UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Sistemas Informáticos y Computación

Clasificación de dominios a nivel de segmento utilizando modelos preentrenados.

Trabajo Fin de Máster

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

AUTOR/A: Gómez Rosabal, Claudia

Tutor/a: Paredes Palacios, Roberto

Cotutor/a externo: CHATZITHEODOROU, KONSTANTINOS

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MIARFID Official Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Clasificación de dominios a nivel de segmento utilizando modelos preentrenados

TRABAJO FIN DE MÁSTER

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Autor: Claudia Gómez Rosabal

Tutores: Roberto Paredes Palacios
Konstantinos Chatzitheodorou

Curso 2022-2023

Resum

Amb l'aparició d'Internet, la quantitat de dades disponibles ha augmentat cada vegada més. D'aquí ve que classificar el text en dominis tinga una gran utilitat a l'hora d'estructurar i organitzar les dades per a traure'ls més partit i ser utilitzats per la comunitat científica per a entrenar models d'Intel·ligència Artificial. A més, l'entrenament de models específics de domini pot millorar significativament la precisió obtinguda en casos com la traducció automàtica i altres tasques de processament de text.

L'objectiu del present treball és obtindre segments etiquetats en 15 dominis, utilitzant conjunts de dades públiques i de la web etiquetats manualment per a realitzar un postanàlisi i una selecció dels mateixos que permeta crear un conjunt de dades prou representatiu. Hui dia, el Processament del Llenguatge Natural compta amb una gran varietat de models pre-entrenats basats en grans corpus de dades que representen una gran quantitat d'informació, la qual cosa facilita la realització de moltes tasques en aquest camp. Després de l'estudi de diversos d'aquests models, com BERT, Roberta i GPT-3, entre altres, s'entrenarà un model classificador d'alta qualitat. Finalment, s'analitzaran els resultats i se seleccionarà el model de millor rendiment segons les mètriques d'avaluació.

Paraules clau: nlp, classificació de textos, etiquetat de temes, classificació de dominis.

Resumen

Con la aparición de Internet, la cantidad de datos disponibles ha aumentado cada vez más. De ahí que clasificar el texto en dominios tenga una gran utilidad a la hora de estructurar y organizar los datos para sacarles más partido y ser utilizados por la comunidad científica para entrenar modelos de Inteligencia Artificial. Además, el entrenamiento de modelos específicos de dominio puede mejorar significativamente la precisión obtenida en casos como la traducción automática y otras tareas de procesamiento de texto.

El objetivo del presente trabajo es obtener segmentos etiquetados en 15 dominios, utilizando conjuntos de datos públicos y de la web etiquetados manualmente para realizar un post-análisis y una selección de los mismos que permita crear un conjunto de datos suficientemente representativo. A día de hoy, el Procesamiento del Lenguaje Natural(PLN) cuenta con una gran variedad de modelos preentrenados basados en grandes corpus de datos que representan una gran cantidad de información, lo que facilita la realización de muchas tareas en este campo. Tras el estudio de varios de estos modelos, como BERT, RoBERTa y GPT-3, entre otros, se entrenó un modelo clasificador de alta calidad. Por último, se analizaron los resultados y se seleccionó el modelo de mejor rendimiento según las métricas de evaluación y consideraciones adicionales.

Palabras clave: nlp, clasificación de textos, etiquetado de temas, clasificación de dominios.

Abstract

With the emergence of the Internet, the amount of data available has increased considerably. Hence classifying text into domains has a great utility in structuring and organizing the data to get more out of it and to be used by the scientific community to train Artificial Intelligence models. Moreover, training domain-specific models can significantly improve the accuracy obtained in cases such as machine translation and other text processing tasks.

The goal of the present work is to obtain labeled segments in 15 domains, using public datasets and manually labeled web crawled data to perform a post-analysis and selection of them to create a sufficiently representative datasets. As of today, Natural Language Processing has a variety of pre-trained models based on large corpora of data that represent a large amount of information, making many tasks in this field easier to perform. A high-quality classifying model was fine-tuned following the study of several of these models, including BERT, RoBERTa, and GPT-3, among others. Finally, the results were analyzed, and the best-performing model was selected according to the evaluation metrics and other considerations.

Key words: nlp, text classification, topic labeling, domain classification.

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

Primero y ante todo, quiero expresar mi más profundo agradecimiento a mis tutores, cuya orientación y dedicación han sido fundamentales en el desarrollo de esta tesis. Su experiencia y conocimientos han sido una fuente constante de inspiración, y me han guiado a través de los momentos más difíciles del proceso de investigación.

Mi gratitud también se extiende a mis amigos, cuyo apoyo incondicional y estímulo me han permitido superar los obstáculos y desafíos que surgieron en el camino. Sus palabras de aliento y actitudes positivas han sido esenciales para mantenerme enfocada y motivada.

No puedo dejar de expresar mi más sincero agradecimiento a mi familia, en especial a mis padres y a mi hermana, quienes siempre han confiado en mis capacidades y me han brindado su amor y apoyo incondicional.

Finalmente, quiero agradecer a la Universidad Politécnica de Valencia por proporcionarme las herramientas, los recursos y el entorno propicio para llevar a cabo esta investigación. Mi gratitud se extiende a todos los profesores, colegas y personal administrativo que han contribuido a mi formación y han sido parte de mi experiencia académica.

A todos ustedes, mi más sincero agradecimiento.

CAPÍTULO 1
Introducción

1.1 Clasificación en dominios

La clasificación de texto en dominios consiste en asignar una categoría específica a este teniendo en cuenta el contenido del mismo y características como estilo, tema, nivel de formalidad, género, etc. Esta tarea puede tener varias variantes, como clasificación de textos cortos o documentos completos.

Con la aparición de Internet, la cantidad de datos disponibles ha aumentado cada vez más. De ahí que clasificar texto en dominios tenga una gran utilidad a la hora de estructurar y organizar datos, siendo esta su principal importancia en el Procesamiento del Lenguaje Natural (PLN) y en el área de la Inteligencia Artificial (IA) en general.

Categorizar datos según el contenido no es solamente útil para la recuperación y organización de la información de forma más rápida y eficiente, sino que además puede ser de gran ayuda para otras tareas de procesamiento de texto. Por ejemplo, para crear sistemas de recomendación basados en contenido textual, al recomendar contenido a los usuarios que se encuentren en el mismo dominio que sus intereses, resumen automático, traducción automática, análisis de sentimientos, y cualquier otra tarea que se beneficie de tener información acerca del tema general de los datos.

Por ejemplo, en [33], encontraron que *BioBERT*, una adaptación de BERT [17] entrenada con un conjunto de datos del dominio biomédico, logró mejoras en los resultados en tareas como detección de entidades y sistemas de preguntas y respuestas en dominio biomédico respecto a la versión original de *BERT*. Resultados similares se han reportado en artículos como [4], [1] y [2] con dominios de artículos científicos, textos clínicos y de finanzas respectivamente. Estos son sólo algunos ejemplos, en general, la adaptación y entrenamiento de los modelos en datos específicos de un dominio pueden mejorar el rendimiento comparado a utilizar un modelo entrenado en datos genéricos para esas tareas de PLN.

1.2 Modelos de Lenguaje Preentrenados

Los modelos de lenguaje preentrenados son modelos de aprendizaje automático que se han entrenado en cantidades enormes de datos textuales, como libros, artículos y páginas web, para aprender patrones y relaciones dentro del lenguaje natural.

Estos modelos han revolucionado el PLN y han supuesto mejoras significativas en diversas tareas, como la traducción automática, la clasificación de textos, el análisis de sentimientos y sistemas de preguntas y respuestas (*question-answering*). Entre los modelos

más utilizados se encuentran BERT [17], GPT [55], XLNet [76], RoBERTa [39], T5 [57], entre otros.

1.3 Motivación

Uno de los propósitos de este trabajo es el estudio e investigación de varios modelos de lenguaje preentrenados, que están cada vez más presentes en las soluciones de la comunidad académica a todo tipo de tareas relacionadas con el lenguaje natural, y que además, han revolucionado este campo. Tener conocimientos de estos modelos es de especial interés, sobre todo teniendo en cuenta el furor de los últimos meses con aplicaciones de los modelos de lenguaje de gran escala como GPT-3. [7].

Otro propósito del mismo, es desarrollar un clasificador de dominios basado en 15 clases que serán especificadas posteriormente. Este clasificador se desarrolla dentro del contexto profesional de la empresa Pangeanic, en la cual me encuentro trabajando actualmente. Pangeanic es una empresa dedicada al PLN, y ofrece productos relacionados con la traducción automática y anonimización de textos, entre otros. El interés en el clasificador de dominios se basa en la organización y categorización de los datos internos de la empresa con el objetivo de mejorar la calidad de las traducciones automáticas al tener modelos entrenados en dominios específicos.

Finalmente, me motiva en gran medida la realización de este trabajo por los conocimientos que serán adquiridos en el área del PLN, la cuál pretendo que sea mi especialización en mi trayectoria profesional, y la consolidación de los conocimientos obtenidos en el máster.

1.4 Objetivos

En el presente trabajo se pretende investigar acerca de varios modelos de lenguaje preentrenados para la clasificación de textos en dominios. Se creará un *dataset* etiquetado en dominios y se seleccionará el modelo que mejor se ajuste a las necesidades y mejores resultados obtenga.

A continuación se listan los objetivos principales que se pretenden lograr con esta tesis:

1. Estudiar el estado del arte en la clasificación de textos, y en especial, clasificación de dominios, utilizando modelos de lenguaje preentrenados.

2. Crear un *dataset* de segmentos etiquetados en 15 clases, haciendo uso de *datasets* públicos y datos obtenidos mediante técnicas de *crawling*.

3. Hacer *fine-tuning* y evaluar distintos modelos de lenguajes preentrenados en el *dataset* creado anteriormente, para generar modelos de clasificación de dominio.

4. Realizar un análisis de los resultados y determinar las ventajas y desventajas de cada modelo utilizado.

1.5 Estructura de la memoria

La memoria se encuentra estructurada siguiendo la siguiente definición de capítulos:

- **Capítulo 1** (actual): Introducción al tema tratado en el trabajo, así como la motivación y objetivos del mismo.
- **Capítulo 2**: Análisis del funcionamiento de varios modelos preentrenados, cómo funciona su arquitectura, y cómo utilizarlos en problemas de clasificación de textos, y en específico, al problema de clasificación de dominios.
- **Capítulo 3**: Análisis del estado del arte de la clasificación de texto, y la clasificación por dominios.
- **Capítulo 4**: Análisis detallado de la metodología utilizada para la creación del *dataset*.
- **Capítulo 5**: Experimentación realizada con el *dataset* creado y análisis de los resultados obtenidos.
- **Capítulo 6**: Presentación de las conclusiones obtenidas durante la realización del trabajo y propuesta de trabajo futuro.

CAPÍTULO 2
Modelos de Lenguaje Preentrenados

Los modelos de lenguaje preentrenados (MLP) son un tipo de modelo de aprendizaje profundo que se ha entrenado con grandes cantidades de datos de texto para aprender la estructura y los patrones de un idioma. Estos modelos han revolucionado el PLN y han supuesto mejoras en la traducción automática, el análisis de sentimientos y la generación de texto, etc.

Dada la importancia de los MLP y el objetivo de este trabajo de realizar un análisis de su funcionamiento en la tarea de clasificación en dominios, en la presente sección se realiza un breve análisis de la historia de estos modelos, sus ventajas y desventajas, los tipos de modelos que existen, sus arquitecturas y características más importantes.

2.1 Introducción a los MLP

Los modelos de lenguaje preentrenados han alcanzado un éxito sorprendente en el PLN, lo que ha llevado a un cambio de paradigma: del aprendizaje supervisado al preentrenamiento seguido de un *fine-tuning*. La comunidad del PLN ha sido testigo de una oleada de interés por investigar y mejorar los modelos preentrenados.

El concepto de preentrenamiento está relacionado con *transfer-learning*. La idea de *transfer-learning* [70] es reutilizar los conocimientos aprendidos en una o más tareas y aplicarlos a tareas nuevas. El aprendizaje de *transfer-learning* tradicional utiliza datos anotados para el entrenamiento supervisado, siendo la práctica más común durante al menos una década. Dentro del aprendizaje profundo, el preentrenamiento con aprendizaje *self-supervised* en datos masivos no anotados se ha convertido en el enfoque de *transfer-learning* dominante. La diferencia radica en que los métodos de preentrenamiento utilizan datos no anotados para el entrenamiento *self-supervised* y pueden aplicarse a varios procesos posteriores mediante *fine-tuning* o *few-shot learning*. [68]

La historia de los modelos lingüísticos preentrenados se remonta al desarrollo de los *word embeddings* y la posterior evolución de las técnicas de aprendizaje profundo aplicadas al PLN. La representación de palabras como vectores densos tiene una larga historia [20]. El *word embedding* "moderno" se introdujo en un trabajo acerca del modelo de lenguaje de red neuronal [6]. En [12] se demostró que los *word embeddings* preentrenados en datos no etiquetados puede mejorar significativamente muchas tareas de PLN. Este trabajo es el primer intento de obtener *word embeddings* genéricos útiles para otras tareas a partir de datos no etiquetados. En el artículo [44] se muestra que no es imprescindible utilizar redes neuronales profundas para construir buenos *word embeddings*. Proponen

dos arquitecturas: *continuous bag-of-words* (CBOW) y *skip-gram* (SG). A pesar de ser simples, estas arquitecturas logran aprender *word embeddings* de alta calidad para captar las similitudes sintácticas y semánticas latentes entre las palabras. *Word2vec* [43] es una de las implementaciones más populares de estos modelos y hace que los *word embeddings* preentrenados sean accesibles para diferentes tareas. *GloVe* [51] es también un modelo muy utilizado para obtener *word embeddings* preentrenados, que se calculan a partir de estadísticas globales de palabras de un gran corpus.

Aunque se ha demostrado que estas representaciones son eficaces en tareas de PLN, carecen de la capacidad de representar diferentes significados de las palabras en su contexto. Para resolver este problema, se propusieron modelos lingüísticos "conscientes" del contexto que incorporan toda la información contextual al procedimiento de entrenamiento, como [13] y [41]. Sin embargo, estos estudios utilizan una pequeña cantidad de datos para el preentrenamiento y no consiguen una mejora coherente del rendimiento en todas las tareas de PLN. No obstante, estos estudios motivaron en gran medida el seguimiento de los métodos de preentrenamiento para el modelado contextual.

En otro estudio pionero sobre modelos preentrenados, se propusieron *embeddings* de modelos de lenguaje para aprovechar las LSTM bidireccionales [61] con el fin de aprender representaciones contextuales de palabras, y desde entonces, han empezado a surgir numerosos MLPs dentro del paradigma de "preentrenamiento y luego *fine-tuning*". El preentrenamiento generativo (GPT) [55] fue el primer modelo que utilizó *Transformers* [65] unidireccionales como columna vertebral del GPT de modelos lingüísticos, ilustrando así el enorme potencial de los métodos de preentrenamiento para diversas tareas posteriores.

2.2 Tipos de MLP

Con el fin de aclarar las relaciones entre los modelos preentrenados existentes y comprender las diferencias entre ellos, analizaremos la taxonomía descrita en [54], utilizando 2 de los 4 aspectos mencionados: la arquitectura y el tipo de tarea utilizada en el preentrenamiento.

2.2.1. Arquitectura

Los modelos lingüísticos preentrenados basados en la arquitectura *Transformer* pueden clasificarse en modelos *encoder-only*, *decoder-only* o *encoder-decoder*, las cuales pueden observarse en la figura 2.1

Los modelos *encoder-only* [68] utilizan la parte *encoder* de la arquitectura *Transformer* para generar representaciones contextualizadas del texto de entrada. Suelen preentrenarse en tareas auto-supervisadas, como el modelado de lenguaje enmascarado (MLM), y se ajustan para tareas posteriores. Algunos ejemplos son BERT [17] y RoBERTa [39].

Los modelos *decoder-only* [68] utilizan la parte decodificadora del *Transformer* y suelen preentrenarse en tareas de modelado autor-regresivo del lenguaje. Generan texto de forma secuencial, prediciendo la palabra siguiente en función de las palabras generadas previamente. Algunos ejemplos son GPT, GPT-2 y GPT-3.

Los modelos *encoder-decoder* [68] combinan las partes codificadora y decodificadora de la arquitectura *Transformer*, lo que les permite procesar el texto de entrada y generar el texto de salida de forma secuencial. Pueden preentrenarse en tareas como la autocodificación con eliminación de ruido o el modelado secuencia a secuencia enmascarado, y perfeccionarse para tareas como la traducción, el resumen o sistemas de preguntas y respuestas. Algunos ejemplos son T5 y BART.

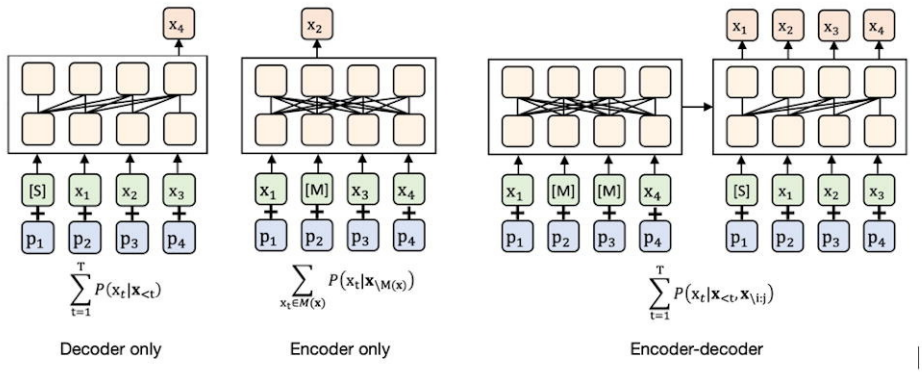


Figura 2.1: Una ilustración de las arquitecturas de preentrenamiento basadas en *Transformers* predominantes en la actualidad [68]

Cada categoría de modelos tiene sus puntos fuertes y es adecuada para distintos tipos de tareas de PLN. Los modelos *encoder-only* se utilizan generalmente para tareas que implican comprender o extraer información del texto, mientras que los modelos *decoder-only* se emplean más a menudo para tareas generativas. Los modelos *encoder-decoder* son adecuados para tareas que requieren tanto la comprensión como la generación de texto, a menudo de forma secuencial. [68]

2.2.2. Tareas de preentrenamiento

Las tareas de preentrenamiento [54] son cruciales para aprender la representación universal del lenguaje. Normalmente, estas tareas de preentrenamiento deben ser exigentes y disponer de datos de entrenamiento sustanciales, las cuales se dividen en aprendizaje supervisado, no supervisado y autosupervisado.

- Aprendizaje supervisado: consiste en aprender una función que asigna una entrada a una salida a partir de datos de pares de entrada-salida como entrenamiento.
- El aprendizaje no supervisado: consiste en encontrar algún conocimiento intrínseco de datos no etiquetados, como *clusters*, densidades y representaciones latentes.
- El aprendizaje auto-supervisado: es una mezcla de aprendizaje supervisado y aprendizaje no supervisado. Este paradigma es idéntico al del aprendizaje supervisado, pero las etiquetas de los datos de entrenamiento se generan automáticamente.

Muchas veces el aprendizaje no supervisado es muy difícil de diferenciar del autosupervisado, y la mayoría de los modelos preentrenados actuales utilizan tareas que caen en esta categoría, las cuales se describen a continuación.

La tarea no supervisada más común en PLN es el modelado del lenguaje probabilístico, o *Language Modeling* (LM) que es un problema probabilístico clásico de estimación de la densidad. El modelado lingüístico unidireccional consiste en predecir la siguiente palabra de una secuencia a partir de la palabra anterior. Uno de los inconvenientes de la LM unidireccional es que la representación de cada *token* codifica sólo los *tokens* del contexto hacia la izquierda y a sí mismo. Sin embargo, una mejor representación contextual debería codificar la información contextual de ambas direcciones. Una solución mejorada es el

LM bidireccional (BiLM), que consiste en dos LM unidireccionales: un LM de izquierda a derecha hacia adelante de izquierda a derecha y hacia atrás de derecha a izquierda.

Masked language modeling (MLM) es una variante al LM en la que el modelo predice las palabras enmascaradas de una frase, dado el contexto circundante. Un determinado porcentaje de palabras de la secuencia de entrada se sustituye por un *token* [MASK], y el modelo aprende a generar las palabras enmascaradas basándose en el contexto. Esta tarea de preentrenamiento permite a modelos como BERT aprender representaciones bidireccionales.

A pesar del amplio uso de la tarea MLM en el preentrenamiento, [75] afirmaba que algunos *tokens* especiales utilizados en el preentrenamiento de MLM, como [MASK], están ausentes cuando el modelo se aplica en tareas posteriores, lo que provoca una brecha entre el preentrenamiento y *fine-tuning*. Para superar este problema, *Permuted Language Modeling* (PLM) [75] es un objetivo de preentrenamiento que sustituye a MLM.

En PLM, el modelo aprende a predecir las palabras de una frase donde las palabras se encuentran en un orden aleatorio. Una permutación se muestrea aleatoriamente de entre todas las permutaciones posibles. Se selecciona una permutación de la secuencia de forma aleatoria. A continuación, de la secuencia permutada se eligen algunos *tokens* como objetivo, y el modelo se entrena para predecir estos objetivos, en función de el resto de los *tokens* y las posiciones naturales de los objetivos. Esta tarea permite al modelo captar dependencias de largo alcance en el texto sin estar restringido por el orden tradicional de izquierda a derecha o de derecha a izquierda. Un ejemplo de modelo entrenado mediante PLM es XLNet [76].

La tarea *Denoising autoencoder* (DAE) consiste en corromper la secuencia de entrada aplicando funciones de ruido como el enmascaramiento, la eliminación o la permutación y, a continuación, hacer que el modelo reconstruya la secuencia original. Esta tarea tiene el objetivo de lograr que el modelo aprenda representaciones robustas de la entrada. El modelo BART [35] es un ejemplo de modelo lingüístico preentrenado que utiliza un DA.

Las tareas de *Contrastive Learning* (CTL) implican aprender a discriminar entre entradas similares y disímiles. En el contexto de la PLN, se trata de predecir si dos frases son consecutivas en el texto original o si dos palabras pertenecen al mismo contexto. La idea que subyace al CTL es "aprender por comparación". En comparación con LM, CTL suele tener menos complejidad computacional y, por tanto, es un criterio de entrenamiento alternativo deseable. Existen varias técnicas de CTL, como *replaced token detection* (RTD), *next sentence prediction* (NSP), *sentence order prediction* (SOP), entre otras.

En la tarea RTD, se selecciona un determinado porcentaje de *tokens* en la secuencia de entrada. Para cada *token* seleccionado, el modelo generador lo sustituye por un nuevo *token* y el modelo discriminador toma la secuencia resultante e intenta predecir, para cada *token*, si ha sido sustituido o no. Un ejemplo de modelo preentrenado que utiliza esta técnica es ELECTRA [11].

En *Next sentence prediction* (NSP) se entrena el modelo para distinguir si dos frases introducidas son segmentos continuos del corpus de entrenamiento. En concreto, se elige un par de frases por cada ejemplo de preentrenamiento, el 50 % de las veces, la segunda frase es la frase siguiente real de la primera, y el 50 % de las veces es una frase aleatoria del corpus. De este modo, el modelo aprende la relación entre dos frases de entrada y beneficia así a las tareas posteriores que son sensibles a esta información.

Sentence Order Prediction (SOP) es una tarea de preentrenamiento utilizada para enseñar a los modelos lingüísticos la coherencia y el orden lógico de las frases de un texto. En la tarea SOP, el modelo se entrena para predecir si un par de frases aparece en su orden original o ha sido intercambiado. Durante el preentrenamiento, se presentan al modelo

pares de frases del corpus de entrenamiento. Para algunos pares, el orden se mantiene, mientras que para otros, el orden se invierte. A continuación, el modelo se entrena para predecir si el orden es correcto o invertido, basándose en la información contextual y la coherencia entre las frases. El SOP se utiliza como tarea auxiliar de preentrenamiento en algunos modelos lingüísticos, como ALBERT [31].

2.2.3. Taxonomía

Dada las características antes descritas respecto a la arquitectura y tipo de tareas de entrenamiento, así como el uso del contexto de las palabras, en la figura 2.2 se puede observar una taxonomía de los modelos de lenguaje preentrenados, basándose en la propuesta de [54] ligeramente modificada y reducida.

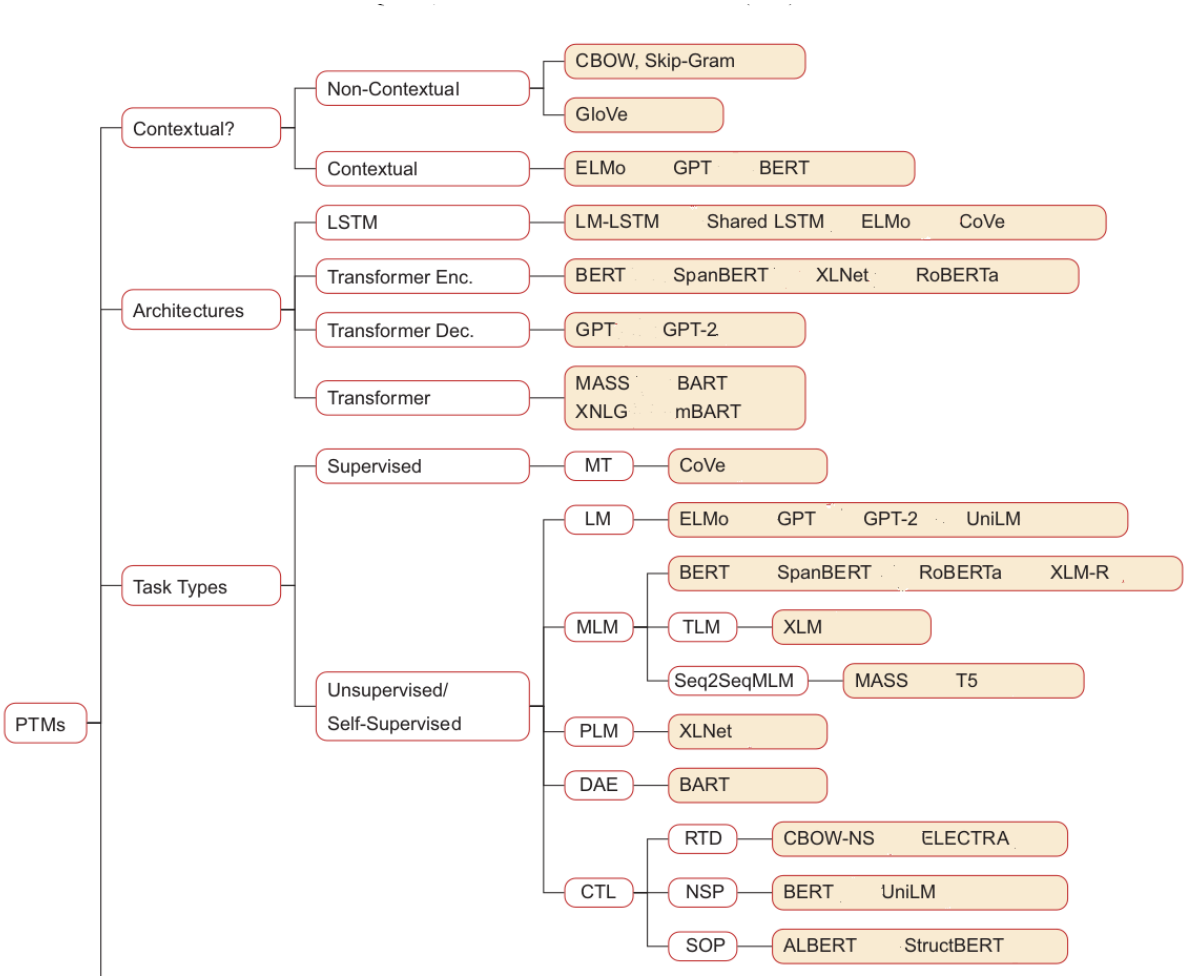


Figura 2.2: Taxonomía de los modelos de lenguaje preentrenados [54]

2.3 Transformer

Entender la arquitectura de *Transformer* [65] es importante para comprender los modelos lingüísticos preentrenados como GPT y BERT, ya que sirve de base para estos modelos, y muchos otros. GPT se centra en la parte del *decoder*, mientras que BERT utiliza la parte del *encoder* del *Transformer*, por lo que si se comprende la estructura y los componentes del *Transformer*, se podrá apreciar mejor las opciones de diseño y las adaptaciones

realizadas en estos modelos. Por lo tanto, a continuación se introduce brevemente cómo funcionan los *Transformer*.

El modelo *Transformer* es una arquitectura de red neuronal diseñada para tareas de secuencia a secuencia, como la traducción automática y el resumen de textos. Desde entonces, se ha convertido en la base de muchos modelos de PLN de última generación.

La arquitectura *Transformer* sigue una estructura *encoder-decoder*, pero no se basa en recurrencias y convoluciones para generar una salida. En la figura 2.3 se puede observar la arquitectura propuesta por el modelo *Transformer*.

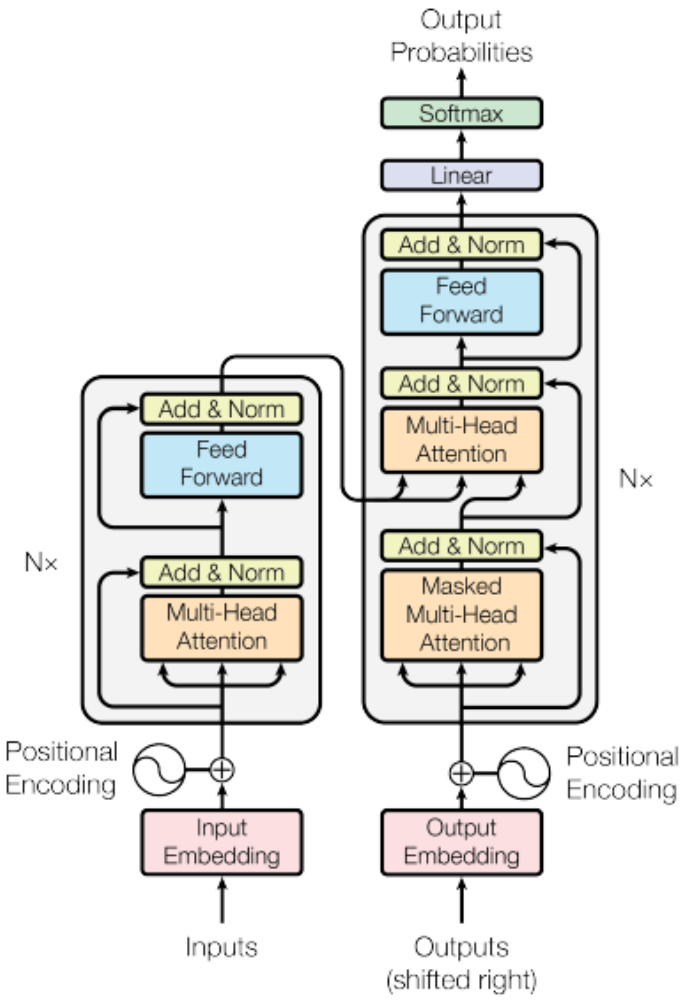


Figura 2.3: Arquitectura de los *Transformers* [65]

En pocas palabras, la tarea del *encoder*, situado en la mitad izquierda de la arquitectura *Transformer*, consiste en convertir una secuencia de entrada en una secuencia de representaciones continuas que, a continuación, se introducen en un *decoder*.

El *decoder*, en la mitad derecha de la arquitectura, recibe la salida del *encoder* junto con la salida del *decoder* en el paso de tiempo anterior para generar una secuencia de salida.

Cada una de estas piezas puede utilizarse de forma independiente, en función de la tarea. Los modelos *encoder-only* son buenos para tareas que requieren comprender la entrada, como la clasificación de frases y el reconocimiento de entidades nombradas. Los modelos *decoder-only* son adecuados para tareas generativas como la generación de texto. Los modelos *encoder-decoder* o modelos secuencia a secuencia son adecuados para tareas generativas que requieren una entrada, como la traducción o el resumen.

Una característica clave de los modelos *Transformer* es que se construyen con capas especiales denominadas capas de atención. Esta capa le dirá al modelo que preste atención específica a ciertos *tokens* (y que ignore más o menos las demás) cuando se ocupe de la representación de cada *token*. El mecanismo de *self-attention* del modelo *Transformer* permite que cada *token* de la secuencia de entrada atienda a todos los demás *tokens*, sopesando su importancia en función de las puntuaciones de atención aprendidas. *Multi-head self-attention* consiste esencialmente en varios mecanismos de *self-attention* que funcionan en paralelo, lo que permite al modelo captar distintos tipos de relaciones entre los *tokens*. Los resultados de todas las cabezas de atención se concatenan y se transforman linealmente para formar el resultado final.

El *encoder* consta de una pila de 6 capas idénticas, donde cada capa se compone de dos subcapas. La primera subcapa implementa un mecanismo de *multi-head self-attention*. La segunda subcapa es una red *feed-forward* completamente conectada que consiste en dos transformaciones lineales con activación (ReLU) en medio. Las seis capas del *encoder* aplican las mismas transformaciones lineales a todas las palabras de la secuencia de entrada, pero cada capa emplea distintos parámetros de peso y sesgo para hacerlo.

Además, cada una de estas dos subcapas tiene una conexión residual a su alrededor. A cada subcapa le sucede también una capa de normalización, que normaliza la suma calculada entre la entrada de la subcapa, y la salida generada por la propia subcapa. La arquitectura *Transformer* no puede captar de forma inherente ninguna información sobre las posiciones relativas de las palabras en la secuencia, ya que no hace uso de la recurrencia. Esta información tiene que inyectarse introduciendo *positional encodings* en los *embeddings* de entrada.

Los vectores de *positional encodings* tienen la misma dimensión que los *embeddings* de entrada y se generan utilizando funciones seno y coseno de distintas frecuencias. A continuación, se suman simplemente a los *embeddings* de entrada para inyectar la información posicional.

El *decoder* comparte varias similitudes con el *encoder*. También consiste en una pila de 6 capas idénticas compuestas cada una de tres subcapas.

La primera subcapa recibe la salida anterior de la pila del *decoder*, la aumenta con información posicional e implementa sobre ella el *multi-head self-attention*. Mientras que el *encoder* está diseñado para atender a todas las palabras de la secuencia de entrada independientemente de su posición en la secuencia, el *decoder* se modifica para atender sólo a las palabras precedentes. Por lo tanto, la predicción de una palabra en la posición sólo puede depender de los resultados conocidos de las palabras que la preceden en la secuencia. En el mecanismo de *multi-head self-attention*, esto se consigue introduciendo una máscara.

La segunda capa implementa un mecanismo de *multi-head self-attention* similar al implementado en la primera subcapa del *encoder*. La tercera capa implementa una red *feed-forward* totalmente conectada, similar a la implementada en la segunda subcapa del *encoder*.

La innovación clave del modelo *Transformer* se encuentra en los mecanismos de *self-attention*, que permiten al modelo captar eficazmente las dependencias de largo alcance y las relaciones complejas entre *tokens*.

2.4 Ejemplos de MLP

2.4.1. GPT

Los modelos *Generative Pre-trained Transformer* (GPT) de OpenAI han irrumpido con fuerza en la comunidad del PLN al introducir modelos lingüísticos muy potentes, lo que ha dado lugar a una amplia publicidad y reconocimiento. Estos modelos pueden realizar varias tareas de PLN, como sistemas de preguntas y respuestas, relacionar textos, resumir textos, entre otros, sin necesidad de un entrenamiento supervisado. Estos modelos lingüísticos necesitan muy pocos o ningún ejemplo para comprender las tareas y su rendimiento es equivalente o incluso mejor que el de los modelos más avanzados entrenados de forma supervisada.

GPT-1

La primera versión de GPT [55] se publicó en junio de 2018. Antes de este trabajo, la mayoría de los modelos de PLN más avanzados se entrenaban específicamente para una tarea concreta, como la clasificación de sentimientos o la vinculación textual, mediante aprendizaje supervisado. Sin embargo, los modelos supervisados tienen dos limitaciones principales:

- Necesitan una gran cantidad de datos anotados para aprender una tarea concreta, que a menudo no están fácilmente disponibles.
- No consiguen generalizar para tareas distintas de aquellas para las que han sido entrenados.

Este artículo [55] propone aprender un modelo de lenguaje generativo utilizando datos no etiquetados y, a continuación, hacer *fine-tuning* al modelo proporcionando ejemplos de tareas posteriores específicas como la clasificación, el análisis de sentimientos, la vinculación textual, entre otras.

Este aprendizaje semisupervisado (preentrenamiento no supervisado seguido de *fine-tuning* supervisado) para tareas de PLN tiene tres componentes fundamentales: modelización no supervisada del lenguaje, *fine-tuning* supervisado y transformaciones de entrada específicas de cada tarea.

Para el aprendizaje no supervisado, se utilizó el objetivo estándar del modelado lingüístico:

$$L_1(U) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta) \tag{2.1}$$

donde U es el conjunto de *tokens* en los datos no supervisados $(u_1, ..., u_n)$, k el tamaño de la ventana de contexto y Θ los parámetros de la red neuronal entrenada usando el descenso de gradiente estocástico.

El *fine-tuning* supervisado tiene como objetivo maximizar la probabilidad de observar la etiqueta y, dadas las características o *tokens* $x_1, ..., x_n$.

$$L_2(C) = \sum_{(x,y)} \log P(y | x_1, \ldots, x_n) \tag{2.2}$$

En lugar de limitarse a maximizar el objetivo mencionado en la ecuación 2.2, los autores añadieron un objetivo de aprendizaje auxiliar para el ajuste fino supervisado con el

fin de obtener una mejor generalización y una convergencia más rápida. El objetivo de entrenamiento modificado se estableció como:

$$L_3(C) = L_2(C) + \lambda \cdot L_1(C) \tag{2.3}$$

donde $L_1(C)$ es el objetivo auxiliar del modelo de aprendizaje del lenguaje y λ es el peso dado a este objetivo secundario de aprendizaje, con un valor fijo de 0,5. El *fine-tuning* supervisado se consiguió añadiendo una capa lineal y una capa *softmax* al modelo *Transformer* para obtener las etiquetas de tareas posteriores.

Para que los cambios en la arquitectura del modelo durante el *fine-tuning* fueran mínimos, las entradas de las tareas posteriores específicas se transformaron en secuencias ordenadas. Se añadieron *tokens* de inicio y fin a las secuencias de entrada y se añadió un *token* delimitador entre los segmentos para que la entrada pudiera enviarse como una secuencia ordenada.

GPT-1 utiliza una estructura de *Transformer* de sólo *decoder* con 12 capas con autoatención enmascarada para entrenar el modelo lingüístico. La arquitectura del modelo sigue siendo en gran medida la misma que la descrita en el trabajo original sobre *Transformers*. Para el entrenamiento, GPT-1 utilizó el conjunto de datos BooksCorpus para entrenar el modelo lingüístico. BooksCorpus tenía unos 7.000 libros no publicados que ayudaron a entrenar el modelo lingüístico con datos no vistos. Se utilizó el vocabulario *Byte Pair Encoding* (BPE) con 40.000 fusiones y el modelo tenía 117 millones de parámetros en total. La arquitectura de GPT-1 y las tareas utilizadas para el *fine-tuning* pueden observarse más claramente en la figura 2.4

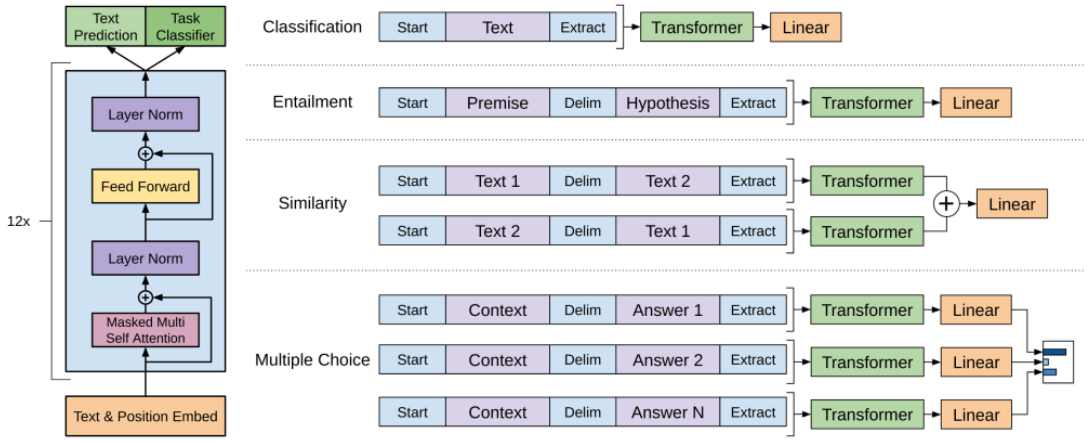


Figura 2.4: Arquitectura del *Transformer* y objetivos de entrenamiento utilizados en GPT-1 Fuente: [55]

GPT-1 obtuvo mejores resultados que los modelos supervisados de última generación entrenados específicamente en 9 de las 12 tareas en las que se compararon los modelos en el momento de su publicación. Su mayor logro fue que demostró que el modelo lingüístico servía como un objetivo de preentrenamiento eficaz que podía ayudar al modelo a generalizar bien. La arquitectura facilitaba el *transfer learning* y podía realizar varias tareas de PLN con muy poco *fine-tuning*. Este modelo demostró la potencia del preentrenamiento generativo y abrió paso para otros modelos que podrían mejorar este potencial con conjuntos de datos más grandes y más parámetros.

GPT-2

Los avances en el modelo GPT-2 [56], publicado en 2019, consistieron principalmente en utilizar un conjunto de datos más amplio y añadir más parámetros al modelo para aprender un modelo lingüístico aún más sólido.

Para lograr esto modificaron el objetivo de entrenamiento del modelo lingüístico de $P(salida|entrada)$ a $P(salida|entrada, tarea)$, con la idea de aprender múltiples tareas utilizando el mismo modelo no supervisado. Esta modificación se conoce como *task conditioning*, en la que se espera que el modelo produzca una salida diferente para la misma entrada y diferentes tareas. En los modelos de lenguaje, la salida, la entrada y la tarea son secuencias de lenguaje natural. Por lo tanto, el *task conditioning* para los modelos lingüísticos se realiza proporcionando ejemplos o instrucciones en lenguaje natural al modelo para que realice una tarea específica. El *task conditioning* constituye la base de la transferencia de tareas *zero shot*, la cual es una capacidad interesante de GPT-2. Se trata de un caso especial en el que no se proporcionan ejemplos y el modelo entiende la tarea basándose en las instrucciones dadas. En lugar de reorganizar las secuencias, como se hizo en GPT-1 para el ajuste fino, la entrada a GPT-2 se dio en un formato que esperaba que el modelo entendiera la naturaleza de la tarea y proporcionara respuestas. Por ejemplo, para la tarea de traducción del inglés al francés, el modelo recibía una frase en inglés seguida de la palabra francés y un *prompt* (:). El modelo debía entender que se trataba de una tarea de traducción y dar el equivalente en francés de la frase en inglés.

Para crear un conjunto de datos amplio y de buena calidad, los autores utilizaron la plataforma *Reddit* y extrajeron datos de los enlaces de los artículos más votados. El conjunto de datos resultante, denominado *WebText*, contenía 40GB de datos de texto de más de 8 millones de documentos. Este conjunto de datos se utilizó para entrenar GPT-2 y era enorme en comparación con el conjunto de datos *BookCorpus* utilizado para entrenar el modelo GPT-1.

GPT-2 tiene 1.500 millones de parámetros, 10 veces más que GPT-1 (117 millones de parámetros). Entre las principales diferencias con GPT-1 están:

- GPT-2 tiene 48 capas y utiliza vectores de 1600 dimensiones para los *word embeddings*.
- Vocabulario más amplio de 50.257 *tokens*.
- Se utiliza un tamaño de *batch* mayor y una ventana de contexto mayor.

El documento afirmaba que con el aumento de la capacidad del modelo, el rendimiento aumentaba de forma log-lineal. Además, la caída de la perplejidad de los modelos lingüísticos no mostró saturación y siguió disminuyendo con el aumento del número de parámetros. De hecho, GPT-2 no se ajustaba lo suficiente al conjunto de datos WebText y un entrenamiento más prolongado podría haber reducido aún más la perplejidad. Esto demostró que el tamaño del modelo de GPT-2 no era el límite y que construir modelos lingüísticos aún mayores reduciría la perplejidad y mejoraría la comprensión del lenguaje natural.

GPT-3

Con el objetivo de construir modelos lingüísticos muy potentes y sólidos que no necesitaran ningún *fine-tuning* y sólo unas pocas demostraciones para entender las tareas y realizarlas, OpenAI construyó el modelo GPT-3 [7] con 175.000 millones de parámetros. Este modelo, liberado en mayo de 2020, tiene 100 veces más parámetros que GPT-2.

Gracias al gran número de parámetros y al extenso conjunto de datos en el que se ha entrenado GPT-3, ofrece buenos resultados en tareas de PLN en entornos de *zero shot* y *few shot*. Un ejemplo de *few shot*, *zero shot* y *one shot* se puede observar en la figura 2.5

Figura 2.5: Ejemplo de *zero-shot*, *one-shot* y *few-shot* en GPT-3 Fuente: [7]

Gracias a su gran capacidad, puede escribir artículos difíciles de distinguir de los escritos por humanos. También puede realizar sobre la marcha tareas para las que nunca se le ha entrenado explícitamente, como sumar números, escribir consultas y códigos SQL, descifrar palabras en una frase, escribir códigos React y JavaScript a partir de una descripción en lenguaje natural de la tarea, etc.

Los grandes modelos lingüísticos desarrollan el reconocimiento de patrones y otras habilidades utilizando los datos de texto con los que se entrenan. Mientras aprenden el objetivo principal de predecir la palabra siguiente a partir de las palabras del contexto, los modelos lingüísticos también empiezan a reconocer patrones en los datos que les ayudan a minimizar la pérdida en la tarea de modelado lingüístico. Más adelante, esta

capacidad ayuda al modelo durante la tarea *zero-shot*. Cuando se le presentan unos pocos ejemplos y/o una descripción de lo que tiene que hacer, el modelo lingüístico compara el patrón de los ejemplos con lo que ha aprendido en el pasado para datos similares y utiliza ese conocimiento para realizar las tareas. Se trata de una potente capacidad de los grandes modelos lingüísticos que aumenta con el incremento del número de parámetros del modelo, llamada *in-context learning*.

GPT-3 se entrenó con una mezcla de cinco corpus diferentes, a cada uno de los cuales se le asignó un peso determinado. Los conjuntos de datos de alta calidad se muestrearon con más frecuencia y el modelo se entrenó durante más de una época. Los cinco conjuntos de datos utilizados fueron *Common Crawl, WebText2, Books1, Books2* y *Wikipedia*.

La arquitectura de GPT-3 es la misma que la de GPT-2. Las principales diferencias con GPT-2 son:

- GPT-3 tiene 96 capas y cada capa tiene 96 cabezas de atención.
- El tamaño de los *word embeddings* se aumentó de 1.600 a 12.888 respecto a GPT-2.
- El tamaño de la ventana de contexto se aumentó de 1.024 en GPT-2 a 2.048 *tokens* en GPT-3.

Aunque GPT-3 es capaz de producir textos de gran calidad, a veces empieza a perder coherencia al formular frases largas y repite secuencias de texto una y otra vez. Además, GPT-3 no rinde muy bien en tareas como la inferencia en lenguaje natural (determinar si una frase implica otra frase), rellenar los espacios en blanco, algunas tareas de comprensión lectora, etc. El artículo cita la unidireccionalidad de los modelos GPT como causa probable de estas limitaciones y sugiere entrenar modelos bidireccionales a esta escala para superar estos problemas.

Además de estas limitaciones, GPT-3 conlleva el riesgo potencial de que se haga un uso indebido de su capacidad para generar texto similar al humano con fines de spam, difusión de información errónea u otras actividades fraudulentas. Además, el texto generado por GPT-3 posee los sesgos propios de la lengua en la que ha sido entrenado. Los artículos generados por GPT-3 pueden tener sesgos de género, etnia, raza o religión. Por eso es tan importante utilizar estos modelos con cuidado y controlar el texto que generan antes de utilizarlo.

Otras limitaciones de GPT-3 incluyen la compleja y costosa inferencia del modelo debido a su pesada arquitectura, la menor interpretabilidad del lenguaje y los resultados generados por el modelo y la incertidumbre en torno a lo que ayuda al modelo a lograr su comportamiento *zero shot*.

GPT-3.5 y GPT-4

Utilizando GPT-3 como modelo base, los modelos GPT-3.5 [78] contienen ajustes adicionales. Esta fase de ajuste añade al modelo GPT-3 un concepto denominado *Reinforcement Learning with human feedback* (RLHF) [48].

RLHF [48] es un subcampo de la IA que se centra en el uso de la retroalimentación humana para mejorar los algoritmos de aprendizaje automático. En RLHF, un humano proporciona información al algoritmo de aprendizaje automático, que se utiliza para ajustar el comportamiento del modelo. Este enfoque se utiliza para abordar las limitaciones del aprendizaje supervisado y no supervisado, en el que los algoritmos de aprendizaje automático tienen una capacidad limitada para aprender a partir de datos etiquetados o no etiquetados.

La retroalimentación humana [27] puede proporcionarse de varias formas, como recompensando o castigando las acciones del modelo, proporcionando etiquetas para datos no etiquetados o ajustando los parámetros del modelo. El objetivo de las RLHF es incorporar la experiencia y los conocimientos humanos a los algoritmos de aprendizaje automático para mejorar su rendimiento y su capacidad para resolver tareas complejas.

Con GPT-3.5 especializado en conversaciones, lanzado en diciembre de 2022, OpenAI obtuvo un millón de usuarios en cinco días y a 100 millones en dos meses a través de una interfaz visual web llamada ChatGPT.

John Schulman, de OpenAI, desarrolló la plataforma ChatGPT, y su popularidad ha sido sorprendente. A pesar de la disponibilidad de los potentes modelos *GPT-3 davinci* y *text-davinci-003*, ChatGPT ofrece una interfaz intuitiva para que los usuarios mantengan una conversación con la IA. Aunque la funcionalidad de ChatGPT no es nueva, la interfaz pública es nueva e innovadora. Los casos de uso de ChatGPT incluyen la creación de contenidos digitales, la escritura y depuración de código y la respuesta a consultas de atención al cliente.

Luego de esto, el mundo estaba esperando la versión actualizada de la familia GPT que es GPT-4 [47], y no decepcionó cuando la hizo pública en marzo de 2023. Anunciaron GPT-4 como un modelo más avanzado que su predecesor, GPT-3.5. Lograron mejoras que permiten al modelo comprender mejor el contexto y distinguir los matices, lo que se traduce en respuestas más precisas y coherentes. Además, GPT-4 tiene un límite máximo de 32.000 *tokens* (equivalentes a 25.000 palabras), lo que supone un aumento significativo respecto a los 4.000 *tokens* de GPT-3.5 (equivalentes a 3.125 palabras).

Aunque GPT-3.5 es bastante capaz de generar texto similar al humano, GPT-4 tiene una capacidad aún mayor para entender y generar diferentes dialectos y responder a las emociones expresadas en el texto. GPT-4 demuestra una gran capacidad para resolver problemas matemáticos y científicos complejos más allá de las capacidades de GPT-3.5. Estas capacidades han sido reportadas por OpenAi en [47], pero también por investigadores, como en [8] y [29]

Con la liberación de ChatGPT, que en la versión *plus* se puede seleccionar qué modelo base utilizar, ya sea GPT-3.5 o GPT-4, y la liberación de la API de GPT-3.5 los experimentos y casos de uso de esta herramienta están por todo Internet, logrando que personas de todas las especialidades se involucren y aparezcan casos de uso y capacidades cada vez más interesantes. Esto quiere decir que queda un largo camino por delante para explorar y comprender mejor el alcance de estos modelos.

2.4.2. BERT

BERT (*Bidirectional Encoder Representations from Transformers*) [17] es un potente modelo lingüístico diseñado para aprender representaciones contextualizadas del texto aprovechando el *encoder* de la arquitectura *Transformer*. Se presentó en octubre de 2018 y fue desarrollado por investigadores de *Google AI Language*.

La naturaleza bidireccional de BERT le permite captar la información contextual de los contextos izquierdo y derecho del texto de entrada, lo que conduce a una comprensión más precisa de la semántica y la sintaxis del lenguaje. Este contexto bidireccional se consigue mediante el uso del mecanismo de *self-attention* en el *encoder* del *Transformer*, que considera simultáneamente todos los *tokens* de la secuencia de entrada. La diferencia de la arquitectura de BERT respecto a otras como GPT y ELMO [52] se puede observar en la figura 2.6.

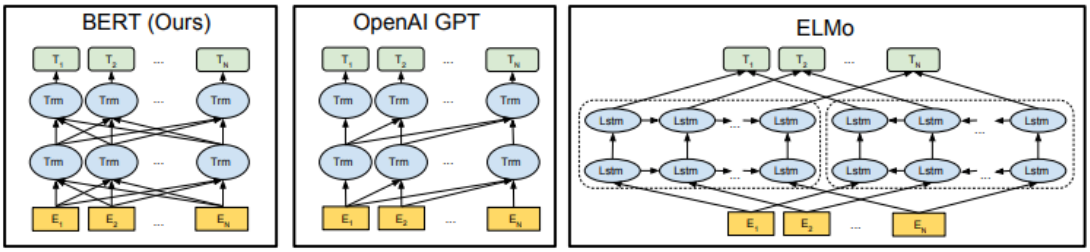


Figura 2.6: Diferencias en las arquitecturas de los modelos de preentrenamiento. BERT utiliza un *Transformer* bidireccional. OpenAI GPT utiliza un *Transformer* de izquierda a derecha. ELMo utiliza la concatenación de LSTMs entrenados independientemente de izquierda a derecha y de derecha a izquierda para generar características para tareas posteriores. [17]

El proceso de tokenización en BERT es un paso crucial para preparar los datos de texto antes de introducirlos en el modelo. BERT utiliza la tokenización WordPiece [73], que es un algoritmo de tokenización de subpalabras que equilibra la tokenización a nivel de carácter y a nivel de palabra, utilizando un vocabulario de 30,000 *tokens*. Para representar las subpalabras se utiliza el prefijo "##" para indicar que son fragmentos de una palabra completa. Adicionalmente, BERT utiliza *tokens* especiales para transmitir información específica al modelo:

- [CLS]: Este *token* se añade al principio de la secuencia y se utiliza para tareas de clasificación como representación agregada de la entrada.
- [SEP]: Este *token* se inserta entre frases o al final de una sola frase para indicar los límites entre frases.
- [MASK]: Este *token* se utiliza durante el proceso de preentrenamiento para la tarea de MLM, sustituyendo un determinado porcentaje de *tokens* de entrada que el modelo está entrenado para predecir.
- [PAD]: BERT requiere secuencias de entrada de longitud fija. Para conseguirlo, las secuencias tokenizadas se truncan hasta una longitud máxima especificada o se rellenan con un *token* especial [PAD] para ajustarse a la longitud deseada.

Para asistir en las tareas de entrenamiento se añaden los siguientes identificadores a los tokens:

- Token IDs: Tras el proceso de tokenización, cada *token* se asigna a un ID único del vocabulario WordPiece
- Segment IDs: En las tareas con varias frases, BERT utiliza identificadores de segmento para diferenciarlas. Normalmente, a todos los *tokens* de la primera frase se les asigna un ID de segmento 0, y a todos los *tokens* de la segunda frase se les asigna un ID de segmento 1.
- Position IDs: BERT también utiliza codificaciones posicionales para representar la posición de cada *token* en la secuencia de entrada, lo que ayuda al modelo a aprender el orden de las palabras en la frase.

En la figura 2.7 se puede observar la representación de BERT de la entrada teniendo en cuenta lo explicado anteriormente.

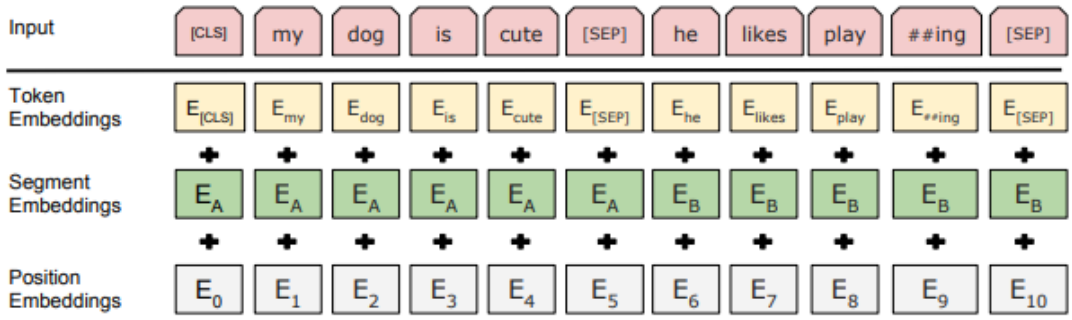


Figura 2.7: Representación de entrada BERT. Los *input embeddings* son la suma de los *token embeddings*, los *segmentation embeddings* y los *position embeddings*. [17]

BERT se entrena previamente con datos de texto sin etiquetar a gran escala, mediante dos tareas no supervisadas: *Masked Language Modeling* (MLM) y *Next Sentence Prediction* (NSP).

La tarea MLM, como se describe en secciones anteriores, consiste en enmascarar aleatoriamente un porcentaje de los *tokens* de entrada y entrenar el modelo para predecir los *tokens* enmascarados basándose en el contexto que los rodea. El proceso de enmascaramiento en BERT sigue las siguientes reglas:

- Predicción del 15 % de los *tokens* de la entrada, elegidos al azar.
- El 80 % de los *tokens* seleccionadas se sustituyen por el *token* [MASK].
- El 10 % de los *tokens* seleccionadas se sustituyen por un *token* aleatorio.
- El 10 % de los *tokens* seleccionados no se modifican.

En el proceso de entrenamiento de BERT para la tarea NSP, el modelo recibe pares de frases como entrada y aprende a predecir si la segunda frase del par es la frase posterior en el documento original. Durante el entrenamiento, el 50 % de las entradas son pares en los que la segunda frase es la frase posterior del documento original, mientras que en el otro 50 % se elige una frase aleatoria del corpus como segunda frase. Se supone que la frase aleatoria estará desconectada de la primera. BERT se entrena utilizando el optimizador Adam con un *learning rate* de $1e-4$, *batch size* de 256 y una longitud de secuencia de 128 o 512 *tokens*, realizando 1 millón de pasos.

Existen dos versiones principales del modelo BERT en función de su tamaño y del número de parámetros:

- **BERT-Base:** Esta versión consta de 12 capas (bloques *transformer*), 768 *hidden units* y 12 *self-attention heads* con un total de unos 110 millones de parámetros.
- **BERT-Large:** Esta versión tiene una arquitectura más grande, con 24 capas (bloques *transformer*), 1.024 *hidden units* y 16 *self-attention heads*, lo que suma unos 340 millones de parámetros.

En el momento de su publicación, BERT-Large mejoró los resultados del estado del arte en la prueba de referencia GLUE [66], incluidas las tareas CoLA [69] y SST-2 [41] . En CoLA, BERT-Large logró una correlación Matthews de 60,5, y en SST-2, un *accuracy* de 94,9. SST-2 El *Stanford Sentiment Treebank* es una tarea de clasificación binaria de una sola frase que consiste en frases extraídas de críticas de películas con anotaciones humanas

de su sentimiento CoLA El Corpus de Aceptabilidad Lingüística es una tarea de clasificación binaria de una sola frase, en la que el objetivo es predecir si una frase en inglés es lingüísticamente"aceptable" o no. Estaremos utilizando estos 2 *datasets* como referencia en el análisis de los próximos modelos por motivos de comparación, y debido a que se tratan de tareas de clasificación de texto, y que se alinea con el tipo de tarea a resolver en este trabajo.

2.4.3. RoBERTa

El modelo RoBERTa [39] es un modelo lingüístico basado en redes neuronales que fue presentado en 2019 por el equipo del actual Meta AI Research. Es una extensión del popular modelo BERT y se basa en la arquitectura *Transformer*. EL nombre de RoBERTa significa *Robustly optimized BERT approach*.

Una diferencia clave entre RoBERTa y BERT es que RoBERTa se entrenó con un conjunto de datos mucho mayor y utilizando un procedimiento de entrenamiento más eficaz. En concreto, RoBERTa se entrenó con un conjunto de datos de 160GB de texto, que es más de 10 veces mayor que el conjunto de datos utilizado para entrenar BERT. La tokenización de RoBERTa es basada en *bytes* en lugar de la basada en caracteres de BERT, obteniéndose un vocabulario de 50.000 *tokens*.

A continuación se indican los *datasets* utilizados para entrenar el modelo RoBERTa:

- BOOK CORPUS y el conjunto de datos de Wikipedia en inglés: Estos datos también se utilizaron para entrenar la arquitectura BERT; contienen 16GB de texto.
- CC-NEWS [46]. Estos datos contienen 63 millones de artículos de noticias en inglés entre septiembre de 2016 y febrero de 2019. El tamaño de este conjunto de datos es de 76GB después del filtrado.
- OPENWEBTEXT [18]: Este conjunto de datos contiene contenido web extraído de las URL compartidas en Reddit con al menos 3 votos favorables. El tamaño de este conjunto de datos es de 38GB.
- STORIES [64]: Estos datos contienen un subconjunto de datos de Common Crawl filtrados para ajustarse al estilo de historias de la tarea Winograd NLP. Contiene 31GB de texto.

RoBERTa tiene una arquitectura casi similar a la de BERT, pero para mejorar los resultados de la arquitectura de BERT, los autores realizaron algunos cambios sencillos de diseño en su arquitectura y procedimiento de entrenamiento. Estos cambios son:

- Eliminación de Next Sentence Prediction (NSP): Los autores experimentaron con la eliminación/adición de la pérdida NSP a diferentes versiones y llegaron a la conclusión de que la eliminación de la pérdida NSP iguala o mejora ligeramente el rendimiento en en entrenamiento de tareas posteriores.
- Entrenamiento con *batch sizes* y secuencias más largas: Originalmente, BERT se entrena para 1M de pasos con un tamaño de *batch* de 256 secuencias. En este trabajo, los autores entrenaron el modelo con 125 pasos de 2K secuencias y 31K pasos con *batch* de 8k secuencias. Esto tiene dos ventajas: los *batch* grandes mejoran la perplejidad en el objetivo de MLM y también la precisión final de la tarea. Los *batches* grandes también son más fáciles de paralelizar mediante el entrenamiento distribuido.

- Cambio dinámico del patrón de enmascaramiento: En la arquitectura BERT, el enmascaramiento se realiza una vez durante el preprocesamiento de datos, lo que da como resultado una única máscara estática. Para evitar el uso de una única máscara estática, los datos de entrenamiento se duplican y enmascaran 10 veces, cada vez con una estrategia de máscara diferente a lo largo de 40 *epochs*, teniendo así 4 *epochs* con la misma máscara. Esta estrategia se compara con el enmascaramiento dinámico en el que se genera un enmascaramiento diferente cada vez que se le pasa datos al modelo.

En la tarea CoLA (*Corpus of Linguistic Acceptability*), RoBERTa obtuvo una puntuación de 68, superando con creces el estado del arte anterior en el momento. En la tarea SST-2 (*Stanford Sentiment Treebank*), RoBERTa obtuvo una puntuación de 96,4, superando de nuevo el estado del arte.

2.4.4. XLNet

Poco después de que BERT arrasara en la comunidad de PLN, investigadores de la Universidad Carnegie Mellon y el equipo *Google AI Brain* presentaron XLNet [76] en una ponencia de la conferencia NeurIPS 2019, dejando una gran impresión en la comunidad de PLN.

XLNet aprovecha lo mejor del modelado autorregresivo (AR) del lenguaje y de la autocodificación (AE), los dos objetivos de preentrenamiento más conocidos, al tiempo que evita sus limitaciones. Los modelos AR tradicionales son unidireccionales. Y los modelos de autocodificación como el BERT tienen varios inconvenientes: utiliza el *token* [MASK] en el preentrenamiento, pero este tipo de símbolos artificiales no aparecen en los datos reales en el momento del ajuste, lo que provoca una discrepancia entre el preentrenamiento y el *fine-tuning*. Otra desventaja de [MASK] es que asume que los *tokens* enmascarados son independientes entre sí dados los *tokens* no enmascarados.

Considerado como uno de los avances más importantes de 2019 en PLN, XLNet combina el modelo autorregresivo del lenguaje, Transformer-XL [14], y la capacidad bidireccional de BERT.

Los autores de XLNet proponen conservar las ventajas del modelo de lenguaje autorregresivo y, al mismo tiempo, hacer que aprenda del contexto bidireccional como los modelos AE durante la fase de preentrenamiento. La interdependencia entre *tokens* se conservará, a diferencia de lo que ocurre en BERT.El nuevo objetivo propuesto se denomina *Permutation Language Modeling*, concepto discutido en secciones anteriores. El método de permutación consiste en obtener permutaciones de una secuencia, y utilizando los *t-1 tokens* anteriores como contexto para predecir el *token* de la posición *t-ésima*. Otro aporte que XLNet propone es el concepto de *Two-Stream Self-Attention*. Una es la *content stream*, la capa de *self-attention* estándar en la arquitectura de *Transformer*. Otra es la *query stream*, que XLNet introduce para sustituir al *token* [MASK] en BERT.

Siguiendo a BERT, utilizan el BooksCorpus y la Wikipedia en inglés como parte de los datos de preentrenamiento. Además, incluyen Giga5 (16GB de texto), ClueWeb 2012-B y Common Crawl para el preentrenamiento. Utilizaron heurísticas para filtrar de forma agresiva los artículos cortos o de baja calidad para ClueWeb 2012-B y Common Crawl, obteniendo textos de 19GB y 110GB, respectivamente. Para la tokenización fue utilizado el método SentencePiece [30]. Al igual que BERT utilizan los *tokens* especiales de [CLS] y [SEP] en las tareas de preentrenamiento.

En la tarea CoLA (*Corpus of Linguistic Acceptability*), XLNet obtuvo una puntuación de 69. En la tarea SST-2 (*Stanford Sentiment Treebank*), obtuvo una puntuación de 97. En ambas superando a modelos como BERT y RoBERTa.

2.4.5. ALBERT

ALBERT [31](*A Lite* BERT) es un modelo de PLN desarrollado por investigadores de Google Research y el *Toyota Technological Institute at Chicago*. Se presentó por primera vez en un artículo titulado *"ALBERT: A Lite BERT for Self-supervised Learning of Language Representations"* publicado en 2019.

ALBERT es una versión modificada del modelo BERT. Se basa en tres puntos clave como son la Compartición de Parámetros, la Factorización de *Embedding* y *Sentence Order Prediction* (SOP).

Cuando se lanzó, BERT obtuvo resultados punteros en muchas tareas de PLN en tablas de clasificación. Sin embargo, el modelo era muy grande, lo que dio lugar a algunos problemas. Uno de ellos es la limitación de memoria y sobrecarga de comunicación. BERT-Large, al ser un modelo complejo, tiene 340 millones de parámetros debido a sus 24 capas ocultas y muchos nodos en la red *feed-forward* y las cabezas de atención. Si se quisiera aprovechar el trabajo sobre BERT y aportarle mejoras, se necesitarían grandes requisitos de computación para entrenar desde cero e iterar sobre él.

Un enfoque popular para este problema es el entrenamiento distribuido, pero el gran número de parámetros que es necesario transferir durante la sincronización de gradientes puede ralentizar el proceso de entrenamiento. El mismo cuello de botella se aplica al paralelismo del modelo, en el que se almacenan diferentes partes del modelo (parámetros) en diferentes máquinas.

Para solucionar este problema ALBERT propuso 2 ideas para que el modelo fuera más ligero: la Compartición de Parámetros y la Factorización de *Embedding*. BERT-Large tiene 24 capas, mientras que su versión base tiene 12 capas. Cuando se añaden más capas, aumenta la cantidad de parámetros de forma exponencial. Para resolver este problema, ALBERT utiliza el concepto de compartición de parámetros entre capas. Para ilustrarlo se puede observar el ejemplo de un modelo BERT-Base de 12 capas en la figura 2.8. En lugar de aprender parámetros únicos para cada una de las 12 capas, sólo aprende parámetros para el primer bloque y reutilizan el bloque en las 11 capas restantes.

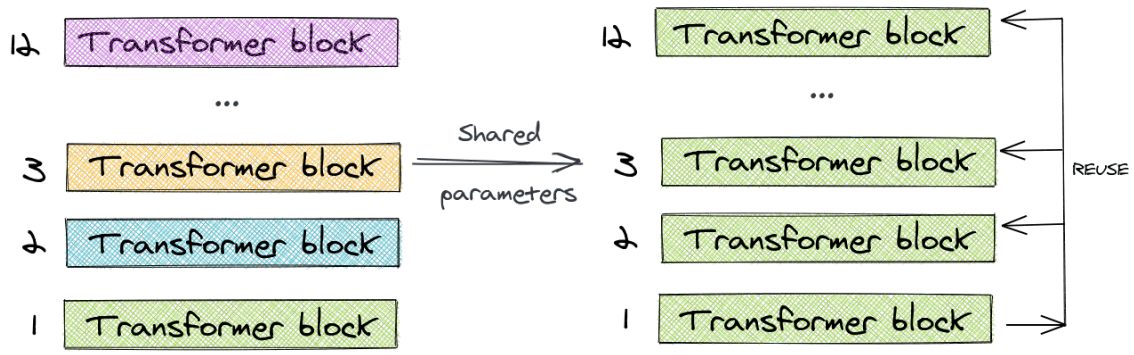


Figura 2.8: Ejemplo que ilustra la compartición de parámetros en ALBERT obtenido de [9]

Se pueden compartir parámetros sólo para la capa *feed-forward*, sólo para los parámetros de atención o compartir los parámetros de todo el bloque en sí, en ALBERT decidieron compartir todo el bloque. En comparación con los 110 millones de parámetros de

BERT-Base, el modelo ALBERT sólo tiene 31 millones de parámetros y utiliza el mismo número de capas y 768 *hidden units*.

En BERT, así como en posteriores mejoras de modelado como XLNet y RoBERTa, el tamaño del *embedding* (E) de WordPiece está vinculado al tamaño de la capa oculta (H). Desde un punto de vista práctico, el procesamiento del lenguaje natural suele requerir que el tamaño del vocabulario V sea grande. Si $E \equiv H$, al aumentar H aumenta el tamaño de la matriz de *embeddings*, que tiene tamaño $V \times E$. Esto puede dar lugar fácilmente a un modelo con miles de millones de parámetros, la mayoría de los cuales sólo se actualizan escasamente durante el entrenamiento. Por lo tanto, para ALBERT se utiliza una factorización de los parámetros de *embedding*, descomponiéndolos en dos matrices más pequeñas. En lugar de proyectar los vectores unidimensionales directamente en el espacio oculto de tamaño H, primero se proyectan en un espacio de *embedding* de menor dimensión de tamaño E, y luego se proyecta al espacio oculto. Mediante esta descomposición, se reduce $O(V \times H)$ a $O(V \times E + E \times H)$. Esta reducción de parámetros es significativa cuando $H \gg E$.

Gracias a estas 2 ideas. ALBERT logra 18 veces menos parámetros que BERT-Large y un entrenamiento 1,7 veces más rápido.

BERT introdujo una pérdida de clasificación binaria denominada *"Next Sentence Prediction"*. Se creó específicamente para mejorar el rendimiento en tareas posteriores que utilizan pares de frases, como la "Inferencia del lenguaje natural". Trabajos como RoBERTa y XLNet han arrojado luz sobre la ineficacia de NSP y han descubierto que su impacto en las tareas posteriores es poco fiable. Por ello, ALBERT propone una tarea alternativa denominada *Sentence Order Prediction*, analizada en secciones anteriores. La idea clave es la siguiente.

- Tomar dos segmentos consecutivos del mismo documento como clase positiva.
- Intercambiar el orden del mismo segmento y utilizarlo como ejemplo negativo.

Esto obliga al modelo a aprender una distinción más fina sobre las propiedades de coherencia a nivel de discurso.

Teniendo en cuenta las diferentes versiones de ALBERT, en la figura 2.9 se puede observar las diferencias entre ALBERT y BERT respecto a la cantidad de parámetros, tiempo de procesamiento y los resultados obtenidos en los principales *datasets* de referencia.

Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg	Speedup
BERT	base	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3	4.7x
	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	1.0
ALBERT	base	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1	5.6x
	large	18M	90.6/83.9	82.3/79.4	83.5	91.7	68.5	82.4	1.7x
	xlarge	60M	92.5/86.1	86.1/83.1	86.4	92.4	74.8	85.5	0.6x
	xxlarge	235M	**94.1/88.3**	**88.1/85.1**	**88.0**	**95.2**	**82.3**	**88.7**	0.3x

Figura 2.9: ALBERT vs BERT. Fuente: [9]

En la tarea CoLA (*Corpus of Linguistic Acceptability*), ALBERT obtuvo una puntuación de 69,1. En la tarea SST-2 (*Stanford Sentiment Treebank*), obtuvo una puntuación de 97,1.

2.4.6. BART

BART [35] (*Bidirectional and Auto-Regressive Transformers*) es un modelo de PLN desarrollado por investigadores de Facebook AI. Fue presentado en octubre de 2019 a través

de un documento de investigación titulado *"BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension."*

BART es fundamentalmente casi idéntica a la arquitectura estándar *sequence-to-sequence* de *Transformers*, con algunas modificaciones. Utiliza GeLU en lugar de ReLU. La versión base tiene 6 capas para el *encoder* y 6 capas para el *decoder*, mientras que la versión grande tiene 12 capas para el *encoder* y 12 capas para el *decoder*. Cada capa del *decoder* realiza además *cross-attention* con la salida de la capa oculta final de los *encoders*, y no tiene red *feed-forward* al final.

Utiliza una arquitectura neuronal como la de traducción automática que, a pesar de su sencillez, puede considerarse una generalización de BERT (gracias al *encoder* bidireccional) y de GPT (con el *decoder* de izquierda a derecha).

Aunque la arquitectura del modelo es bastante sencilla, la contribución clave de este trabajo es una elaborada experimentación sobre las distintas tareas de preentrenamiento. BART se entrena corrompiendo el texto con una función de ruido arbitraria y aprendiendo un modelo para reconstruir el texto original.

La tarea de preentrenamiento consiste en barajar aleatoriamente el orden de las frases originales y un novedoso esquema de relleno, en el que se sustituyen espacios de texto por un único *token* de máscara.

Se utilizan cinco tipos de métodos de ruido:

- Enmascaramiento de token: la misma técnica de BERT.
- Borrado de token: borrar el *token* y hacer que el modelo restaure el *token* borrado en la posición correcta.
- Relleno de texto: se seleccionan varias palabras en un intervalo y se sustituyen por un único *token* de enmascaramiento. Esto enseñará al modelo a predecir cuántos *tokens* faltan.
- Permutación de frases: barajar las frases y hacer que el modelo las restaure.
- Rotación de documentos: seleccionar un *token* al azar, cambiar el orden del documento para que empiece por el *token* seleccionado y hacer que el modelo prediga el inicio del documento original.

En la figura 2.10 se puede observar un ejemplo de cómo funciona cada método de ruido intuitivamente.

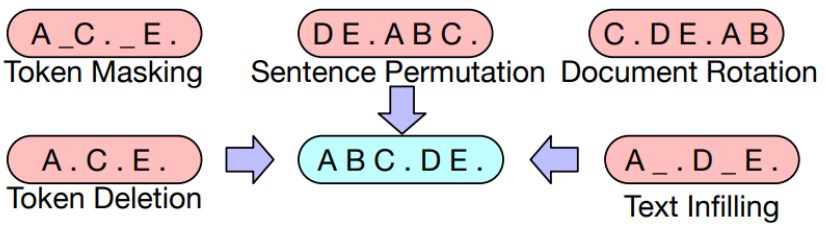


Figura 2.10: Transformaciones de ruido de entrada de BART obtenidas de [35]

Para las tareas de clasificación de secuencias, se introduce la misma entrada en el *encoder* y el *decoder*, y el estado oculto final del *token* del *decoder* final se introduce en un nuevo clasificador lineal multiclase. Este enfoque está relacionado con el *token* [CLS] de BERT; sin embargo, se añade el *token* adicional al final para que la representación del *token*

en el *decoder* pueda utilizar a los estados del *decoder* de la entrada completa. Ver en figura 2.11

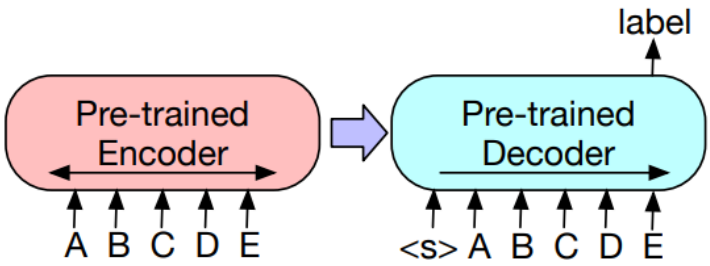


Figura 2.11: Para utilizar BART en problemas de clasificación, se introduce la misma secuencia en el *encoder* y el *decoder*, y se utiliza la representación de la salida final. [35]

BART es especialmente eficaz en la generación de textos, pero también funciona bien en tareas de comprensión. Iguala el rendimiento de RoBERTa con recursos de entrenamiento comparables en GLUE y SQuAD, y logra nuevos resultados de vanguardia en una serie de tareas de diálogo abstracto, sistemas de preguntas y respuestas, y resumen de documentos.

En la tarea CoLA (*Corpus of Linguistic Acceptability*), ALBERT obtuvo una puntuación de 62,8. En la tarea SST-2 (*Stanford Sentiment Treebank*), obtuvo una puntuación de 96,6.

2.4.7. ELECTRA

ELECTRA [11] fue presentado por investigadores de Brain Team y la Universidad de California en Berkeley en un artículo publicado en marzo de 2020. El modelo se desarrolló como una alternativa más eficiente a otros modelos basados en *Transformers*, como BERT, al hacer un mejor uso de los recursos computacionales.

ELECTRA utiliza un enfoque de preentrenamiento que entrena dos modelos de *Transformer*: el generador y el discriminador. La función del generador es sustituir los *tokens* de una secuencia, por lo que se entrena como un modelo lingüístico enmascarado. Las entradas son corrompidas por ese modelo de lenguaje, que toma un texto de entrada enmascarado aleatoriamente y produce un texto en el que ELECTRA tiene que predecir qué *token* es original y cuál ha sido sustituido.

Al igual que en el entrenamiento de GAN [19], el pequeño modelo lingüístico se entrena durante unos pasos (pero con los textos originales como objetivo, no para engañar al modelo ELECTRA como en un entorno GAN tradicional) y después se entrena el modelo ELECTRA durante unos pasos. En la figura 2.12 se muestra cómo funciona la tarea RTD en ELECTRA.

Realizaron experimentos que demuestran que esta tarea de preentrenamiento, llamada en secciones anteriores como *Replaced token detection* (RYD), es más eficaz que el MLM, ya que la tarea se define sobre todos los *tokens* de entrada y no sólo sobre el pequeño subconjunto que fue enmascarado. Como resultado, las representaciones contextuales aprendidas por este enfoque superan sustancialmente a las aprendidas por BERT con el mismo tamaño de modelo y datos.

En la tarea CoLA (*Corpus of Linguistic Acceptability*), ELECTRA obtuvo una puntuación de 71,1. En la tarea SST-2 (*Stanford Sentiment Treebank*), obtuvo una puntuación de 97,1.

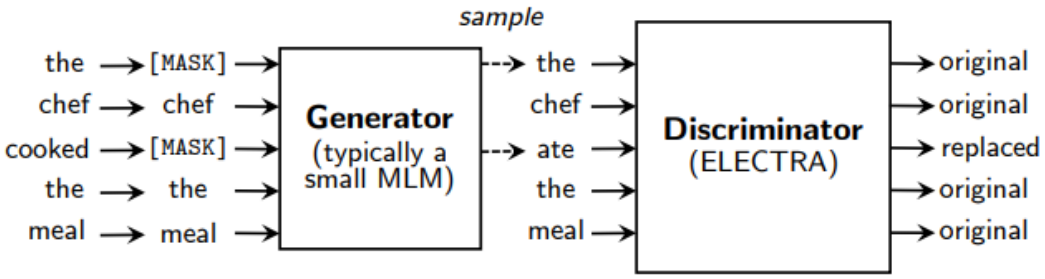


Figura 2.12: Visión general de *Replaced token detection*. Fuente: [11]

2.5 Ventajas y desventajas

Los modelos lingüísticos preentrenados han cobrado cada vez más importancia en el campo del PLN. La principal ventaja está basada en el *Transfer learning* [40], en el cual los modelos de lenguaje preentrenados se entrenan con datos de texto a gran escala y sin etiquetar, aprendiendo representaciones de palabras contextualizadas que pueden ajustarse para tareas posteriores específicas. Este enfoque de *Transfer learning* reduce significativamente la cantidad de datos etiquetados y el tiempo de entrenamiento necesarios para alcanzar el rendimiento más avanzado en diversas tareas de PLN.

Con los MLP se logra una mejora del rendimiento, ya que los modelos lingüísticos preentrenados, como BERT, GPT y RoBERTa, han obtenido resultados del estado del arte en una amplia gama de tareas de PLN, como el análisis de sentimientos, el reconocimiento de entidades nombradas, sistemas de preguntas y respuestas, y la traducción automática. Estos modelos han superado a los modelos tradicionales para tareas específicas, lo que demuestra el poder del *Transfer learning* y el preentrenamiento a gran escala.

Otra ventaja de los modelos lingüísticos preentrenados, especialmente los basados en la arquitectura *Transformer*, es que son muy eficaces a la hora de captar la información sintáctica y semántica del texto [10]. Esta capacidad les permite manejar estructuras lingüísticas complejas y obtener mejores resultados en tareas que requieren un profundo conocimiento del contexto y de las relaciones entre palabras y frases.

Las técnicas tradicionales de *word embeddings*, como Word2Vec y GloVe, tienen dificultades para representar eficazmente las palabras poco frecuentes debido a su escasa presencia en los datos de entrenamiento. En cambio, los modelos lingüísticos preentrenados utilizan la tokenización de subpalabras [42], lo que les permite representar las palabras raras como una combinación de unidades de subpalabras más frecuentes. De este modo, se obtienen representaciones más significativas y precisas de las palabras raras.

Otra de las ventajas de los MLP es el *few-shot learning*. Con los recientes avances en los modelos lingüísticos preentrenados, como el GPT-3, *few-shot learning* se ha convertido en una posibilidad [3]. Estos modelos pueden realizar varias tareas de PLN con un ajuste mínimo o incluso sin ningún entrenamiento específico, basándose en unos pocos ejemplos proporcionados durante la inferencia. Esta capacidad reduce significativamente la necesidad de datos etiquetados y simplifica el despliegue de modelos de PLN en aplicaciones del mundo real.

Los grandes modelos preentrenados presentan varias desventajas a pesar de su impresionante rendimiento en diversas tareas de PLN. El entrenamiento de estos modelos requiere gran cantidad de recursos computacionales, como GPUs o TPUs potentes [81]. Esta elevada demanda de recursos puede suponer un aumento de los costes y del con-

sumo de energía, lo que limita la accesibilidad del entrenamiento de estos modelos para investigadores y organizaciones con presupuestos limitados.

Los modelos de gran tamaño suelen tener millones o incluso miles de millones de parámetros, lo que puede dar lugar a tiempos de inferencia más lentos durante el despliegue. Esto puede ser problemático para las aplicaciones en tiempo real o en situaciones en las que las respuestas de baja latencia son cruciales.

Un asunto a tener en cuenta, también relacionado con los recursos computacionales, es que el gran tamaño de los modelos preentrenados hace que consuman mucha memoria, lo que dificulta su despliegue en dispositivos periféricos o en entornos con recursos de memoria limitados. Esto puede limitar su aplicabilidad en determinados escenarios o requerir técnicas de compresión de modelos para reducir su tamaño, lo que puede afectar a su rendimiento. Con el objetivo de resolver este problema, se han realizado varias investigaciones como [36].

Otra desventaja de los grandes modelos preentrenados es que suelen considerarse "cajas negras" debido a su compleja arquitectura y al gran número de parámetros que contienen [24]. Esta falta de interpretabilidad puede dificultar la comprensión de las predicciones del modelo o la identificación y tratamiento de posibles sesgos.

Un potencial problema a la hora de utilizar los MLP es el *overfitting* y *overparameterization* [71]. Los modelos grandes son más propensos al *overfitting* cuando se ajustan con precisión a conjuntos de datos pequeños, ya que tienen más parámetros de los necesarios para algunas tareas. Para resolver este problema, pueden ser necesarias técnicas de regularización o modelos más pequeños.

Muy importante a tener en cuenta es el tema ético en la utilización de los MLP. Los modelos preentrenados aprenden a partir de datos de texto a gran escala, que pueden contener sesgos presentes en los datos de entrenamiento. Estos sesgos pueden propagarse a tareas posteriores, lo que puede dar lugar a resultados injustos o sesgados en determinadas aplicaciones [37]. Abordar estas cuestiones éticas es un reto constante en este campo.

A pesar de estas desventajas, los modelos preentrenados han sido fundamentales para el avance del PLN, y los investigadores y profesionales trabajan continuamente para desarrollar modelos más eficientes y éticos.

CAPÍTULO 3
Estado del Arte

La clasificación por dominios es una tarea vital en el PLN, cuyo objetivo es categorizar automáticamente segmentos de texto en temas predefinidos. Este capítulo examina el estado del arte de la clasificación de texto en general, y clasificación de dominio en particular, a nivel de segmento, centrándose en los avances recientes y en la bibliografía relacionada. Se discuten los principales enfoques, métricas de evaluación y resultados obtenidos por diversos estudios.

3.1 Métricas de evaluación

El rendimiento de los modelos de clasificación de texto suele evaluarse mediante las métricas *accuracy*, *error rate*, *precision*, *recall* y *F1-score* [21].

La forma más directa de medir el rendimiento de un clasificador es utilizar la métrica *accuracy*, en donde se compara la clase real y la predicha de cada segmento, y cada coincidencia cuenta como una predicción correcta. La *precision* se obtiene dividiendo el número de predicciones correctas por el número total de predicciones.

Otra forma de calcular el *accuracy* es teniendo en cuenta los siguientes conceptos:

- *True Positives* (TP): número de textos reconocidos correctamente en la clase particular.
- *False Positive* (FP): número de textos reconocidos incorrectamente en una clase determinada.
- *True Negatives* (TN): número de textos que correctamente no se reconocen en una clase determinada.
- *False Negatives* (FN): el número de textos que incorrectamente no se reconocen en una clase determinada.

Se puede calcular el *accuracy* (Acc), teniendo en cuenta las cantidades de TP, FP, TN y FN con la siguiente ecuación:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

En la clasificación de textos, el *error rate* (ER) es el porcentaje de ejemplos mal clasificados en un conjunto de datos. Se calcula dividiendo el número de ejemplos mal clasificados por el número total de ejemplos del conjunto de datos, utilizando la siguiente

ecuación:

$$Err = \frac{FP + FN}{TP + TN + FP + FN} = 1 - Acc \tag{3.2}$$

El *accuracy* [53] se utiliza a menudo como medida del rendimiento de la clasificación porque es sencilla de calcular y fácil de interpretar. Sin embargo, en algunos casos puede resultar engañosa.

Esto es especialmente cierto cuando se trata de datos desequilibrados, un escenario en el que ciertas clases contienen muchos más puntos de datos que las otras.

A veces, el *accuracy* puede ocultar los matices de los conjuntos de datos desequilibrados. La razón es que en ciertos tipo de conjuntos de datos, hay clases donde la categoría TN domina, diluyendo el efecto del resto. Por lo tanto, incluso si el clasificador obtuviera malos resultados en las otras clases, su *accuracy* seguiría pareciendo buena, enmascarando sus deficiencias.

La *precision* (P) [53] mide la proporción de verdaderos positivos entre todos los positivos predichos, mientras que el *recall* (R) mide la proporción de verdaderos positivos entre todos los positivos reales. El *F1-score* (F1) es la media armónica de la *precision* y el *recall* y es una métrica comúnmente utilizada para equilibrar ambas medidas.

Las ecuaciones para calcular las métricas mencionadas anteriormente son las siguientes:

$$P = \frac{TP}{TP + FP} \tag{3.3}$$

$$R = \frac{TP}{TP + FN} \tag{3.4}$$

$$F1 = \frac{2 \times P \times R}{(P + R)} \tag{3.5}$$

El *F1-score* [53] es una métrica popular para evaluar el rendimiento de un modelo de clasificación.

En el caso de la clasificación multiclase, se adoptan métodos de promediación para calcular el *F1-score*, lo que da lugar a un conjunto de puntuaciones medias diferentes (micro, macro, *weighted*) en el informe de clasificación.

El macro-F1 [50] es quizás el más sencillo de los numerosos métodos de promediado. Se calcula utilizando la media aritmética de todas las puntuaciones F1 por clase. Este método trata todas las clases por igual independientemente de sus valores de soporte, donde soporte se refiere al número de ocurrencias reales de la clase en el conjunto de datos.

El *weighted-F1* [50] se calcula tomando la media de todas las puntuaciones F1 por clase teniendo en cuenta el soporte de cada clase. La ponderación se refiere esencialmente a la proporción del apoyo de cada clase en relación con la suma de todos los valores de apoyo. Con la media ponderada, la media de salida tiene en cuenta la contribución de cada clase ponderada por el número de ejemplos de esa clase dada.

El micro-F1 [50] calcula una puntuación F1 media global contando las sumas de los TP, los FN y los FP. Primero se suman los valores respectivos de TP, FP y FN en todas las clases y luego se introducen en la ecuación F1 para obtener la puntuación micro-F1. Esto se debe a que el micro-F1 calcula esencialmente la proporción de observaciones correctamente clasificadas de entre todas las observaciones, muy parecido a calcular el *accuracy*

En general, si se trabaja con un conjunto de datos desequilibrado en el que todas las clases tienen la misma importancia, utilizar el macro-F1 sería una buena opción, ya que trata a todas las clases por igual. Si se tiene un conjunto de datos desequilibrado pero se desea asignar una mayor contribución a las clases con más ejemplos en el conjunto de datos, entonces es preferible el *weighted-F1*. Esto se debe a que, en la media ponderada, la contribución de cada clase a la media F1 se pondera por su tamaño. Si se tiene un conjunto de datos equilibrado y se desea una métrica fácilmente comprensible para el rendimiento general, independientemente de la clase, se puede optar por el *accuracy*, que es esencialmente el micro-F1.

Otra métrica útil en clasificación multiclase es la matriz de confusión [50]. Una matriz de confusión es una tabla que suele utilizarse para evaluar el rendimiento de un modelo de aprendizaje automático en tareas de clasificación. Muestra las clases reales y las predichas para un conjunto de ejemplos y proporciona un resumen del número de predicciones correctas e incorrectas realizadas por el modelo.

La matriz de confusión suele organizarse en forma de tabla con las clases reales en filas y las clases predichas en columnas. Cada celda de la tabla representa el número de ejemplos que se clasificaron en una combinación concreta de clases reales y predichas. La matriz de confusión puede utilizarse para calcular varios parámetros de rendimiento del modelo de clasificación, como el *accuracy*, la *precision*, el *recall* y el *F1-score*. Estas métricas proporcionan una comprensión más detallada del rendimiento del modelo y pueden ayudar a identificar áreas de mejora.

3.2 Métodos tradicionales

Tradicionalmente, el proceso de clasificación de textos suele dividirse en varias etapas principales que incluyen la recopilación de documentos de datos, el preprocesamiento del texto, la extracción de características, la reducción de la dimensionalidad, diferentes técnicas de clasificación y la evaluación del rendimiento.

La recopilación de datos es un paso esencial en el desarrollo de un modelo de clasificación de textos. Consiste en reunir un conjunto diverso y representativo de documentos de texto que puedan utilizarse para entrenar y probar el modelo.

El preprocesamiento de texto puede incluir varios pasos, como la tokenización, la eliminación de *stopwords* y la *stemming*.

La tokenización se utiliza para eliminar los espacios en blanco y los caracteres especiales. Las *stopwords* son palabras muy comunes, que se eliminan al tener poco significado informativo, por ejemplo, *"the"*, *"a"*, *"and"*, *"that"*. El *stemming* se utiliza para eliminar los sufijos y prefijos de las palabras clave, es decir, el procedimiento de *stemming* consiste en reducir los términos modificados a su raíz.

La extracción y selección de características ayuda a identificar palabras importantes en un documento de texto. Para ello se utilizaban métodos como TF-IDF [62], LSI [38], multipalabra [79], entre otras, o una combinación de estas técnicas. El TF-IDF es una técnica puramente estadística para evaluar la importancia de una palabra basándose en su frecuencia de aparición en el texto. Las técnicas LSI y las técnicas multipalabra son técnicas orientadas a la semántica que también intentan superar los dos problemas básicos: la polisemia (una palabra con varios significados distintos) y la sinonimia (palabras diferentes con el mismo significado). La técnica LSI trata básicamente de utilizar la semántica en el texto mediante manipulaciones de matrices SVD (*Singular Value Decomposition*). Una palabra múltiple es una secuencia de palabras consecutivas con un significado semántico (por ejemplo, "Tecnología de la Información", "Escuela Pública de Delhi", "Departamen-

to de Ingeniería Informática", "Banco Estatal de la India"). Las palabras múltiples son útiles para la clasificación y la desambiguación. [15]

Entre los algoritmos de clasificación más utilizados se encuentran *Naïve Bayes*, las máquinas soporte vectorial (SVM), los modelos ocultos de Markov (HMM), los árboles de *gradient boosting* y bosques aleatorios (*random forests*).

3.3 Enfoques de aprendizaje profundo

El enfoque de los métodos tradicionales tienen varias limitaciones. Por ejemplo, la dependencia de las características requiere una tediosa ingeniería y análisis de características para obtener un buen rendimiento. Además, la fuerte dependencia del conocimiento del dominio para diseñar características dificulta la generalización del método a nuevas tareas. Por último, estos modelos no pueden aprovechar al máximo grandes cantidades de datos de entrenamiento porque las características están predefinidas.

Se han explorado enfoques neuronales para abordar las limitaciones debidas al uso de características manuales. El componente central de estos enfoques es un modelo de *embeddings* que convierte el texto en un vector de características continuas de baja dimensión, por lo que no es necesario crear a mano las características.

Uno de los primeros modelos de *embeddings* es el análisis semántico latente (LSA) [16] en 1989. El LSA es un modelo lineal con menos de 1 millón de parámetros, entrenado con 200.000 palabras. En 2001, [5] proponen el primer modelo neural del lenguaje basado en una red neural *feed-forward* entrenada con 14 millones de palabras. Sin embargo, estos primeros modelos de *embeddings* obtienen peores resultados que los modelos clásicos que utilizan características elaboradas a mano, por lo que no se adoptan de forma generalizada.

El cambio de paradigma se produce cuando se desarrollan modelos de *embeddings* mucho más grandes utilizando cantidades mucho mayores de datos de entrenamiento. En 2013, Google desarrolla una serie de modelos word2vec [44] que se entrenan con 6.000 millones de palabras e inmediatamente se popularizan para muchas tareas de PLN.

En 2017, los equipos de AI2 y la Universidad de Washington desarrollan un modelo de *embeddings* contextual basado en una LSTM bidireccional de 3 capas con 93M de parámetros entrenado en 1B palabras. El modelo, denominado ELMo [52] funciona mucho mejor que word2vec porque captura información contextual.

En 2018, OpenAI comienza a construir modelos de *embeddings* utilizando *Transformers*, lo que mejora sustancialmente la eficiencia del entrenamiento de modelos a gran escala en TPU. Su primer modelo se llama GPT, que ahora se utiliza ampliamente para tareas de generación de texto. Ese mismo año, Google desarrolla BERT, basado en el *Transformer* bidireccional. BERT consta de 340 millones de parámetros, entrenados con 3.300 millones de palabras. Pero existen muchos más modelos que estos.

A continuación se listan las técnicas basadas en aprendizaje profundo más utilizadas en tareas basadas en texto:

1. **Redes feed-forward:** Estas redes consideran el texto como una bolsa de palabras y no capturan dependencias entre palabras o estructuras del texto.

2. **Modelos basados en RNN:** Estos modelos tratan el texto como una secuencia de palabras y están diseñados para capturar dependencias entre palabras y estructuras del texto.

3. **Modelos basados en CNN:** Estos modelos se entrenan para reconocer patrones en el texto, como frases clave, mediante convoluciones.

4. **Capsule networks:** Abordan el problema de la pérdida de información que sufren las operaciones de agrupamiento en las CNN.

5. **Mecanismo de atención:** Es eficaz para identificar palabras correlacionadas en el texto y se ha convertido en una herramienta útil en el desarrollo de modelos de aprendizaje profundo.

6. **Memory-augmented networks:** Combinan redes neuronales con una forma de memoria externa, de la que los modelos pueden leer y escribir.

7. **Graph neural networks:** Están diseñadas para capturar las estructuras gráficas internas del lenguaje natural, como los árboles sintácticos y semánticos.

8. **Redes neuronales siamesas:** Están diseñadas para la concordancia de textos, es decir, comparar la similitud entre dos o más textos.

9. **Modelos híbridos:** Combinan atención, RNNs, CNNs, entre otros, para capturar características locales y globales de frases y documentos.

10. ***Transformers*:** Permiten mucha más paralelización que las RNN, lo que hace posible entrenar de forma eficiente modelos lingüísticos muy grandes utilizando GPU.

11. **Tecnologías de modelado más allá del aprendizaje supervisado:** Incluyen el aprendizaje no supervisado mediante *autoencoders* y entrenamiento adversarial, y el aprendizaje por refuerzo.

En la línea de tiempo en la figura 3.1 se pueden apreciar algunos de los modelos de aprendizaje profundo más destacados para los *embeddings* y clasificación de texto publicados entre 2013 y 2020.

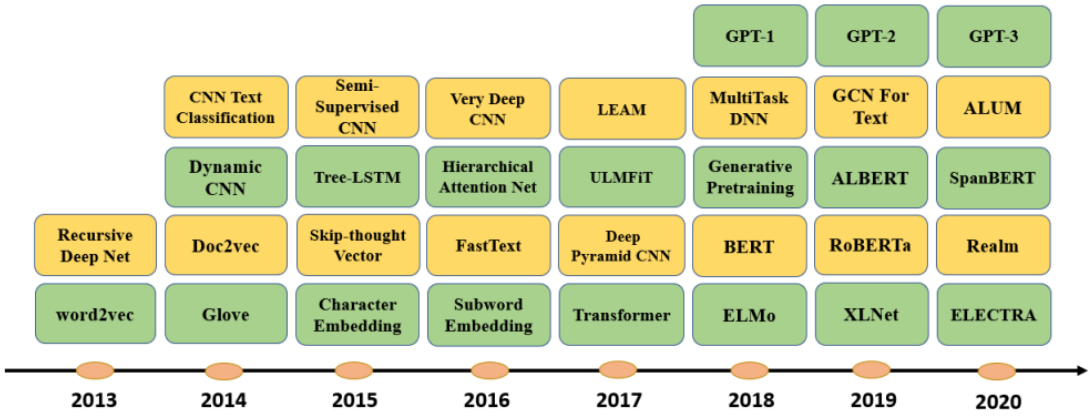


Figura 3.1: Modelos de aprendizaje profundo más destacados para los *embeddings* y clasificación de texto publicados entre 2013 y 2020. Fuente: [45]

Method	AG News	20NEWS	Sogou News	DBpedia
Text GCN [77]	67,61	86,34	-	-
Simplfied GCN [72]	-	88,5	-	-
Char-level CNN [80]	90,49	-	95,12	98,45
CCCapsNet [58]	92,39	-	97,25	98,72
LEAM [67]	92,45	81,91	-	99,02
fastText [26]	92,5	-	96,80	98,60
CapsuleNet B [82]	92,6	-	-	-
Deep Pyramid CNN [25]	93,13	-	98,16	99,12
ULMFiT [22]	94,99	-	-	99,20
L MIXED [59]	95,05	-	-	99,30
BERT-Large [74]	-	-	-	99,32
XLNet [76]	95,51	-	-	99,38

Tabla 3.1: Resultados de modelos en *datasets* de categorización de texto

En particular, para la categorización en dominios, existen populares *datasets* utilizados como referencia. Entre ellos se encuentran:

- AG News [80]: colección de artículos de noticias recopilados de más de 2.000 fuentes de noticias por ComeToMyHead, un motor de búsqueda de noticias académicas. Este conjunto de datos incluye 120.000 muestras de entrenamiento y 7.600 muestras de prueba. Cada muestra es un texto breve con una etiqueta de cuatro clases (Mundo, Deportes, Negocios y Ciencia/Tecnología).
- 20 Newsgroups [32]: colección de documentos de grupos de noticias publicados sobre 20 temas diferentes como ciencia, política, religión, deporte, entre otros. Varias versiones de este conjunto de datos se utilizan para la clasificación de textos, la agrupación de textos, etcétera. Una de las versiones más populares contiene 18.821 documentos clasificados por igual en todos los temas.
- Sogou News [63]: mezcla de los corpus de noticias SogouCA y SogouCS. Las etiquetas de clasificación de las noticias vienen determinadas por sus nombres de dominio en la URL. Por ejemplo, las noticias con URL http://sports.sohu.com se clasifican como clase deporte.
- Reuters news: Reuters-21578 es una de las colecciones de datos más utilizadas para la categorización de textos, y se recopiló a partir del servicio de noticias financieras de Reuters en 1987. ApteMod es una versión multiclase de Reuters-21578 con 10.788 documentos. Tiene 90 clases, 7.769 documentos de entrenamiento y 3.019 documentos de prueba. Otros conjuntos de datos derivados de Reuters son R8, R52, RCV1 y RCV1-v2.
- DBpedia [34]: El conjunto de datos DBpedia es una base de conocimientos multilingüe a gran escala creada a partir de las *infoboxes* más utilizadas de Wikipedia. DBpedia se publica cada mes y en cada versión se añaden o algunas clases y propiedades. La versión más popular de DBpedia contiene 560.000 muestras de entrenamiento y 70.000 muestras de prueba, cada una con una etiqueta de 14 clases.

En 3.1 se puede observar el *accuracy* obtenido por algunos modelos de aprendizaje profundo para tareas de categorización de texto, obtenidos de [45], donde se puede observar que los resultados obtenidos por modelos preentrenados son bastante satisfactorios, y que son muy dependientes del *dataset* utilizado en la evaluación.

CAPÍTULO 4
Creación del Dataset

En este capítulo, nos adentramos en el proceso de generación del *dataset* para la tarea de clasificación por dominios basada en segmentos, en el idioma inglés. La creación de un *dataset* de alta calidad es crucial para entrenar y evaluar modelos de aprendizaje automático que reconozcan y categoricen eficazmente los distintos temas de los texto. Por tanto, analizaremos el proceso de recopilación, preprocesamiento y anotación de datos, así como las técnicas empleadas para intentar garantizar la calidad, diversidad y representatividad de los datos.

La calidad de un modelo depende de varios factores relacionados con el *dataset* de entrenamiento.

- Tamaño del *dataset*: Un *dataset* más grande generalmente ayuda al modelo a aprender más características y matices del lenguaje, lo que conduce a una mejor generalización y un mejor rendimiento en datos no vistos.
- Equilibrio de clases: Si el *dataset* tiene una distribución desigual de las clases, el modelo puede sesgarse hacia la clase mayoritaria. Garantizar un *dataset* equilibrado ayuda a lograr una mayor precisión en la clasificación de todas las clases.
- Calidad de los datos: Los datos de alta calidad son esenciales para entrenar un modelo fiable. El conjunto de datos debe estar libre de ruido, errores e incoherencias. Limpiar los datos puede mejorar significativamente el rendimiento del modelo.
- Representatividad: El *dataset* debe ser representativo del ámbito del problema y contener ejemplos de todos los escenarios posibles. Si el conjunto de datos no capta la diversidad del problema, el modelo puede no funcionar bien en situaciones reales.
- Calidad del etiquetado: El etiquetado preciso y coherente del conjunto de datos es fundamental. Los datos mal etiquetados pueden hacer que el modelo aprenda patrones incorrectos y afectar negativamente a su rendimiento.

Con el objetivo de lograr un *dataset* con estas características, en las siguientes secciones se describe la metodología utilizada para la creación del mismo, las características deseadas del *dataset* e información estadística de los datos obtenidos finalmente.

4.1 Características de los dominios

Para la generación del *dataset* para la clasificación por dominios, primero es importante definir cuáles son las clases de dominios que sean más útiles, con una definición clara. En este caso, se definieron los siguientes dominios y sus correspondientes descripciones:

- **AUT**: Automoción, transporte, normas de tráfico.
- **LEG**: Jurídico, derecho, recursos humanos, certificados, diplomas.
- **MWM**: *Marketing*, web, *merchandising*, soporte y servicio al cliente, *e-commerce*, publicidad, encuestas.
- **LSM**: Medicina, ciencias naturales, alimentos/nutrición, biología, sexología, cosméticos, química, genética.
- **ENV**: Medio ambiente, agricultura, silvicultura, pesca, ganadería, zoología, ecología.
- **FIN**: Finanzas, economía, contabilidad, seguros, negocios, empresa, certámenes, trabajo, empleo.
- **POL**: Política, relaciones internacionales, Unión Europea, organizaciones internacionales, defensa, ejército.
- **PRN**: Porno, contenido inapropiado.
- **COM**: Ordenadores, informática, robótica, domótica, consolas, telecomunicaciones.
- **ING**: Ingeniería pura (mecánica, eléctrica, electrónica, aeroespacial, etc), meteorología, minería, navegación, marítima, acústica.
- **ARC**: Arquitectura,ingeniería civil,construcción, obras públicas.
- **MAT**: Matemáticas, estadística, física.
- **HRM**: Historia, religión, mitología, folclore, filosofía, psicología, ética, antropología, turismo.
- **CUL**: Arte, poesía, literatura, cine, videojuegos, teatro, guiones de teatro o cine, esoterismo, astrología, deportes, música, fotografía.
- **GEN**: General - clase genérica con temas como ropa, textiles, gastronomía.

Esta definición de etiquetas para una tarea de clasificación de dominio proporciona varias ventajas:

- **Amplia cobertura**: Estas etiquetas cubren una amplia gama de temas de diversos dominios.
- **Especificidad**: Cada etiqueta está bien definida y es específica, lo que permite una clasificación precisa y reduce las posibilidades de clasificación errónea o solapamiento entre categorías.
- **Estructura jerárquica**: Algunas de estas etiquetas pueden considerarse categorías más amplias que engloban subcategorías, lo que proporciona una estructura jerárquica a la clasificación. Por ejemplo, **LEG** incluye legal, derecho, RRHH, certificados, diplomas, etc.
- **Inclusión de contenidos sensibles**: La inclusión de etiquetas como **PRN** para contenido inapropiado permite identificar y filtrar potencialmente material sensible o explícito.

- **Adaptabilidad**: Estas etiquetas pueden aplicarse a una amplia gama de fuentes de datos, como artículos, documentos o publicaciones en redes sociales, lo que hace que el sistema de clasificación sea versátil y adaptable.
- **Multidisciplinar**: Las etiquetas abarcan varias disciplinas, como las finanzas, la ingeniería y la política, lo que hace que el sistema de clasificación sea útil para estudios y aplicaciones interdisciplinares.
- **Aplicabilidad multilingüe**: Estas etiquetas pueden aplicarse en distintos idiomas, ya que se basan en temas generales y no en características específicas de un idioma. Poder aplicarlo a varios idiomas es un plus importante, ya que esta tarea en inglés es solo el primer paso, el siguiente paso sería realizarla en otros idiomas, incluidos aquellos con bajos recursos.
- **Facilidad de ampliación**: Este sistema de clasificación puede ampliarse o modificarse fácilmente añadiendo o fusionando etiquetas, en función de los requisitos específicos del proyecto o del dominio.

4.2 Fuente de datos

Para la obtención de los datos por dominios se utilizaron una variedad de fuentes de datos, como Wikipedia, *datasets* públicos, blogs, forums, redes sociales, entre otros. A continuación se describirán cuáles fueron las fuentes de datos por dominio que se tuvieron en cuenta para la creación de los datos.

Para la obtención de los datos de páginas web, se utilizó la técnica *Web scraping* [28]. El *web scraping*, también conocido como *web crawling*, es una técnica utilizada para recopilar y extraer información de sitios web de Internet. Consiste en navegar mediante programación por las páginas web, identificar el contenido relevante y recuperar los datos para su posterior procesamiento o análisis. El *web scraping* se utiliza ampliamente para diversas aplicaciones, como la minería de datos, el análisis de datos, el análisis de sentimientos, la comparación de precios y el análisis de la competencia, en este caso se utilizará para la minería de texto.

El proceso de *web scraping* se realizó siguiendo los pasos descritos a continuación:

- Identificación de los sitios web objetivo: Se determinaron los sitios web de los que se deseaba extraer datos. Estos sitios web contenían información relevante para cada dominio y subdominio específico.
- Solicitud y acceso a las páginas web: Envío de peticiones HTTP a las URL de los sitios web objetivo para acceder a su contenido. Las herramientas o bibliotecas de *web scraping* suelen tener funciones integradas para gestionar las solicitudes y respuestas HTTP, en nuestro caso se utilizó Selenium con el lenguaje JavaScript.
- Análisis del contenido HTML: Una vez que se accedió a las páginas web, se analizó el contenido HTML para localizar e identificar los elementos de datos específicos necesarios, en este caso título, resumen, y el cuerpo del artículo/documento correspondiente. Este paso implicó el uso de una biblioteca de análisis sintáctico para navegar por la estructura HTML y extraer la información relevante, en este caso BeautifulSoup en Python, aunque existen otras como Jsoup en Java.
- Extracción de datos: Una vez identificados los elementos de datos deseados, se extrajeron del contenido HTML. Aunque esto puede implicar la extracción de imágenes, enlaces u otros tipos de datos, se obtuvo solo el texto en los casos en los que era posible.

- Limpieza y preprocesamiento de los datos: Se limpiaron los datos extraídos eliminando cualquier información irrelevante, etiquetas HTML o caracteres especiales. Los detalles referentes a este paso serán descritos en más profundidad en la siguiente sección.

Para la obtención de datos del dominio **ARC** se obtuvieron los datos principalmente de Wikipedia y de blogs acerca de este tema. Entre las páginas web consultadas se encuentran:

- ArchDaily: blog que cubre obras de arquitectura, urbanismo y diseño. Es la plataforma online de arquitectura y diseño más visitada del mundo. Cubre obras de arquitectura, noticias, productos, acontecimientos, entrevistas, competiciones, columnas de opinión, entre otros.
- Architects' Journal: Architects' Journal (conocida en las bibliografías arquitectónicas por las siglas AJ) es una revista especializada en Arquitectura que se publica semanalmente en Londres
- Designboom: Fundada en Milán en 1999, Designboom es la primera revista en línea del mundo. Es conocida como la referencia para todo lo relacionado con la arquitectura, según la revista Forbes. Es una plataforma que también es muy popular, ya que se ha convertido en una revista de culto durante más de 20 años.
- Design Milk: Es un blog enfocado en el diseño con lo último y lo más novedoso en arquitectura, interiorismo, automoción, moda, tecnología y arte. En particular, se tuvo en cuenta solo la sección de arquitectura.
- Wikipedia: Wikipedia es una enciclopedia libre, políglota y editada de manera colaborativa. De aquí se obtuvieron las páginas relacionadas con la arquitectura de varios lugares del mundo, utilizando el índice de arquitectura.

Para la obtención de datos del dominio de comunicación (**COM**) se revisaron y seleccionaron datos de varios *datasets* públicos encontrados en Kaggle, una plataforma web que reúne la comunidad *Data Science* más grande del mundo. Entre ellos se encuentran:

- Yahoo! Answers Topic Classification: El *dataset* de clasificación de temas de Yahoo Respuestas se construye utilizando las 10 categorías principales más grandes. Cada clase contiene 140.000 muestras de entrenamiento y 6.000 muestras de prueba. De estos datos, solo se tienen en cuenta los clasificados como *Computers & Internet*.
- Topic Labeled News Dataset: datos obtenidos por el equipo de NewsCatcher, que contiene más de 100 mil artículos etiquetados en 8 clases, obtenidos en agosto de 2020 sobre miles de sitios web de noticias. Solo se tiene en cuenta la clase *TECHNOLOGY*
- BBC Full Text Document Classification: Consta de 2.225 documentos del sitio web de noticias de la BBC correspondientes a historias de cinco áreas temáticas de 2004-2005 como negocios, entretenimiento, política, deportes, tecnología, donde solo se observaron las correspondientes a tecnología.

Para la obtención de datos del dominio de finanzas (**FIN**) también se tuvieron en cuenta datos de Kaggle:

- Financial Sentiment Analysis: Está pensado para avanzar en la investigación del análisis del sentimiento financiero. Se trata de dos conjuntos de datos (FiQA, *Financial PhraseBank*) combinados en un archivo CSV fácil de usar. Proporciona frases financieras con etiquetas de sentimiento, las cuáles fueron ignoradas.

- US Economic News Articles : El conjunto de datos consta de unos 8.000 artículos de noticias, que se etiquetaron como relevantes o no relevantes para la economía estadounidense. Las etiquetas fueron ignoradas.
- News Category Dataset: Este *dataset* contiene alrededor de 210.000 titulares de noticias de HuffPost desde 2012 hasta 2022. Hay alrededor de 200k titulares entre 2012 y mayo de 2018 y 10k titulares entre mayo de 2018 y 2022. Está etiquetado en más de 42 categorías, solo se tuvo en cuenta la categoría *BUSINESS*

Para la obtención de datos del dominio de contenido inapropiado (**PRN**), se tuvieron en cuenta 2 *datasets*, 1 de Kaggle y 1 de Hugging Face.

- Adult-content-dataset: 850 descripciones de artículos clasificadas en dos categorías diferentes: *Adult* y *Non Adult*, donde solo se tuvieron en cuenta las clasificadas como *Adult*.
- Hate Speech and Offensive Language Dataset: Conjunto de datos de Twitter para investigar la detección del lenguaje de odio. El texto se clasifica en: lenguaje de odio, lenguaje ofensivo y ninguno de los dos. Se tuvieron en cuenta los que tenían lenguaje ofensivo y de odio.

Para obtener datos de política (**POL**) solo se tuvo en cuenta el *dataset* News Category Dataset, pero utilizando los datos etiquetados en la clase *POLITICS*

Los datos de marketing(**MWM**) se obtuvieron principalmente de blogs como:

- HubSpot Blog: Es un blog con las últimas tendencias en marketing, ventas, servicio al cliente, gestión empresarial y desarrollo de sitios web.
- Content Marketing Institute: Blog que crea experiencias de contenido que enseñan a vendedores, creadores de marcas empresariales, pequeñas empresas y agencias a atraer y retener a los clientes a través de una narración convincente y multicanal.

Los dato de la clase de matemáticas y física (**MAT**) se obtuvieron principalmente del *dataset* arXiv Dataset, un repositorio de 1,7 millones de artículos, con características relevantes como títulos de artículos, autores, categorías, resúmenes, textos completos en PDF, entre otros. Solo se obtuvieron los artículos relacionados con los subtemas relacionados con este dominio, y solo se utilizaron los resúmenes como fuente de texto.

Para obtener los datos de la clase **HRM**, se utilizaron varias fuentes:

- History of Philosophy: El *dataset* contiene más de 300.000 frases de más de 50 textos que abarcan 10 grandes escuelas filosóficas. Las escuelas representadas son: Platón, Aristóteles, Racionalismo, Empirismo, Idealismo Alemán, Comunismo, Capitalismo, Fenomenología, Filosofía Continental y Filosofía Analítica.
- Religion News Service (RNS): Es una agencia de noticias que cubre temas de religión, ética, espiritualidad y moral. Publica noticias, información y comentarios sobre creencias y movimientos religiosos para periódicos, revistas, organizaciones de radiodifusión y publicaciones religiosas. Se fundó en 1934.
- PositivePsychology: Blog que contiene recursos basados en la ciencia para profesionales de la ayuda.

En el caso de los textos legales, se utilizó el *dataset* Legal Case Reports, que contiene casos judiciales australianos del Tribunal Federal de Australia (FCA). Están todos los casos de los años 2006, 2007, 2008 y 2009.

Para la clase **LSM**, se utilizaron los siguientes *datasets* de Kaggle:

- GENIA Bio-medical event dataset: El conjunto de datos no es más que una versión simplificada del conjunto de datos GENIA anotado por eventos. Consta del texto biomédico original, las palabras desencadenantes etiquetadas, la ubicación de la palabra desencadenante en el texto y el tipo de evento asociado a la palabra desencadenante. Hay 3 conjuntos de datos: entrenamiento (más de 8.000 frases), desarrollo (unas 3.000 frases) y prueba (unas 3.000 frases). En este caso sólo se utilizaron los datos de entrenamiento y solo el texto original.
- American Chemical Society Journals: La American Chemical Society publica una amplia gama de revistas de cada área de la química. Dispone de información sobre JACS, *Inorganic Chemistry, Organometallics, Biochemistry* o *Chemical reviews*. Este *dataset* contiene todos los números completos publicados en el momento de su creación (mediados-finales de junio de 2020). Cada publicación tiene título, autores, revista, año, volumen, número, *abstract*, utilizándose solo el *abstract*.
- PubMed 200k RCT: PubMed 200k RCT es un *dataset* basado en PubMed para la clasificación secuencial de frases. El conjunto de datos consta de aproximadamente 200.000 resúmenes de ensayos controlados aleatorios, con un total de 2,3 millones de frases.

Para la clase **ENV** se utilizaron los siguientes *datasets*:

- Query Based Agricultural Data System (QuADS): Este conjunto de datos contiene información sobre la agricultura en 6 *intents* y otros *subintents* con preguntas relacionadas que pueden hacer los agricultores para comprender más en detalle el tema.
- NLP : Reports & News Classification: *Dataset* de noticias e informes relacionados con el medio ambiente y etiquetado en 5 clases como problemas medioambientales, indicadores climáticos, contaminación, entre otros.
- Environmental News NLP Dataset: Estos datos contienen breves fragmentos de noticias medioambientales desde 2017 hasta enero de 2020.

Los textos culturales (**CUL**) se obtuvieron de News Category Dataset utilizando los datos etiquetados en las clases *SPORTS* y *ENTERTAINMENT*.

Los datos de **AUT** se buscaron en Intelligent Transport, una plataforma al servicio de la industria del transporte desde hace más de 15 años, es la principal fuente de información en el sector del transporte público urbano. Cubriendo todas las nuevas tecnologías y desarrollos dentro de este sector de vital importancia, *Intelligent Transport* proporciona un análisis de alta calidad a través de varios temas principales: *MaaS* e Integración de Servicios, Ciudades Inteligentes y Conectividad, El Pasajero, Seguridad y Protección, y Personal.

Para los datos de la clase **ING**, se utilizó la página web Revolutionized, que contiene artículos relacionados con los sectores tecnológico, industrial y científico. En particular, se utilizaron artículos en la categoría *Engineering*.

La clase **GEN**, como por definición es ambigua, se consideró que encontrar datos lo suficientemente representativos no era posible, por lo que solo se tuvo en cuenta el *dataset* News Category Dataset utilizando los datos etiquetados en las clases *STYLE & BEAUTY*, *PARENTING* y *HOME & LIVING*. Con estos datos se realizaron experimentos, pero no fueron utilizados en el proceso de entrenamiento de los modelos utilizados.

Adicionalmente, para cada dominio, fueron seleccionados y etiquetados manualmente 500 segmentos, seleccionados principalmente de páginas webs.

4.3 Preprocesamiento

El preprocesamiento del texto antes de su clasificación es crucial porque influye significativamente en el rendimiento de los modelos de clasificación. Los tipos de preprocesamientos aplicados a los datos obtenidos en las fuente mencionadas en la sección anterior son muy diversos debido a la variedad de las fuentes, ya que pueden ser datos obtenidos directamente de la web o *datasets* públicos, cada uno con características diferentes. Pero de forma general los pasos seguidos para el procesamiento son los siguientes:

- Normalización presegmentación, que consiste en hacer modificaciones al texto para que el proceso de segmentación sea efectivo, y tenga la menor cantidad de errores o fallos posibles.
- Segmentación, que consiste en dividir los textos en varios textos de menor longitud.
- Normalización postsegmentación, que consiste en eliminar partes del texto que no reportan ninguna información de utilidad o que generan ruido a la hora de clasificar.
- Validación, que consiste en descartar y eliminar los segmentos que no sean útiles o tengan información de mala calidad o ruidosa.

A continuación se describirá más en profundidad en qué consiste cada uno de estos pasos y por qué son necesarios.

4.3.1. Normalización presegmentación

El proceso de segmentación [49], al ser tan dependiente de los signos de puntuación, puede generar segmentaciones incorrectas en presencia de algunos tipos de textos específicos, como urls o emails, ya que puede dividir un texto a la mitad de algunas de estas estructuras. Esto puede provocar problemas al obtener textos con contexto semántico incompleto, u obtener palabras sin sentido, y esto puede afectar negativamente al clasificador.

Para intentar evitar potenciales problemas como este, se aplicó un proceso de normalización presegmentación. Por ejemplo, se sustituyeron partes del texto con signos de puntuación fuera de lo común por etiquetas, enmascarando el texto, y guardando un identificador con cada máscara y el texto original. Este proceso se realizó con los *emails*, urls, menciones(@) y *hashtags* (#) de redes sociales, y tener en cuenta que luego de la segmentación fue revertido y regresado a su forma original. Luego, se añadieron espacios después de los puntos para apoyar al segmentador. Teniendo en cuenta que los datos se obtuvieron, no solo de páginas de noticias y blogs, sino también de *datasets* obtenidos de redes sociales, esta solución fue de gran utilidad.

Un ejemplo de cómo funcionaría este procesamiento se puede observar a continuación:

- Texto original:

  > Hey @username, I found this interesting article on AI: https://example.com/ai-article. It's a great read! Let's discuss it during our next meetup. By the

way, you can reach out to the author at author@example.com for any questions.#ArtificialIntelligence

- Texto procesado:

  Hey [ID0], I found this interesting article on AI: [ID1] It's a great read! Let's discuss it during our next meetup. By the way, you can reach out to the author at [ID2] for any questions.[ID3]

- Información guardada:

  [(ID0, @username),
  (ID1, https://example.com/ai-article),
  (ID2, author@example.com),
  (ID3,#ArtificialIntelligence)]

4.3.2. Segmentación

La segmentación de textos [49] es el proceso de dividir un texto extenso en unidades más pequeñas, como oraciones, palabras o frases. Se trata de un paso crucial en muchas tareas de procesamiento del lenguaje natural, ya que ayuda a simplificar los datos del texto y permite un mejor análisis.

Los enfoques más comunes son los métodos basados en reglas, donde se utilizan signos de puntuación (por ejemplo, puntos, signos de interrogación, signos de exclamación) y reglas de capitalización para identificar los límites de las frases. Los métodos basados en el aprendizaje automático se basan en el entrenamiento de modelos en datos anotados para identificar los límites de las frases, a menudo utilizando características como la puntuación, las mayúsculas y el contexto.

Para esta tarea, se decidió utilizar PySBD, una biblioteca de segmentación de *python*. *PySBD* [60] es un segmentador basado en reglas que funciona de forma inmediata para 22 idiomas. Es capaz de ofrecer un segmentador realista capaz de proporcionar frases lógicas incluso cuando se desconoce el formato y el dominio del texto de entrada. Está basado en el segmentador pragmático implementado en una gema de *ruby*, que llevaron a *python* mejoras y funcionalidades adicionales. *PySBD* supera el 97,92 % de los ejemplos del *Golden Rule Set*.

Este segmentador fue utilizado para segmentar textos largos en frases, y luego fue revertido el proceso de enmascaramiento realizado como uno de los pasos de normalización presegmentación.

4.3.3. Normalización postsegmentación y validación

La normalización postsegmentación se basa en limpiar partes del texto que no contienen información útil y puede resultar ruidosa. Entre los procesos de normalización aplicados luego de la segmentación se encuentra la eliminación de espacios múltiples, signos de puntuación duplicados, etiquetas HTML, *hashtags* y menciones. Además se eliminaron signos de puntuación innecesarios, urls y comillas dobles.

Luego de los procesos de segmentación y normalización, se realizó un análisis de validación para descartar los segmentos que no cumplen los requisitos diseñados para la selección de los mismos, o que contengan ruido. Entre los requerimientos y comprobaciones aplicadas a los segmentos se encuentran:

- Cantidad de palabras, entre 10 y 40 aproximadamente.
- Comprobación de idioma, asegurarse de que sea realmente inglés.
- No puede contener solamente números.
- No puede ser la cadena vacía.
- No puede contener solo un *email*.
- No puede contener solo una url.

4.4 Dataset final

Finalmente, luego de aplicar los procesos de segmentación, normalización y validación, se obtuvo un *dataset* de aproximadamente 2.000 segmentos por dominio para entrenamiento, y 500 segmentos por dominio para test. Cada segmento está en el idioma inglés y tiene un tamaño aproximado entre 10 y 40 palabras.

Algunas estadísticas del *dataset* final creado se muestran a continuación:

- Tamaño máximo de segmento: 518
- Tamaño medio de segmento: 288
- Tamaño mínimo de segmento: 10
- Total de caracteres: 4.632.441
- Total de caracteres únicos: 378
- Cantidad de segmentos de entrenamiento: 27.530
- Tamaño del archivo csv del dataset: 7,8 MiB

Las categorías y los bloques de caracteres hacen referencia a diferentes formas de organizar y clasificar los caracteres *Unicode*.

Los caracteres *Unicode* se agrupan en categorías según sus propiedades y funciones generales. Estas categorías, definidas en la Base de Datos de Caracteres Unicode (UCD), ayudan a identificar el tipo o propósito de un carácter. Algunas categorías de caracteres comunes incluyen letras mayúsculas (por ejemplo, A, B, C), letras minúsculas (por ejemplo, a, b, c), números decimales (por ejemplo, 0, 1, 2), etc.

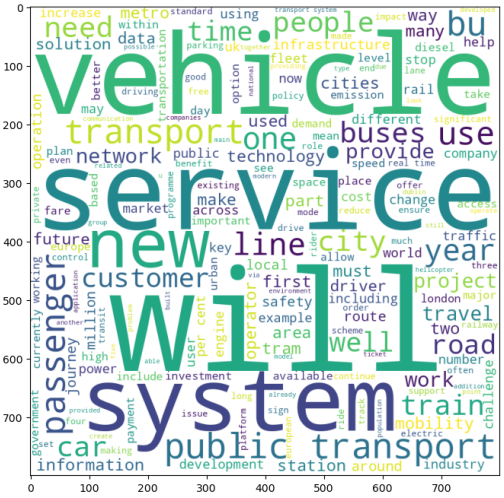
Los caracteres *Unicode* también se organizan en bloques, que son rangos contiguos de puntos de código que generalmente corresponden a un sistema de escritura específico, un guión o una colección de caracteres relacionados.

Algunas estadísticas referentes a las categorías y bloques de caracteres en el *dataset* se pueden observar en las tablas 4.1 y 4.2 respectivamente.

Adicionalmente, en la tabla 4.3 se pueden observar los símbolos de moneda más utilizado, donde el que más se destaca es el dolar, lo que llevaría un análisis adicional futuro para obtener datos con mayor variedad de países y monedas.

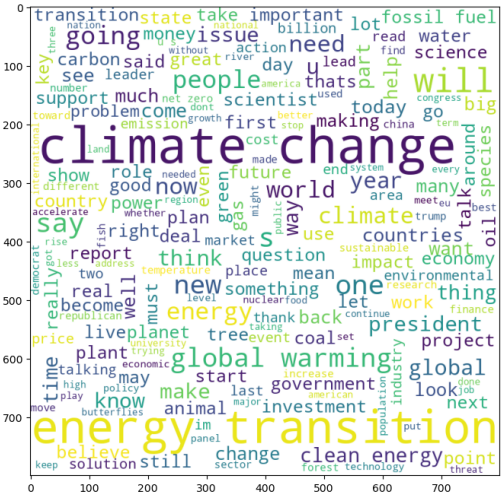
En la figura 4.1, se encuentran los *wordclouds* de algunas de las clases para que se aprecie de forma gráfica la distribución de las palabras más repetidas en cada una de ellas, obviando las *stopwords*. Tener en cuenta que la clase **PRN** no será presentada al tener palabras de contenido sensible.

(a) ARC *wordcloud*

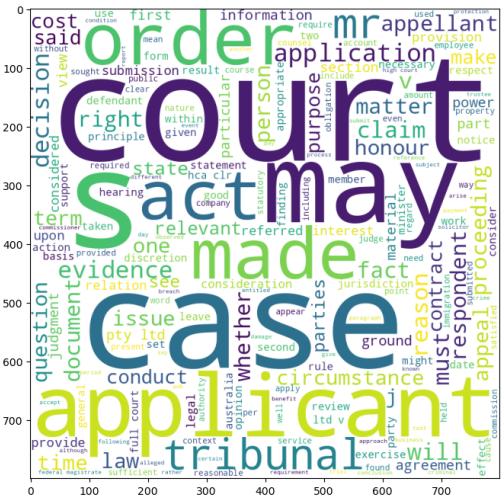


(b) AUT *wordcloud*

(c) COM *wordcloud*



(d) ENV *wordcloud*



(e) LEG *wordcloud*

(f) LSM *wordcloud*

Figura 4.1: *Wordclouds* de 6 dominios de ejemplo.

Categoría	Cantidad	Frecuencia (%)
Letra minúscula	3.639.992	78,6
Separador de espacio	746.140	16,1
Letra mayúscula	106.121	2,3
Otra puntuación	72.420	1,6
Número decimal	40.274	0,9
Puntuación de guión	13.473	0,3
Control	6.190	0,1
Símbolo de moneda	1.966	<0,1
Símbolo matemático	1.241	<0,1
Puntuación de cierre	1.043	<0,1
Otros valores (11)	3.581	0,1

Tabla 4.1: Categorías de caracteres presentes en el *dataset*, con sus cantidades y frecuencias correspondientes

Bloque	Cantidad	Frecuencia (%)
ASCII	4.627.055	99,9
Signos de puntuación	1.328	$<$ 0,1
Emoticonos	369	$<$ 0,1
Operadores matemáticos	165	$<$ 0,1
Alfanuméricos encerrados Sup	46	$<$ 0,1
VS	27	$<$ 0,1
Flechas	25	$<$ 0,1
Símbolos varios	17	$<$ 0,1
Dingbats	15	$<$ 0,1
Otros valores (11)	63	$<$ 0,1

Tabla 4.2: Bloques de caracteres presentes en el *dataset*, con sus cantidades y frecuencias correspondientes

Moneda	Cantidad	Frecuencia (%)
$	1.690	86,0
£	271	13,8
€	4	0,2
¢	1	0,1

Tabla 4.3: Frecuencia de valores monetarios en el *dataset*

CAPÍTULO 5
Experimentación y resultados

En este capítulo, se presentan los experimentos realizados y los resultados obtenidos para la tarea de clasificación de dominios en segmentos. El objetivo principal de los experimentos es investigar el rendimiento de varios modelos de lenguaje preentrenados (BERT, RoBERTa, DistilBERT, DistilRoBERTa, ALBERT, ELECTRA, XLNet, GPT-3, GPT-3.5 y BART) en el *dataset* analizado en el capítulo anterior. Además, fijando uno de estos modelos, se realizaron experimentos con diferentes tamaños de *dataset*, y se exploró la forma de clasificar los segmentos genéricos **GEN**, ya que esta clase no está incluida en el *dataset* creado.

5.1 Metodología

Como se comenta al principio del capítulo, los experimentos principales se basan en analizar el comportamiento de varios modelos de lenguaje preentrenados en la tarea de clasificación en dominios. Cada uno de los modelos utilizados consta de varias versiones o tamaños de arquitectura. Para esto se seleccionaron específicamente los modelos siguientes:

- BERT: bert-base-uncased.
- DistilBERT: distilbert-base-uncased
- RoBERTa: roberta-base
- DistilRoBERTa: distilroberta-base
- ALBERT: albert-base-v2
- ELECTRA: electra-small-discriminator
- XLNet: xlnet-base-cased
- BART: bart-base
- GPT-3: text-ada-001
- GPT 3.5: gpt-3.5-turbo

En forma de aclaración, referente a los modelos DistilRoBERTa y DistilBERT, una versión destilada de un modelo se refiere a un modelo más pequeño y eficiente que se deriva de un modelo más grande y complejo a través de un proceso llamado destilación de modelos o destilación de conocimientos. La idea principal de la destilación es transferir los

conocimientos aprendidos por un modelo más grande, también conocido como modelo del profesor, a un modelo más pequeño, denominado modelo del alumno. El modelo destilado está diseñado para tener menos parámetros y una arquitectura más simple, lo que lo hace más rápido y más eficiente en memoria sin comprometer significativamente su rendimiento. DistilBERT y DistilRoBERTa son versiones destiladas de BERT y RoBERTa, respectivamente.

Estos modelos fueron seleccionados porque son de los más utilizados en la comunidad científica. Las versiones de los mismos utilizadas fueron las *base* porque son las más ligeras, y por cuestiones de capacidad computacional y el tiempo limitado para la realización de experimentos.

En el caso de GPT-3, se seleccionó el *text-ada-001* al ser el recomendado por OpenAI para tareas de clasificación. Y en el caso de GPT-3.5, se utilizó *gpt-3.5-turbo* también por ser el más recomendado por esta compañía teniendo en cuenta sus capacidades y coste.

A todos estos modelos se les hizo *fine-tuning*, excluyendo a GPT-3.5 al no estar disponible la API que permite este procesamiento. En este caso se realizó *zero-shot prompting*.

Zero-shot prompting es una técnica utilizada en el contexto de los modelos de PLN, en particular los modelos lingüísticos a gran escala como la serie GPT de OpenAI. Se refiere al proceso de pedir al modelo que realice una tarea específica sin ningún *fine-tuning* previo de esa tarea ni proporcionar ejemplos del dominio de la tarea. Se espera que el modelo generalice a partir de su preentrenamiento para resolver la tarea utilizando únicamente la información presente en el *prompt*.

Para realizar *el zero-shot prompting*, se proporciona al modelo un *prompt* cuidadosamente elaborado que contiene suficiente contexto e instrucciones para guiar al modelo hacia el resultado deseado. El modelo genera entonces una respuesta basada en su conocimiento y comprensión preexistentes del lenguaje.

El *prompt* utilizado se construyó siguiendo las recomendaciones en la documentación de OpenAI. El diseño del *prompt* fue el siguiente:

> *Classify the text into one of the following topics: AUT, LEG, MWM, LEG, ENV, FIN, POL, PRN, COM, ING, ARC, MAT, HRM, CUL. Here is the definition of each topic:*
>
> - *AUT: Automotive, transportation and traffic regulations*
>
> ⋮
>
> - *LEG: Legal, law, HR, certificates and diplomas*
>
> ***Text***: {*text*}
> ***Category***:

Para obtener una versión de cada modelo con el *fine-tuning* sobre el *dataset* creado, primero se realizó búsqueda de hiper-parámetros en cada uno utilizando Optuna. Optuna es una biblioteca de optimización de hiper-parámetros de código abierto para *Python*.

Los parámetros a optimizar fueron los siguientes:

- *Batch*: valores entre 8 y 32.
- *Learning rate*: entre $1 * e^{-4}$ y $1 * e^{-5}$.
- *Epochs*: entre 2 y 5.

La búsqueda de hiper-parámetros para cada modelo se realizó utilizando de 10 a 25 experimentos en cada uno, y el mejor experimento por modelo se eligió utilizando el *loss* de validación, donde la validación representaba el 15 % del *dataset* de entrenamiento.

5.2 Comparación entre modelos

En esta sección primeramente se exponen datos de la optimización de hiper-parámetros para algunos modelos, mostrando gráficos que ilustran la importancia de los mismos y el espacio de búsqueda.

Luego se comparan los resultados obtenidos para los diferentes modelos entrenados, teniendo en cuenta la combinación de hiper-parámetros de menor *loss* de validación para cada uno.

Las métricas de evaluación para comparar los modelos fueron el *accuracy* para obtener una visión general, ya que el *dataset* de test, que consta de aproximadamente 7.000 segmentos, está balanceado. Además se tiene el cuenta la matriz de confusión para un análisis más profundo.

En la figura 5.1 se puede observar algunos datos de la búsqueda de hiper-parámetros en los modelos BERT, BART, XLNet y DistilRoBERTa, como la importancia de los parámetros y el contorno de búsqueda por pares de parámetros.

La importancia de parámetros se calculó utilizando *fANOVA* [23]. *fANOVA* entrena un modelo de regresión *random forest* que predice los valores objetivos de los experimentos completados dadas sus configuraciones de parámetros. Cuanto más preciso sea este modelo, más fiables serán las importancias evaluadas por esta clase.

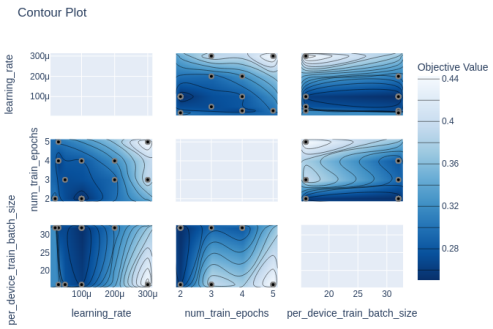
En todos los modelos, los parámetros más importantes no necesariamente coinciden. En el caso de DistilRoBERTa y BART, el más importante fue el *learning rate*, mientras que para BERT fue la cantidad de *epochs* de entrenamiento, y para XLNet fue el tamaño del *batch*.

Saber los parámetros más importantes en cada modelo es útil ya que se puede hacer una búsqueda más exhaustiva en los mismos, y lograr modelos de mayor calidad. Tener en cuenta que mientras más experimentos en la optimización de parámetros se realicen, más posibilidad de que la predicción del parámetro más importante sea correcta, ya que está basado en un modelo de regresión.

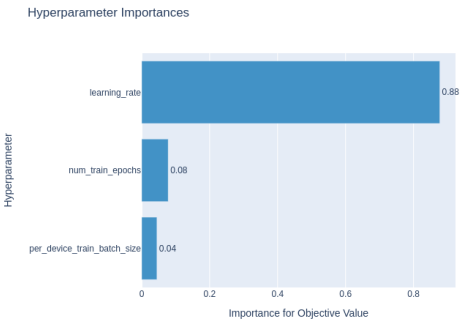
Los gráficos del contorno de búsqueda representan por cada par de parámetros, en color azul oscuro las áreas donde menor *loss* de validación (valor objetivo) se obtuvo, que es lo deseado, y lo contrario en azul claro. Por ejemplo, en DistilRoBERTa se puede observar que en contorno alrededor del *learning rate* con valor $1 * e^{-4}$ y 2 *epochs* de entrenamientos es el de color más oscuro, y por tanto el de menor valor del *loss* de validación.

Ahora bien, luego de finalizados los experimentos para la optimización de hiper-parámetros, se entrenaron los modelos finales con los mejores parámetros en cada modelo, y se evaluaron utilizando el *dataset* de test con la métrica *accuracy*, como se menciona anteriormente.

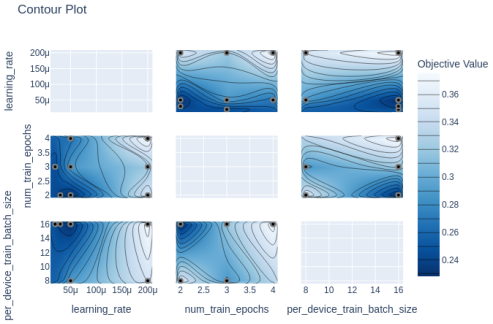
En la tabla 5.1 se puede observar el *accuracy* y tiempo de predicción obtenido por cada modelo mencionado anteriormente, incluyendo además los modelos GPT, a los que no se le realizó búsqueda de hiper-parámetros. Los experimentos se realizaron en un servidor con 66 GB de RAM, un procesador Intel(R) Core(TM) i9-7900X, y 2 GPUs NVIDIA TITAN Xp de 12 GB.



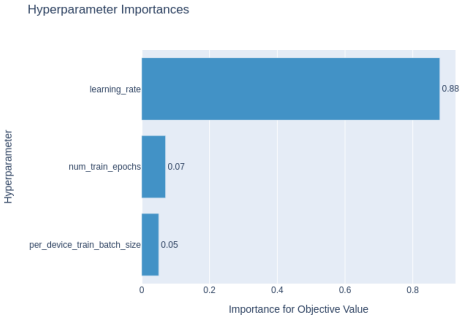
(a) DistilRoBERTa contornos de búsqueda



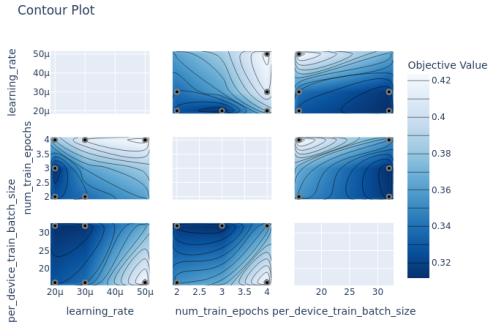
(b) DistilRoBERTa importancia de parámetros



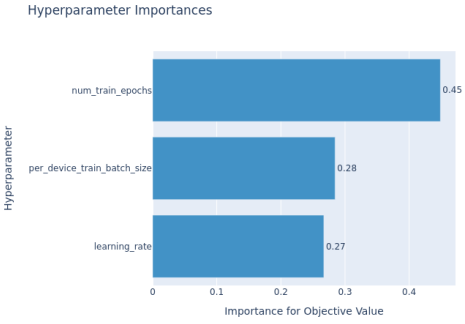
(c) BART contornos de búsqueda



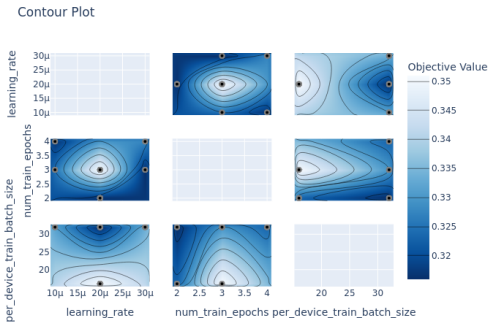
(d) BART importancia de parámetros



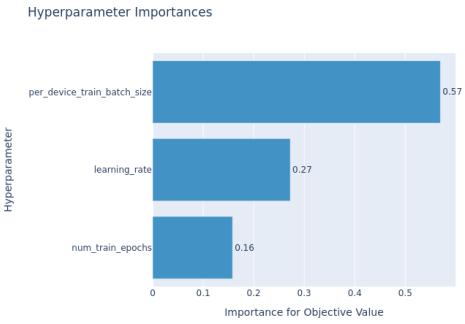
(e) BERT contornos de búsqueda



(f) BERT importancia de parámetros



(g) XLNet contornos de búsqueda



(h) XLNet importancia de parámetros

Figura 5.1: Búsqueda de hiper-parámetros en varios modelos.

Modelo	Accuracy	Tiempo predicción (seg)
BERT	0,9111	88
DistilBERT	0,9122	**46**
RoBERTa	0,9370	157
DistilRoBERTa	0,9304	84
ALBERT	0,8980	216
ELECTRA	0,8861	58
XLNet	0,9185	137
BART	0,9401	189
GPT-3	**0,9440**	-
GPT-3.5	0,6700	-

Tabla 5.1: Comparación entre los modelos de clasificación.

Clase	BART	RoBERTa	DistilRoB	BERT	DistBERT	ELECT	XLNet	GPT-3	ALB
ARC	**0,9562**	0,9511	0,9364	0,9446	0,9477	0,9246	0,938	0,9528	0,9344
AUT	0,9274	0,9359	0,9238	0,9084	0,9116	0,8791	0,928	**0,9437**	0,9019
COM	0,9418	0,9418	0,9405	0,8968	0,8858	0,8673	0,9217	**0,9542**	0,8858
CUL	**0,9618**	0,9494	0,941	0,9189	0,9206	0,8943	0,9365	0,9591	0,9187
ENV	**0,9544**	0,9487	0,934	0,9237	0,9254	0,9002	0,9292	0,9538	0,9
FIN	**0,8612**	0,8503	0,8366	0,7935	0,8022	0,7732	0,8478	0,8466	0,8157
HRM	0,8938	0,8853	0,8659	0,8691	0,8634	0,8107	0,8528	**0,9074**	0,8233
ING	0,9518	0,9463	0,9431	0,9012	0,8967	0,8556	0,9006	**0,9645**	0,8734
LEG	0,9730	0,9691	0,9728	0,968	0,9657	0,9352	0,9638	**0,9788**	0,9405
LSM	0,9390	0,9486	0,9366	0,9288	0,9299	0,8958	0,9211	**0,9523**	0,914
MAT	0,9960	0,997	0,9950	0,9489	0,9484	0,9082	0,9364	**0,9970**	0,9223
MWM	0,9494	0,9577	0,961	0,9332	0,9395	0,9287	0,9462	**0,9626**	0,9386
POL	**0,8695**	0,8608	0,8545	0,844	0,8463	0,8164	0,8582	0,8574	0,829
PRN	**0,9901**	0,9859	0,9869	0,9869	0,986	0,9808	0,9807	0,9900	0,9801

Tabla 5.2: F1 score para cada modelo y dominio.

En la tabla 5.2 se puede encontrar información más detallada de cómo se comportó cada dominio en cada modelo. Como se puede observar, el mejor F1 en cada dominio se logró siempre en los 2 modelos de mejor *accuracy* en la tabla 5.1, o sea, BART y GPT-3. También se puede apreciar que las clases con los mejores resultados por dominio fueron **MAT** y **PRN**, mientras que las que tienen más problemas en clasificarse son **FIN**, **POL** y **HRM**.

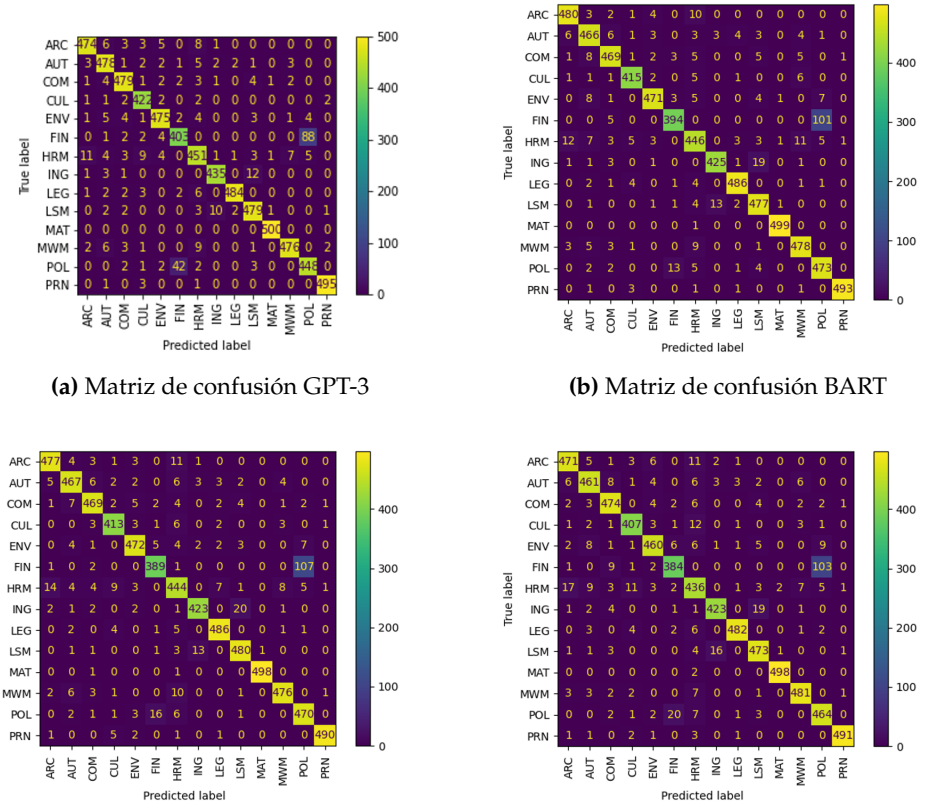
Como se puede observar, el modelo que obtuvo mejor *accuracy* fue GPT-3 y el más rápido en realizar las predicciones fue DistilBERT. La elección de cuál es el mejor modelo es relativa y depende de varios factores, como el caso de uso, limitaciones computacionales, limitaciones de tiempo, costo, entre otros. En el problema específico que trata este trabajo, las limitaciones de tiempo son importantes, porque es necesario un modelo que no tarde mucho, ya que se realizará la clasificación sobre millones de datos.

El modelo basado en GPT-3 es el que obtuvo mejores resultados, pero debido a que la inferencia es a través de la API de OpenAI, no se puede determinar un tiempo específico fiable para tener en cuenta los requerimientos de tiempo en predicción. Además, que esta API no es gratis, en el caso del modelo específico de GPT-3 utilizado el precio es de $0,0004 por cada 1K *tokens*, que aunque no es para nada el modelo más caro de OpenAI, sigue siendo un factor bastante determinante. También tener en cuenta las cuestiones de privacidad al usar un modelo que para hacer *fine-tuning* e inferencia es necesario compartir los datos con terceros, lo que puede ser particularmente problemática si se quieren

clasificar datos de clientes. Por tanto, GPT-3 quedaría descartado por estos motivos como el modelo seleccionado.

Los próximos mejores son BART, RoBERTa y DistilRoBERTa. BART obtuvo un *accuracy* de 0,9401, bastante cerca del modelo de GPT-3, pero tiene la desventaja de ser el que más demora en hacer las predicciones, luego de ALBERT, por tanto, queda descartado al no cumplir con las expectativas de tiempo. Luego, respecto a RoBERTa y DistilRoBERTa, ambos obtuvieron valores de *accuracy* bastante cercanos, pero DistilRoBERTa, al ser una versión más ligera de RoBERTa, es mucho más rápido en realizar las predicciones, por tanto es el modelo seleccionado al tener un balance entre *accuracy* y tiempo.

Finalizado el análisis de cuál modelo es el más adecuado para el problema que se quiere resolver, es importante estudiar en mayor profundidad cómo funcionaron, ya que como se mencionó en capítulos anteriores, la información que da el *accuracy* no es suficiente para determinar la calidad. Por este motivo, analizamos la matriz de confusión de algunos de estos modelos para observar el comportamiento entre las diferentes clases.



(a) Matriz de confusión GPT-3

(b) Matriz de confusión BART

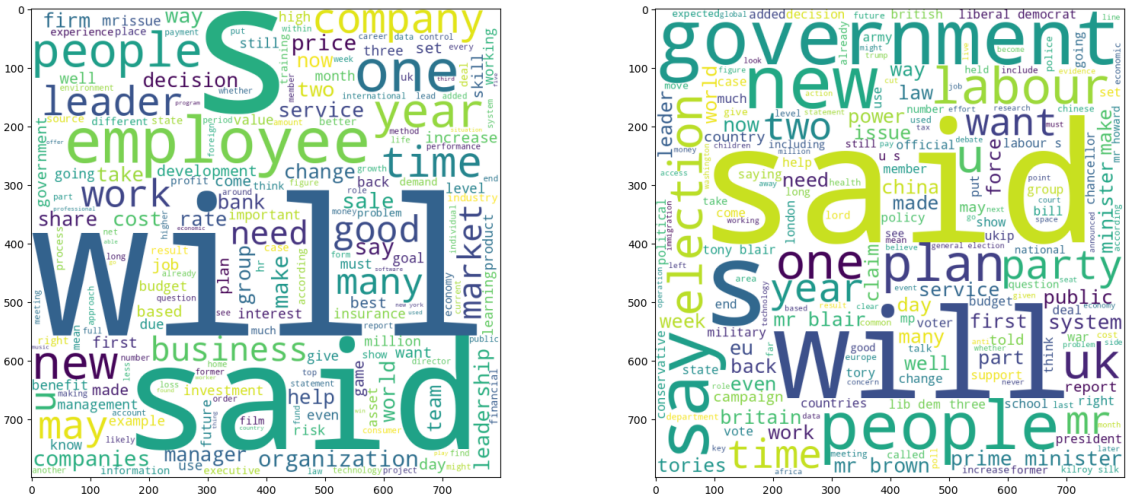
(c) Matriz de confusión RoBERTa

(d) Matriz de confusión DistilRoBERTa

Figura 5.2: Matrices de confusión de los modelos de mejor *accuracy*.

Como se puede observar en la figura 5.2, en casi todas las clases la cantidad de segmentos categorizados equivocadamente es razonable, excepto en el par de clases **FIN**-**POL**. En las 4 matrices de confusión la cantidad de segmentos de **FIN** etiquetados como **POL** es bastante grande, aproximadamente sobre los 100 segmentos de los 500 de test para la clase **FIN**. Esta situación aparenta un problema en el *dataset* para estas clases.

Observando los *wordclouds* de **FIN** y **POL** en la figura 5.3, se nota que existen bastantes palabras que se repiten en ambas clases, lo que puede provocar que los modelos no logren clasificar correctamente los textos afectados. Luego de identificar este problema en el *dataset*, queda como trabajo futuro realizar mejoras en los segmentos de **FIN** y **POL**.



(a) FIN *wordcloud* **(b) POL** *wordcloud*

Figura 5.3: *Wordclouds* de **FIN** y **POL**.

En la tabla 5.1, llama la atención que GPT-3.5, modelo de alta calidad con capacidades en clasificación *zero-shot* muy potentes, obtenga resultados tan pobres. Esto probablemente se debe a que muchos segmentos de test fueron clasificados con etiquetas que no existen entre las 15 definidas en este trabajo, lo cual afecta muy negativamente al *accuracy*. Lo ideal era volver a clasificar estos segmentos, lo cual no se hizo por varias razones. La predicción tardaba, y había que realizar varios intentos debido a la alta demanda que está teniendo este modelo ahora mismo. Además, el precio de GPT-3.5 es de $0,002 por cada mil tokens, lo cual limitaba la cantidad de pedidos a la API de OpenAI por cuestiones de coste.

5.3 Experimentos con diferentes *datasets*

La aparición de grandes modelos preentrenados, como GPT-3, ha revolucionado el campo del PLN y la inteligencia artificial. Estos modelos son conocidos por su capacidad para aprender de grandes cantidades de datos y generar resultados de alta calidad, lo que los hace ideales para una amplia gama de aplicaciones. Un aspecto interesante de estos modelos es su capacidad para ajustarse a conjuntos de datos relativamente pequeños, lo que les permite adaptarse a nuevas tareas sin necesidad de crear *datasets* muy grandes. Por este motivo, se realizaron experimentos para analizar el impacto de la variación del tamaño del *dataset* en el rendimiento de modelo seleccionado en la sección anterior, DistilRoBERTa.

Para esto se obtuvieron 3 versiones nuevas del *dataset* construido originalmente, disminuyendo el tamaño del mismo. Esto se logró haciendo muestreo aleatorio del *dataset* original. Se realizó un muestreo de 1.500, 1000 y 500 segmentos por clase. Luego, por cada *dataset* creado, se entrenó un modelo y se evaluó sobre el mismo test utilizado en la sección de experimentación con diferentes modelos.

Los resultados de estos experimentos se pueden observar en la tabla 5.3. El *accuracy* en los modelos fue directamente proporcional a la cantidad de datos utilizados en entrenamiento. Pero se puede observar que la diferencia entre utilizar 2000 segmentos y 1500 segmentos por dominio fue mínima, pero utilizar menos que eso si afecta en mayor medida a la calidad del modelo.

Segmentos por dominio	*Accuracy*
500	0,8950
1000	0,9170
1500	0,9291
2000	0,9304

Tabla 5.3: Comparación del modelo DistilRoBERTa con diferentes tamaños de *dataset*.

5.4 Dominio GEN

Como se mencionó anteriormente, es poco fiable añadir la clase **GEN** en el *dataset*. Esta clase, esencialmente, representa todo lo que no se incluya en los dominios restantes, y crear un dataset de esta etiqueta que sea representativo es muy complicado. La solución ideada fue entrenar un modelo con 14 clases sin incluir datos de **GEN**, y luego encontrar un *threshold* tal que, si la clase de mayor probabilidad es menor que ese *threshold*, entonces el segmento se etiqueta como **GEN**.

Para encontrar este *threshold* se realizaron experimentos que se pueden observar en el gráfico 5.4. Se probaron *thresholds* con valores entre 0,9 y 0,5, y se analizó si un pequeño conjunto de datos genéricos eran clasificados correctamente o no utilizando estos valores, y cómo se afectaban los datos del *dataset* de test original.

Estos experimentos se realizaron sobre el modelo BART, que tenía un *accuracy* de 0,9401 en el *dataset* de test original. La idea es encontrar un *threshold* que cuando se reclasifiquen los datos teniendo en cuenta la condición mayor-que, el *accuracy* del modelo no baje más de 0,005, y este límite está marcado en la línea vertical roja.

Como se puede apreciar en el gráfico, esto se logra con un *threshold* de 0,55, donde se logran identificar el 22 % de los datos genéricos. Lo más importante es que el modelo logre identificar correctamente a los dominios no genéricos, porque al ser tan completa la definición de las clases no existen muchos textos que no pertenezcan a al menos una de las 14 clases.

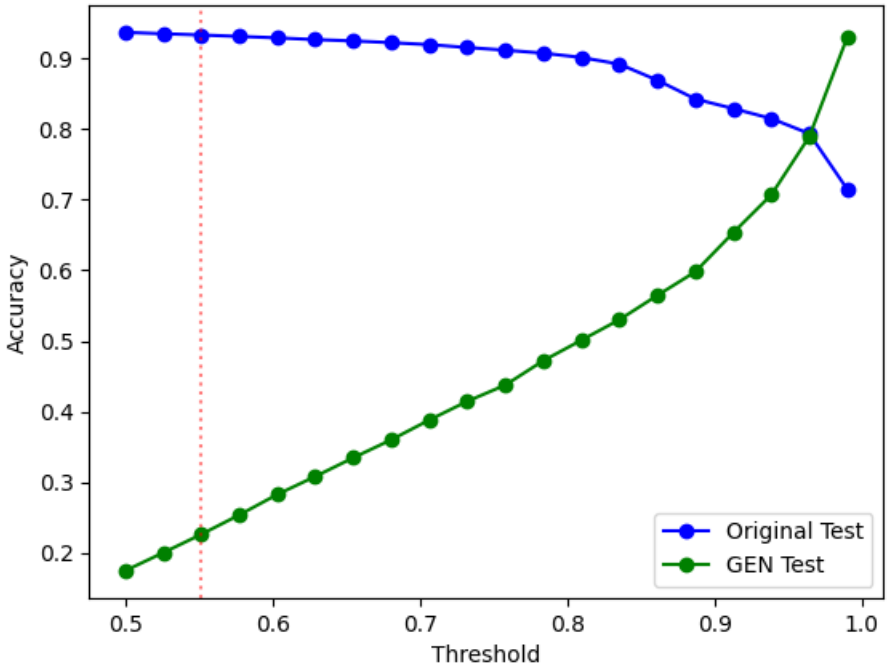


Figura 5.4: Experimentos con la clase **GEN**

CAPÍTULO 6
Conclusiones

Con este trabajo se logró la creación de un modelo de clasificación en 15 dominios y la creación de un *dataset* para entrenamiento y test, utilizando modelos de lenguaje preentrenados del estado del arte y técnicas de *web crawling*. Este capítulo resume las conclusiones obtenidas para lograr esto, así como las posibles mejoras y ampliaciones a realizar como trabajo futuro.

6.1 Conclusiones

Con este trabajo, concretamente en el Capítulo 2, se logró una comprensión de cómo funcionan los modelos de lenguaje preentrenados, lo cuál es fundamental ya que estos están liderando el estado del arte y las noticias tecnológicas en los últimos meses, particularmente los modelos GPT. Además, se logró ver las diferencias entre ellos, lo que permite seleccionar los modelos que probablemente funcionan mejor para diferentes tareas de procesamiento de texto.

Se logró comprender que un modelo con mayor precisión puede parecer inicialmente más atractivo, pero no siempre es la opción más adecuada si se tienen en cuenta las limitaciones de tiempo y costes.

Un modelo más preciso suele ser más complejo, lo que puede conllevar tiempos de entrenamiento e inferencia más largos. Esto puede no ser factible para aplicaciones con estrictas restricciones de tiempo o recursos informáticos limitados. Un modelo más sencillo con una precisión ligeramente inferior podría ser más eficaz y rentable en estos casos.

Por lo tanto, si se quiere el modelo con más *accuracy* lo mejor es utilizar GPT-3 o BART. No obstante, si existen limitaciones de tiempo, corte y recursos computacionales, lo mejor es ir por el modelo DistilRoBERTa, ya fue el que logró un mejor balance entre *accuracy* y tiempo de predicción. Tener en cuenta que es muy complicado comparar los resultados obtenidos con el estado del arte de clasificación de dominios visto en el Capítulo 3, ya que la definición de clases y el *dataset* utilizado para el entrenamiento no coinciden.

Además, se logró ver la relación entre *accuracy* y tamaño del *dataset*, donde estos dos valores son directamente proporcionales, y que no hace mucha diferencia entre tener 2000 segmentos por clase o 1500 en la calidad del modelo.

Otra de las conclusiones es que el modelo que se utilice no necesariamente es el que dicta la calidad de los resultados, ya que si existe algún problema con la calidad del *dataset*, ni los modelos más potentes pueden arreglarlo. Esto se observó con las clases **FIN** y **POL**, que tenían similaridades en la distribución de palabras, lo que impacta negativamente en los resultados de los modelos.

6.2 Trabajo futuro

Aunque el trabajo realizado ha logrado sus objetivos y se ha obtenido un modelo de clasificación de buena calidad, se pueden seguir trabajando con el objetivo de obtener un modelo aún mejor. Las mejoras propuestas como trabajo futuro son las siguientes:

- Como el *dataset* es de los elementos más importante, la primera propuesta de trabajo futuro consiste en rehacer las clases **FIN** y **POL**, que se determinó que tenían problemas.
- Extender el *dataset* para que los dominios sean aún más representativos y contengan datos de varios estilos, ya sean redes sociales, manuales, emails, noticias, entre otros. De esta manera evitar que existan dominios con datos de un solo estilo, y los modelos solo logren clasificar correctamente textos de un dominio si corresponden con el estilo de los datos de entrenamiento.
- Dado de muchos de los segmentos se obtuvieron utilizando *web crawling*, puede suceder que un texto dentro de un artículo de un dominio no corresponda con este, ya que en un artículo largo se pueden tratar varios temas. Para evitar datos ruidosos como este, se propone hacer una revisión manual del *dataset*. Otra idea para resolver este potencial problema, sería utilizar un clasificador de dominios de alta calidad entrenado con otras clases, hacer la equivalencia entre las clases de este clasificador y las propuestas en este trabajo, y de esta forma sustituir el etiquetado humano, que puede consumir mucho tiempo y aumentar los costes, sobre todo si se expande el *dataset*.
- Realizar experimentos con las versiones más grandes de los modelos utilizados, con el objetivo de aumentar el *accuracy* del clasificador.
- Experimentación con modelos que no se hayan utilizado, y modelos liberados recientemente.
- Realizar una búsqueda de hiper-parámetros más exhaustiva, haciendo más experimentos en los parámetros de mayor importancia en cada modelo.
- Realizar experimentos con otros diseños de *prompt* en el caso de GPT-3.5.
- Crear un modelo basado en clases con estructura jerárquica. De esta manera no serían clases estáticas, sino dinámicas y más flexibles, que se puedan utilizar para otros casos de uso y con otros requerimientos.

Bibliografía

[1] Emily Alsentzer, John Murphy, William Boag, Wei-Hung Weng, Di Jindi, Tristan Naumann, and Matthew McDermott. Publicly available clinical BERT embeddings. In *Proceedings of the 2nd Clinical Natural Language Processing Workshop*, pages 72–78, Minneapolis, Minnesota, USA, June 2019. Association for Computational Linguistics.

[2] Dogu Araci. Finbert: Financial sentiment analysis with pre-trained language models. *arXiv preprint arXiv:1908.10063*, 2019.

[3] Iz Beltagy, Arman Cohan, Robert Logan IV, Sewon Min, and Sameer Singh. Zero- and few-shot NLP with pretrained language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, pages 32–37, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[4] Iz Beltagy, Kyle Lo, and Arman Cohan. SciBERT: A pretrained language model for scientific text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3615–3620, Hong Kong, China, November 2019. Association for Computational Linguistics.

[5] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. *Advances in neural information processing systems*, 13, 2000.

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3(null):1137–1155, mar 2003.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[8] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. Sparks of artificial general intelligence: Early experiments with gpt-4, 2023.

[9] Amit Chaudhary. Visual paper summary: Albert (a lite bert), 2020. https://amitness.com/2020/02/albert-visual-summary.

[10] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning. What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*, 2019.

[11] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

[12] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(ARTICLE):2493–2537, 2011.

[13] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. *Advances in neural information processing systems*, 28, 2015.

[14] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, and Quoc V Le. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, 2019.

[15] Mita K Dalal and Mukesh A Zaveri. Automatic text classification: a technical review. *International Journal of Computer Applications*, 28(2):37–40, 2011.

[16] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. http://web.archive.org/save/http://Skylion007.github.io/OpenWebTextCorpus, 2019.

[19] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[20] Geoffrey E. Hinton, James L. McClelland, and David E. Rumelhart. Distributed representations. In *The Philosophy of Artificial Intelligence*, 1986.

[21] Mohammad Hossin and Md Nasir Sulaiman. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, 5(2):1, 2015.

[22] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[23] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *International conference on machine learning*, pages 754–762. PMLR, 2014.

[24] Sarthak Jain and Byron C. Wallace. Attention is not explanation, 2019.

[25] Rie Johnson and Tong Zhang. Deep pyramid convolutional neural networks for text categorization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 562–570, 2017.

[26] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[27] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[28] T Karthikeyan, Karthik Sekaran, D Ranjith, JM Balajee, et al. Personalized content extraction and text classification using effective web scraping techniques. *International Journal of Web Portals (IJWP)*, 11(2):41–52, 2019.

[29] Daniel Martin Katz, Michael James Bommarito, Shang Gao, and Pablo Arredondo. Gpt-4 passes the bar exam. *Available at SSRN 4389233*, 2023.

[30] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

[31] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[32] Ken Lang. Newsweeder: Learning to filter netnews. In Armand Prieditis and Stuart Russell, editors, *Machine Learning Proceedings 1995*, pages 331–339. Morgan Kaufmann, San Francisco (CA), 1995.

[33] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, 2020.

[34] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. Dbpedia large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2):167–195, 2015.

[35] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[36] Bingbing Li, Zhenglun Kong, Tianyun Zhang, Ji Li, Zhengang Li, Hang Liu, and Caiwen Ding. Efficient transformer-based large scale language representations using hardware-friendly block structured pruning, 2020.

[37] Paul Pu Liang, Chiyu Wu, Louis-Philippe Morency, and Ruslan Salakhutdinov. Towards understanding and mitigating social biases in language models. In *International Conference on Machine Learning*, pages 6565–6576. PMLR, 2021.

[38] Tao Liu, Zheng Chen, Benyu Zhang, Wei-ying Ma, and Gongyi Wu. Improving text classification using local latent semantic indexing. In *Fourth IEEE International Conference on Data Mining (ICDM'04)*, pages 162–169. IEEE, 2004.

[39] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[40] Aditya Malte and Pratik Ratadiya. Evolution of transfer learning in natural language processing, 2019.

[41] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. *Advances in neural information processing systems*, 30, 2017.

[42] Sabrina J. Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y. Lee, Benoît Sagot, and Samson Tan. Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp, 2021.

[43] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[44] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[45] Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. Deep learning-based text classification: a comprehensive review. *ACM computing surveys (CSUR)*, 54(3):1–40, 2021.

[46] Sebastian Nagel. Cc-news. http://web.archive.org/save/http://commoncrawl.org/2016/10/news-dataset-available, 2016.

[47] OpenAI. Gpt-4 technical report, 2023.

[48] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

[49] Irina Pak and Phoey Lee Teh. Text segmentation techniques: a critical review. *Innovative Computing, Optimization and Its Applications: Modelling and Simulations*, pages 167–181, 2018.

[50] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[51] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[52] ME Peters, M Neumann, M Iyyer, M Gardner, C Clark, K Lee, and L Zettlemoyer. Deep contextualized word representations. arxiv 2018. *arXiv preprint arXiv:1802.05365*, 12, 2018.

[53] David M. W. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation, 2020.

[54] Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, 63(10):1872–1897, 2020.

[55] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[56] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[57] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2020.

[58] Hao Ren and Hong Lu. Compositional coding capsule network with k-means routing for text classification. *Pattern Recognition Letters*, 160:1–8, 2022.

[59] Devendra Singh Sachan, Manzil Zaheer, and Ruslan Salakhutdinov. Revisiting lstm networks for semi-supervised text classification via mixed objective function. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6940–6948, 2019.

[60] Nipun Sadvilkar and Mark Neumann. Pysbd: Pragmatic sentence boundary disambiguation, 2020.

[61] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[62] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[63] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *Chinese Computational Linguistics: 18th China National Conference, CCL 2019, Kunming, China, October 18–20, 2019, Proceedings 18*, pages 194–206. Springer, 2019.

[64] Trieu H. Trinh and Quoc V. Le. A simple method for commonsense reasoning. *arXiv preprint arXiv:1806.02847*, 2018.

[65] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[66] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[67] Guoyin Wang, Chunyuan Li, Wenlin Wang, Yizhe Zhang, Dinghan Shen, Xinyuan Zhang, Ricardo Henao, and Lawrence Carin. Joint embedding of words and labels for text classification. *arXiv preprint arXiv:1805.04174*, 2018.

[68] Haifeng Wang, Jiwei Li, Hua Wu, Eduard Hovy, and Yu Sun. Pre-trained language models and their applications. *Engineering*, 2022.

[69] Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019.

[70] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

[71] Chuhan Wu, Fangzhao Wu, Tao Qi, Yongfeng Huang, and Xing Xie. Noisytune: A little noise can help you finetune pretrained language models better, 2022.

[72] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.

[73] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[74] Qizhe Xie, Zihang Dai, Eduard Hovy, Thang Luong, and Quoc Le. Unsupervised data augmentation for consistency training. *Advances in neural information processing systems*, 33:6256–6268, 2020.

[75] Z Yang, Z Dai, Y Yang, J Carbonell, RR Salakhutdinov, and XLNet Le QV. generalized autoregressive pretraining for language understanding; 2019. *Preprint at https://arxiv. org/abs/1906.08237 Accessed June*, 21, 2021.

[76] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

[77] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 7370–7377, 2019.

[78] Junjie Ye, Xuanting Chen, Nuo Xu, Can Zu, Zekai Shao, Shichun Liu, Yuhan Cui, Zeyang Zhou, Chao Gong, Yang Shen, Jie Zhou, Siming Chen, Tao Gui, Qi Zhang, and Xuanjing Huang. A comprehensive capability analysis of gpt-3 and gpt-3.5 series models, 2023.

[79] Wen Zhang, Taketoshi Yoshida, and Xijin Tang. Text classification using multi-word features. In *2007 IEEE International Conference on Systems, Man and Cybernetics*, pages 3519–3524. IEEE, 2007.

[80] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

[81] Zhengyan Zhang, Yuxian Gu, Xu Han, Shengqi Chen, Chaojun Xiao, Zhenbo Sun, Yuan Yao, Fanchao Qi, Jian Guan, Pei Ke, et al. Cpm-2: Large-scale cost-effective pre-trained language models. *AI Open*, 2:216–224, 2021.

[82] Wei Zhao, Jianbo Ye, Min Yang, Zeyang Lei, Suofei Zhang, and Zhou Zhao. Investigating capsule networks with dynamic routing for text classification. *arXiv preprint arXiv:1804.00538*, 2018.