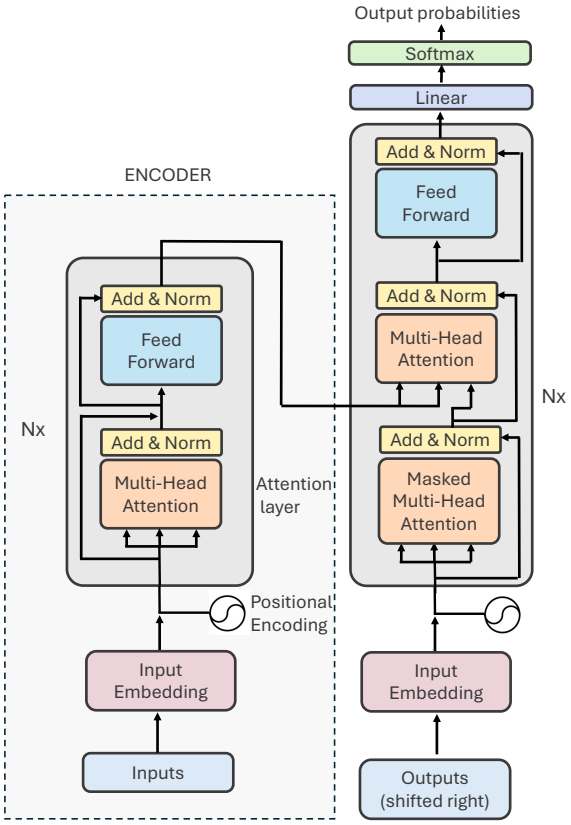

Figura 2.4: [12] Esquema de la arquitectura del modelo Transformer. La zona recuadrada izquierda representa el codificador y la parte derecha el decodificador. Capa de atención representada como *Attention layer*

Por otra parte, dado z, el decodificador genera una secuencia de salida $(y_1, ..., y_m)$ de símbolos. Además, en cada paso el decodificador es auto-regresivo, esto es que produce cada elemento de la secuencia de salida de manera secuencial, donde cada símbolo generado se utiliza como entrada adicional para general el siguiente símbolo, es decir, el modelo "regresa" y utiliza su propia salida previa para continuar generando la secuencia. De manera contraria al codificador, esta parte de la arquitectura es ampliamente utilizada en aplicaciones generativas y no lo es tanto para capturar significados semánticos.

El mecanismo de autoatención es la clave de esta arquitectura y permite que cada token [10] de la secuencia de entrada se relacione con todos los demás tokens de la misma secuencia, proporcionando una forma efectiva de capturar dependencias a largo plazo y contextos globales. Para ello, se proponen tres elementos esenciales: *Query* (Consulta), *Key* (Clave) y *Value* (Valor), donde la consulta y la clave son transformaciones del valor:

$$|K_i\rangle = \mathcal{O}_\mathcal{K} |V_i\rangle \qquad |Q_i\rangle = \mathcal{O}_\mathcal{Q} |V_i\rangle \tag{2.4}$$

donde $|K_i\rangle$, $|Q_i\rangle$ y $|V_i\rangle$ son la clave, consulta y valor respectivamente y $\mathcal{O}_\mathcal{K}$, $\mathcal{O}_\mathcal{Q}$ son matrices de transformación.

[8] No es de gran interés prestar atención en la estructura general y compleja de esta arquitectura para el entendimiento de este trabajo, pues el modelo de lenguaje utilizado para la representación numérica del lenguaje es solo una parte de este, explicado en más detalle más adelante

[9] Se denomina "símbolos" en el contexto del NLP a las unidades discretas de datos que el modelo maneja, estos pueden ser palabras completas, tokens, caracteres...

[10] Un token es una unidad de texto que ha sido segmentada a partir de una secuencia mayor

El vector de salida proporcionado por el mecanismo de autoatención es,

$$|V_i'\rangle = \sum_{j=1}^{L} \text{softmax}\left(\frac{\langle K_j|Q_i\rangle}{\sqrt{d_k}}\right)|V_j\rangle \tag{2.5}$$

donde $|V_i'\rangle$ es el valor de salida correspondiente a un token, $|V_j\rangle$ el valor correspondiente al j-ésimo token de entrada y L el número de tokens o también denominado longitud de contexto.

Sustituyendo ambos elementos de la Ecuación 2.4 en Ecuación 2.5 y tomando $M = \mathcal{O}_{\mathcal{K}}^T\mathcal{O}_{\mathcal{Q}}$:

$$|V_i'\rangle = \sum_{j=1}^{L} \text{softmax}\left(\frac{\langle V_j|\,\mathcal{O}_{\mathcal{K}}^T\mathcal{O}_{\mathcal{Q}}\,|V_i\rangle}{\sqrt{d_k}}\right)|V_j\rangle = \sum_{j=1}^{L} \text{softmax}\left(\frac{\langle V_j|\,M\,|V_i\rangle}{\sqrt{d_k}}\right)|V_j\rangle \tag{2.6}$$

De manera más detallada, $\langle K_j|Q_i\rangle$ es el producto escalar de los embeddings generados por el codificador, este producto escalar es realizado en un espacio vectorial de dimensión d_k y mide la similitud entre vectores, podríamos decir que genera unas "puntuaciones de atención" que indican cuanta atención debe prestar el decodificador a cada token de entrada. Además, los valores resultantes del producto escalar son escalados mediante $\frac{1}{\sqrt{d_k}}$, esto es relevante para valores de d_k grandes [12]. Finalmente, la función *softmax* convierte los "puntuaciones de atención" en "pesos de atención", probabilidades que suman 1 para cada entrada.

Numerosas variantes del *Transformer* han ido surgiendo a lo largo del tiempo, entre ellas, BERT (*Bidirectional Encoder Representations from Transformers*), modelo desarrollado por investigadores de Google en 2018. Este tuvo una gran importancia en el campo del NLP gracias a su capacidad para comprender el contexto bidireccional de las palabras en una secuencia de texto.

El aprendizaje bidireccional se fundamenta en la comprensión del lenguaje no solo de izquierda a derecha, sino también en el otro sentido, ayudando a tener una visión global de las sentencias que recibe.

El modelo utilizado para la representación numérica del lenguaje en este trabajo es un modelo DistilBERT, esto es básicamente una variante del modelo BERT con una capa de *pooling*, véase Figura 2.5 para una representación esquemática del mismo. Esta arquitectura aún conservando aproximadamente el 97 % del rendimiento de BERT reduce su tamaño y aumenta su velocidad de procesamiento. Este modelo es particularmente útil para tareas como la búsqueda semántica, la agrupación y la comparación de similitudes semánticas entre textos, tarea crucial en el desarrollo de este trabajo. Es por esta razón que utiliza únicamente la parte de codificación del modelo general *Transformer*, zona encuadrada con línea de puntos y sombreada en la Figura 2.4.

Dicha arquitectura procesará la secuencia de datos mediante los mecanismos de autoatención, generando así un vector de dimensión 768 para cada token de entrada. Después de procesar la secuencia, el modelo aplica una capa de *pooling* de la cual se obtiene un vector semántico de dimensión 512. Es este vector el que recibirá el correspondiente clasificador para evaluar la calidad de la respuesta.

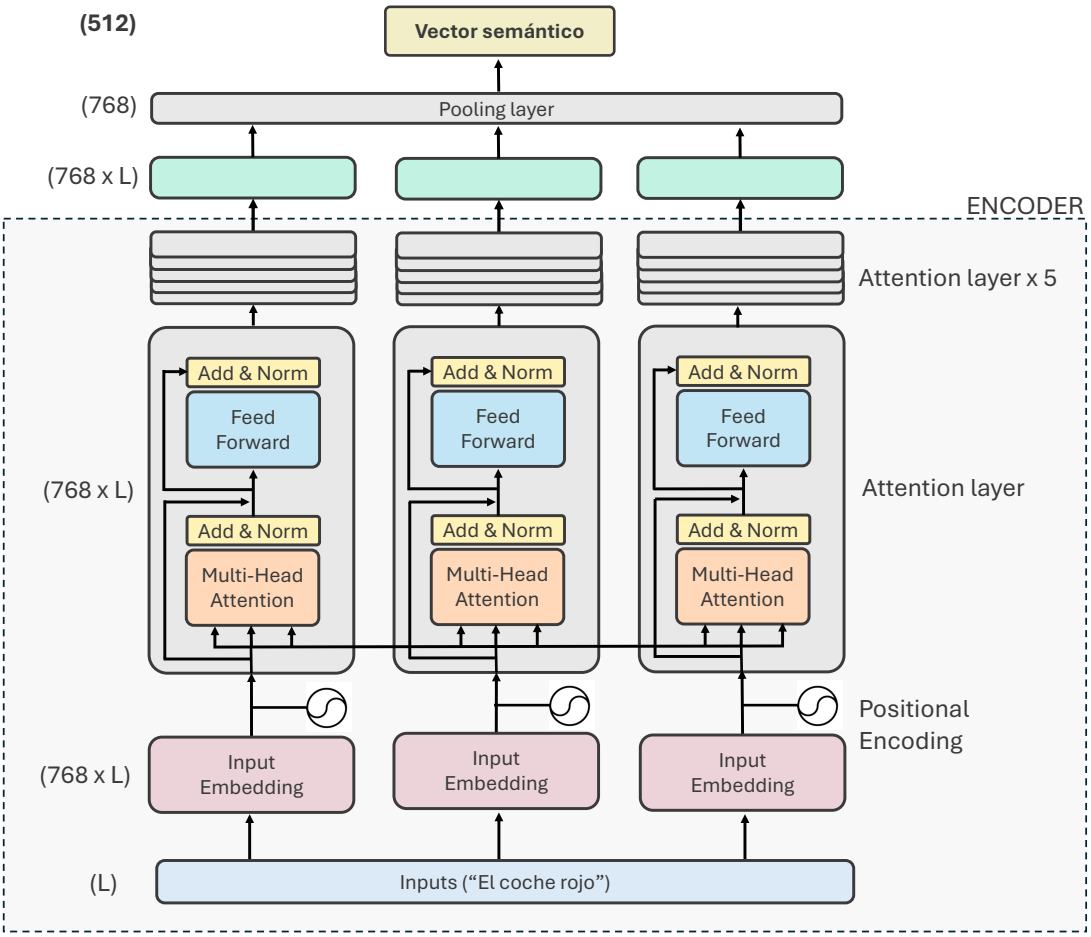

Figura 2.5: Esquema de la arquitectura Transformer seguida por el modelo de lenguaje utilizado en este trabajo. Notar que la zona encuadrada y sombreada representa la parte de codifiación o *encoder* presentada en la Figura 2.4.

3. Enfoque de la solución y herramientas utilizadas

La línea de trabajo seguida en este proyecto fue la búsqueda en todo momento del mejor modelo capaz de predecir una métrica de evaluación de un conjunto de respuestas. Como se ha comentado anteriormente, los ordenadores no comprenden el lenguaje natural, es por ello que existe un paso previo al modelo de clasificación, donde debemos transformar nuestras entradas (textos) en un formato legible para la computadora, esta etapa es la que denominaremos ”Extracción de características”, véase Figura 1.3. En ella abordaremos la representación numérica del lenguaje de distintas formas.

La siguiente etapa, ”Algoritmo de clasificación” en Figura 1.3, consiste en introducir las características extraídas en un clasificador, este será entrenado con un conjunto de entrenamiento para la tarea que nos interesa y posteriormente su rendimiento será evaluado con un conjunto de test.

Las diferentes combinaciones planteadas fueron:

1. Extracción de características mediante la técnica *Bag of Words* (Bolsa de palabras) y clasificación mediante el algoritmo Naive Bayes. Estas herramientas serán planteadas para dos corpus de datos.

1.1 Datos de prueba limpios

1.2 Datos de *Kampal Collective Learning*

2. Extracción de características mediante Bolsa de palabras y clasificación mediante red neuronal de tipo perceptrón multicapa. Datos de *Kampal Collective Learning*.
3. Extracción de características mediante un modelo BERT, (Arquitectura *Transformer*) y clasificación mediante red neuronal de tipo perceptrón multicapa. Datos de *Kampal Collective Learning*.

Todo ello teniendo en cuenta dos maneras de categorizar las respuestas: clasificación binaria $C = 0, 1$ y clasificación multiclase de 7 categorías $C = 1, ..., 7$.

Tanto la técnica de bolsa de palabras como el clasificador Naive Bayes son relativamente muy simples, en cambio, la extracción de características mediante una arquitectura *Transformer* y una red neuronal como clasificador son técnicas más sofisticadas. De esta manera, observaremos la manera en la que la calidad de los resultados evoluciona a medida que introducimos mejores herramientas.

3.1. Análisis de los datos

El primer paso de este trabajo fue conseguir un gran corpus de datos etiquetados, es así como los posteriores modelos aprenderán los patrones necesarios para unas buenas predicciones.

Los datos en crudo fueron proporcionados por *Kampal Collective Learning*, estos consistían en un corpus de 72628 textos sin etiquetar, respuestas de individuos que habían sido conectados a dicha aplicación y que tras el planteamiento de una pregunta o situación y el transcurso de las 7 fases, sus respuestas habían quedado registradas con una serie de parámetros útiles. Estos parámetros junto con un óptimo procesamiento de los datos podrán proporcionarnos información acerca de la evolución de la respuesta y su calidad. Es con esta información que se establecerá un criterio para el etiquetado de las respuestas.

Algunos de los muchos parámetros proporcionados e interesantes para el etiquetado de los datos fueron:

- `Question_id`: Identificador de la pregunta.
- `Frecuency`: Frecuencia de aparición, número total de veces que una idea aparece entre los usuarios (Recordamos que las respuestas se repiten debido a la dinámica de copia entre individuos o de manera espontánea).
- `Frequency_now`: Número de veces que una solución aparece en el sistema en cada momento.
- `First_seen`: Fecha y hora que se registró por primera vez la respuesta en el sistema.
- `Last_seen`: Fecha y hora en la que deja de aparecer la respuesta en el sistema (esto puede ser por la dinámica de extinción, dinámica de copia...).
- `Last_seen_fase`: Número de fase en la que deja de aparecer la respuesta en el sistema.
- `Time_difference_in_seconds`: Tiempo de "vida" de una respuesta. Realmente esto es `last_seen` − `first_seen`.

Tras un análisis exploratorio de los datos, se llega a la conclusión de que los parámetros de `Frecuency`, `Frequency_now` y `Last_seen_fase` serían las mejores opciones para categorizar nuestro conjunto. Pues si una respuesta ha llegado a una fase alta (`Last_seen_fase` alta) significa que ha sido aceptada globalmente y las dinámicas de la red no han decidido eliminarla. Además, es un buen criterio pensar que una respuesta con una alta frecuencia total (`Frecuency`) es una

buena respuesta, pues este parámetro es indicativo de las veces que los usuarios han decidido copiarla.

Si además combinamos el parámetro `Frequency_now` y `Last_seen_fase` podremos afirmar que las respuestas llegadas a la fase 7 y con una `Frequency_now` > 0 son buenas soluciones. Esto es así porque si además de haber conseguido llegar a la fase 7 (recordar que en esta fase los usuarios votan en un Top10), la frecuencia de uso en ese momento es distinta de 0 significará que ha sido al menos votada más de una vez como la mejor solución de entre las 10 mejores.

En cuanto al parámetro `Time_difference_in_seconds` se concluye que no es buen indicativo acerca de la calidad de una respuesta, pues una buena solución puede haber nacido en la fase 6 por aprendizaje de vecinos, tener un bajo `Time_difference_in_seconds` y seguir siendo buena respuesta.

3.2. Extracción de características

Si deseamos obtener características numéricas de un conjunto de textos, podríamos pensar en algo tan simple como la cantidad de palabras, el número de conectores... o algo más complejo que tenga en cuenta el significado semántico de las palabras y el contexto, esto es la arquitectura *Transformer* explicada anteriormente.

La primera técnica que se llevó a cabo fue lo que comúnmente se denomina *Bag of Words* (Bolsa de Palabras en su traducción), y consiste en explorar todas las palabras del conjunto de entrenamiento, para posteriormente unificar un diccionario de todas ellas. De esta manera, toda palabra perteneciente a un texto estará también en dicho diccionario o "bolsa de palabras". Sin embargo, hay palabras como preposiciones, conjunciones, articulos... que se consideran irrelevantes para el análisis del texto. Es por esta razón que se descartan del diccionario los siguientes casos:

- *Stopwords*, conjunto de palabras sin significado del contenido del texto. Ejemplo: el, la, los, como, para, con...
- Palabras que contienen "http", "@" 0 "#".
- Palabras de menos de 3 letras.

La segunda variante y más sofisticada fue el modelo "sentence-transformers/distiluse-base-multilingual-cased-v1" [11], se trata de un modelo *Transformer* de la familia "Sentence Transformers" y su arquitectura esta basada en una variante del modelo de lenguaje DistilBERT [17], este está optimizado para mapear oraciones y párrafos a vectores de 512 dimensiones. Su arquitectura de forma más detallada fue la presentada en la Subsección 2.3.

3.3. Algoritmos de clasificación

Como modelos de clasificación entrenados mediante aprendizaje por refuerzo se tomaron en primera instancia el clasificador *Naive Bayes* y posteriormente una red neuronal de 3 capas.

Los métodos Naive Bayes son un conjunto de algoritmos de aprendizaje supervisado basados en la aplicación del teorema de Bayes con el supuesto "ingenuo" de independencia condicional entre cada par de características dado el valor de la etiqueta del item [13].

El teorema de Bayes enuncia que dada una partición del suceso B, $B_1, ..., B_n$, la probabilidad condicionada de B_k dado el suceso A cumple [16]:

[11] Cabe destacar que otra opción con la que se podía haber trabajado es el modelo de lenguaje "MarIA", para más información consultar [18].

$$P(B_k|A) = \frac{P(A|B_k)P(B_k)}{\sum_{i=1}^{n} P(A|B_i)P(B_i)} \tag{3.1}$$

Si extrapolamos a nuestro caso:

$$P(C_k|\text{text}) = \frac{P(\text{text}|C_k)P(C_k)}{\sum_{i=1}^{n} P(\text{text}|C_i)P(C_i)} = \frac{P(\text{text}|B_k)P(C_k)}{P(\text{text})} \tag{3.2}$$

donde C_k con $k = 1, ..., N$ representa las N clases en las que serán categorizadas las respuestas. En nuestro caso, $N = 2$ para clasificación binaria y $N = 7$ para clasificación en 7 categorías. Notar que "text" hace referencia a una respuesta y realmente text $= (x_1, ...x_n)$ es un vector de dimensión n que representa numéricamente las características de dicha respuesta.

Usando el supuesto de independencia condicional entre características:

$$P(\text{word}_i|\text{text}, \text{word}_1, ..., \text{word}_n) = P(\text{word}_i|\text{text}) \tag{3.3}$$

podemos afirmar que para todo i:

$$P(\text{text}|B_k) = \prod_{i=1}^{n} P(\text{word}_i|\text{text}) \tag{3.4}$$

Esto significa que la aparición de una palabra en una respuesta no tiene correlación con la aparición de cualquier otra.

Sustituyendo Ecuación 3.4 en Ecuación 3.2 obtenemos finalmente,

$$P(B_k|\text{text}) = \frac{P(B_k)\prod_{i=1}^{n} P(\text{word}_i|\text{text})}{P(\text{text})} \tag{3.5}$$

Dada una única respuesta, $P(\text{text})$ es constante y podemos escribir,

$$P(B_k|\text{text}) \propto P(B_k)\prod_{i=1}^{n} P(\text{word}_i|\text{text}) \tag{3.6}$$

de esta manera podemos definir finalmente el criterio de clasificación como:

$$\hat{B} = \arg\max_{B} P(B_k)\prod_{i=1}^{n} P(\text{word}_i|\text{text}) \tag{3.7}$$

eligiendo así la clase cuya probabilidad condicionada es máxima para un mismo texto.

Los clasificadores Naive Bayes poseen suposiciones aparentemente demasiado simplificadas, pero a pesar de ello han funcionado bastante bien en muchas situaciones del mundo real, como la clasificación de documentos y el filtrado de spam [13], posteriormente veremos cómo de óptimo es en nuestra tarea.

El segundo algoritmo de clasificación seleccionado fue una red neuronal de tipo perceptrón de 3 capas, diseñada e implementada acorde a la tarea que se requería. Su diseño fue prácticamente el mismo en todos los casos salvo ligeras diferencias entre clasificación binaria y multiclase.

Los distintos elementos que conformaron la red neuronal fueron:

- Tres capas lineales de dimensiones variables en función del vector de características de entrada.
- Función de activación ReLU (Rectified Linear Unit) aplicada después de las dos primeras capas y función de activación softmax después de la última capa, esta era crucial en el diseño de la red, pues convierte las salidas en probabilidades, asignando a cada clase una probabilidad entre 0 y 1.
- Función pérdida de entropía cruzada binaria (*Binary Cross Entropy Loss)* para el caso de clasificación binaria y función de entropía cruzada generalizada (*Cross Entropy Loss)* para el caso de multiclase. Esta función medirá la discrepancia entre las prediciones del modelo y las etiquetas reales.
- Optimizador SGD o Descenso de Gradiente Estocástico (Stochastic Gradient Descent). Este algoritmo será el que ajuste los parámetros del modelo en la dirección opuesta al gradiente de la función de pérdida. Todo ello con un *learning rate* (tasa de aprendizaje) que variará en cada caso, este hiperparámetro controlará el tamaño del paso de actualización.

La red neuronal será entrenada mediante un proceso iterativo, a través del cual ajustará los parámetros de la propia red (pesos y sesgos). A grandes rasgos, el algoritmo coge datos de entrenamiento, los pasa a través de la red neuronal (con los valores que en aquel momento tengan sus parámetros), compara el resultado obtenido con el esperado y calcula la función de pérdida. La función de pérdida guiará al optimizador para calcular un nuevo valor de cada uno de los parámetros. Todo ello con la finalidad siempre de reducir la función de pérdida en cada iteración y obtener así un modelo que genere mejores predicciones.

Más en detalle, las funciones de pérdida mencionadas, siguen las siguientes ecuaciones [14] [15]:

$$\text{Entropía cruzada binaria: } l(x,y) = -w \cdot (y \cdot log(x) + (1-y) \cdot log(1-x)) \tag{3.8}$$

$$\text{Entropía cruzada generalizada: } l(x,y) = -w \cdot log\left(\frac{exp(x_y)}{\sum_{c=1}^{C} exp(x_c)}\right) \tag{3.9}$$

donde x representa la entra, y la salida y C el número de clases.

3.4. Métricas de rendimiento

Las métricas de rendimiento se emplean para cuantificar cómo de bueno o de malo es un modelo en cuanto a la calidad de sus predicciones. Entre ellas, encontramos la matriz de confusión, esta es una matriz de dimensión $C \times C$ donde los elementos de la diagonal se corresponden con los aciertos, y los de fuera de ella con los errores.

Particularizando al problema de clasificación binaria, las dimensiones de la matriz son 2x2 y tiene la forma presentada en Tabla 3.1.

		Modelo	
		Negativa	Positiva
Real	Negativa	TN	FP
	Positiva	FN	TP

Tabla 3.1: Esquema de una matriz de confusión.

Cada elemento de la matriz representa un resultado:

- True Positive (TP): Un dato que se toma como verdadero tanto por el humano como por el modelo.
- True Negative (TN): Un dato que se toma como falso tanto por el humano como por el modelo
- False Positive (FP): Un dato que para nosotros es falso pero el modelo considera que es verdadero.
- False Negative (FN): Un dato que para nosotros es verdadero pero que el modelo considera que es falso.

A partir de la matriz de confusión se puede extraer mucha información. De entre los muchos indicadores existentes, se consideran en este trabajo TPR (True Positive Rate o sensibilidad) y TNR (True Negative Rate o especifidad). Estos serán los que se incorporarán en los resultados como métrica de calidad del modelo, columna "Rendimiento" a partir de ahora.

$$TNR = \frac{TN}{N} = \frac{TN}{TN + FP} \tag{3.10}$$

$$FNR = \frac{FN}{P} = \frac{FN}{TP + FN} \tag{3.11}$$

Más en detalle se define sensibilidad como la proporción de verdaderos positivos correctamente identificados por el modelo y especificidad como la proporción de verdaderos negativos correctamente identificados. En otras palabras, ambas miden la capacidad de identificar correctamente la clase correspondiente.

Notar que lo que en la matriz de confusión comúnmente se denomina "Negativa" y "Positiva" será para nosotros "Respuesta mala" y "Respuesta buena" respectivamente.

Para nuestro caso, todas las métricas de rendimiento fueron obtenidas sobre el 10 % del conjunto total de los datos (conjunto de test). El 90 % restante sería reservado para el entrenamiento (conjunto de entrenamiento).

El conjunto de test estará conformado por una muestra aleatoria de N respuestas del conjunto original, de esta manera mantendremos las propiedades iniciales del origen de los datos. Cabe destacar que ese conjunto en ningún caso será utilizado para el entrenamiento de los modelos.

3.5. Extracción de características mediante Bolsa de palabras. Clasificador Naive Bayes

Procederemos a estudiar la primera de las combinaciones citadas de extracción de características y algoritmo de clasificación. Estas técnicas serán inicialmente probadas en un corpus de datos de prueba limpio y correctamente etiquetado, posteriormente aplicaremos dichas herramientas a los datos de *Kampal Collective Learning*.

3.5.1. Corpus de datos limpio

Este conjunto de datos o *dataset* constaba de 12498 reseñas en inglés recogidas por un supermercado acerca de sus productos, servicios o instalaciones, etiquetadas en función de si eran opiniones positivas o negativas.

Cuando hablamos de un conjunto de datos o *dataset* limpio, estamos queriendo decir que los registros son textos bien escritos, con sentido y coherencia, sin emoticonos, sin caracteres que carezcan de sentido... y con un correcto y fiable etiquetado, a todos los efectos "datos de laboratorio" [12].

Tras un filtrado de palabras con criterios similares [13] a los comentados en la Subsección 3.2 puede observarse una representación visual de la bolsa de palabras en la Figura 3.1.

(a) Reseñas totales (b) Reseñas negativas (c) Reseñas positivas

Figura 3.1: Nube de palabras del conjunto total de datos tras el filtrado de las mismas, el tamaño de cada palabra es proporcional a la frecuencia de aparición en el conjunto de datos. Representación escogida para 3 conjuntos de reseñas.

En la Figura 3.1 podemos observar varios aspectos interesantes:

- En el conjunto de reseñas totales obtenemos palabras tales como *smoothie*, *texture* (textura), *green* (ecológico), *powder* (polvo, referente a las especias y productos en polvo), *vanilla* (vainilla), *flavor* (sabor)... Esto sugiere que la mayoría de las reseñas son acerca de productos de comida.
- En el conjunto de reseñas negativas encontramos términos como *price* (precio), *good* (bueno), *order* (pedido), *store* (tienda), *post* (referente a *post office*, oficina postal). Esto puede indicar que las opiniones negativas pueden estar relacionadas con el precio, los pedidos, la oficina postal, aspectos de la tienda...
- En el conjunto de reseñas positivas observamos palabras como *back* (referente a *come back*, volver), *finding* (encontrar), *eat* (comer), *meat* (carne), *chicken* (pollo)... Dando a entender que las reseñas positivas se enfocan en productos de comida, más concretamente productos de carne.

Tras el breve análisis de los datos podemos proceder a comentar el entrenamiento y rendimiento del modelo. Uno de los factores importantes en el buen entrenamiento es el equilibrado de sus datos, es por ello que en este apartado han sido propuestos los resultados para tres opciones de equilibrado del conjunto de entrenamiento: desequilibrio positivo (más reseñas positivas que negativas), equilibrio y desequilibrio negativo.

[12] Debe tenerse en cuenta que la realidad del trabajo con datos masivos y el análisis de los mismos no es esta.
[13] Mismos criterios pero dejando las palabras con menos de 3 caracteres.

		Modelo		
		Negativa	Positiva	Rendimiento
Real	Negativa	171	224	43,3 %
	Positiva	104	2000	95,1 %

Tabla 3.2: Matriz de confusión. Datos de entrenamiento originales, desequilibrio positivo: 8384 reseñas positivas y 1615 reseñas negativas.

		Modelo		
		Negativa	Positiva	Rendimiento
Real	Negativa	295	100	74,4 %
	Positiva	229	1875	89,1 %

Tabla 3.3: Matriz de confusión. Datos de entrenamiento equilibrados: 1615 reseñas positivas y 1615 negativas.

		Modelo		
		Negativa	Positiva	Rendimiento
Real	Negativa	375	20	94,9 %
	Positiva	1004	1100	52,3 %

Tabla 3.4: Matriz de confusión. Datos de entrenamiento con desequilibrio negativo: 1615 reseñas positivas y 500 reseñas negativas.

Observamos de qué manera el equilibrado de los datos es influyente en el rendimiento del modelo, siendo el mejor de ellos el entrenado con datos equilibrados. Esto es así porque si una clase está mucho más poblada que la otra, los modelos tienden a clasificar mejor la primera de ellas para así maximizar el número de aciertos. Esto nos sugiere que en los próximos desarrollos deberemos trabajar con datos equilibrados.

3.5.2. Datos de *Kampal Collective Learning*

El *dataset* proporcionado por *Kampal Collective Learning* constaba de 69825 respuestas, las cuales debían ser etiquetadas tanto desde el punto de vista binario ($C \in 0, 1$) como desde el punto de vista multiclase ($C \in 1, ..., 7$). En este primer apartado presentaremos de nuevo 3 opciones de equilibrado para el conjunto de entrenamiento.

De manera análoga al caso planteado anteriormente puede observarse una representación visual de los términos más frecuentes de la bolsa de palabras, Figura 3.1.

(a) Respuestas totales. (b) Respuestas malas. (c) Respuestas buenas.

Figura 3.2: Nube de palabras del conjunto de entrenamiento tras el filtrado de las mismas, el tamaño de cada palabra es proporcional a la frecuencia de aparición en el conjunto de datos. Representación escogida para 3 conjuntos.

A diferencia del conjunto de reseñas de un supermercado, para este caso es mucho más difícil identificar diferencias de palabras importantes entre las respuestas buenas y malas. Es lógico si recordamos que los datos recogidos por *Kampal Collective Learning* son respuestas preguntas con temas muy distintos. Esto tendrá relevancia cuando el modelo tenga que aprender a clasificar la bondad de las respuestas.

Para el caso de clasificación binaria, los conjuntos propuestos para el etiquetado (recordamos que partimos de datos no etiquetados) y equilibrado de los mismos es:

- Datos originales (desequilibrio negativo): las respuestas son etiquetadas como respuestas buenas aquellas llegadas a la fase 7 y con una `Frequency_now`> 0. De esta manera se obtiene 61171 respuestas malas y 1671 buenas.
- Datos seleccionados equilibrados: son etiquetadas como buenas respuestas aquellas llegadas a la fase 6 y 7, y como malas aquellas llegadas a las fases 1,2,3. Esto se reduce a 15512 respuestas buenas y 19274 respuestas malas. Finalmente, el equilibrio se consigue realizando una selección de 15512 respuestas de entre las 19274 consideradas como malas[14].
- Datos seleccionados con desequilibrio negativo: Tomando el mismo criterio de etiquetado que en el punto anterior, se consigue un desequilibrio negativo realizando una selección 6000 respuestas de entre las 19274 consideradas como malas.

Para el caso de clasificación multiclase se utilizó el parámetro `last_seen_fase` como etiqueta de bondad de la respuesta, pues se considera un buen indicativo la fase a la que una respuesta ha llegado en el sistema.

Los resultados para los 3 conjuntos de entrenamiento son presentados en la Tabla 3.5, Tabla 3.6 y Tabla 3.7.

[14]Tener en cuenta que hay casos de equilibrado más potentes como las técnicas de *Triplet loss* o *Contrastive loss* donde se ha demostrado que los modelos funcionan mucho mejor. Las tareas que abordan estas herramientas son tareas de comparación y por esa misma razón no fueron incluidas en este trabajo, para más información visitar [19]

		Modelo		
		Negativa	Positiva	Rendimiento
Real	Negativa	6646	175	97,4 %
	Positiva	155	7	4,3 %

Tabla 3.5: Matriz de confusión. Datos de entrenamiento con desequilibrio negativo, 61171 respuestas malas y 1671 respuestas buenas.

		Modelo		
		Negativa	Positiva	Rendimiento
Real	Negativa	1691	453	78,9 %
	Positiva	1021	726	40,9 %

Tabla 3.6: Matriz de confusión. Datos de entrenamiento equilibrados, 15512 respuestas buenas y 15512 malas.

		Modelo		
		Negativa	Positiva	Rendimiento
Real	Negativa	901	1243	42,0 %
	Positiva	472	1225	16,2 %

Tabla 3.7: Matriz de confusión. Datos de entrenamiento con desequilibrio positivo, 15512 respuestas buenas y 6000 malas.

Podemos observar en los resultados como en el caso de un entrenamiento con desequilibrio negativo, Tabla 3.5, el modelo ha aprendido que toda respuesta es mayoritariamente negativa independientemente de sus características. De manera análoga ocurre para el entrenamiento con desequilibrio positivo, Tabla 3.7, aunque esta vez de manera no tan acentuada, pues la proporción de desequilibrio no es tan grande.

En cuanto al caso equilibrado, Tabla 3.6, podríamos decir que se trata del mejor entre los tres entrenamientos propuestos, pues obtiene el mejor balance de rendimiento para ambas clases. Sin embargo, atendiendo a la matriz de confusión podríamos sugerir que el clasificador funciona mejor con las respuestas malas que buenas por alguna razón intrínseca del mismo [15], es por ello que se decide llevar a cabo un entrenamiento con un ligero desequilibrio positivo (15512 respuestas buenas y 9000 malas) obteniendo un rendimiento del 66,6 % y del 52,6 % para el caso de malas y buenas respuestas respectivamente. Con ello podemos afirmar que el equilibrado de los datos en el conjunto de entrenamiento es conveniente pero no necesariamente un equilibrado exacto es siempre la mejor opción, esto dependerá de las técnicas utilizadas en cada caso. Por simplicidad, a partir de ahora los resultados serán obtenidos para un equilibrado exacto [16].

Para el caso de clasificación multiclase, el modelo es entrenado con 4000 registros de las 7 clases, de esta manera se espera que sea capaz de predecir cuan buena es una respuesta dando una puntuación del 1 al 7.

[15] Esto podría ser porque quizás las respuestas malas tengan más cosas en común que las buenas y la probabilidad condicionada funcione mejor sobre ellas.

[16] Cabe destacar que las pruebas realizadas apuntaban a que un equilibrio exacto era la mejor opción para el resto de técnicas

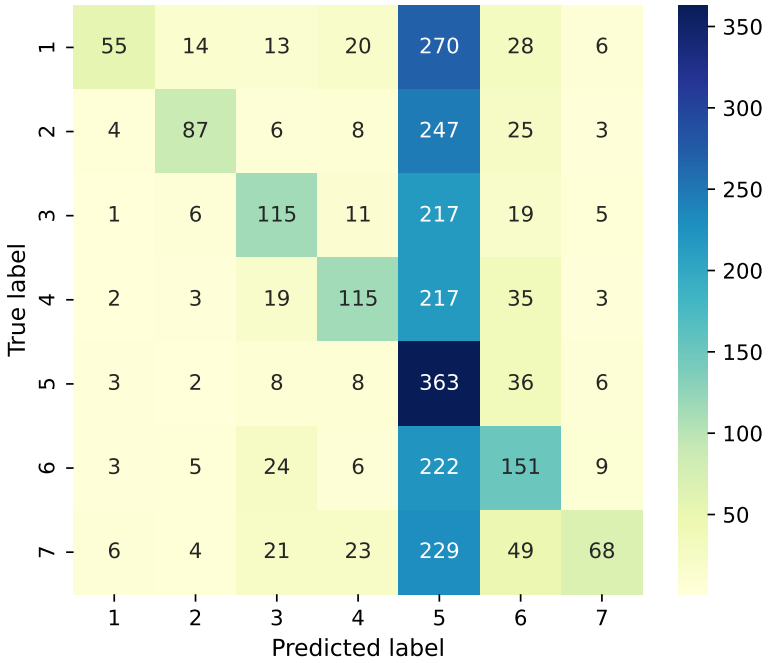

Figura 3.3: Matriz de confusión. Datos de entrenamiento equilibrados, 4000 registros por clase.

Atendiendo a la Figura 3.3 observamos que según nuestro modelo, la gran mayoría de las respuestas pertenecen a la fase 5. Es un resultado curioso si recordamos que este había sido entrenado con datos equilibrados. Sin embargo, para el resto de fases observamos buenas predicciones (números altos en la diagonal). La interpretación de esto es que el modelo ha conseguido un conjunto de "características útiles" que le hacen predecir bien la fase de una respuesta, pero en el momento que debe analizar un texto sin esas "características útiles" entonces traslada la respuesta al valor más seguro, en este caso es la fase 5. Que haya tomado como valor más seguro la fase 5 puede sugerir que las respuestas de esta fase tengan más cosas en común con el resto (recordamos que el clasificador está basado en probabilidades condicionadas).

Finalmente, podemos concluir que las técnicas de extracción de características y clasificación utilizadas en este apartado pueden llegar a funcionar bien para clasificación binaria pero no para el caso de multiclase. Esto es porque el enfoque binario puede ser más directo y menos susceptible a ciertos tipos de ruido en los datos. Con el fin de mitigar los errores en la clasificación multiclase se introduce una red neuronal de tipo perceptrón en lugar del clasificador Naive Bayes.

3.6. Extracción de características con Bolsa de palabras. Red neuronal como clasificador.

Tras la extracción de características análoga al apartado anterior y un entrenamiento con datos equilibrados, se obtuvieron los resultados presentados en la Tabla 3.8 y Figura 3.4.

		Modelo		
		Negativa	Positiva	Rendimiento
Real	Negativa	1005	1126	47,2 %
	Positiva	526	1201	69,5 %

Tabla 3.8: Matriz de confusión. Datos de entrenamiento equilibrados, 9000 respuestas buenas y 9000 respuestas malas.

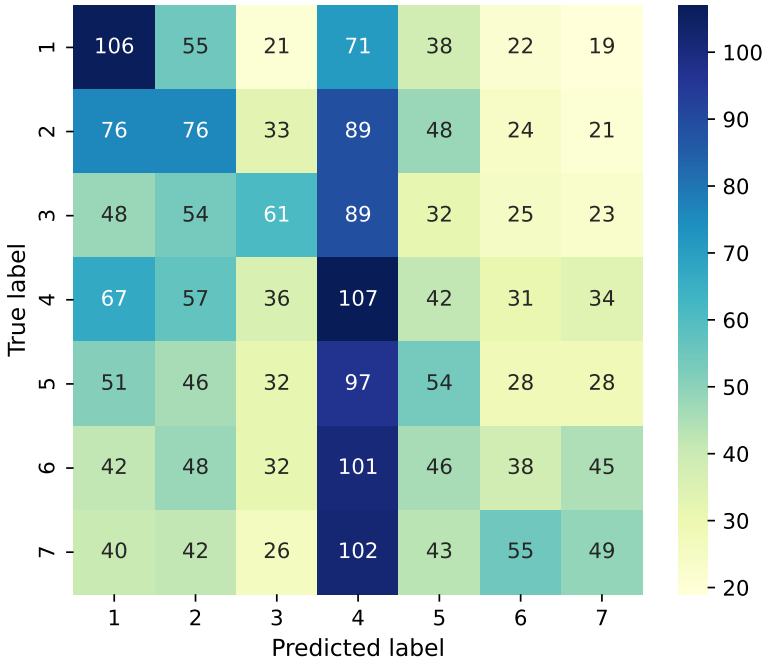

Figura 3.4: Matriz de confusión. Datos de entrenamiento equilibrados, 4000 registros por clase.

Observamos como para el caso binario se obtiene un rendimiento parecido al caso anterior, esto revela que cambiar el clasificador no establece mejoras significativas. En cambio es curioso observar la manera en la que el clasificador Naive Bayes conseguía predecir mejor las respuestas malas y en el caso de la red neuronal lo hace más fácilmente con las buenas.

En el caso de clasificación multiclase obtenemos mejoras muy sutiles. Lo más notorio es que el modelo ha aprendido a categorizar la gran mayoría de las veces en la fase 4, independientemente de sus características. Sin embargo podemos ver como la fase 1 la ha predicho bastante bien.

Además, cuando diseñamos la red para problemas de clasificación, la red no ”sabe” que la clase 1 es cercana a la 2 y lejana a la 7, no se trata de un problema de regresión, para la red son clases independientes. Sin embargo, vemos que en las fases iniciales ha conseguido aprender esta relación, de manera que si no categoriza bien en una fase, la clasifica en una cercana a ella.

Con el fin de obtener mejoras en el caso de clasificación multiclase, se introduce a continuación una técnica más sofisticada de extracción de características, una variante del modelo de lenguaje BERT (Subsección 2.3), el cual proporcionará vectores semánticos mucho más útiles para la red neuronal. Tanto esta técnica como la red neuronal como clasificacor conformarán los resultados finales del trabajo.

4. Modelo final, análisis y resultados obtenidos

Esta vez entraremos más en detalle acerca del progreso del entrenamiento, presentando así la evolución de la función de pérdida para cada época, tanto en el caso de clasificación binaria, Figura 4.1, como en el caso de clasificación multiclase, Figura 4.2.

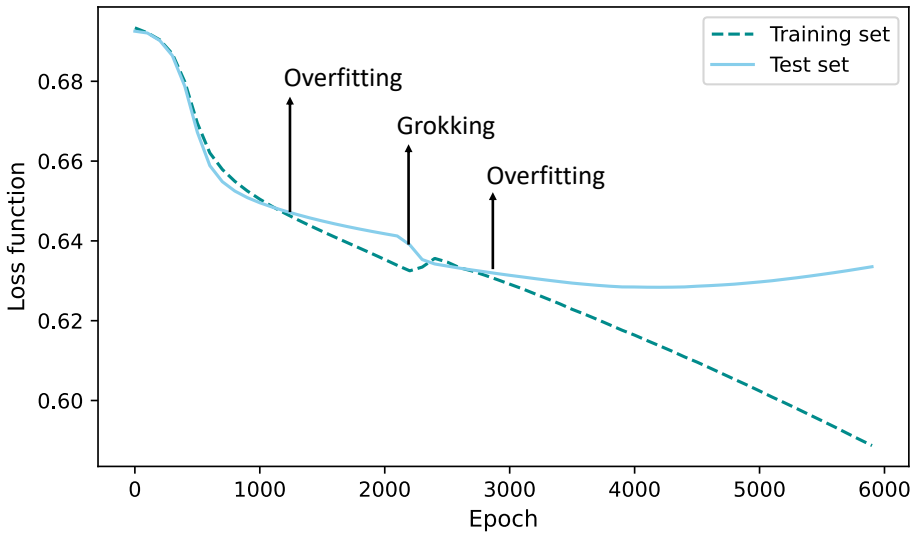

Figura 4.1: Función de pérdida para el conjunto de entrenamiento y conjunto de test. Clasificación binaria. Entrenamiento con datos equilibrados. Tasa de aprendizaje de 0,1.

Para el entrenamiento de clasificación binaria, véase Figura 4.1 observamos varios fenómenos:

- De 0 a 1000 épocas obtenemos lo esperado, el modelo entrena como es debido (descenso del conjunto de entrenamiento) y mejorando su rendimiento (descenso del conjunto de test)
- De 1000 a 2000 épocas la función de pérdida del conjunto de entrenamiento disminuye mientras que la del conjunto de test comienza a estabilizarse e incluso aumenta ligeramente. Esto indica que el modelo está empezando a aprender patrones específicos del conjunto de entrenamiento que no generaliza bien al conjunto de test. Este fenómeno es denominado *overfitting* o sobreajuste.
- En torno a las 2000 épocas observamos que a pesar de estar el modelo en condiciones de sobreajuste, tiene una caída brusca. Esto sugiere que el modelo ha capturado algún patrón generalizable que le hace mejorar su rendimiento. Este fenómeno es denominado como *grokking*.
- En torno a las 2500 épocas comienza de nuevo el fenómeno de sobreajuste, cada vez más notorio con el paso de las épocas.

Una vez visualizada la evolución del entrenamiento se decide optar por diseñar el modelo con un total de 2000 épocas, dejando aparecer el fenómeno de grokking y descartando el segundo sobreajuste. Los resultados finales se presentan en la Tabla 4.1, donde observamos una pequeña mejora con los resultados presentados en secciones anteriores.

		Modelo		
		Negativa	Positiva	Rendimiento
Real	Negativa	1379	775	64,0 %
	Positiva	659	1070	61,9 %

Tabla 4.1: Matriz de confusión. Datos de entrenamiento equilibrados, 15512 respuestas buenas y 15512 malas. Entrenamiento hasta 2000 épocas.

La evolución de la función de pérdida para el el caso de clasificación multiclase se presenta en la Figura 4.2.

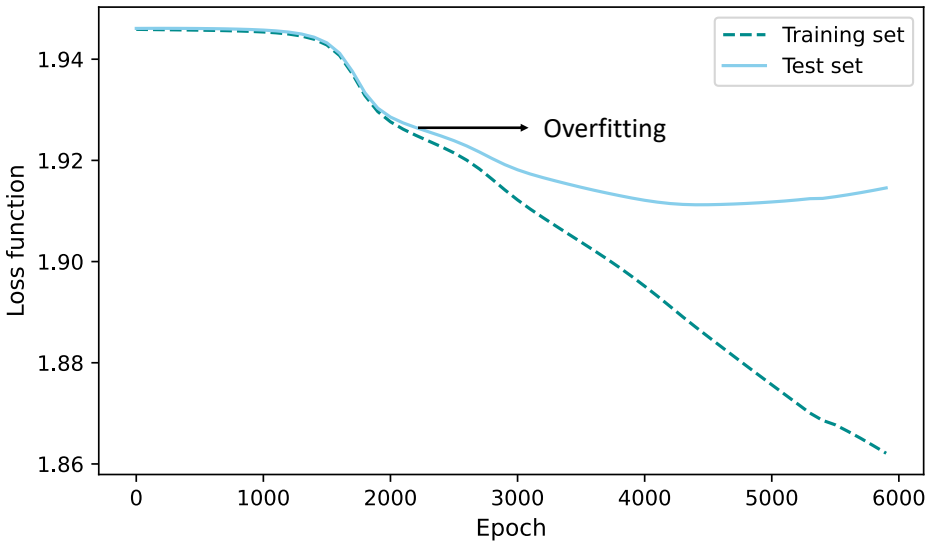

Figura 4.2: Función de pérdida para el conjunto de entrenamiento y conjunto de test. Clasificación multiclase. Entrenamiento con datos equilibrados. Tasa de aprendizaje de 0,9.

Observamos de nuevo en el entrenamiento del modelo la manera en la que este aprende adecuadamente hasta las 2000 épocas, a partir de entonces se produce el fenómeno de sobreajuste. Es por esta misma razón que el modelo final fue diseñado hasta las 2000 épocas.

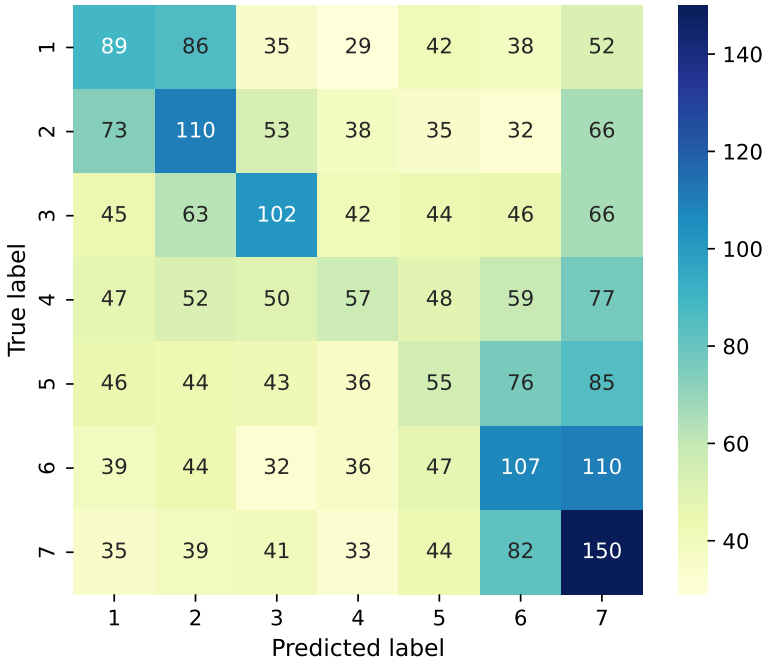

Figura 4.3: Matriz de confusión. Datos de entrenamiento equilibrados, 4000 registros por clase. Entrenamiento hasta 2000 épocas.

Atendiendo a la matriz de confusión, Figura 4.3, observamos la notable mejora en la calidad de nuestro modelo, donde ha conseguido categorizar las respuestas en la diagonal o por lo menos, cerca de ella.

5. Conclusiones

En este trabajo de fin de grado se ha explorado el desarrollo de diferentes técnicas de inteligencia artificial y *machine learning* para la evaluación de respuestas generadas por grandes modelos de lenguajes en el campo del NLP, teniendo una aplicación directa en uno de los proyectos llevados a cabo por *Kampal Data Solutions*, una herramienta *online* generadora de inteligencia colectiva (*Kampal Collective Learning*).

A lo largo del desarrollo del proyecto, se han abordado diversas metodologías y algoritmos con el objetivo de optimizar la capacidad de predicción y clasificación de las respuestas en función de su calidad. Establecer la conexión entre el lenguaje humano y el lenguaje de una computadora ha sido uno de los retos de este trabajo, así como la comprensión del aprendizaje de los modelos en función de su diseño.

Los resultados obtenidos han podido desvelar la manera en la que las predicciones evolucionan con la implementación de técnicas más avanzadas. Para la clasificación binaria ya podíamos recoger buenos resultados con las sencillas herramientas Bolsa de palabras y clasificador Naive Bayes. Además, la red neuronal como clasificador los mejoraba pero no significativamente. Sin embargo, en el caso de categorizar en varias clases no es hasta la implementación de la arquitectura *Transformer* que se obtuvieron resultados óptimos. Esta ha podido ofrecer una representación más rica y contextual del lenguaje, dotándole al clasificador (red neuronal) de información mucho más útil tanto para el entrenamiento como para las prediccione futuras.

Este trabajo ha proporcionado una base sólida para el desarrollo de una herramienta *online* generadora de inteligencia colectiva, dotándola de dos modelos de evaluación de respuestas para el entrenamiento e implementación de *bots*. Estos *bots* serán capaces de generar ideas interesantes a tener en cuenta por los usuarios conectados. Todo ello con el objetivo siempre de generar una mayor inteligencia colectiva.

Esta herramienta puede tener grandes aplicaciones si pensamos en cualquier ámbito de la vida cotidiana donde varios individuos deban trabajar de manera colaborativa. Entornos escolares, universitarios, debates políticos... son algunas de las opciones donde puede implementarse esta aplicación de manera beneficiosa. Gracias al contexto digital y sus dinámicas en la red se consigue mitigar peligrosos sesgos sociales, jerarquías de poder, aislamiento de individuos... situaciones que no favorecen a la convergencia de la respuesta cooperativa más inteligente.

Referencias

[1] Woolley, A. W., and Aggarwal, I. (2020). "Collective intelligence and group learning." *Oxford: Oxford University Press, 2020, doi:10.1093/oxfordhb/9780190263362. 013.46*

[2] Teresa Garcia, Alejandro Rivero, Alfonoso Tarancón, Carlos Tarancón, "The Physics of Collective Human Intelligence and Opinion Propagation on the Lattice" `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4870094`

[3] PPO Trainer. `https://huggingface.co/docs/trl/ppo_trainer`

[4] KTO Trainer. `https://huggingface.co/docs/trl/kto_trainer`

[5] I. Gil leiva y j. V. Rodríguez muñoz. "El procesamiento del lenguaje natural aplicado al análisis del contenido de los documentos" *Revista General de Información y Documentación, vol. 6, nº 2, pp. 205-218, 1996.*

[6] A. Cortez Vásquez, H. Vega Huerta y J. Pariona Quispe, "Procesamiento de lenguaje natural" *Revista de Ingeniería de Sistemas e Informática, vol. 6, nº 2, pp. 45-54, 2009*

[7] L. Munárriz, Fundamentos de inteligencia artificial, Murcia: Universidad de Murcia, 1994, pp. 19- 21.

[8] A.Tarancón Latifa. "Teoría, problemas y prácticas", 2021

[9] Jordi Torres. "Deep Learning. Introducción práctica con Keras. Primera parte", 2001.

[10] Carlos Arana, "Redes neuronales recurrentes: Análisis de los modelos especializados en datos secuenciales", Junio 2021.

[11] Néstor Camilo Beltrán, Edda Camila Rodríguez (2021), "Procesamiento del lenguaje natural (PLN) - GPT-3, y su aplicación en la Ingeniería de Software". Tecnol.Investig. Academia TIA, ISSN: 2344- 8288, 8 (1), pp. 18-37. Bogotá-Colombia

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser y I. Polosukhin. "Attention Is All You Need". 31st Conference on Neural Information Processing Systems, Long Beach, CA, USA, 2017.

[13] Naive Bayes, librería scikit-learn. `https://scikit-learn.org/stable/modules/naive_bayes.html`

[14] BCELoss, librería scikit-learn. `https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html`

[15] CrossEntropyLoss, librería scikit-learn. `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html`

[16] Luis Mario Floría "Lecciones de Física Estadística", 16 Enero 2023.

[17] sentence-transformers/distiluse-base-multilingual-cased-v1 `https://huggingface.co/sentence-transformers/distiluse-base-multilingual-cased-v1`

[18] Asier Gutiérrez-Fandiño, Jordi Armengol-Estapé, Marc Pàmies, Joan Llop-Palao, Joaquín Silveira-Ocampo, Casimiro Pio Carrino, Carme Armentano-Oller, Carlos Rodriguez-Penagos, Aitor Gonzalez-Agirre, Marta Villegas. "MarIA: Modelos de Lenguaje en Español"

[19] Raia Hadsell, Sumit Chopra, Yann LeCun. "Dimensionality Reduction by Learning an Invariant Mapping", Noviembre 2005 *The Courant Institute of Mathematical Sciences*