GRADO EN CIENCIA DE DATOS

Emotion recognition with a multimodal approach in conversations

Presentado por:

RODRIGO FLORENCIO RIVAS ARÉVALO

Dirigido por:

VICTOR MANUEL YESTE MORENO

CURSO ACADÉMICO 2023-2024

Resumen y palabras claves (castellano)

El reconocimiento de las emociones es y ha sido un punto clave en nuestra sociedad para entendernos los unos a los otros a lo largo de la historia, puesto que somos animales sociales, tenemos que saber cómo hacemos sentir a los demás para poder sobrevivir. Es por eso por lo que hoy en día con la revolución tecnológica que estamos viviendo con la inteligencia artificial el análisis de estas puede ser algo mucho más sencillo para nosotros y para los demás, ya que el análisis de estas es un tema tremendamente interesante y valioso para nosotros, pero que solo se suele abordar de una manera simple y con un solo enfoque, ya que se intenta analizar los sentimientos en textos aislados sin tener en cuenta que forman parte de un todo, de una conversación, que puede tener muchos y diversos aspectos y omitiendo información muy rica como puede ser la cara del locutor o su tono de voz. Es por eso por lo que se piensa que es interesante analizar estas emociones desde diversos puntos de vista y combinarlos, en este caso en concreto mediante el lenguaje y las conversaciones que podamos tener. Con los datos obtenidos de *SemEval* clasificados en parejas de causa-emoción es con lo que se va a intentar analizar y estudiar como de viable es que una maquina sea capaz de entender que efecto pueden tener estas palabras y como se dicen en los interlocutores, en este caso personajes de la sitcom *Friends.*

Palabras Clave

Procesamiento del Lenguaje natural, Procesamiento de audio con IA, Reconocimiento de emociones, Redes neuronales, Multimodalidad

Abstract y palabras claves (inglés)

The recognition of emotions is and has been a key point in our society to understand each other throughout history. Since we are social animals, we need to know how we make others feel in order to survive. That is why today, with the technological revolution we are experiencing with artificial intelligence, analysing these emotions can be much easier for us and for others. The analysis of emotions is an extremely interesting and valuable topic for us, but it is usually approached in a simple way and with a single focus, as there is an attempt to analyse feelings in isolated texts without considering that they are part of a whole, of a conversation that can have many diverse aspects, omitting rich information such as the speaker's facial expression or tone of voice. This is why it is thought to be interesting to analyse these emotions from various points of view and combine them, in this specific case through language and the conversations we might have. Using the data obtained from SemEval, classified in pairs of cause-emotion, we will try to analyse and study how viable it is for a machine to be capable of understanding the effect these words and the way they are said can have on interlocutors, in this case, characters from the sitcom Friends.

Keywords

Natural language processing, Audio processing with AI, Emotion Recognition, Neural Network, Multimodality

Índice de contenidos

Resumen y palabras claves (castellano) 2
Palabras Clave 2
Abstract y palabras claves (inglés) 3
Keywords 3
Introducción 7
Motivación. 7
Justificación 7
Objetivos 8
Marco Teórico 9
Estado del arte 9
PLN o NLP 9
IA para el análisis de audio 10
CNN (redes neuronales convolucionales) 10
Partes de una CNN 11
Transformers 12
Scaled dot-product attention (Atención por producto punto escalado) 13
Multi-head attention (atención multi-cabeza) 14
Codificador (encoder) y decodificador (decoder): 16
Combinación de encoder y decoder en transformers modernos 17
Entrenamiento y Fine-Tuning 18
Google Colaboratory 19
Características de Google Colaboratory 19
Limitaciones y Consideraciones 20
BERT 20
Transformador Bidireccional: 20
Componentes y Tamaños del Modelo: 21
Representación de Entrada/Salida 21
Preentrenamiento: 22
Fine-tuning: 22
RoBERTa 23
Mejoras en RoBERTa: 23

Comparación de rendimiento en tareas de clasificación de texto: 24
DistilBERT 25
Comparación de BERT y DistilBERT 25
Wav2vec 2.0 27
Codificador de características (feature encoder) 27
Red de contexto (context network): 28
Módulo de Cuantización: 28
Proceso de entrenamiento 28
Resultados Experimentales 28
Ensembles 29
Multimodalidad en IA 30
Merge 30
Librosa 31
Accuracy o precisión 31
Metodología 32
Objetivos e hipótesis 32
Datos originales 33
Transformaciones iniciales 35
Entrenamiento general 38
Entrenamiento con texto 39
Entrenamiento con audio 41
Ensemble final 42
Resultados 44
Accuracy 44
Resultados texto entero 45
Resultados texto parcial 46
Resultados Audio 47
Resultados ensemble 48
Discusión y Conclusiones 49
Conclusiones del texto 49
Conclusiones del audio 50
Multimodalidad 51

Conclusiones generales. 52
Limitaciones y futuras líneas de investigación 53
Limitaciones 53
Futuras líneas de Investigación 54
Referencias bibliográficas 56
Índice de figuras 58
Índice de tablas 59

Introducción

Motivación.

En la sociedad que vivimos en este 2024, está en boca de todos la Inteligencia Artificial (IA, en adelante), ese término que se usa muchas veces a la ligera, con el que se agrupan términos que muchas veces no tienen que ver, otras tantas la definición de IA se quedaría corta y muchas donde las que llamar a eso IA es pasarse.

Todo esto se ha ido viendo en los años pasados, pero se ha visto reforzado y expuesto al mundo con la llegada del famoso y para muchos milagroso ChatGPT-3. Este último es especialmente interesante debido a que es un modelo que entraría dentro del procesamiento de lenguaje natural (PNL o NLP, a partir de ahora por sus siglas en Ingles *Natural language processing*) un campo dentro de la IA se dedica a analizar estudiar y comprender las interacciones que podemos tener nosotros como humanos con los ordenadores como comentan (Gelbukh & Sidorov, 2010).

Estos modelos son extremadamente importantes en la actualidad debido a él gran impacto que pueden tener en facilitar la vida de las personas y en las múltiples aplicaciones de estos, desde el Alexa que tenemos en casa, pasando por las redes sociales e incluso llegando a ayudar a muchas personas facilitando la accesibilidad a las ingentes cantidades de contenido que se producen hoy en día.

Cabe destacar que ChatGPT-3 dentro de estos modelos entraría en una categoría conocida como los Modelos de lenguaje grande (LLM, a partir de ahora por sus siglas en Ingles *Large language model*) que serían aquellos modelos que están formados por grandes redes neuronales capaces de ser entrenados de manera auto supervisada o semisupervisada y que han proliferado gracias a los avances en la potencia computacional tanto en el procesamiento como en la obtención de datos, como explica (Goled, Shraddha, 2021) y como veremos más adelante.

Pero es importante destacar que nuestros modelos de lenguaje aun basándose en arquitecturas similares, por la propia naturaleza de cómo trabaja, no tiene que ver nada con los GPTs.

Justificación

El trabajo de la clasificación y predicción de las emociones mediante el lenguaje y todas sus representaciones gracias al análisis multimodal está justificado de una manera directa por parte de la

comunidad científica, ya que la idea de este mismo trabajo sale de un workshop propuesto por esta, como se muestra en el documento, además de ser muy interesante por sus futuras aplicaciones, dentro de diversos campos, como pueden ser el de la salud mental, la literatura, etc. Pues aun no siendo tan complejo este análisis de emociones como puede ser el que haga una persona real, es un paso hacia adelante en que las maquinas puedan facilitar y ayudar con tareas complejas y delicadas, en este caso tanto a nivel mental como a nivel emocional.

Es por ello por lo que creo que sería lo más conveniente e interesante que esta fuera la temática de mi trabajo, ya que es un problema real por el cual se ha visto que existe interés, ya que ha sido propuesto por una institución reconocida para una feria de evaluación semántica que se realiza todos los años como SemEval.

Además de que se puede ver un interés por parte de la sociedad en estos últimos años en cuanto al estudio de la psicología de los individuos y la mejora de esta para mejorar las condiciones de salud de estos, por lo que este trabajo aun no beneficiando de una manera directa a esto, puede ser un buen punto de partida para el desarrollo de herramientas que ayuden a esto mismo.

Objetivos

Este trabajo tratara de entrenar un modelo que sea capaz de interpretar y clasificar las emociones que se generan en una conversación, en concreto, se intentara predecir que efecto emocional puede tener una frase en una persona, teniendo en cuenta no solo lo que se dice a nivel de palabras (texto) sino también la entonación y en general el audio cuando la frase es dicha, pues creemos que factores como la entonación pueden ser clave para lograr entender lo que algo puede significar.

A parte de ese gran objetivo, con este trabajo planeo lograr descubrir si es posible la realización de esta hazaña, ya que parto de la hipótesis de que es posible y va a dar unos resultados valiosos, pero esto puede ser que no ocurra, es por eso por lo que una parte muy importante del trabajo también es el análisis de los resultados y el aprendizaje, el cual haremos comprobando la *acuracy* o precisión de nuestro modelo score de nuestro modelo.

Marco Teórico

Estado del arte

El mundo de las inteligencias artificiales, como se expuso, es un mundo muy amplio y rico, el cual solo su descripción hasta la actualidad, podría ser un trabajo entero de investigación, es por eso por lo que dentro del estado del arte en correspondencia a el trabajo que se expone, vamos solo a tocar puntos que creemos de gran relevancia para la comprensión de este.

Algunos de los términos más relevantes que se pretende explicar y analizar para el lector son el NLP (Procesamiento del Lenguaje Natural), el análisis de audio con IA y el *speech*, así como diversas y relevantes arquitecturas de distintos Transformers, como son BERT y variaciones, así como wav2vec 2.0

La investigación de estos modelos que existen ya entrenados por empresas con millones de datos y el cómo podemos tomar estas arquitecturas y adaptarlas de la mejor manera a un problema en concreto ha sido fundamental para el desarrollo del proyecto, además de que son de gran aportación para el lector por su novedad.

Además de investigar acerca de los modelos investigamos acerca del acercamiento que se le podría dar a este problema y el cómo afrontarlo, es ahí donde hay que entrar a definir tanto el NLP como el análisis de audio con IA.

PLN o NLP

El procesamiento del lenguaje natural (PLN o NLP) es una disciplina de la lingüística y el aprendizaje automático que se enfoca en entender el lenguaje humano. Las tareas comunes de PLN incluyen la clasificación de oraciones, la clasificación de palabras, la generación de texto, la extracción de respuestas y la traducción de textos. Estas tareas son desafiantes porque las computadoras no procesan la información como los humanos, requiriendo métodos complejos para representar y entender el lenguaje. (Gelbukh & Sidorov, 2010)

Maneras y herramientas para abordar estas tareas existen muchas y variadas, además de que existen muchos problemas dentro del NLP, como pueden ser la generación de texto, la síntesis de textos, la etiquetación de textos, la clasificación de textos etc.

IA para el análisis de audio

Las tareas de análisis de audio usando las herramientas que nos proporciona la IA se basan en varios principios, pues dependiendo del problema este puede implicar generar el audio, reconocer el texto que saldría de la transcripción de este, traducir, encontrar patrones en las frecuencias, clasificar, etc. pero en todas ellas implica extraer características y patrones relevantes de las señales de audio. (Hershey et al., 2017)

La IA y, en particular, el aprendizaje profundo (*deep learning*), han revolucionado el análisis de audio. Antes del *deep learning*, se usaban técnicas basadas en características diseñadas manualmente, como MFCCs (*Mel-frequency cepstral coefficients*). Ahora, las redes neuronales profundas como las CNN pueden aprender estas características automáticamente a partir de los datos. (Hershey et al., 2017)

CNN (redes neuronales convolucionales)

Las redes neuronales convolucionales (CNN, por sus siglas en inglés de ahora en adelante) son un tipo de redes neuronales que fueron diseñadas para destacar en el procesamiento de datos con una estructura de rejilla, como pueden ser las imágenes u otras representaciones no numéricas de datos como el audio.

Las CNNs como se expone en (Hershey et al., 2017) han logrado ser extremadamente efectivas en tareas de visión por ordenador, como el reconocimiento de imágenes o la detección de objetos, así como en todo tipo de tareas que involucren representar la información en matrices de elementos, como el análisis de factores o la clasificación del audio.

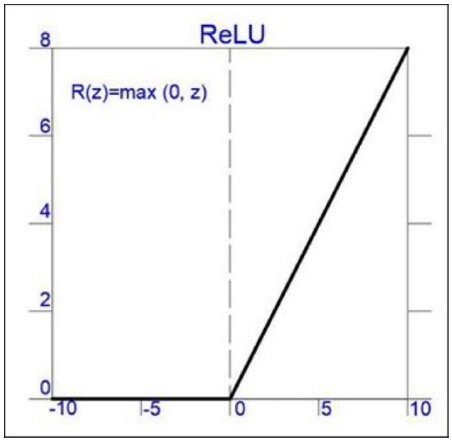
Las CNNs se inspiran en la organización del cerebro animal, particularmente en la corteza visual. En lugar de utilizar una red completamente conectada, como en una red neuronal artificial tradicional, las CNNs emplean capas convolucionales que permiten que la red aprenda características locales y espaciales de los datos. (Hershey et al., 2017)

Partes de una CNN

Según el artículo (Hershey et al., 2017) podemos diferenciar las siguientes capas dentro de una CNN:

Capa de entrada (*Input layer*)**:** La capa de entrada recibe los datos en forma de una matriz de números. Para una imagen en escala de grises, esta matriz sería bidimensional, mientras que para un audio recibiría una matriz numérica con la onda del sonido.

- Capas convolucionales (c*onvolutional layers*): Estas capas aplican filtros (también conocidos como *kernels*) a la entrada. Un filtro es una pequeña matriz que se desliza sobre los datos y realiza una operación de convolución, produciendo un mapa de características (*feature map*). Los filtros permiten detectar características locales como bordes y patrones. La convolución es esencialmente una multiplicación punto a punto entre el filtro y una porción de los datos, seguida de una suma de los productos. El filtro se desliza (con un cierto *stride* o paso) sobre los datos para producir el mapa de características.
- Función de activación (*activation function*): Después de cada operación de convolución, se aplica una función de activación no lineal a los datos, en este caso se aplica la conocida como ReLU (*rectified linear unit*). La función ReLU introduce no linealidad en la red, ya que cambia los valores negativos a cero.

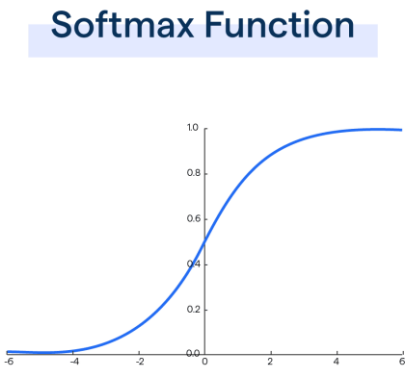


1 Función de activación ReLU

- Capas de p*ooling* (p*ooling layers*): Las capas de *pooling* reducen la dimensión de los mapas de características obtenidos por la red, lo que disminuye el número de parámetros y el costo computacional de la misma. El *pooling* más común es el *max pooling*, que toma el valor máximo

en una dentro de una ventana de los datos. Esto ayuda a que la red sea invariante a pequeñas traducciones en la entrada.

- Capas completamente conectadas (*fully connected layers*): Al final de todas esas capas convolucionales y del *pooling*, las características que se extraen se aplanan en un vector y se pasan a través de las capas completamente conectadas. Cada neurona en estas capas está conectada a todas las neuronas de la capa anterior, lo que hace que las capas completamente conectadas actúen como un clasificador final. Estas suelen usar la función de activación *softmax* para estas tareas de clasificación multiclase.



2 Función de activación SoftMax

- Capa de salida (*output layer*): La capa de salida produce la predicción final de la red. Para una tarea de clasificación, la capa de salida puede tener tantas neuronas como clases posibles, y cada neurona dará una probabilidad de pertenencia a una clase específica.

Transformers

Un *Transformer* es un modelo de aprendizaje profundo que usa el mecanismo de autoatención de los modelos, con el que en función de lo que va aprendiendo va asignando un peso diferente a cada parte de las entradas de datos que recibe. (Vaswani et al., 2023)

Este tipo de estructura fue por primera vez expuesta al mundo como un nuevo acercamiento para la traducción usando redes neuronales, pues hasta la fecha a la cabeza de las tareas de transducción (modelado del lenguaje, traducción maquina…) estaban las RNN (Recurrent Neural Networks o Redes

Neuronales Recurrentes) y las LSTM (*Long Short-Term Memory* o Memoria a Largo Corto Plazo) combinadas con mecanismos de atención para ser capaces de conectar los codificadores y los decodificadores.

Es en este momento cuando cambian todo los Transformers, modelos que están basados únicamente en el concepto de la atención, en el que se eliminan por completo de la ecuación los factores de la recurrencia y de las convoluciones, para quedarnos únicamente con un modelo basado en la autoatención semisupervisada para crear relaciones y dependencias entre los datos de entrada y los resultados.

El mecanismo de atención es una parte fundamental del modelo, este le permite enfocarse en diferentes partes de la secuencia de entrada para capturar dependencias contextuales, lo que hace que entienda el texto de una manera más global.

Esto está basado en diversas funciones matemáticas que asignan diferentes pesos las cuales vienen explicadas en el siguiente articulo (Vaswani et al., 2023) y son:

Scaled dot-product attention (Atención por producto punto escalado)

El Scaled dot-product attention es un mecanismo de atención que hace que los elementos de una secuencia cualquiera de entrada se enfoquen en otros elementos de la misma secuencia (lo que se conoce como autoatención) o de otra de las secuencias (lo que se conoce como atención cruzada, como en el caso de la relación codificador-decodificador).

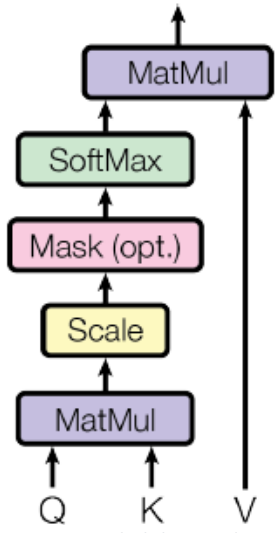
Esta recibe como inputs Q (*queries* o secuencias), K (*keys* o claves) y V (*values* o valores) que son matrices derivadas de la entrada de datos y de d_k (la dimensión de *keys*). En la función se calcula el producto punto entre las *queries* Q y los *keys* K, resultando en una matriz de puntajes que indica la similitud entre las *queries* y los *keys*.

El resultado es dividido por $\sqrt{d_k}$ para lograr escalar los productos, evitando así que los valores resultantes del *softmax* que se realiza sobre toda la expresión para obtener una distribución de probabilidades, sean demasiado pequeños, logrando así estabilizar el entrenamiento. Esta distribución pondera la importancia de cada valor del input. Finalmente, la distribución de probabilidades resultante se multiplica por los valores V para obtener las representaciones finales.

Resultando en la siguiente expresión:

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

3 Función Scaled dot-product



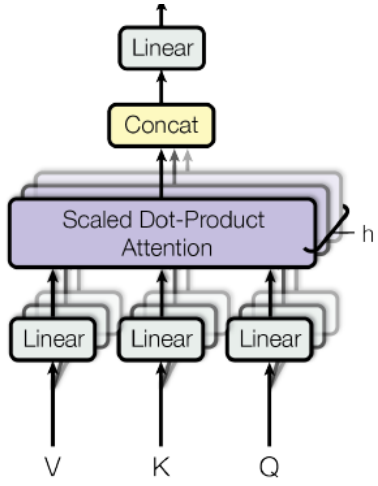
4 Esquema Scaled dot-product

Multi-head attention (atención multi-cabeza)

Este extiende el concepto de atención por producto punto escalado a ocarse en diferentes partes de la secuencia de entrada simultáneamente con múltiples "cabezas" de atención. Esto lo consigue haciendo varias atenciones en paralelo, cada una con diferentes parámetros, y luego combinándolas. Esto se realiza de la siguiente manera:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

6 Función multi-head



5 Esquema multi-head

Donde las proyecciones son matrices de parámetros Q$W_i^Q \in R^{d_{\text{model}} \times dk}$, KI$W_i^K \in R^{d_{\text{model}} \times dk}$, V$W_i^V \in R^{d_{\text{model}} \times dk}$ y $W_i^O \in R^{\text{hd}_v \text{ x } d_{\text{model}}}$. En el ejemplo emplean h=8, lo que significa que tiene 8 capas de atención paralelas, o cabezas. Para cada una de estas se usa d_k=d_v=d_{model}/h=64.

Debido a estas dimensiones de cada cabeza, el costo computacional total es similar al de la atención de una sola cabeza con dimensionalidad completa, lo que ahorra computacionalmente hablando.

Este modelo, en el caso del análisis de texto, al solo utilizar estas capas basadas en la atención logra ciertas ventajas:

- La primera es que, en los RNN o recurrentes, la distancia entre dos "tokens" o valores en la secuencia está relacionada con la longitud de esta, lo que puede dificultar el aprendizaje de relaciones a largo plazo debido al problema de desaparición del gradiente. En modelos convolucionales o CNN, aunque la distancia es menor, sigue dependiendo del tamaño del *kernel* y la profundidad de la red. En los Transformers, a diferencia de en los expuestos anteriormente la distancia entre dos tokens va a ser siempre de 1, esto se debe a que cada uno de los tokens es capaz de atender directamente a otro de ellos en una sola capa, lo que permite que el Transformer vea relaciones a largo plazo de manera más eficiente.
- Las redes recurrentes o RNN, además, procesan los datos de una manera secuencial, lo que limita su capacidad de paralelización aumentando así el tiempo de entrenamiento. Los modelos basados en convoluciones o CNN, por otro lado, también sufren limitaciones en la paralelización debido a que dependen de la posición de estos tokens. Mientras que los Transformers permiten la paralelización completa de los cálculos en cada capa, mejorando significativamente la eficiencia del entrenamiento.
- Por otra parte, las RNN tienen dificultades para manejar secuencias muy largas debido a la propagación del gradiente a través de muchas etapas. Las CNN, sin embargo, requieren un número fijo de operaciones de convolución para capturar dependencias a larga distancia. Mientras que los Transformers pueden manejar secuencias de cualquier longitud sin necesidad de cambiar la arquitectura o la cantidad de cálculos, ya que cada token puede directamente atender a cualquier otro token independientemente de la distancia en la secuencia.
- Gracias a la atención, los Transformers son capaces de capturar relaciones complejas y contextuales entre los tokens de una manera más efectiva que las convoluciones o las

recurrencias. Esto se debe a que cada token puede interactuar directamente con todos los demás tokens en cada capa, lo que permite una mejor representación del contexto global de la secuencia.

Todo esto demostró a la comunidad que esta arquitectura era superior a las basadas en convoluciones y recurrencia en las tareas de transducción, y más adelante con el desarrollo de diferentes modelos por grandes empresas basados en estas arquitecturas como lo son los BERT y los GPT que se estableció como la mejor opción para muchas tareas relacionadas con la inteligencia artificial, haciendo hincapié en las tareas de procesamiento de lenguaje (PLN).

Por otra parte, en el paper (Vaswani et al., 2023) también se destaca que los Transformers se basan en dos componentes principales, el codificador o *encoder* y el decodificador *decoder*:

Codificador (encoder) y decodificador (decoder):

El *encoder* se utiliza para procesar y representar secuencias de entrada. Consiste en una pila de capas de codificación, mientras que el *decoder* se encarga de generar la secuencia de salida en los Transformers y también consiste en una pila de capas.

Ambas estructuras comparten las capas de *Feed-Forward*, que es una red neuronal completamente conectada aplicada de forma independiente a cada posición de la secuencia, y la Normalización por Capas (*layer normalization*) que se aplica después de cada capa para estabilizar y acelerar el entrenamiento.

Mientras que el *encoder* cuenta con:

- La capa de atención autorregresiva (*self-attention*)**,** que permite al modelo enfocarse en diferentes aspectos de la secuencia de entrada para capturar dependencias contextuales. Cada palabra en la entrada puede "atender" a todas las otras palabras, un aspecto clave que le permite aprender qué palabras son relevantes para cada una en relación con todo el resto.
- El mecanismo de posición (*positional encoding*), necesaria porque los Transformers no tienen una estructura recurrente, por lo que se agrega información de posición para que el modelo pueda distinguir la posición de cada palabra en la secuencia.

Por su parte el *decoder* se diferencia del anteriormente mencionado *encoder* en:

- La capa de atención autorregresiva enmascarada (m*asked self-attention*), la cual es parecida a la de atención autorregresiva del *encoder*, pero en este caso se utiliza una máscara para asegurarse de que las posiciones futuras no sean atendidas, lo que mantiene así la auto regresión necesaria para la generación de secuencias.

La capa de atención *encoder-decoder*, que hace que cada posición en la secuencia de salida preste atención a todas las posiciones en la secuencia de entrada. Esto facilita la generación de la salida basada en la entrada, lo que hace que la salida siga un orden lógico en función de la entrada, haciendo que se centre así más en lo que tiene detrás en general y ya no tanto en particular en el token anterior.

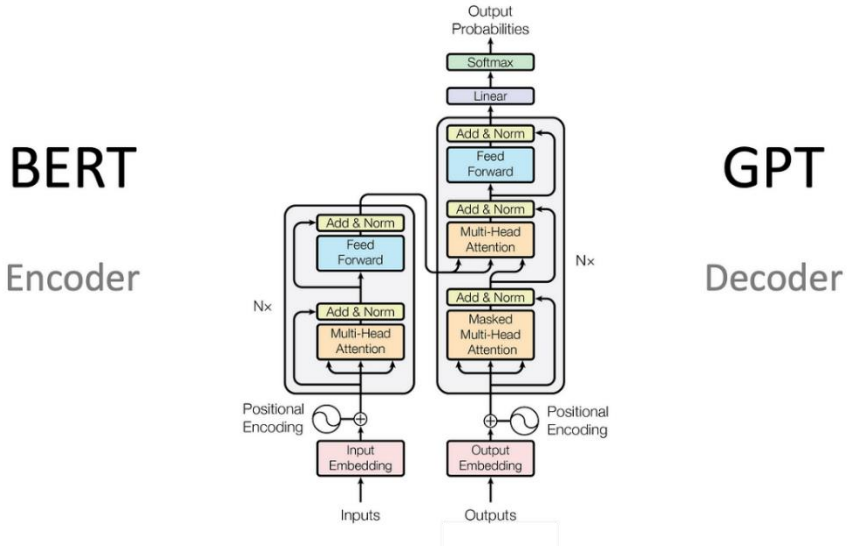
Combinación de encoder y decoder en transformers modernos

Aunque las estructuras de *encoder* y *decoder* se pueden usar de manera independiente como se ve en (Vaswani et al., 2023), siendo los modelos basados en *encoders* bastante eficaces para las tareas como la clasificación o el reconocimiento de entidades, por su propia naturaleza, pues estos ayudan en estas tareas permitiendo al modelo comprender el input de una manera más global y teniendo más en cuenta el contexto para la asignación, mientras que por otro lado los modelos basados en *decoders* son buenos para la generación de texto, pues en su naturaleza está que en cada entrada no se tiene en cuenta los valores siguientes o "futuros" por lo que intenta cada vez adivinar lo que se debería decir. Al final a día de hoy en la actualidad, como se expone en (Aitken et al., 2021) la mayoría de los modelos acaban juntando ambas estructuras y capas para mejorar los datos y sobre todo los resultados.

Es por eso por lo que los Transformers modernos combinan estos *encoders* y *decoders* de manera efectiva para tareas complejas de procesamiento del lenguaje natural y de análisis imágenes y audio.

Los datos de entrada primero son procesados a través de las múltiples capas de *encoders* previamente explicadas. Cada una de estas capas logra producir una representación más rica y abstracta de los datos que la anterior. Tras esto, el *decoder* recibe las representaciones enriquecidas del *encoder* junto con los datos originales y las utiliza en sus capas de atención para así generar contextos relevantes para cada posición en la secuencia de salida.

Utilizando la atención máscara autorregresiva, el *decoder* genera la secuencia de salida una posición a la vez, asegurándose de no utilizar información futura incorrectamente.



7 Esquema de un transformer

Entrenamiento y Fine-Tuning

Las redes neuronales actuales, ya sean CNN o Transformers se entrenan inicialmente en grandes corpus de datos de manera auto supervisada y luego se ajustan finamente para tareas específicas utilizando aprendizaje supervisado.

Estos modelos se entrenan en grandes cantidades de datos de esta manera auto supervisada para permitir que las maquinas sean capaces de comprenderlos como si de números se tratasen. Es por eso por lo que es calve entender que crear desde 0 estos modelos y sobre todo entrenarlos desde 0 es un proceso muy costoso, por lo que normalmente lo que se usa a un nivel más general y lo que se usó para este trabajo fueron modelos preentrenados por grandes corporaciones que se ajustan para las tareas específicas utilizando aprendizaje supervisado.

Los modelos más grandes y entrenados con más datos suelen ser más efectivos. Sin embargo, entrenar estos modelos es costoso en términos de tiempo, recursos computacionales e impacto ambiental.

Compartir modelos preentrenados ayuda a reducir estos costos. Es gracias a las capas que los modelos tienen y el funcionamiento de estas y a plataformas como huggingface que grandes compañías con muchos recursos pueden crear complejas arquitecturas y sobre todo entrenarlas desde 0 para las diversas funciones y para que luego los diferentes usuarios o empresas puedan fine-tunearlos con sus propios datos y casos. Esto ayuda a reducir tiempos, cantidades de datos necesarias e impacto ambiental del proceso de tener un modelo para un caso específico. Esto es un punto importante que se da gracias a que tanto las CNN como los Transformers se basan en capas, dentro de las cuales se pueden añadir las

necesarias al problema tanto en la entrada o en la salida, como se expone en (Vaswani et al., 2023) y en (Hershey et al., 2017).

Google Colaboratory

Google *Colaboratory*, es una plataforma que proporciona un entorno similar a *Júpiter Notebook*, pero con la ventaja adicional de ofrecer acceso a potentes recursos de hardware como GPUs y TPUs, lo que facilita el desarrollo y la experimentación en aprendizaje profundo, y que además permite a los usuarios escribir y ejecutar código Python directamente en su navegador. Es una herramienta poderosa para la investigación y desarrollo en campos como el aprendizaje automático y la inteligencia artificial debido a su accesibilidad y características integradas. Algunas de sus características que le aportan gran relevancia en la implementación de proyectos de aprendizaje automático son:

Características de Google Colaboratory

Google Colaborayory está basado en la nube, lo que significa que se puede acceder a los notebooks desde cualquier lugar con una conexión a Internet, además de permitir que varios usuarios pueden colaborar en el mismo notebook en tiempo real, similar a cómo se puede colaborar en documentos de Google Docs.

- GPU y TPU: Google Colaboratory proporciona acceso gratuito a GPUs (Unidades de Procesamiento Gráfico) y TPUs (Unidades de Procesamiento Tensor), que son esenciales para entrenar modelos de aprendizaje profundo de manera eficiente. También permite que los usuarios pueden cargar y descargar archivos desde Google Drive, lo que facilita el manejo de grandes conjuntos de datos. Esto aporta gran valor al trabajo pues una de las limitaciones que se comentara es la potencia de los equipos con los que se contaba. La posibilidad de acceder a clústeres de Google de GPU y TPU permite a proyectos como este poder tener una potencia computacional mayor sin coste alguno.
- Compatibilidad con Júpiter: Google Colaboratory es compatible con Júpiter Notebooks, lo que significa que se pueden utilizar celdas de código y texto de la misma manera. Hay que destacar que Google Colaboratory cuenta con muchas librerías de aprendizaje automático y ciencia de datos preinstaladas, como TensorFlow, Keras, PyTorch, Pandas, y Numpy.

Limitaciones y Consideraciones

Límites de Recursos: Aunque Colaboratory proporciona acceso gratuito a GPUs y TPUs, existen límites en el uso continuo y la disponibilidad puede variar, es por esto por lo que tampoco se pueden entrenar grandísimos modelos ni automatizarlos en esta plataforma.

Privacidad y Seguridad: Al ser un servicio basado en la nube, es importante considerar la privacidad y la seguridad de los datos que se suben y procesan en Google Colaboratory. Así como la dependencia de la conectividad**,** pues al ser esta una plataforma en línea, el acceso a Colaboratory depende de una conexión a Internet estable.

La información acerca de la existencia y rendimiento de Google Colaboratory, así como justificación de su potencia fue sacada del articulo posteriormente citado (Carneiro et al., 2018).

BERT

Según (Devlin et al., 2019), BERT (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers) es una arquitectura que destaca por implementar los Transformers bidireccionales, mejorando así su capacidad para la resolución de ciertas tareas de PLN, en concreto aquellas que requieran de una comprensión más general de los datos y su contexto. Los puntos principales de la arquitectura de BERT son:

Transformador Bidireccional:

Según el artículo (Devlin et al., 2019) BERT se basa en una arquitectura bidireccional, lo cual significa que a la hora tanto de tokenizar como de asignar pesos y valores a las palabras considera el contexto tanto de la izquierda como de la derecha del token o palabra en cada una de las capas del modelo.

Es por esto por lo que podríamos decir que BERT se centra en la utilización de un "*encoder*" bidireccional para obtener representaciones profundas del lenguaje, empleando tareas de preentrenamiento como el "*Masked LM*" y "*Next Sentence Prediction*" para mejorar su capacidad de comprensión contextual.

Esto contrasta con otros modelos unidireccionales o de generación de texto, como OpenAI GPT, que utilizan un "*decoder*" de atención autorregresiva unidireccional, limitando la atención de cada token solo al contexto a su izquierda o al "pasado".

Componentes y Tamaños del Modelo:

Como se explica en el artículo (Devlin et al., 2019) el modelo de BERT se puede configurar en diferentes tamaños, siendo los más comunes BERTBASE y BERTLARGE.

- BERTBASE: 12 capas, tamaño de la representación oculta (hidden size) de 768, y 12 cabezas de atención, con un total de 110 millones de parámetros.
- BERTLARGE: 24 capas, tamaño de la representación oculta de 1024, y 16 cabezas de atención, con un total de 340 millones de parámetros.

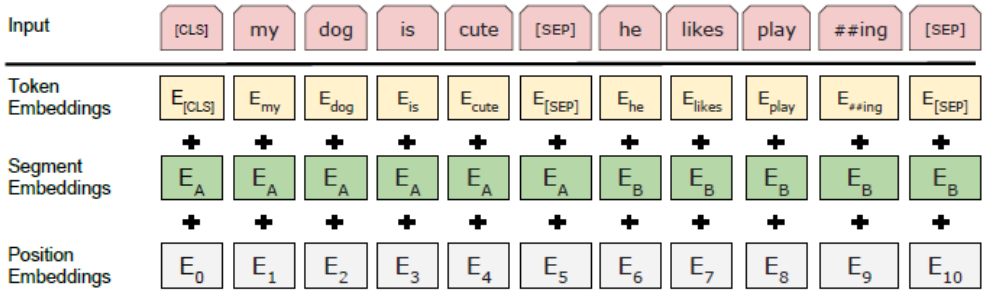
En nuestro caso cabe destacar que en base a nuestro volumen de datos y complejidad del problema no es necesario usar una estructura BERTLARGE, ya que con reinterpretaciones de BERT (como RoBERTa o DistilBERT) que tiene versiones BASE y LARGE, pero que con la BASE nos basta.

Representación de Entrada/Salida

BERT utiliza *WordPiece embeddings*, que cuenta con un vocabulario de 30,000 tokens. Cada secuencia de entrada comienza con un token especial de clasificación [CLS] y las oraciones dentro de una secuencia están separadas por un token [SEP].

Los *WordPiece embeddings* son una técnica de tokenización, lo que significa que las palabras se dividen en unidades más pequeñas llamadas "tokens" que pueden ser partes de palabras (subpalabras) o palabras completas, que se utiliza para manejar eficientemente un vocabulario amplio y reducir el problema de palabras fuera del vocabulario (*out-of-vocabulary*, OOV).

La representación de entrada para cada token se construye sumando las *embeddings* correspondientes al token, al segmento y a la posición.



8 Ejemplo esquemático tokenización

Preentrenamiento:

BERT se pre-entrena utilizando dos tareas no supervisadas: *Masked language model* (MLM) y *next sentence prediction* (NSP).

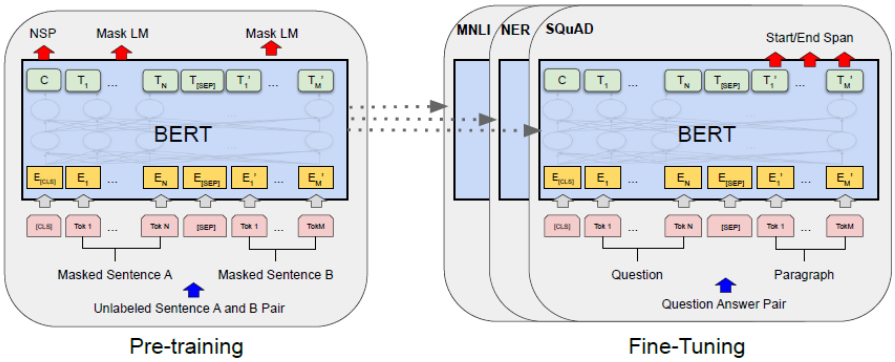
- MLM: Algunos tokens en la secuencia de entrada se enmascaran aleatoriamente y el modelo debe predecir el token original basado en su contexto. Esto permite que el modelo aprenda representaciones profundas bidireccionales, lo que no es posible con los modelos de lenguaje unidireccionales tradicionales.
- NSP: Esta tarea involucra predecir si una oración dada sigue a otra en el corpus, lo que ayuda a entrenar representaciones que capturan relaciones entre oraciones.

Fine-tuning:

Después del preentrenamiento, en general, BERT se ajusta para tareas específicas agregando una capa de salida y entrenando el modelo completo con datos etiquetados de la tarea objetivo. Este proceso permite que BERT alcance resultados de última generación en una amplia gama de tareas de NLP sin necesidad de modificaciones sustanciales en la arquitectura del modelo. Es lo que se conoce como *fine-tune o fine-tuning*.

Es por todo esto que BERT ha demostrado ser muy eficaz para mejorar el rendimiento en múltiples tareas de NLP, estableciendo nuevos estándares en once tareas diferentes, incluyendo el reconocimiento

de entidades nombradas la respuesta a preguntas y más importante en nuestro caso, la clasificación de sentencias.



9 Estructura por capas de BERT

RoBERTa

RoBERTa (*Robustly Optimized BERT Approach*) es una mejora de BERT que realiza varios cambios en el proceso de preentrenamiento para aumentar su rendimiento en tareas *downstream*, que son aquellas tareas específicas que se realizan después de que el modelo ha sido pre-entrenado. (Liu et al., 2019)

Estas tareas utilizan las representaciones aprendidas durante el preentrenamiento para realizar tareas más especializadas y aplicadas.

Mejoras en RoBERTa:

Las mejoras que teóricamente RoBERTa implementa vienen por la eliminación de la tarea de predicción de la próxima oración, ya que RoBERTa al omitir esta tarea simplifica el modelo y permite un uso más eficiente de los datos.

- El entrenamiento más prolongado y con mayores lotes de datos, lo que hace que RoBERTa se entrena durante más tiempo, utilizando lotes más grandes y una mayor cantidad de datos, lo que mejora significativamente el rendimiento del modelo.

- Cuenta con más datos de entrenamiento, ya que aparte de entrenar más tiempo, RoBERTa utiliza un corpus de datos más extenso y variado (CC-NEWS), lo que le permite aprender representaciones lingüísticas más ricas y generales.
- Finalmente, cuenta con más pasos de entrenamiento y cambio dinámico de patrones de enmascaramiento, pues RoBERTa emplea una técnica de enmascaramiento dinámico en cada época de entrenamiento, lo que mejora la capacidad del modelo para generalizar.

Comparación de rendimiento en tareas de clasificación de texto:

En pruebas controladas y comparaciones directas expuestas en (Liu et al., 2019), RoBERTa supera de manera consistente a BERT en una distintas de tareas de clasificación de texto. Esto se debe a las mejoras comentadas anteriormente, como el entrenamiento más prolongado y el uso de más datos.

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7
$\text{XLNet}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6

10 Comparación RoBERTa y BERT

Por ejemplo, en la evaluación de la tabla anterior, RoBERTa muestra una mejora significativa en tareas como SQuAD, MNLI, y SST-2, indicando su superioridad en la comprensión y clasificación de texto.

SQuAD, MNLI, y SST-2 son benchmarks estándar en el campo del procesamiento de lenguaje natural (NLP) utilizados para evaluar el rendimiento de los modelos pre-entrenados

En resumen, las mejoras estructurales y de entrenamiento de RoBERTa le permiten superar a BERT en tareas de clasificación de textos, haciendo de RoBERTa una opción preferida para aplicaciones que requieren alta precisión en la comprensión y clasificación de contenido textual.

Toda esta información acerca del modelo RoBERTa fue obtenida del estudio del siguiente artículo en el cual se presentaba este modelo al mundo, aquí citado. (Liu et al., 2019)

DistilBERT.

DistilBERT se define en el artículo (Sanh et al., 2020) como una versión más ligera y eficiente de BERT, desarrollada mediante un proceso llamado destilación del conocimiento, donde un modelo más pequeño (DistilBERT) aprende a imitar el comportamiento de un modelo más grande y complejo (BERT).

La destilación de conocimiento es un método de compresión de modelos en el que un modelo grande y bien entrenado (conocido como el modelo maestro o *teacher model*) se utiliza para entrenar un modelo más pequeño y eficiente (conocido como el modelo estudiante o *student model*). El objetivo es que el modelo estudiante aprenda a imitar el comportamiento del modelo maestro.

Para entender mejor cómo se compara DistilBERT con BERT y por qué podría funcionar mejor o peor en tareas de clasificación de textos, repasemos algunos puntos clave de ambos modelos:

Comparación de BERT y DistilBERT

Tamaño y Eficiencia:

BERT tiene 110 millones de parámetros en su versión base. Es conocido por su alta precisión en una variedad de tareas NLP, pero es computacionalmente costoso. Mientras que DistilBERT Tiene aproximadamente 66 millones de parámetros, lo que es alrededor de un 40% menos que BERT. Esta reducción de tamaño se traduce en tiempos de inferencia más rápidos y menos requerimientos de memoria.

Rendimiento en Tareas Downstream:

BERT generalmente tiene un rendimiento superior en la mayoría de las tareas NLP. En las pruebas de benchmarking como GLUE, IMDb y SQuAD, BERT suele obtener puntajes más altos. Mientras que DistilBERT aun siendo más pequeño, retiene aproximadamente el 97% del rendimiento de BERT en tareas de GLUE, y solo está marginalmente detrás en precisión en tareas como IMDb y SQuAD. Específicamente, en IMDb, DistilBERT está solo 0.6 puntos porcentuales por detrás de BERT y en SQuAD

Model	IMDb (acc.)	SQuAD (EM/F1)
BERT-base	93.46	81.2/88.5
DistilBERT	92.82	77.7/85.8
DistilBERT (D)	-	79.1/86.9

13 Rendimiento DistilBERT 1

Model	# param. (Millions)	Inf. time (seconds)
ELMo	180	895
BERT-base	110	668
DistilBERT	66	410

12 Rendimiento DistilBERT 2

Model	Score	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
ELMo	68.7	44.1	68.6	76.6	71.1	86.2	53.4	91.5	70.4	56.3
BERT-base	79.5	56.3	86.7	88.6	91.8	89.6	69.3	92.7	89.0	53.5
DistilBERT	77.0	51.3	82.2	87.5	89.2	88.5	59.9	91.3	86.9	56.3

11 Rendimiento DistilBERT 3

dentro de 3.9 puntos de diferencia. Esto se ve reflejado en las tablas expuestas del articulo (Sanh et al., 2020)

Velocidad de Inferencia:

BERT, debido a su tamaño tiene una inferencia es más lenta y requiere más recursos computacionales. Por otro lado, DistilBERT ofrece una inferencia significativamente más rápida, siendo constantemente más eficiente en términos de tiempo de procesamiento.

Por lo que podríamos concluir que BERT, como modelo, ofrece mejores números a nivel de precisión pura, pero como hemos visto, DistilBERT es significativamente más eficiente y aún mantiene altos valores de precisión, lo que lo convierte en una excelente opción para afrontar tareas que requieran o tengan muchos datos o para entornos con limitaciones de recursos computacionales.

La elección entre BERT y DistilBERT debería basarse en un balance entre la necesidad de precisión máxima y las limitaciones de recursos y velocidad de inferencia, como se expone en los artículos (Devlin et al., 2019) y (Sanh et al., 2020).

Wav2vec 2.0

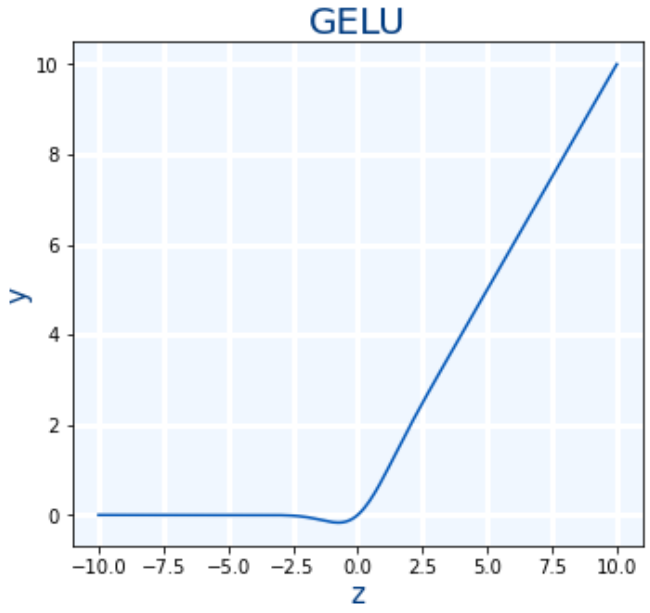
Wav2Vec2 es un modelo de Inteligencia Artificial que fue creado por Facebook para el aprendizaje auto supervisado del audio, las conversaciones y sus representaciones, expuesto en (Baevski et al., 2020)

Se basa en enmascarar partes del audio (en el espacio latente) y resolver una tarea contrastiva definida sobre una cuantización de las representaciones latentes, las cuales se aprenden conjuntamente.

El espacio latente es un conjunto de características ocultas en las cuales el audio original se transforma después de pasar por nuestra red neuronal convolucional (CNN). Las representaciones latentes son una versión comprimida y transformada en forma de matriz de los datos originales que conservan la información importante. Este método ha mostrado mejorar significativamente el reconocimiento del habla utilizando menos datos etiquetados.

Codificador de características (feature encoder)

Es una red neuronal convolucional que, como se explica en (Baevski et al., 2020), toma como entrada el audio crudo y produce representaciones del habla latentes. El audio de entrada es normalizado para tener media cero y varianza unitaria, lo que hace que sea más sencillo de analizar. Las representaciones latentes generadas se obtienen a través de varias capas convolucionales, seguidas por una normalización de capas y una función de activación GeLU.



14 Función de activación GeLU

Red de contexto (context network):

Las representaciones latentes del codificador se vuelcan en un Transformer, para así construir representaciones contextuales, como vimos anteriormente. Además, se utilizan convoluciones para incorporar información posicional relativa, en lugar de *embeddings* posicionales fijos como se hace en los *transformers* por atención, como vimos antes.

Módulo de Cuantización:

Las representaciones latentes se cuantizan usando un módulo de cuantización basado en softmax de Gumbel. La cuantización se aplica solo en la tarea contrastiva para las representaciones objetivo, no para las entradas al Transformer, lo cual ayuda a retener más información útil en las representaciones de entrada.

Proceso de entrenamiento.

El proceso de entrenamiento en este modelo, viene definido por las dos siguientes interacciones expuestas en (Baevski et al., 2020), el preentrenamiento auto supervisado, donde modelo enmascara ciertas partes de las representaciones del habla latentes y debe distinguir la representación correcta de entre varios distractores. Esto se realiza mediante una tarea contrastiva, lo que crea representaciones latentes que se cuantizan para crear unidades de habla discretas, lo que se encuentra más efectivo que las metas no cuantizadas. Y el ajuste fino (fine-tuning), donde después del preentrenamiento en datos de habla no etiquetados, el modelo se ajusta finamente con datos etiquetados usando una pérdida de Clasificación temporal conectista (CTC).

Resultados Experimentales

En el artículo (Baevski et al., 2020) se expone como en las pruebas con el conjunto de datos *Librispeech*, wav2vec 2.0 logra tasas de error de palabras (WER) de 1.8/3.3 en los conjuntos de prueba limpio/otro, utilizando todos los datos etiquetados.

Con solo diez minutos de datos etiquetados, aún logra una WER de 4.8/8.2, demostrando la eficacia del modelo con cantidades mínimas de datos etiquetados.

Por lo que, según lo expuesto anteriormente que esta reforzado por el artículo (Baevski et al., 2020) Wav2vec 2.0 representa un avance significativo en el aprendizaje auto supervisado para el procesamiento de audio, particularmente en el reconocimiento del habla. Su capacidad para funcionar con cantidades mínimas de datos etiquetados y su rendimiento superior en comparación con métodos anteriores lo hacen una opción prometedora para aplicaciones de reconocimiento del habla en diversos idiomas y contextos con recursos limitados.

Ensembles

Dentro del campo del aprendizaje automático y la inteligencia artificial existen los métodos de ensemble, estos se definen como la combinación de varios modelos individuales para obtener un modelo predictivo más robusto y preciso (Dietterich, 2000).

La premisa fundamental detrás de los métodos de ensembles es que, si se combinan múltiples modelos se pueden mitigar los errores y la varianza presentes en los modelos individuales, aprovechando las fortalezas y compensando las debilidades de cada uno.

Dentro de los ensembles se exponen en (Dietterich, 2000) varios tipos:

Bagging: Este implica entrenar múltiples versiones del mismo modelo en diferentes subconjuntos del conjunto de datos y luego sacar la media de sus predicciones (para tareas de regresión) o utilizar la votación mayoritaria (para tareas de clasificación).

Boosting: Este implica entrenar múltiples y distintos modelos secuenciales, donde cada modelo intenta corregir los errores de su predecesor. Los modelos se ponderan según su rendimiento, y las predicciones finales se combinan en consecuencia.

Stacking: Esta combina múltiples modelos (que pueden ser de diferentes tipos) en un modelo de nivel superior o metamodelo. Las predicciones de los modelos base se utilizan como características para entrenar el metamodelo.

Voting: Esta es una técnica sencilla donde se entrenan múltiples modelos y se combinan sus predicciones mediante votación para clasificación o promedio para regresión, dentro de la votación, cada modelo de los entrenados vota por la opción que devuelve y la mas popular es la elegida.

Los ensembles, como se expone en las conclusiones del articulo (Opitz & Maclin, 1999), son un acercamiento a la resolución de los problemas muy popular debido a que suelen tener una mayor precisión que los modelos individuales debido a la reducción de errores y varianza. Además de que, al combinar múltiples modelos, el ensemble es menos susceptible a errores individuales y ruidos en los datos, así como también tienden a generalizar mejor en conjuntos de datos no vistos, proporcionando predicciones más coherentes con el mundo real y no tan cuadriculadas.

Multimodalidad en IA

Según (Durante et al., 2024) la multimodalidad se refiere a la capacidad de un modelo de inteligencia artificial para procesar y comprender múltiples tipos de datos (o "modos") al mismo tiempo.

Esto se puede conseguir de muchas maneras, pero los enfoques más comunes suelen ser aquellos que desarrollan modelos o soluciones para cada uno de estos tipos de datos o modos y luego juntan los datos en estructuras conocidas como *ensembles*, que son redes neuronales que combinan varias redes distintas y sus resultados por muy distintos que sean, siempre que devuelvan y trabajen al final con datos en las mismas estructuras.

Merge

Un merge o join es la combinación de dos conjuntos de datos en función de una o más columnas comunes. Esto es útil en nuestro trabajo ya que así podemos unir inequívocamente nuestros dos datasets. Es importante saber que dentro de los merges hay 4 tipos:

Inner merge: Devuelve solo las filas que tienen llaves coincidentes en ambos dataframes.

Left merge: Devuelve todas las filas del dataframe de la izquierda y las filas coincidentes del dataframe de la derecha. Si no hay coincidencia, se coloca NaN (valor nulo) en las columnas del dataframe derecho. El dataframe de la izquierda es el primero de los que se le pasa a la función.

Right merge: Devuelve todas las filas del dataframe de la derecha y las filas coincidentes del dataframe de la izquierda. Si no hay coincidencia, se coloca NaN en las columnas del dataframe izquierdo. El dataframe de la izquierda es el segundo de los que se le pasa a la función.

Outer merge: Devuelve todas las filas cuando hay una coincidencia en cualquiera de los dataframes. Las áreas no coincidentes se llenan con NaN.

Librosa

Librosa es una popular biblioteca de Python para el análisis y procesamiento de señales de audio. Es especialmente útil en aplicaciones de música y audio, como la extracción de características de audio, la manipulación de tiempo y frecuencia, y la visualización de datos de audio. Este fue creado y desarrollado para una tarea del workshop SciPy en la edición de 2015.

Accuracy o precisión

La precisión, o accuracy en inglés, se define en (Borja-Robalino et al., s. f.) como una medida usada para evaluar el rendimiento de los clasificadores, tanto en el caso dicotómico (que solo existan dos clases que clasificar) como en el multiclase (que la clasificación sea entre varias clases). Se define como la proporción de clasificaciones correctas realizadas por el modelo en relación con el total de instancias evaluadas.

En otras palabras, la precisión es usada para medir cuántas predicciones el modelo fue capaz de hacer correctamente (tanto positivas como negativas) sobre el total de predicciones realizadas. Es una métrica sencilla y fácil de interpretar, lo que la convierte en una opción popular entre los investigadores.

Sin embargo, tiene limitaciones, especialmente en escenarios de clases desbalanceadas, donde puede no ser suficientemente discriminatoria para evaluar adecuadamente el rendimiento del modelo, así como limitaciones a la hora de concretar que tipos de errores fueron los más comunes en el modelo, lo cual dependiendo del problema puede ser un factor fundamental.

Metodología

Objetivos e hipótesis

Una vez abordado todos los conocimientos teóricos necesarios para la correcta ejecución de este trabajo describiremos como fue todo este proceso.

Este trabajo se realizará con la base de datos que la organización de SemEval proporciona a sus participantes, esto es así ya que este análisis de texto ha de ser supervisado y encontrar unos datos correctamente etiquetados es complicado y podría suponer un esfuerzo que hiciera que no se pudiera desarrollar el trabajo de manera correcta.

Esto además ayuda a que una vez salgan los resultados del workshop se puedan analizar los expuestos en este documento, así como el enfoque para con los modelos usados y el tratamiento que pudiéramos darles, comparándolo con el resto de los participantes.

Esta sería la representación gráfica de cómo se estructuran los datos a nivel conceptual que SemEval proporciona a sus participantes en esta tarea:

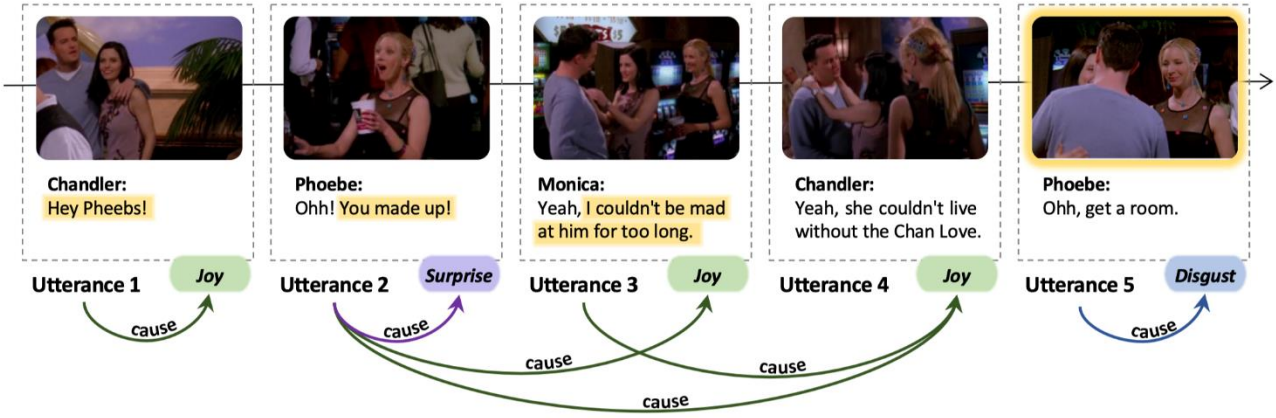


Figure 1: An example of our task and annotated dataset. Each arc points from the cause utterance to the emotion it triggers. The cause spans have been highlighted in yellow. Background: Chandler and his girlfriend Monica walked into the casino (they had a quarrel earlier but made up soon), and then started a conversation with Phoebe.

15 Gráfico datos y su estructura

Aquí se puede observar una escena, dentro de esta, existen varias líneas de dialogo dichas por diversos personajes, cada una en su momento, se tiene en la aparición o *Utterance:* El personaje que habla, la frase que dice, lo que siente en ese momento y porque viene dado ese sentimiento.

Con ello vemos como por ejemplo la frase de Phoebe en la aparición 2 causa su propia sorpresa, pero a la vez causa la felicidad de Mónica y Chandler en las apariciones 3 y 4 respectivamente.

Con esto lograríamos un "input" para nuestro trabajo, siendo cada una de estas frases lo que tendríamos como entrada, mientras que la reacción que genera como etiqueta de nuestros datos.

Esto como concepto suena sencillo, se tienen relacionados en los datos las frases con los audios y con las etiquetas, que en este caso son las emociones, pero para lograr tener estos datos y sobre todo para lograr tenerlos en el formato adecuado han tenido que pasar un proceso de preprocesado bastante complejo.

Datos originales

Originalmente en este trabajo, los datos que se han descrito venían en un .json que se puede encontrar en el repositorio de GitHub[1]. Este se organizaba de la siguiente forma:

Dentro de estos y en cada una de las conversaciones, hay dos diccionarios. Uno contiene la información tal cual de la escena (*conversation*) y otro la relación entre la emoción causada con lo que la causo (*emotion-cause_pairs*).

Dentro del primer diccionario (*conversation*) podemos encontrar las diferentes intervenciones de los personajes en la escena.

Aquí es importante conocer los conceptos de diccionario y lista, además de comprender la estructura y jerarquía en la que los datos están estructurados, ya que gracias a que los diccionarios son estructuras de datos que se basen en que los datos vienen con una clave y un valor, podemos identificar dentro de cada interacción que es cada cosa, mientras que para las parejas de emoción y causa no es necesario recurrir a esto.

[1] https://github.com/RodrigoRivasArevalo/TFG_Multimodal_final

Cada entrada es un valor en la lista que conforma la conversación, a su vez siendo cada uno de los elementos de esta lista la propia intervención.

Esto a su vez organizándose en un diccionario que contiene el id de la intervención, el texto que se dice en inglés, quien dice la frase, el nombre del archivo donde está el clip del personaje diciendo la frase y por ultimo y no menos importante la emoción que siente el personaje que dice la frase mientras dice esa frase.

Esto puede parecer lo contrario a lo que queremos, ya que con esta distribución de la información sabemos lo que siente la persona cuando dice lo que dice, pero no sabemos cuál es la emoción que genera en otra persona, pero en el fondo es como tienen que estar ya que así somos capaces de relacionar cada momento, frase y persona con una emoción.

Esto se hace porque se cuenta con que cada persona solo siente una emoción a la vez, pero la misma frase puede generar distintas emociones en distintas personas.

Por esto existe la segunda lista dentro de cada una de las conversaciones, la conocida como *emotion-cause_pairs*, en la cual hay una lista en la que cada uno de los elementos es otra lista con tan solo dos elementos y siempre dos elementos.

Estos son “n_emocion” y un número, siendo en el primer elemento la n en numero de la intervención de la que sale la emoción que va después de la _ y por otro lado siento el segundo numero de la lista el numero de la intervención de la conversación que genera la dicha emoción, relacionando así la emoción obtenida con su generador. Haciendo así posible la finalidad de este trabajo.

Cabe destacar que este por lo general las escenas suelen tener entre 6 y 15 intervenciones y parejas de emociones que suceden de las interacciones entre los diversos personajes.

Todo esto se puede encontrar de manera más detallada, como comenté, en el archivo original que *SemEval* nos proporcionaba y desde el que se ha partido, Subtask_2_train.json.

Una vez se ha comprendido como es la estructura de estos datos, se debían de pasar a un formato que fuera entendible para el fine-tuneo del modelo. Esto es algo fácil de decir, pero acabo siendo gran parte del trabajo.

Ya que estos datos son algo complicados de cargar para que quede en un *dataframe* ordenado, y además que mucha información de la que explicamos que contiene no es relevante ni nos interesa, ni a nosotros ni al propio modelo, es aquí donde empiezan las primeras transformaciones.

Transformaciones iniciales

Lo primero es cargar el json con pandas, pero ocurre que teniendo en cuenta la estructura de este, solo tenemos dos columnas en el *dataframe* si se carga sin más con pandas, una con todos los datos de la lista conversación y otra con los datos de *emotion_cause-pairs*.

Por lo que hay que empezar a hacer transformaciones para tener el *dataframe* depurado para poder trabajar.

Aunque para poder hacer esto antes se tienen que plantear cuáles son los campos que se quieren obtener, que cosas aportan valor dentro de los datos y que cosas no, tras un análisis se concluyó que los campos que eran necesarios para poder realizar este trabajo eran:

- La emoción objetivo a predecir, es decir, la emoción que genera, no confundir con la que siente el personaje en concreto (*emotion_x*)
- El numero de la conversación en la que nos encontramos (*conv*) junto con el id de la frase o intervención del personaje dentro de la conversación (*ID_frase*)
- El propio texto que se dice, es decir, la transcripción al inglés (*text*)
- Que personaje dice la frase (*speaker*)
- El archivo de audio (*video_name*)

Hay que destacar que este fue el primer filtrado de características del trabajo, el cual no será el último, ya que dependiendo de la tarea a realizar se harán otras modificaciones o eliminaciones sobre estas características.

Ya que a estos datos se les tendría que sumar la información necesaria para el análisis del audio, que en este caso es un array con los valores del espectro de sonido del audio representado a 16KHz, lo que significa que la frecuencia de muestreo del audio es de 16Khz.

La frecuencia de muestreo es el número de *samples* o muestras de audio tomadas por segundo durante la grabación o reproducción de sonido digital. En este caso, 16 kHz significa que se toman 16,000 muestras por segundo.

Estos son los campos finales a los que se debía de llegar, pero no es tan sencillo como que se ordenen conforme están en el archivo original, ya que no interesa la estructura en base a las conversaciones en general si no que se va a centrar en cada frase independientemente del contexto, ya que esto de por si supone la suficiente complejidad para el modelo.

Para poder tener cada frase con lo necesario, se crearon dos tablas, una que tuviera los registros de *conversation* y otra que tuviera los registros de la columna emotion_cause-pairs.

En la tabla de *conversation*, están cada una de las frases que tiene el *dataset*, siendo cada una distinta, a la vez que teníamos registrado el resto de los campos, destacando dos de estos, que son la conversación a la que pertenece cada frase y el ID de la frase dentro de la conversación, haciendo así un índice con el que podemos tener ordenada e identificada de manera única cada una de las frases que tenemos en nuestros datos.

Por otro lado, está el *dataset* de *emotion_cause-pairs* el cual cuenta con 3 columnas, estas serían la emoción (es el valor n_emocion que antes explicamos, en el que la n hace referencia a la intervención del personaje que la siente), la conversación en la que esta esa emoción y por último el ID de la frase que genera esa emoción en concreto.

Es una vez que existe esta pareja de *datasets*, que se puede obtener la relación necesaria entre las frases y las emociones que generan, además de la emoción que se siente cuando se dice la frase, que ya teníamos de antes.

Esto lo podemos lograr haciendo un *merge* entre ambos *datasets*, usando como claves las columnas que tenemos del id de la conversación y el id de la intervención, para así tener cada frase relacionada gracias a estos índices. Es importante hacer el *merge* por la tabla de *emotion_cause-pairs*.

En este caso el *merge* que realizamos es un *left merge* tomando como primera de las tablas la de *emotion_cause-pairs*, ya que se quieren tener los registros para todas las emociones que se generen, aunque los datos de *conversation* se repitan, ya que la misma frase puede generar dos emociones distintas en distintos personajes.

Por otro lado, las transformaciones no acaban aquí, pues como se comentaba, también es necesario insertar en los datos la información que se tiene referente a los clips de audio, los cuales vienen citados en las distintas intervenciones, siendo lo que viene la ruta relativa de estos, alojándose en una carpeta a parte y que no se ha logrado subir al cloud, ya que estos llegaban a sumar 6 GB de información.

Dentro de nuestros datos, lo que teníamos era el *path* relativo al clip de video (ya que al principio los datos que daban para Sem-Eval eran videos, pero simplemente transformamos esos archivos de video en .wav con un simple script de Python) pero no contenía más información del video/audio.

Para poder solventar esto, lo que hice fue codificar los datos en *strings* de números, siendo estos números la frecuencia en 16KHz de los clips de audio.

Esto se puede lograr gracias al script audio.ipynb que se encuentra en el repositorio de GitHub[2] con el que, dentro del *dataframe*, se accede a las rutas, para transformar con la librería los datos de audio y después insertar ese *string* en nuestros datos.

Este proceso presento bastantes problemas, pues los *strings* de audio que generaba, a pesar de que los clips no duraban los más largos más de 30 segundos, eran demasiado largos como para acabar produciendo un csv de datos manejable, pues el resultado era un archivo de 9 GB de datos para procesar, lo cual me era inviable de procesar tanto en local como en la nube de Google C*olaboratory* (además de presentar problemas incluso en la ejecución del modelo tras crear una cuenta de Google únicamente para guardar en su nube estos datos e intentar ejecutar el script que fine-tunea el modelo con estos datos).

Es por eso por lo que aparte de que los modelos de texto entrenaran con todos los datos, se decidió hacer la división entre los datos de *train*, *test* y *validation* desde fuera.

Así todos los modelos podrían a parte ser comparados y que la CNN de wav2vec fuera capaz de entrenar con menos datos y que fuera viable esto su entrenamiento, pues la propia RAM de *Google Colaboratory* no era capaz de leer todos los datos originales.

Igualmente, dentro de que los datos se separaron, para el de audio al final fueron usados solo una parte de estos pues seguía siendo demasiado, acabo entrenando con 1692 registros y haciendo el *test* con 424

[2] https://github.com/RodrigoRivasArevalo/TFG_Multimodal_final

registros, todo esto por la falta de recursos y el coste computacional, ya que al final como explicamos antes este modelo se basa en CNN, más costosas también más que los Transformers.

En este caso al querer abordarse el problema desde un punto de vista multimodal, se van a realizar distintos análisis basados en diferentes inputs, como se explicó antes, el texto y las representaciones del audio.

Estos datos y los scripts usados para llegar a ellos se encuentran en el siguiente repositorio de GitHub[3].

Para este problema en concreto se decidió que la mejor manera de abordarlo sería desde el punto de vista de la clasificación y la predicción, para así lograr unir cada frase con la etiqueta que le corresponda de las 6 que tenemos.

Entrenamiento general

Una vez se logra tener los datos preparados, es momento de *fine-tunear* los distintos modelos escogidos, el proceso para lograr esto es bastante parecido para cuando usamos los datos de las mismas estructuras, es decir, para los modelos de texto la manera de fine-tunearlos para adaptarlos a tu problema y a tus datos es muy parecida entre modelos, mientras que, para los modelos de audio, esto cambia un poco.

Pero aun con estas diferencias ambos modelos comparten la característica de ser entrenados en Google Colaboratory, a diferencia del resto de la implantación del código, esta se decidió hacer así debido a que esta plataforma nos proporcionaba la GPU que fue necesaria, ya que sin esta no se podría haber desarrollado el proyecto.

Dentro de los datos que se usan para el proyecto, además, hay varios documentos, estos son:

- Datos_wav: estos datos contienen todos los registros para el modelo sin las representaciones del audio, teniendo todas las columnas anteriormente mencionadas de los casi 10k registros. Este conjunto se usa para entrenar los modelos de texto al 100%, dividiendo dentro del script entre *test*, *train* y *validation*.
- Train, Test y Val: Estos datos son los mismos que se tienen en Datos_wav, pero estos están divididos en esos tres subconjuntos para poder entrenar todos los modelos con los mismos

[3] https://github.com/RodrigoRivasArevalo/TFG_Multimodal_final

datos y testearlos en igualdad de condiciones, estos *datasets* no contienen la información de las representaciones del audio, pues estos eran con los que entrenarían los modelos de texto.

- Train_final, Test_final y Val_final: Estos datos son los mismos registros que en Train, Test y Val respectivamente, pero esta vez conteniendo las representaciones del audio, para así subirlos a drive e intentar pasarlos al fine-tuneo de las redes del audio.

Es importante destacar que estos datos se consiguieron y plantearon su uso, pero al final no se pusieron a usar por el tamaño de estos y las limitaciones técnicas que Google *Colaboratory* te pone a sus recursos.

Es por eso por lo que se hizo una partición del 30% de los datos para poder manejar y alimentar el modelo de audio, esta partición con las representaciones del audio está en Datos_finale, dejando la tarea de *split* entre *train*, *test* y *validation* para que se realizara dentro de los scripts, asegurándonos de usar una semilla para esto.

- Texto_final: Por último, estos datos son los mismos que están en Datos_finale, pero sin las representaciones del audio y solo con el texto, estos datos existen porque para entrenar los modelos de texto la columna de las representaciones de audio se acabaría eliminando, por lo que no tiene sentido cargarla, y menos teniendo en cuento que esto requiere tiempo y computo.

Todos estos datos fueron subidos a Google Drive, y dependiendo de las necesidades, se cargan unos u otros

Entrenamiento con texto

Para el entrenamiento con los datos de texto, independientemente del modelo que se use, a nivel practico y de código para entrenar y adaptar el modelo, lo que se necesita hacer es lo que se hace en los notebooks que se encuentran en la carpeta de modelos, estos son: debert+test.ipynb, distilbert+test.ipynb, roberta+test.ipynb, Finanzas+test.ipynb, que se encuentran en el siguiente repositorio de GitHub[4].

En estos notebooks, lo que se hace es primero, importar las librerías necesarias para poder ejecutar el código.

[4] https://github.com/RodrigoRivasArevalo/TFG_Multimodal_final

Lo segundo que se hace es montar el Google *drive*, esto se hace para poder cargar más datos y de manera más rápida, los distintos *datasets* que acabaron resultando, estos se encuentran en el repositorio de GitHub, en la carpeta data.

Después se cargan los datos necesarios, dentro de los ya expuestos con anterioridad, se procede a eliminar las columnas que no son necesarias para el entrenamiento y a renombrar la columna de las etiquetas.

Después, se importan las métricas y las maneras de medir estas, así como se importa el tokenizador que se esté usando, el cual cambia dependiendo del modelo que se pretenda usar.

El tokenizador, como se explica antes, es el modelo que codifica las palabras, lo que cambia de uno a otro es el cómo separa estas mismas, ya que algunos modelos lo hacen de manera semántica, mientras que otro lo hacen por silabas, etc.

Tas cargar este tokenizador, se define la función que mapea los datos a través de este tokenizador, dejándolos en representaciones numéricas y capas de atención, para que el modelo las relacione con las distintas etiquetas, como se explica en el marco teórica, añadiéndole pesos para la atención.

Por último, se llama al transformer, la versión y base que se esté usando en el momento dentro de la lista de los usados y expuestos anteriormente, pasándole además el número de etiquetas con el que se cuenta a la hora de cargarlo.

Para después pasarle los distintos argumentos con los que la red tendrá que entrenar, como son el tamaño de los batches, la evaluación que se va a realizar (y que viene definida antes en el código), los datos tanto de entrenamiento como de testeo etc.

Esto se realiza con los tensores y estructuras de transformadores base que proporciona pytorch.

Este proceso logra devolver los distintos modelos que se pueden encontrar en mi página de *hugginface*, con los cuales, una vez entrenados, pueden ser expuestos a los datos de *validation* para así lograr sacar el *accuracy* y lograr ver como el modelo se comporta, además de poder comparar así los distintos modelos, con los distintos datos y más tarde, juntar los mejores.

Entrenamiento con audio

Para el entrenamiento con los datos de audio y las representaciones de este en los vectores a 16KHz, independientemente del modelo que se use, a nivel practico y de código para entrenar y adaptar el modelo, lo que se necesita hacer es lo que se hace en los notebooks de: HubertAudio.ipynb, Wav2Vec.ipynb y Wav2Vec_robust.ipynb que se encuentran en el siguiente repositorio de GitHub[5].

En estos notebooks, lo que se hace es primero, importar las librerías necesarias para poder ejecutar el código.

Lo segundo que se hace es montar el Google *drive*, esto se hace para poder cargar más datos y de manera más rápida, los distintos *datasets* que acabaron resultando, estos se encuentran en el repositorio de GitHub[6], en la carpeta data.

Después se cargan los datos necesarios, dentro de los ya expuestos con anterioridad, se procede a eliminar las columnas que no son necesarias para el entrenamiento y a renombrar la columna de las etiquetas.

El siguiente paso, que no es necesario para los modelos que entrenan y clasifican texto, es el de crear una función para asegurarnos que el array con el que va a tratar el modelo este en el formato adecuado, para después pasársela a nuestros datos.

Lo siguiente es hacer un *split* de los datos entre *train*, *test* y *validation*.

Después, se importan las métricas y las maneras de medir estas, así como se importa el *feature_extractor* que se esté usando, el cual cambia dependiendo del modelo que se pretenda usar.

El *feature_extractor*, es para el modelo que trata con audio (en concreto para los modelos basados en Wav2Vec) como un tokenizador, que juega un papel crucial, porque convierte las señales de audio en bruto en representaciones que son más manejables y relevantes para el modelo.

Por último, se llama al *transformer*, la versión y base que se esté usando en el momento dentro de la lista de los usados y expuestos anteriormente, pasándole además el número de etiquetas con el que se cuenta a la hora de cargarlo.

[5] https://github.com/RodrigoRivasArevalo/TFG_Multimodal_final
[6] https://github.com/RodrigoRivasArevalo/TFG_Multimodal_final

Para después pasarle los distintos argumentos con los que la red tendrá que entrenar, como son el tamaño de los batches, la evaluación que se va a realizar (y que viene definida antes en el código), los datos tanto de entrenamiento como de testeo etc. Esto se realiza con los tensores y estructuras de transformadores base que proporciona pytorch.

Este proceso logra devolver los distintos modelos que se pueden encontrar en mi página de *hugginface*, con los cuales, una vez entrenados, pueden ser expuestos a los datos de *validation* para así lograr sacar el *accuracy* y lograr ver como el modelo se comporta, además de poder comparar así los distintos modelos, con los distintos datos y más tarde, juntar los mejores.

Ensemble final

Es en este momento cuando entra el factor de la multimodalidad, para intentar mejorar las puntuaciones obtenidas en los pasos anteriores con los modelos tanto de texto como de audio, y que se exponen en el apartado de resultados.

En nuestro caso, tras analizar nuestros modelos y el factor de que estos son distintos, pues para poder aplicar la multimodalidad mencionada tienen que serlo, se ha optado por realizar un ensemble del tipo *Stacking.*

Se eligió este tipo de ensemble porque, como se comenta, al tener que combinar modelos distintos, uno para el texto y otro para el audio, creemos que la creación un metamodelo que combine las predicciones de los dos mejores modelos de estas dos disciplinas distintas seria el mejor acercamiento.

Este ensemble viene realizado en el notebook del GitHub[7] EnsembleBien.ipynb, dentro de la carpeta de código y de la de modelos.

En este código lo que se hace es cargar los modelos que se preentrenaron y fine-tunearon anteriormente y que se encuentran en el siguiente perfil de *hugginface*[8].

Tras eso se llama a las funciones de preprocesado y tokenizacion de los modelos, para que se les pasen a los datos que se tienen y con los que se va a entrenar este nuevo ensemble de ambos modelos, estos datos que anteriormente fueron expuestos, y con los que posteriormente se evaluara.

[7] https://github.com/RodrigoRivasArevalo/TFG_Multimodal_final
[8] https://huggingface.co/Drigoro

Después se crea la clase del ensemble, en el cual se podrán pasar los hiperparametros necesarios y a la que se llamara tanto en la función con la que se entrenara al mismo y en la función que analiza los resultados.

Por último se llama a las librerías de evaluate para obtener los resultados con el dataset de validation y el ensemble con mejores resultados.

Resultados

Una vez ha sido expuesto el marco teórico necesario y la metodología seguida, tanto los paso para el preprocesado de los datos, así como la estructura y fine-tuneo de las distintas redes neuronales, tanto de atención como convolucionales usadas, es momento de que se expongan los resultados obtenidos con los distintos modelos.

Los resultados obtenidos se pueden analizar en base a distintos grupos, pero nosotros hemos preferido hacer una división de los resultados en función de los datos con los que entrena, tanto en formato como en cantidad, pues como se expuso con anterioridad, los distintos modelos de texto se han entrenado, tanto con el conjunto total de datos, como con un subconjunto más pequeño.

La manera de mostrar estos resultados va a ser mediante una comparación de la precisión (en ingles *accuracy*) entre los distintos modelos que son similares, así como una demostración de la precisión del ensemble de los mejores modelos de audio y texto entrenados con el subconjunto reducido para que el audio pueda analizarse.

Accuracy

Antes de mostrar los resultados obtenidos, es importante saber y exponer en base a que se están midiendo estos, pues dentro del mundo del análisis de datos existen muchas métricas para comparar y evaluar los resultados, para este proyecto se eligió el *accuracy*, previamente expuesto en el estado del arte, aquí se van a exponer los motivos por los que se decidió que esta medida es la indicada.

Se decidió optar por esta medida porque en comparación con otras también usadas en este campo como son el F1 o el MCC por varios motivos. El primero de estos es la sencillez de análisis de esta métrica, ya que esta medida es fácil de interpretar y explicar a todo tipo de audiencias sin perder legitimidad por ello. Un *accuracy* del 80% significa que el modelo predice correctamente 80 de cada 100 ejemplos, lo cual es intuitivo.

Además, como los datos están balanceados en las 6 etiquetas que se intentan predecir el *accuracy* es una métrica adecuada, pues contando con aproximadamente el mismo número de ejemplos en cada una de las seis clases, el *accuracy* da una buena idea del rendimiento general del modelo, porque no está sesgado hacia ninguna clase en particular.

Por último, se decidió optar por esta medida porque en el problema no existe ningún tipo de sesgo por importancia en el tipo de error que se obtenga, lo que significa que cada error de clasificación tiene la misma importancia, no importa que sea del tipo uno o del tipo dos.

Estas conclusiones se obtuvieron gracias al artículo (M & M.N, 2015).

Resultados texto entero

Los primeros resultados que se van a mostrar en este apartado son aquellos obtenidos de los distintos modelos que se entrenaron con los 9 mil registros de texto y sus respectivas emociones, los modelos utilizados para esta tarea y que se van a exponer en la siguiente tabla son:

RoBERTa, DistilBERT, DeBERTa, DistilRoBERTa-sentiment-finetunned (DistilRoBERTa)

	RoBERTa	DistilBERT	DeBERTa	DistilRoBERTa
Punctuation de Precision	0.5206	0.5385	0.5273	0.4777

1 Tabla precisión texto

Además de la puntuación de precisión también se han obtenido resultados de los modelos referentes al rendimiento de estos, esa información es la siguiente:

	RoBERTa	DistilBERT	DeBERTa	DistilRoBERTa
Tiempo de ejecucion	21.565 Sec	26.247 Sec	50.964 Sec	29.034 Sec

2 Tabla tiempo texto

Hay que destacar que estos valores son el tiempo tardado en hacer la validación del modelo, y no el tiempo tardado en entrenar.

Estos modelos ya entrenados y fine-tuneados se encuentran subidos en el siguiente perfil de huggingface[9].

Estos serían los resultados de los distintos modelos de textos entrenados con la totalidad de los datos, que se analizaran, compararan y usaran más adelante, en el apartado de discusión y conclusiones.

Resultados texto parcial

Ahora se van a mostrar los resultados obtenidos de los distintos modelos que se entrenaron con una parte de los registros del texto y sus respectivas emociones, esto se ha hecho para así hacer una comparación justa con los modelos de clasificación de audio, puesto que este modelo no puede entrenar con tantos datos, como se expuso anteriormente.

Los modelos utilizados para esta tarea y que se van a exponer en la siguiente tabla son los mismos que en el apartado anterior:

RoBERTa, DistilBERT, DeBERTa, DistilRoBERTa-sentiment-finetunned (DistilRoBERTa)

	RoBERTa	DistilBERT	DeBERTa	DistilRoBERTa
Punctuation de Precision	0.5068	0.4982	0.4829	0.4183

3 Tabla precisión texto pequeño

Además de la puntuación de precisión también se han obtenido resultados de los modelos referentes al rendimiento de estos, esa información es la siguiente:

	RoBERTa	DistilBERT	DeBERTa	DistilRoBERTa
Tiempo de ejecucion	74.600 Sec	51.252 Sec	21.565 Sec	5.608 Sec

4 Tabla tiempo texto pequeño

[9] https://huggingface.co/Drigoro

Hay que destacar que estos valores son el tiempo tardado en hacer la validación del modelo, y no el tiempo tardado en entrenar.

Estos modelos ya entrenados y fine-tuneados se encuentran subidos en el siguiente perfil de huggingface[10].

Estos serían los resultados de los distintos modelos de textos entrenados con una parte de los datos, que se analizaran, compararan y usaran más adelante, en el apartado de discusión y conclusiones.

Resultados Audio

En este apartado, se van a mostrar los resultados de los modelos que entrenaron para clasificar las emociones con las pistas de audio, aquí solo se van a mostrar unos resultados de estos modelos, pues como se explicó, no se podía entrenar con todos los datos.

Los modelos fine-tuneados para esta tarea fueron:

HuBERT-ls960, Wav2Vec2, Wav2vec2-large-robust-12-ft-emotion-msp-dim

	HuBERT	Wav2Vec2	Wav2Vec2-large-robust
Puntuación de Precisión	0.3108	0.3543	0.3254

5 Tabla precisión audio

Además de la puntuación de precisión también se han obtenido resultados de los modelos referentes al rendimiento de estos, esa información es la siguiente:

	HuBERT	Wav2Vec2	Wav2Vec2-large-robust
Tiempo de ejecucion	113.600 Sec	87.252 Sec	97.321 Sec

6 Tabla tiempo audio

[10] https://huggingface.co/Drigoro

Hay que destacar que estos valores son el tiempo tardado en hacer la validación del modelo, y no el tiempo tardado en entrenar.

Estos modelos ya entrenados y fine-tuneados se encuentran subidos en el siguiente perfil de huggingface[11].

Estos serían los resultados de los distintos modelos de audio entrenados con una parte de los datos, que se analizaran, compararan y usaran más adelante, en el apartado de discusión y conclusiones.

Resultados ensemble

Por último, visto los resultados anteriores y como se expuso, se realizo un ensemble con los modelos Wav2Vec2 y con RoBERTa, por lo que en este apartado se van a mostrar los resultados del ensemble con estos modelos:

	Wav2Vec2 y RoBERTa
Puntuación de Precisión	62.971%

7 Tabla precisión ensemble

[11] https://huggingface.co/Drigoro

Discusión y Conclusiones

En base a los resultados presentados anteriormente se pueden sacar distintas conclusiones en cuanto al rendimiento del modelo.

Estas conclusiones, al igual que los resultados, van a estructurarse alrededor de los distintos grupos de modelos en función de los tipos de datos con los que entrena y el tamaño de estos mismos.

Estas conclusiones, lo único, es que no se van a centrar solo en los grupos de manera individual, ya que además se van a interrelacionar los grupos tanto para compararlos como para crear y mostrar los resultados del modelo multimodal que se va a crear con los dos mejores modelos pequeños, que a su vez se comparara con el resto.

Conclusiones del texto

Las primeras conclusiones que sacamos a simple vista y más obvias son que, evidentemente, en los modelos de texto, los modelos con más entradas, o los modelos que cuentan con todos los datos para entrenar son más eficaces y tienen mejores puntuaciones de precisión que sus contrapartes fine-tuneados con menos datos.

Dentro de esto, igualmente, podemos destacar que los modelos base son mejores que otros modelos más específicos y fine-tuneados para tareas similares a esta, ya que, en ambos casos, se puede observar que el modelo fine-tuneado con anterioridad para el análisis y predicción de sentimientos DistilRoBERTa-sentiment-finetunned, ha sido el que peores resultados ha obtenido.

Esto se debe a que el modelo más específico, al final, tiene unos sesgos que no somos capaces de controlar nosotros, mientras que los otros modelos base como son DistilBERT, RoBERTa y DeBERTa, al ser más generales son capaces de adaptarse mejor a este marco más específico, pues es más sencillo ir de lo general a lo especifico para estos modelos, que ir de lo especifico a otra cosa especifica, parecida, pero distinta.

Dentro de los modelos entrenados con todos los datos, el que mejor resultados ha dado tanto en la validación como en el testeo dentro del propio entrenamiento del modelo ha sido DistilBERT, lo cual nos ha resultado sorprendente a la vez que gratificante, ya que estos modelos destilados de otros más

generales existen para lograr resultados parecidos a los originales usando muchos menos recursos, pero por lo general, estos suelen dar peores resultados.

Es de ahí de donde viene nuestra sorpresa, pues, estos modelos suelen dar peores resultados, pero en nuestro caso fue el que mejor lo hizo, al menos con todos los datos, por lo que una de las conclusiones que podríamos obtener de este trabajo es que, para realizar una clasificación por una etiqueta con varios niveles, de muchas muestras de poco texto, tanto por rendimiento como por resultados, el mejor de los modelos sería un DistilBERT, dando casi un 55% de acierto.

Por otra parte, si nos desplazamos a estos mismos modelos, pero entrenados con ese pequeño porcentaje de los datos originales para compararlo en las mismas condiciones que los modelos de audio, nos encontramos que el que mejor resultados devuelve ahora no es DistilBERT, sino que es RoBERTa.

Creemos que esto tiene sentido, ya que, al tener menos entradas, un modelo más robusto es capaz de encontrar mejores relaciones entre los datos en el entrenamiento de sus capas de atención, por lo que aquí la potencia y robustez del propio modelo juega un papel clave.

También se tiene que destacar de estos modelos en general, que, a comparación con sus versiones con todos los datos, no sufren demasiado por contar solo con una fracción de los datos.

Es cierto que la diferencia de datos se puede notar, ya que donde sus predecesores superan el 50% de precisión con facilidad, en estos es solo el basado en RoBERTa el que logra llegar al 50% mientras que el resto se quedan a las puertas.

Pero esta bajada es menor de la que esperábamos al principio, pues en ningún caso se desploma la precisión más de un 5%, lo cual, como hemos comentado, nos parece sorprendente.

Conclusiones del audio

Por último, en cuanto a los modelos de una sola modalidad, nos quedaría por analizar y comentar los resultados de los modelos basados en las representaciones de audio de los momentos de las escenas.

Para estos modelos, tras el estudio y análisis de las posibilidades que se tenían, vimos que para lo que nosotros queríamos, que era clasificar el audio en función de las propias ondas en distintas etiquetas, los modelos desarrollados y fine-tuneados solo se basaban en dos, HuBERT y Wav2Vec2, por lo que

acabamos fine-tuneando estos dos modelos base, así como un tercero, basado en Wav2Vec2 y fine-tuneado previamente para la clasificación de sentimientos.

Dentro de las conclusiones que hemos obtenido de esto, empezaremos hablando de la comparación entre HuBERT y Wav2Vec2, ya que, dentro de estos dos, hay un claro ganador en cuanto a precisión dentro del problema, y ese es Wav2Vec2.

Esto lo podemos ver ya que tanto el Wav2Vec2 base como el Wav2vec2-large-robust-12-ft-emotion-msp-dim han dado resultados de entre un 2% y un 4% mejores.

Además, en este caso, nos pasa como con los modelos de texto, que se puede ver que son más eficaces los modelos bases que van desde un punto de vista más general y están optimizados para ser fine-tuneados frente a modelos de la comunidad o de otras empresas más específicos.

A parte de esto, otra de las conclusiones que hemos podido sacar de la comparación entre estos modelos y los de los otros dos grupos ha sido que, si de un estudio de pura comparación se tratara, los modelos entrenados con el audio serian bastante inferiores en cuanto a la precisión que ofrecen, pues los modelos de texto, hasta los peores, se mueven cerca de la precisión del 50%, mientras que el mejor de los modelos de audio se queda en el 35% de precisión.

Esto se debe a que al final es más complejo para los modelos actuales analizar y encontrar relaciones entre los *strings* de audio que encontrarlo en el NLP que se puede hacer de las cadenas de texto.

Esto debido tanto a que las representaciones de audio son más complejas, como a que los modelos de texto están más estudiados, analizados y por lo general, optimizados, debido a la gran popularidad del NLP.

Una vez visto, analizado y comparado los distintos modelos por ellos mismos, podemos llegar a la conclusión de que los mejores modelos son: DistilBERT para todo el texto y Wav2Vec2 y RoBERTa para los datasets más pequeños, lo que nos muestra lo que comentamos, que los modelos de texto son mejores para este problema.

Multimodalidad

Tras un entrenamiento con 10 épocas usando como base los dos modelos expuestos antes, se logró obtener una precisión del `62.971%`.

Este resultado es bastante positivo, pues confirma una de las hipótesis de las que se parte a la hora de realizar el trabajo, que es que el concepto de la multimodalidad es algo que será beneficioso para la clasificación de estas emociones, por lo que la primera de las conclusiones que podemos sacar de esto es esa, que una de nuestras hipótesis es cierta.

Esto se debe a que la combinación de ambos modelos permite al ensemble ser más preciso en sus decisiones y tener que tomar menos riesgos, pues ya no solo cuenta con dos modelos, sino que estos al analizar cosas distintas le aportan diferentes puntos de vista.

Por lo que nos podríamos atrever a concluir que ha sido un éxito.

Además, creo personalmente que esto ha sido todo un éxito, pues, aunque no se obtengan unos resultados loquísimos en cuanto a precisión, se ve que son buenos, pues al final el modelo tiene que predecir entre 6 etiquetas distintas y logra acertar el 63% de los casos.

Conclusiones generales.

En resumen, en este trabajo, por un lado, se ha logrado crear un modelo que es revolucionario pues combina varias disciplinas de una manera multimodal para lograr la clasificación de las emociones, consiguiendo resultados bastante positivos y con los que se puede estar contento, no obstante, quedando mucho posible trabajo de mejora para un futuro.

Por otro lado, a nivel personal opino que es un trabajo con el que estoy muy contento, pues a parte de haber conseguido unos resultados que ni me esperaba que fueran tan positivos, he logrado aprender muchísimo e introducirme al mundo real de las Inteligencias artificiales y las redes neuronales, algo que no pensaba que a estas alturas fuera a lograr de la carrera, pero que, gracias a mi esfuerzo, la verdad, he conseguido.

Limitaciones y futuras líneas de investigación

Limitaciones

Durante la realización del trabajo han existido muchos momentos donde las limitaciones han jugado un papel crucial, ya que estas han dificultado y entorpecido el camino, haciendo que así el proyecto se pudiera ver comprometido.

Dentro de las limitaciones con las que me he podido encontrar realizando este trabajo, creo que es importante destacar que han existido tanto limitaciones técnicas, como limitaciones de conocimiento.

Las limitaciones técnicas han sido aquellas que han afectado al desarrollo del trabajo a nivel de recursos, debido a que los equipos de los que disponía para hacer el trabajo, pues mi equipo personal no era el más potente para realizar las acciones que entrenar un modelo de IA requiere, pues estas son bastante costosas.

Es cierto que esta limitación se logró solventar gracias a los recursos que Google Colaboratory ofrece para la programación, esto también se podría haber solventado con los equipos de la universidad, pero no fue posible tenerlos listos teniendo en cuenta el ajustado calendario del que disponíamos y la gran cantidad de cosas que había que hacer para que esto fuera posible.

Además, dentro del apartado técnico también hubo ciertas limitaciones de espacio, pues al final al ser el modelo multimodal este contaba con datos que, por sus propias características, son más pesados, como lo son el video y el audio, esto me trajo problemas de espacio y memoria dentro de los equipos y dentro del procesado y puesta a punto de estos datos.

Esto se acabó solventando, mejorando mis equipos y recursos dentro de las posibilidades que tuve tanto por temas económicos como de tiempo.

Por otra parte, hay que destacar las limitaciones en cuanto al conocimiento del tema tratada y en el que se basa el trabajo, que es la Inteligencia Artificial, y más en concreto los modelos usados como son los Transformers y las CNN.

Queremos destacar esto ya que este trabajo ante todo ha sido un trabajo de investigación por la parte de los creadores, pues dentro de mi formación apenas existe una formación mínimamente teórica en estos temas, y mucho menos una formación práctica en las complejidades que puede tener desarrollar y fine-

tunear una solución a un problema de inteligencia artificial como este, ya sea al problema de los modelos individuales, como por otra parte, al problema de crear el ensemble de ambos modelos.

También hay que destacar que este problema ha sido el reto más duro, pero a su vez el más gratificante, pues se ha podido solventar gracias a horas de estudio, artículos y consejos por parte del tutor.

Por último, pero no menos importante, se tiene que comentar las limitaciones en cuanto a los recursos humanos y de calendario, pues la final todo este trabajo es uno que se podría expandir muchísimo más, pero al final una sola persona y durante solo algunos meses, pues ha sido capaz de llegar hasta aquí.

Futuras líneas de Investigación

En cuanto a las futuras líneas de investigación del proyecto, estas pueden ser muchas y variadas, pues al final con este trabajo solo se logra tocar la superficie del complejo y fascinante mundo de la multimodalidad.

Una de estas líneas sería el estudio de los propios datos, alguna clase de análisis estadístico o análisis exploratorio de estos mismos, para ver cómo se agrupan, analizarlos sin necesidad de Inteligencia artificial para ver si existe alguna relación más simple entre estos mismos que pueda derivar en nuevas maneras de abordar el problema o en relaciones más simples con las que lograr optimizar así la obtención de estos resultados.

Por otra parte, dentro de las futuras líneas de investigación estaría el centrarse en la propia investigación y adaptación de los distintos modelos.

Tanto centrándose en sus hiperparametros y como cambiando estos se puede maximizar la precisión de los resultados obtenidos, así como explorando distintos modelos para intentar abordar el problema y mejorar los resultados.

Aunque se tiene que destacar que esto sería más eficaz para los modelos de texto, pues como se comenta anteriormente, solo se utilizan dos tipos de modelos base para realizar la clase de análisis que nosotros queremos en este trabajo con los datos de audio.

Por último, la línea de investigación que creemos que sería más interesante intentar continuar en futuras actualizaciones o mejoras sería la de mejorar el modelo de *ensembles*.

Esto se podría lograr a su vez por distintos medios, ya que se puede hacer de manera indirecta, estudiando más y ampliando y mejorando los propios modelos en los que se basa, como se explica anteriormente.

Así como se puede lograr aumentando la potencia de este ensemble, cambiando como interactúan los modelos para que esto sea más complejo, aumentando el número de modelos que componen el ensemble, aumentando y ajustando los valores de algunos hiperparametros, etc.

También hay que destacar, que las líneas de investigación, no creemos que se puedan centrar únicamente en mejorar de manera bruta los resultados, pues también creo que sería muy interesante avanzar en temas de aplicabilidad para el proyecto, pues la base de este es un concepto que, aunque complejo de realizar, para las personas es demasiado simple como para ser real.

Es por eso por lo que un estudio más profundo de las emociones, y de la preparación y etiquetación de estos datos podría aportar una visión mucho más realista y aplicable a la solución.

Así como un estudio más profundo de los posibles campos donde esto puede ser útil y sobre todo de su implementación en ellos podría ser una line da investigación muy interesante.

Referencias bibliográficas

Aitken, K., Ramasesh, V. V., Cao, Y., & Maheswaranathan, N. (2021). *Understanding How Encoder-Decoder Architectures Attend* (arXiv:2110.15253). arXiv. http://arxiv.org/abs/2110.15253

Baevski, A., Zhou, H., Mohamed, A., & Auli, M. (2020). *wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations* (arXiv:2006.11477). arXiv. http://arxiv.org/abs/2006.11477

Borja-Robalino, R., Monleón-Getino, A., & Rodellar, J. (s. f.). *Estandarización de métricas de rendimiento para*.

Carneiro, T., Medeiros Da Nobrega, R. V., Nepomuceno, T., Bian, G.-B., De Albuquerque, V. H. C., & Filho, P. P. R. (2018). Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications. *IEEE Access*, *6*, 61677-61685. https://doi.org/10.1109/ACCESS.2018.2874767

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* (arXiv:1810.04805). arXiv. http://arxiv.org/abs/1810.04805

Dietterich, T. G. (2000). Ensemble Methods in Machine Learning. En G. Goos, J. Hartmanis, & J. Van Leeuwen, *Multiple Classifier Systems* (Vol. 1857, pp. 1-15). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-45014-9_1

Durante, Z., Huang, Q., Wake, N., Gong, R., Park, J. S., Sarkar, B., Taori, R., Noda, Y., Terzopoulos, D., Choi, Y., Ikeuchi, K., Vo, H., Fei-Fei, L., & Gao, J. (2024). *Agent AI: Surveying the Horizons of Multimodal Interaction* (arXiv:2401.03568). arXiv. http://arxiv.org/abs/2401.03568

Gelbukh, A., & Sidorov, G. (2010). *Procesamiento automático del español con enfoque en recursos léxicos grandes* (2. ed. ampliada y rev). Instituto Politécnico Nacional.

Hershey, S., Chaudhuri, S., Ellis, D. P. W., Gemmeke, J. F., Jansen, A., Moore, R. C., Plakal, M., Platt, D., Saurous, R. A., Seybold, B., Slaney, M., Weiss, R. J., & Wilson, K. (2017). *CNN Architectures for Large-Scale Audio Classification* (arXiv:1609.09430). arXiv. http://arxiv.org/abs/1609.09430

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). *RoBERTa: A Robustly Optimized BERT Pretraining Approach* (arXiv:1907.11692). arXiv. http://arxiv.org/abs/1907.11692

M, H., & M.N, S. (2015). A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process*, *5*(2), 01-11. https://doi.org/10.5121/ijdkp.2015.5201

Opitz, D., & Maclin, R. (1999). Popular Ensemble Methods: An Empirical Study. *Journal of Artificial Intelligence Research*, *11*, 169-198. https://doi.org/10.1613/jair.614

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2020). *DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter* (arXiv:1910.01108). arXiv. http://arxiv.org/abs/1910.01108

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023). *Attention Is All You Need* (arXiv:1706.03762). arXiv. http://arxiv.org/abs/1706.03762

Índice de figuras

1 Función de activación ReLU 11
2 Función de activación SoftMax 12
3 Función Scaled dot-product 14
4 Esquema Scaled dot-product 14
5 Esquema multi-head 14
6 Función multi-head 14
7 Esquema de un transformer 18
8 Ejemplo esquemático tokenización 22
9 Estructura por capas de BERT 23
10 Comparación RoBERTa y BERT 24
11 Rendimiento DistilBERT 3 26
12 Rendimiento DistilBERT 2 26
13 Rendimiento DistilBERT 1 26
14 Función de activación GeLU 27
15 Gráfico datos y su estructura 32

Índice de tablas

1 Tabla precisión texto 45
2 Tabla tiempo texto 45
3 Tabla precisión texto pequeño 46
4 Tabla tiempo texto pequeño 46
5 Tabla precisión audio 47
6 Tabla tiempo audio 47
7 Tabla precisión ensemble 48