



UNIVERSITAT POLITÈCNICA DE CATALUNYA

BARCELONATECH

Facultat d'Informàtica de Barcelona



RECOMENDADOR DE RESTAURANTES MEDIANTE ANÁLISIS DE SENTIMIENTOS DE SUS RESEÑAS

CARLOS FULGENCIO VELASCO

Director/a

DANIEL JIMENEZ GONZALEZ (Departamento de Arquitectura de Computadores)

Titulación

Grado en Ingeniería Informática (Computación)

Memoria del trabajo de fin de grado

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Abstract

Castellano

El análisis de sentimiento es una tarea del campo de la inteligencia artificial cuyo propósito es determinar la polaridad del sentimiento de un texto. El objetivo de este proyecto es desarrollar un modelo basado en la arquitectura de transformers para analizar reseñas de restaurantes publicadas en Google Maps y asignar una puntuación del 1 al 5, donde 1 representa un sentimiento negativo y 5 un sentimiento positivo.

El trabajo realizado en este proyecto ha consistido en adaptar el modelo BERT a la tarea de clasificación y entrenarlo. Además, se ha indagado en la mejora de los resultados y se han realizado experimentos utilizando una implementación propia de la misma arquitectura, para evaluar si un modelo más pequeño puede obtener mejores o peores resultados que BERT.

El principal desafío de este trabajo ha sido el desarrollo de la arquitectura de transformers, incluyendo el de la capa de encoder y la transformación de su salida en probabilidades para predecir cada puntuación. Otro reto importante ha sido la transformación de datos categóricos a numéricos, para que el modelo sea capaz de procesarlos. Esta implementación ha permitido explorar en mayor profundidad el funcionamiento del mecanismo de atención, un componente clave en el avance de la inteligencia artificial, proporcionando una base sólida para trabajos futuros.

Català

L'anàlisi de sentiments és una tasca del camp de la intel·ligència artificial que té com a objectiu determinar la polaritat del sentiment d'un text. L'objectiu d'aquest projecte és desenvolupar un model basat en l'arquitectura de transformers per analitzar ressenyes de restaurants publicades a Google Maps i assignar una puntuació d'entre 1 i 5, on 1 representa un sentiment negatiu i 5 un sentimient positiu.

El treball fet en aquest projecte ha consistit a adaptar el model BERT a la tasca de classificació i entrenar-lo. A més, s'ha investigat la millora dels resultats i s'han dut a terme experiments utilitzant una implementació pròpia de la mateixa arquitectura, per avaluar si un model més petit pot obtenir millors o pitjors resultats que BERT.

El principal repte d'aquest treball ha estat el desenvolupament de l'arquitectura de transformers, incloent-hi la capa d'encoder i la transformació de la seva sortida en probabilitats per predir cada puntuació. Un altre desafiament important ha estat la transformació de dades categòriques a numèriques, perquè el model sigui capaç de processar-les. Aquesta implementació ha permès explorar amb més profunditat el funcionament del mecanisme d'atenció, un component clau en l'avenç de la intel·ligència artificial, proporcionant una base sòlida per a treballs futurs.

English

Sentiment analysis is a task in the field of artificial intelligence aimed at determining the polarity of a text's sentiment. The goal of this project is to develop a model based on the transformers architecture to analyze restaurant reviews published on Google Maps and assign a score from 1 to 5, where 1 represents a negative sentiment and 5 represents a positive sentiment.

The work carried out in this project involved adapting the BERT model to the classification task and training it. Additionally, efforts were made to improve the results, and experiments were conducted using a custom implementation of the same architecture to evaluate whether a smaller model could achieve better or worse results than BERT.

The main challenge of this project was the development of the transformers architecture, including the encoder layer and the transformation of its output into probabilities to predict each score. Another significant challenge was the transformation of categorical data into numerical data so the model could process it. This implementation has enabled a deeper exploration of the attention mechanism, a key component in the advancement of artificial intelligence, providing a solid foundation for future work.

Agradecimientos

Antes de agradecer a las personas que me han ayudado a la realización de este trabajo, ya se lo he agradecido previamente porque decir “Gracias” no cuesta nada.

Me gustaría agradecer a mi pareja por la ayuda a lo largo del proyecto.

Me gustaría agradecer a mi director de proyecto, Dani, sin su ayuda y consejos no hubiese sido posible realizar este trabajo.

Me gustaría agradecer a mi familia por el apoyo durante el grado y durante el proyecto.

Me gustaría agradecer a mis amigos por los consejos sobre el proyecto, en especial a Javi.

Gracias a vosotros este proyecto ha sido posible.

Índice

1. Introducción	10
1.1. Contextualización	11
1.2. Conceptos	11
1.2.1. <i>Embedding</i>	11
1.2.2. <i>Transformers</i>	12
1.2.3. <i>Transfer Learning</i>	14
1.2.4. Aprendizaje supervisado	15
1.3. Identificación de problemas	15
1.4. Agentes implicados	16
2. Justificación	17
2.1. Estudio de soluciones existentes	17
2.2. Herramientas	18
3. Alcance	19
3.1. Objetivos	19
3.2. Requisitos	19
3.3. Obstáculos y riesgos	19
3.4. Metodología de trabajo	20
4. Metodología	21
4.1. Adquisición de datos	21
4.2. Modelo	26
5. Implementación	33
5.1. Implementación web scraping	33
5.2. Implementación interfaz revisión datos	34
5.3. Implementación modelos	35
5.3.1. Implementación recursos comunes	37
5.3.2. Implementación BERT	39
5.3.3. Implementación myBert	40
5.3.4. Implementación GRU	45
6. Entorno de trabajo	47
7. Experimentos	48
7.1. Datos revisados	49
7.1.1. Configuración	49
7.1.2. Resultados	50
7.2. Datos aumentados: revisados y sin revisar	54
7.2.1. Configuración	54
7.2.2. Resultados	56
7.3. Arquitectura GRU	60
7.3.1. Configuración	60

7.3.2. Resultados	61
7.4. Sinónimos	62
7.4.1. Configuración	62
7.4.2. Resultados	64
8. Conclusiones	70
8.1. Análisis de los resultados del proyecto	70
8.2. Futuro trabajo	70
8.3. Valoración personal	71
A. Planificación	74
A.1. Problemas durante el desarrollo	78
A.2. Costes	78
A.2.1. Costes de personal	78
A.2.2. Gastos de material	79
A.2.3. Coste total	80
B. Sostenibilidad	80
B.1. Autoevaluación	80
B.2. Dimensión económica	81
B.3. Dimensión medioambiental	81
B.4. Dimensión social	82

Índice de figuras

1.	Proceso del análisis de una reseña. [Elaboración propia]	10
2.	Proceso de transformación de vector de <i>tokens</i> a <i>word embedding context</i> . [4]	12
3.	Arquitectura de un modelo basado en <i>Transformers</i> . [5]	13
4.	Ejemplo de la multiplicación de las matrices key y query. [6]	14
5.	Fórmula del mecanismo de atención. [5]	14
6.	Categorías que ofrece la herramienta de Mapal OS. [7]	18
7.	Petición para obtener la lista de reseñas. [12]	21
8.	Enlace de la petición para obtener el identificador de un restaurante. [13]	21
9.	Proceso que realiza el <i>WebDriver</i> para la obtención de las reseñas de un restaurante. [Elaboración propia]	23
10.	Petición para obtener la lista de nombres de restaurantes. [Elaboración propia]	24
11.	Interfaz de selección de archivo Excel. [Elaboración propia]	24
12.	Interfaz de revisión de una reseña. [Elaboración propia]	25
13.	Pseudocódigo de la creación del vocabulario de <i>tokens</i> . [Elaboración propia]	27
14.	Arquitectura de la capa de <i>encoder</i> . [5]	29
15.	Arquitectura de GRU. [20]	30
16.	Pseudocódigo del cálculo de la función de pérdida. [Elaboración propia]	31
17.	Pseudocódigo de la obtención de datos. [Elaboración propia]	33
18.	Flujo de la implementación de la creación y entrenamiento del modelo BERT. [Elaboración propia]	36
19.	Flujo de la implementación de la creación y entrenamiento del modelo myBert. [Elaboración propia]	36
20.	Flujo de la implementación de la creación y entrenamiento del modelo GRU. [Elaboración propia]	37
21.	Gráfica de los modelos de BERT, congelando o entrenando sus capas. [Elaboración propia]	51
22.	Gráfica de la media de acierto en el conjunto de validación de los diferentes modelos de BERT eliminando o conservando las palabras vacías. [Elaboración propia]	52
23.	Gráfica de los tamaños del vocabulario y el acierto de validación del modelo myBert. [Elaboración propia]	54
24.	Gráfica aumentando el porcentaje del conjunto de datos revisados para el conjunto de entrenamiento con BERT entrenado en español. [Elaboración propia]	56
25.	Gráfica aumentando el porcentaje del conjunto de datos revisados para el conjunto de entrenamiento con myBert. [Elaboración propia]	57
26.	Gráfica variación del porcentaje del conjunto de datos revisados y sin revisar utilizando myBert. [Elaboración propia]	58

27.	Gráficas con el conjunto de datos revisados y sin revisar de la diferencia de etiquetas y su probabilidad con el modelo myBert. [Elaboración propia]	59
28.	Gráficas con el conjunto de datos revisados de la diferencia de etiquetas y su probabilidad con el modelo myBert. [Elaboración propia]	59
29.	Gráfica de los tamaños del vocabulario y el acierto de validación del modelo myBert. [Elaboración propia]	65
30.	Gráfica de los diferentes valores para el vector de representación de un <i>token</i> , cabezas de atención y vector de la capa <i>feed forward</i> para el entrenamiento de myBert.[Elaboración propia]	66
31.	Gráficas con la diferencia de las etiquetas reales y sus predicciones del modelo BERT. [Elaboración propia]	68
32.	Gráficas con la diferencia de las etiquetas reales y sus predicciones del modelo myBert. [Elaboración propia]	69
33.	Diagrama de Gantt realizado con <i>onlinegantt</i> [28]. [Elaboración propia]	77
34.	Tabla de tiempos de la planificación definitiva. [Elaboración propia]	77
35.	Tabla de gastos de personal. [Elaboración propia]	79
36.	Fórmula para calcular la amortización del hardware.[Elaboración propia]	79
37.	Tabla de gastos de la amortización de los recursos materiales. [Elaboración propia]	80
38.	Tabla de gastos totales del proyecto. [Elaboración propia]	80

Índice de Tablas

1.	Media del acierto y el error de cada conjunto para cada modelo. [Elaboración propia]	53
2.	Resultados del acierto y el error de cada conjunto para cada combinación. [Elaboración propia]	61
3.	Resultados del acierto y error para cada tamaño de lote. [Elaboración propia]	64
4.	Resultados del acierto y error para cada tamaño de lote. [Elaboración propia]	67

1. Introducción

La inteligencia artificial es un campo de la computación en constante crecimiento durante el siglo XXI. Los avances de la inteligencia artificial han posibilitado que las computadoras ejecuten modelos capaces de detectar y replicar patrones para interactuar con el usuario a través del lenguaje natural. Previo a la aparición de técnicas de *deep learning*, los motores de búsqueda también permitían un texto en lenguaje natural como entrada. El proceso de dicha implementación se basaba en técnicas clásicas de recuperación de información, la entrada se *tokenizaba* por palabra para posteriormente buscar los documentos donde se encontraba dicha palabra exactamente. Actualmente, las técnicas de *deep learning* han sustituido a la búsqueda del término en el documento permitiendo realizar búsquedas más complejas. Los métodos de *Natural Language Processing* son capaces de detectar relaciones gramaticales, es decir, permiten obtener un conocimiento más profundo sobre el contexto.

Las redes neuronales han reemplazado algoritmos y técnicas tradicionales en diversas áreas, debido a su gran capacidad de detección de patrones y a los resultados obtenidos en diversas tareas. Además, ha permitido automatizar tareas que antes requerían de supervisión humana, como el análisis de la polaridad de los comentarios para determinar su sentimiento. El trabajo está relacionado con dicha tarea, conocida como el análisis de sentimientos, donde la red neuronal recibe las reseñas publicadas en Google Maps con el objetivo de predecir la puntuación para cada reseña.

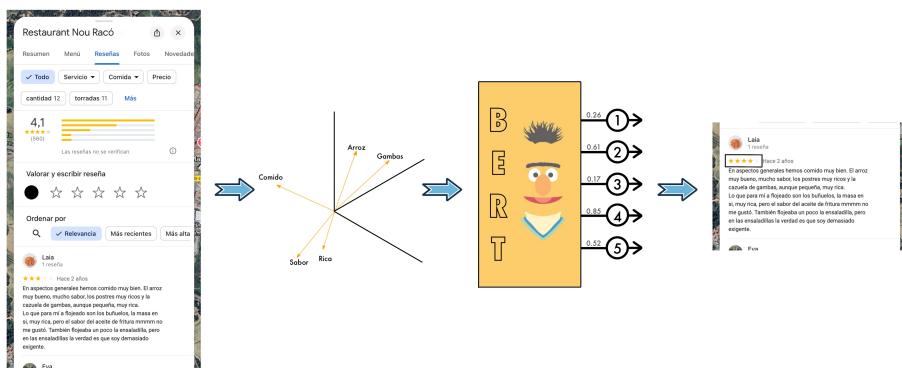


Figura 1: Proceso del análisis de una reseña. [Elaboración propia]

El proceso de obtención de nuevas puntuaciones para las reseñas seguirá los pasos ilustrados en la Figura 1. El modelo analizará el texto y determinará un número entre 1 y 5, el que considere que más se ajusta al sentimiento del comentario. En primer lugar, se obtendrán las reseñas de Google Maps, previo al procesamiento del modelo se realiza una transformación debido a que el modelo

sólo comprende datos numéricos. En segundo lugar, se envían a BERT [1], *Bi-directional Encoder Representation from Transformer*, modelo desarrollado por Google para el procesamiento del lenguaje natural y predecirá cuál es el valor más adecuado para el comentario. Por último, se actualiza la puntuación de la reseña.

El objetivo de este trabajo es diseñar una métrica que refleje de manera precisa la experiencia de visitar un establecimiento de comida, mediante el análisis de los sentimientos expresados en las reseñas de Google.

1.1. Contextualización

Este trabajo se encuentra dentro del marco de la especialidad de Computación de la Facultad de Informática de Barcelona, en concreto, en el campo de la inteligencia artificial. El proyecto surge con la idea de alinear las críticas gastronómicas con la realidad, con el objetivo de que los nuevos clientes puedan tomar una decisión sobre si visitar o no un restaurante. En caso que decidan asistir, se busca que la experiencia se ajuste a las expectativas generadas a partir de la puntuación y las reseñas.

Hasta el momento, se han desarrollado varios modelos capaces de analizar el sentimiento de textos, pero pocos se han enfocado en el ámbito de la restauración. Recientemente, TheFork ha propuesto una herramienta para analizar el mercado [2], pero dicha herramienta no se centra en la experiencia del cliente. Por ello, este trabajo propone una nueva herramienta para la evaluación de la calidad del establecimiento, de manera que tanto propietarios como nuevos clientes tengan una visión más realista de la situación del local.

1.2. Conceptos

A continuación se definen los conceptos básicos del trabajo, que serán utilizados repetidamente a medida de su desarrollo.

1.2.1. Embedding

El modelo no es capaz de procesar datos categóricos, como el texto, tan solo puede procesar datos numéricos. Por lo tanto, es necesario realizar una transformación previa al procesamiento del modelo mediante la técnica de *embedding* [3].

El *embedding* es un vector numérico que representa elementos como palabras o frases. Esta representación permite capturar la relación semántica entre elementos lingüísticos, es decir, si los significados de los elementos son similares, sus vectores serán próximos en el espacio vectorial. Además, los *embeddings* permiten realizar operaciones vectoriales para obtener relaciones semánticas. Por ejemplo, $\text{vec}(\text{"Madrid"}) - \text{vec}(\text{"Spain"}) + \text{vec}(\text{"France"})$ debería dar como resultado un vector cercano a $\text{vec}(\text{"París"})$.

Existen dos tipos de *embedding*: *word embedding* o *sentence embedding*, dependiendo de la tarea que se desea resolver es mejor uno u otro. Para este trabajo, se utilizará un *word embedding context*, ya que BERT está implementado con este tipo de *embedding*. Esta implementación ajusta la representación vectorial de las palabras con múltiples significados en función de su contexto.

Primeramente, se divide el texto recibido por espacios y se *tokeniza* cada palabra. La *tokenización* consiste en transformar las palabras en *tokens* que pertenecen al diccionario de BERT. En el caso que la palabra no esté completa en el diccionario, se sustituye por el prefijo más largo disponible y los sufijos correspondientes.

Una vez se obtiene el vector de *tokens*, se le suman tres *embeddings* a cada posición. El primer *embedding* es la representación vectorial del *token*, el segundo *embedding* contiene la información de la frase a la que pertenece el *token* y el tercer *embedding* representa la información posicional del *token* en el vector de entrada.

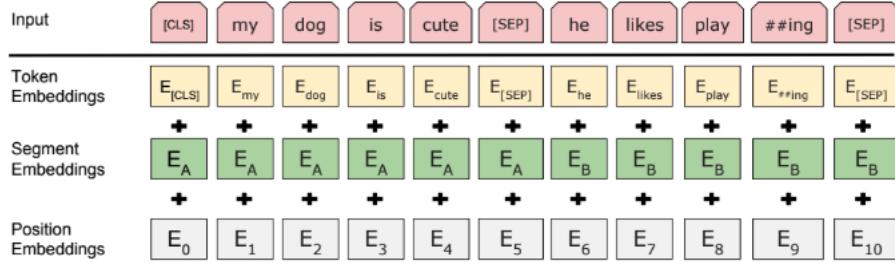


Figura 2: Proceso de transformación de vector de *tokens* a *word embedding context*. [4]

1.2.2. *Transformers*

Transformers es una arquitectura de red neuronal que emplea el mecanismo de atención, lo que permite calcular simultáneamente la importancia de cada palabra con respecto al resto y extraer información sobre las partes más importantes de la entrada. A diferencia de las redes neuronales recurrentes, que procesan la entrada secuencialmente y tienden a perder información a medida que avanzan por las capas debido al problema del *vanishing gradient*.

En la parte izquierda de la Figura 3 se encuentra el *encoder*, que está conectado con al *decoder* situado en la parte derecha. Para este trabajo, sólo se usará la parte de la izquierda de la arquitectura, ya que BERT está compuesto únicamente por *encoders* y la tarea de análisis de sentimiento tan solo requiere la información contextual de la entrada. El *decoder* tiene como objetivo generar una salida y se utiliza en tareas como la traducción de textos o la generación de resúmenes.

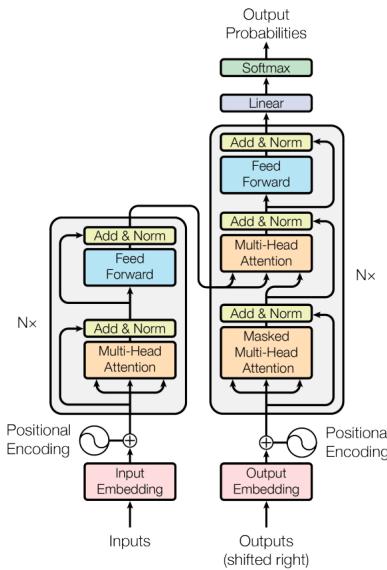


Figura 3: Arquitectura de un modelo basado en *Transformers*. [5]

El *encoder* se encarga de procesar la secuencia de entrada y generar una representación enriquecida mediante el mecanismo de atención, la salida se envía al *decoder*. Primero, se añade la información posicional a la secuencia de entrada. Luego, se aplica el mecanismo de atención, que añade información contextual a la secuencia. Finalmente, a través de la capa *Feed Forward*, se aplica una función no lineal a cada *token* con la finalidad de mejorar la capacidad de la red para capturar relaciones no lineales en los datos.

El *decoder* recibe la representación generada por el *encoder* y calcula las probabilidades de todos los *tokens* del diccionario del modelo para predecir la siguiente palabra de la secuencia. Los *embeddings* de salida se desplazan una posición para garantizar que las predicciones de la posición *i* dependan únicamente de las salidas en posiciones anteriores a *i*.

El *positional encoding* añade a cada *token* información de su posición en el vector de entrada mediante funciones oscilantes, como las sinusoidales y cosenoideales. Esto permite conocer la posición de los *tokens* sin necesidad de procesar la secuencia de manera secuencial.

El mecanismo de atención, clave en el éxito de los *transformers*, permite capturar relaciones contextuales de cada palabra con respecto al resto.

El proceso comienza con tres copias idénticas de la matriz de entrada a la capa, etiquetadas como *key*, *value* y *query*. Primero, se multiplica la matriz *query* por la matriz transpuesta *key*, como podemos observar en la Figura 4, donde cada elemento de la matriz resultante corresponde a la multiplicación de los vectores que representan los *tokens* de la secuencia. Luego, el resultado se divide por la raíz cuadrada de la longitud del *embedding* y se aplica la función *Softmax* por columna. Finalmente, el resultado se multiplica por la matriz *value*.

	<i>a</i>	<i>fluffy</i>	<i>blue</i>	<i>creature</i>	<i>roamed</i>	<i>the</i>	<i>verdant</i>	<i>forest</i>
	\vec{E}_1	\vec{E}_2	\vec{E}_3	\vec{E}_4	\vec{E}_5	\vec{E}_6	\vec{E}_7	\vec{E}_8
	\downarrow^{w_2}							
	\vec{Q}_1	\vec{Q}_2	\vec{Q}_3	\vec{Q}_4	\vec{Q}_5	\vec{Q}_6	\vec{Q}_7	\vec{Q}_8
<i>[a] $\rightarrow \vec{E}_1 \xrightarrow{w_k} \vec{K}_1$</i>	$\vec{K}_1 \cdot \vec{Q}_1$	$\vec{K}_1 \cdot \vec{Q}_2$	$\vec{K}_1 \cdot \vec{Q}_3$	$\vec{K}_1 \cdot \vec{Q}_4$	$\vec{K}_1 \cdot \vec{Q}_5$	$\vec{K}_1 \cdot \vec{Q}_6$	$\vec{K}_1 \cdot \vec{Q}_7$	$\vec{K}_1 \cdot \vec{Q}_8$
<i>[fluffy] $\rightarrow \vec{E}_2 \xrightarrow{w_k} \vec{K}_2$</i>	$\vec{K}_2 \cdot \vec{Q}_1$	$\vec{K}_2 \cdot \vec{Q}_2$	$\vec{K}_2 \cdot \vec{Q}_3$	$\vec{K}_2 \cdot \vec{Q}_4$	$\vec{K}_2 \cdot \vec{Q}_5$	$\vec{K}_2 \cdot \vec{Q}_6$	$\vec{K}_2 \cdot \vec{Q}_7$	$\vec{K}_2 \cdot \vec{Q}_8$
<i>[blue] $\rightarrow \vec{E}_3 \xrightarrow{w_k} \vec{K}_3$</i>	$\vec{K}_3 \cdot \vec{Q}_1$	$\vec{K}_3 \cdot \vec{Q}_2$	$\vec{K}_3 \cdot \vec{Q}_3$	$\vec{K}_3 \cdot \vec{Q}_4$	$\vec{K}_3 \cdot \vec{Q}_5$	$\vec{K}_3 \cdot \vec{Q}_6$	$\vec{K}_3 \cdot \vec{Q}_7$	$\vec{K}_3 \cdot \vec{Q}_8$
<i>[creature] $\rightarrow \vec{E}_4 \xrightarrow{w_k} \vec{K}_4$</i>	$\vec{K}_4 \cdot \vec{Q}_1$	$\vec{K}_4 \cdot \vec{Q}_2$	$\vec{K}_4 \cdot \vec{Q}_3$	$\vec{K}_4 \cdot \vec{Q}_4$	$\vec{K}_4 \cdot \vec{Q}_5$	$\vec{K}_4 \cdot \vec{Q}_6$	$\vec{K}_4 \cdot \vec{Q}_7$	$\vec{K}_4 \cdot \vec{Q}_8$
<i>[roamed] $\rightarrow \vec{E}_5 \xrightarrow{w_k} \vec{K}_5$</i>	$\vec{K}_5 \cdot \vec{Q}_1$	$\vec{K}_5 \cdot \vec{Q}_2$	$\vec{K}_5 \cdot \vec{Q}_3$	$\vec{K}_5 \cdot \vec{Q}_4$	$\vec{K}_5 \cdot \vec{Q}_5$	$\vec{K}_5 \cdot \vec{Q}_6$	$\vec{K}_5 \cdot \vec{Q}_7$	$\vec{K}_5 \cdot \vec{Q}_8$
<i>[the] $\rightarrow \vec{E}_6 \xrightarrow{w_k} \vec{K}_6$</i>	$\vec{K}_6 \cdot \vec{Q}_1$	$\vec{K}_6 \cdot \vec{Q}_2$	$\vec{K}_6 \cdot \vec{Q}_3$	$\vec{K}_6 \cdot \vec{Q}_4$	$\vec{K}_6 \cdot \vec{Q}_5$	$\vec{K}_6 \cdot \vec{Q}_6$	$\vec{K}_6 \cdot \vec{Q}_7$	$\vec{K}_6 \cdot \vec{Q}_8$

Figura 4: Ejemplo de la multiplicación de las matrices key y query. [6]

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Figura 5: Fórmula del mecanismo de atención. [5]

1.2.3. Transfer Learning

El *transfer learning* es una técnica que permite aprovechar el conocimiento de un modelo previamente entrenado para resolver una nueva tarea. Se utiliza principalmente porque facilita obtener buenos resultados sin destinar recursos a un entrenamiento desde cero. También existe la posibilidad de entrenar únicamente una parte del modelo o incluso el modelo completo, esta técnica se denomina *fine-tuning*.

En este trabajo se aprovechará el conocimiento que ofrece BERT, añadiendo y entrenando únicamente las capas densas al final del modelo. El objetivo es adaptar BERT a la tarea específica de análisis de sentimientos de reseña.

1.2.4. Aprendizaje supervisado

El aprendizaje supervisado es un método que utiliza un conjunto de datos etiquetados, donde cada muestra tiene una entrada y su correspondiente etiqueta. El objetivo es que el modelo sea capaz de generalizar y realizar predicciones precisas sobre nuevos datos, basándose en patrones aprendidos durante el entrenamiento.

El proceso consiste en ajustar los pesos del modelo a medida que procesa los ejemplos del conjunto de entrenamiento. Al realizar una predicción, el modelo compara la predicción con la etiqueta real del ejemplo y calcula el error. Este error se propaga a través del modelo utilizando el algoritmo de *Backpropagation*, que ajusta los pesos del modelo.

1.3. Identificación de problemas

Existe una relación entre el tamaño del modelo, la cantidad de datos de entrenamiento y la precisión del modelo. Es decir, cuanto más capas tenga el modelo y más datos procese durante el entrenamiento más porcentaje de acierto obtendrá el modelo. Sin embargo, esto también implica un mayor requerimiento de recursos computacionales, lo que plantea como principal problema la limitación en la capacidad de cómputo.

La limitación en la capacidad de cómputo influye en los tiempos de entrenamiento, lo que puede derivar en modelos con tiempos de entrenamiento excesivos. Por lo tanto, será crucial planificar las tareas de manera que se disponga de un margen para la obtención de los resultados y realizar tareas en paralelo al entrenamiento. Además, será crucial realizar pruebas para identificar la mejor configuración del modelo y así poder obtener mejores resultados.

El modelo será entrenado mediante el método de aprendizaje supervisado, lo que requiere el uso de datos etiquetados. Para este trabajo se revisarán manualmente todos los datos para garantizar una calidad mínima, dado que el modelo replica los patrones que detecta durante el entrenamiento, y si estos no cumplen una calidad necesaria el modelo no realizará predicciones precisas. En conclusión, el problema será el tiempo de revisión de dicho proceso, porque si el modelo requiere más datos se debe repetir el proceso de revisión para los datos nuevos.

1.4. Agentes implicados

La comprobación y ajuste de las críticas gastronómicas está dirigido tanto a los propietarios de los locales como a las personas que buscan un lugar donde comer. En general, la industria de la restauración es la que puede beneficiarse de este trabajo.

Por un lado, los propietarios de los restaurantes dependen de los clientes, por lo que necesitan conocer su feedback para tomar decisiones. La opinión de los clientes indicará si es necesario mejorar algún aspecto o si las cosas se están haciendo bien y deben mantenerse igual.

Por otro lado, tenemos a los clientes, que se dividen en dos tipos: aquellos que ya han vivido la experiencia y aquellos que están decidiendo vivirla. El cliente que ya ha experimentado la vivencia es quien proporciona el feedback sobre el servicio, el ambiente y la comida. Previamente, ese cliente fue parte del otro grupo, personas que leyeron las experiencias de otros comensales y decidieron visitar el restaurante. Las reseñas son clave para tener una percepción del local y tomar la decisión de ir o no ir.

2. Justificación

Aunque la inteligencia artificial no es un campo creado recientemente, su popularidad ha crecido significativamente en los últimos años. Actualmente, gran parte del interés de la comunidad informática se ha centrado en el desarrollo de nuevos modelos de inteligencia artificial, debido a su enorme potencial para automatizar una amplia variedad de tareas.

A continuación, se listan las soluciones existentes relacionadas con el análisis de sentimientos de restaurantes, y se nombran las herramientas utilizadas para el desarrollo del trabajo.

2.1. Estudio de soluciones existentes

Actualmente, en el mercado existen dos empresas que dicen ofrecer servicios de categorización de reseñas mediante inteligencia artificial: TheFork y Mapal OS. Ninguna de las dos empresas ofrece acceso gratuito a sus funcionalidades, por lo que el análisis sobre sus servicios se basará en la información proporcionada a través de sus páginas web o blog.

En agosto de 2024, TheFork publicó un blog donde anunció novedades [2], donde incluye el análisis de reseñas con inteligencia artificial. Sin embargo, el blog no menciona cuál es la fuente de datos utilizada, lo que sugiere que probablemente se trate únicamente de las reseñas publicadas en la propia plataforma. Además, en el blog no proporciona información sobre cómo realiza esta categorización, es decir, no menciona qué categorías proporcionará el modelo. En conclusión, esta solución todavía se encuentra en una fase *beta* y actualmente no se podría considerar como solución que resuelve satisfactoriamente el problema.

Por otro lado, Mapal OS se presenta como una empresa más orientada a proporcionar datos al cliente. Esta empresa va más allá y ofrece una clasificación de las reseñas basada en el contenido, según se muestra en las ilustraciones de su página web. No obstante, aunque la empresa afirma utilizar inteligencia artificial, también menciona en el mismo párrafo que se utilizan reglas basadas en *machine learning*. Esto genera incertidumbre sobre si realmente se aplica un análisis de sentimientos mediante inteligencia artificial o si simplemente se detectan palabras clave y realizan una clasificación mediante reglas. Por lo tanto, no se puede afirmar con certeza si esta solución realmente utiliza la misma tecnología que este trabajo o si el término se emplea únicamente para captar la atención del cliente.



Figura 6: Categorías que ofrece la herramienta de Mapal OS. [7]

Existen soluciones que podrían ser útiles para el público objetivo de este trabajo, pero no se puede afirmar que representen el estado del arte porque las herramientas no ofrecen un servicio gratuito. Es decir, no existe una solución como la que se propone en este trabajo, un modelo basado en *transformers* y entrenado con datos específicos de reseñas gastronómicas. Por lo tanto, se diseñará el mismo servicio con una implementación moderna para la categorización de reseñas de restaurantes.

2.2. Herramientas

La primera herramienta necesaria para este proyecto es Selenium [8], un entorno de pruebas para aplicaciones web. Sin embargo, para este trabajo Selenium se empleará como un entorno que permite ejecutar la web con el fin de extraer datos necesarios. En otras palabras, la herramienta será utilizada para desarrollar un proceso de *web scraping*.

La segunda herramienta es PyTorch [9], una biblioteca de código abierto para el aprendizaje automático. PyTorch está diseñada para la creación y el desarrollo de redes neuronales, es ampliamente utilizada en investigación debido a su flexibilidad para la creación de los modelos. Los tensores de PyTorch, estructuras que representan matrices multidimensionales, facilitan la optimización de los cálculos en la GPU. El propósito de utilizar PyTorch en este proyecto es crear y entrenar un modelo encargado de predecir el sentimiento de las reseñas.

PyTorch permite la modificación de un modelo pre-entrenado como BERT, la decisión de seleccionar BERT se debe a sus exitosos resultados para tareas de comprensión de texto, debido a su entrenamiento de manera bidireccional obteniendo información de la parte previa y posterior para cada token.

La tercera herramienta es Vue.js [10], un *framework* de JavaScript diseñado para la construcción de interfaces gráficas y *Single Page Applications*. La decisión de usar Vue.js en este trabajo se debe al conocimiento previo del autor, se desarrollará la interfaz de revisión de datos mediante Vue.js.

3. Alcance

A continuación, se definen los objetivos que se buscan alcanzar en este proyecto, los requisitos necesarios para que se cumplan y los posibles riesgos y obstáculos que pueden aparecer a lo largo del desarrollo del trabajo.

3.1. Objetivos

El objetivo principal de este trabajo es conseguir un modelo de inteligencia artificial que realice predicciones fiables de las reseñas de Google Maps, para desarrollar una métrica que refleje la realidad de los restaurantes. Para ello, se evaluará el rendimiento del modelo pre-entrenado BERT, entrenado con los datos revisados manualmente, analizando la diferencia de la puntuación real e inferida de las reseñas y la diferencia entre la puntuación mediana real e inferida de los restaurantes.

3.2. Requisitos

Para considerar que el objetivo se han conseguido, se tienen que cumplir ciertos requisitos.

El requisito es que el modelo realice predicciones fiables, se considera que un modelo es veraz si alcanza un acierto del conjunto de validación igual o superior al 75 %. Esta métrica se calcula dividiendo el número de ejemplos que el modelo predice correctamente la etiqueta entre el número de ejemplos total.

3.3. Obstáculos y riesgos

La implementación de un modelo de inteligencia artificial puede presentar varios obstáculos y riesgos que pueden dificultar el desarrollo del proyecto.

Uno de los objetivos de este trabajo es lograr que el modelo analice de manera autónoma el sentimiento de las reseñas, es decir, que la red neuronal clasifique el texto de forma similar a como lo haría una persona, sin necesidad de revisión humana.

Uno de los riesgos asociados a la creación del modelo es que no logre detectar correctamente el sentimiento de las reseñas, lo que podría comprometer su capacidad para clasificarlas correctamente. Este problema puede estar relacionado con diversos factores, como la calidad o la cantidad de los datos. Por lo tanto, será necesario realizar múltiples pruebas, variando la cantidad de los datos y variando la configuración de los modelos, para detectar los factores que afectan al rendimiento del modelo.

El entrenamiento de algunos modelos requiere una gran capacidad computacional, aunque durante este trabajo no se entrenará grandes modelos del lenguaje, es indispensable contar con recursos de GPU.

Un riesgo asociado al entrenamiento es que su duración sea excesiva, esto implicaría un retraso en la obtención de los resultados y, en consecuencia, un retraso en las tareas siguientes.

3.4. Metodología de trabajo

Con el objetivo de realizar un desarrollo eficaz del trabajo y realizar un seguimiento adecuado de la realización de los objetivos, se aplicará la metodología en cascada aprendida durante el grado.

Esta metodología se basa en realizar secuencialmente los pasos de diseñar, desarrollar y verificar cada tarea del proyecto. La metodología ha sido escogida debido a la dependencia entre tareas. Por ejemplo, los datos necesitan la verificación de la herramienta que obtiene las reseñas de manera automática y el modelo necesita la verificación de los datos para el entrenamiento.

Para verificar la correcta obtención de los datos, será necesario comprobar tanto el funcionamiento adecuado de la herramienta encargada de obtener los datos como el de la interfaz de revisión. La interfaz debe permitir visualizar los comentarios y clasificarlos según la acción correspondiente.

Para verificar el modelo, será necesario alcanzar al menos un 75 % de acierto de los ejemplos del conjunto de validación.

Adicionalmente, en colaboración con el director del proyecto, se realizarán reuniones cada dos semanas, o semanalmente en el caso que haya suficiente material que comentar. Esto permitirá al director estar al tanto del progreso y debatir dudas y soluciones.

4. Metodología

4.1. Adquisición de datos

En este apartado se detalla el proceso para obtener reseñas de Google Maps, describiendo las diversas subtareas y las herramientas utilizadas.

A lo largo del apartado se hace referencia a Google Maps Platform [11], un servicio de pago que ofrece la posibilidad de realizar peticiones a *APIs* que proporcionan información relacionada con Google Maps. La plataforma ofrece un período de prueba de 30 días, durante el cual se realizaron todas las peticiones mencionadas a continuación. Al disponer de un período gratuito, se consideró este servicio como una opción viable para automatizar la obtención de datos.

El objetivo principal de esta tarea es automatizar la obtención de reseñas publicadas en Google Maps. La primera alternativa consiste en implementar un programa que realice peticiones al servicio de Google, ya que esta solución es más rápida de implementar y, si cumple con el objetivo, no será necesario explorar otras opciones. La segunda alternativa consiste en desarrollar una herramienta de *web scraping*, simulando una solicitud a Google Maps mediante un navegador y extrayendo la información almacenada en el HTML correspondiente.

Primero se investigaron las *APIs* disponibles en Google Maps Platform. Entre las diferentes opciones ofrecidas, se encuentra la petición de la Figura 7 que devuelve un objeto con un campo que contiene las reseñas del lugar buscado.

Después de seleccionar la API, se realiza una prueba para verificar que las reseñas incluyan el comentario y su respectiva puntuación. La solicitud de la Figura 7 requiere el identificador de la ubicación, por lo tanto, previamente se realiza una petición con el nombre del restaurante a la Figura 8 para obtener su identificador. Finalmente, se selecciona el nombre de un restaurante para realizar la prueba del programa, ejecutando las solicitudes de forma secuencial.

```
$ GET  
https://mybusiness.googleapis.com/v4/accounts/{accountId}/locations/{locationId}/reviews
```

Figura 7: Petición para obtener la lista de reseñas. [12]

```
https://places.googleapis.com/v1/places:searchText
```

Figura 8: Enlace de la petición para obtener el identificador de un restaurante. [13]

Sin embargo, el resultado obtenido no fue el esperado. La petición de la Figura 8 tan solo proporciona 5 reseñas, independientemente de la ubicación.

Google Maps Platform cobra según la cantidad de solicitudes realizadas [14], lo que implica que si el proyecto requiere más datos, una vez finalizado el período de prueba, el coste del proyecto aumentaría significativamente. Por ejemplo, para obtener 10.000 reseñas adicionales, sería necesario realizar 2000 peticiones a la Figura 7 y 2000 a la Figura 8, lo que incrementaría en 100 dólares el coste del proyecto.

Finalmente, esta alternativa fue descartada, debido a que si en el futuro es necesario entrenar al modelo con más datos, el coste del proyecto aumentaría cada vez que se requieran datos nuevos.

La siguiente alternativa consiste en desarrollar una herramienta que emule la experiencia de realizar una búsqueda en Google Maps mediante un navegador. Para el desarrollo de dicha herramienta se utiliza Selenium, un entorno de pruebas para aplicaciones web que permite configurar y ejecutar un *WebDriver* [15]. El *WebDriver* puede realizar peticiones y ofrecer la misma experiencia de usuario que al interactuar directamente con el navegador, esto es posible debido a que el driver obtiene el HTML dinámico de la web.

El proceso, que muestra la Figura 9, comienza realizando una petición a <https://www.google.es/maps> para obtener el HTML dinámico. A continuación, se inserta el nombre del restaurante que se desea buscar en el componente del buscador y se ejecuta la acción asociada al componente, lo que genera la solicitud para obtener la página del restaurante en Google Maps. Una vez completada la búsqueda, el *WebDriver* obtiene el código de la página del restaurante y realiza la acción del clic en el botón “Reseñas” para acceder a la sección donde se encuentran las reseñas.

Una vez en la página de las reseñas, el programa accede a los elementos HTML que contienen la información necesaria de cada reseña: la puntuación, la presencia de una foto adjunta y el texto. Estos datos se almacenan en una matriz, donde cada fila corresponde a una reseña y las columnas representan los diferentes tipos de información. Posteriormente, este objeto se exporta como un archivo Excel.



Figura 9: Proceso que realiza el *WebDriver* para la obtención de las reseñas de un restaurante. [Elaboración propia]

El elemento HTML donde se encuentran todas las reseñas, inicialmente contiene 20 reseñas, ya que la página web limita la carga de información en el cliente y solo realiza peticiones al servidor cuando es necesario. Por ello, el programa accede a la sección de las reseñas y ejecuta la acción de *scroll* hasta el límite permitido por el *viewport*, para provocar que la web cargue 20 reseñas adicionales.

Este proceso se repite hasta obtener la mayor cantidad de reseñas posible. Google permite al usuario visualizar un máximo de 360 reseñas, incluso si el restaurante tiene más de 360 valoraciones. Por lo tanto, el programa finalizará cuando alcance el límite de 360 o cuando haya obtenido todas las reseñas del restaurante, es decir, cuando el número de reseñas obtenidas deje de variar.

Finalmente, se exportan todas las reseñas obtenidas y el *WebDriver* cierra la instancia del navegador.

En este punto, solo queda automatizar la obtención de los nombres de los restaurantes. Durante la implementación de esta alternativa, la prueba gratuita de Google Maps Platform aún no había expirado, por lo que los nombres se obtuvieron a partir de la petición de la Figura 10. Esta petición requiere como parámetros la latitud y la longitud de una ubicación [16] y el radio del perímetro máximo. El número máximo de resultados que devuelve la petición es 20.

Para obtener 10000 reseñas adicionales con esta alternativa, solo se requieren 3 solicitudes a la Figura 10, ya que cada petición devuelve 20 nombres de restaurante y cada restaurante proporciona 360 reseñas. Por lo tanto, esta alternativa realiza 3.997 peticiones menos que la alternativa anterior y aumenta el coste del proyecto en 0,07 dólares.

```

params = {
    'location': f'{lat},{long}',
    'radius': 1000, # Radio de búsqueda en metros
    'type': 'restaurant',
    'key': self.api_key
}
response = requests.get('https://maps.googleapis.com/maps/api/place/nearbysearch/json' , params=params)

```

Figura 10: Petición para obtener la lista de nombres de restaurantes. [Elaboración propia]

Finalmente, se integra la solicitud de la Figura 10 con la herramienta implementada con Selenium para lograr la automatización completa del proceso de obtención de reseñas de Google Maps.

Después de obtener los datos, se inicia la fase de revisión, que implica una evaluación manual de los datos. Esta revisión es esencial, ya que el modelo aprende los patrones de los datos procesados durante el entrenamiento para predecir nuevos ejemplos.

Como se ha mencionado anteriormente, los datos se exportan en un archivo Excel, lo cual dificulta su lectura y modificación. Para solucionar esto, se optó por desarrollar una interfaz sencilla que permita leer rápidamente el comentario, modificar la puntuación sin necesidad de escribir y añadir una marca que indique si un comentario contiene suficiente información gramatical para ser utilizado en el entrenamiento o si debe ser descartado.

Primero, se muestra la interfaz de la Figura 11, donde se hace clic en el botón “Elegir archivos” que abre el gestor de archivos del dispositivo donde se está ejecutando el programa. Luego, se selecciona el archivo Excel que contiene los datos y se confirma la selección haciendo clic en el botón “Subir archivo”, para que el programa comience a procesar el fichero.



Figura 11: Interfaz de selección de archivo Excel. [Elaboración propia]

Después de confirmar la subida del archivo, se muestra la interfaz de la Figura 12. La interfaz esta compuesta por un área de texto que contiene el último comentario pendiente de revisión del archivo, un texto que indica si el comentario contenía una foto y un botón de selección, donde la opción marcada corresponde a la puntuación original.

Sitio muy concurrido. Aun así, teníamos reserva a las 14:30 y nos hemos sentado a esa hora. Comida rica y con la proporción justa y correcta. El arroz con leche muy bueno. Hay ruido en el local porque está siempre muy lleno. Experiencia muy recomendable

Tiene foto?	Valoración
<input type="checkbox"/> Si	<input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input checked="" type="radio"/> 5
Eliminar	Guardar
Cerrar archivo	

Figura 12: Interfaz de revisión de una reseña. [Elaboración propia]

Una vez leído el comentario, se toma la decisión de modificar o mantener la puntuación de la reseña, seleccionando la opción correspondiente del botón de radio.

La puntuación final se determina según un criterio que evalúa el contenido de la reseña, siendo el factor más determinante la comida, ya que, al fin y al cabo se trata de una experiencia gastronómica. Por ejemplo, si la opinión sobre la comida es negativa, la puntuación final oscila entre 1 y 2 estrellas. Si la comida es mediocre o buena, la puntuación final oscila entre 3 y 4 estrellas. En el caso que la comida sea excelente, generalmente, la puntuación final es de 5 estrellas.

Otro factor que se tiene en cuenta para visitar un restaurante es el servicio. Una atención irrespetuosa al cliente se clasifica con 1 estrella. Aunque no es el elemento principal de la experiencia, no es agradable recibir un trato desagradable y es un factor que empaña la experiencia.

La mayoría de reseñas revisadas tienen una puntuación de 1, 4 o 5 estrellas. Esto se debe a que las personas dedican tiempo a dejar una buena reseña como muestra de agradecimiento por una buena experiencia, o una mala reseña para advertir a otros que no es un sitio recomendable. La diferencia entre una calificación de 4 y 5 estrellas es mínima, y depende de detalles como el precio o el ambiente.

A continuación, se considera si el comentario es útil para el modelo. En el caso que se decida conversar la reseña para el entrenamiento del modelo, se hace clic en el botón “Guardar” y en caso contrario se hace clic en el botón “Eliminar”. Ambos botones añaden una etiqueta a la fila de la reseña indicando la acción seleccionada. Posteriormente, se muestra el siguiente comentario sin revisar, y así sucesivamente hasta que todos los datos estén revisados.

Al final se encuentra el botón “Cerrar archivo”, cuya función es exportar a un archivo Excel los datos con las etiquetas añadidas durante el proceso. Este botón está diseñado para permitir al usuario detener el proceso de revisión y retomarlo más adelante. La información revisada se acumula, permitiendo que el proceso de revisión sea paulatino.

4.2. Modelo

En este apartado se describe el proceso para entrenar un modelo de inteligencia artificial, detallando cada etapa desde su creación hasta su entrenamiento.

El objetivo principal de esta tarea es conseguir un modelo capaz de realizar predicciones fiables del sentimiento de las reseñas. A lo largo del trabajo, se han realizado pruebas con dos modelos basados en la arquitectura de los *transformers*: BERT, desarrollado por Google y myBert, una implementación propia. También, se han realizado pruebas con un modelo basado en la arquitectura de redes neuronales recurrentes: un modelo GRU. Los procesos de carga de datos y entrenamiento son los mismos para los tres modelos.

El proceso inicial consiste en cargar los datos almacenados en el fichero Excel generado a partir de la revisión de los datos obtenidos del *web scraping*. El programa lee el contenido del fichero mediante una función de la librería Pandas, a partir de la ruta donde se encuentra el archivo, lee su contenido y lo transforma en una matriz. Tras cargar los datos, se seleccionan los datos que han sido etiquetados como ‘Revisado’, ya que serán los datos que se utilizarán para entrenar al modelo. De cada dato, se conserva únicamente la información de la puntuación y el texto, descartando el resto.

A continuación, se aplica el preprocesamiento adecuado para la realización de cada experimento y se seleccionan la misma cantidad de datos por clase, siendo dicha cantidad la de la clase que contiene menos datos. El balanceo de clases es fundamental para evitar sesgos en el modelo, ya que procesar muchos datos de una misma clase tendería a predecir dicha clase.

Por último, los datos se dividen en tres conjuntos: entrenamiento, validación y evaluación. El conjunto de entrenamiento se utiliza para ajustar los pesos del modelo, el más importante y el más grande. Generalmente, se le asigna el 70 % o más de los datos, mientras que el porcentaje restante se reparte equitativamente

entre el resto de conjuntos. El conjunto de validación se utiliza para evaluar el modelo durante el entrenamiento y tomar decisiones sobre los hiperparámetros. Este conjunto no afecta a los pesos del modelo, lo que significa que el modelo los procesa como si fueran datos nuevos. El conjunto de evaluación se utiliza después del entrenamiento para evaluar el rendimiento del modelo con datos que no ha visto, simulando su comportamiento en un entorno real.

El modelo no puede procesar datos categóricos como el texto y por lo tanto, es necesario realizar un proceso de *tokenización* antes del entrenamiento. Los tres modelos realizan este proceso de la misma manera, pero el modelo BERT carga el tokenizador de la librería Hugging Face [17] con su vocabulario asociado, mientras que el resto cargan el *tokenizador* con el vocabulario generado a partir de las palabras de los datos obtenidos.

La Figura 13 muestra el pseudocódigo del proceso de creación del vocabulario, que comienza añadiendo todos los caracteres del texto de los comentarios etiquetados como ‘Revisado’ y crea las separaciones de con los *tokens* generados de cada palabra. Una vez creado el vocabulario inicial y las divisiones, se realiza la búsqueda del par de *tokens* con el valor más alto de la fórmula siguiente: frecuencia de la aparición de la pareja entre la multiplicación de frecuencias de cada *token* en el par. El par con la puntuación más alta se añade al vocabulario y se actualiza la separación de cada palabra. Este proceso se repite hasta que el tamaño del vocabulario sea igual al indicado o hasta que todas las palabras sean representadas por un único *token*.

Algorithm 1: Algoritmo para la creación de vocabulario

Input: Datos
Output: Vocabulario de tokens

```

1 Palabras ← {};
2 foreach dato en Datos do
3   | if dato tiene la etiqueta Revisado " then
4   |   | Añade dato.palabra a Palabras;
5   | end
6 end
7 Vocab ← {tokens especiales} ∪ {caracteres iniciales de cada palabra};
8 Separaciones ← {palabra : [tokens] | para cada palabra en Palabras};
9 while longitud(Vocab) < long_max do
10  | Scores ← calcula:
11    |   | 
$$\frac{\text{frecuencia\_par}}{\text{frecuencia\_token1} \times \text{frecuencia\_token2}}$$

12    |   | para cada par de tokens secuenciales;
13    |   | if Scores =  $\emptyset$  then
14    |   |   | termina;
15    |   | end
16    |   | Par_max ← max(Scores);
17    |   | Actualiza Separaciones con Par_max;
18    |   | Añade Par_max a Vocab;
19 end

```

Figura 13: Pseudocódigo de la creación del vocabulario de *tokens*. [Elaboración propia]

Al crear el *tokenizador*, se obtienen los *tokens* del vocabulario correspondiente y se les asigna un identificador único. El proceso de *tokenización* de un comentario comienza eliminando los caracteres chinos [18] y los emoticonos [19], después se separa el texto por espacios. Tras obtener el vector del comentario separado, se transforma cada palabra a *tokens* que pertenecen al diccionario. Para cada palabra, se busca el sufijo más largo de la palabra que aparezca en el vocabulario y se repite el mismo proceso para la subpalabra restante.

Una vez obtenido el vector de *tokens*, se añaden los *tokens* especiales que indican el inicio y el final de la frase. En el caso que el vector sea menor que la longitud máxima de entrada permitida por el modelo, se añade al final del vector el *token* de padding hasta que alcance la longitud máxima. Esto es necesario para agrupar datos y que sean procesados en paralelo, ya que deben tener la misma longitud. En caso contrario, si el vector excede la longitud máxima se seleccionan los primeros *tokens*, desde el índice cero hasta el límite.

Luego, se calcula el vector de la máscara de atención, el cual contiene un 1 en la posición “i” si en dicha posición del vector de *tokens* no es un token de padding, en caso contrario, la máscara contiene un 0. Finalmente, se sustituyen en el vector, los *tokens* por sus respectivos identificadores.

El DataLoader emplea el *tokenizador* para transformar el texto y añade el resultado con su puntuación correspondiente, encapsulándolos en un objeto para facilitar su procesamiento. Posteriormente, estos objetos se agrupan en lotes del tamaño especificado, creando mini conjuntos de datos para el procesamiento en paralelo y así optimizar los recursos computacionales.

El modelo de BERT no se crea desde cero, ya que la librería Hugging Face [17] permite cargar el modelo preentrenado, tan solo es necesario adaptarlo a la tarea específica. Al principio, se configuran los hiperparámetros necesarios: la decisión de congelar o no los pesos del modelo, el nombre del modelo a cargar, el número de neuronas de salida de la última capa y el porcentaje de neuronas que se desactivan durante el entrenamiento.

Luego, se configura el modelo con las capas lineales que se conectarán al final de BERT. La primera capa posterior a BERT debe tener 768 neuronas de entrada, correspondientes al tamaño del vector que representa un *token*, mientras que la última capa debe tener 5 neuronas de salida, una para cada clase. El dato se procesa secuencialmente por las capas del modelo, al procesarlo recibe el vector de identificadores de *tokens* y su máscara de atención.

MyBert es un modelo basado en la parte izquierda de la arquitectura de *transformers*, Figura 14. Uno de los beneficios de desarrollar una implementación propia es la flexibilidad que ofrece, como la posibilidad de modificar el número de capas de *encoder*, algo que no es posible con BERT. Sin embargo, a diferencia de BERT, myBert requiere realizar un entrenamiento debido a que sus pesos se inicializan de forma aleatoria.

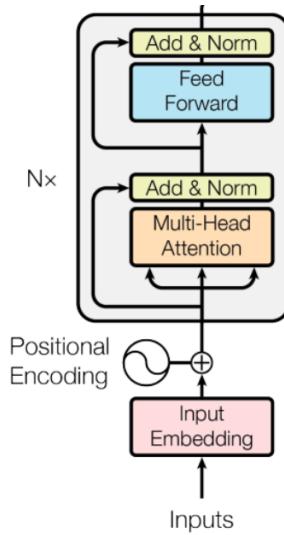


Figura 14: Arquitectura de la capa de *encoder*. [5]

Al instanciar el modelo, se especifican los tamaños de entrada y salida de cada capa del modelo. También se define el número de capas de *encoder*. Una vez definida la estructura base, que simula la funcionalidad de BERT, se configuran las capas lineales que se conectarán a la última capa de *encoder*. La primera capa lineal toma como entrada el tamaño del vector que representa un *token*, definido al inicializar myBert, mientras que la última capa debe tener 5 neuronas de salida, una para cada clase. El dato se procesa de forma secuencial a través de las capas del modelo, utilizando el vector de identificadores de *tokens* y su máscara de atención.

El modelo GRU es una variante de la arquitectura de redes neuronales recurrentes, Figura 15, donde la entrada a cada capa está compuesta por el elemento “t” de la secuencia y el vector de salida de la capa “t-1”. Esta arquitectura puede operar de manera unidireccional o bidireccional. En el modelo unidireccional, la secuencia únicamente se procesa hacia delante, mientras que en el modelo bidireccional se combina el resultado de la secuencia en ambas direcciones. La bidireccionalidad resulta especialmente útil para captar mejor el contexto de toda la secuencia.

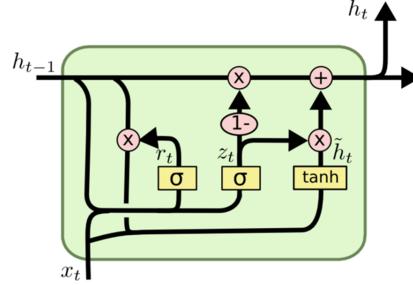


Figura 15: Arquitectura de GRU. [20]

Este modelo aprovecha la clase que ofrece PyTorch para construir una arquitectura GRU, lo que significa que no se implementan manualmente las operaciones que realiza la arquitectura, sino que se configuran los parámetros mediante el constructor que define la librería [21]. Los parámetros son: el tamaño del vector de entrada, el tamaño de los vectores de salida de cada capa, el número de capas GRU y la elección entre procesamiento bidireccional o unidireccional.

Por último, se inicializa el estado oculto inicial, un vector de ceros. El estado oculto es la representación que mantiene la red sobre las entradas procesadas hasta el momento. Tras la clase GRU de PyTorch, se añaden capas lineales con el mismo objetivo que en los modelos anteriores: asegurar que la salida del modelo tenga tantas neuronas como clases.

Tras la creación del modelo y la división de los datos en tres conjuntos, se lleva a cabo el último proceso: el entrenamiento. Antes de iniciar este proceso, se define la función de pérdida, el optimizador y la condición de parada. Si esta condición no se cumple, el entrenamiento se detiene cuando alcance el número máximo de iteraciones.

La función de pérdida cuantifica el error entre las predicciones del modelo y los valores reales. Su objetivo es guiar al modelo a obtener predicciones correctas. Si la función de pérdida obtiene un valor alto, indica que el modelo está realizando predicciones significativamente incorrectas, mientras que en el caso contrario las predicciones se asemejan a las etiquetas reales.

En este trabajo, se utiliza CrossEntropyLoss [22] como función de pérdida, diseñada específicamente para tareas de clasificación. La Figura 16 muestra el pseudocódigo del cálculo de la función de pérdida, que convierte la salida del modelo en probabilidades, donde cada posición corresponde a una clase. El índice del vector con la probabilidad más alta es la clase que el modelo predice. Para calcular el error, se niega el resultado de aplicar el logaritmo a la probabilidad

asignada a la clase real. El logaritmo genera valores bajos cuando la probabilidad es alta y valores altos cuando la probabilidad es baja, exactamente el comportamiento deseado en una función de pérdida. Esto se debe a que, si la probabilidad asignada a la etiqueta real es alta, el modelo tenderá a predecir como resultado esa etiqueta y la función de pérdida ajusta los pesos con un valor pequeño.

Algorithm 2: Cálculo de la función de pérdida

Input: Vector_id_tokens, Máscara de atención
Output: Error de la salida del modelo

```

1 Salida.modelo ← model(Vector_id_tokens, Máscara de atención);
2 N ← longitud(Salida.modelo);
3 suma_total ←  $\sum_{j=0}^N e^{\text{Salida.modelo}[j]}$ 
4 Probabilidades ←  $\emptyset$ 
5 for  $i \leftarrow 0$  to  $N$  do
6   |
   |   Probabilidades[ $i$ ] ←  $\frac{e^{\text{Salida.modelo}[i]}}{\text{suma_total}}$ 
7 end
8 Prob_max ← max(Probabilidades);
9 Error ← -log(Prob_max)

```

Figura 16: Pseudocódigo del cálculo de la función de pérdida. [Elaboración propia]

El optimizador es el algoritmo que ajusta los pesos del modelo con el objetivo de minimizar la función de pérdida. Utiliza el algoritmo del descenso del gradiente, restando al valor del parámetro la derivada de la función de pérdida en función del parámetro multiplicado por la tasa de aprendizaje.

El optimizador empleado a lo largo del trabajo es Adam, el cual ajusta automáticamente la tasa de aprendizaje para cada peso del modelo según su historial de actualizaciones. Adam permite avanzar más rápidamente en direcciones que minimizan la función de pérdida y más lentamente en direcciones menos relevantes. Por lo tanto, logra una convergencia más rápida.

La condición de parada evalúa el error de iteración sobre el conjunto de validación, con el objetivo de evitar continuar realizando más iteraciones si el modelo no mejora, lo que indicaría que ha alcanzado un valle de rendimiento. Se establece un número máximo de iteraciones elevado para que el modelo se detenga por la condición de parada. Se fija el número de 1000, ya que se considera suficientemente elevado para que el modelo acabe el entrenamiento por la condición de parada.

El entrenamiento se detiene cuando transcurren un número determinado de iteraciones consecutivas sin registrar un error menor que el mínimo error alcanzado. En el caso que haya mejorado, el contador de iteraciones se reinicia a 0 y se guardan los pesos para cargarlos nuevamente antes de finalizar el entrenamiento.

En cada iteración del entrenamiento, se procesan los datos del conjunto de entrenamiento y los datos del conjunto de validación para calcular las métricas de error y el porcentaje de acierto de ambos.

El conjunto de entrenamiento se utiliza para entrenar al modelo. Para cada lote de datos, se extraen los vectores de entrada, las máscaras de atención y las etiquetas correspondientes. Primero, se restablecen los gradientes del modelo a cero y se envían los datos al modelo para obtener las salidas. Luego, se calcula el error mediante las salidas y las etiquetas, lo que permite calcular las derivadas del error con respecto a los parámetros del modelo [23]. A continuación, se aplica el algoritmo del descenso del gradiente para actualizar los pesos. Finalmente, se obtienen las predicciones del modelo y se comparan con las etiquetas reales para calcular la precisión del modelo.

El conjunto de validación realiza el mismo procedimiento que el conjunto de entrenamiento, con la diferencia de que durante el procesamiento de los datos de validación no se calculan los gradientes. Este conjunto se utiliza para evaluar el rendimiento del modelo al procesar datos que no modifican al modelo, las métricas de este conjunto sirven para la toma de decisión de los hiperparámetros.

5. Implementación

5.1. Implementación web scraping

En este apartado se detalla el flujo de la obtención de datos a partir del *web scraping*. Se han implementado dos procesos que corresponden a la obtención de los nombres de los restaurantes y la información extraída de la web Google Maps. Los procesos se han desarrollado en la clase RequestGoogleAPI y GoogleReviewsTrain.

Tal y como se muestra en la Figura 17, primero se envían las coordenadas de latitud y longitud a la función search_lat_long para obtener los nombres de los restaurantes ubicados a un kilómetro de distancia. Segundo, se verifica si ya se ha intentado buscar previamente el nombre del restaurante, evitando la duplicidad de los datos. En el caso que sea la primera vez que se intenta obtener las reseñas de el restaurante, se envía a la función obtener_datos_google el nombre para recopilar las reseñas. Los resultados obtenidos se almacenan en un archivo Excel.

Algorithm 3: Obtención de datos de Google Maps

Input: Lista de coordenadas (*latitud, longitud*) de ubicaciones
Output: Archivo Excel con los datos de los restaurantes

```
1 Request ← RequestGoogleAPI();
2 Google_rev ← GoogleReviewsTrain();
3 foreach coordenada en Coordenadas do
4   if coordenada no ha sido vista then
5     | Restaurante ← Request.search.lat.long(coordenada);
6     | foreach restaurante en Restaurantes do
7       |   | datos ← Google_rev.obtener_datos.google(restaurante);
8       |   | Añadir datos al archivo Excel;
9     | end
10   end
11 end
```

Figura 17: Pseudocódigo de la obtención de datos. [Elaboración propia]

La clase RequestGoogleAPI se encarga de realizar las peticiones al servicio de Google Maps Platform. Los atributos asociados a esta clase son: la clave para autenticar las peticiones, el enlace al que se envían las peticiones y el radio de búsqueda. Las funciones de la clase son las siguientes:

- **search_lat_long:** Recibe como parámetros una latitud y una longitud. Estas coordenadas se añaden al objeto que contiene las propiedades de la petición y se realiza la solicitud. La función retorna el resultado de la petición, en el caso que haya fallado, devuelve una lista vacía.

La clase GoogleReviewsTrain se encarga de crear un driver que simula una instancia de un navegador para realizar una solicitud a la página de Google Maps. Su función principal es extraer las reseñas a través de los componentes HTML. Los atributos asociados a esta clase son: los identificadores de los elementos HTML que contienen la información necesaria y una lista con los nombres de las columnas del archivo Excel. Las funciones de la clase son las siguientes:

- **obtener_datos_google:** Recibe como parámetro el nombre de un restaurante y realiza una solicitud a Google Maps para obtener el HTML dinámico. Inserta el nombre del restaurante en el buscador y navega a la página de dicho restaurante en Google Maps. Luego, ejecuta la acción de clicar en el botón de “Reseñas” para acceder a este apartado. Dentro del apartado de reseñas, realiza un desplazamiento continuo en el contenedor de toda la información, para cargar más reseñas. Finalmente, retorna un objeto bidimensional que agrupa todos los datos obtenidos, donde cada fila representa una reseña y cada columna corresponde a un tipo de información.
- **get_driver:** Configura el *driver* y descarga la versión del *ChromeDriver* compatible con el sistema. Retorna el *driver* ensamblando con la configuración correspondiente.
- **wait_elem:** Función que detiene la ejecución del código hasta que el *driver* encuentra el componente con el identificador que recibe como parámetro. Si el componente no se encuentra después de 20 segundos, la función lanza una excepción.
- **obtener_comentarios_train:** Función que extrae la información de la sección que almacena las reseñas. Para cada reseña, verifica que no sea de Tripadvisor, comprueba si tiene asignado un contenedor de fotos. Posteriormente, obtiene el texto y la puntuación de los elementos que los almacenan. Si una reseña no tiene texto, no se guarda. Retorna en un objeto bidimensional el conjunto de todas las reseñas obtenidas, donde cada fila es una reseña.

5.2. Implementación interfaz revisión datos

En esta sección se detalla el funcionamiento de la interfaz de revisión de los datos, es decir, se describen las acciones asociadas a cada componente. El beneficio de usar Vue [10] para el desarrollo de una interfaz sencilla radica en permitir unificar el código HTML y Javascript en un mismo archivo.

Lo primero que muestra la interfaz es un formulario compuesto por un input para seleccionar un fichero Excel y un botón. El input almacena el archivo seleccionado en una variable, mientras que el botón inicia el proceso de lectura del buffer, para transformarlo en una matriz. La función vinculada al botón lee

la primera hoja del archivo y ejecuta una función que recibe un objeto XLSX y devuelve un *array* de objetos. Por último, indica a la interfaz que el archivo ha sido leído correctamente y se llaman a las funciones necesarias para insertar los datos en los componentes correspondientes.

Primero, se realiza una búsqueda del primer comentario que no tenga una etiqueta que indique que ha sido revisado, es decir, que la fila no contenga los elementos ‘Revisado’ ni ‘Eliminado’. Luego, se accede a la fila mediante el índice obtenido y se asignan los valores a cada componente.

El área de texto mostrará el contenido de la columna “Review”, el campo que indica si la reseña tiene foto mostrará un texto de “Si” o “No” y el botón de radio seleccionará el mismo número que la puntuación de la columna.

Los botones laterales capturan el evento de cuando se realiza un clic sobre ellos, lo que activa una función que obtiene el contenido de los componentes de la interfaz a través de las variables ligadas a cada componente. Según el botón que haya recibido el evento, se añade una etiqueta u otra: el botón verde añade la etiqueta ‘Revisado’, indicando que se usará la reseña para el entrenamiento, y el botón rojo añade la etiqueta ‘Eliminado’. Después de crear la nueva fila de datos se accede a la posición de la reseña en revisión, mediante la variable índice, y se actualiza la fila con la nueva lista.

A continuación, se actualizan los componentes previamente mencionados con los valores de la siguiente fila, es decir, la fila en la posición del índice más uno. En el caso que el índice sea igual o mayor al número de filas, todos los datos han sido revisados y se exportan a un archivo Excel.

Cuando el botón de ‘Cerrar archivo’ captura el evento de clic, ejecuta la función que convierte la matriz de datos en un archivo Excel. Utilizando la biblioteca de XLSX, se crea una hoja de cálculo, se añaden los datos y se le asigna un nombre para exportar el archivo a la carpeta de descargas del dispositivo.

5.3. Implementación modelos

En esta sección se detalla el flujo del entrenamiento de los tres modelos utilizados, para posteriormente detallar las funciones que llevan a cabo este proceso. En las tres figuras siguientes, los recuadros naranjas simbolizan una implementación común, es decir, los módulos que realizan estas implementaciones son los mismos para los tres. Los recuadros grises son implementaciones propias de cada modelo.

La Figura 18 muestra el flujo para realizar el entrenamiento del modelo BERT. Se le aplica el preprocesamiento correspondiente a los datos, para posteriormente sean *tokenizados* y agrupados por tamaño de lote. Una vez se completa la carga de datos, se procede a la creación del modelo BERT. Por último, se utiliza el modelo y los lotes de datos para efectuar el entrenamiento, que proporciona métricas como resultado del proceso anterior.

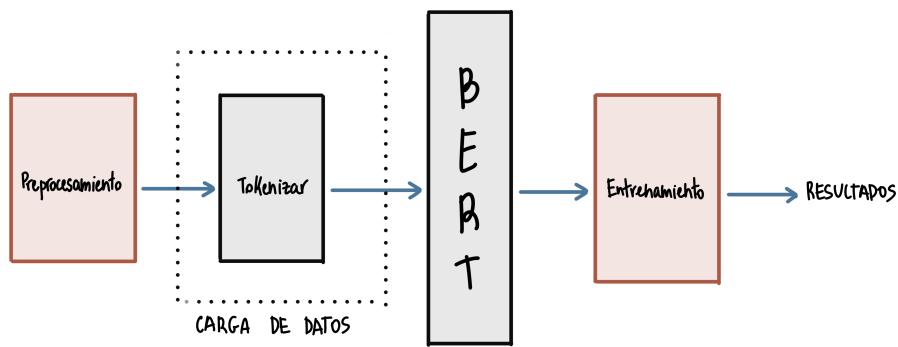


Figura 18: Flujo de la implementación de la creación y entrenamiento del modelo BERT. [Elaboración propia]

La Figura 19, muestra el flujo del entrenamiento de myBert. Como se ha mencionado al inicio del apartado, se aplica el mismo preprocesamiento que el resto. Posteriormente, se crea el vocabulario de *tokens* a partir de las palabras de los datos y sus respectivos identificadores. A continuación, se utilizan los identificadores de *tokens* generados para transformar los datos a vectores numéricos. Seguidamente, se crea y entrena el modelo que proporciona unas métricas de evaluación.

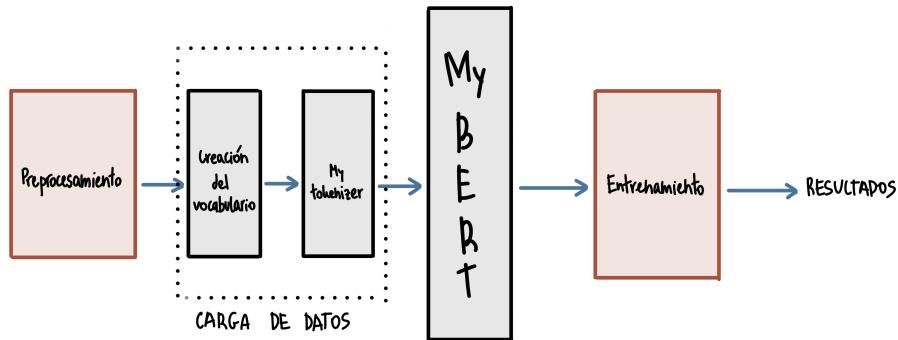


Figura 19: Flujo de la implementación de la creación y entrenamiento del modelo myBert. [Elaboración propia]

El último flujo, representado en la Figura 20, realiza los mismos procesos que myBert, pero con la creación de un modelo basado en la arquitectura GRU.

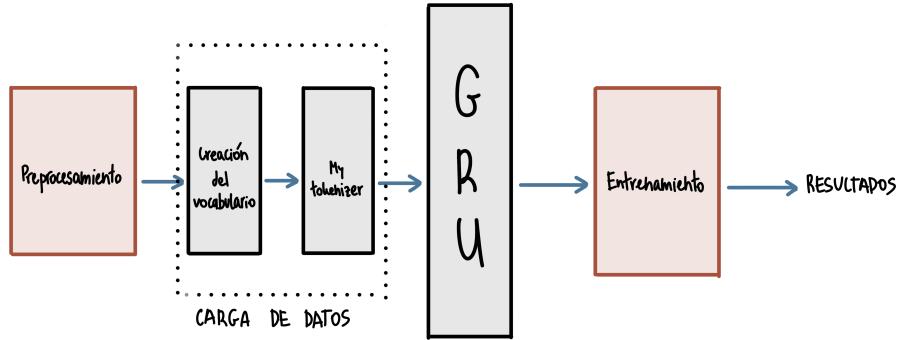


Figura 20: Flujo de la implementación de la creación y entrenamiento del modelo GRU. [Elaboración propia]

5.3.1. Implementación recursos comunes

En este apartado se describen los recursos utilizados por los tres modelos, detallando la implementación del preprocesamiento, el empaquetado de datos, la condición de parada y el proceso de entrenamiento.

El archivo Preprocessing implementa el preprocesamiento de los datos, aplicando los procesos necesarios para que puedan ser utilizados por el modelo. Al ejecutarse, descarga las palabras vacías de la librería nltk [24], para almacenar en una variable global las palabras vacías en español. Las palabras vacías son términos comunes en un idioma que, debido a su alta frecuencia, no aportan información útil para el análisis de texto. Las funciones del archivo son:

- **get_min_num_class:** Función que devuelve una cantidad específica de datos asociados a una clase determinada. Filtra las filas del *Dataframe* que coincidan con el valor de la clase en la columna especificada y devuelve la cantidad de datos solicitada.
- **remove_stop_words:** Elimina las palabras vacías del texto recibido como parámetro. Divide el texto en palabras utilizando espacios como separadores, recorre la lista resultante y elimina aquellas palabras que estén presentes en la variable global. Devuelve la lista filtrada uniendo las palabras con espacios.
- **preprocessing_dataframe:** Recibe como parámetro un *Dataframe* con los datos etiquetados como ‘Revisado’ y un booleano que determina si se deben eliminar las palabras vacías o no. Primero, resta uno a los valores de la columna correspondiente a la puntuación, ya que el modelo requiere

etiquetas en el rango de 0 al número total de clases menos uno. Luego, obtiene la cantidad de datos de la clase menos representada y selecciona esa misma cantidad para todas las clases, equilibrando el conjunto de datos. Por último, si el booleano es *True*, aplica la función `remove_stop_words` a la columna que contiene las reseñas.

El archivo DataLoader implementa el proceso de *tokenizar* y empaquetar los datos en conjuntos de tamaños de lote. Los atributos asociados a la clase son: la lista de las reseñas, la lista de las puntuaciones, el *tokenizador*, la longitud máxima de una reseña y la decisión de incluir o no el texto original en el objeto. La función de la clase es:

- **`__getitem__`:** Función que accede a los elementos mediante un índice, *tokeniza*, empaqueta el resultado y lo devuelve a la misma posición. El índice accede a las listas de las reseñas y la puntuación y le envía el texto al *tokenizador* para que lo convierta en un vector de números. La configuración que le indica al *tokenizador* es la siguiente: añadir los *tokens* especiales, que todas las secuencias tengan la misma longitud, que trunque si supera la longitud máxima, que devuelva la máscara de atención y que devuelva el resultado en tensores de PyTorch. Por último, se añade a un diccionario el vector de identificadores de *tokens*, la máscara de atención y la puntuación. Devuelve el diccionario creado.

El archivo Callback implementa la condición de parada del entrenamiento, que se encarga de monitorear el error del conjunto de validación y decidir cuándo detener el entrenamiento. Los atributos asociados a esta clase son: el número máximo de iteraciones consecutivas sin mejora, la decisión de restaurar o no los pesos del menor error obtenido, el número de iteraciones consecutivas sin mejora, el menor error obtenido durante el entrenamiento y los pesos correspondientes a la iteración del error mínimo. La función de la clase es:

- **`__call__`:** Función que evalúa si el entrenamiento debe detenerse. En primer lugar, comprueba si es la primera iteración del entrenamiento. Si es así, asigna el error recibido al atributo correspondiente al error mínimo y guarda los pesos actuales del modelo en el atributo que almacena los mejores pesos. En segundo lugar, comprueba si el error que recibe es menor que el mínimo visto hasta ahora. Si es así, actualiza los atributos correspondientes y reinicia el contador de iteraciones sin mejora. Si es mayor, suma uno al contador de iteraciones consecutivas sin mejora. En tercer lugar, comprueba si el contador de iteraciones consecutivas es igual o mayor al máximo y en el caso que se cumpla, restaura los mejores pesos almacenados e indica que el entrenamiento debe detenerse.

El archivo Training implementa el proceso del entrenamiento del modelo y evalúa el rendimiento con los datos de validación. La función del entrenamiento modifica los pesos del modelo con el objetivo de minimizar la función de pérdida, calcula la pérdida promedio y la precisión promedio. La función de validación únicamente calcula la pérdida promedio y la precisión del modelo con los datos de validación. Las funciones del archivo son:

- **model_train:** Función que realiza el entrenamiento del modelo. Al inicio, se activa el modo entrenamiento y se inicializan a cero las variables que almacenan el error y el número de aciertos. La función recorre cada lote de datos, extrae el vector de identificador de *tokens* y sus respectivas máscaras de atención. Luego, se ponen a cero los gradientes y se envía la información al modelo para que realice predicciones. Una vez se obtiene la salida del modelo, se compara con las etiquetas reales aplicando la función de pérdida. A continuación, se calcula el gradiente de los parámetros del modelo involucrados en el cálculo y se actualizan los pesos de dichos parámetros. Por último, se suma la pérdida del lote a la pérdida total del entrenamiento y el número de datos predichos correctamente en la iteración al acumulado. Las etiquetas que predice el modelo son los índices con el valor más elevado del vector. La función devuelve la suma del error acumulado entre el número de lotes y la suma de los aciertos entre el número de datos.
- **model_eval:** Función que evalúa el comportamiento del modelo. Al inicio, se activa el modo de evaluación y se inicializan a cero las variables que almacenan el error y el número de aciertos. Además, se asigna la lista vacía a la variable que almacena las diferencias entre etiquetas. Previo al recorrido de los datos, se desactiva el cálculo de gradientes. Luego, recorre y extrae la información de los datos, se envía al modelo el vector de identificador de *tokens* y su respectiva máscara de atención. Una vez se obtiene la salida del modelo, se compara con las etiquetas reales aplicando la función de pérdida. A continuación, se suma la pérdida del lote a la pérdida total del entrenamiento y se suma el número de datos que el modelo ha predicho correctamente. Además, calcula la diferencia en valor absoluto de las etiquetas reales y las etiquetas predichas y la diferencia de sus respectivas probabilidades. La función devuelve el error promedio de los datos, la precisión promedio y la lista de diferencias.

5.3.2. Implementación BERT

En este apartado se describe la implementación de BERT adaptado a la tarea de análisis de sentimiento, conectando capas lineales a la salida de BERT.

El archivo Model implementa la creación del modelo BERT, que define el flujo de datos a través de las capas del modelo. Los atributos asociados a la clase son: el nombre del modelo pre-entrenado de BERT en Hugging Face [17], la decisión que indica congelar o entrenar los pesos de BERT, el número de

neuronas de salida del modelo y el porcentaje de neuronas que no se entrena n. Al crear la clase, se carga el modelo pre-entrenado. Despu s se recorren todas las capas de BERT, indicando al atributo “requires_grad” si se deben entrenar o congelar los pesos, seg n la decis n recibida como par metro. La \'unica capa que siempre se entrena es la *pooler*, que procesa el primer *token*, ya que el modelo en espa ol inicializa aleatoriamente los pesos de esta capa. Por \'ultimo, se crearan las cuatro capas lineales que procesar n la salida de BERT, que reducir n la dimensionalidad de las representaciones hasta un vector de tama o igual que el n mero de clases. BERT devuelve un vector de tama o 768, las tres primeras capas reducen la dimensionalidad a la mitad y la \'unica transforma el vector de longitud 96 a 5. La func n de la clase es:

- **forward:** Func n que define el flujo de los datos a trav s del modelo. Recibe como par metro los identificadores de los *tokens* del texto y la m scara de atenci n. Se env a la informaci n a BERT y debido a que cada vector representativo contiene informaci n agregada del resto, del resultado se selecciona el vector que representa el primer *token*. Las capas lineales procesan secuencialmente dicho vector, transform ndolo hasta un vector de dimensi n igual al n mero de clases. Devuelve el resultado, sin normalizar, de la salida de la \'unica capa, una matriz de tama o [batch_size, num_labels].

5.3.3. Implementaci n myBert

En este apartado se describen las implementaciones de la creaci n del vocabulario propio, el proceso de *tokenizaci n* y la implementaci n de la arquitectura de *transformers*.

El archivo VocabModelo implementa la creaci n del vocabulario de *tokens* en base al contenido de los datos. Los atributos asociados a la clase son: los signos de puntuaci n, los caracteres chinos y emoticonos a eliminar, la frecuencia de cada palabra del *corpus*, el vocabulario inicial de *tokens* y las divisiones iniciales de las palabras en *tokens*. Las funciones asociadas a la clase son:

- **get_freq_words:** Func n que calcula el n mero de apariciones de la palabra en el *corpus*. Recibe un listado de textos por par metro. Recorre la lista, separa cada texto en espacios y recorre las palabras del texto. Actualiza el n mero de apariciones de las palabras, sumando uno al contador correspondiente. Devuelve el diccionario de palabras y apariciones.
- **get_vocab_ini:** Func n que genera el vocabulario inicial y las divisiones iniciales de las palabras. Recorre las palabras que se encuentran en el diccionario de frecuencias, a ade al vocabulario el primer caracter de cada palabra y la concatenaci n de dos almohadillas con cada uno del resto de caracteres. Despu s, ordena alfab ticamente el vocabulario y a ade al inicio los *tokens* especiales. A continuaci n, crea un diccionario con las

palabras del *corpus* y su correspondiente división en *tokens*. Devuelve el vocabulario y el diccionario de divisiones.

- **transform_txt:** Función que realiza la limpieza del texto. Recibe un texto como parámetro. Primero, elimina los caracteres chinos, los emoticonos, los signos de puntuación deseados y los dobles espacios. Luego, transforma todos los caracteres en minúsculas y devuelve la lista de palabras separadas por espacio.
- **new_token:** Función que concatena dos *tokens*. Recibe como parámetro un par de *tokens*. Elimina las almohadillas del segundo *token* en caso que tenga. Devuelve la concatenación del resultado del segundo *token* con el primero.
- **calc_pair_score:** Función que calcula el par de *tokens* que obtiene la puntuación más elevada. Al inicio, crea dos diccionarios, uno almacenará cada *token* y su frecuencia y el otro almacenará cada par secuencial de *tokens* generados y su frecuencia. La función recorre el diccionario de palabras y su número de apariciones. Para cada lista de divisiones comprueba si la lista contiene un único *token* y si es así, actualiza el diccionario de *tokens* individuales con el número de apariciones. En el caso que la palabra se divida en más de un *token*, se crean pares de *tokens* consecutivos y se actualiza la frecuencia del diccionario de par de *tokens*. Además, se suma la frecuencia de cada *token* individual. Por último, se calcula la puntuación de todos los pares de *tokens* generados, siendo el resultado de la división de la frecuencia del par entre la multiplicación de frecuencias de cada *token* del par. Devuelve el diccionario donde la clave son las parejas de *tokens* consecutivos y los valores son las puntuaciones de la fórmula.
- **recal_splits:** Función que actualiza las divisiones con el nuevo *token* creado. Recibe como parámetro el par de *tokens* que se acaba de añadir al diccionario. Recorre todas las divisiones de las palabras del *corpus* y cuando encuentra los *tokens* de manera secuencial que recibe por parámetro, actualiza dicha lista con el resultado de new_token. En el caso que se modifique la lista de divisiones, se actualiza el diccionario que contiene las palabras y las divisiones.
- **create_vocab:** Función que realiza el proceso de creación del vocabulario, que combina los *tokens* más frecuentes hasta obtener un tamaño máximo de vocabulario. Recibe como parámetro el tamaño máximo del vocabulario. Mientras el tamaño del vocabulario actual sea menor que el máximo, calcula las puntuaciones de cada par de tokens mediante calc_pair_score. Si esta función devuelve la lista vacía, todas las palabras del vocabulario se representan como un único *token*, por lo tanto, no se pueden crear más *tokens* y termina la creación del vocabulario. En caso contrario, se obtiene el par con la puntuación más elevada y se envía a recal_splits para actualizar las divisiones. Por último, añade la concatenación de *tokens* resultante al vocabulario. Devuelve un listado de *tokens* que representa el vocabulario.

El archivo Tokenizer implementa el proceso de transformar las reseñas en representaciones numéricas, para que el modelo sea capaz de procesarlas. Los atributos asociados a la clase son: el diccionario que contiene los *tokens* y su correspondiente identificador, el diccionario anterior invirtiendo las claves y los valores, los signos de puntuación, los caracteres chinos y los emoticonos a eliminar. Las funciones asociadas a la clase son:

- **load_vocab:** Función que carga el vocabulario y genera los identificadores de los *tokens*. Recibe como parámetro la ruta del archivo. Lee el contenido y devuelve la lista donde cada posición corresponde a un *token* del vocabulario. Devuelve un diccionario con los *tokens* y sus identificadores, donde cada identificador es el índice de la lista donde se encuentra.
- **convert_tokens_to_ids:** Función que convierte una lista de *tokens* en una lista de identificadores. Recibe como parámetro la lista de *tokens*. Para cada posición se sustituye el *token* por su identificador. Devuelve la lista de identificadores.
- **convert_ids_to_tokens:** Función que convierte una lista de identificadores en una lista de *tokens*. Recibe como parámetro la lista de identificadores. Para cada posición se sustituye el identificador por su *token*. Devuelve la lista de *tokens*.
- **transform_txt:** Función que realiza la limpieza del texto. Recibe un texto como parámetro. Primero, elimina los caracteres chinos, los emoticonos, los signos de puntuación deseados y los dobles espacios. Luego, transforma todos los caracteres en minúsculas y devuelve la lista de palabras separadas por espacio.
- **tokenize_word:** Función que *tokeniza* una palabra. Recibe como parámetro una palabra. Mientras la palabra contenga caracteres, busca el prefijo más largo que exista en el vocabulario, reduciendo el tamaño del prefijo hasta encontrar una subcadena que pertenezca al vocabulario. Si no se encuentra ninguna, devuelve el *token unknown*. Si se encuentra un prefijo, se añade a la lista de *tokens* y se elimina de la palabra para procesar el resto. En el caso que todavía queden caracteres, se añaden dos almohadillas al inicio del resto de la palabra para indicar que no es un *token* independiente y se realiza el mismo proceso. Devuelve la lista de *tokens* en los que se divide la palabra.
- **tokenize:** Función que *tokeniza* un texto completo. Recibe como parámetro el texto. Primero, envía el texto a *transform_txt* para obtener el texto limpio y en forma de lista. Luego, se recorre el resultado obtenido y se concatenan los resultados de enviar palabra por palabra a *tokenize_word*. Devuelve la lista de *tokens* que representan al texto.

- **encode_plus**: Función que transforma un texto a un formato adecuado para el modelo, generando los vectores numéricos que representan el texto y la máscara de atención. Recibe como parámetro el texto, la decisión de añadir los *tokens* especiales, la máxima longitud de los vectores y la decisión de devolver la máscara de atención. Primero se envía el texto a *tokenize* para recibir la lista de *tokens* correspondientes. Si se decide añadir los *tokens* especiales, se incluye al inicio de la lista resultante de *tokens* el *token* CLS y al final el *token* SEP. Si se debe devolver la máscara de atención, se genera un vector donde las posiciones tendrán valor 0 si la misma posición en el vector de *tokens* es de *padding* y 1 en caso contrario. Si ambos vectores superan la longitud máxima, recibida como parámetro, se trunca hasta el número que indique el límite. Por último, se transforma el vector de *tokens* a un vector con sus correspondientes identificadores mediante *convert_tokens_to_ids*. Devuelve el vector de identificadores de *tokens* y la máscara de atención si se indica.

El archivo Model contiene las implementaciones para la construcción de un modelo siguiendo el mismo fundamento que la arquitectura transformers. A continuación, se detalla cada clase.

La clase PositionalEncoding se encarga de añadir información posicional a las representaciones de los *tokens*, permitiendo procesar la secuencia de manera paralela. Los atributos asociados a la clase son: el tamaño de los vectores que representan a un *token*, la longitud máxima de la secuencia y el dispositivo en el que se realizan los cálculos. Al crear la clase, se crea una matriz de ceros que representa la información posicional con las dimensiones siguientes: longitud de la secuencia y tamaño del vector que representa a un *token*. Además, se crea la matriz de posiciones, donde cada fila tendrá una única dimensión con su correspondiente índice de fila. Finalmente, se genera un vector con valores del rango 0 hasta el tamaño que representa un *token*, de dos en dos. Cada valor se actualiza multiplicando por $\log(10000)$ y dividiendo entre el tamaño del vector representativo. Posteriormente, al resultado obtenido se le aplica la función exponencial con el signo invertido. A continuación, se actualizan las columnas pares de la matriz de la información posicional, con el resultado de aplicar la función seno a la multiplicación de los índices de posiciones por el vector. Se realiza el mismo procedimiento para las columnas impares, pero aplicando la función coseno. Por último, se añade una dimensión a la matriz posicional para que la suma se ajuste al tamaño de lote. La función asociada a la clase es:

- **forward**: Función que añade información posicional a cada vector que representa un *token*. Recibe como parámetro la secuencia de vectores representativos. Devuelve la suma de los vectores representativos y la información posicional para cada elemento de la secuencia.

La clase MultiHeadAttention se encarga de aplicar el mecanismo de atención para que el modelo capture las relaciones de la secuencia de entrada y asigne la importancia a cada parte. La clase únicamente se crea si el tamaño del vector

que representa un *token* es divisible entre el número de cabezas de atención. Los atributos asociados a la clase son: cuatro capas lineales con el tamaño del vector representativo en la entrada y salida y la dimensión de la cabeza de atención. Las funciones asociadas a la clase son:

- **forward:** Función que define el procesamiento de las entradas: *query*, *key* y *values*. Recibe como parámetros las entradas mencionadas y la máscara de atención. Las entradas son procesadas por su correspondiente capa lineal, que aplica un redimensionamiento de [tamaño_lote, seq_len, vec_rep] a [tamaño_lote, cabeza_aten, seq_len, dimensión_aten]. Se envía a la función scale_dot_product_att los resultados de las tres entradas redimensionadas y la máscara de atención. Al resultado de la función se le aplica un redimensionamiento para volver al tamaño original, [tamaño_lote, seq_len, vec rep]. Devuelve la salida de la última capa lineal.
- **scale_dot_product_att:** Función que implementa la Figura 5. Recibe como parámetros las matrices de *query*, *key*, *value* y el vector de la máscara de atención. Multiplica la matriz *query* por la transpuesta de la matriz *key* y divide el resultado entre la raíz cuadrada de la dimensión de la cabeza de atención. En el caso que se proporcione una máscara, se ajusta a las dimensiones de la matriz resultante y se sustituyen los ceros por un valor negativo muy grande. El objetivo de la sustitución es que la *Softmax* le asigne valores cercanos a cero, evitando las divisiones entre cero. A continuación, se aplica una *Softmax* a cada columna de la matriz. Devuelve la multiplicación de la matriz *value* y la matriz obtenida como resultado de la *Softmax*.

La clase FeedForward se encarga de aplicar una transformación no lineal a los datos con el objetivo de que el modelo aprenda representaciones más complejas. Los atributos asociados a la clase son: una capa lineal que aumenta la dimensión del vector representativo, una capa lineal que restablece la dimensionalidad original del vector y una función de activación *ReLU*, que transforma los valores negativos a cero y deja los valores positivos sin cambio. La función asociada a la clase es:

- **forward:** Función que aumenta la dimensión del vector que representa a un *token*, aplica una transformación no lineal y reduce la dimensión del vector al tamaño original. Recibe como parámetro la secuencia de vectores representativos. Primero, pasa por la capa lineal que aumenta la dimensión de los vectores. Luego, se aplica la función *ReLU* a todos los vectores y pasa por la segunda capa lineal. Devuelve la salida de la segunda capa lineal.

La clase MyBertLayer se encarga de replicar el funcionamiento de una capa de encoder de la arquitectura de *transformers*, siguiendo el flujo representado en la Figura 14. Los atributos asociados a la clase son: una capa MultiHeadAttention, una capa FeedForward y dos capas de normalización. La función asociada a la clase es:

- ***forward***: Función que define cómo se procesarán los datos. Recibe la secuencia de vectores de representación y la máscara de atención. Primero, envía a la capa de atención la secuencia de entrada para cada una de las tres consultas y la máscara de atención. Segundo, se suma la entrada inicial con la salida de la capa de atención y se envía a la capa de normalización. Tercero, envía el resultado de la normalización a la capa de FeedForward. Finalmente, suma la salida de la capa de FeedForward al resultado de la normalización y se envía a la segunda capa de normalización. Devuelve el resultado de la segunda capa de normalización.

La clase MyBert se encarga de conectar secuencialmente las clases mencionadas anteriormente y añade capas lineales a la salida para adaptar la arquitectura de *transformers* a la tarea específica. Los atributos asociados a la clase son: la raíz cuadrada del tamaño del vector representativo, una capa de *embedding*, una capa de PositionalEncoding, una lista de capas de MyBertLayer, una capa de normalización y cuatro capas lineales que reducirán la dimensionalidad de la salida de la última MyBertLayer. La función asociada a la clase es:

- ***forward***: Función que define el flujo de los datos a través de las capas. Recibe como parámetro el vector de identificadores de *tokens* y la máscara de atención. El vector de identificadores pasa por una capa de *embedding* que transforma números en vectores numéricos, permitiendo capturar relaciones más complejas. Los vectores resultantes se multiplican por la raíz cuadrada del tamaño de un vector representativo. A continuación, se procesa por la capa que añade información posicional y por todas las capas de MyBertLayer. Igual que BERT, la entrada de las capas lineales es el vector representativo que corresponde al primer *token* de la secuencia. Este se procesa secuencialmente por las capas lineales. Devuelve el resultado, sin normalizar, de la salida de la última capa, una matriz de tamaño [batch_size, num_labels].

5.3.4. Implementación GRU

En este apartado se describe la implementación de una arquitectura GRU, basada en redes neuronales recurrentes, adaptada a la tarea de clasificación para generar predicciones de los diferentes tipos de clase.

El archivo Rnn_GRU implementa la creación del modelo GRU, que define el flujo de datos a través de las capas del modelo. Los atributos asociados a la clase son: el tamaño del vector que representa a un *token*, el tamaño del vector que representa el estado oculto, el número de capas GRU, el número de neuronas de

salida del modelo, la decisión de una arquitectura bidireccional o unidireccional y el porcentaje de neuronas que no se entrenan. Al crear la clase, se instancia la clase GRU de PyTorch [21] definiendo la configuración con los tamaños de vectores recibidos, el número de capas, indicando que la primera dimensión será el tamaño de lote, el porcentaje de neuronas que se desactivarán y la decisión de bidireccional o unidireccional. Por último, se crean las dos capas lineales que reducirán la dimensionalidad de la salida de la GRU. La primera reduce el tamaño de representación de la salida a la mitad y la segunda reduce el tamaño al número de clases. La función de la clase es:

- **forward:** Función que define el flujo de datos a través del modelo. Recibe como parámetro los identificadores de los *tokens* del texto. Primero, se añade una nueva dimensión en el índice de la dimensión 1, para indicar al modelo que procese por separado cada vector que representa a un *token*. A continuación, se crea el estado oculto inicial con ceros y se envía, junto con el vector de representaciones, a la clase GRU de PyTorch. Por último, las capas lineales procesan secuencialmente la salida de GRU, que contiene la información de toda la secuencia. Devuelve el resultado de la salida de la última capa: una matriz de tamaño [tamaño_lote, num_clases].

6. Entorno de trabajo

Este apartado describe el entorno de trabajo empleado para la realización de los experimentos, explicando tanto el *hardware* como el *software* utilizados.

Los experimentos se dividen en dos subtareas: la implementación del programa que genera las combinaciones para un modelo y la ejecución del programa. La tarea de implementación se puede realizar sin necesidad de un ordenador con GPU. Para esta tarea se ha utilizado un ordenador sin tarjeta gráfica, específicamente, un *MacBook Air*. Por otro lado, la ejecución del programa realiza las operaciones del entrenamiento del modelo y sí requiere una tarjeta gráfica. En este trabajo, se ha usado un ordenador con un procesador *Intel i9-10900KF*, una memoria RAM de 64 GB y una *NVIDIA GeForce RTX 3060 Ti* con 8 GB de memoria, un ancho de banda de 448 GB/s y una frecuencia de 1410 MHz.

Un contenedor de Docker [25] es el entorno de *software* donde se realizan los experimentos, es un *software* que virtualiza un contenido específico en una instancia del sistema operativo. A diferencia de una máquina virtual, un contenedor de Docker comparte recursos con el sistema operativo.

El contenedor se crea a partir de una imagen, un conjunto de reglas que definen su contenido. Para este proyecto, se utiliza la imagen oficial que distribuye Docker, `pytorch/pytorch:2.4.1-cuda12.4-cudnn9-runtime`, que incluye versiones compatibles de PyTorch y CUDA [26].

A continuación, se crea y establece la carpeta a la que el contenedor tendrá acceso y se instalan las dependencias necesarias. Las versiones de las librerías instaladas son: *jupyterlab 4.3*, *numpy 2.2*, *matplotlib 3.9*, *scipy 1.15*, *pandas 2.2*, *seaborn 0.13*, *nltk 3.9*, *scikit-learn 1.6*, *tokenizers 0.21* y *transformers 4.46*.

Por último, se configura el contenedor para que tenga acceso a los recursos necesarios del dispositivo. La configuración incluye la solicitud de acceso a una GPU, el acceso al *driver* de NVIDIA y el acceso al puerto 6002. Además, se define el comando que inicia automáticamente el entorno interactivo de JupyterLab al arrancar el contenedor.

Las especificaciones de los datos de entrenamiento usados y las métricas de evaluación se detallan en el apartado 7, ya que cada experimento tiene su propia configuración. Sin embargo, el *hardware* y *software* mencionados son comunes a los experimentos realizados.

7. Experimentos

En este apartado se describen los experimentos realizados del entrenamiento de los modelos y se muestran y comentan sus resultados. Previo a la configuración de cada experimento, se detallan las especificaciones idénticas para todos las pruebas.

Los datos revisados del *web scraping* contienen 4 columnas, en la que la primera indica la puntuación de la reseña, con valores entre 1 y 5. La segunda contiene el texto de la reseña extraído de Google Maps. La tercera columna indica con un 1 si la reseña tiene foto y con un 0 si no. La cuarta tiene como valor ‘Revisado’ si el comentario será utilizado para el entrenamiento y ‘Eliminado’ si será descartado.

El preprocessamiento aplicado a los datos es ajustar las puntuaciones a un rango de 0 a 4 y balancear las clases para que el modelo procese los mismos ejemplos de cada una. La eliminación de palabras vacías se aplica únicamente si el hiperparámetro asociado a este proceso lo indica.

La función de pérdida utilizada es CrossEntropyLoss [22], que convierte la salida del modelo en probabilidades mediante la aplicación de una función *Softmax*. Posteriormente, para cuantificar el error del modelo, se calcula el logaritmo de la probabilidad correspondiente a la etiqueta real y se cambia el signo del resultado. El optimizador que se utiliza es Adam, con una tasa de aprendizaje de 0.001, actualiza los gradientes del modelo en función del resultado de la función de pérdida y los gradientes actuales. El proceso que realiza el optimizador únicamente se realiza durante la evaluación del conjunto de entrenamiento.

En cada entrenamiento de un modelo se calcula el tiempo, la precisión y el error tanto en el conjunto de entrenamiento como en el de validación.

El tiempo se mide utilizando la función perf_counter de la librería time, restando el valor obtenido al final del entrenamiento con el valor del inicio.

El error se calcula sumando todos los resultados de la función de pérdida y dividiéndolos entre el cociente del número total de datos y el tamaño del lote. El porcentaje de acierto se calcula comparando los índices de los valores más altos del vector resultante de la función de pérdida con las etiquetas reales de los datos, el resultado se divide entre el número total de datos.

La condición de parada verifica en cada iteración el valor del error en el conjunto de validación. Si el error de la iteración actual no supera el mejor error registrado durante el entrenamiento y se acumulan diez iteraciones sin mejora, el proceso de entrenamiento se detiene. En el caso que no se detenga el entrenamiento por esta condición, el finalizará cuando se hayan realizado 1000 iteraciones.

7.1. Datos revisados

7.1.1. Configuración

El objetivo de este experimento es lograr que el modelo alcance un porcentaje de acierto cercano al 75 %, lo cual indicaría que es capaz de predecir resultados coherentes sobre el sentimiento expresado en las reseñas. Este experimento se realiza para comprobar si el modelo podría alcanzar un porcentaje de acierto elevado utilizando un conjunto de datos pequeño para el entrenamiento.

Los datos utilizados provienen de la herramienta de *web scraping* de Google Maps, que posterior a su obtención se le ha aplicado un proceso de revisión manual. Dado que la herramienta de *scraping* no guarda los comentarios sin texto ni comentarios que no contienen todos los elementos requeridos, no existen datos vacíos en el conjunto de datos. Además, no existen valores fuera del rango de 1 a 5, ya que Google Maps únicamente permite calificar las reseñas con uno de esos valores.

El número total de datos conseguidos es 8366, de los cuales solo 6052 contienen la palabra ‘Revisado’ en la última columna. Sin embargo, tras balancear las clases para evitar sesgos en el modelo, el número final de datos se reduce a 2220. El 70 % de 2220 se utiliza para el entrenamiento del modelo, el 15 % para su validación y el 15 % restante para su evaluación.

Este experimento se ha realizado utilizando todos los posibles modelos: BERT entrenado en inglés, BERT entrenado con diferentes lenguajes, BERT entrenado en español y myBert. Para los modelos de BERT, se definen diversas combinaciones de hiperparámetros, parámetros que se configuran antes del entrenamiento y no cambian durante el proceso. En el caso de myBert, se definen unos hiperparámetros estándar para evaluar cuál es el tamaño óptimo del vocabulario.

Los hiperparámetros de BERT son: el nombre del modelo preentrenado, la opción de eliminar o no las palabras vacías de las reseñas, la decisión de congelar o no los pesos del modelo y el tamaño de lote, que son potencias de 2, desde dos elevado a cero hasta dos elevado a la seis. Los hiperparámetros que permanecen constantes son el tamaño del vector de entrada del modelo, que es 512, y el porcentaje de neuronas que no se entranan en el entrenamiento, que es el 10 %.

El hiperparámetro que varía para las pruebas con el modelo myBert es el tamaño del vocabulario de *tokens*. Para determinar el tamaño óptimo, se define un modelo con el resto de hiperparámetros fijos. Los posibles tamaños de vocabulario que se evaluarán son: 118, 1000, 2500, 5000, 7500, 10000, 12500, 15000, 17500, 20000, 25000, 30000 y 35000. El vocabulario de *tokens* estará compuesto por palabras o composiciones de palabras presentes en los comentarios del conjunto de 6052 datos que se consideran para el entrenamiento del modelo. Los

hiperparámetros fijos son 512 para el tamaño del vector de entrada al modelo, 4 para el tamaño de lote, 10 % para el porcentaje de neuronas que no se entrenan en el entrenamiento, 512 para tamaño del vector que representa un *token*, 8 para el número de cabezas de atención y 2048 para el tamaño del vector de la capa de *feed forward*.

7.1.2. Resultados

Primero, se analizan los resultados obtenidos por los modelos de BERT al entrenarlos con un conjunto de datos reducido. Antes de verificar si algún modelo ha alcanzado un porcentaje de acierto cercano al 75 % en el conjunto de validación, se identifican cuáles son los mejores hiperparámetros para los modelos.

El primer hiperparámetro que se evalúa es la decisión de reentrenar o no las capas de BERT. La Figura 21 muestra los resultados obtenidos de todos los modelos de BERT. En el eje “x” la precisión sobre el conjunto de validación y en el eje “y” el tiempo de entrenamiento en minutos. Los puntos azules representan los resultados obtenidos al modificar los pesos del modelo y los puntos de color naranja representan los resultados obtenidos al mantener los pesos iniciales.

Al observar los resultados de la gráfica, parece evidente que modificar pesos preentrenados produce peores resultados, tanto en tiempo como en precisión. Todos los modelos entrenados con los pesos congelados han obtenido una precisión más elevada en comparación con los modelos que han reentrenado las capas del modelo. Además, ninguna combinación que ha congelado los pesos ha completado el entrenamiento en más de una hora, mientras que ocho combinaciones que han modificado los pesos de las capas de BERT han superado las 4 horas de tiempo de entrenamiento.

Estos resultados podrían explicarse por el hecho de que los modelos BERT importados ya han sido entrenados, lo que significa que los pesos de sus capas ya se ajustan a la tarea general de comprensión del texto. Ajustar estos pesos para la tarea específica como la comprensión de texto gastronómico podría obtener mejores resultados si se dispusiera de un volumen inmenso de datos.

En conclusión, debido a los resultados tan contrastados entre mantener o modificar los pesos, para la realización de los siguientes experimentos se ha decidido mantener los pesos del modelo preentrenado. Por ende, en los siguientes experimentos las capas del BERT no serán entrenables.

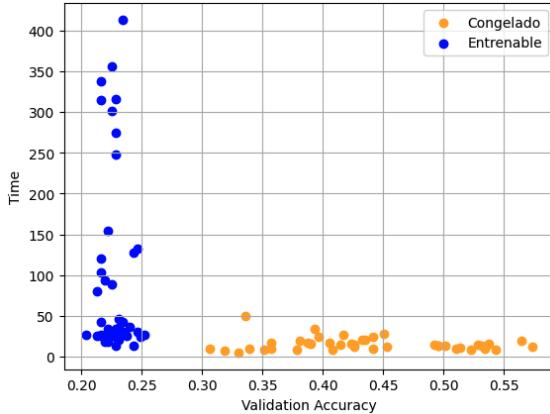


Figura 21: Gráfica de los modelos de BERT, congelando o entrenando sus capas.
[Elaboración propia]

El siguiente hiperparámetro evaluado es la decisión de eliminar o conservar las palabras vacías. La Figura 22 muestra la media de las precisiones de cada uno de los modelos de BERT. Las barras azules representan los resultados obtenidos al eliminar las palabras vacías y las barras de color naranja representan los resultados sin eliminarlas.

El primer modelo representado en el eje "x" corresponde al BERT entrenado con textos en inglés, dicho modelo obtiene los peores resultados en comparación al resto. El porcentaje de acierto promedio de este modelo no supera el 30% para los datos del conjunto de validación. Además, el modelo obtiene mejores resultados conservando todas las palabras que eliminando las palabras vacías. El segundo representado en el eje "x" corresponde al BERT entrenado con textos en español, el cual obtiene los mejores resultados en comparación a los demás. De los tres modelos, el porcentaje de acierto promedio de este modelo es el más próximo al 40 %. Al igual que el modelo anterior, el BERT entrenado con textos en español obtiene mejores resultados cuando no se eliminan las palabras vacías de los datos. El último representado en el eje "x" corresponde al BERT entrenado con múltiples lenguajes, el cual obtiene mejores resultados que el modelo en inglés y peores que el modelo en español. Al contrario que el resto, este modelo obtiene mejores resultados cuando los datos que procesa no incluyen las palabras vacías.

Para la toma de decisión de este hiperparámetro no se ha considerado el tiempo de entrenamiento, ya que previamente se ha tomado la decisión de congelar las capas de BERT. Como hemos observado en la figura anterior, todos los modelos con pesos congelados realizan el entrenamiento en menos de una hora. Por lo tanto, eliminar las palabras vacías no reduce el tiempo de entrenamiento significativamente.

En conclusión, aunque la mejora es mínima, para la realización de los siguientes experimentos se ha decidido no eliminar las palabras vacías de los datos porque esta opción proporciona mejores resultados en dos de los tres modelos evaluados.

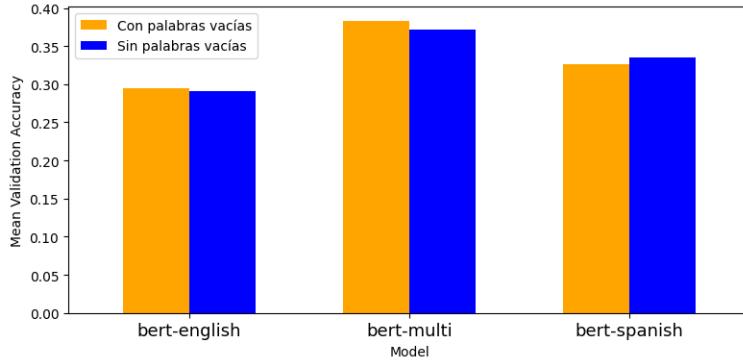


Figura 22: Gráfica de la media de acierto en el conjunto de validación de los diferentes modelos de BERT eliminando o conservando las palabras vacías. [Elaboración propia]

El último hiperparámetro evaluado relacionado con los modelos BERT es la selección del mejor modelo. La Tabla 1 muestra el promedio de las métricas para ambos conjuntos de datos. La columna Train_loss es el promedio de la función de pérdida para el conjunto de entrenamiento y la columna Train_acc es el porcentaje de acierto promedio de los datos. La columna Val_loss refleja el mismo valor que el Train_loss, pero para el conjunto de validación y el Val_acc realiza el mismo cálculo del porcentaje que el Train_acc, pero con los datos de validación.

Como se puede observar tanto en la Tabla 1 como en la Figura 22, el modelo entrenado en español obtiene mejores resultados que el resto. El BERT entrenado con textos en español, al procesar los datos del conjunto de validación, obtiene un 4% más de acierto que el modelo multilingüe y un 8% más que el modelo preentrenado con textos en inglés. Además, es el modelo con menor cantidad de error tanto en el conjunto de entrenamiento como en el conjunto de validación. Por lo tanto, si el modelo se ha entrenado con datos del mismo idioma que procesa, obtiene mejores resultados que cuando ha sido entrenado con un idioma diferente.

El vocabulario del modelo influye en el resultado ya que cuando se realiza el proceso de *tokenización*, si una palabra se encuentra en el diccionario, se le asigna una representación. Sin embargo, si la palabra no está en el diccionario, se le

asigna el *token* que indica que es una palabra desconocida. Cuando el modelo inglés procesa datos en español, no traduce el comentario a su idioma. Realizar la inferencia del sentimiento de un texto sin conocer la mayoría de las palabras conduce a peores resultados, lo que explica las diferencias en el porcentaje de acierto entre el modelo en inglés y el español.

En conclusión, para la realización de los siguientes experimentos se ha decidido realizarlos únicamente con el modelo entrenado en español, porque proporciona mejores resultados.

Name model	Train_loss	Train_acc	Val_loss	Val_acc
bert-english	1.599	0.2776	1.5481	0.2931
bert-multi	1.5578	0.3213	1.4904	0.3310
bert-spanish	1.3318	0.3996	1.13646	0.3769

Tabla 1: Media del acierto y el error de cada conjunto para cada modelo. [Elaboración propia]

A continuación, se realiza la búsqueda del tamaño de vocabulario óptimo para el modelo myBert. El procedimiento para probar un tamaño consiste en crear el vocabulario de *tokens*, *tokenizar* el texto utilizando los *tokens* generados y entrenar el modelo.

Dado que la búsqueda del tamaño del vocabulario realiza un entrenamiento de un modelo estándar de myBert, los resultados del eje *z*” de la Figura 23 no variarán significativamente con respecto al myBert configurado con los mejores hiperparámetros.

El modelo de myBert alcanza los mejores resultados con un tamaño de 12500 *tokens*, aproximadamente un tercio del tamaño máximo, que es 35000. Esto implica que la mayoría de las palabras se dividen en dos o más *tokens*, cosa que significa que el modelo tiene la capacidad de asociar y generar nuevos conceptos.

Este comportamiento podría explicarse por la capacidad de generalización del modelo. Cuando el vocabulario es un tercio del *corpus*, el modelo mejora su capacidad de generar nuevos conceptos a través de la combinación de *tokens*. Sin embargo, si cada palabra se representa como un único *token*, el modelo tendrá una peor capacidad de comprender adecuadamente una palabra nueva cuando la procese.

El porcentaje de acierto promedio se sitúa alrededor del 31 %, un 24 % menos que el mejor modelo de BERT de la Figura 21. Aunque la arquitectura de BERT incluye 11 capas más de *encoder*, incrementar las capas del modelo myBert no sería un factor determinante para lograr aumentar el porcentaje de

acierto considerablemente. Aunque el aumento de capas ayuda a captar relaciones más complejas entre palabras, se ha decidido no realizar el aumento de capas *encoder*, ya que aún así sería insuficiente para alcanzar el umbral del 75 %.

En conclusión, no podemos determinar el tamaño óptimo del vocabulario porque en los siguientes experimentos puede variar el modelo myBert o el *corpus*, por lo tanto, se debería volver a realizar la búsqueda del tamaño óptimo.

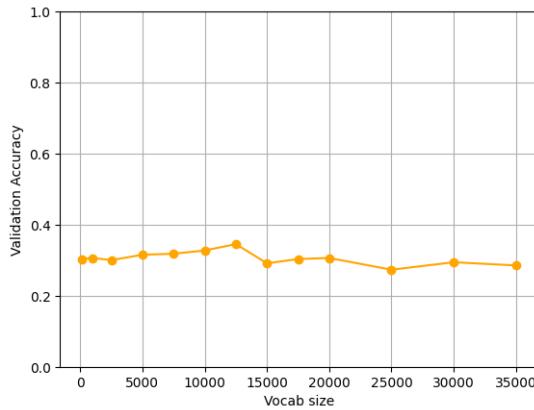


Figura 23: Gráfica de los tamaños del vocabulario y el acierto de validación del modelo myBert. [Elaboración propia]

A partir de la realización de este experimento, se pueden extraer conclusiones sobre el comportamiento de algunos modelos e intuir que el problema no parece radicar en la arquitectura de los modelos, sino en la cantidad de datos.

Finalmente, no se ha cumplido el objetivo de este experimento, debido a que el volumen de datos es insuficiente para conseguir el 75 % de acierto esperado.

7.2. Datos aumentados: revisados y sin revisar

7.2.1. Configuración

El objetivo es intentar aumentar la precisión del modelo con respecto al experimento anterior, al aumentar el volumen de datos de entrenamiento incluso si se añaden comentarios no revisados.

El primer conjunto de datos que se utilizará en este experimento será el mismo que el del apartado 7.1, a estos datos se les hará referencia como datos revisados.

El segundo conjunto de datos que se utilizará proviene de la herramienta de *web scraping*. A diferencia del primero, este conjunto no se ha revisado manualmente, por lo que no incluye la cuarta columna y todos los datos son considerados candidatos para el entrenamiento del modelo. A este conjunto se le hará referencia como datos sin revisar.

El número total de datos sin revisar son 3173, pero tras balancear las clases para evitar sesgos en el modelo, el número final de datos se reduce a 1165. Por lo tanto, el experimento se realizará con 3385 datos, la suma de los dos datos revisados y sin revisar tras el balanceo.

Se realiza una prueba que aumenta el porcentaje del conjunto de entrenamiento únicamente con los datos revisados, con el objetivo de evaluar si el aumento de datos está directamente relacionado con el aumento de la precisión. Los valores usados para ese proceso son de 70 %, 80 % y 90 %. Para el conjunto de validación, el porcentaje será de la mitad del porcentaje sobrante. Por ejemplo, si el conjunto de entrenamiento es el 80 %, el conjunto de validación será el 10 %.

El resto de las pruebas se realizan utilizando el 70 % de 3385 datos para el entrenamiento del modelo, el 15 % para su validación y 15 % restante para su evaluación. Se utiliza el mismo porcentaje que en el experimento anterior para poder verificar si con una mayor cantidad de datos se pueden obtener mejores resultados.

Este experimento se ha realizado utilizando el BERT entrenado en español y myBert. Para ambos modelos se definen unos hiperparámetros estándar, para evaluar si aumenta la precisión.

Los hiperparámetros fijos de BERT incluyen la conservación de palabras vacías, la congelación de los pesos de las capas, 4 para el tamaño de lote, 512 para el tamaño del vector de entrada al modelo y un 10 % de neuronas que no se entrenan durante el entrenamiento.

Los hiperparámetros fijos de myBert incluyen 512 para el tamaño del vector de entrada, 4 para el tamaño de lote, 10 % para el porcentaje de neuronas que no se entrenan durante el entrenamiento, 512 para el tamaño del vector que representa un *token*, 8 para el número de cabezas de atención, 2048 para el tamaño del vector de la capa de feed forward y 12500 para el tamaño de vocabulario.

7.2.2. Resultados

Antes de obtener el conjunto de datos sin revisar, era necesario verificar si el problema realmente se debía a la falta de datos. Un método rápido para comprobar si el modelo requiere más datos es aumentar el porcentaje del conjunto de entrenamiento, ya que, al procesar un mayor volumen de datos durante el entrenamiento puede mejorar su capacidad de generalización.

Dado que las pruebas del aumento del conjunto de entrenamiento se realizaron antes de obtener datos nuevos, las pruebas de la Figura 24 y la Figura 25 se han realizado únicamente con el conjunto de datos revisados.

Al analizar las Figuras 24 y 25, se confirma que los malos resultados del experimento anterior se debían a la falta de datos, ya que ambos modelos obtienen una mayor precisión a medida que aumenta la cantidad de datos de entrenamiento. Esto nos indica que si conseguimos ampliar el volumen de datos de entrada, el modelo debería mejorar su precisión. Sin embargo, el proceso de revisión de nuevos datos implica un alto coste temporal. Por esta razón, los próximos datos que se incorporarán serán datos sin revisar, obtenidos directamente de la herramienta de *web scraping*.

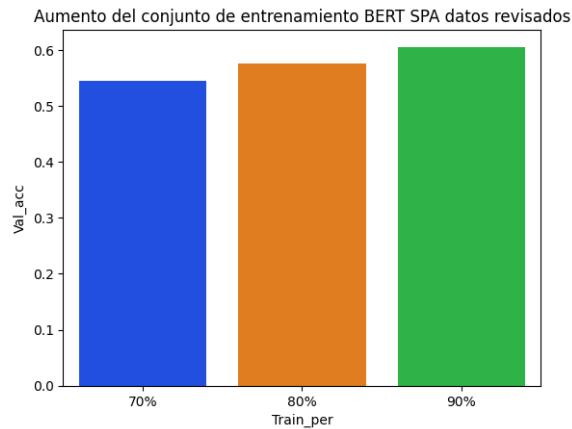


Figura 24: Gráfica aumentando el porcentaje del conjunto de datos revisados para el conjunto de entrenamiento con BERT entrenado en español. [Elaboración propia]

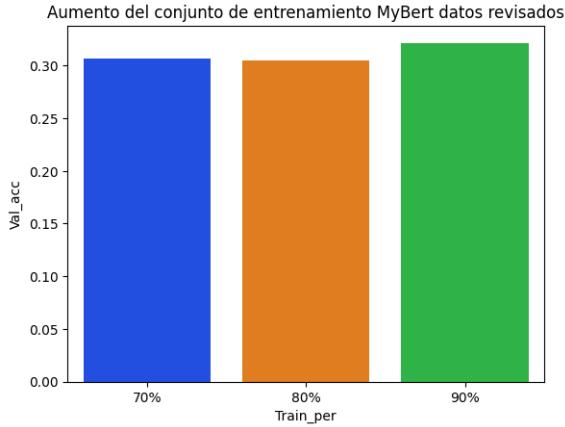


Figura 25: Gráfica aumentando el porcentaje del conjunto de datos revisados para el conjunto de entrenamiento con myBert. [Elaboración propia]

A continuación, se lleva a cabo la prueba de myBert utilizando los mismos hiperparámetros, pero con los datos revisados y sin revisar. Considerando el aspecto mencionado anteriormente, se ha decidido evaluar el comportamiento y la calidad del modelo al aumentar el conjunto de datos con datos sin revisar.

Al comparar la Figura 25 con la Figura 26 se observa que ambos obtienen resultados similares. El 70 % de los datos revisados corresponde a 1554, mientras que el mismo porcentaje de datos revisados y sin revisar es de 2222. Aunque haya una diferencia de 668 datos entre las dos pruebas, ambos modelos obtienen una precisión cercana al 30 %. En cambio, la precisión del modelo con el 90 % de los datos revisados es mejor que cuando el modelo se entrena con el 70 % de datos revisados y sin revisar. Los resultados indican que añadir mil datos sin revisar generan una mejora insuficiente para el volumen incrementado.

Al observar que el comportamiento no era el esperado, se decidió investigar en la calidad de los datos, comprobando las probabilidades de la salida del modelo y la diferencia de la etiqueta predicha con respecto a la etiqueta real.

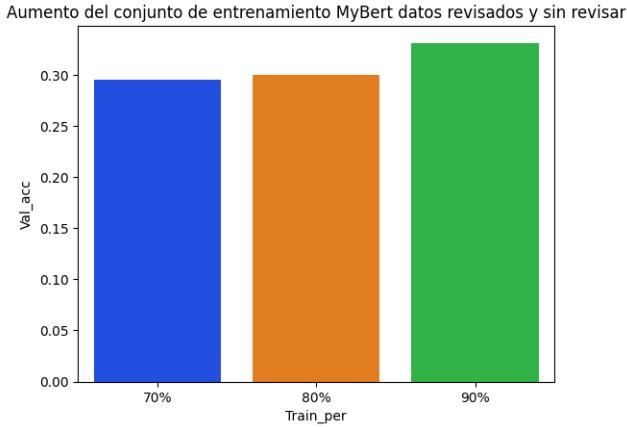


Figura 26: Gráfica variación del porcentaje del conjunto de datos revisados y sin revisar utilizando myBert. [Elaboración propia]

El primer criterio para evaluar la calidad de los datos es la diferencia en valor absoluto entre la etiqueta predicha y la etiqueta real. El segundo criterio es la diferencia en valor absoluto entre las probabilidades asociadas a ambas etiquetas, que se obtienen del resultado de aplicar una función *Softmax* a la salida del modelo. Estas evaluaciones se realizaron utilizando el 15 % del total para los datos de validación. Por lo tanto, la prueba con ambos conjuntos incluye 476 datos y la prueba con el conjunto de datos revisados incluye 333 datos.

La Figura 27 muestra las gráficas correspondientes a la diferencia de valor entre la etiqueta predicha y la real. El eje “x” es el número de ejemplos que corresponden a la diferencia de la gráfica y el eje “y” corresponde a la diferencia de probabilidades entre la etiqueta predicha y la realidad.

En la gráfica con diferencia 0 de la Figura 27, se observa que el modelo myBert ha predicho correctamente 145 ejemplos, es decir, un 30 % de los datos. El mismo porcentaje se encuentra en la gráfica con diferencia 1, donde la mayoría de casos obtiene una diferencia de probabilidad muy baja. Esto indica que el modelo presenta una indecisión a la hora de predecir, lo que sugiere una necesidad de un aumento de datos para refinar estos detalles. Por último, destaca la gráfica con una diferencia de 4, donde el modelo predice lo opuesto para el 7 % de los datos.

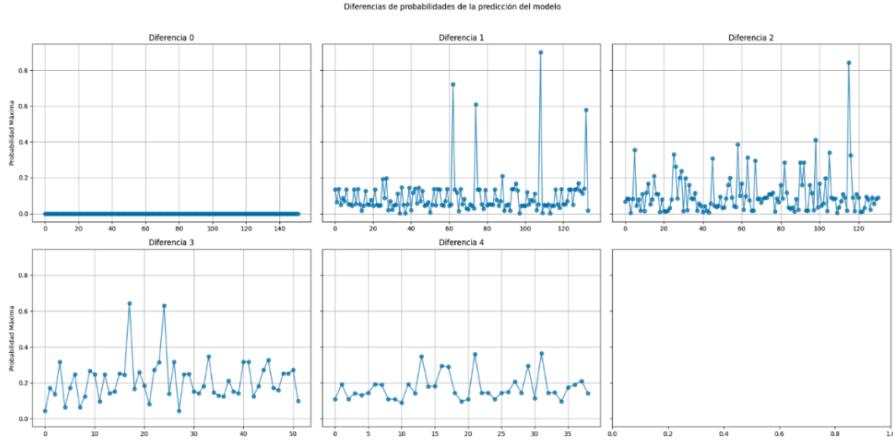


Figura 27: Gráficas con el conjunto de datos revisados y sin revisar de la diferencia de etiquetas y su probabilidad con el modelo myBert. [Elaboración propia]

En la Figura 28, en la gráfica con diferencia 0 se observan 90 datos, es decir, el modelo ha acertado el 27 % de los datos, un resultado similar al anterior. El porcentaje asciende a 36 %, cuando la diferencia entre etiquetas es 1. Sin embargo, la diferencia de probabilidades con los datos revisados es menor que la gráfica de diferencia 1 de la Figura 27. Por último, destaca el resultado de la gráfica de la diferencia 4, el cual el modelo no ha predicho lo opuesto en ningún ejemplo.

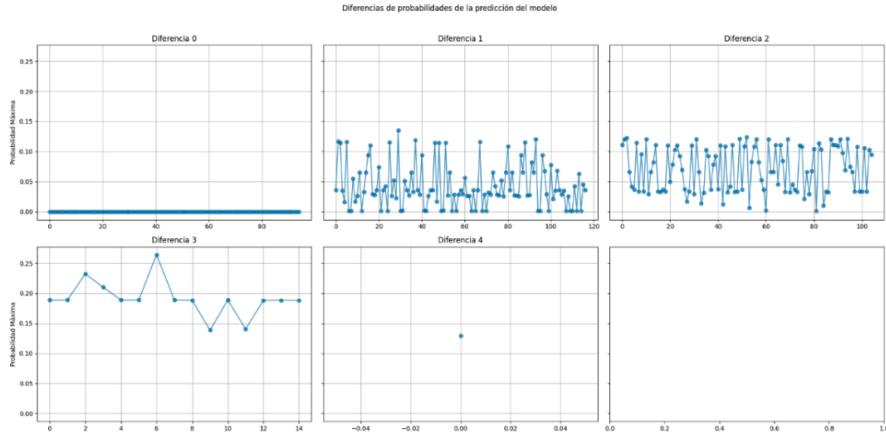


Figura 28: Gráficas con el conjunto de datos revisados de la diferencia de etiquetas y su probabilidad con el modelo myBert. [Elaboración propia]

En conclusión, los malos resultados se deben a la baja calidad de los datos. Aumentar la cantidad del conjunto con datos que no tienen una calidad mínima no produce buenos resultados. Añadir los datos sin revisar ha mejorado la precisión, pero también ha aumentado el porcentaje de que el modelo prediga resultados opuestos. Por lo tanto, se reafirma la importancia del proceso de revisión manual para garantizar que el modelo se entrena con datos de alta calidad.

Una vez finalizado este experimento, se descartó la opción de añadir una gran cantidad de datos sin revisar. En su lugar, se decidió explorar soluciones que permitieran entrenar el modelo con datos de calidad teniendo en cuenta el tiempo disponible.

7.3. Arquitectura GRU

7.3.1. Configuración

El objetivo de este experimento es superar la precisión obtenida en el experimento del apartado 7.1, utilizando los mismos datos pero con una arquitectura basada en redes neuronales recurrentes. Se plantea evaluar si aumenta el porcentaje de acierto al utilizar una arquitectura diseñada específicamente para procesar series temporales de datos.

El primer conjunto de datos que se utilizará en este experimento será el mismo que el apartado 7.1, a estos datos se les hará referencia como datos revisados. El 70 % de los 2220 datos se utiliza para el entrenamiento del modelo, el 15 % para su validación y el 15 % restante para su evaluación.

El vocabulario de *tokens* que se utilizará en este experimento será el mismo creado en el experimento del apartado 7.1.2, con un tamaño de 12500 *tokens*.

Este experimento se ha realizado con el modelo basado en la arquitectura GRU, procesando de manera bidireccional los datos. Se realiza la búsqueda del número de capas de GRU y del tamaño óptimo del vector del estado oculto, que representa la combinación de parte de la información de la secuencia y la salida de la capa anterior. Las combinaciones son los valores de las dos primeras columnas en la Tabla 2.

7.3.2. Resultados

La Tabla 2 contiene la columna Hidden_size que indica el tamaño del vector del estado oculto. La columna Num_layers es el número de capas GRU por las que se procesa el dato. La columna Train_loss es el promedio de la función de pérdida para el conjunto de entrenamiento y la columna Train_acc es el porcentaje de acierto promedio de los datos. La columna Val_loss refleja el mismo valor que el Train.loss, pero para el conjunto de validación y el Val.acc realiza el mismo cálculo del porcentaje que el Train.acc, pero con los datos de validación.

Como se observa en la Tabla 2, las dos primeras filas obtienen un porcentaje de acierto significativamente mayor en el conjunto de entrenamiento en comparación al conjunto de validación. Este suceso, es conocido como *overfitting* y ocurre cuando el modelo aprende patrones específicos de los datos de entrenamiento. Cuando predice nuevos datos intenta replicar patrones aprendidos, lo que genera malos resultados debido a la falta de capacidad para generalizar conceptos. El motivo por el cual las dos primeras filas obtienen malos resultados es por la sencillez del modelo para la tarea. El hecho de procesar los datos únicamente por dos capas, disminuye la capacidad de generalización que aumenta cuando el modelo procesa el dato por 4 capas.

Ningún modelo basado en la arquitectura de redes recurrentes logra superar el 35 % de acierto en el conjunto de validación. Aunque los modelos GRU están diseñados específicamente para tratar datos secuenciales, la simplicidad de estos modelos resulta insuficiente para capturar relaciones complejas como las gramaticales. En conclusión, esta arquitectura ha sido descartada como solución al no obtener mejores resultados con respecto al primer experimento.

Hidden_size	Num_layer	Train_loss	Train_acc	Val_loss	Val_acc
64	2	1.1199	0.5585	1.5438	0.2837
128	2	1.1568	0.5259	1.6957	0.2927
128	4	1.1454	0.3012	1.5602	0.2702
256	4	1.5661	0.2691	1.5621	0.3108

Tabla 2: Resultados del acierto y el error de cada conjunto para cada combinación. [Elaboración propia]

Finalmente, no se ha cumplido el objetivo de este experimento, ya que ningún modelo ha logrado superar los resultados obtenidos en el experimento del apartado 7.1. Estos resultados reafirman que el problema de no alcanzar el umbral del 75 % de acierto no está relacionado con la arquitectura, sino está relacionado con la cantidad insuficiente de datos disponibles.

7.4. Sinónimos

7.4.1. Configuración

El objetivo de este experimento es lograr que el modelo alcance un porcentaje de acierto cercano al 75 %, lo que indicará que es capaz de predecir puntuaciones confiables. Si el modelo supera este umbral, se evaluará la calidad de los datos para garantizar que realiza predicciones correctas y que no genere valores contrarios al sentimiento expresado en el texto.

Los malos resultados de los experimentos anteriores se debían a la escasez de datos, en concreto, a la falta de datos de calidad. Por ello, se han investigado técnicas de aumento de datos mediante transformaciones que no modifiquen el sentimiento del texto, para obtener nuevos ejemplos clasificados con la etiqueta del texto revisado.

Sin embargo, debido a la limitación del tiempo, no todas las técnicas eran válidas, ya que se necesitaba un método de automatización rápida. Por ejemplo, eliminar parte del texto requiere una revisión manual para comprobar si el sentimiento ha variado, lo cual resulta tan laborioso como revisar un comentario completo. Finalmente, se optó por la sustitución de palabras por sinónimos, ya que este procedimiento no altera el significado del texto ni su sentimiento.

La técnica de sustitución se aplica al conjunto de datos revisados utilizado en el apartado 7.1. Este método ha sido automatizado mediante programas que ejecutan diferentes procesos.

El primer proceso consiste en obtener los sinónimos de las palabras existentes en los datos. Para evitar realizar consultas innecesarias a la librería, previamente se realizan peticiones a la *API* de la RAE [27] para identificar cuales son las que forman parte del léxico y pueden tener sinónimos. Para cada palabra encontrada, se obtienen los sinónimos que proporciona el diccionario *WordNet* de la librería nltk [24]. Sin embargo, no todas las palabras se encuentran en el diccionario, por lo tanto, no se obtienen sinónimos. El número de palabras de los datos incluidas en la RAE [27] es 10418, obteniendo sinónimos de 2128.

El segundo proceso consiste en revisar sinónimos generados en el archivo Excel obtenido del proceso anterior, donde cada fila contiene la palabra original en la primera columna y sus sinónimos se encuentran en las columnas siguientes. Se ha realizado una revisión manual de los sinónimos de la mitad de las palabras, con el objetivo de garantizar una calidad mínima y asegurar que sean afines para su uso en reseñas gastronómicas. Aunque el objetivo principal de automatizar el proceso del aumento de datos no ha podido ser efectuado, revisar sinónimos de palabras individuales conlleva menos tiempo que revisar comentarios completos.

El tercer proceso consiste en reemplazar las palabras que pertenecen a los datos revisados por sus sinónimos. Para cada dato, se comprueba si contiene alguna palabra que tenga sinónimos revisados. Si es así, se sustituyen todas las apariciones de esa palabra por un sinónimo, y se le asigna la misma etiqueta que el comentario original. La sustitución se realiza para todos los sinónimos de la palabra.

El número total de datos generados mediante la técnica de aumento de datos es de 81342, pero tras balancear las clases se reduce a 33015. Para este experimento, se han utilizado tanto los datos revisados como los datos generados mediante la sustitución de sinónimos, lo que da como resultado 35235 datos, que se han distribuido entre los siguientes tres conjuntos: 70 % de estos 35235 datos se utiliza para el entrenamiento del modelo, el 15 % para su validación y el 15 % restante para su evaluación.

En este experimento, únicamente se ha realizado la búsqueda del mejor tamaño de lote para el modelo BERT, ya que la selección del resto de los hiperparámetros se ha llevado a cabo en el experimento del apartado 7.1. Por lo tanto, todas las combinaciones del tamaño de lote se han realizado utilizando el BERT entrenado en español, la conservación de palabras vacías, la congelación de los pesos de BERT, un tamaño de 512 para el vector de entrada al modelo y un 10 % de neuronas que no se entrena durante el entrenamiento.

Durante el experimento, se repite la búsqueda del tamaño óptimo del vocabulario del modelo myBert realizada en el apartado 7.1.2, ya que la incorporación de sinónimos ha modificado el *corpus* de los datos. Los posibles tamaños de vocabulario son los mismos que los utilizados en el experimento del apartado 7.1. Los hiperparámetros fijos son: 512 para el tamaño del vector de entrada al modelo, 4 para el tamaño de lote, 10 % para el porcentaje de neuronas que no se entrena durante el entrenamiento, 512 para el tamaño del vector que representa un *token*, 8 para el número de cabezas de atención y 2048 para el tamaño del vector de la capa de *feed forward*.

Una vez se ha obtenido el tamaño óptimo del vocabulario, se ha realizado la búsqueda de los mejores hiperparámetros de myBert. Se definen diversas combinaciones de hiperparámetros, que incluyen: el tamaño de lote, el tamaño del vector que representa un *token*, el número de cabezas de atención y el tamaño del vector de la capa *feed forward*. El tamaño del vector que representa a un *token* debe ser divisible por el número de cabezas de atención. Por último, con la selección de los mejores hiperparámetros, se ha realizado la búsqueda del número óptimo de capas de *encoder*.

7.4.2. Resultados

Primero, se analizan los resultados obtenidos por el modelo BERT con los diferentes tamaños de lote, teniendo en cuenta la precisión del conjunto de validación y el tiempo de entrenamiento. La Tabla 3 muestra las métricas resultantes, donde el tiempo de entrenamiento se expresa en minutos.

Al observar la Tabla 3, podemos observar una mejora significativa con respecto a los experimentos anteriores, ya que todas las filas superan el 85 % de acierto en el conjunto de validación. Además, la precisión aumenta a medida que aumenta el tamaño del lote, debido al cálculo del gradiente. El resultado de la función de pérdida es un promedio de los errores del lote, en el que un mayor número de ejemplos proporciona un resultado más estable para el cálculo de los gradientes. Esto permite un ajuste más preciso de los pesos, lo que deriva en un incremento de la precisión.

La Tabla 3 también muestra que el tiempo de entrenamiento disminuye a medida que aumenta el tamaño del lote, lo cual es lógico, ya que se procesan más datos en paralelo. Sin embargo, el entrenamiento con un tamaño de lote 8 dura más que con tamaño de lote 4. Por lo tanto, un tamaño de lote más grande no garantiza un menor tiempo de entrenamiento, debido a que los datos y los cálculos intermedios se almacenan en la memoria de la tarjeta gráfica. Si no se dispone de suficiente memoria, los datos de lote se procesan de manera secuencial, lo que ralentiza el entrenamiento.

En conclusión, el tamaño de lote óptimo es 4, ya que la diferencia de precisión es mínima, pero la diferencia del tiempo de entrenamiento es significativa. Realizar el entrenamiento en una hora y media menos es más conveniente, por si se plantea volver a entrenar el modelo en un futuro con más datos.

Batch size	Time	Train_loss	Train_acc	Val_loss	Val_acc
1	514	0.6962	0.7477	0.3888	0.8611
2	459	0.4684	0.8253	0.2589	0.9038
4	377	0.3139	0.8819	0.1580	0.9443
8	567	0.1644	0.9414	0.0830	0.9755

Tabla 3: Resultados del acierto y error para cada tamaño de lote. [Elaboración propia]

A continuación, se ha realizado la búsqueda del tamaño de vocabulario óptimo para el modelo myBert. El procedimiento para probar un tamaño consiste en crear el vocabulario de *tokens*, *tokenizar* el texto utilizando los *tokens* generados y entrenar el modelo.

El modelo alcanza los mejores resultados con un tamaño de vocabulario de 20000 *tokens*, 7500 *tokens* más que el mejor vocabulario del experimento del apartado 7.1. Teniendo en cuenta que se han añadido más palabras al vocabulario, este resultado se encuentra dentro del rango esperado. Además, el modelo sigue obteniendo mejores resultados cuando la mayoría de las palabras se dividen en dos o más *tokens*.

En la Figura 29, se observa una diferencia significativa de error entre el mejor y el peor tamaño de vocabulario, en concreto, un 27 %. Esto puede deberse a la granularidad con la que se dividen las palabras en *tokens* según el tamaño del vocabulario. Un vocabulario pequeño puede derivar en que se dividan las palabras en varios *tokens*, lo que dificulta al modelo a comprender relaciones. En cambio, con un vocabulario grande es más probable que las palabras se dividan en menos *tokens*, preservando mejor el significado.

Por último, cabe destacar que la precisión alcanzada con el mejor tamaño de vocabulario es notablemente inferior a la obtenida con el mejor modelo BERT, que puede verse incrementada durante la búsqueda de los mejores hiperparámetros.

En conclusión, para la realización de las siguientes pruebas se ha decidido utilizar el vocabulario de 20000 *tokens*, porque proporciona mejores resultados.

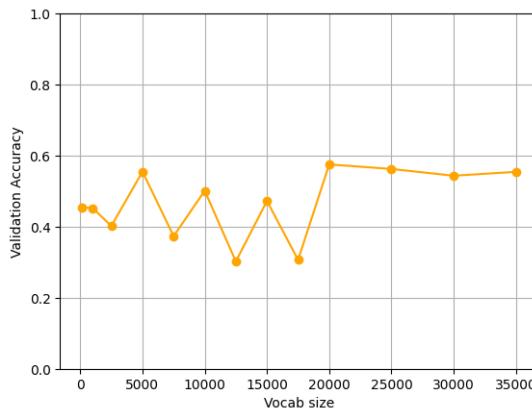


Figura 29: Gráfica de los tamaños del vocabulario y el acierto de validación del modelo myBert. [Elaboración propia]

Una vez definido el tamaño del vocabulario, se evalúan diferentes combinaciones de hiperparámetros para el modelo myBert. En la Figura 30, se muestran los resultados de las dos posibles combinaciones para los hiperparámetros. La primera combinación consiste en un tamaño de 512 para el vector que representa un *token*, 8 cabezas de atención y 2048 para el tamaño del vector de la capa *feed*

forward. La segunda combinación tiene un tamaño de 768 para el vector que representa un *token*, 12 cabezas de atención y 3072 para el tamaño del vector de la capa *feed forward*. En la Tabla 4, se muestran únicamente las pruebas del tamaño de lote para la mejor combinación anterior, donde el tiempo de entrenamiento se mide en minutos.

En primer lugar, se evalúan las dos posibilidades de los hiperparámetros: el tamaño del vector que representa un *token*, el número de cabezas de atención y el tamaño del vector de la capa de *feed forward*. Los tres valores están relacionados, ya que el tamaño del vector que representa un *token* debe ser divisible por el número de cabezas de atención, y si el tamaño del vector aumenta, el tamaño del vector de la capa *feed forward* también debe aumentar.

Al analizar la Figura 30, parece evidente que un mayor tamaño para el vector representativo de *tokens* produce mejores resultados. Ningún modelo con un tamaño 512 ha obtenido un porcentaje de acierto superior al 50 %, mientras que cuatro modelos con tamaño de 768 obtienen un acierto cercano al 70 %. Estos resultados se deben a que un mayor tamaño del vector que representa un *token* proporciona más información, lo que permite al modelo capturar una mayor cantidad de características y comprender relaciones más complejas.

En conclusión y debido a los resultados tan contrastados entre las posibles combinaciones, para la realización de las pruebas siguientes se ha decidido utilizar la segunda combinación al tener mejores resultados.

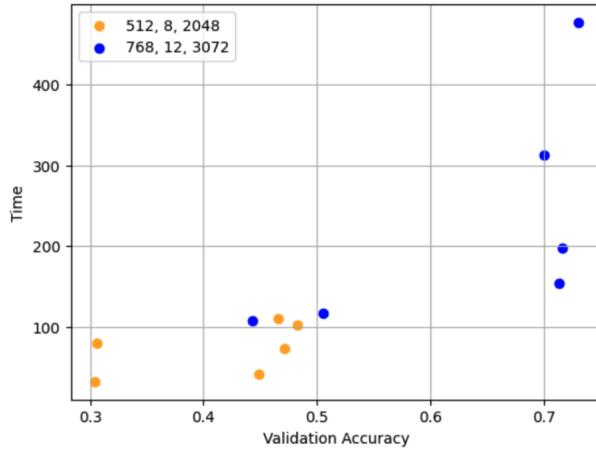


Figura 30: Gráfica de los diferentes valores para el vector de representación de un *token*, cabezas de atención y vector de la capa *feed forward* para el entrenamiento de myBert.[Elaboración propia]

Como se ha mencionado anteriormente, los resultados del porcentaje de acierto que se muestran en la Tabla 4 corresponden a los puntos azules de la figura anterior. Todavía falta seleccionar el mejor tamaño de lote, donde únicamente se muestran los resultados de la segunda combinación.

Al analizar la Tabla 4, podemos descartar los tamaños de lote de 8 y 16, ya que no superan el 60 % de acierto. Los malos resultados de 8 y 16 pueden deberse a que myBert es un modelo más pequeño y probablemente se beneficie de tamaños de lote menores, ya que las variaciones de los gradientes ayudan a salir de mínimos locales. Por otro lado, un tamaño de lote mayor genera gradientes más estables, pero un tamaño intermedio, como el de 16, introduce variaciones en los gradientes que no ayudan a salir de un mínimo local ni proporciona la robustez necesaria.

El resto de valores obtienen resultados similares, por lo que se tiene en cuenta el tiempo de entrenamiento para tomar la decisión. El tamaño de lote 1 obtiene la mejor precisión, sin embargo, su tiempo de entrenamiento es el doble que el de valores 4 o 32. Por lo tanto, al igual que en la elección del tamaño de lote para el modelo BERT, se prioriza la reducción significativa del tiempo de entrenamiento sobre un pequeño aumento de precisión.

En conclusión, para la realización de las siguientes pruebas se ha decidido utilizar el tamaño de lote de 4. Aunque este tamaño dure más que el de 32, si se realiza la prueba de aumento capas de encoder con el tamaño de lote de 32, se puede acabar llenando la memoria de la tarjeta gráfica y realizar el entrenamiento en más tiempo.

Batch size	Time	Train_loss	Train_acc	Val.loss	Val.acc
1	477	0.5503	0.7895	0.8592	0.7305
2	313	0.6126	0.7593	0.9634	0.7006
4	198	0.5782	0.7826	0.8426	0.7167
8	108	1.2204	0.4663	1.3109	0.4435
16	116	1.0702	0.5329	1.2308	0.5050
32	154	0.5387	0.7879	0.8535	0.7135

Tabla 4: Resultados del acierto y error para cada tamaño de lote. [Elaboración propia]

La última prueba del experimento consiste en determinar el número óptimo de capas de *encoder* para el modelo myBert. Al no obtener los resultados esperados al aumentar las capas de *encoder*, entrenar el modelo myBert con una capa de *encoder*.

Finalmente, se evalúa la calidad de los modelos entrenados con sus mejores hiperparámetros. Para ello, se analizan las diferencias entre las etiquetas reales y las predichas en el conjunto de validación, con un total de 5285 predicciones realizadas.

Las Figuras 31 y 32 muestran las gráficas correspondientes a la diferencia de valor entre la etiqueta predicha y la real. El eje “x” es el número de ejemplos que corresponden a la diferencia de la gráfica y el eje “y” corresponde a la diferencia de probabilidades entre la etiqueta predicha y la realidad.

Como hemos visto anteriormente, al entrenar el modelo BERT con los datos generados mediante la técnica de sustitución de sinónimos, dicho modelo alcanza un porcentaje de acierto del 94 %. Esto se refleja en la gráfica de diferencia 0, donde el eje “x” indica que el modelo ha predicho la misma etiqueta en 5000 ejemplos. Sin embargo, lo más destacable es la cantidad de predicciones con una diferencia de 2 o más, lo que significa que en 24 ejemplos el modelo predice algo significativamente diferente de lo que el texto intenta transmitir. Además, en ningún caso el modelo predice completamente lo opuesto.

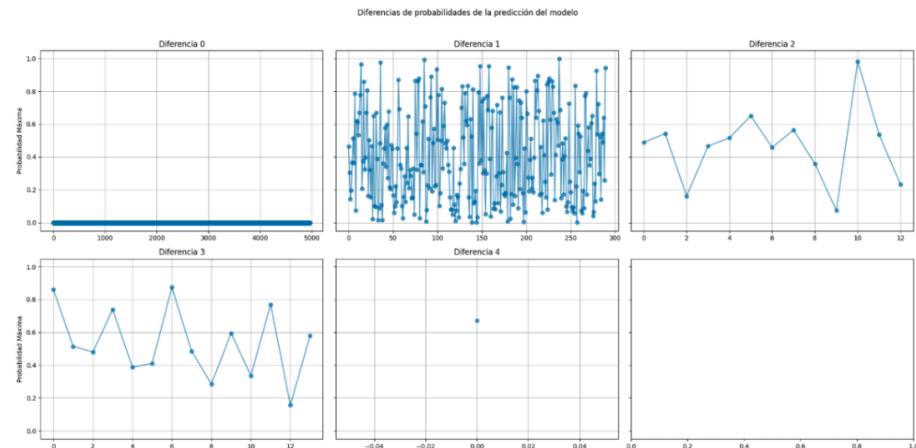


Figura 31: Gráficas con la diferencia de las etiquetas reales y sus predicciones del modelo BERT. [Elaboración propia]

El modelo myBert, configurado con los mejores hiperparámetros, obtiene un porcentaje de acierto del 66 %, acertando 1500 datos menos en comparación con el modelo BERT. Aún así, myBert muestra un progreso notable en comparación con los experimentos anteriores, aproximándose al umbral definido del 75 %. A pesar de la mejoría, las predicciones de myBert son de menor calidad en comparación con el mejor modelo BERT. Predice la etiqueta opuesta en el 2 % de los casos y una diferencia de 2 o más respecto a la etiqueta real en el 13 % de los datos.

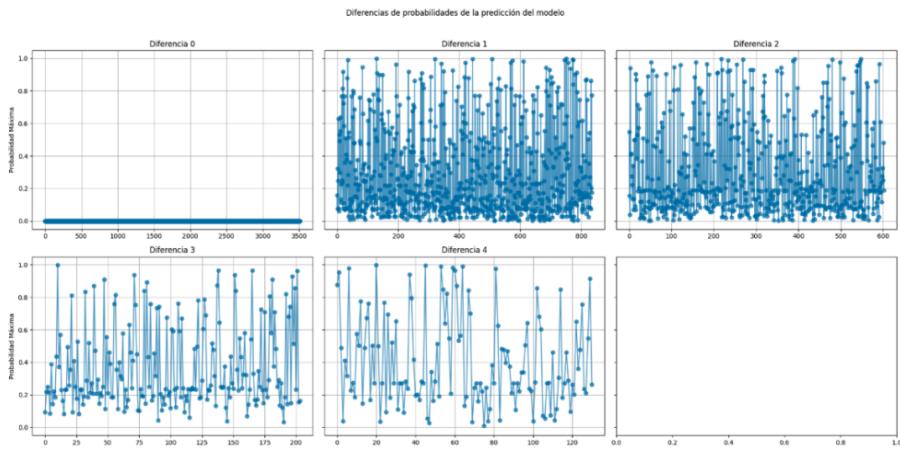


Figura 32: Gráficas con la diferencia de las etiquetas reales y sus predicciones del modelo myBert. [Elaboración propia]

La gran diferencia de precisión entre los modelos se debe al preentrenamiento de BERT, con millones de datos. El modelo de Google se entrena con tareas generales, como la predicción de palabras enmascaradas y la predicción de la siguiente frase, lo que le permite obtener pesos generalizados. En cambio, una implementación propia inicializa los pesos de manera aleatoria. Por lo tanto, BERT ya cuenta con un conocimiento de base sólido que le facilita identificar patrones en el texto y transferir este conocimiento a la tarea específica.

Otro factor determinante es el número de capas. BERT procesa los datos a través de doce capas de *encoder*, lo que le permite capturar representaciones más detalladas y complejas. En cambio, myBert únicamente puede procesar los datos a través de una capa, debido a la limitación de los datos.

En conclusión, con una implementación desde cero de la arquitectura de *transformers* es complicado lograr los mismos resultados que modelos entrenados con millones de datos.

Finalmente, se ha cumplido el objetivo de este experimento. Aunque el modelo myBert no ha alcanzado un porcentaje de acierto del 75 %, el modelo BERT ha superado con creces este umbral, demostrando su capacidad de realizar predicciones fiables.

8. Conclusiones

En este proyecto se ha evaluado el rendimiento del entrenamiento de diferentes modelos utilizando diversos conjuntos de datos. Esta sección resume los principales resultados obtenidos y las posibles tareas a realizar en un futuro.

8.1. Análisis de los resultados del proyecto

En primer lugar se evaluó el rendimiento utilizando un conjunto pequeño de datos revisados, lo que demostró que alcanzar precisiones elevadas requiere que el modelo procese una gran cantidad de datos durante el entrenamiento. Además, los experimentos realizados han mostrado la importancia del tratamiento previo de los datos; en concreto, la importancia del idioma con el que se creó el vocabulario de *tokens*.

En segundo lugar se analizó el comportamiento al entrenar un modelo únicamente con datos revisados y entrenar un modelo con la combinación de datos revisados y sin revisar. Los resultados mostraron un aumento de precisión al añadir más datos. Sin embargo, al analizar las predicciones erróneas, se demostró que el modelo entrenado solo con datos revisados predecía la valoración opuesta en un porcentaje más bajo que el modelo entrenado con datos revisados y sin revisar. Por lo tanto, este experimento afirmó la importancia de la calidad de los datos de entrenamiento.

En tercer lugar se implementó una arquitectura basada en redes neuronales recurrentes con el objetivo de evaluar si una arquitectura más simple podría lograr un porcentaje de acierto elevado con un conjunto de datos pequeño. Sin embargo, los resultados de este experimento no superaron a los obtenidos en el primer experimento. Dado que ambos experimentos utilizaron los mismos datos y las redes recurrentes obtenían un peor rendimiento, este modelo se descartó.

Finalmente, se evaluó el rendimiento de la técnica de aumentar la cantidad de datos revisados mediante la sustitución de sinónimos, con el objetivo de incrementar el conjunto de datos con datos de calidad. Los resultados obtenidos alcanzaron el umbral definido del 75 % de precisión, con el modelo BERT superando ampliamente este valor, mientras que myBert no logró llegar a dicho umbral. El objetivo principal del proyecto se ha cumplido, ya que se ha obtenido un modelo capaz de realizar predicciones fiables. Tras analizar el comportamiento de las predicciones, se puede afirmar que el modelo BERT predice resultados fiables.

8.2. Futuro trabajo

Aunque el objetivo principal del trabajo se ha cumplido, durante la elaboración del proyecto se planteó utilizar las predicciones del modelo sobre las reseñas

de un restaurante, aplicar una fórmula y mostrar la puntuación calculada de cada restaurante en una página web. El propósito era crear una plataforma similar a Google Maps, donde los usuarios pudieran obtener una valoración más precisa del restaurante, ayudándoles a que tengan una percepción más adecuada del sitio. Debido al tiempo invertido en cada uno de los experimentos y los entrenamientos, esta idea que surgió al inicio del desarrollo del proyecto se ha dejado como trabajo futuro.

La fórmula consistía en predecir el sentimiento de todas las reseñas de un restaurante, asignándoles la puntuación generada por el modelo y aplicando un factor de importancia basado en el tiempo transcurrido desde su publicación. Por ejemplo, si el modelo asigna una puntuación de 4 a una reseña publicada hace dos años y también a una reseña publicada hace un mes, ambas no tendrían el mismo peso al calcular la puntuación final del restaurante. Este enfoque considera que el restaurante puede cambiar durante el tiempo, tanto para bien como para mal.

La implementación de una página web no afecta a los entrenamientos de los modelos, por lo tanto, no sería necesario recopilar más datos ni realizar nuevos experimentos. Este trabajo adicional tiene como único objetivo proporcionar información a los usuarios.

8.3. Valoración personal

En este trabajo se ha integrado el conocimiento aprendido a lo largo del grado, especialmente en las asignaturas de Proyectos de Programación y Aprendizaje Automático. La primera asignatura ha proporcionado el conocimiento de la metodología en cascada, aplicada en el desarrollo de las tareas del proyecto. La segunda asignatura ha sido fundamental, al proporcionar las bases del funcionamiento de las redes neuronales.

La realización de este proyecto ha permitido profundizar el conocimiento relacionado con el campo de la inteligencia artificial, aprendiendo el funcionamiento de las técnicas punteras del sector: la arquitectura de *transformers*. Además, la realización del proyecto ha servido para conocer el flujo del procesamiento de datos, desde las transformaciones necesarias hasta la evaluación de los resultados.

Por lo tanto, como se menciona en la competencia CCO2.1, se ha aprendido el conocimiento de las técnicas propias de los sistemas inteligentes y se ha proporcionado un modelo capaz de representar el sentimiento de un texto, competencia CCO2.2. Durante el trabajo también se ha aplicado la competencia CCO2.4, para la obtención de los datos que posteriormente procesa el modelo.

Referencias

- [1] *Qué es y cómo funciona Google BERT*. Consultado: 23-09-24. URL: <https://bgan.es/blog-marketing-digital/que-es-y-como-funciona-google-bert/>.
- [2] *Análisis competitivo y reseñas IA: nuestras novedades para restaurantes*. Consultado: 23-09-24. URL: <https://www.theforkmanager.com/es-es/blog/herramientas-thefork/analisis-competitivo-y-resenas-ia-nuestras-novedades-para-restaurantes>.
- [3] Kai Chen Tomas Mikolov Ilya Sutskever. “Distributed Representations of Words and Phrases and their Compositionality”. En: *arxiv* (2013). URL: <https://arxiv.org/abs/1310.4546>.
- [4] Kenton Lee Jacob Devlin Ming-Wei Chang. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. En: *arxiv* (2019). URL: <https://arxiv.org/abs/1810.04805>.
- [5] Niki Parmar Ashish Vaswani Noam Shazeer. “Attention Is All You Need”. En: *arxiv* (2017). URL: <https://arxiv.org/abs/1706.03762>.
- [6] *Visualizar la atención, el corazón de un transformador*. Consultado: 23-09-24. URL: <https://www.youtube.com/watch?v=eMlx5fFNoYc>.
- [7] Consultado: 23-09-24. URL: <https://mapal-os.com/es/soluciones/reputation/funciones-de-reputation>.
- [8] *Selenium*. Consultado: 23-09-24. URL: <https://www.selenium.dev/>.
- [9] *¿Qué es PyTorch?* Consultado: 23-09-24. URL: <https://www.ibm.com/es-es/topics/pytorch>.
- [10] *Vue.js*. Consultado: 23-09-24. URL: <https://es.wikipedia.org/wiki/Vue.js>.
- [11] *Google Maps Platform*. Consultado: 2-01-25. URL: <https://mapsplatform.google.com/?hl=es-419>.
- [12] Consultado: 2-01-25. URL: <https://developers.google.com/my-business/content/review-data?hl=es#:~:text=La%20API%20de%20Google%20My,lista%20con%20todas%20las%20rese%C3%B1as>.
- [13] Consultado: 2-01-25. URL: <https://developers.google.com/maps/documentation/places/web-service/text-search?hl=es-419>.
- [14] Consultado: 2-01-25. URL: <https://mapsplatform.google.com/intl/es/pricing/>.
- [15] Sergey Pirogov. *Webdriver Manager for Python*. Repositorio en GitHub. 2024. URL: https://github.com/SergeyPirogov/webdriver_manager.
- [16] *Datos geográficos y toponomía*. Consultado: 4-10-24. URL: <https://www.ign.es/web/ane-datos-geograficos/-/datos-geograficos/datosPoblacion?tipoBusqueda=capitales>.
- [17] *Hugging Face*. Consultado: 28-12-24. URL: <https://huggingface.co/>.

- [18] *FInd all Chinese text in a string using Python and Regex.* 8-11-2024. URL: <https://stackoverflow.com/questions/2718196/find-all-chinese-text-in-a-string-using-python-and-regex>.
- [19] *Removing emojis from a string in Python.* 8-11-2024. URL: <https://stackoverflow.com/questions/33404752/removing-emojis-from-a-string-in-python>.
- [20] Consultado: 13-12-24. URL: <http://dprogrammer.org/rnn-lstm-gru>.
- [21] *GRU.* Consultado: 10-12-24. URL: <https://pytorch.org/docs/stable/generated/torch.nn.GRU.html>.
- [22] *CrossEntropyLoss.* 2-11-2024. URL: <https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html>.
- [23] *What does backward() function do?* 5-11-2024. URL: <https://discuss.pytorch.org/t/what-does-the-backward-function-do/9944/2>.
- [24] *Natural Language Toolkit.* Consultado: 23-09-24. URL: <https://www.nltk.org/>.
- [25] *Docker.* Consultado: 13-11-24. URL: <https://www.docker.com/>.
- [26] *Imagen Docker PyTorch y CUDA.* Consultado: 13-11-24. URL: <https://hub.docker.com/layers/pytorch/pytorch/2.4.1-cuda12.4-cudnn9-runtime/images/sha256-0a3b9fedfe1f61ac4d5a9de9015c0863db27-ca0fde2d4e37e6268147980b726>.
- [27] *Real Academia Española.* Consultado: 23-09-24. URL: <https://www.rae.es/>.
- [28] *Onliegantt.* Consultado: 3-10-24. URL: <https://www.onliegantt.com/#/gantt>.
- [29] *Jooble.* Consultado: 5-10-24. URL: <https://es.jooble.org/>.
- [30] *El coste de los trabajadores para tu empresa: ¿qué es y cómo calcularlo?* Consultado: 5-10-24. URL: <https://www.personio.es/glosario/coste-de-trabajador-para-empresa/#cul-es-el-coste-de-la-seguridad-social-por-cada-trabajador>.

A. Planificación

A continuación, se presenta la planificación definitiva del proyecto y se listan las tareas realizadas a lo largo del desarrollo.

GP - Gestión del proyecto

Dentro de esta área se engloban todas las subtareas que conciernen a los aspectos que no aplican directamente a las tareas de programación, pero son necesarias para realizar el desarrollo de forma eficiente. Actualmente, la gestión del proyecto no está acabada debido a que quedan partes sin documentar y falta preparar la defensa del trabajo.

1. **GP.1 - Alcance:** Realización de un estudio de las soluciones existentes para el problema y un estudio de los conceptos relacionados. Además, se definió el objetivo principal del proyecto. Duración: 35 horas.
2. **GP.2 - Planificación:** Definición de las tareas del proyecto y los recursos necesarios para su realización. Duración: 20 horas.
3. **GP.3 - Presupuesto:** Se estiman los costes del proyecto, tanto los recursos materiales como humanos. Duración: 25 horas, 5 horas más con respecto a la planificación inicial.
4. **GP.4 - Documentación:** El objetivo de esta tarea es documentar el proceso para la memoria final. La tarea todavía no ha sido completada, ya que su fecha de finalización prevista es el 9 de enero de 2025. Hasta la fecha, se han completado 20 horas de las 180 horas previstas.
5. **GP.5 - Presentación:** El objetivo de esta tarea es preparar la defensa del trabajo. La tarea aún no ha sido iniciada, ya que su fecha de inicio prevista es el 10 de enero de 2025.

OD - Obtención de datos

Dentro de esta tarea se engloban todas las subtareas relacionadas con los datos, reseñas de Google Maps, que posteriormente serán usados para el entrenamiento del modelo. El desarrollo de la tarea consiste en la implementación de un sistema automático para la obtención de los datos y su respectiva revisión. La duración de la obtención de datos ha sido de 195 horas.

1. **OD.1 - Web Scraping:** Implementación de una herramienta que realiza una petición de los restaurantes a Google Maps, obteniendo todas las reseñas y sus respectivas valoraciones. Duración: 40 horas.
2. **OD.2 Interfaz para la revisión de los datos:** Implementación de una interfaz para mejorar la revisión de los datos. Duración: 30 horas.

3. **Validación de los datos:** Revisión de los datos para comprobar que la puntuación corresponde a lo que se percibe leyendo el comentario. La primera validación de los datos se realizó en 70 horas, además se definió la rúbrica para determinar la valoración del comentario. La segunda validación de los datos se realizó en 55 horas, debido a que se revisaron menos datos porque tan solo eran necesarios los comentarios con valoración 1, 2 y 3 estrellas. Duración: 125 horas, 55 horas más con respecto a la planificación inicial.

M - Implementación del modelo

Dentro de esta tarea se engloban todas las subtareas relacionadas con la creación de un modelo *fine-tuned* de BERT, con el objetivo de obtener un modelo que realice predicciones fiables de las reseñas obtenidas. La duración de la implementación del modelo ha sido de 55 horas.

1. **M.1 - Creación del modelo:** Implementación de BERT adaptado a la tarea necesaria, con la adición de capas para obtener la probabilidad de cada una de las 5 clases. La duración incluye la preparación del entorno para la ejecución de experimentos y el aprendizaje de PyTorch. Duración: 40 horas, 10 horas menos con respecto a la planificación inicial.
2. **M.2 - Entrenamiento del modelo:** Implementación del código necesario para el entrenamiento de un modelo de PyTorch y creación de las posibles combinaciones de hiperparámetros del modelo. Un hiperparámetro es un parámetro ajustable que permite controlar el proceso de entrenamiento de un modelo, por ejemplo, la decisión de que se actualicen o no los pesos de BERT es un hiperparámetro. Duración: 10 horas, 30 horas menos con respecto a la planificación inicial.
3. **M3 - Validación del modelo:** Comprobación de los resultados obtenidos del entrenamiento, se comprueban las métricas de los conjuntos y la duración. Duración: 5 horas, 25 horas menos con respecto a la planificación inicial.

AT - Implementación de la arquitectura del transformador

Dentro de esta tarea se engloban las subtareas relacionadas con la implementación desde cero de la arquitectura de *transformers* [5]. La decisión de añadir a la planificación esta tarea se debe a la sobreestimación de la implementación de BERT, se consideró invertir las horas sobrantes para implementar la arquitectura del transformador y así enriquecer el trabajo. La duración de la implementación de la arquitectura de transformers ha sido de 61 horas.

1. **AT.1 - Creación del modelo:** Implementación de la arquitectura de *transformers* con la adición de capas para obtener la probabilidad de cada

una de las 5 clases. El objetivo de esta tarea era comprender en profundidad los conceptos de *transformers*. Duración: 50 horas, 50 horas más con respecto a la planificación inicial.

2. **AT.2 - Entrenamiento del modelo:** Generalización del código del entrenamiento comentado anteriormente. Duración: 10 horas, 10 horas más con respecto a la planificación inicial.
3. **AT.3 - Validación del modelo:** Comprobación de los resultados obtenidos y comparación con los resultados generados por BERT. Duración: 1 hora, 1 hora más con respecto a la planificación inicial.

E - Realización de los experimentos

Dentro de esta tarea se engloban las subtareas relacionadas con la ejecución de los experimentos realizados durante el transcurso del proyecto. Inicialmente no se planteó una tarea específica para los experimentos, pero los malos resultados obtenidos han derivado en una búsqueda de diferentes alternativas con el objetivo de alcanzar el 75 % de acierto en el conjunto de validación. La duración de la realización de los experimentos ha sido de 45 horas.

1. **E.1 - Experimento con datos revisados:** Realización del experimento con tan solo con datos revisados, al obtener malos resultados se revisaron más datos y se volvió a realizar el experimento. Duración: 10 horas, 10 horas más con respecto a la planificación inicial.
2. **E.2 - Experimento con datos revisados y sin revisar:** Realización del experimento con datos revisados y datos nuevos sin revisar. Duración: 10 horas, 10 horas más con respecto a la planificación inicial.
3. **E.3 - Experimento con arquitectura GRU:** Implementación de la arquitectura GRU [21], el entrenamiento se realiza con los mismos datos que los experimentos anteriores. Se implementa una nueva arquitectura para probar una alternativa a generación de los datos. Duración: 10 horas, 10 horas más con respecto a la planificación inicial.
4. **E.4 - Experimentos con sinónimos:** Implementación de un sistema para aumentar los datos. El sistema revisa las palabras del *corpus* y comprueba qué palabras pertenecen a la RAE [27]. A continuación, se obtienen sinónimos de la librería nltk [24] y los resultados son revisados manualmente. Finalmente, se sustituyen las palabras por sus respectivos sinónimos para generar más datos. Duración: 15 horas, 15 horas más con respecto a la planificación inicial.

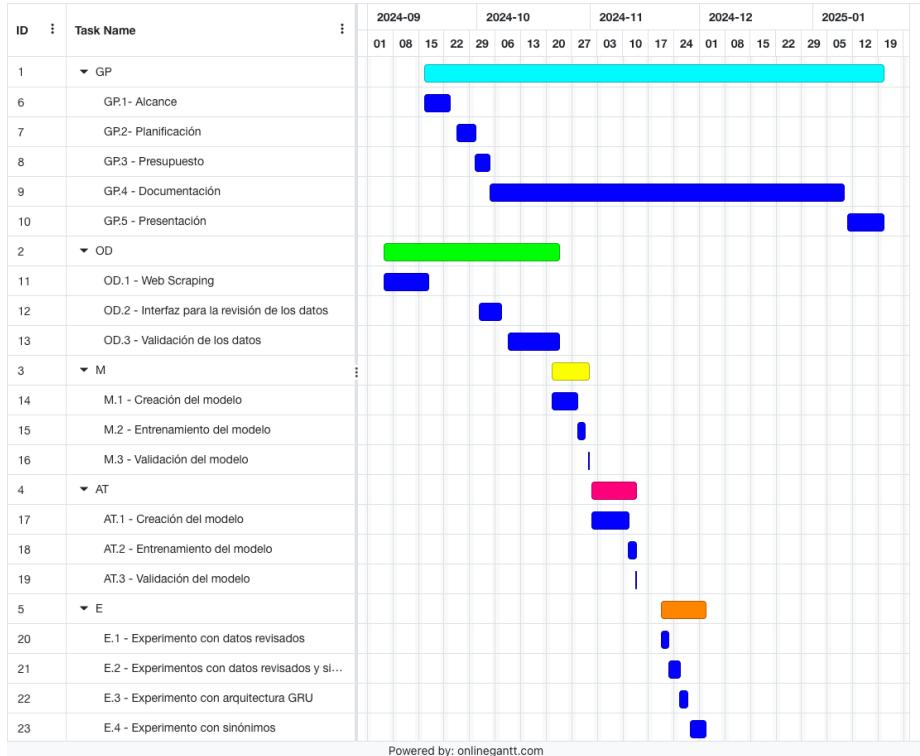


Figura 33: Diagrama de Gantt realizado con *onlinegantt* [28]. [Elaboración propia]

GP	Gestión del Proyecto	Tiempo	Dependencia	Recursos		Roles
				Portátil	Analista	
GP.1	Alcance	35h		Portátil		Jefe de Proyecto
GP.2	Planificación	20h	GP.1	Portátil		Jefe de Proyecto
GP.3	Presupuesto	25h	GP.2	Portátil		Jefe de Proyecto
GP.4	Documentación	180h	GP.3	Portátil		Jefe de Proyecto
GP.5	Presentación	25h	GP.4	Portátil		Jefe de Proyecto
OD	Obtención de datos	195h				
OD.1	Web Scraping	40h		Portátil		Programador
OD.2	Interfaz de revisión de datos	30h	OD.1	Portátil		Programador
OD.3	Validación de datos	125h	OD.2	Portátil		Programador
M	Implementación del modelo	120h				
M.1	Creación del modelo	40h	OD.3	Portátil	Analista Programador, Programador	
M.2	Entrenamiento del modelo	10h	M.1	Ordenador con GPU	Analista Programador, Programador	
M.3	Validación del modelo	50h	M.2	Ordenador con GPU		Programador
AT	Implementación de la arquitectura del transformador	61h				
AT.1	Creación del modelo	50h	OD.3	Portátil	Analista Programador, Programador	
AT.2	Entrenamiento del modelo	40h	AT.1	Ordenador con GPU	Analista Programador, Programador	
AT.3	Validación del modelo	1h	AT.1	Ordenador con GPU		Programador
E	Realización de los experimentos	45h				
E.1	Experimento con datos revisados	10h	AT.3 + M.3	Ordenador con GPU		Programador
E.2	Experimento con datos revisados y sin revisar	10h	E.1	Ordenador con GPU		Programador
E.3	Experimento con arquitectura GRU	10h	E.2	Ordenador con GPU		Programador
E.4	Experimento con sinónimos	15h	E.3	Ordenador con GPU		Programador
Total: 701h						

Figura 34: Tabla de tiempos de la planificación definitiva. [Elaboración propia]

A.1. Problemas durante el desarrollo

Durante la realización del trabajo han habido dos decisiones que han modificado la planificación inicial.

La primera decisión ha sido implementar la arquitectura de *transformers*, esta decisión se debe a que la implementación de BERT resultó ser más sencilla de lo esperado. Con el fin de hacer más robusto el trabajo se optó por realizar implementación propia de la arquitectura de *transformers*. Además, implementar dicha arquitectura permitió afianzar los conceptos de la arquitectura y obtener un modelo más flexible a la hora de añadir o quitar capas de *encoder*.

La segunda decisión surgió debido al riesgo de que el modelo no obtuviera un acierto del 75 % en el conjunto de validación. Se realizaron pruebas con todos los modelos, obteniendo resultados alejados del objetivo, siendo el mejor el BERT entrenado en español, con un 50 % de acierto. El problema parecía venir por la falta de datos, entonces se realizó una búsqueda de restaurantes mal valorados para obtener más datos de las clases con menos ejemplos y así aumentar el conjunto de datos. Sin embargo, los resultados fueron similares a los anteriores. Además, se realizó un experimento aumentando el porcentaje del conjunto de entrenamiento para confirmar que la causa era la falta de datos, y efectivamente, se obtuvieron mejores resultados con un 90 % de los datos que con un 80 % y un 70 %.

El experimento con datos revisados y sin revisar no obtuvo un acierto mayor con respecto al experimento con solo datos revisados, es decir, aumentar la cantidad disminuyendo la calidad de los datos no implicaba una mejoría en el modelo. Por lo tanto, no se consideró la alternativa de añadir una gran cantidad de datos no revisados para el entrenamiento del modelo.

Al final, se optó por una técnica de aumento de datos a partir de los datos revisados. La técnica consiste en sustituir una palabra por su sinónimo, esta sustitución genera un nuevo dato y se le asigna la misma clasificación que el comentario original. Esta técnica permite modificar el contenido sin modificar el sentimiento, generando una gran cantidad de datos con la certeza de que están correctamente clasificados.

A.2. Costes

A.2.1. Costes de personal

Los roles en este proyecto son tres: Jefe de proyecto, Programador y Analista Programador. A continuación, se muestra una aproximación de los costes de contratación de los tres roles, para la ciudad de Barcelona y teniendo en cuenta el coste de la Seguridad Social. Los salarios brutos han sido extraídos de la plataforma *Jooble* [29].

El cambio de planificación afecta directamente al coste de personal, ya que modificar las horas implica que los trabajadores trabajen más o menos. En este proyecto han habido cambios que han aumentado las horas de trabajo, por lo tanto, han aumentado los costes de personal.

Rol	Salario por hora bruto	Salario bruto del proyecto	Coste Seguridad Social	Coste total
Jefe de Proyecto	30€	21.300€	7.029€	28.329€
Analista Programador	25€	17.750€	5.858€	23.608€
Programador	15€	10.650€	3.515€	14.165€
Total:				66.101€

Figura 35: Tabla de gastos de personal. [Elaboración propia]

El coste del proyecto se ha calculado teniendo en cuenta una duración de 703 horas. A la hora de calcular el coste de la Seguridad Social se ha tomado como referencia los valores de las contingencias y prestaciones de un contrato indefinido [30], por lo tanto, se le añade un coste del 33 % del sueldo bruto.

A.2.2. Gastos de material

A continuación, se listan los recursos materiales que se van a utilizar.

1. Portátil: Se utilizará un *MacBook Air*, con un coste de 1500€.
2. Ordenador con GPU: Se usará un ordenador para el entrenamiento y la validación del modelo con una *NVIDIA GeForce RTX 3060 Ti*, un procesador *Intel i9-10900KF*, una memoria *RAM de 64GB*, etc. El coste del ordenador se estima en unos 2800€.

Para los siguientes cálculos se estima que un año tiene 52 semanas, que cada semana tiene 5 días laborables a jornada completa y que el trabajador dispone de 30 días de vacaciones al año. Cuando hemos obtenido el coste por hora del hardware se multiplica por las horas que se utilizará el dispositivo en el proyecto. Por lo tanto, la amortización del hardware se calcula con la siguiente fórmula:

$$\frac{\text{Precio} \times \text{Cantidad}}{\text{VidaÚtil} \times (52 \times 5 - 30) \times 8} \times \text{HorasProyecto}$$

Figura 36: Fórmula para calcular la amortización del hardware.[Elaboración propia]

Finalmente, durante el desarrollo del proyecto no ha sido necesario alquilar más cómputo para el entrenamiento de los modelos. Por lo tanto, el coste del alquiler de la GPU se ha eliminado.

Hardware	Precio	Unidades	Vida útil	Horas	Amortización
Portátil	1.500€	1	5 años	560h	91€
Ordenador con GPU	2.800€	1	8 años	80h	15€
				Total:	106€

Figura 37: Tabla de gastos de la amortización de los recursos materiales. [Elaboración propia]

A.2.3. Coste total

Una vez finalizado el proyecto, se suman los costes totales de cada apartado para conocer el coste total del proyecto. El coste total del proyecto ha sido de 66.207 euros, 1362 euros más con respecto al coste estimado al inicio del proyecto.

Tipo	Coste
Personal	66.101€
Hardware	106€
Total:	66.207€

Figura 38: Tabla de gastos totales del proyecto. [Elaboración propia]

B. Sostenibilidad

Cada vez es más importante tener en cuenta el impacto que tienen los proyectos que desarrollamos en el ámbito de la sostenibilidad, tanto económica, social como ambiental. Es por esta razón que a continuación se expone una autoevaluación sobre este dominio, y se analizan las tres dimensiones dentro del marco de este trabajo.

B.1. Autoevaluación

Dentro de la gestión de un proyecto, el aspecto económico siempre ha tenido una predominancia importante sobre el resto de aspectos de la sostenibilidad, mientras que el aspecto medioambiental y social han tenido un papel secundario. Esto ha cambiado durante los últimos años, debido a que acontecimientos como el cambio climático han concienciado más a las personas y eso se ve reflejado en la planificación de los proyectos.

En el marco de la Ingeniería Informática, estos aspectos pueden parecer distantes, ya que actuamos principalmente con ordenadores. Sin embargo, el consumo energético o el coste ambiental que implica fabricar un ordenador son

aspectos que afectan al medio ambiente, por lo tanto, tenemos que ser más conscientes de estos hechos a la hora de tomar decisiones.

Como reflexión personal, siempre le daba más importancia a la planificación económica porque así se define lo que se va a hacer y su coste. Consideraba que sin un buen plan económico el proyecto no podría realizarse satisfactoriamente. Pero desde que cursé el grado de Ingeniería Informática, y gracias a algunas asignaturas que la facultad ha llevado a cabo en el ámbito de la sostenibilidad, cada día me doy más cuenta de la importancia que tiene el aspecto ambiental y social dentro del mundo de la informática.

B.2. Dimensión económica

El mercado objetivo de la aplicación es un colectivo concreto, en términos económicos va dirigido a los dueños de los restaurantes, conocer la opinión pública les ayuda a tomar decisiones respaldadas. Por lo tanto, considero que el coste del desarrollo está justificado teniendo en cuenta que puede evitar inversiones innecesarias.

Como hemos comentado en el apartado de Soluciones Existentes, las principales compañías que ofrecen este servicio te obligan a contratar un plan de pago. Sin embargo, la herramienta que se ofrecerá cuando el proyecto finalice no tendrá ningún coste para cualquier usuario. Por lo tanto, si se logra poder realizar una interfaz para que los usuarios puedan interactuar con ella será un servicio que se podrá distribuir gratuitamente.

En resumen, si no disponían de este servicio ahora podrán conocer la opinión de sus clientes y considerarla a la hora de tomar decisiones. Por otro lado, quienes ya lo tenían contratado podrán cancelarlo y ahorrar ese dinero.

B.3. Dimensión medioambiental

El principal impacto medioambiental que puede tener el proyecto es el alto consumo energético que puede comportar el uso de las GPUs. Es difícil de solucionar, debido a la necesidad del alto consumo para el entrenamiento del modelo. Los recursos materiales no han sido comprados específicamente para este proyecto, de modo que se considera que se reutiliza el hardware y se aprovecha el alquiler de GPUs para utilizarlo tan solo horas necesarias y así no se compra hardware.

En el caso que finalmente se pueda distribuir la aplicación, se podrían evitar decisiones que implican un impacto en el medioambiente. Por ejemplo, si un dueño considera que debe realizar una reforma para cambiar la estética del local para atraer a más gente, pero la opinión de los clientes es buena hará reflexionar al dueño y evitará la reforma. La reforma del local podría haber supuesto un

impacto en la huella ambiental, pero al leer las opiniones ha decidido que no es necesario y se ha evitado aumentar la huella de carbono.

B.4. Dimensión social

Personalmente, el mundo de la inteligencia artificial me ha llamado la atención debido al potencial de automatizar trabajos. El hecho de poder obtener resultados similares en menos tiempo que un humano es asombroso porque permite ahorrarnos tiempo y dedicarlo a otras tareas más importantes. Por lo tanto, desarrollar este proyecto me permitirá profundizar sobre el funcionamiento de estos algoritmos, y me permitirá desarrollar conocimiento para poder continuar mi trayectoria, tanto profesional como personal, en esta área.

Actualmente, la aplicación por donde se puede conocer la opinión de un local es mediante Google Maps pero no se aplica ningún filtro a las reseñas, por lo tanto, hay reseñas que no contribuyen para tomar una decisión y afectan a la valoración del local. En el caso del ámbito social, la aplicación mejoraría la experiencia de comer fuera de casa y así sabrían las personas si realmente van a un sitio que merece la pena. La mayoría de las personas solo comprueba la valoración numérica sin mirar los comentarios y a veces el número no representa la experiencia del local.

La consideración sobre si un comentario merece una buena valoración o mala valoración estará sesgado por el entrenamiento, por ende por la opinión del autor. Sin embargo, se aplicará la imparcialidad durante la revisión de las reseñas y se evaluará el comentario sin saber el nombre del local ni de la persona. Por lo tanto, mi opinión será desde un punto de vista objetivo.