Instituto Tecnológico de Buenos Aires - ITBA
Escuela de Ingeniería y Gestión

Generación de Texto Condicionado

Autores

Dey, Patrick. 59290

Lombardi, Matías. 60527

Tutor

Pérez Sammartino, Francisco.

TRABAJO FINAL PRESENTADO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN INFORMÁTICA

Buenos Aires
14 de Diciembre de 2023

Índice

Resumen

En el ámbito de la inteligencia artificial, uno de los desafíos más significativos reside en la adquisición de datos, dado que con frecuencia se enfrenta a la problemática de contar con información incompleta, inexacta o que requiere de intervención humana (por ejemplo, para clasificarla). Los datos faltantes no son el único problema en la obtención de información, sino también el no tener un conjunto balanceado. Por ejemplo, si en un modelo que busca identificar las diferencias entre frutas se cuenta con un 90 % de manzanas y el restante en bananas, la calidad de los resultados generados por este modelo probablemente se vea afectada debido al desbalance en la información proporcionada. Este problema se agrava en trabajos donde la disponibilidad de datos confiables y completos es limitada, lo que remarca la importancia de implementar estrategias eficaces que disminuyan el efecto negativo de estas problemáticas.
Una de las fuentes de datos más extensas en la actualidad es el texto escrito: proveniente de redes sociales, evaluaciones de desempeño por parte de jefes de equipo, artículos científicos, revistas de entretenimiento, etc. En particular, para el Procesamiento del Lenguaje Natural (**NLP**, por sus siglas en inglés), la calidad de los datos es esencial para entrenar modelos capaces de comprender, interpretar y generar texto de manera coherente y contextualmente relevante. Este proyecto se enfoca en el análisis de técnicas que mitiguen los problemas mencionados, evaluando la capacidad de estos modelos para generar texto que refleje fielmente la distribución del conjunto de datos original. La evaluación de los resultados obtenidos es un aspecto clave, ya que permite medir la capacidad del modelo para capturar la complejidad semántica y sintáctica del lenguaje, así como para reproducir las estructuras y patrones presentes en el conjunto de datos de referencia.

Introducción

Una de las principales tareas en el Procesamiento del Lenguaje Natural es definir un modelo de lenguaje (*LM*, por sus siglas en inglés). Los *LM* se encargan de predecir cuál será la siguiente palabra dentro de una oración, en particular, predicen el siguiente *token* (definido como una secuencia de caracteres). Dada una serie de *tokens* x(1), x(2)..., x(n), el *LM* computa la distribución de probabilidad del siguiente *token* x(n+1).
Por ejemplo, dada la frase "Hoy me gustaría...", la palabra "jugar" es más coherente en este contexto que la palabra "banco". Un *LM* debería asignarle una probabilidad mayor a la primera palabra.
Los *LM* son útiles a la hora de aprender una representación (en particular, una distribución de probabilidades sobre el vocabulario utilizado) sobre diversos *corpus* (textos en el dominio dónde se quiere trabajar) sin necesidad de tener secuencias clasificadas, ya que la misma secuencia es la que se supervisa. Esto se denomina **aprendizaje autosupervisado** [1].

Estado del arte

Existen diversas arquitecturas que se encargan de procesar texto, sin embargo, la mayoría tienen en común distintas actividades como preprocesamiento o vectorización.

Técnicas de preprocesamiento de texto

Para el análisis de texto es fundamental que el formato que se utiliza en el *input* (entrada del modelo) sea adecuado para la tarea que se está intentando resolver. Según la aplicación que tendrán los datos, o los datos en sí, se requerirán distintos tipos de preprocesado. De igual manera, muchas técnicas se repiten sin importar el área de uso.

En los siguientes puntos se enumeran las técnicas más utilizadas seguidas de su aplicación a la siguiente oración, a modo de ejemplo: "Bailando estaba la bailarina".

1. **Segmentación** [2]: se divide un documento en partes más pequeñas, que no necesariamente son palabras. Pueden ser oraciones, tópicos o frases, también denominadas "secciones de interés".

2. **Tokenización**: esta técnica tiene como fin dividir una oración en una secuencia de *tokens* de tamaños variados, siendo este un caso particular de la segmentación. A su vez, las palabras pueden ser segmentadas en subpalabras. Por ejemplo, puede resultar útil, en el contexto del inglés, contar con la palabra "don't" dividida en "do"y "n't"(el modelo podría aprender relaciones en donde la negación del verbo "hacer" tenga cierta probabilidad dependiendo del contexto de la oración).
   Es la técnica más utilizada en NLP, ya que los modelos trabajan a nivel de *token* [3]. Siguiendo la oración de ejemplo, esta puede dividirse en *tokens* separando por los espacios en blanco. Los mismos resultan: "Bailando", "estaba", "la" y "bailarina".

3. **Uniformidad en la capitalización**: transforma todo el texto de entrada a minúscula para que luego no se identifiquen dos palabras iguales con distinta capitalización como distintas. La oración de ejemplo varía únicamente la primera letra: "bailando estaba la bailarina".
   Si el objetivo es un modelo capaz de generar texto, esta técnica no resulta de interés, ya que la forma sintácticamente correcta de escribir es con una mayúscula al principio de cada oración (además de los nombres de pila, etc.).

4. **Extracción de *Stop words***: las *stop words* son aquellas palabras que se utilizan en la construcción de las oraciones, entre las que se encuentran conectores, artículos, pronombres, etc. Estas palabras, en su mayoría, no agregan significado a las oraciones [4], ya que generalmente se repiten sin importar de lo que se está hablando. Al no contemplarlas, los *tokens* necesarios a procesar se reducen. En la oración de ejemplo se considera *stop word* a la palabra "la", por lo que la salida de aplicar esta técnica es "Bailando estaba bailarina".

Es importante remarcar que las *stop words* pueden ser definidas arbitrariamente. Nuevamente, los modelos encargados de generar texto tienen que contemplar el uso de estas palabras para una salida coherente.

5. ***Stemming***: esta técnica consiste en recortar las palabras para obtener una secuencia en común [5]. Por ejemplo, "bailar" y "baile" son palabras distintas, pero ambas refieren a la misma actividad. Al aplicar esta técnica, ambas pasarán a considerarse iguales, debido a que tienen una misma raíz: "bail". Se suele utilizar una tabla que relaciona raíz con palabra.
   Para la oración de ejemplo, el resultado de aplicar esta técnica es: "bail esta la bail".

6. ***Lemmatization***: un paso más allá de *stemming*, donde se busca el lexema de las palabras. Por ejemplo, las palabras "juegos" y "jugando" hacen referencia a la palabra "jugar", por lo que podría ser considerada su lexema y las primeras 2 palabras serían transformadas en ella.
   A diferencia de *Stemming*, el resultado de aplicar esta técnica de preprocesamiento es "Bailar estar la bailar".

Existen muchos algoritmos de *tokenización* basados en reglas que hacen uso de estas técnicas (como *Byte Pair Encoding, WordPiece*, etc.). Si bien existen *tokenizadores* ya entrenados, también es posible volver a entrenarlos ajustándolos al dominio de trabajo. Es importante conocer ventajas y desventajas de cada uno y tener en cuenta que, si se utiliza un modelo ya entrenado, es importante reutilizar el mismo *tokenizador* con el cual fue entrenado si se quiere usar dicho modelo.

Embeddings

Previo a que el modelo reciba el texto preprocesado, existen 3 problemáticas a resolver:

- ***Input***: las redes neuronales reciben números o arreglos en la capa de entrada, mientras que los *tokens* que se obtienen luego del preprocesado se encuentran en formato texto (secuencia de caracteres).
- **Dimensionalidad**: si se busca representar cada *token* como un arreglo de números, hay que definir la dimensión de este (cantidad de valores que hay en el vector). Una dimensión muy grande podría ayudar a representar más características de la palabra, pero requerirá de mayor cantidad de recursos.
- **Similitud entre palabras**: tomando como ejemplo la frase "Está muy caliente el...", es razonable que las palabras "té" y "café" tengan una probabilidad alta de ser la siguiente, por lo que hay que buscar la manera de identificar que las mismas son similares para que la red se pueda retroalimentar e identificar patrones. Un *LM* que resuelva este problema puede predecir tanto la palabra té como café con un error aceptable si se esperaba una en vez de la otra (si predice la palabra "cine", no tiene

mucho sentido, por lo que se debería penalizar más este resultado).

Una de las técnicas utilizadas para resolver dichos factores se denomina *embeddings*. Una implementación notable es *Word2Vec* [6], una de las primeras técnicas de vectorización (transformación de *token* a *embeddings*) que resuelve las 3 problemáticas mencionadas de manera eficiente.
El objetivo es representar las palabras como un vector de características numéricas (también llamadas *features*) que en principio no necesariamente son conocidas y es trabajo del algoritmo encontrarlas. Esto permite representar los *tokens* en un espacio de **n** dimensiones. Como resultado se obtendrá un vector por cada *token* de la misma dimensión, sin importar su longitud original.
Al ser arreglos de números se les puede aplicar operaciones vectoriales, por lo que una forma de determinar si dos *embeddings* son similares es utilizando la similitud coseno (Fórmula 1).

$$\text{Similitud}(A, B) = \frac{A \cdot B}{\|A\| \cdot \|B\|} \tag{1}$$

El resultado de aplicar esta ecuación se encuentra entre -1 y 1, siendo -1 un mínimo de similitud y 1 el máximo. Tomando como ejemplo las palabras, "computer", "digital" y "food", las 2 primeras refieren al campo de la informática, por lo que deberían ser similares, mientras que la última no. Considerando vectores de 2 dimensiones, una posible representación de estos se puede ver en la Figura 1.

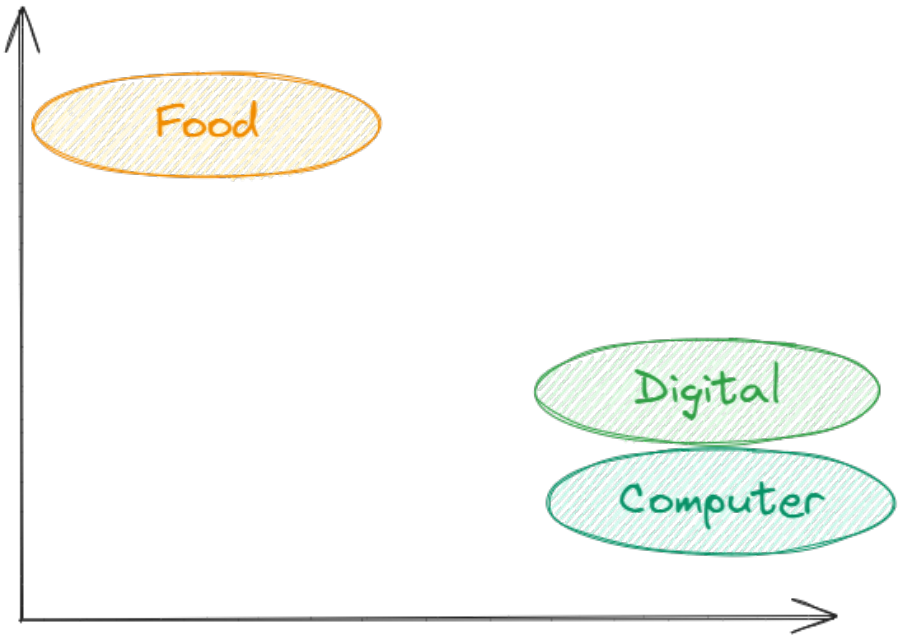

Figura 1: Representación 2-dimensional de los *tokens* "Computer", "Digital"y "Food". Como se puede apreciar, los 2 primeros refieren al campo de la informática, por lo que se encuentran a una distancia menor que a la palabra restante.

Dos *tokens* iguales serán representados por el mismo *embedding*, por lo que se les denomina estáticos y es el que recibirán los *LM* como entrada. El hecho de que se los nombre estáticos no quiere decir que no pueden modificarse, es posible entrenarlos para adaptarlos al dominio de trabajo, pero la entrada inicial siempre será la misma para cualquier par de *tokens* iguales.

Dependiendo de cómo sea entrenado un modelo, se puede obtener como salida un *embedding* por cada *token* en donde el resultado tenga en cuenta el contexto de la secuencia (puede ser a derecha de la palabra, a la izquierda o en ambos sentidos). En esos casos, el resultado del *embedding* será distinto dependiendo del contexto y se los denomina **dinámicos** o **contextualizados**.

Existe otro problema que resuelven los *embeddings*: poder identificar la posición de los *tokens* dentro de la secuencia. En caso de ser necesario, los *LM* pueden recibir un *embedding* adicional, el cual toma valores desde 1 hasta la máxima longitud posible que el modelo pueda procesar y se lo denomina *embedding posicional*.

Modelos

Es importante remarcar que hay 2 etapas en el desarrollo de un modelo. Primero está el entrenamiento, en donde a la red neuronal se le proporciona un conjunto de datos y ajusta sus pesos de acuerdo con una tarea objetivo y se busca que el modelo pueda generalizar sobre la base del conjunto de entrenamiento. En el caso de los *LM*, se aprenden patrones sobre el *corpus*, en donde el resultado será una distribución de probabilidades sobre el vocabulario.
Finalmente, está la inferencia, donde el sistema ejecuta la tarea para la cual fue entrenado (por ejemplo, clasificar un texto en sentimiento positivo, negativo o neutral). Es decir, el modelo realiza predicciones sobre nuevos datos. Los *LM* predicen cuál será el siguiente *token* dada una secuencia.

N-Gram

Para computar la probabilidad del siguiente *token*, se utiliza un algoritmo de ventana fija en donde se tienen en cuenta los n-1 *tokens* anteriores. Por ejemplo, en el caso de bigramas (en donde **n** es 2), la probabilidad dependerá únicamente del *token* inmediatamente anterior. El aprendizaje en este caso se da al obtener todas las probabilidades condicionales, ya que el problema se reduce a computar P(x(n+1) | x(n), x(n-1), ...). En este modelo únicamente se cuenta con las probabilidades correspondientes aprendidas durante el entrenamiento.
Una desventaja es que si **n** es un valor bajo, se pierde mucha información en caso de ser una secuencia muy larga.
Caso contrario, si **n** es muy grande, surge la escasez (*sparsity*), donde las oraciones con las que se entrenó son demasiado específicas y al momento de computar la probabilidad de una secuencia con la que no se entrenó dará como resultado 0. Existen varias formas de resolverlo, entre ellas: suavizado de Laplace (consiste en aumentar la probabilidad de eventos nuevos, reduciendo la de los conocidos), *backoff* (retroceder hasta el n-grama que tenga frecuencia mayor a 0) o interpolación (combinar probabilidad de todos los n-gramas).
La mayor desventaja es la dificultad de generalizar en contextos similares y largos (intrínsecamente relacionada con el problema de *sparsity*).
Por ejemplo, poder identificar la relación entre 2 palabras muy similares (como café y té). Si el *LM* da como resultado palabras similares, hay que penalizarlo menos que si habla de tópicos distintos. Al trabajar únicamente con probabilidades dadas por las frecuencias de aparición durante el entrenamiento, nunca se obtendrían palabras similares que no hayan aparecido tan seguido como el resto, aunque sean semántica y sintácticamente correctas. A esto se lo llama "problema de generalización".

Neural LM

Es un modelo que soluciona el problema de generalización haciendo uso de los *embeddings*: el *input* de la red será un vector por cada *token*. Se pueden utilizar *embeddings* ya entrenados como **Word2Vec** y dejar la capa congelada o se pueden entrenar junto con el resto de las capas.
Una primera aproximación es una red *feedforward fully connected* [7] (aquellas redes donde todas las neuronas de una capa están conectadas con las de la siguiente y las conexiones únicamente son hacia adelante).
El objetivo es similar a los n-gramas: a partir de los *embeddings*, obtener la probabilidad del siguiente *token* con una ventana fija.
La entrada del modelo será un *embedding* por cada *token*, que luego podrá tener distinta cantidad de capas ocultas, en donde la última capa tiene una neurona por cada *token* del vocabulario. Cada una de las neuronas tiene asociada un vector de pesos **W** que se actualiza con base en los *inputs* que se le presentan a la red durante el entrenamiento. Se aplica *softmax* [8] sobre la capa de salida para obtener un vector de probabilidades. Con esta función también es posible tomar muestras utilizando esas probabilidades para generar una nueva secuencia.

La desventaja de estos modelos es que, al extenderse las secuencias, para obtener una distribución correcta hace falta referenciar contextos anteriores para definir si un *token* es más o menos probable. Tomando como ejemplo la secuencia "Fui al banco, después estaba jugando al fútbol y me di cuenta de que me faltaba la plata", la palabra "plata" hace referencia al banco, sin embargo, ¿cómo hace la red para aprender este patrón? Como el contexto al que alude la palabra se encuentra al principio de la secuencia, hay que encontrar la manera de "recordarlo".

RNN

Las Redes Neuronales Recurrentes (**RNN**, por sus siglas en inglés) proponen una solución al problema presentado en las redes neuronales *feedforward*: los valores de salida de una neurona dependen exclusivamente del *output* de la capa anterior. Estas redes trabajan de forma secuencial: para poder procesar un *token* en particular, debe haber procesado todos los anteriores (ver Figura 2).
En todos los casos es posible apilar varias redes con el fin de obtener más *features* sobre la secuencia a procesar: cada capa se encargaría de aprender características diferentes sobre el texto. Igual que en los casos anteriores, se obtendrá una salida por cada *token* de entrada.

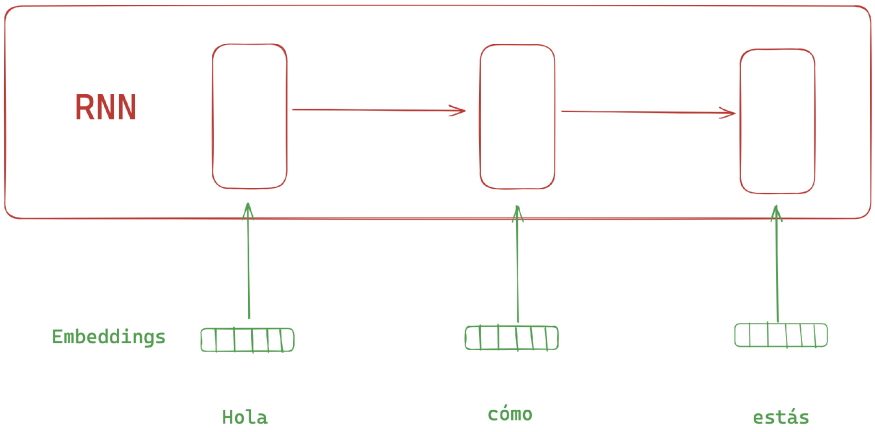

Figura 2: Arquitectura Red Neuronal Recurrente. Notar que el procesamiento del *token* "cómo" requiere el resultado de "Hola ", por lo que se trata de una red secuencial: se debe terminar de procesar un *token* y usar su resultado para procesar al siguiente.

En otras tareas como clasificación de secuencias, es común apilar redes en sentidos opuestos (una que procese de izquierda a derecha y otra en el sentido contrario, ya que el contexto a la derecha puede ser tan importante como el izquierdo). Para el resultado final, se puede realizar alguna operación, como el promedio, sobre todas las salidas intermedias.
Este modelo presenta dos grandes desventajas [9]. Primero, la imposibilidad de trabajar de forma paralela debido a que cada etapa de procesamiento depende de la anterior. Segundo, si bien presenta una mejora con respecto al aprendizaje del contexto del *token* procesado, puede existir la posibilidad que la información más relevante se encuentre al principio de la secuencia.
Como se utiliza la salida inmediatamente anterior y se realizan múltiples operaciones matemáticas con valores pequeños, al retropropagar el error para actualizar los pesos, los gradientes pueden desvanecerse. Por el contrario, también pueden volverse sumamente grandes. A estos casos se los conoce como el problema del desvanecimiento del gradiente [10] y problema del gradiente explosivo.
La ventaja es que permite procesar secuencias de largo arbitrario, aunque el procesamiento siempre debe ser secuencial.
Existe una arquitectura más compleja, llamada **LSTM** [11], en donde se incorpora el concepto de "memoria" y en cada etapa de procesamiento se computa, además de la salida, un vector que contiene el contexto.

Transformers

Introducen el concepto *self-attention* [12], que permite acceder no solamente al *token* inmediatamente anterior, sino a cualquiera dentro de la secuencia (en el caso de generación solamente se puede mirar al pasado (ver Figura 3)).

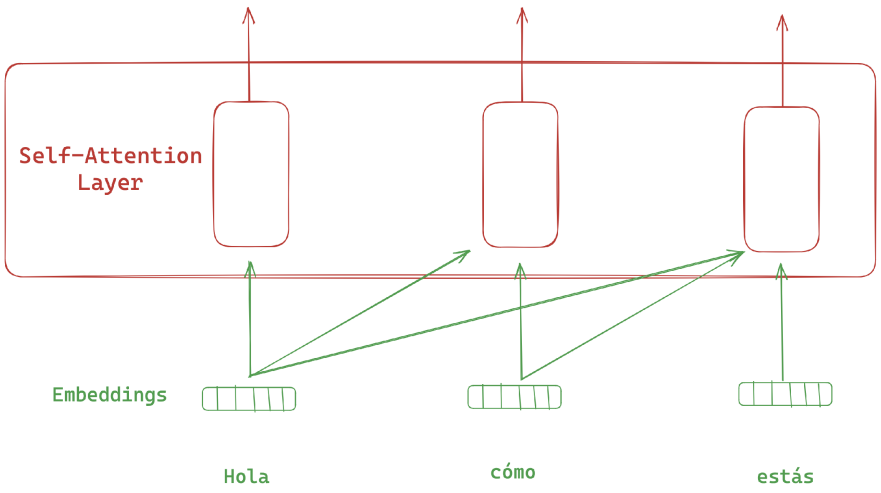

Figura 3: Arquitectura de la capa *self-attention*. Notar que se tiene acceso a todos los *tokens* de la secuencia (exceptuando los futuros, a esto se lo denomina modelo de lenguaje causal) y que el procesamiento de cada *token* es independiente del anterior (simplemente se accede al resto de la secuencia, en lugar de utilizar el resultado inmediatamente anterior).

A diferencia de las RNN, el procesamiento de cada *token* es independiente del anterior, por lo que se puede paralelizar tanto el entrenamiento como la inferencia.

Si bien en *RNN* está resuelto el problema del orden (debido a su secuencialidad), con *transformers* se pierde debido a la paralelización de las operaciones.
Una forma de solucionarlo es utilizar *embeddings posicionales*, en donde se tiene un *embedding* adicional, específico para cada posición, por lo que la entrada del modelo consistirá en un *embedding* por cada *token* en donde también se contempla la posición en la secuencia (Figura 4).

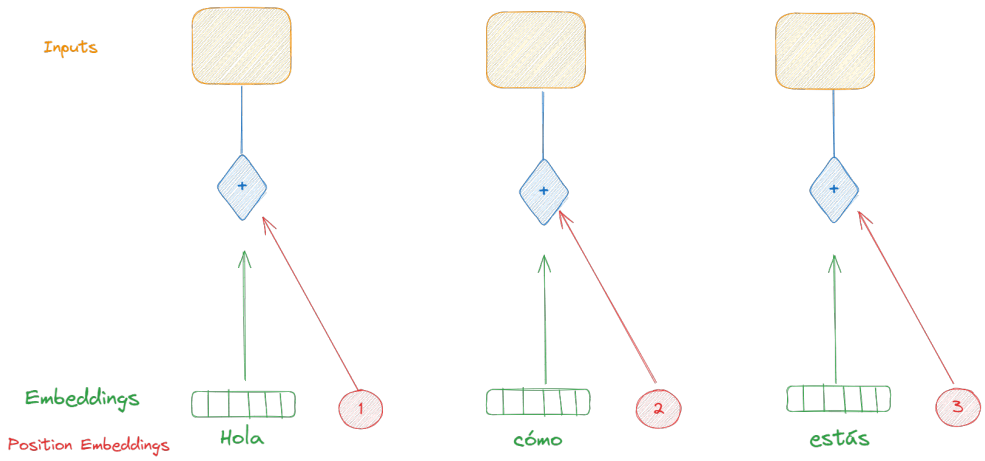

Figura 4: *Input* que reciben los *transformers*. Para considerar la posición, se realiza una operación entre el *embedding* original y la posición que ocupa el *token* que está por ser procesado. A modo de referencia, se muestra una implementación en donde se concatenan dichos *embeddings* (representado por el símbolo +).

El aporte más importante de la capa de *self-attention* es que permite "prestar más atención" a ciertos *tokens* en particular (recordar las dependencias de largo alcance). Es posible apilar varias capas de atención para que cada una identifique patrones diferentes, como se evidencia en la Figura 5.

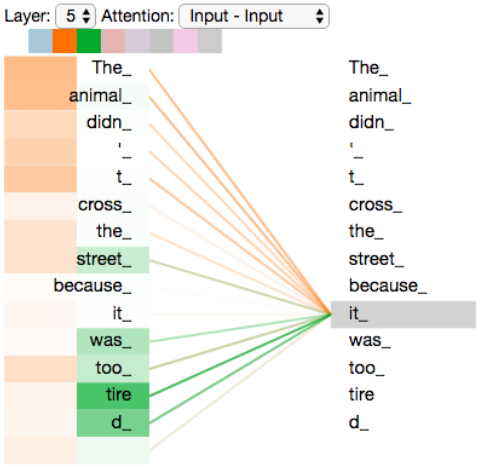

Figura 5: Ejemplo de los valores de atención para un *token* en una red con 2 capas de atención. La palabra *it*, ¿hace referencia a la calle o al animal? Notar que al procesar la misma, una de las capas se concentra más (indicado por el grosor de las líneas) en referencia al animal, mientras que la otra pone el foco en el hecho de que el mismo estaba cansado.

El funcionamiento detrás de este mecanismo está basado en los sistemas de búsqueda como el que utiliza **Youtube**. Se introduce una consulta para buscar un vídeo, esta es

comparada contra un conjunto de claves (puede ser título, descripción del vídeo, etc.) y sobre esa comparación se recuperan los valores (los vídeos que se corresponden con la búsqueda). Estos conceptos son conocidos comúnmente por su denominación en inglés: *query*, *key* y *value* respectivamente.
En el caso de los *LM*, dado un **embedding** estático, se busca obtener una representación que dependa del contexto. Los *tokens* dentro de una secuencia pueden tomar los 3 roles mencionados (*query*, *key* y *value*), por lo que se cuenta con 3 vectores de pesos. En función de esta comparación entre *query* y *key*, se busca ponderar a ciertos *tokens* sobre otros ("prestar más atención", teniendo en cuenta el contexto). En la capa de atención se obtienen los 3 vectores correspondientes a los roles aplicando las siguientes funciones:

$$q_i = x_i \cdot W^Q \tag{2}$$

$$k_i = x_i \cdot W^K \tag{3}$$

$$v_i = x_i \cdot W^V \tag{4}$$

En todos los casos, se realiza una proyección del *embedding* correspondiente al *token* **i** de la secuencia sobre cada uno de los roles (obteniendo 3 arreglos) mediante la multiplicación del mencionado *embedding* (x_i) con la matriz de pesos W correspondiente a cada rol (siendo W^K la matriz de pesos del rol *key*, W^V la matriz de *value* y W^Q la de *query*, las cuales serán entrenadas). En la Figura 6 se observan los elementos involucrados en la proyección de los *tokens*.

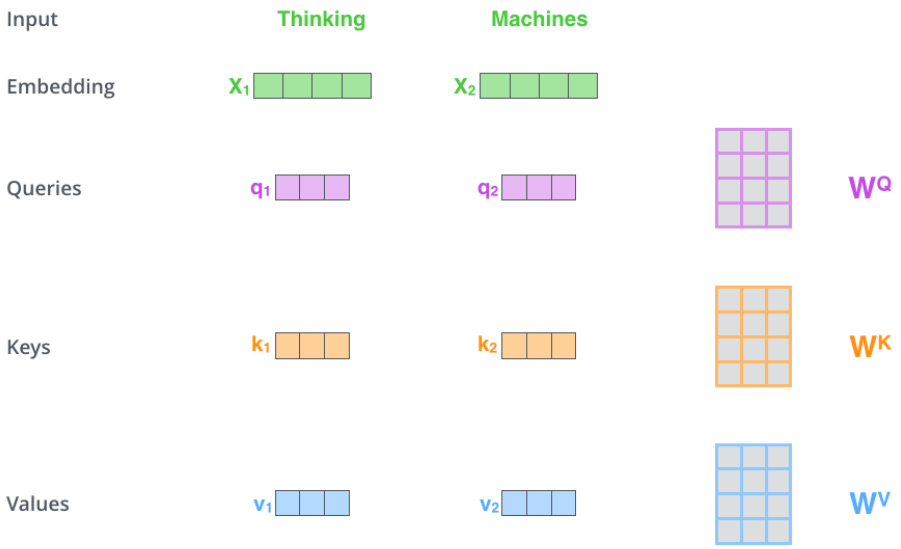

Figura 6: Multiplicando cada *token* x_i por las matrices de peso W, se obtienen sus 3 proyecciones correspondientes (por ejemplo: q_1, k_1 y v_1 para X_1).

Una vez obtenidos estos vectores, se deben computar los *attention scores*. Estos representan la importancia relativa de diferentes partes de una secuencia de entrada. Con ellos se realiza la ponderación para prestar más atención a ciertas partes de la secuencia. Este procesamiento implica computar la *query* frente a todas las *keys* y está dado por la siguiente fórmula:

$$score(x_i, x_j) = q_i \cdot k_j \tag{5}$$

Sin embargo, los productos escalares entre las matrices pueden tomar valores muy grandes, por lo que los *scores* (puntajes) se deben normalizar con la dimensión de estos.

$$score(x_i, x_j) = \frac{q_i \cdot k_j}{\sqrt{d_k}} \tag{6}$$

Esta ecuación retorna un valor que representa la importancia relativa que hay entre q_i y la clave k_j. Cuanto mayor es el resultado, mayor es la importancia que tiene la clave para dicha consulta. Para obtener puntajes positivos, que sumados den 1 y determinar cuánto de cada palabra se encuentra "expresada" en la posición analizada (es decir, cuánto influye) se le aplica la función *softmax*.

Una consideración a tener en cuenta, al encontrarse en el dominio de la generación de texto, es que no es necesario computar todas estas posibles combinaciones entre *query* y *key*, ya que solo se permite "mirar" hacia el pasado. Esto se hace mediante la anulación de las entradas posteriores, también llamado *masked attention*. Esta técnica permite al modelo poder mirar hacia atrás, adelante o en ambos sentidos dependiendo de la tarea realizada.

De esta manera, se obtiene un mecanismo mediante el cual la atención que debe prestar el modelo cuando recibe una entrada x_i se ve reflejada en la influencia que tienen los *tokens* ($\alpha_{i,j}$) sobre la posición que está siendo procesada.

$$\alpha_{i,j} = softmax(score(x_i, x_j)) \quad \forall j \leq i \tag{7}$$

Luego de computar una *query* frente a las *keys* adecuadas, se pondera a cada vector de *value* con los puntajes obtenidos en la Fórmula 7 y se los suma para obtener la salida correspondiente al *token* en la posición actual (Fórmula 8).

$$y_i = \sum_{j \leq i} \alpha_{i,j} \cdot v_j \tag{8}$$

Finalmente, se obtiene la salida de la capa *self-attention* para el *token* que está siendo procesado. Esta ponderación es la que se corresponde con la Figura 5 (el grosor de las líneas dependerá de los valores $\alpha_{i,j}$ asignados a cada *token*).

Es importante remarcar que los vectores de *key* y *query* deben tener la misma dimensión, pero el de *value* puede ser distinto.

Además, como son operaciones muy complejas, es muy costoso procesar secuencias muy largas, por lo que en general se utiliza un largo máximo.

Como el cómputo es paralelizable, las operaciones se realizan con matrices, en donde se apilan en filas todos los *embeddings* en una matriz $\mathbf{X}$, por lo que el resultado de la capa de *self-attention* se puede obtener de la siguiente forma:

$$Q = X \cdot W^Q \tag{9}$$

$$K = X \cdot W^K \tag{10}$$

$$V = X \cdot W^V \tag{11}$$

$$SelfAttention(Q, K, V) = softmax(\frac{Q \cdot K^T}{\sqrt{d_k}}) \cdot V \tag{12}$$

En referencia a *masked attention*, para generación de texto se debería anular el triángulo superior, ya que no se puede mirar al futuro para predecirlo, como se observa en la Figura 7.

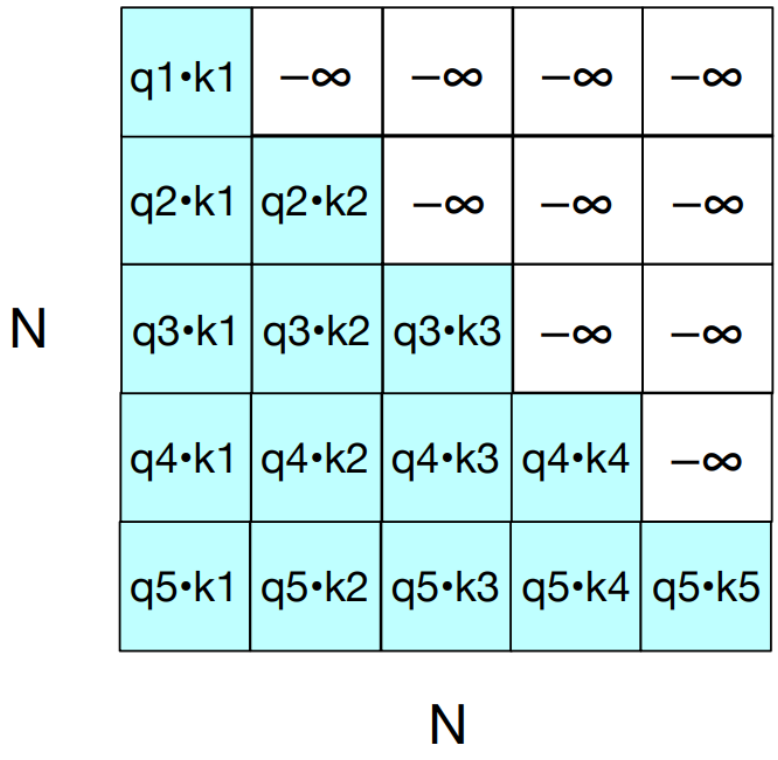

Figura 7: Matriz QK^T con el triángulo superior anulado, en donde la función *softmax* convertirá a 0 dichos valores.

Es posible considerar varias capas de *self-attention* consecutivas, lo que se denomina *Multihead-Attention Layer*. Cada una se encarga de aprender distintos aspectos (*features*)

en las relaciones que hay entre los *tokens*. Los *outputs* de cada una de estas capas se combinan, se llevan a la dimensión original y se entregan a la siguiente capa del modelo, como se muestra en la Figura 8.

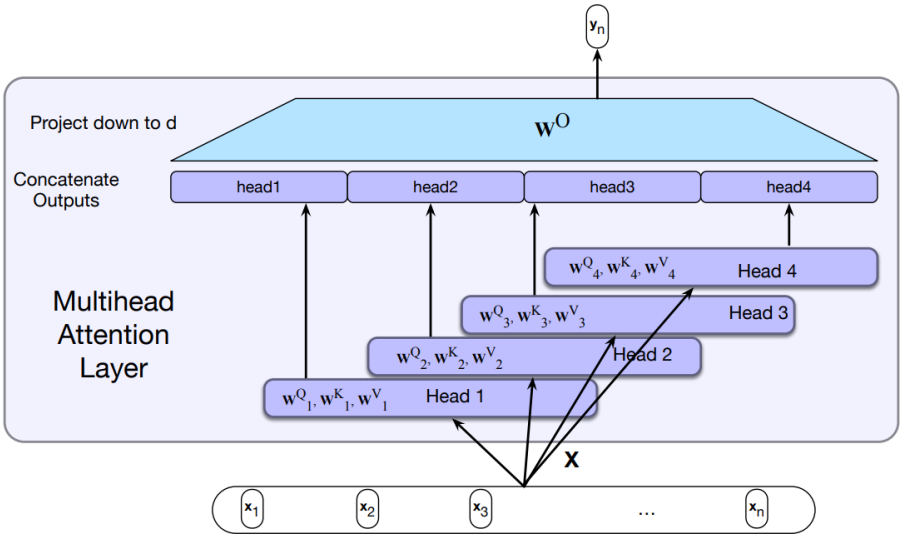

Figura 8: *Multihead self-attention layer*. Cada una de las capas tiene su propio conjunto de matrices de *key*, *query* y *value*. Las salidas de cada capa son concatenadas y luego proyectadas a la dimensión original.

La capa de atención es solamente uno de los componentes dentro de un bloque del *transformer* (ver Figura 9, en particular es la primera capa).

El flujo del procesamiento de cada *token* es el siguiente:

1. A la salida de la capa de atención se suman los *outputs* con los *inputs* originales (es decir, los *embeddings* sin tener en cuenta el contexto de la secuencia). Esto se denomina conexión residual, sirve para mejorar el acceso a la información de capas anteriores (principalmente en redes neuronales muy profundas). Se normaliza este resultado.

2. Luego el vector es procesado por una capa *feedforward* para introducir una no linealidad. Esto es muy importante, ya que permite representar un conjunto de funciones más amplio (hasta ahora las únicas operaciones aplicadas fueron productos, sumas y divisiones).

3. Finalmente, se utiliza otra conexión residual con las entradas anteriores a la capa *feedforward* y se normaliza la salida.

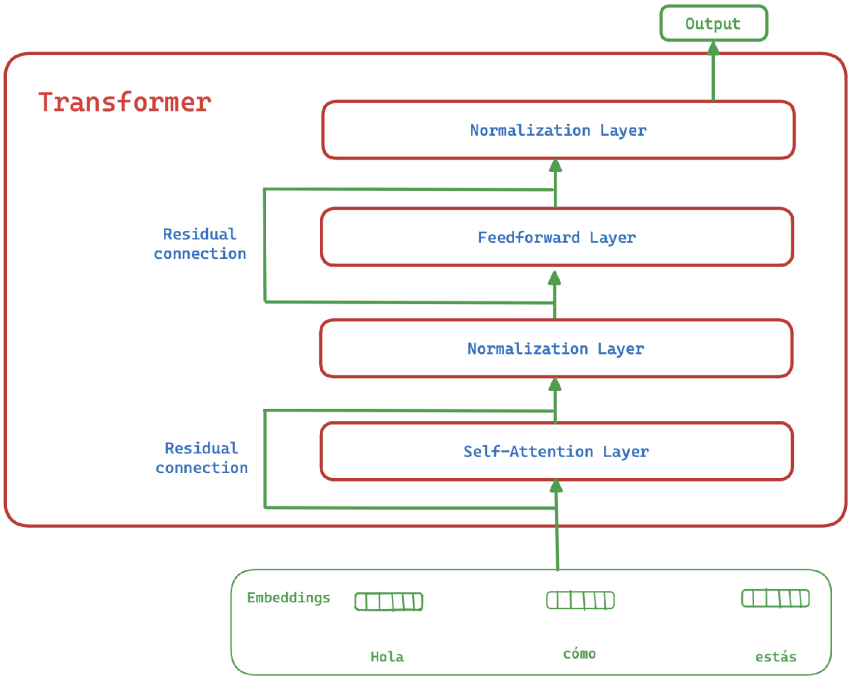

Figura 9: Bloque del *transformer*. Se parte de un *embedding* por *token* que contempla también la posición hasta obtener una salida por cada uno.

Todas estas capas constituyen un único bloque, que a su vez pueden ser apilados. La salida de la capa final será una representación contextual (otro *embedding*), en alguna dirección (teniendo en cuenta el contexto a derecha, a izquierda o ambos), del *token* procesado en la posición actual. Es decir, para cada uno de los *tokens* de entrada, al terminar de ser procesados por los bloques del *transformer*, se obtiene un *embedding* que representa características del *token* teniendo en cuenta el resto de la secuencia.

Estos resultados iniciales (el *embedding* final asociado a cada token) constituyen una "codificación" de la secuencia. Es importante entender este concepto, ya que hay arquitecturas conocidas como *encoder-decoder* que utilizan otro *transformer* (u otra red, como *RNN*) cuyo *input* es el *output* del primero.

Si bien es posible trabajar únicamente con *transformers* unidireccionales (de izquierda a derecha en el caso de generación de texto, también llamado causal), también es posible que sean bidireccionales. Existen modelos que, al procesar un *token*, consideran pasado y futuro, por lo que el *embedding* resultante pondera todo el contexto [13].
Por ejemplo, la palabra "llave" puede ser utilizada en 2 secuencias: "Me olvidé la llave de casa" y "Quedaron emparejados en la misma llave del torneo". Los *embeddings* estáticos (de entrada) serán los mismos para el *token* "llave", sin embargo, los contextos, y en consecuencia sus significados, son diferentes.
Para poder interpretar las palabras de forma correcta, es necesario considerar ambas

direcciones (no solamente el pasado). Un modelo como **BERT** [14] es capaz de devolver *embeddings* contextualizados de forma bidireccional, como se evidencia en la Figura 10. Por ejemplo, en tareas de clasificación, si se mira únicamente al "pasado" en la capa de atención, hay información a la derecha que es potencialmente útil y no es tenida en cuenta.

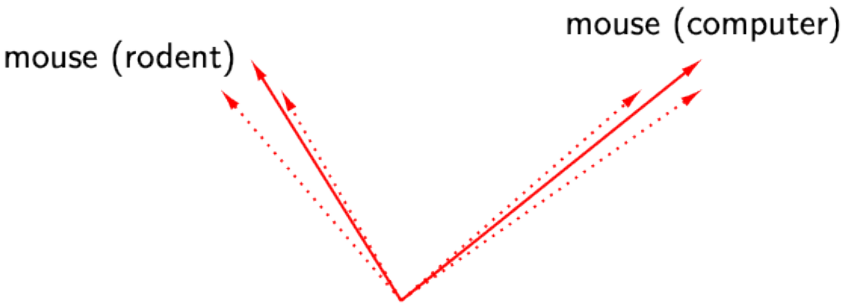

Figura 10: Posible representación de la salida de un *embedding* contextualizado en toda la secuencia. En inglés, el término *mouse* puede referir tanto al animal como al periférico utilizado en las computadoras.

En general, el *encoder* es bidireccional, ya que su tarea consistiría en codificar la secuencia original (ponderando todo el contexto), mientras que el *decoder* la decodificaría, por lo que debería ser causal (de izquierda a derecha). Un ejemplo de dicha arquitectura es la traducción automática [15] (*Machine Translation*, traducción entre idiomas).
También pueden aparecer por sí solas. Arquitecturas de solo *encoder* son útiles para tareas de clasificación de secuencias (ya que tienen en cuenta ambos sentidos), mientras que tareas de generación de texto son más apropiadas para los *decoders*.

La salida final del modelo se adecuará a la tarea objetivo. Por ejemplo, si la finalidad es generar texto, la capa final se encargará de proyectar la dimensión del *embedding* a la del vocabulario original y luego aplicar *softmax* para obtener el siguiente *token* más probable (como se explicó en la sección de *Neural LM*).

Es importante remarcar que el sentido de los *transformers* (izquierda a derecha, bidireccional) implica un cambio en la forma de entrenarlos (si es bidireccional no tiene sentido predecir la próxima palabra).

Aplicación en otros dominios

Una de las tareas que más se realizan en el ámbito del procesamiento del lenguaje natural es el de obtener un modelo entrenado en un dominio para que luego sea aplicado en otro o especializarse en algún conjunto de datos. Es decir, usar un modelo entrenado con gran cantidad de parámetros y datos para luego adaptarlo al trabajo de interés. Este proceso

implica primero contar con un *LM* preentrenado y luego realizar *fine-tuning* (ajuste de parámetros) para resolver una nueva tarea (también denominada *downstream task*, es decir, la tarea que realmente se quiere resolver). En la Figura 11 se encuentra un detalle de las partes que se modifican de la arquitectura.

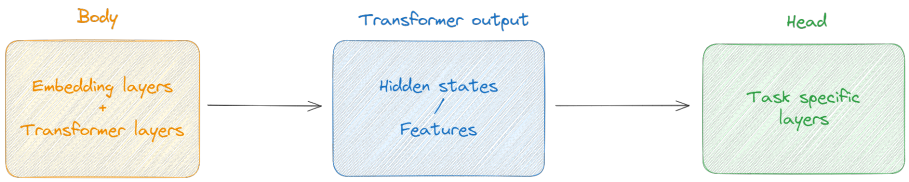

Figura 11: Partes de una arquitectura de *transformers*. El *body* consiste en las capas explicadas en la sección de *transformers* en **El estado del arte**. La salida de este son los *embeddings* contextualizados (ya sea en toda la secuencia o solamente considerando el pasado). El *head* es la capa específica a la tarea objetivo.

A esto se lo denomina *transfer learning* y no hay una única manera de realizarlo. Entre ellas existen:

1. LLM (*Large Language Model*, por sus siglas en inglés) como extractor de características: implica utilizar las salidas de las últimas capas como entradas de un nuevo modelo (que serán las características que el modelo anterior pudo captar). Puede ser únicamente la última (*hidden states*) o una combinación de varias, cuyo resultado podrá ser utilizado, por ejemplo, para entrenar diversos algoritmos de clasificación como *Random Forest* [16].

2. Reemplazar el *head* y seguir entrenando el modelo completo. Por ejemplo, en lugar de utilizar una última capa que clasifique en 5 clases, utilizar una nueva que únicamente considere 2.

3. Reemplazar el *head* y aplicar un concepto denominado congelamiento. Esto implica entrenar únicamente ciertas capas del modelo. Es decir, al actualizar los pesos de una capa, las que se encuentren congeladas no serán afectadas. En particular, es útil entrenar la capa agregada y las últimas del modelo, dejando las primeras congeladas (ya que seguramente guardan información sobre semántica del lenguaje, en lugar de la tarea en particular).

4. Ajustar el modelo a un nuevo dominio o a uno más específico, utilizando un nuevo *corpus* (por ejemplo, pasar de artículos de **Wikipedia** a reseñas de películas).

Definición de objetivo de proyecto

El proyecto consta de dos partes. En el contexto de aprendizaje automático, uno de los problemas más comunes es la ausencia de datos o un desbalance en ellos (por ejemplo, en una red clasificadora de animales, contar con 10 gatos y 100 perros), lo que podría dificultar la tarea de generalización de los modelos. Al realizar un estudio sobre clases dentro de un conjunto de datos, resulta de gran utilidad contar con cantidades similares de entradas dentro de cada categoría. Existen diversas técnicas para solucionar el desbalance de un conjunto, entre las que se encuentra la generación de nuevos datos que conserven la distribución de los originales. En búsqueda de este objetivo, la primera parte radica en poder generar datos con estas características, que puedan ser utilizados para el entrenamiento de un nuevo modelo. En particular, se adopta el marco de las reseñas de películas con el propósito de producir datos acordes a un puntaje deseado.

La segunda parte consiste en el desarrollo de una métrica que verifique la fidelidad del texto generado. La evaluación óptima sería la ejecutada por un ser humano, sin embargo, no es factible que el mismo pueda analizar todas las secuencias generadas por el modelo, por lo que se tiene como objetivo la automatización de las evaluaciones.
Una posible implementación es comparar los *embeddings* (por ejemplo, con una **similitud coseno**) de las reseñas originales y las generadas. Sin embargo, al estar en el contexto del procesamiento del lenguaje natural, resulta conveniente aprovechar el poder de expresión y comprensión que los modelos de *transformers* preentrenados obtienen sobre un gran conjunto de datos, ya que los mismos constituyen el estado del arte en lo que refiere al análisis de texto. Como resultado, se propone la implementación de una red clasificadora que se encargue de obtener una valoración para los resultados de la red generadora. Luego de obtener los modelos, se busca encontrar los parámetros que alcancen los mejores resultados, además de ser eficientes en tiempo y uso de memoria.

Las técnicas utilizadas en la primera parte pueden ser útiles en cualquier ámbito que amerite algún tipo de análisis sobre texto, es decir, si bien se utilizará el contexto de reseñas de películas para poder estudiar la capacidad generadora de estos modelos, no debería ser un limitante para poder aplicarla en otras situaciones.

Descripción de los métodos utilizados

Transformers

Los **Transformers** constituyen el estado del arte en lo que respecta al análisis del lenguaje natural. El uso de un mecanismo de atención resulta innovador y permite acceder a mejores rendimientos en un tiempo reducido con respecto a arquitecturas anteriores como **LSTM**, principalmente por la posibilidad de paralelizar las operaciones. Por lo tanto, se decide utilizar esta arquitectura.

Elección de los modelos

Red Generadora

A la hora de seleccionar el modelo que mejor se ajusta a la generación de texto, surgen varias opciones:

1. Modelo encoder-decoder cómo T5 [17] con pares: cantidad de estrellas y reseña. Está pensado para tareas como traducción o resumen.
2. Un modelo que genere reseñas basándose en un prefijo dado, siendo este un *LM* causal.
3. Un modelo diferente por cada número de estrellas, es decir, contar con un *transformer* por cada una de las valoraciones que se quiere generar (también siendo estos *LM* causales).
4. Modificar la capa de *embeddings* para que cada *token* además de contar con su posición tenga la cantidad de estrellas de la reseña. Esto implicaría hacer una modificación en la arquitectura original.

La finalidad del proyecto es poder unificar la generación en un único modelo, por lo que se utilizará la segunda opción: un *LM* causal (un decodificador) que sea capaz de generar reseñas a partir de un prefijo. El objetivo es adaptarlo al dominio de reseñas de películas, es decir, utilizar las técnicas mencionadas en la sección de **Aplicación en otros dominios**. En particular, para la resolución de este trabajo se emplea GPT-2 [18] en su versión *distil*: una versión más liviana y rápida que la completa. En la Tabla 1 se puede apreciar una comparación entre ambos modelos.

Característica	**distilgpt2**	**GPT-2/*small***
Parámetros	81.9 M	124 M
Velocidad	2X	X
Perplexity	21.1	16.3
Capas	6	12
Dimensión de capas ocultas	768	768
Attention Heads	12	12

Tabla 1: Comparación entre distilgpt2 y GPT-2/*small* (la versión más pequeña de GPT-2). El valor de *perplexity* (una métrica en donde un valor menor indica un mejor rendimiento) refiere al valor obtenido sobre el *benchmark* (prueba de referencia) **WikiText-103** [19]

Para que el *LM* sea capaz de producir reseñas de forma condicionada, se agrega una

"etiqueta" al principio de cada reseña.
La etiqueta tendrá el formato "Reseña de x estrellas", siendo x la cantidad de estrellas. Al momento de generar, también se agrega dicha etiqueta como prefijo.

Red Clasificadora

Para la red clasificadora de reseñas se utilizará el modelo **BERT**, ya que se necesita una arquitectura bidireccional (un codificador) capaz de operar a partir de representaciones de un *token* contextualizado en toda la secuencia, en particular su versión *distil* igual que en la red generadora. Cabe destacar que las versiones *distil* fueron propuestas con este modelo.
Distilbert [20] presenta 40 % menos parámetros, es 60 % más rápido y preserva el 97 % del desempeño obtenido por **BERT** en el *benchmark* **GLUE** [21]. También reduce la cantidad de capas de 12 a 6 (ver Figura 12).

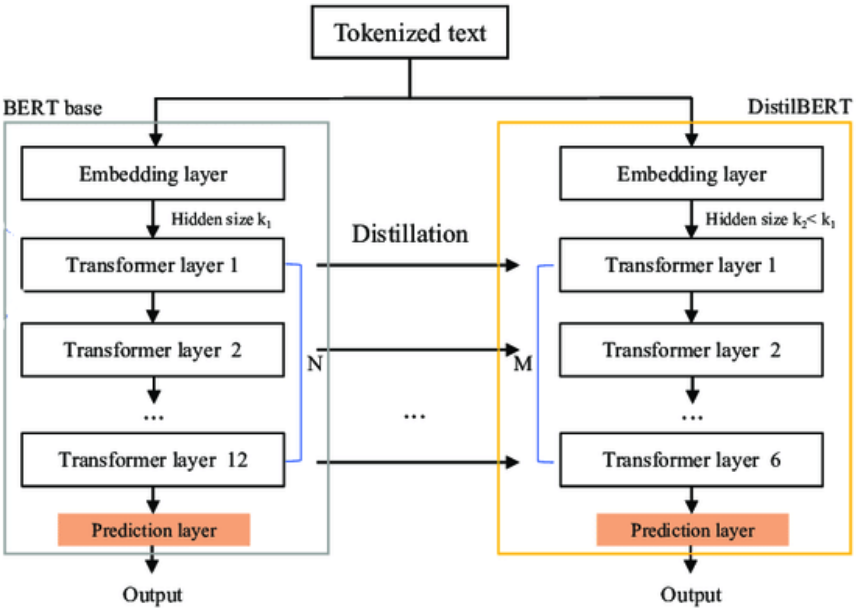

Figura 12: Comparación de las arquitecturas de BERT y DistilBERT, haciendo énfasis en la cantidad de capas.

Se utilizará la misma técnica mencionada en **Aplicación en otros dominios**: agregar un nuevo *head* al modelo con la tarea objetivo de clasificar considerando 5 clases (una por cada puntaje) y se realizará *fine-tuning*.
De esta manera, la capa final de la red se encargará de clasificar en dichas clases los textos generados, cuyos resultados permitirán extraer métricas como *accuracy*, *precision*, *f1-score*, etc.

La arquitectura final del proyecto se encuentra en la Figura 13.

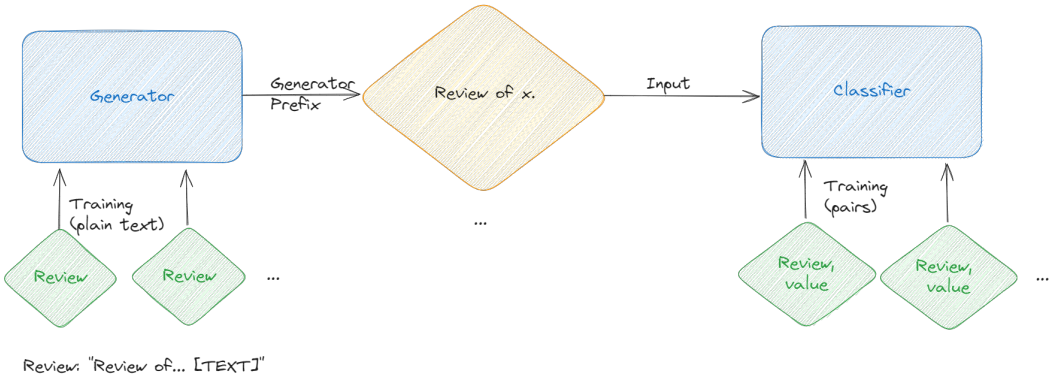

Figura 13: Arquitectura del proyecto. El entrenamiento del generador se realizará con reseñas etiquetadas según la cantidad de estrellas, mientras que el clasificador contendrá pares: reseña y valor. Para evaluar el rendimiento del generador, se generarán reseñas con el prefijo deseado y luego se las clasificará.

Obtención del *dataset*

Una de las grandes dificultades que trae un proyecto de análisis de datos es encontrar un conjunto que cuente con la suficiente cantidad de entradas para poder entrenar los modelos. En particular, cuando se trabaja con categorías (o clases), como es el caso del puntaje asignado a una reseña, es importante que las entradas estén balanceadas. En caso de haber más entradas con puntaje 5 que 2 por ejemplo, la capacidad generalizadora del modelo será perjudicada. Además, es importante realizar un análisis del *dataset* para determinar cuáles serán las etapas para ejecutar en el preprocesamiento.

Datasets analizados

Reseñas Amazon

Contiene alrededor de 8 millones de reseñas provenientes de películas compradas por la plataforma *Amazon* [22]. Si bien tiene una cantidad considerable de reseñas, este *dataset* es descartado debido a que el contenido de estas refiere más al producto (por ejemplo, las condiciones de envío, el *dvd*, etc.) que a la película en sí.
Entrada de ejemplo:

```
I have all of the doo wop DVD's and this one is as good or better than
the 1st ones. Remember once these performers are gone, we'll never get
to see them again. Rhino did an excellent job and if you like or love doo
wop and Rock n Roll you'll LOVE this DVD !!
```

IMDB: *Movie Review Dataset*

Se decide utilizar este *dataset* [23], ya que cumple con los requisitos solicitados: cuenta con un millón de reseñas que hablan exclusivamente acerca de las películas. Inicialmente, las películas están clasificadas del 1 al 10, por lo que se aplica una transformación de los puntajes para reducir la complejidad de los modelos. Se puede ver en la Figura 14 que hay una notoria diferencia en la cantidad de reseñas con 5 puntos en comparación con el resto. Para solucionar el desbalance, el preprocesamiento se encarga de que todos los puntajes tengan la misma participación en el *dataset* que se proporciona al modelo para entrenar.

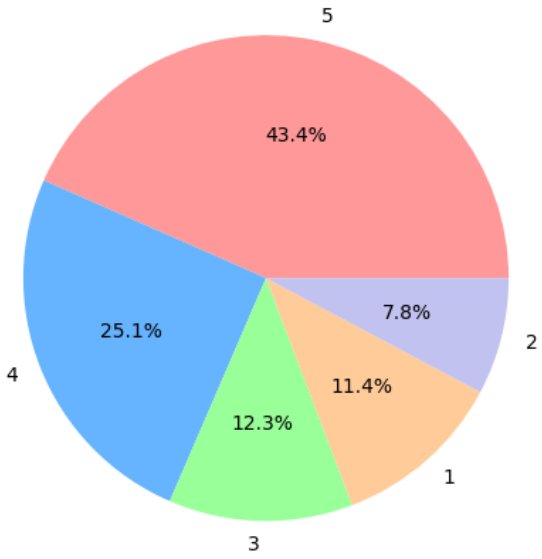

Figura 14: Distribución del puntaje de las reseñas.

Preprocesamiento del *dataset*

Uno de los principales requisitos a la hora de seleccionar el *dataset* es que el conjunto de datos ya esté etiquetado con su puntaje, ya que la implementación propuesta para el generador requiere agregar un prefijo al principio y el clasificador utiliza las etiquetas para entrenar.
Antes de entrenar cualquiera de los modelos, es necesario efectuar una limpieza sobre los datos para no interferir sobre la etapa de aprendizaje, además de eliminar datos inválidos (por ejemplo, columnas con valores nulos o vacíos).

Para demostrar las etapas del preprocesamiento, se toma como referencia la siguiente reseña:

```
"Avengers: Endgame 2019;Something like this movie won't happen again.
This is literally the best superhero movie ever!!! <br/><br/>Thank u MCU
☺;10 "
```

Primero se realiza un análisis de todos los símbolos y palabras presentes en el *dataset*. De esta forma se concluye que es necesario:

- Eliminar *emojis*, hipervínculos ("http://"), etiquetas de **HTML** (**
**) y cualquier símbolo fuera de letras, números, comillas (simples y dobles), paréntesis, signos de puntuación y exclamación.
- Recortar todos los espacios, signos de puntuación y exclamación (es decir, si existe una reseña con "!!!!!", quedará únicamente "!").
- Eliminar las reseñas que contienen insultos (caracterizados por palabras censuradas con asteriscos) y aquellas que tienen puntaje nulo (representado como "*Null*").

```
"Avengers: Endgame 2019;Something like this movie won't happen again.
This is literally the best superhero movie ever! Thank u MCU ;10 "
```

Luego se transforma el puntaje de las reseñas utilizando la Fórmula 13.

$$\text{rating} = \lceil \frac{\text{rating}}{2} \rceil \tag{13}$$

```
"Avengers: Endgame 2019;Something like this movie won't happen again.This
is literally the best superhero movie ever! Thank u MCU ;5 "
```

En el caso de la red generadora, a cada una de las reseñas se le antepone el prefijo que contiene el puntaje.

```
"Avengers: Endgame 2019;Review of 5. Something like this movie won't
happen again.This is literally the best superhero movie ever! Thank u MCU
;5 "
```

Finalmente, antes de presentarle al modelo los datos, se crea un conjunto "balanceado" donde todos los puntajes tienen la misma cantidad de reseñas.

Partiendo de la Figura 14, se puede observar este balanceo en la Figura 15.

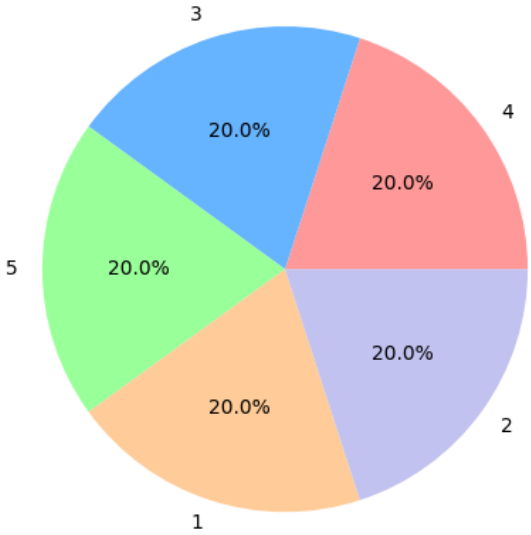

Figura 15: Distribución del puntaje de las reseñas (balanceadas).

Entrenamiento de los modelos

HuggingFace

Para la implementación de los modelos y su entrenamiento se hace uso de la librería *HuggingFace* [24], que ofrece diversos métodos para poder cargar, entrenar y guardar distintas arquitecturas y *tokenizadores* (estos se encargan de aplicar técnicas de preprocesamiento del texto, además de obtener los *tokens* a partir de la secuencia original y asociarlos con su respectivo *embedding*) con foco en el procesamiento del lenguaje natural.
Esta cuenta con una extensa documentación acerca de las distintas tareas que se pueden realizar, además de una gran cantidad de modelos preentrenados que se encuentran subidos a la plataforma (como es el caso de *distilgpt2* o *distilbert-base-uncased*).

Este proyecto utiliza principalmente el módulo *Trainer*, el cual facilita el entrenamiento de un modelo base a partir de la configuración de un conjunto de parámetros de interés. Esta clase contiene el ciclo de entrenamiento básico que puede ser personalizado mediante el uso de la clase *"TrainingArgs"* y permite establecer los valores para los parámetros deseados, además de definir algunos comportamientos específicos (como por ejemplo qué métrica utilizar para seleccionar el mejor modelo).
Otra de las funcionalidades importantes que ofrece es poder ejecutar una función al terminar una época de entrenamiento (se presentan todas las entradas del *dataset* una vez al modelo). En el caso de este proyecto, es utilizada para obtener métricas con el estado actual del generador: se pide al modelo actualizado al finalizar una época que genere reseñas para que sean clasificadas por la red clasificadora (siendo esta la entrenada con los parámetros óptimos).

Otro módulo utilizado es *Pipeline*. Este brinda una interfaz similar a una **API** que abstrae al usuario de la configuración del modelo para la realización de una tarea específica. *Pipeline* requiere únicamente el nombre del modelo elegido, la tarea a realizar y el *tokenizador*.

Por último, el módulo *Dataset* permite cargar los *datasets* utilizados de manera eficiente y en el formato que los modelos esperan recibir.

Cabe destacar que los modelos seleccionados recomiendan fuertemente utilizar el *tokenizador* asociado. De lo contrario, es muy probable que no funcione correctamente, por lo que se utilizará el mismo *tokenizador* con el cual fueron entrenados los modelos.

División del dataset

Para el entrenamiento y la evaluación del modelo es necesaria la división de los datos a utilizar para estos pasos. El conjunto de entrenamiento contiene todas las entradas que serán presentadas al modelo para actualizar los pesos de la red. El conjunto de evaluación es de menor tamaño, se utiliza para evaluar los resultados de la red ante casos no vistos durante el entrenamiento y se obtendrán métricas una vez terminada una época. Adicionalmente, se toma un pequeño conjunto de datos que se corresponden al *dataset* de "validación", el cual se selecciona inicialmente y no se utiliza en ninguna instancia del entrenamiento (ni para entrenar ni para calcular las métricas intermedias). El propósito es poder evaluar el rendimiento del modelo con un conjunto de datos que nunca le fueron presentados.

Para esta división se decide tomar el 10 % de los datos como conjunto de validación. De esta manera, el *dataset* restante cuenta con 719.514 entradas.

Para el estudio de hiperpárametros se toman muestras reducidas del 90 % restante, cuyos resultados serán promediados. En estos casos, las muestras entre ambos modelos podrían contener datos en común. Sin embargo, para el entrenamiento final de ambas redes, se decide separar el *dataset* en 2 partes con el objetivo de lograr independencia entre ellas.

Técnicas utilizadas

La métrica que se utiliza para definir el mejor modelo es *Cross Entropy Loss* (también llamada pérdida, Fórmula 14). La misma consiste en el logaritmo de la probabilidad de la secuencia y está intrínsecamente relacionada con los conceptos de entropía y *perplexity* (Fórmula 15).
El primero, en el contexto de teoría de la información, refiere a la cantidad de bits necesarios para codificar: mientras menos bits se necesiten, más información hay al respecto (es decir, que se puede saber con más probabilidad el valor de lo que se está codificando).
El segundo mide cuán extraña es la secuencia para el modelo. Se define como la inversa de la probabilidad de la secuencia normalizada por la longitud. Es decir, mientras más

elevado sea el valor, menos verosímil es la secuencia para el modelo (menor probabilidad indica que el inverso será más elevado).

$$L_{CE} = -\log p(w_i|w_1, w_2, \ldots, w_{i-1}) \tag{14}$$

$$\text{Perplexity(w)} = \sqrt[N]{\frac{1}{p(w_1, w_2, \ldots, w_N)}} \tag{15}$$

Por lo tanto, al entrenar modelos generadores, secuencias coherentes en el contexto entrenado deberían devolver valores bajos de *perplexity* (indicando que al modelo no le "sorprende" la existencia de estas).

Para mejorar tanto el rendimiento en cuanto a métricas como el tiempo que lleva entrenar los modelos, se utilizan diversas técnicas comunes en cuanto al procesamiento del lenguaje natural y al uso de redes neuronales.

- El concepto más importante implementado es el uso de la **GPU**. En particular se utiliza un entorno de desarrollo con el conjunto de herramientas de **CUDA**, por lo que las operaciones matriciales son ejecutadas en la GPU.
- Se decide utilizar como optimizador **AdamW** [25].
- Se utiliza un decaimiento lineal de la tasa de aprendizaje a lo largo de las épocas.
- En lugar de presentar uno por uno los datos a la red, se entrena en *batches* (lotes), es decir, se toma un conjunto de entradas a presentar, se los entrena en paralelo hasta recibir la salida de la red y luego se actualizan los pesos. Es importante remarcar que todos deben tener el mismo largo, por lo que se toma como referencia la mayor longitud y el resto de los datos se rellenan con un *token* especial, generalmente llamado *padding token*.
- Todas las entradas son acortadas a un máximo de 200 *tokens* (aplicando *padding* a aquellas entradas cuyo tamaño es menor a este). La decisión de este valor se fundamenta en la máxima cantidad de *tokens* que acepta el modelo *distilbert* y en el promedio de palabras por reseña (alrededor de 215) del *dataset* luego del preprocesamiento.
- Sumado al entrenamiento en lotes, en vez de actualizar los pesos cada vez que se completa un paso (se obtiene la salida del lote), se acumulan los gradientes y se actualiza luego de n pasos, a esto se lo denomina *gradient accumulation*.
- No todas las variables necesitan estar almacenadas en formatos de 32 bits, al reducir a la mitad la precisión, la velocidad de cómputo aumenta, lo cual afecta a la rapidez con la que el modelo entrena. Esta configuración es la de precisión mixta **fp16**.

En cuanto a los bloques del *transformer*, es fundamental la estrategia de congelamiento de capas. Las arquitecturas que se utilizan consisten en varios bloques de *transformers* apilados, en particular, *distilgpt2* y *distilbert* cuentan con 6 capas (además del *head*, es decir, la última capa específica a la tarea). Seguramente en las primeras capas haya más información relacionada con la gramática del lenguaje y relación entre palabras, mientras que las últimas sean más específicas a la tarea, por lo que resulta conveniente entrenar únicamente las últimas capas dejando las primeras fijas. En particular, se decide entrenar el *head* (la capa final específica a la tarea) y la última capa del *transformer* (la sexta para ambas redes).

Hardware

Para llevar a cabo los distintos entrenamientos y análisis, es necesario utilizar hardware de considerable potencia debido a que no puede ser replicado eficientemente en una computadora convencional. Con este propósito, el ITBA proporciona un entorno Linux alojado en su servidor privado PAMPERO, el cual presenta los componentes detallados en la tabla 2.
La conexión al *host* se establece mediante el protocolo SSH, permitiendo al usuario operar en él de manera análoga a como lo haría localmente.

Category	**Details**
Machine Type	Desktop, Gigabyte B550 AORUS PRO V2
Motherboard	Gigabyte B550 AORUS PRO V2
UEFI	American Megatrends F12, 01/18/2021
CPU	AMD Ryzen 7 5800X, 8 cores, 32 MiB cache
CPU Speed	Avg: 3938 MHz, High: 4574 MHz, Min/Max: 2200/4850 MHz
GPU	NVIDIA Quadro RTX 6000/8000, Driver: NVIDIA 525.105.17

Tabla 2: Hardware perteneciente al host brindado por ITBA en servidor PAMPERO

Elección de configuración óptima

Para encontrar la configuración óptima de los modelos, se realiza un análisis sobre los posibles hiperparámetros partiendo de la base recomendada por *HuggingFace*.
De los hiperparámetros posibles, se estudian los siguientes:

- **cantidad de épocas**: Cuántas veces serán presentados a la red todos los datos del conjunto de entrenamiento. Permite identificar cuando hay sobreajuste (la pérdida para el entrenamiento sigue bajando mientras que para la evaluación empieza a subir).
- ***learning_rate***: Determina qué tan grandes serán las actualizaciones de los pesos del modelo. A mayor valor, el tiempo de entrenamiento se acorta, pero puede no converger a una solución, mientras que un menor valor incrementará la duración del entrenamiento.
- ***batch_size***: Aumenta o disminuye la cantidad de elementos que forman parte del lote de entrenamiento y afecta inversamente al tiempo de ejecución.
- ***AdamW*** β_1 ***y*** β_2: Ambos parámetros se utilizan en el método de optimización **AdamW**.

Para determinar los valores óptimos de los hiperparámetros, se realizan varios entrenamientos para ambos modelos con las siguientes condiciones:

1. Se mantienen fijos los valores recomendados por *HuggingFace* salvo por el parámetro a analizar.
2. Se toman muestras del *dataset* (sin considerar validación) tres veces sin repetidos. Cada muestra es de 50.000 datos. Se toma la misma cantidad de reseñas para cada puntaje.
3. Las 3 muestras se mantienen fijas para la evaluación de los distintos valores del hiperparámetro y de esta forma permitir la comparación.
4. Se promedian los resultados de las 3 muestras.

Con la configuración óptima se evalúan también los siguientes parámetros para obtener los modelos finales:

- **Tamaño del *dataset***: Es ideal contar con gran cantidad de datos de entrenamiento. Sin embargo, a mayor cantidad, mayor será la duración.
- **Porcentaje *train-test***: Determina la cantidad de datos del *dataset* que son utilizados para entrenar y para realizar las evaluaciones. Se mide en porcentaje de evaluación (por ejemplo, 20 % de todo el conjunto). El resultado será el modelo final, por lo que cada modelo será entrenado con una mitad del *dataset* (sin los datos de

validación) y no se tomarán muestras.

Cantidad de épocas

La primera evaluación se enfoca en la cantidad de épocas necesarias para entrenar ambos modelos considerando el sobreajuste. Se entrenan los modelos con máximo 10 épocas y se analiza a partir de qué momento el valor de pérdida en evaluación aumenta considerablemente, mientras que para el entrenamiento sigue disminuyendo (ya que "aprende de memoria" las entradas presentadas).

Como se puede observar en la Figura 16, para el modelo BERT, el sobreajuste se da a partir de 2-3 épocas, por lo que se decide utilizar 3 épocas de este momento en adelante. Por otro lado, en el generador no se observa un claro sobreajuste del modelo, por lo que se toma como valor óptimo 5 épocas.

La cantidad obtenida será utilizada para el estudio del resto de hiperparámetros.
Para la obtención de los modelos finales se considerarán mayor cantidad de épocas, ya que serán entrenados con mayor cantidad de datos.

((a)) ((b))

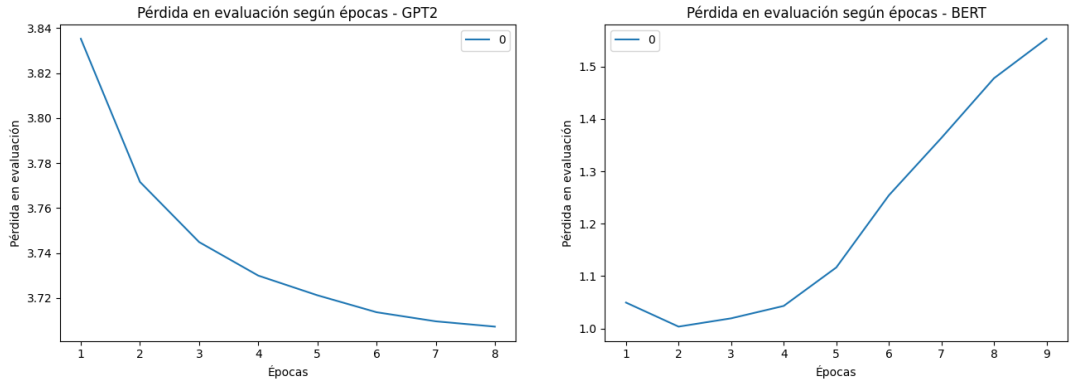

Figura 16: Evolución del valor de pérdida en evaluación en el generador (a) y en el clasificador (b) en función de las épocas.

Learning Rate

Se analiza el impacto de distintos valores de la tasa de aprendizaje para ambos modelos, con la cantidad de épocas establecida y con los valores 1×10^{-6}, 5×10^{-5}, 1×10^{-5} y 5×10^{-4}. En la Figura 17 se observa que para el modelo generador (a), el valor óptimo es 5×10^{-4}. Para el clasificador es 5×10^{-5} (c).

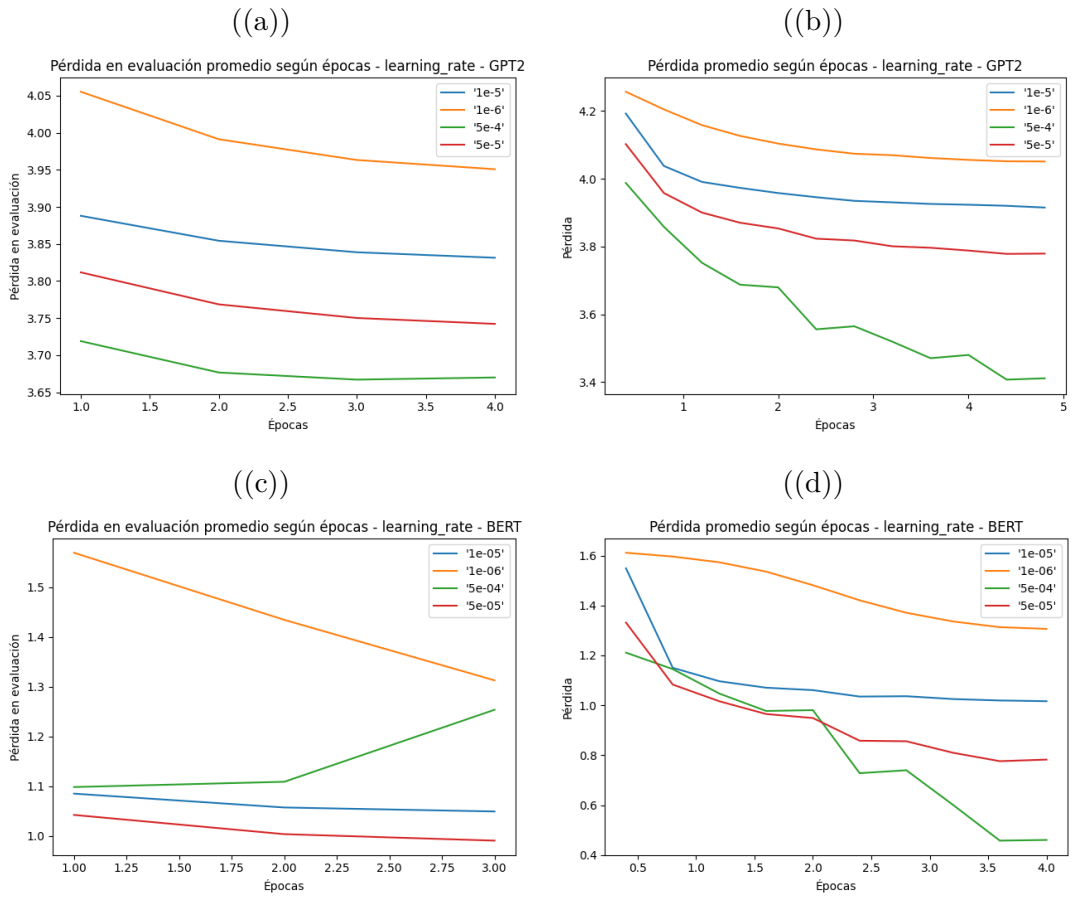

Figura 17: Evolución de pérdida en evaluación y a lo largo del entrenamiento en función del parámetro *Learning Rate* para el generador (a y b) y para el clasificador (c y d).

Como en esta arquitectura el clasificador es el encargado de evaluar a la red generadora, se estudia cómo varía el *Accuracy* [26] en función de la tasa de aprendizaje (Figura 18). Si la tasa de aprendizaje es de 1×10^{-6}, el modelo no logra superar un clasificador aleatorio, ya que se encuentra por debajo del 50 %. Por el contrario, si es la seleccionada como óptima, la tasa de acierto llega al 58.08 %.

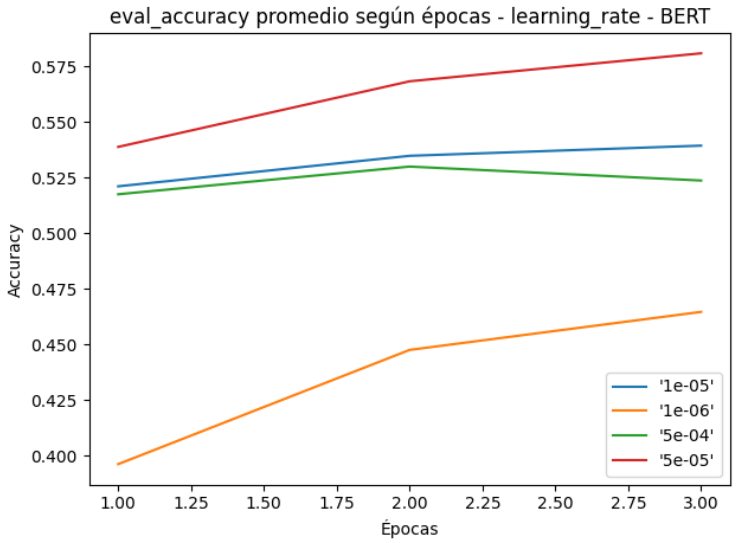

Figura 18: Evolución de *Accuracy* del clasificador en función de la tasa de aprendizaje con 3 épocas.

Si bien clasifica las reseñas correctamente con una tasa de acierto mayor a lo que haría el azar, no es tan alta como se esperaba. Esto puede deberse a que el modelo clasifica "cerca" de lo que debería: por ejemplo, predice un valor que se encuentra a distancia 1 del verdadero. A pesar de no ser un error considerable, las predicciones no son valoradas como correctas para las métricas.
Se decide entonces implementar métricas que consideren correcta una predicción si la distancia es menor o igual a 1 con la etiqueta real. Puede visualizarse en la Fórmula 16 cómo se obtienen las predicciones. En esta, se toma como valor predicho el de la etiqueta real, si la predicción (**P**) está a una distancia menor o igual a 1 que la real (**R**), caso contrario se toma el valor con el que se clasificó. Luego, se calculan las métricas de la misma manera que se definen las originales. A estas se las denomina métricas **Cercanas**.

$$P = \begin{cases} R, & \text{si } |P - R| \leq 1 \\ P, & \text{en otro caso} \end{cases} \tag{16}$$

AdamW β_1

Se analizan los valores 0.85, 0.9 y 0.95. Los mismos no presentan ninguna variación significativa para ninguna de las redes, evidenciado por la Figura 19. Se toma como valor final 0.95.

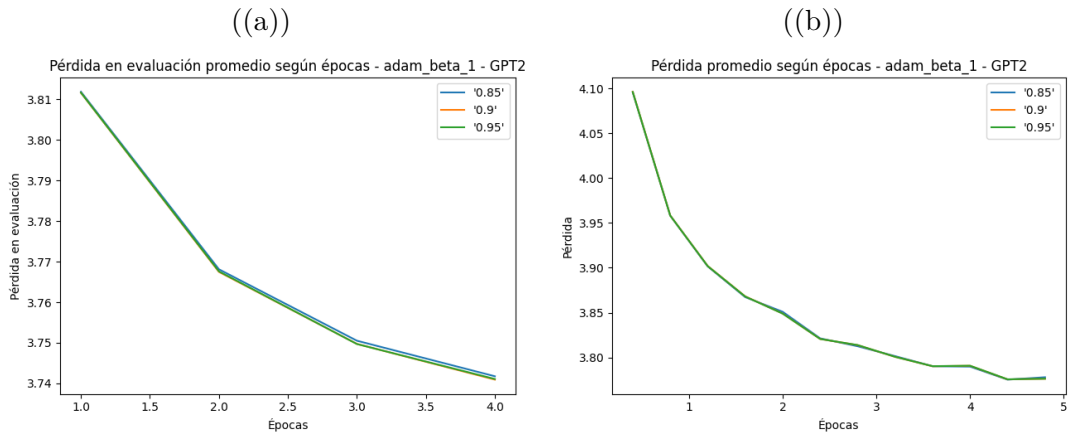

Figura 19: Evolución de pérdida en evaluación (a) y a lo largo del entrenamiento (b) en función del parámetro *Adam* β_1 para el modelo generador.

AdamW β_2

Para β_2 se decide evaluar con los valores: 0.8, 0.9 y 0.999. A diferencia del resultado anterior, los gráficos de la Figura 20 muestran mayores diferencias entre sí, siendo 0.999 el valor óptimo.

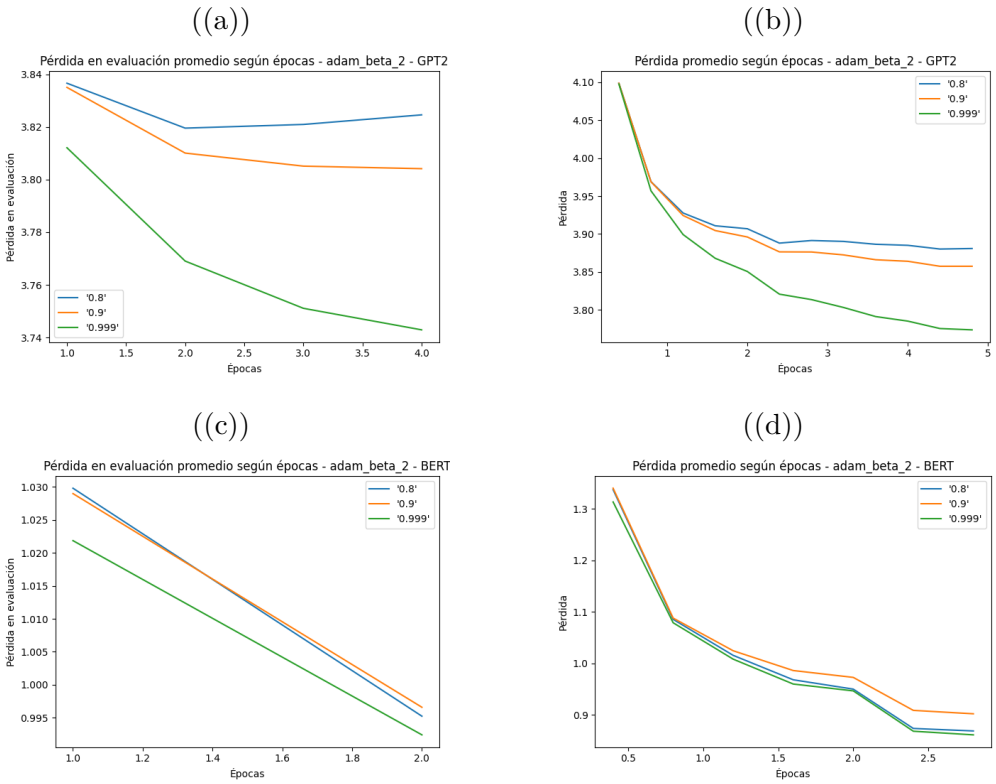

Figura 20: Evolución de pérdida en evaluación y a lo largo del entrenamiento en función del parámetro *Adam* β_2 para el generador (a y b) y para el clasificador (c y d).

Tamaño de lote

Se evalúa el tamaño del lote tanto de entrenamiento como de evaluación. Para ello se utilizan los valores 4, 8 (recomendado por *HuggingFace*) y 16.
Observando los gráficos que se encuentran tanto en la Figura 21 como en la Figura 22, se puede determinar que a medida que el tamaño del lote disminuye, el valor de pérdida también lo hace, mientras que el tiempo de entrenamiento aumenta.
A pesar de la mejora observada al entrenar el modelo con lotes de 4 respecto de 8, no se justifica el tiempo requerido para hacerlo (aumenta en un 17 %), por lo que se decide utilizar tamaño 8.

((a)) ((b))

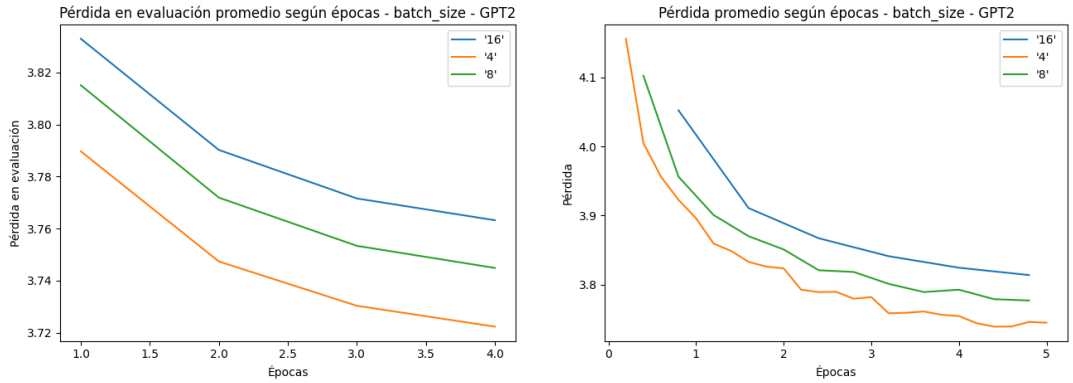

Figura 21: Evolución de pérdida en evaluación (a) y a lo largo del entrenamiento (b) en función del parámetro *batch_size* para el modelo generador.

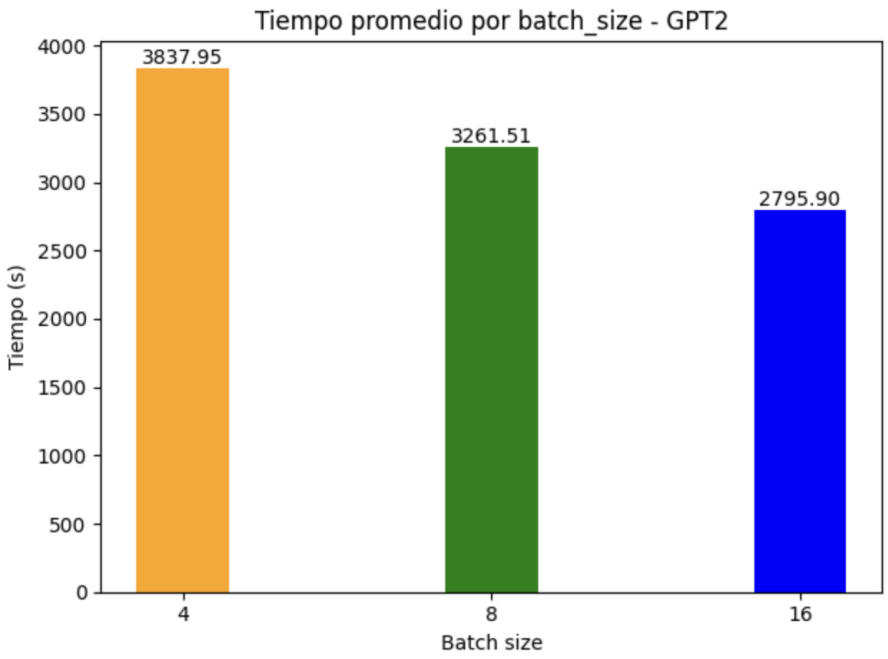

Figura 22: Tiempo promedio de entrenamiento variando el tamaño de lote (evaluación y entrenamiento) para el modelo generador.

Hiperparámetros óptimos

Se concluye que los valores óptimos para ambos modelos son los que se encuentran en la Tabla 3.

Hiperparámetro	**Valor**
`evaluation_strategy`	epoch
`num_train_epochs`	**5 (GPT-2) | 3 (BERT)**
`log_level`	error
`save_strategy`	epoch
`fp16`	true
`per_device_train_batch_size`	**8**
`per_device_eval_batch_size`	**8**
`gradient_accumulation_steps`	**4**
`load_best_model_at_end`	true
`optim`	adamw_torch
`learning_rate`	$\mathbf{5 \times 10^{-4}}$ **(GPT-2)** | $\mathbf{5 \times 10^{-5}}$ **(BERT)**
`lr_scheduler_type`	linear
`weight_decay`	0.1
`adam_epsilon`	1×10^{-8}
`adam_beta1`	**0.95**
`adam_beta2`	**0.999**
`disable_tqdm`	true
`overwrite_output_dir`	true
`warmup_ratio`	0.1
`do_eval`	true

Tabla 3: Configuración óptima de los hiperparámetros, tanto para el clasificador como para el generador.

Cantidad de datos

Para ambos modelos se utiliza el mismo conjunto de datos que en la búsqueda de hiperparámetros óptimos. Además, se evalúan más épocas que en las secciones anteriores debido a la diferencia en la cantidad de datos con la que se entrena.

BERT

Se entrena el clasificador con 3 subconjuntos de tamaños 10.000, 50.000 y 100.000 para 10 épocas y se obtienen los resultados visibles en la Figura 23. Como el gráfico indica, independientemente de la cantidad de datos, el modelo comienza a sobreajustar entre 2 y 4 épocas. Es importante destacar que, a mayor cantidad de datos, sobreajusta más

lento. Dado que el valor de la métrica *Accuracy* es máxima en la época 2 con un valor de 59.07 %, se considera que esta es la cantidad de épocas máxima antes de que el modelo sobreajuste para un conjunto de datos de 100.000.

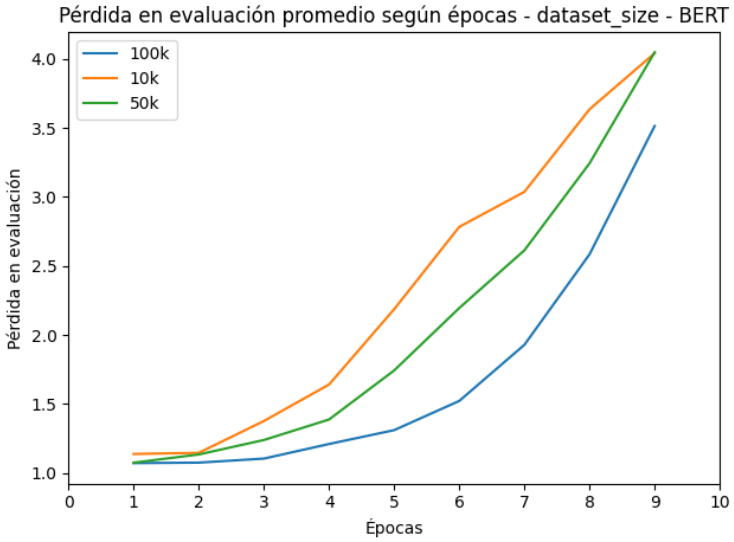

Figura 23: Evolución de la pérdida de evaluación en función de la cantidad de datos presentados para el clasificador.

GPT-2

Se entrena el generador de igual manera: 3 subconjuntos independientes para la cantidad de datos 10.000, 50.000 y 100.000 con 10 épocas cada uno. Además de medir la pérdida de evaluación (Figura 24 (a)), al terminar cada época se generan 250 reseñas de cada puntaje con el modelo entrenado hasta ese momento y se realizan predicciones con la red clasificadora (entrenada con la configuración óptima) para obtener la métrica *Accuracy*. Como se puede ver en la Figura 24 (b), el modelo comienza a sobreajustar en la segunda época para la menor cantidad de datos, mientras que para los demás, esto se da a partir de la tercera o cuarta época. También es observable que la curva correspondiente a los 100.000 datos es menos pronunciada.
Por su parte, el valor de la tasa de acierto del clasificador sobre las reseñas generadas aumenta. Esto puede deberse a que las predicciones no sean representativas del modelo generador (ya que únicamente se predicen 1.250 reseñas), además que podría estar generando reseñas muy parecidas a las presentadas durante el entrenamiento (debido al sobreajuste). Se puede ver que, a mayor cantidad de datos, mejores resultados, por lo que se decide tomar 100.000 datos.

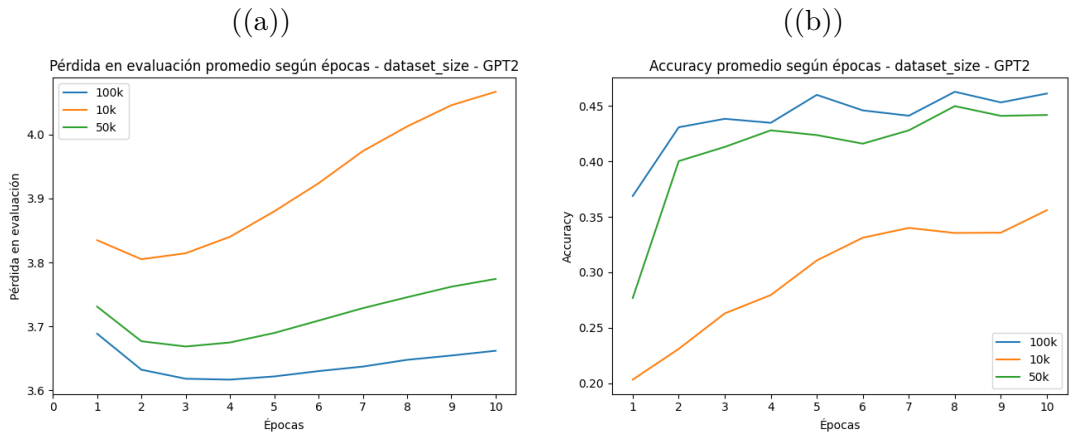

Figura 24: Evolución de la pérdida de evaluación (a) en función de la cantidad de datos y el valor de *Accuracy* sobre la predicción de las reseñas (b) para el generador.

Porcentaje *train-test*

Se estudia también cómo impacta el porcentaje de datos de entrenamiento y evaluación en las métricas de los modelos. Como el resultado de estos entrenamientos serán los modelos finales, se utiliza el *dataset* dividido en 2, por lo que cada modelo es entrenado con su mitad correspondiente y no se toman muestras para ser promediadas.
Se utilizan conjuntos de evaluación que representan el 10 %, 20 % y 30 %.

BERT

Los resultados observables en la Figura 25 permiten determinar que la tasa de acierto es más alta cuando la división es de 90 % para entrenamiento y 10 % para evaluación, además de tener el menor valor de pérdida al finalizar la segunda época.

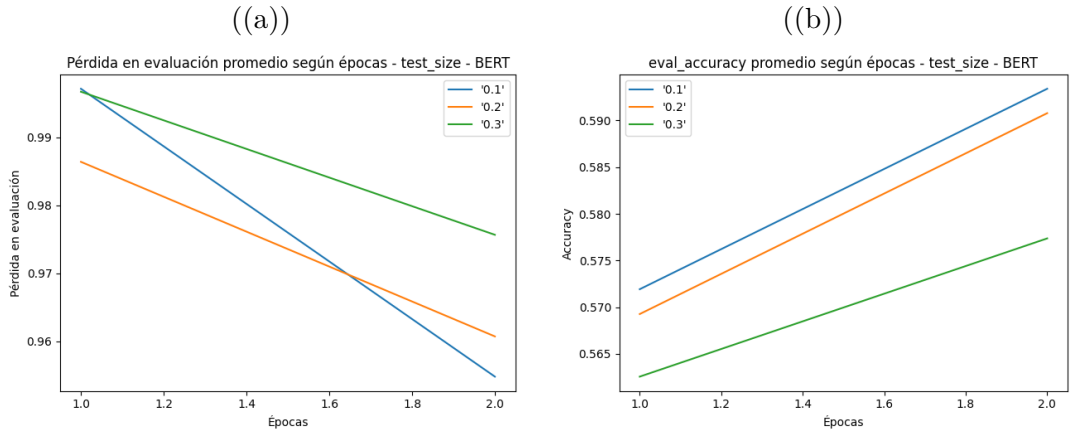

Figura 25: Gráfico de pérdida en evaluación (a) en función del porcentaje de datos y *Accuracy* (b) del clasificador, siendo 0.1 (90/10), 0.2 (80/20) y 0.3 (70/30) los valores analizados.

GPT-2

Se realiza el mismo estudio para el generador, considerando la mitad restante del *dataset* (sin validación). En la Figura 26 se puede ver que hay una gran diferencia en la pérdida de evaluación cuando se dividen los datos en 90 % de entrenamiento y 10 %. En este, el sobreajuste es menos pronunciado.

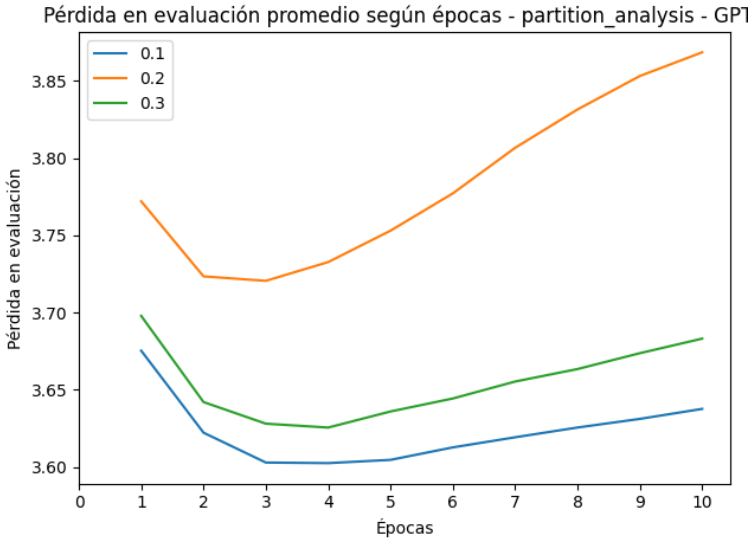

Figura 26: Gráfico de pérdida en evaluación (a) en función del porcentaje de datos y *Accuracy* de las reseñas generadas (b), siendo 0.1 (90/10), 0.2 (80/20) y 0.3 (70/30) los valores analizados para el generador.

A su vez, se clasifican las reseñas generadas por el modelo al finalizar cada época con el clasificador óptimo y se calculan *Accuracy* y *Accuracy cercana*, obteniendo así las curvas presentadas en la Figura 27.

((a)) ((b))

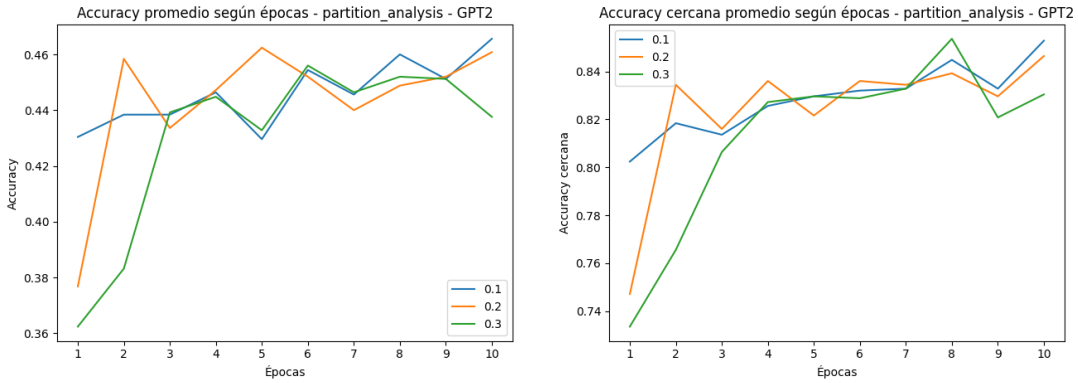

Figura 27: Evolución de Accuracy (a) y Accuracy Cercana (b) en función de la división *train-test*, siendo 0.1 (90/10), 0.2 (80/20) y 0.3 (70/30) los valores analizados para el generador.

Nuevamente, los valores de *Accuracy* son cercanos al clasificador aleatorio, sin embargo, al considerar la tolerancia, aumenta considerablemente (oscila el 85 %). Para ambos modelos, la división 90/10 es la que obtiene mejores resultados.

Modelos finales

BERT

Dado que la red clasificadora supervisará al generador, se decide realizar un análisis más granular del sobreajuste: se obtienen métricas cada cierta cantidad de pasos (sobre el conjunto de evaluación), en lugar de hacerlo al finalizar cada época. Los resultados se pueden ver en la Figura 28. En el gráfico (a) se compara la pérdida en el conjunto de entrenamiento y en el de evaluación. El comportamiento es el esperado: en el conjunto de evaluación la pérdida es mayor que en el de entrenamiento, ya que evalúa sobre datos que el modelo no utilizó para entrenar.
Por otro lado, en el gráfico (b), considerando la tolerancia de un error menor o igual a 1, se pueden llegar a valores de acierto del orden del 90 %. Es notorio que acierta con una tasa que oscila el 93.33 %, mientras que, si no se tiene en cuenta la mencionada tolerancia, baja a 59.21 %. Este modelo es el utilizado para clasificar las reseñas generadas. Específicamente, la versión entrenada con 2 épocas, ya que luego comienza a sobreajustar.

((a)) ((b))

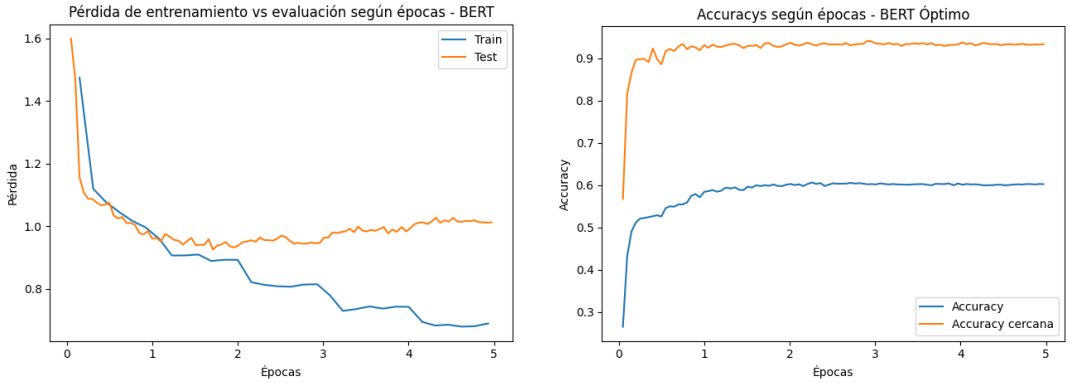

Figura 28: Evolución de la pérdida de evaluación y entrenamiento (a) para el clasificador, considerando mayor cantidad de pasos para evaluar. *Accuracy* vs. *Accuracy cercana* en función de las épocas (b).

Finalmente, se utiliza el conjunto de validación para evaluar el modelo obtenido. En la Figura 29 se puede observar la matriz de confusión multiclase. Es notorio destacar la predominancia de la diagonal. Si bien los valores están por encima del 50 %, se destacan las puntas: las reseñas con puntajes más altos y bajos.

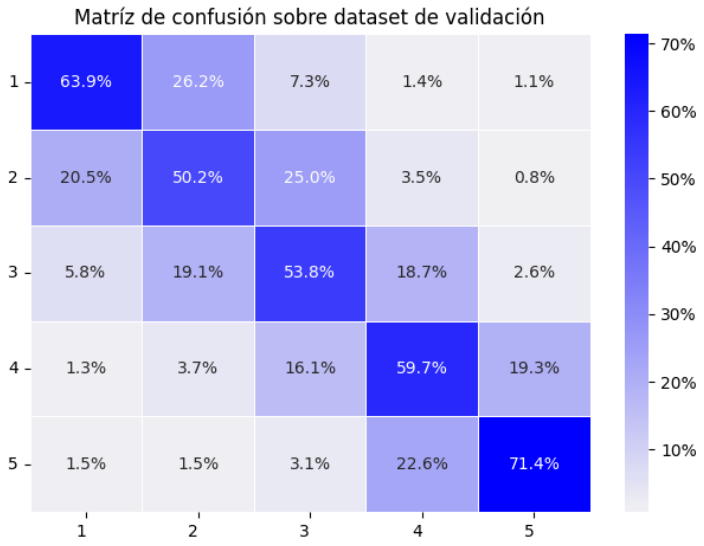

Figura 29: Matriz de confusión con porcentaje de acierto entre cada puntaje utilizando el conjunto de validación y el clasificador óptimo.

Las métricas correspondientes a la matriz se encuentran en la Tabla 4.

Métrica	**Original**	**Cercana**
Accuracy	60.31 %	93.17 %
Recall	60.31 %	93.17 %
Precision	59.78 %	93.19 %
F1-Score	59.89 %	93.16 %

Tabla 4: Métricas de validación obtenidas para el clasificador sobre el conjunto de validación. En la segunda columna se encuentran los valores de las métricas originales, mientras que en la tercera se considera una diferencia de 1 o menos como una predicción correcta.

GPT-2

Para evaluar el rendimiento del generador, se utiliza el mismo conjunto de validación utilizado para el clasificador. Se presentan las entradas y se calcula el *perplexity*. Cuanto más grande es la métrica, más sorprendido está el modelo, por lo que, al tratarse de reseñas de películas, se espera que el valor sea menor que para el modelo sin entrenar. Como se puede observar en la tabla 5, *distilgpt2* sin entrenar tiene un *perplexity* más alto que el modelo óptimo encontrado.

Perplexity	**Valor**
distilgpt2 **Inicial**	61.10
distilgpt2 **Entrenado**	46.28

Tabla 5: *Perplexity* calculado sobre el conjunto de validación para *distilgpt2* sin *fine-tuning* y para el modelo final entrenado.

Además, para verificar la confiabilidad de los puntajes de las reseñas generadas, se utiliza el clasificador óptimo para realizar predicciones sobre un conjunto de datos generados con el generador óptimo. Las métricas se encuentran en la Tabla 5

Métrica	**Original**	**Cercana**
Accuracy	43.92 %	82.64 %
Recall	43.92 %	82.64 %
Precision	41.97 %	84.58 %
F1-Score	41.39 %	82.28 %

Tabla 6: Métricas obtenidas sobre un conjunto de 2.500 reseñas (500 por cada puntaje) generadas con la red generadora óptima. En la segunda columna se encuentran los valores de las métricas originales, mientras que en la tercera se considera una diferencia de 1 o menos como una predicción correcta.

Conclusión

Queda demostrada la efectividad de la técnica implementada en un modelo generador para abordar el problema de la disponibilidad limitada de datos. Al incorporar el prefijo de etiqueta deseada, se pueden generar datos que complementan al *corpus* inicial.
La evaluación de los datos generados permite verificar la confiabilidad de estos. Al evaluar el clasificador óptimo obtenido sobre el conjunto de validación, supera al clasificador aleatorio para las 5 clases y alcanza una tasa de acierto considerable (93.17 %, teniendo en cuenta errores menores o iguales a 1), por lo que se puede considerar adecuado para supervisar al generador. Al obtener métricas similares sobre el *dataset* generado, se puede concluir que los datos generados por la red son confiables y útiles para el propósito específico del problema abordado.

Trabajos futuros

Una de las mejoras propuestas es trabajar con modelos más grandes, en lugar de utilizar la versión *distil* de los mismos. Esto requeriría un *hardware* más especializado y tiempos mucho más largos de entrenamiento. A su vez, se podría realizar una etapa de preprocesamiento más detallada, como por ejemplo transformar las palabras sintácticamente mal redactadas para que sean correctas.

In context learning

Con la salida de GPT-3, surge la posibilidad de entrenar sobre la base de un contexto, es decir, a la hora de presentar un *prompt* (prefijo, instrucción al modelo) hacerlo en formato de "tarea", con o sin ejemplos. Esto permite realizar *transfer learning* sin *fine-tuning*, actualizaciones de gradiente, ni capas adicionales. Por ejemplo, un posible *prompt* podría ser "Traducir de español a inglés la siguiente frase: Hola, ¿cómo estás?". Existen diversas variantes [27]:

1. *Zero-Shot*: solamente el *prompt*.
2. *One-Shot*: se utiliza un caso de prueba, por ejemplo "Me gusta el fútbol → I like football".
3. *Few-Shot*: se utilizan varios ejemplos.

Esto permite que sea posible resolver distintas tareas solo con el *LM*, sin tener que cambiarle el *head* para ajustarlo a una nueva tarea objetivo. Es un caso interesante para analizar, ya que es un comportamiento emergente: no fue entrenado para eso.

PEFT

PEFT [28] (*fine tuning* eficiente en parámetros, por sus siglas en inglés) es una estrategia que surge producto de los largos tiempos que implican hacer *fine-tuning* sobre un modelo preentrenado. Si bien utilizar un modelo preentrenado trae sus ventajas y también existe la posibilidad de utilizar una versión reducida del mismo (*distil*), siguen siendo parámetros del orden de millones. PEFT busca trabajar sobre un conjunto reducido de los mismos, sin perder rendimiento. Existen tres alternativas:

1. *Prompt tuning*: agrega un prefijo al principio de la secuencia para identificar la tarea a realizar. Permite resolver distintos problemas al cambiar únicamente el *prompt*, por ejemplo, se podría pedir "Clasificar la siguiente secuencia en positiva o negativa" y únicamente se actualizarán los pesos relacionados con este prefijo, buscando el óptimo para la tarea (similar al aprendizaje *few-shot*). El prefijo puede ser inicializado de manera aleatoria o se puede dar uno inicial que se ajustará. Hay que tener en cuenta que, al agregar este prefijo, se reduce la cantidad de *tokens* máximos que puede procesar el modelo.

2. *Prefix tuning*: igual a *prompt*, la diferencia es que el prefijo se agrega a todas las capas del *transformer*.

3. *LORA*: se demostró que cuando un *LM* se ajustaba a una nueva tarea, los pesos se podían representar en una matriz de menor rango [29], por lo que, al actualizar los pesos, las operaciones son más eficientes, ya que no hay que actualizar todas las matrices.

Referencias

[1] Linus Ericsson et al. «Self-supervised representation learning: Introduction, advances, and challenges». En: *IEEE Signal Processing Magazine* 39.3 (2022), págs. 42-62.

[2] Irina Pak y Phoey Lee Teh. «Text segmentation techniques: a critical review». En: *Innovative Computing, Optimization and Its Applications: Modelling and Simulations* (2018), págs. 167-181.

[3] Jonathan J Webster y Chunyu Kit. «Tokenization as the initial phase in NLP». En: *COLING 1992 volume 4: The 14th international conference on computational linguistics*. 1992.

[4] Tze Yuang Chong, Rafael E Banchs y Eng Siong Chng. «An empirical evaluation of stop word removal in statistical machine translation». En: *Proceedings of the Joint Workshop on Exploiting Synergies between Information Retrieval and Machine Translation (ESIRMT) and Hybrid Approaches to Machine Translation (HyTra)*. 2012, págs. 30-37.

[5] Divya Khyani et al. «An interpretation of lemmatization and stemming in natural language processing». En: *Journal of University of Shanghai for Science and Technology* 22.10 (2021), págs. 350-357.

[6] Tomas Mikolov et al. «Efficient estimation of word representations in vector space». En: *arXiv preprint arXiv:1301.3781* (2013).

[7] Murat H Sazli. «A brief review of feed-forward neural networks». En: *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering* 50.01 (2006).

[8] John Bridle. «Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters». En: *Advances in neural information processing systems* 2 (1989).

[9] Razvan Pascanu, Tomas Mikolov y Yoshua Bengio. «On the difficulty of training recurrent neural networks». En: *International conference on machine learning*. Pmlr. 2013, págs. 1310-1318.

[10] Sepp Hochreiter. «The vanishing gradient problem during learning recurrent neural nets and problem solutions». En: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), págs. 107-116.

[11] Sepp Hochreiter y Jürgen Schmidhuber. «Long short-term memory». En: *Neural computation* 9.8 (1997), págs. 1735-1780.

[12] Ashish Vaswani et al. «Attention is all you need». En: *Advances in neural information processing systems* 30 (2017).

[13] Kawin Ethayarajh. «How contextual are contextualized word representations? Comparing the geometry of BERT, ELMo, and GPT-2 embeddings». En: *arXiv preprint arXiv:1909.00512* (2019).

[14] Jacob Devlin et al. «Bert: Pre-training of deep bidirectional transformers for language understanding». En: *arXiv preprint arXiv:1810.04805* (2018).

[15] Philipp Koehn. *Statistical machine translation*. Cambridge University Press, 2009.

[16] Leo Breiman. «Random forests». En: *Machine learning* 45 (2001), págs. 5-32.

[17] Colin Raffel et al. «Exploring the limits of transfer learning with a unified text-to-text transformer». En: *The Journal of Machine Learning Research* 21.1 (2020), págs. 5485-5551.

[18] Alec Radford et al. «Language models are unsupervised multitask learners». En: *OpenAI blog* 1.8 (2019), pág. 9.

[19] URL: https://blog.salesforceairesearch.com/the-wikitext-long-term-dependency-language-modeling-dataset/.

[20] Victor Sanh et al. «DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter». En: *arXiv preprint arXiv:1910.01108* (2019).

[21] Alex Wang et al. «GLUE: A multi-task benchmark and analysis platform for natural language understanding». En: *arXiv preprint arXiv:1804.07461* (2018).

[22] URL: https://snap.stanford.edu/data/web-Movies.html.

[23] URL: https://ieee-dataport.org/open-access/imdb-movie-reviews-dataset.

[24] URL: https://huggingface.co.

[25] Ilya Loshchilov y Frank Hutter. «Decoupled weight decay regularization». En: *arXiv preprint arXiv:1711.05101* (2017).

[26] Mohammad Hossin y Md Nasir Sulaiman. «A review on evaluation metrics for data classification evaluations». En: *International journal of data mining & knowledge management process* 5.2 (2015), pág. 1.

[27] Yaqing Wang et al. «Generalizing from a few examples: A survey on few-shot learning». En: *ACM computing surveys (csur)* 53.3 (2020), págs. 1-34.

[28] Neil Houlsby et al. «Parameter-efficient transfer learning for NLP». En: *International Conference on Machine Learning*. PMLR. 2019, págs. 2790-2799.

[29] Edward J Hu et al. «Lora: Low-rank adaptation of large language models». En: *arXiv preprint arXiv:2106.09685* (2021).

Anexo

Reseñas generadas con el modelo sin entrenar

Puntaje: 1

- Review of 1.For its length and depth, it is the best book of the three main series: The Black List, The White List and the New Book.
- Review of 1. The best part, after all, is the amount of time they have spent in there by each of us.
- On May 9, 2013, the first of their 4 episodes are re-aired as a followup, titled "The Day When I Go On; on May 8, 2013, they also return to episode 4, "P.S. It all took place a couple days for the other two shows to catch up to the fact that the show has been doing all right.

Puntaje: 2

- Review of 2. Â The only really scary thing about this set to go is that it's a way for the characters to find themselves in a way that is a little like your favorite movie but they can't do this right.
- Review of 2. In an excerpt from the book titled "Easily I Tried to Give Back", John S. Fisher, and Barbara J. Schuett, all of whom published the book, describes the personal struggle to get back on the road. At the very least, it provides a powerful glimpse into the path of the past, and the lessons they learned.
- Review of 2. The next day, Mr Riz, who works at the business office of the State Department in Washington, D.C., was forced to resign from the position at a company called Zoning, which has many employees. The company claims it is owned by a handful of Washington office workers.

Puntaje: 3

- Review of 3. This is what I need.
- Review of 3. I don't know if I feel the need to go to a gym, but if it turns out that my mind could work in that direction I may just get a little creative as I do. The following article provides a good starting point to start the day and help you take this much closer to your full-time career. I'll be putting on plenty of work before I dive into some personal writing skills and get started.
- You are going to want to make sure the new device you are using will be properly formatted by XDA, too — you need to make sure it has a number of tabs. Next, go to Settings > XDA.

Puntaje: 4

- Review of 4. The price is 24, but the base is 39.99 (with the price is 38.99).
- Review of 4. I have seen that as the most powerful and strong thing in a game.On the other hand its not really a game that you are just playing with.
- Review of 4. We could have easily been more involved (and even now), especially since in 2015 we needed more of our team if we wanted to win a championship in the NHL!

Puntaje: 5

- Review of 5. The following is the list by which the average temperature in one cup of coffee is measured. The ""Temperature in Coffee""value represents the number of cups that have been sold or shipped from one cup (samples) of a coffee that has been shipped or sold in one container. The average value for each cup of coffee is measured using both the ""Temperature in Coffee""and ""Temperature in Coffee""values; both ""Temperature in Coffee""...
- Review of 5. It is only a matter of time before the new version comes out and features are included in the pack that are just as good and unique and not just on any download site.
- Review of 5. ____ This article is a good and informative introduction to the great thing about writing good scripts. I am one to look at and check out and share a list of the best scripts I can write.

Reseñas generadas con el modelo entrenado

Puntaje: 1

- Review of 1. This movie was awful.No one has ever watched it.I hated it because it kept the movie flowing forward and did not feel like it was at all entertaining and not boring.I had just left the theatre and not enough time to give it a chance because it was a complete let down.The first time I saw this movie I thought it was good.It just doesn't work if you are a comic book fan or a Marvel fan...
- Review of 1. There is no real logic behind why somebody would think that the makers of this movie and others have taken the initiative.What did it mean if they had the courage to do all kinds of crap that they would have in the history of war?.I guess what would have happened if they had made no effort to tell the audience that the war was not over...
- Review of 1. It was an interesting film with good acting by Robert De Niro, but the story was poorly told.This film had none in common with other mafia films.I don't

recommend that this film be seen as a cheap watch...

Puntaje: 2

- Review of 2. The movie was really boring, I expected.It made no sense.It did not even deserve a single viewing. What really made it interesting was how the film was not even remotely scary at least. The first 15 minutes were really boring.And the rest were really silly.The movie really wasn't interesting enough to keep you on the edge of your seat.In the beginning I think the movie ended okay with the monster...
- Review of 2. The plot sucks, not a good story.Don't be fooled though! Not worth your time to watch.Seth Rogen is in every way a great actor, and with it is a new level of special effect, which you should see with some kind of open hearted, if not hearted drama...
- Review of 2. The film could have been much more serious! The first part of the film is so very, very boring.The first part is a man who becomes a recluse.He has to give up, and tries to keep his head.He also gets lost in the hospital, and tries to find someone else, he is now a drug dealer with the plans to get him the money to get him.One day in a hospital that seems like a safe haven he is attacked by a guy next door.He has to be hospitalized so he can return what he paid for, but a guy also killed her...

Puntaje: 3

- Review of 3. I expected too much from this film.But it never lost it.It had the same tone it had when it was first released, but it lacked the warmth of the old Batman which made for a very interesting Batman film in the end, the same kind of movie with a darker side or deeper which really made for a great Batman film! The Dark Knight Rises was also very good.I'm still surprised by some of the changes made with these new movies, and why this is considered one of the best Batman sequels ever made it is that it is actually not an original one, just changes that...
- Review of 3. In fact, this film had a tendency to overdo it, and often just didn't achieve the intended effect that was expected.I am baffled by people saying this film couldn't even have been cut.While the plot was very intriguing, I was sure it would have easily fallen into the trap of being over, rather than getting rid of it.The special effects are incredible, but not enough to keep a viewer interested for a full two hours...
- Review of 3. A good movie.A great movie.But it just didn't make sense like in The Sixth Sense.And even the movie itself lacked a story.The ending left me with the impression that this ending wasn't worth the money.There were a lot of things happening in the movie that can be put together ...

Puntaje: 4

- Review of 4. I was a little upset when the trailer for this movie came out.It was not a movie that I expected, and as I am normally the sole one looking in for, it was not a movie that I really expected.The first was a really weird, weird movie.The movie had some sort of atmosphere and a little drama, but it was not really that scary.The first time I saw the trailer, I thought that the actors were trying to give the impression that the movie was scary, but then I realized that people were saying that it doesn't really scare us.If you are one of those people who is trying to give a bad scare to you, I think this can definitely be one of the greatest horror movies that has ever been made...
- Review of 4. This one is just OK, but why do I say to my children like that when they aren't all very nice at school at least, they don't try to be nice or really like them.I've seen all kinds of movies like these but what this one really makes me more attracted to is that the movie does not leave you feeling like you are part of the story.On the other hand, what this one stands out to you is the feeling of an innocence that makes you think.I think this kind of feeling is very different from that in other movies.
- Review of 4. How do we know that one of my new favorite comedies will win an oscar? Or is it a cult following? Maybe it is because of its message on family issues and social pressures.Maybe the message might be too far fetched, too realized and too subtle (but it's not to deny that this movie was entertaining, because it is so close to a masterpiece), but it is a mustsee at any time...

Puntaje: 5

- Review of 5. Intense, fun, funny, and totally worth waiting for.My vote was 7.0.For all these reasons, this movie is for people who are not into sports or baseball, or just as baseball is a baseball movie.There is no sports movies out there.And the movie contains great plot holes.I must say, this movie deserves 10 points of this movie.It's also not going the way it needs to.It's not for everyone.Even the average person can understand baseball's plot.It's very easy to overlook the bad aspects of this movie.Because this movie contains a great cast.
- Review of 5. That was one of the best movies I have ever seen.If you think the movie about that subject was a little too heavy handed, then don't bother seeing the movie about that subject.When the movie begins, you are left with a great time.I would like to thank everyone involved for making the movie work.I couldn't even believe the hype surrounding that movie but I just want to thank those who did it.And please, if you like it, I want to thank that one big thank you because I cannot wait till the end of this movie! Now this is my favorite movie of all times...

- Review of 5. Like most critics I have read, I have been one of the few movies that are based on an intelligent story to understand things.I watched this movie because i am not biased in it.It is a love story with great special effects.The plot of this movie is simply mind blowing, which really makes it a real feast for the eyes...