UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Generación automática de resúmenes de textos utilizando técnicas de puesta a punto de modelos de lenguaje pre-entrenados

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Carpena Caicedo, Moisés

Tutor/a: Casacuberta Nolla, Francisco

CURSO ACADÉMICO: 2023/2024

Resumen

El objetivo de este trabajo es desarrollar un sistema que genere automáticamente resúmenes de documentos utilizando técnicas de aprendizaje automático, concretamente grandes modelos de lenguaje neuronales. Para lograr esto, utilizamos un enfoque que se base en la puesta a punto "fine-tuning" de modelos de lenguaje pre-entrenados de tipo Transformer, para adaptarlo a la tarea específica de generación de resúmenes.

Trabajamos con un corpus de textos previamente procesado y preparado que sirve para entrenar el modelo y ayudar a que aprenda las características y estructuras necesarias.

Posteriormente, se estudia una tarea sobre la que hacer experimentos y pruebas con este sistema, más concretamente una aplicación móvil, permitiendo a los usuarios cargar documentos y obtener resúmenes de ellos de manera automática.

Palabras clave: resumen, fine-tuning, Transformer, modelo.

Abstract

The goal of this work is to develop a system that automatically generates document summaries using machine learning techniques, specifically large neural language models. To achieve this, we use an approach that is based on the fine-tuning of pre-trained Transformer-type language models, to adapt it to the specific task of summary generation.

We work with a previously processed and prepared corpus of texts that is used to train the model and help it learn the necessary characteristics and structures.

Subsequently, a task is studied on which to carry out experiments and tests with this system, more specifically a mobile application, allowing users to upload documents and obtain summaries of them automatically.

Key words : summary, fine-tuning, Transformer, model.

Resum

L'objectiu d'este treball és desenvolupar un sistema que genere automàticament resums de documents utilitzant tècniques d'aprenentatge automàtic, concretament grans models de llenguatge neuronals. Per a aconseguir això, utilitzem un enfocament que es base en la posada a punt “fine-*tuning” de models de llenguatge *pre-entrenats de tipus *Transformer, per a adaptar-ho a la tasca específica de generació de resums.

Treballem amb un corpus de textos prèviament processat i preparat que servix per a entrenar el model i ajudar al fet que aprenga les característiques i estructures necessàries.

Posteriorment, s'estudia una tasca sobre la qual fer experiments i proves amb este sistema, més concretament una aplicació mòbil, permetent als usuaris carregar documents i obtindre resums d'ells de manera automàtica.

Paraules clau: resum, fine-tuning, Transformer, model.

Tabla de contenidos

1. Introducción

La idea que vamos a tratar en este trabajo será la generación automática de resúmenes de texto. Para ello, utilizaremos la técnica de puesta a punto para modelos pre-entrenados, esto es, el uso de modelos de lenguaje ya existentes que han sido entrenados previamente para abarcar una amplia cantidad de tareas generales en el campo del lenguaje natural.

Hemos decidió abordar la problemática de la generación automática de texto debido al aumento exponencial de la información disponible en línea. Debido a esto, es necesario sintetizar grandes volúmenes de texto en resúmenes concisos y precisos, ya que se ha vuelto una herramienta esencial para mejorar la eficiencia en muchos ámbitos de la gestión de información. Esta investigación no solo busca mostrar y enseñar los modelos actuales de generación de texto, sino también optimizarlos en contextos específicos, como puede ser el multilingüe, dado que nuestros datos se encuentras disponibles en inglés y deberemos trabajar tanto en pre-entrenamiento como post-entrenamiento con ellos. Resolver este problema es vital, ya que permite a los usuarios y profesionales acceder rápidamente a la información relevante, facilitando así la toma de decisiones y reduciendo el tiempo de uso perdido.

Para esto, utilizaremos la arquitectura Transformers debido a su gran abanico de posibilidades a la hora de entrenar modelos en tareas específicas. Ya que existen multitud de modelos que trabajan con esta arquitectura, escogeremos una de las más usadas, GPT-2.

Este modelo destaca por su facilidad de compresión y uso, así como su gran potencial a la hora de generar texto coherente, diverso y creativo. Por lo cual, éste es un candidato perfecto para nuestra tarea de resúmenes de texto.

Para nuestra puesta a punto, necesitaremos grandes corpus de datos que harán posible el aprendizaje de nuestro modelo en la tarea que le especifiquemos. En este caso, usaremos corpus de noticias del CNN y Daily Mail, ya que abarca muchos ámbitos y nuestro modelo podrá reconocer el mayor número de patrones y significados.

Haremos pruebas de generación, en donde veremos como de preciso es a la hora de recoger la información esencial del texto de entrada y si el resumen generado es coherente para el usuario. Para perfeccionar este apartado, iremos jugando con las distintas variables de entrada y especificaciones a la hora de la puesta a punto.

Pondremos en práctica nuestro modelo fine-tuned dentro de una aplicación móvil, que será capaz de recoger un documento de texto que pasemos y devolvernos otro con su respectivo resumen. Podremos elegir varios parámetros a la hora de la generación para acomodarse a los gustos del usuario.

Este trabajo se divide la siguiente manera: El primer capítulo se centrará en la introducción, es decir la motivación y los objetivos del trabajo. El segundo capítulo será el más extenso y hablaremos del estado del arte/de la cuestión, que engloba toda la historia de los resúmenes automáticos. En el tercer capítulo tendremos los fundamentos teóricos del trabajo, tanto definiciones como fórmulas importantes. El cuarto nos centraremos en la metodología. El

quinto capítulo veremos los corpus elegidos y su explicación. En el sexto hablaremos del desarrollo experimental del modelo, tanto el entrenamiento como la evaluación y resultados. El séptimo hablará de la aplicación móvil y la implementación del modelo ajustado a ella. En el octavo capítulo serán los resultados generales del trabajo y discutiremos tanto desafíos encontrados como futuras investigaciones. Y por último en el noveno capítulo trataremos las conclusiones.

1.1. Motivación

Tras haber establecido el contexto general y los puntos en los que se trabajará, es crucial comprender la motivación por la cual se impulsa esta investigación. El principal motivo es la admiración hacia la Inteligencia Artificial y su funcionamiento, más concretamente la generación de texto y la capacidad de la IA de comprender lo que está escribiendo.

Debido al auge de éste en la última década y más concretamente, en los últimos años, multitud de personas, comunidades y empresas, han conseguido grandes avances en la ardua tarea de crear una inteligencia con la capacidad de pensar y razonar de manera autónoma. Esto nos proporciona un abanico de opciones todavía más grande a la hora de generar y crear herramientas, además de facilitarnos las tareas y permitirnos automatizarlas. Sin ir más lejos, el campo de las redes neuronales está siendo uno de los más explotados en estos momentos y con los que más avances estamos viendo.

Por supuesto, éste no es el único campo de interés que existe en el ámbito de la Inteligencia Artificial, previos estudios también han demostrado avances significativos en áreas como la traducción automática de texto, generación de diálogos o síntesis de resúmenes. Gracias a todo esto, se ha podido allanar el camino para investigaciones adicionales, como en la presente, que busca explorar y mejorar técnicas de generación de resúmenes utilizando modelos pre-entrenados.

1.2. Objetivos

El objetivo principal es entrenar un modelo pre-entrenado mediante técnicas de puesta a punto (fine-tuning) para la tarea específica de generación de resúmenes de texto. Para ellos se plantean los siguientes objetivos específicos:

- Recopilación de datos: se recopilarán conjunto de datos extensos sobre temas varios, como noticias, entrevistas y opiniones. Estos conjuntos de datos, conocidos como corpus, servirán para entrenar nuestro modelo.
- Selección de modelos pre-entrenados: Se investigarán y recopilarán diversos modelos con arquitectura Transformer, que ha demostrado ser de las más efectivas en generación de texto coherente.
- Pruebas y evaluación de modelos: se realizarán pruebas con los modelos anteriores en tareas menos demandantes, como puede ser generación de texto continuo, así comprenderemos su funcionamiento interno y podremos determinar cual ofrece los mejores resultados.

Una vez completadas las etapas previas, deberemos escoger aquel modelo que mejores resultados nos haya mostrado, en este caso veremos que será el modelo GPT-2 y procederemos con el fine-tuning específico para nuestra tarea.

Además del objetivo principal, trabajaremos en la implementación de una aplicación móvil en la que podremos introducir un documento PDF, y nos devolverá su resumen correspondiente.

2. Estado del arte/de la cuestión

(Jin, Zhang, Meng, Wang, & Tan, 2024)

La generación automática de resúmenes de texto es una técnica muy importante en el campo del procesamiento del lenguaje natural, que tiene como objetivo condensar la información contenida en un documento extenso para hacerla más breve y coherente. Este proceso es valioso en la sobrecarga de información, ya que la capacidad de acceder rápidamente a la información esencial en grandes cantidades de texto puede mejorar significativamente la eficiencia en la toma de decisiones o la gestión del conocimiento.

Tradicionalmente, los métodos de generación de resúmenes se han dividido en dos categorías principales: extractivos y abstractivos. Los extractivos seleccionan frases o sentencias directamente del texto original para construir el resumen. Aunque estos métodos aseguran la precisión del contenido al mantener frases exactas del documento fuente, a menudo producen resúmenes que carecen de coherencia y fluidez debido a la falta de conexión lógica entre sus partes.

Por otro lado, los métodos abstractivos generan nuevas frases que capturan la esencia original del texto, proporcionando así, resúmenes más naturales y coherentes similares a los que podría escribir un ser humano. Sin embargo, estos métodos enfrentan desafíos técnicos considerables, ya que se requiere una comprensión profunda del texto, así como la capacidad de reformular la información de manera efectiva. Estos desafíos incluyen mantener la coherencia temática y asegurar la precisión de los datos resumidos.

Con el avance de las tecnologías de aprendizaje profundo, los modelos de lenguaje pre-entrenados han transformado el panorama del lenguaje natural. La introducción de los modelos basados en la arquitectura Transformer ha sido muy impactante. Estos modelos, como BERT (Bidirectional Encoder Representations from Transformers) o GPT2 (Generative Pre-trained Transformer), han demostrado grandes capacidades en diversas tareas de compresión y generación de texto. Esta arquitectura destaca por su capacidad de capturar dependencias a largo plazo en el texto, gracias a su mecanismo de atención, que permite ponderar la importancia de diferentes partes del texto de entrada al generar cada palabra de salida.

GPT-2, desarrollado por OpenAI, es uno de los modelos pre-entrenados más avanzados disponibles en la actualidad. Gracias que ha sido entrenado en un extenso corpus de datos textuales, le permite generar texto coherente y contextualmente relevante. GPT-2, basada casi en su totalidad en mecanismos de atención, ha demostrado un potencial significativo para tareas abstractivas de generación de resúmenes. La arquitectura Transformer sobre la que se basa, permite manejar dependencias a largo plazo del texto, esto es, que guarda toda información importante durante todo el texto sin pérdidas importantes, por lo que es capaz de producir resúmenes de alta calidad.

El gran potencial de GPT-2 para la generación automática de resúmenes radica en su capacidad para comprender el contexto y generar nuevas frases que reflejan la esencia del contenido original, superando las limitaciones de los métodos extractivos. La puesta a punto o fine-tuning de este modelo implica ajustarlo utilizando conjuntos de datos relevantes o Corpus, lo que permite adaptar sus capacidades generales a necesidades específicas.

En este trabajo de fin de grado, exploramos las técnicas de puesta a punto o fine-tuning de GPT-2 para la tarea específica de generación de resúmenes de texto. En la investigación nos centraremos en comparar diferentes enfoques de fine-tuning y evaluaremos su rendimiento usando conjuntos de corpus de datos. Además, desarrollaremos una aplicación móvil sencilla que integra el modelo afinado donde podremos evaluar su efectividad en un entorno real.

La selección de GPT-2 para este estudio se basa en varias consideraciones clave. Primero, su capacidad comprobada para generar texto natural y coherente lo hace adecuado para tareas abstractivas de generación de resúmenes. Segundo, su flexibilidad a la hora de ser afinado en tareas específicas mediante técnicas de puesta a punto. Finalmente, el uso de GPT-2 en una aplicación móvil demuestra la aplicabilidad práctica del modelo, mostrando cómo puede integrarse en soluciones que faciliten la compresión de grandes volúmenes de texto.

La estructura de este trabajo está diseñada para proporcionar una visión completa y detallada de la investigación realizada. Comenzamos con una revisión exhaustiva del estado del arte, abordando los enfoques tradicionales y modernos para la generación de resúmenes, y destacando los avances más recientes de los modelos Transformers. A continuación, describimos la metodología utilizada para poner a punto nuestro modelo GPT-2, incluyendo los detalles sobre los corpus seleccionados y el proceso de entrenamiento. Posteriormente, se presenta el desarrollo experimental, donde analizaremos los resultados obtenidos. Finalmente, describiremos la implementación de la aplicación móvil, seguida de las conclusiones y recomendaciones para futuras investigaciones.

Este trabajo proporciona una herramienta práctica que puede ser utilizada en diversos contextos, tanto profesionales como académicos. La integración de GPT-2 en la aplicación móvil demuestra que es posible implementar modelos avanzados del lenguaje natural en entornos accesibles y útiles para el público general, marcando así un paso adelante en la aplicación práctica en el manejo de la información.

2.1. Revisión de la literatura

2.1.1. Historia y evolución de técnicas Extractivas y Abstractivas

La generación automática de resúmenes ha sido un campo de interés en el procesamiento del lenguaje natural durante varias décadas. La necesidad de condensar grandes cantidades de texto en resúmenes breves y coherentes ha llevado al desarrollo de diversos enfoques y metodologías. Estos métodos se dividen principalmente en extractivos y abstractivos, cada uno con sus ventajas y desventajas.

Técnicas Extractivas:

1. Primera generación de Algoritmos Extractivos (1960s – 1980s) (N.Moratanch & S.Chitrakala, 2017)

- *Métodos basados en la frecuencia de palabras*: Los primeros métodos extractivos se centraron en la frecuencia de las palabras clave como principal criterio de selección. Los algoritmos contaban el número de veces que las palabras aparecían en un documento y seleccionaban las frases que contenían aquellas palabras más frecuentes.

- *TF-IDF (Term Frequency-Inverse Document Frenquency)*: Introducido en el año 1970, TF-IDF mejoró la selección de considerar no solo la frecuencia de las palabras en un documento, sino también su frecuencia inversa en un corpus más amplio, destacando términos importantes que no son comunes.

2. Algoritmos basados en Heurísticas y Puntuación de Frases (1990s) (Carmona & Aldón, 2013)

- *Algoritmos de puntuación*: Estos métodos asignan puntuajes a frases basados en diversas heurísticas, como la posición en el texto, la presencia de términos clave, y las características del formato, como por ejemplo títulos o encabezados.

- *Modelos de grafos*: La introducción de modelos como TextRank a finales de los años 90 y principios de los 2000, permitió la construcción de grafos creados por frases y la identificación de las más importantes mediante propagación de puntajes en el grafo.

3. Métodos basados en el Aprendizaje Automático (2000s) (Carmona & Aldón, 2013)

- *Modelos Supervised Learning*: Con la disponibilidad de grandes cantidades de datos etiquetados, se empezaron a usar técnicas de aprendizaje supervisado, donde los modelos de clasificación aprendían a seleccionar frases relevantes basándose en características extraídas del texto.

- *Modelos No Supervised Learning*: Métodos como clustering y algoritmos de agrupamiento también se aplicaron para identificar temas y seleccionar frases representativas sin necesidad de datos etiquetados.

Técnicas Abstracticas:

1. Primera generación de Técnicas Abstractivas (1990s – 2000s) (Wikipedia, Lenguaje Natural (LNP), 2024)

- *Plantillas y Reglas*: Los primeros métodos abstractivos eran capaces de generar frases nuevas, predefinidas y reglas gramaticales. Estos sistemas eran capaces de generar frases nuevas, pero estaban limitados por las plantillas y la dificultad de manejar variaciones lingüísticas.

- *Sistemas basados en Plantillas*: Se desarrollaron sistemas que utilizaban plantillas para generar resúmenes. Aunque estos métodos podían producir texto más coherente, eran limitados por su capacidad para adaptarse a diferentes dominios.

2. Redes Neuronales Recurrentes (RNN) y Seq2Seq (2010s)

- *Modelos Seq2Seq* (Sutskever, Vinyals, & Le, 2014): La introducción de modelos de secuencia a secuencia con redes neuronales recurrentes revolucionó la generación abstractiva. Estos modelos, especialmente con los mecanismos de atención, permitieron generar resúmenes más coherentes y relevantes.

- *Mecanismo de Atención* (Wikipedia, Mecanismos de Atención, 2024): Este avance permitió a los modelos enforcarse en diferentes partes del texto de entrada mientras generaban cada palabra del resumen, mejorando la capacidad de generar resúmenes que mantenían la coherencia y la relevancia del tema.

3. Transformers y modelos de lenguaje pre-entrenados (Finales de 2010s – Presente)

- *Transformers* (Vaswani, y otros, 2017)*:* La introducción de la arquitectura Transformer (2017) mejoró significativamente la capacidad de los modelos para manejar dependencias a largo plazo en el texto. Transformers como GPT han superado a los modelos basados en RNN.

- *Modelos pre-entrenados* (Javi, 2023): Modelos como BERT, GPT-2 y GPT-3 han sido pre-entrenados en grandes corpus de texto y luego afinados para tareas específicas. Estos modelos han demostrado capacidades avanzadas en generación de texto, incluyendo la generación de resúmenes. Además, pueden generar texto sin necesidad de plantillas rígidas ni reglas predefinidas.

Comparación entre enfoques Extractivos y Abstractivos:

- *Precisión y coherencia*: Los métodos extractivos tienden a ser precisos en términos de mantener el contenido original, pero a menudo carecen de coherencia. Los métodos abstractivos, sin embargo, pueden generar texto más natural, pero corren el riesgo de introducir errores semánticos.
- *Facilidad de implementación*: Los enfoques extractivos suelen ser más simples de implementar y requieren menos recursos en comparación con los abstractivos, que con más complejos y demandan más potencia de cómputo para ser entrenados.
- *Aplicaciones y usos*: Los resúmenes extractivos son útiles en aplicaciones donde es crucial mantener la exactitud del texto original, como en resúmenes médicos, por ejemplo. En cambio, los abstractivos son preferibles en contextos donde la fluidez y naturalidad son importantes, como en noticias o páginas web.

2.1.2. Evolución hacia modelos Transformer

(Vaswani, y otros, 2017)

La arquitectura Transformer, introducida por Vaswani en el artículo **"Attention is all you need"** en 2017, marcó un punto de inflexión en el campo del procesamiento del lenguaje natural (NLP). Antes de la aparición de los Transformers, las redes neuronales recurrentes (RNN) y sus variantes como puede ser LSTM (Long Short-Term Memory) o GRU (Gated Recurrent Unit) eran la base de los modelos NLP. Sin embargo, estos modelos enfrentaban algunas limitaciones significativas en los términos de capturar dependencias a largo plazo en la paralelización del entrenamiento.

Los Transformers abordaron estas limitaciones mediante el uso de un mecanismo de atención 'auto-regresivos', que permite al modelo ponderar la importancia de diferentes palabras en una secuencia independientemente de la posición en la que se encuentren dentro del texto. Esto permite capturar de manera más efectiva las dependencias contextuales a largo plazo y facilita la paralelización durante el entrenamiento, mejorando así el rendimiento significativamente en diversas tareas de NLP, incluida la generación de resúmenes.

Principios fundamentales de los modelos Transformer:

(Ahmed, y otros, 2023)

1. Mecanismo de Atención

- *Atención escalable:* La atención autorregresiva permite que cada palabra en una secuencia preste atención a todas las demás de manera simultánea, escalando con el tamaño de la secuencia en comparación con las RNNs

- *Atención multi-cabeza:* Los Transformers utilizan múltiples "cabezas" de atención, lo que permite al modelo enfocarse en diferentes partes de la secuencia simultáneamente y capturar diversas relaciones contextuales.

2. Estructura en capas

- *Capas de codificación y decodificación:* Los Transformers se componen de bloques de codificación y decodificación. Cada bloque contiene capas de atención y capas feed-forward totalmente conectadas, con mecanismos de normalización y residualización para mejorar la estabilidad del entrenamiento.

- *Positional Encoding:* Dado que los Transformers no tienen una estructura secuencial inherente como las RNNs, utilizan codificaciones posicionales para incorporar nueva información sobre el orden de las palabras en la secuencia.

Primeras implementaciones y aplicaciones

(Trabado, 2024)

1. Transformer original (Vaswani, 2017)

- *Traducción automática*: El modelo Transformer original fue probado en tareas de traducción automática, logrando resultados sobresalientes en comparación con modelos basados en RNN

- *Paralelización y escalabilidad*: La capacidad de paralelizar el procesamiento de secuencias completas permitió entrenar modelos más grandes y complejos en menos tiempo.

2. BERT (Bidirectional Encoder Representations from Transformers)

- *Pre-entrenamiento en máscaras de lenguaje*: BERT utiliza una técnica de enmascaramiento de lenguaje bidireccional, pre-entrenado en grandes corpus de texto para capturar relaciones contextuales profundas.

- *Tareas de comprensión de lenguaje natural*: BERT ha establecido nuevos estándares en una variedad de tareas de comprensión de lenguaje natural, desde clasificación de texto hasta respuesta a preguntas.

3. GPT (Generative Pre-trained Transformer)

- *Generación de texto:* A diferencia de BERT, que se centra en tareas de comprensión, GPT se ha especializado en la generación de texto. GPT-2 y GPT-3, las versiones posteriores, han demostrado una capacidad impresionante para esta tarea. Con la nueva versión GPT-4, lanzada hace poco, las dimensiones de estas generaciones, así como de cohesión han sido mejorados hasta tal punto que es difícil distinguir la realidad de la máquina.

- *Fine-tuning para tareas específicas:* La versatilidad de GPT radica en su capacidad para ser afinado en tareas específicas, mejorando su desempeño mediante el uso de conjuntos de datos específicos para la tarea.

4. Gemini (Google DeepMind)

- *Versatilidad y "todoterreno":* Gemini es un modelo multimodal que entiende varios tipos de información, tanto textos como imágenes, audio o código de programación. Esto lo convierte en uno de los modelos más flexibles en el mercado.

- *AlphaCode2:* Gemini también introduce el nuevo sistema de generación de código llamado AlphaCode2. Este sistema mejora la compresión de matemáticas complejas y la teórica de ciencias de la computación, así como el razonamiento y la capacidad de entender código.

Impacto de los Transformers en la Generación de Resúmenes

1. Mejora de la coherencia y la relevancia

- *Atención y dependencias a largo plazo:* Los Transformers permiten que el modelo mantenga la coherencia temática a lo largo de largos pasajes de texto, lo cual ayuda a que la generación de un resumen tenga mayor sentido y esté mejor formada.

- *Generación contextualmente rica:* Los modelos basados en Transformers pueden generar texto que es tanto relevante coherente de manera simultánea, capturando matices y detalles contextuales importantes.

2. Versatilidad y adaptabilidad

- *Modelos pre-entrenados:* Estos modelos pueden ser afinados para tareas específicas de generación de resúmenes, adaptándose a diferentes dominios y estilos de escritura.

- *Transferencia de aprendizaje:* La capacidad de estos modelos para transferir el conocimiento adquirido durante el pre-entrenamiento a nuevas tareas ha permitido avances significativos en la calidad de los resúmenes generados.

3. Implementaciones prácticas

- *Aplicaciones en la industria:* Desde asistentes virtuales hasta herramientas de resumen de noticias y documentos legales, los Transformers están siendo integrados en una variedad de aplicaciones prácticas.

- *Investigación y desarrollo continuo:* La comunidad de investigación continúa explorando mejoras en la arquitectura de los Transformers y su aplicación en la generación de resúmenes, asegurando un progreso constante en el campo.

Desafíos y futuras direcciones

1. Costo computacional

- *Requerimientos de hardware:* El entrenamiento de modelos Transformer grandes requiere recursos computacionales significativos, lo cual puede ser una barrera para algunas aplicaciones y usuarios.

- *Optimización y eficiencia:* La investigación continua busca formas de optimizar la eficiencia de estos modelos, haciendo posible su uso en aplicaciones con limitaciones de recursos.

2. Calidad y precisión del resumen

- *Manejo de información crítica:* Asegurar que los resúmenes generados no omitan información crítica o introduzcan errores es un desafío continuo.

- *Evaluación automática:* Desarrollar métricas automáticas que evalúen con precisión la calidad de los resúmenes sigue siendo un área de investigación activa.

3. Interpretabilidad y control

- *Transparencia del modelo:* Entender como los Transformers generan resúmenes y como se puede controlar este proceso es esencial para aplicaciones donde la transparencia y control son importantes.

-*Interacción humano-modelo:* Facilitar una interacción efectiva entre usuarios y modelos Transformer, permitiendo ajustes y personalizaciones en tiempo real.

2.1.3. Enfoque específico en GPT-2

(Wikipedia, GPT-2, 2024)

Introducción a GPT-2

GPT-2 (Generative Pre-trained Transformer 2) es un modelo de lenguaje desarrollado por OpenAI y lanzado en 2019. Es una extensión del modelo GPT original, y está basado en la arquitectura Transformer. GPT-2 ha sido entrenado con un enorme corpus de texto (alrededor de 40GB de datos), lo que le permite generar texto coherente y contextualmente relevante. La principal innovación de esta extensión con su modelo original radica en su capacidad de generar texto de alta calidad a partir de un simple prompt, demostrando una notable comprensión y generación de lenguaje natural.

Arquitectura de GPT-2

1. Transformer decoder

- *Unidireccionalidad*: GPT-2 utiliza un enfoque unidireccional donde la predicción de cada palabra depende únicamente de las palabras anteriores en la secuencia. Esto contrasta con modelos como BERT donde se utiliza una bidireccionalidad enmascarada.

- *Capas de atención multi-cabeza:* Al igual que otros Transformers, este modelo emplea múltiples cabezas de atención para capturar diversas dependencias en el texto. Cada cabeza procesa la información de una manera diferente, lo que enriquece la representación final.

- Capas feed-forward: Además de las capas de atención, GPT-2 incluye capas feed-forward totalmente conectadas que transforman la representación interna del modelo después de la atención.

2. Escalabilidad y tamaños del modelo

- *Distintas variantes:* GPT-2 se lanzó en varias versiones de deferentes tamaños, desde modelos pequeños (aproximadamente 120 millones de parámetros) hasta el modelo más grande (más de 1,5 mil millones de parámetros). Cada incremento en el tamaño del modelo mejoró significativamente la capacidad de generar texto.

- *Capacidad de generalización:* Los modelos más grandes han demostrado una capacidad superior para generalizar a nuevos datos y tareas sin necesidad de ajustes adicionales.

Entrenamiento y pre-entrenamiento de GPT-2

1. Corpus de entrenamiento

- *Diversidad del corpus:* El corpus de entrenamiento de GPT-2 incluye una amplia variedad de fuentes textuales de internet, como artículos de noticias, sitios web, foros, lo que proporciona una base diversa y rica en contenido.

- *Tamaño del corpus:* La gran cantidad de datos permite al modelo aprender patrones y relaciones lingüísticas complejas, contribuyendo a su capacidad de generar texto.

2. Proceso de pre-entrenamiento

- *Auto-regresión:* GPT-2 se entrena de manera autorregresiva, es decir, predice cada palabra e una secuencia basada en palabras anteriores. Este método aprovecha el contexto acumulado par generar el texto paso a paso.

- *Objetivo de modelo de lenguaje:* El objetivo del pre-entrenamiento es minimizar la pérdida de predicción de palabras en secuencias largas, optimizando la capacidad del modelo para continuar un texto a partir de un prompt inicial.

Fine-tuning de GPT-2 para tareas específicas

1. Transferencia de aprendizaje

- *Adaptabilidad:* Uno de los mayores beneficios de GPT-2 es su capacidad para ser afinado (fine-tuned) en tareas específicas con relativamente pocos datos adicionales. Este proceso ajusta los pesos del modelo pre-entrenado para optimizar su rendimiento en la tarea deseada.

- *Proceso de fine-tuning:* El proceso de fine-tuning implica entrenar el modelo pre-entrenado con un conjunto de datos etiquetado específico para la tarea, ajustando los pesos del modelo para mejorar su desempeño en esa tarea.

2. Aplicaciones en resúmenes automáticos

- *Generación de texto extractivos y abstractivos*: GPT-2 se puede ajustar para generar tanto resúmenes extractivos como abstractivos. En el enfoque extractivo, el modelo selecciona las partes más relevantes del texto original, mientras que, en el abstractivo, el modelo reformula y sintetiza el contenido.

- *Calidad:* Los resúmenes generados por este modelo tienden a ser más coherentes y fluidos en comparación con los métodos tradicionales, gracias a su capacidad para entender el contexto y generar lenguaje natural.

Desempeño de GPT-2 en tareas de generación de resúmenes

1. Evaluación y métricas

- *Métricas automatizadas:* Para evaluar el rendimiento de GPT-2 en la generación de resúmenes, se utilizan métricas como ROUGE (Recall-Oriented Understudy for Gisting Evaluation), que compara la superposición de n-gramas, frases y secuencias entre el resumen generado y los resúmenes de referencia.

- *Evaluación humana:* Además de las métricas automatizadas, la evaluación humana es esencial para determinar la calidad general de los resúmenes generados. Los evaluadores humanos proporcionan una perspectiva cualitativa que las métricas cuantitativas pueden no captar completamente.

2. Resultados en diversos dominios

- *Noticias y artículos:* Se ha demostrado un rendimiento sólido en la generación de resúmenes para artículos de noticias y contenido web, donde la diversidad y la riqueza del corpus de entrenamiento son ventajosas.

- *Documentos técnicos y científicos:* Aunque resulta desafiante, GPT-2 puede ser afinado para resumir este tipo de documentos, capturando la esencia de contenidos complejos con precisión aceptable.

Limitaciones y desafíos de GPT-2

1. Costo computacional

- *Requerimiento de recursos:* Así como su arquitectura base Transformer, el entrenamiento y la inferencia de GPT-2 requieren recursos computacionales significativos, lo que dificulta su implementación en entornos con recursos limitados.

- *Optimización y eficiencia:* Mejorar la eficiencia computacional sigue siendo un área activa de investigación, con enfoques como la compresión de nuevos modelos y la optimización del hardware.

2. Calidad del resumen

- *Hallucinations:* Uno de los problemas comunes con los modelos generativos de este estilo es la introducción de información no presente en el texto original, llamadas hallucinations. Se tiene que garantizar que los resúmenes de sean precisos y fieles al contenido fuente.

- *Manejo de información compleja:* Si bien GPT-2 maneja bien los textos generales, resumir documentos muy técnicos o especializados puede requerir ajustes adicionales para mantener la precisión esperada.

Futuras direcciones y mejoras

1. Mejoras en la arquitectura

- *GPT-3, GPT-4 y más:* El desarrollo de nuevas versiones de este modelo y otros aún más avanzados promete mejorar aún más la capacidad de generación de texto, ofreciendo así, mejores resultados en los resultados.

-*Integración con otros modelos:* Combinar GPT-2 con otros enfoques NLP, como modelos de compresión de texto y sistemas de recuperación de información, puede mejorar significativamente la calidad de los resúmenes generados.

2. Aplicaciones prácticas y adaptaciones

- *Implementaciones en herramientas de software:* Integrar GPT-2 en herramientas de software para la generación de resúmenes en aplicaciones empresariales, educativas o de consumo puede proporcionar beneficios significativos en términos de productividad.

- *Personalización:* Desarrollar interfaces que permitan a los usuarios personalizar y controlar la generación de resúmenes puede hacer que estas herramientas sean más útiles adaptables.

2.2. Análisis crítico de las fuentes

2.2.1. Comparación de enfoques

En el análisis crítico de la literatura sobre la generación de resúmenes de texto, es esencial comparar los enfoques extractivos y abstractivos, así como redes neuronales recurrentes (RNNs) y Transformers. Esta comparación se centrará en varios criterios clave, incluyendo la precisión, coherencia, adaptabilidad y el costo computacional.

Enfoques extractivos

En cuanto a sus **ventajas**, se puede destacar la precisión, donde mantienen la exactitud del contenido original ya que las frases con las que se compone el resumen no se alteran. Esto es especialmente útil en contextos donde la precisión es crítica e importante, como en investigaciones científicas.

También podemos hablar de su simplicidad, ya que son fáciles de implementar y menos costosos computacionalmente. Estos métodos requieres, además, menos procesamiento y recursos, lo que los hace accesibles para aplicaciones con limitaciones.

En cuanto a sus **desventajas**, hay que mencionar su coherencia y fluidez. A menudo carecen de estas características en el texto, ya que no generan nuevas frases. Esto puede resultar en resúmenes que parecen algo fragmentados o desorganizados.

También destaca su carencia de flexibilidad, ya que no pueden parafrasear y condensar la información de manera eficiente. Esto limita su capacidad para producir textos concisos y adaptativos.

Enfoques abstractivos

En cuanto a sus **ventajas**, se puede destacar su coherencia y fluidez, ya que pueden producir resúmenes más legibles y naturales que los extractivos. Generan nuevo texto que conecta las ideas de manera lógica, lo que facilita el entendimiento en el lector. Esta capacidad es especialmente útil en contextos como el periodismo, donde se requiere una síntesis clara de la información.

Otra ventaja importante es su capacidad de parafraseo. Los métodos abstractivos pueden condensar la información, mejorando así la brevedad y claridad del resumen sin perder la esencia del contenido original. Esto permite crear resúmenes cortos y que comuniquen mejor la información.

En cuanto a sus **desventajas**, hay que mencionar la precisión, ya que existe el riesgo de omitir información importante o introducir errores semánticos. Esto es así porque la generación de nuevas frases puede llevar a interpretaciones incorrectas del texto original.

Además, destaca su alto costo computacional. Estos métodos requieren más recursos para el entrenamiento y generación. Estos procesos son intensos en términos de tiempo y potencia de procesamiento, lo que puede hacer que estos métodos sean menos accesibles para algunas aplicaciones.

Modelos basados en RNNs

En cuanto a sus **ventajas**, se puede destacar su capacidad para capturar secuencias. Las RNNs son eficaces en la captura de dependencias temporales en textos cortos, lo que les permite mantener el contexto a lo largo de secuencias de palabras. Esto las hace adecuadas para tareas como la generación de texto y el procesamiento del lenguaje natural en textos breves.

Otra ventaja es su historial de uso. Las RNNs cuentan con muchas aplicaciones y optimizaciones disponibles. Tecnologías como LSTM y GRU han mejorado la capacidad para retener información a largo plazo, haciéndolas más robustas y efectivas en diversas tareas.

En cuanto a sus **desventajas**, hay que mencionar las limitaciones en dependencias a largo plazo. Estos modelos tienen dificultades para ello debido a problemas de desvanecimiento de gradientes, lo que afecta su rendimiento en textos más largos y complejos.

Otra desventaja es la dificultad en la paralelización. Estos modelos tienen dificultades a la hora de paralelizar el entrenamiento, lo que limita su eficacia y escalabilidad. Este problema puede ralentizar significativamente el proceso de entrenamiento, especialmente en conjuntos de datos grandes, y hacer que la implementación de RNNs sea más compleja y costosa en términos de tiempo y recursos.

Modelos Transformer (General)

En cuanto a sus **ventajas**, se puede destacar su capacidad para capturar dependencias a largo plazo. Estos modelos son excelentes en esta tarea gracias a su mecanismo de atención. Esto les permite manejar mejor la complejidad de textos extensos y proporcionar resúmenes contextualmente ricos.

Otra ventaja importante es su capacidad de paralelización. Los Transformers permiten una paralelización más eficiente del entrenamiento e inferencia, lo que mejora su escalabilidad y rapidez. Esto los hace ideales para aplicaciones que requieren el procesamiento de grandes volúmenes de texto en tiempo real, como motores de búsqueda y sistemas de recomendación.

En cuanto a sus **desventajas**, hay que mencionar los altos requerimientos computacionales. Los Transformers tienen altos costos en términos de memoria y procesamiento, lo que puede ser un obstáculo para su implementación en sistemas con recursos limitados. Este aspecto puede hacer que sean menos accesibles para algunas aplicaciones debido a los costos asociados con el hardware y el tiempo de procesamiento.

Otra desventaja es la complejidad en la implementación y ajuste de hiperparámetros. Requieren un conocimiento técnico avanzado para su configuración y uso óptimo, lo que puede ser una barrera para su adopción en proyectos que no cuentan con expertos en aprendizaje profundo.

Modelos Extractivos basados en Transformer

En cuanto a sus **ventajas**, se puede destacar la mejora en la selección de frases. Estos modelos utilizan mecanismos de atención bidireccional para seleccionar las frases más relevantes del texto original, mejorando la calidad del resumen y asegurando que se incluyan las partes más importantes del contenido.

Otra ventaja es su facilidad de implementación. Al aprovechar modelos pre-entrenados, estos enfoques reducen el tiempo y esfuerzo de desarrollo, permitiendo a los desarrolladores implementar sistemas efectivos de generación de resúmenes con menos recursos y en menos tiempo. Esto es especialmente útil en aplicaciones donde la rapidez de desarrollo es crucial.

En cuanto a sus **desventajas**, hay que mencionar las limitaciones en coherencia. Aunque mejorados en comparación con los métodos extractivos tradicionales, estos modelos pueden seguir careciendo de fluidez en la narrativa, especialmente en textos muy fragmentados. Esto puede resultar en resúmenes que, aunque precisos, no son tan legibles o coherentes como los generados por métodos abstractivos.

Modelos abstractivos basados en Transformer

En cuanto a sus **ventajas**, se puede destacar su coherencia y creatividad. Los modelos abstractivos basados como BERT y GPT-2, son capaces de capturar el contexto de manera efectiva. Esta capacidad es particularmente valiosa en aplicaciones donde la calidad del lenguaje y la naturalidad del resumen son importantes, como en la creación de contenido y la generación de resúmenes para medios de comunicación.

Otra ventaja es su adaptabilidad. Estos modelos pueden ser afinados para diferentes dominios y estilos de resumen, lo que los hace muy versátiles. Esta adaptabilidad permite a los desarrolladores personalizar los modelos para aplicaciones específicas, mejorando su efectividad y relevancia en distintos contextos.

En cuanto a sus **desventajas**, hay que mencionar el alto costo computacional. lo que puede ser una limitación significativa para su implementación. Este aspecto puede hacer que estos modelos sean menos accesibles para algunas aplicaciones debido a los costos asociados con el hardware y el tiempo de procesamiento.

Otra desventaja es el riesgo de "hallucinations". Estos modelos pueden introducir información no presente en el texto original, lo que puede afectar la precisión del resumen. Este problema puede ser particularmente preocupante en contextos donde la exactitud de la información es importante, como en informes médicos.

Enfoques en aplicaciones prácticas

Una vez visto las diferencias a grandes rasgos de los modelos generales en las tareas de resúmenes de texto, también es interesante comentar cómo se usan estos enfoques en distintos sectores, diferenciándolos mediante ejemplos.

1. En el sector legal

- **Resúmenes extractivos***:* Utilizados para resumir leyes, regulaciones y precedentes judiciales. La precisión es crucial, por lo que los métodos extractivos son preferidos en este sector.

- **Resúmenes abstractivos***:* Podrían ser utilizados para generar resúmenes de casos complejos, proporcionando una visión general más accesible, aunque se requiere verificar y validar los resultados para asegurar la precisión.

2. En el periodismo

- **Resúmenes extractivos***:* Comunes para generar resúmenes de artículos de noticias, destacando las partes más relevantes sin alterar el contenido original.

- **Resúmenes abstractivos***:* Utilizados para crear versiones condensadas de noticias largas, mejorando el atractivo para los lectores.

3. En la investigación científica

- **Resúmenes extractivos***:* Utilizados para generar resúmenes de artículos científicos, capturando la metodología y los resultados.

- **Resúmenes abstractivos***:* Podrían proporcionar resúmenes de revisión que capturen la esencia de múltiples artículos sobre un tema, aunque deben ser evaluados también para asegurar la precisión.

Conclusión de la comparación

Estas comparaciones revelan que cada método tiene sus propias fortalezas y debilidades. Los modelos basados en Transformers, especialmente los abstractivos como GPT-2, ofrecen una capacidad ejemplar para la generación de texto, aunque a un alto costo computacional. La elección entre métodos extractivos y abstractivos depende en gran medida del contexto y los requisitos de la aplicación. Mientras que los enfoques extractivos son preferibles para escenarios donde la precisión prevalezca ante la creatividad, los métodos abstractivos, son más adecuados para generar resúmenes contextualmente ricos.

2.2.2. Fortalezas y debilidades de los modelos existentes

En este análisis nos centraremos en los modelos basados en técnicas tradicionales, RNNs y Transformers, comentadas con brevedad anteriormente, dejando a un lado nuestro modelo a usar GPT-2, que está bien detallado en el apartado 2.1.3.

Modelos basados en estadísticas (TF-IDF)

En cuanto a las **fortalezas** de estos modelos cabe destacar su simplicidad ya que es fácil de implementar y comprender, lo que ayuda en el uso para aplicaciones básicas. Es importante su velocidad debido a que es rápido en términos de procesamiento y adecuado para conjuntos de datos pequeños y medianos. Su cálculo es computacionalmente ligero, permitiendo una rápida generación de resúmenes. Además, es muy eficaz en la selección de términos relevantes en documentos y textos cortos porque evalúa la importancia de las palabras en el contexto de un documento especifico en relación con un corpus más grande.

En cuanto a sus **debilidades** hay que mencionar la falta de contexto, ya que no son capaces de capturar el contexto semántico ni las relaciones entre términos. Esto puede llevar a la selección de palabras o frases incorrectas. También destaca su coherencia limitada donde los resúmenes generados suelen ser menos legibles. Esto es así debido a que el modelo no considera la estructura del texto y solo la frecuencia de los términos. Además, es poco adaptable en textos diversos y complejos.

Modelos basados en grafos (TextRank)

En cuanto a las **fortalezas** de TextRank, destaca su capacidad para capturar relaciones importantes entre frases y oraciones, lo que permite identificar las partes más relevantes del texto. Su implementación no requiere un corpus de entrenamiento específico, lo que es útil para dominios desconocidos o textos en los que no se dispone de grandes conjuntos de datos anotados. Además, es relativamente fácil de implementar y ajustar, utilizando algoritmos de clasificación de grafos conocidos.

Sin embargo, en cuanto a sus **debilidades**, al igual que TF-IDF, TextRank puede generar resúmenes que carecen de fluidez, ya que no crea nuevas frases, sino que selecciona las existentes basándose en la conectividad del grafo. Su efectividad también depende de la estructura del texto original, siendo menos eficaz para textos desestructurados o aquellos que no tienen una clara conexión entre frases.

Long Short-Term Memory (LSTM)

LSTM presenta varias **fortalezas**, incluyendo su eficacia al capturar dependencias a corto y mediano plazo dentro del texto, lo que le permite mantener el contexto a lo largo de secuencias de palabras. Su flexibilidad es notable, ya que puede manejar secuencias de longitud variable y adaptarse a diferentes tipos de texto, haciéndolo versátil para diversas aplicaciones de procesamiento del lenguaje natural. Además, se beneficia del pre-entrenamiento con grandes corpus de datos, mejorando su rendimiento a través de transferencias de aprendizaje y optimizaciones previas.

En cuanto a las **debilidades**, LSTM puede tener dificultades con dependencias a largo plazo debido a problemas de desvanecimiento de gradientes, lo que afecta su rendimiento en textos extensos. El entrenamiento y la generación son más lentos en comparación con modelos más recientes, limitando su aplicabilidad en entornos que requieren resultados rápidos. También tiene limitaciones en la paralelización del procesamiento, lo que afecta su eficiencia y escalabilidad en grandes volúmenes de datos.

Gated Recurrent Unit (GRU)

Las **fortalezas** de GRU incluyen una estructura más simple en comparación con LSTM, con menos parámetros, lo que reduce el costo computacional y hace que el modelo sea más rápido y fácil de entrenar. Mantiene un buen rendimiento en la captura de dependencias temporales, similar a LSTM, pero con menor complejidad. Además, es eficaz para una gran variedad de tareas generales en procesamiento del lenguaje natural, ofreciendo un equilibrio entre rendimiento y simplicidad.

En cuanto a las **debilidades**, al igual que en LSTM, GRU tiene problemas con dependencias a largo plazo, aunque estos problemas están mitigados en cierta medida. La simplicidad de la estructura puede reducir la capacidad de capturar patrones complejos en algunos casos, lo que puede limitar su efectividad en tareas muy específicas o detalladas.

Bidirectional Encoder Representations from Transformers (BERT)

BERT presenta varias **fortalezas**, como su capacidad para la contextualización bidireccional, capturando relaciones contextuales de ambas direcciones (izquierda a derecha y derecha a izquierda), mejorando la comprensión del texto y la generación de resúmenes más coherentes. Puede ser afinado para tareas específicas con conjuntos de datos pequeños, lo que permite su aplicación en diversos dominios y tareas con alta precisión. Además, BERT mejora su rendimiento en comparación con otros modelos en varias tareas de procesamiento del lenguaje natural, como la clasificación de textos y el reconocimiento de entidades.

Sin embargo, BERT también tiene **debilidades** y no se suele utilizar para resúmenes abstractivos, lo que puede limitar la fluidez y naturalidad del resumen generado. También tiene un alto consumo de memoria y potencia de cómputo, tanto en entrenamiento como en inferencia, lo que puede ser una barrera para su implementación en sistemas con recursos limitados. La complejidad en el ajuste de hiperparámetros y la integración del modelo en sistemas de producción también son desafíos significativos.

Bidirectional and Auto-Regressive Transformers (BART)

Las **fortalezas** de BART combinan las ventajas de modelos auto-regresivos y bidireccionales, mejorando la contextualización y generación de texto. Esto lo hace muy potente para la creación de resúmenes abstractivos de alta calidad. Sobresale en tareas de resúmenes abstractivos, proporcionando resúmenes coherentes y contextualmente ricos que son comparables a los escritos por humanos. Además, puede ser afinado para multitud de tareas, lo que lo hace extremadamente versátil y adecuado para aplicaciones en diversos dominios.

Sin embargo, presenta **debilidades** y una mayor complejidad en la implementación y ajuste en comparación a otros modelos, requiriendo un conocimiento avanzado de aprendizaje profundo y optimización de modelos. Necesita una gran cantidad de conjuntos de datos para alcanzar su máximo potencial, lo que puede ser un desafío para aplicaciones con datos limitados. Además, tiene altos requerimientos de memoria y procesamiento, lo que puede limitar su accesibilidad para algunas organizaciones o aplicaciones.

2.2.3. Brechas y oportunidades en la investigación

En el campo de la generación de resúmenes de texto, nos encontramos con varias brechas significativas y oportunidades de investigación que podrían dar lugar a futuros avances y optimizaciones. A continuación, detallaremos algunas de estas áreas:

Mejora en la coherencia y fluidez de los resúmenes

En las **brechas** nos encontramos que los modelos extractivos, aunque sean precisos, suelen producir resúmenes que carecen de fluidez. Por otra parte, en los abstractivos, especialmente los que se basan en RNNs y Transformers, a veces generan texto con errores semánticos y pueden introducir información inventada (hallucinations).

Para ellos, encontramos **oportunidades** como el desarrollo de nuevas arquitecturas, como por ejemplo las híbridas que combinen las fortalezas tanto de los modelos extractivos como abstractivos. También podemos incorporar mecanismos de control, así garantizamos la coherencia y evitamos la introducción de información errónea al texto.

Reducción del costo computacional

En las **brechas** nos encontramos que los modelos basados en Transformers, aunque tienen un gran potencial, presentan altos costos computacionales tanto en términos de entrenamiento como de inferencia. Además, la implementación y ajuste de estos modelos requieren recursos significativos que no siempre están disponibles para todo el público.

Para ello, encontramos **oportunidades** en la optimización de modelos, desarrollando técnicas de compresión y optimización, como la destilación de conocimientos, que reduzcan el tamaño y la complejidad de los modelos sin comprometer su rendimiento. También podemos investigar en hardware especializado, creando nuevo hardware que ejecute modelos de lenguaje de manera más eficiente, mejorando así la accesibilidad y aplicabilidad de estas tecnologías.

Manejo de datos multimodales

En las **brechas** nos encontramos que la mayoría de los modelos actuales están diseñados para procesar texto de manera aislada y no integran eficientemente información de otras modalidades como imágenes, audio o video. Esto restringe su aplicabilidad en contextos donde la información multimodal es esencial.

Para ello, encontramos **oportunidades** en la integración multimodal, desarrollando nuevos modelos que puedan procesar y generar resúmenes a partir de datos multimodales, ofreciendo al usuario resúmenes más ricos y completos. También podemos explorar aplicaciones en nuevos dominios como la educación o la atención médica, donde la integración de diferentes tipos de datos puede proporcionar resúmenes más útiles y relevantes.

Evaluación y medición de la calidad

En las **brechas** nos encontramos que las métricas actuales, aunque útiles, no siempre capturan completamente la calidad semántica y la coherencia de los resúmenes generados. Además, la evaluación humana es muy costosa y no siempre es escalable.

Para ello, encontramos **oportunidades** en el desarrollo de nuevas métricas que evalúen mejor la calidad de los resúmenes en términos de relevancia y precisión semántica. También podemos implementar sistemas automatizados de evaluación, que combinen evaluación automatizada con la retroalimentación humana para proporcionar resultados más precisos y útiles.

3. Fundamentos teóricos

3.1. Conceptos básicos

3.1.1. Procesamiento del lenguaje natural (NLP)

(Wikipedia, Lenguaje Natural (LNP), 2024)

El procesamiento del lenguaje natural (NLP) es un campo interdisciplinario que se sitúa en la intersección de la lingüística, la informática y la inteligencia artificial. Su objetivo principal es permitir que las máquinas comprendan, procesen y generen el leguaje humano de manera natural y útil. El NLP abarca una variedad de tareas, cada una con desafíos únicos, que van desde la compresión del lenguaje natural hasta la generación de texto coherente.

El NLP se basa en la idea de que el lenguaje humano, aunque complejo y variado, sigue patrones estructurales que pueden ser reconocidos y replicados por algoritmos computacionales. Estos algoritmos están diseñados para procesar el texto de entrada, analizar su estructura y significado, y producir una salida adecuada. Los avances en el NLP han sido impulsados por el desarrollo de técnicas y modelos de aprendizaje automático y profundo, que permiten a las máquinas aprender a mejorar a partir de grandes conjuntos de datos textuales.

Entre las aplicaciones más comunes, se encuentran la traducción automática, el análisis de sentimientos, la clasificación de textos, la extracción de información o la generación de resúmenes de texto. Estas aplicaciones son posibles gracias a una serie de procesos fundamentales como:

- *Tokenización:* Dividir el texto en unidades más pequeñas, como palabras o frases y deshaciéndose también de aquellos símbolos extraños, para facilitar su análisis.
- *Análisis sintáctico (Parsing):* Determinar la estructura gramatical de las oraciones para comprender cómo se relacionan las palabras entre sí.
- *Análisis semántico:* Interpretar el significado del texto, asignando etiquetas (como nombres, lugares, etc.)
- *Análisis pragmático:* Considerar el contexto más amplio en el que se usa el lenguaje, incluyendo el propósito y la intención detrás de cada palabra.

El procesamiento del lenguaje natural es fundamental para la generación automática de resúmenes de texto, ya que permite a los algoritmos comprender el contenido y el contexto del texto original, identificar las partes más relevantes y generar un resumen que mantenga el significado esencial.

En conclusión, el procesamiento del lenguaje natural es una disciplina muy importante que habilita la interacción fluida entre humano y maquina a través del lenguaje. Los avances en este campo no solo han mejorado la capacidad de las máquinas para entender y generar texto, sino también han abierto nuevas posibilidades para aplicaciones prácticas, incluyendo el tema principal de nuestra investigación, la generación de resúmenes de texto.

3.1.2. Generación de resúmenes

La generación de resúmenes es una tarea clave en el procesamiento del lenguaje natural que tiene como objetivo condensar la información contenida de un documento largo en un resumen breve y conciso. Este proceso es esencial para gestionar la sobrecarga de información en la era digital, facilitando el acceso rápido a la parte importante de textos extensos. Como vimos en el punto 2.1.1, la generación de resúmenes se clasifica en dos enfoques principales: extractivos y abstractivos.

En los resúmenes extractivos se siguen una serie de pasos a la hora de la generación del resumen:

- *Tokenización:* Dividir el texto en unidades más manejables.
- *Análisis de relevancia:* Evaluar y asignar una puntuación a cada unidad basada en su importancia dentro del contexto del texto.
- *Selección:* Elegir las unidades con puntuación más alta y combinarlas para formar un resumen.

En cambio, en los resúmenes extractivos se siguen otra serie de pasos completamente distintos a los extractivos:

- *Comprensión del texto:* Utilizar modelos de lenguaje para interpretar y entender el contenido.
- *Generación del texto:* Crear oraciones nuevas que capturen la esencia de la información.
- *Paráfrasis y reformulación:* Modificar el texto generado para mejorar la claridad y naturalidad.

En conclusión, la generación de resúmenes es una tarea crucial en el procesamiento del lenguaje natural, que puede abordarse mediante estos dos enfoques. Cada uno de ellos tiene sus ventajas y desafíos, y la elección entre ellos dependerá de los requisitos que el usuario tenga en el momento, así como los recursos disponibles. La integración de los modelos Transformer en este ámbito han mejorado significativamente la calidad y efectividad de los resúmenes generados, haciendo posible aplicaciones más sofisticadas y útiles.

3.2. Teoría de la arquitectura Transformer

(Vaswani, y otros, 2017)

Vamos a profundar ahora en la teoría de la arquitectura Transformer para entender cómo funciona internamente su mecanismo, separando las partes más importantes de su estructura y haciendo uso de funciones matemáticas.

Primeramente, debemos conocer el esquema propuesto por Vaswani para los modelos Transformer. Estos modelos constan de dos partes principales: el codificador (encoder) en la parte izquierda y el decodificador (decoder) en la parte derecha, como se muestra en la *Imagen 1.1*. Cada uno de estos componentes está compuesto por múltiples capas idénticas que se apilan para formar una red profunda.

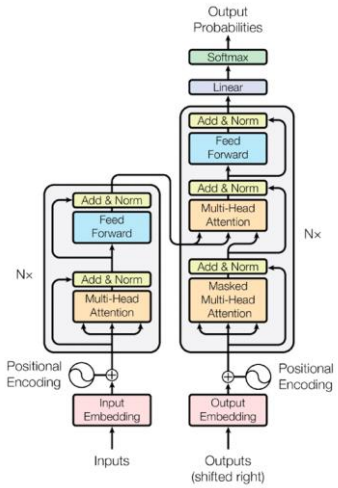

Imagen 1.1: esquema para modelos Transformer

Codificador

El codificador recibe una secuencia de entrada y genera una representación contextual para cada palabra de la secuencia. Cada etapa del codificador consta de:

- Multi-Head Attention: Visto anteriormente, es una capa que permite al modelo enfocarse en diferentes partes de la secuencia simultáneamente, mejorando así la capacidad de capturar las relaciones contextuales entre palabras.
- Add & Norm: Esta operación realiza una adición seguida de una normalización por capas, La adición ayuda a mitigar el problema de gradientes desvanecientes, mientras que la normalización estabiliza y acelera el entrenamiento.
- Feed Forward: Visto anteriormente, es una red neuronal que transforma la información obtenida y permite aprender relaciones entre las palabras de la secuencia.
- Positional Encoding: Visto anteriormente, estructuran la información y les da un orden.

En cuanto a su fórmula general de funcionamiento tenemos:

$$\boldsymbol{Encoder\ Layer}(\boldsymbol{x}) = \boldsymbol{LayerNorm}(\boldsymbol{x} + \boldsymbol{FeedForward}(\boldsymbol{SelfAttention}(\boldsymbol{x})))$$

Donde:

- 'x': es la entrada del codificador
- 'LayerNorm': es una capa de normalización que normaliza la suma de la entrada x y el resultado de FeedForward
- 'FeedForward': es la red neuronal que transforma la información.
- 'SelfAttention': viene a ser el mecanismo de atención visto antes, pero para la entrada x, donde representa la importancia de cada parte de la entrada para cada elemento individual.
- Este proceso permite al codificador capturar relaciones complejas entre las palabras en la secuencia de entrada y generar una representación vectorial más rica de la información.

Este proceso permite al codificador capturar relaciones complejas entre las palabras en la secuencia de entrada y generar una representación vectorial más rica de la información.

Decodificador

El decodificador también se compone de múltiples capas similares al codificador, pero incluye mecanismos adicionales para manejar la generación de texto secuencial. Esta capa incluye:

- Masked Multi-Head Attention: Similar a la capa de atención del decodificador, pero con un enmascarado adicional para asegurar que la predicción de una palabra solo dependa de las anteriores y no de las futuras.
- Multi-Head Attention: Esta capa permite que el decodificador se enfoque tanto en la secuencia de salida generada hasta el momento como en las representaciones producidas por el decodificador.
- Feed Forward: Similar al codificador.
- Positional Encoding: Igual al codificador.

En cuanto a su fórmula general de funcionamiento tenemos:

$$\begin{aligned}&\boldsymbol{Decoder\ Layer}(\boldsymbol{y}, \boldsymbol{memory})\\ &\qquad = \boldsymbol{LayerNorm}(\boldsymbol{y}\\ &\qquad + \boldsymbol{FeedForward}\big(\boldsymbol{SelfAttention}(\boldsymbol{y}) + \boldsymbol{CrossAttention}(\boldsymbol{y}, \boldsymbol{memory})\big))\end{aligned}$$

Donde:

- 'y', 'memory': son las salidas del decodificador
- 'LayerNorm': es igual que el codificador y normaliza, además de estabilizar el entrenamiento y acelerar el trabajo.
- 'FeedForward': es la red que transforma la información.
- 'SelfAttention': similar al codificador, en este caso con *y*, que representa la salida parcial del decodificador
- 'CrossAttention': es la atención cruzada, que permite al decodificador atender la *memory*, que típicamente es la salida.

Este proceso se repite en múltiples capas del decodificador, permitiendo generar la secuencia de salida palabra a palabra.

Habiendo visto las partes en las que se estructura cada modelo Transformer de manera general, debemos comentar también los algoritmos y funciones usados en cada subcomponente comentado.

El mecanismo de atención, como bien explicamos en la revisión a la literatura, es el núcleo de la arquitectura Transformer. Permite que cada elemento de la secuencia preste atención a todos los demás elementos, lo que facilita la captura de relaciones contextuales más complejas.

1. Atención escalable (self-attention)

Podemos calcular la atención para cada palabra en la secuencia, permitiendo así que cada una de ellas se relaciones con las demás. Esta es su fórmula:

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

Donde:

- 'Q': es la consulta, una representación vectorial de la palabra que se está procesando.
- 'K': es la clave, una representación vectorial de cada palabra en la entrada.
- 'V': es el valor, que es otra representación vectorial de cada palabra en la entrada.
- 'd_k': es la dimensión de las representaciones vectoriales.
- 'softmax': es una función que normaliza los valores de la atención entre 0 y 1.

Esta fórmula calcula la importancia de cada palabra en la entrada para la palabra actual que se está procesando. Los valores de atención más altos indican que la palabra es más relevante.

2. Atención multi-cabeza (multi-head attention)

Para calcular las diferentes relaciones contextuales simultáneas, utilizamos atención multi-cabeza. Cada una de estas cabezas tiene su propio conjunto de parámetros, lo que permite al modelo enfocarse en diferentes aspectos. Esta es su fórmula:

$$\boldsymbol{Mh(Q,K,V) = C(h_1, \dots, h_h)W^o}$$

Donde:

- 'Mh' se refiere a la multi-cabeza (Multi-Head)
- 'h_1' = es la cabeza formulada como Attention (QW_i^Q, KW_i^K, VW_i^V)
- W_i^Q, W_i^K, W_i^V son las matrices de pesos para la cabeza i.
- W^O es la matriz de pesos para la salida combinada.
- 'c' es la concatenación de las cabezas

Con esta fórmula seremos capaces de calcular la atención para un conjunto de cabezas en una palabra específica.

3. Redes neuronales Feed-Forward

Como bien hemos visto arriba, cada capa de atención es seguida por una red neuronal feed-forward que se aplica de manera independiente a cada posición. Su función es:

$$\boldsymbol{FFN_{(x)} = max(0, xW_1 + b_1)\, W_2 + b_2}$$

Donde:

- x representa la entrada de la FFN, que puede ser tanto del codificador como decodificador.
- W_1 y W_2 son matrices de pesos de las diferentes capas lineales
- b_1 y b_2 son sesgos de las diferentes capas lineales.
- max(0,x) es la función de activación ReLU (Rectified Linear Unit), que deja pasar sin cambios los valores positivos de x y pone cero los valores negativos.

Esta función transforma la entrada 'x', que permite a la red neuronal aprender relaciones no lineales entre los elementos de entrada. Esto ayuda a capturar dependencias complejas en las secuencias de texto.

4. Codificaciones posicionales

Se utilizan codificaciones posicionales para inyectar información sobre el orden de las palabras. Tenemos dos funciones que calculan estas posiciones:

$$\mathbf{PE_{(p,2i)}} = \mathbf{sin}\left(\frac{\boldsymbol{p}}{\mathbf{10000}^{\frac{\mathbf{2i}}{\boldsymbol{d}}}}\right)$$

$$\mathbf{PE_{(p,2i+1)}} = \mathbf{cos}\left(\frac{\boldsymbol{p}}{\mathbf{10000}^{\frac{\mathbf{2i}}{\boldsymbol{d}}}}\right)$$

Donde:

- 'p', representa la posición de la palabra en la secuencia (comenzando desde 0).
- 2i, representa la dimensión del vector de codificación de posición, elemento par.
- 2i + 1, lo mismo, pero elemento impar.
- 'd' es la dimensionalidad del vector de representación de la palabra.
- 'sin' y 'cos', calculan el valor del seno y coseno basado en la posición '*p'*, escalado por un factor determinado por la dimensión 2i, 2i + 1, y la dimensión 'd_m'.

Las funciones seno y coseno son periódicas, lo que significa que sus valores se repiten a lo largo de un intervalo específico. Esta propiedad ayuda al modelo a aprender información posicional de secuencias largas.

Además, estas ecuaciones crean un vector de codificación. Cada elemento de este vector captura información sobre la ubicación de la palabra correspondiente en la secuencia, sumándose a las embeddings de entrada, evitando así, el problema del orden.

3.3. Teoría del modelo GPT-2

(Ahmed, y otros, 2023)

Al igual que con la arquitectura Transformer, en este punto vamos a dedicarnos a explicar de manera teórica y matemática el modelo GPT-2 que estamos usando, el cual deriva de la arquitectura Transformer, pero añade nuevos métodos y fórmulas que son aconsejables tratar.

Framework del modelo (marco de trabajo)

(Radford, Narasimhan, Salimans, & Sutskever, 2018)

GPT-2 tiene dos etapas en el proceso de entrenamiento: el preentrenamiento no supervisado y el ajuste fino supervisado

En la primera etapa, **preentrenamiento no supervisado**, se utiliza un corpus no supervisado de tokens. El objetivo del modelado del lenguaje es maximizar la probabilidad condicional de los tokens dados sus contextos anteriores:

$$L_1(U) = \sum_{i=1}^{N} \sum_{j=1}^{n_i} logP(u_j^i | u_1^i, \dots, u_{j-1}^i; \theta)$$

Donde:

- '$L_1(U)$': Es la función de pérdida del preentrenamiento no supervisado. Representa la suma de los logaritmos de las probabilidades condicionales de cada token en la secuencia de entrenamiento.
- 'U': Es el corpus no supervisado de oraciones sobre el que se entrena el modelo. Este corpus está compuesto por una serie de oraciones U = {u^1, …, u^n}, y cada oración u^i está formada por una secuencia de tokens $u^i = \{u_1^i, \dots, u_{n_i}^i\}$
- 'i': Es un índice que recorre cada oración en el corpus.
- 'n_i': Es el número de tokens totales en la oración 'i'.
- 'j': Es un índice que recorre cada token dentro de la oración 'i'. El sumatorio interno indica que la función de pérdida se calcula sobre los tokens de cada oración del corpus.
- 'log': Es la función logaritmo. Aquí se usa para ayudar a manejar la multiplicación de pequeñas probabilidades y facilita la optimización.
- $P(u_j^i | u_1^i, \dots, u_{j-1}^i; \theta)$: Es la probabilidad condicional del token 'u_j^i' dado su contexto anterior en la misma oración. Esta es la probabilidad modelada por la red neuronal con parámetros 'θ'.
- 'u_j^i': es el token actual de la oración 'i' que se está tratando de predecir
- '$u_1^i, \dots, u_{j-1}^i$': Son los tokens anteriores al token actual 'u_j^i' dentro de la misma oración 'i'. Estos tokens forman el contexto que usa el modelo para predecir el token actual.
- 'θ': Representa los parámetros del modelo, que se están entrenando.

En la segunda etapa, **ajuste fino supervisado**, después de pre-entrenar el modelo en una gran cantidad de texto sin etiquetas (es decir, sin indicaciones específicas sobre qué significa cada

parte del texto), esta segunda etapa adapta el modelo a tareas específicas utilizando datos etiquetados. Esto mejora la capacidad del modelo para realizar las tareas con mayor precisión.

Para lograr este objetivo, se ajustan los parámetros del modelo para optimizar su rendimiento en dicha tarea.

Esta etapa sigue una serie de procesos:

- **Selección de un conjunto de datos etiquetados:** Se comienza con un conjunto de datos etiquetados. En nuestro caso, si se quiere ajustar el modelo para realizar resúmenes a partir de artículos de noticias, el conjunto consistirá en textos con etiquetas correspondientes a su categoría.
- **Entrada de datos al modelo pre-entrenado:** Los datos etiquetados se pasan a través del modelo pre-entrenado. Cada instancia de datos (una oración o un párrafo) se convierte en una representación de vectores a través de las capas del modelo.
- **Representaciones finales:** Se extraen las representaciones finales del modelo que capturan características complejas del texto que son útiles para la tarea específica.
- **Capa de salida:** Se añade una capa de salida al modelo que, además, es específica para nuestra tarea.
- **Entrenamiento supervisado:** Se entrena esta nueva capa de salida utilizando nuevamente los datos etiquetados. Durante este proceso, el modelo ajusta sus parámetros para minimizar el error en la predicción de las etiquetas correctas para los datos de entrada. Este proceso implica calcular una función de pérdida que mide la discrepancia entre las predicciones del modelo y las etiquetas reales, y luego utilizar técnicas de optimización (como el algoritmo Adam) para ajustar los parámetros nuevamente y reducir esta pérdida.
- **Evaluación y validación:** Durante el entrenamiento, se evalúa el rendimiento del modelo en un conjunto de datos de validación para asegurarse que no está sobre ajustándose a los datos de entrenamiento. Esto se hace midiendo la precisión de las predicciones del modelo en el conjunto de validación y ajustando los parámetros a partir de los resultados.
- **Modelo afinado:** Al final del proceso de ajuste fino, se obtiene un modelo que está específicamente adaptado para nuestra tarea en cuestión. Este modelo puede ahora ser utilizado para realizar predicciones en nuevos datos no vistos que sean similares a los de entrenamiento.

Embeddings y positional encoding

Las palabras de entrada se convierten en embeddings antes de ser procesadas por el modelo. GPT-2 también utiliza codificaciones posicionales para inyectar información sobre la posición de cada palabra en la secuencia, pero esto último ya lo hemos visto con detalle en la sección 3.2. La función es:

$$\boldsymbol{h_0 = W_e x + W_{pe}}$$

Donde:

- 'h_0': Es la representación inicial de las palabras en el modelo. Es el vector que calcula la información inicial antes de que se apliquen más transformaciones a través de la red neuronal.
- 'W_e': Es la matriz de embeddings. Se utiliza para convertir las palabras en vectores, donde la similitud de estas palabras se refleja en la proximidad de estos vectores.
- 'W_{pe}': Es la matriz de codificaciones posicionales
- 'x': Es la secuencia de palabreas, donde cada palabra está codificada como un vector caliente. Este vector es un vector binario de longitud igual al tamaño del vocabulario, con un solo 1 que indica la palabra correspondiente y los demás elementos a 0.

Esto es un paso crucial a la hora de preparar la entrada para las capas posteriores y generar texto coherente.

Optimización y regularización

Para entrenar un modelo tan grande, se utilizan técnicas avanzadas de optimización y regularización, como el uso de Adam como optimizador. Su algoritmo es:

$$\boldsymbol{Adam(\theta_t) = \theta_{t-1} - \alpha \frac{m_t}{\sqrt{v^t + \epsilon}}}$$

Donde:

- 'θ_t': Son los parámetros del modelo en el paso t. Estos son los valores que estamos optimizando y actualizando iterativamente.
- 'θ_{t-1}':Son los parámetros antes de la actualización en el paso actual.
- 'm_t' y 'v^t' son las estimaciones del primer y segundo momento de los gradientes.
- 'α' es la tasa de aprendizaje.
- 'ϵ' es un pequeño valor para evitar divisiones por cero.

Este algoritmo es muy popular para el entrenamiento de redes neuronales, así como modelos de gran tamaño. Ayuda a obtener un mejor rendimiento en tareas de procesamiento de lenguaje natural.

4. Metodología

(jbagnato, 2022)

El objetivo de este trabajo como hemos comentado es la generación de resúmenes automáticos de textos utilizando el modelo pre-entrenado GPT-2. Este enfoque se basa en la técnica de fine-tuning, que permite adaptar un modelo generalista a una tarea específica mediante el ajuste de sus pesos en un conjunto de datos particular (corpus).

Habiendo comentado la parte teórica en puntos anteriores, entraremos ahora a la parte práctica, donde seremos capaces de ajustar este modelo a nuestra tarea mediante una serie de pasos generales: preparación de datos, configuración del modelo, entrenamiento, evaluación y aguardado del modelo ajustado para futuros usos y prácticas.

En la preparación de datos, deberemos recoger un corpus válido, esto es, que sea amplio y contenga tanto documentos de texto como sus respectivos resúmenes. Posteriormente, veremos cómo lo tratamos para que el modelo pueda reconocerlo correctamente.

Una vez los datos preparados, tendremos que configurar nuestro modelo pre-entrenado, para que aprenda a desempeñar la tarea de generar resúmenes, a través de los datos que le pasamos por entrada.

Entrenaremos nuestro modelo con los datos de entrenamiento y validación recogidos del corpus, ajustando hiperparámetros y recogiendo el porcentaje de pérdida en cada iteración.

En la evaluación, deberemos mostrar por consola los resultados obtenidos de nuestro entrenamiento y observar si ha sido satisfactorio.

Una vez estemos conforme con lo anterior, procederemos a guardar nuestro modelo ya ajustado a la generación de resúmenes para su posterior práctica y pruebas.

4.1. Preparación del entorno y carga de datos para el modelo GPT-2

En este apartado, profundizaremos en las técnicas específicas utilizadas para afinar nuestro modelo.

Lo primero de todo, para garantizar un entorno de desarrollo adecuado, es necesario instalar librerías y dependencias necesarias. Además, se utilizó Python como lenguaje de programación principal junto con las librerías especializadas para el aprendizaje profundo:

- Transformers: Proporciona herramientas para trabajar con modelos de lenguaje pre-entrenados.
- Torch: Se utiliza para el desarrollo y el entrenamiento del modelo.
- Pandas: Utilizada para la manipulación y análisis de datos tabulares, como CSV.

En primer lugar, se configuró el entorno de trabajo, seleccionando el dispositivo de procesamiento adecuado (CPU o GPU) y estableciendo semillas para garantizar la reproducibilidad. *Código 4.1*.

Se cargó el modelo GPT-2 pre-entrenado desde la librería que se encuentra en la página Hugging Face, así como su tokenizador asociado. Además, para mejorar posteriormente el entrenamiento del modelo, se creó una configuración personalizada donde añadimos dropout para reducir el sobreajuste. *Código 4.2*.

Se añadió el token especial *"Resumen"* al tokenizador y se ajustó el modelo para reconocer este nuevo token. Esto le permitirá reconocer al modelo cuando empieza y cuando termina el resumen de un texto dentro del corpus. A la hora de añadirlo, manejamos una excepción en caso de que este token ya exista, por lo que no funcionaría correctamente y deberíamos cambiar su nombre. *Código 4.3*.

Se creó una función llamada load_and_filter_data que nos ayudará a la hora de filtrar los datos del corpus. Esta función lee los conjuntos de datos de entrenamiento y validación y los filtra de manera que solo recoja aquellos cuya longitud no exceda los 768 tokens. Este proceso previo ralentizará el tiempo total de ejecución del entrenamiento, pero garantizará que el tamaño de los datos de entrada sea parecido y mejorará el entrenamiento. *Código 4.4.*

Se desarrolló una clase "GPT2Dataset" para procesar los datos de entrenamiento y validación, incluyendo la tokenización y el padding necesario. *Código 4.5*. Como es un poco extenso, explicaremos por pasos lo que ocurre aquí dentro:

- El **__init__** es un método que inicializa la clase con el tokenizador y las variables necesarias.
- El self.tokenizer lo utilizaremos para guardar nuestro tokenizador anterior.
- La variable self.dataframe se encargará de almacenar los datos del archivo CSV cargado y filtrado previamente.
- self.special_token recogerá el token especial mandado por entrada y lo guardará para posteriormente unirlo al texto de entrada.
- El **__len__** es un método que devuelve simplemente la longitud del dataframe, que contiene todos los datos.
- El **__getitem__** se encarga de organizar la lógica principal de nuestro modelo, donde instanciaremos tanto el input_text como el target para que el modelo sea capaz de reconocer qué parte es el artículo de entrada que tiene que leer, y qué parte es el resumen que tiene que aprender.

- En article y summary recogemos los pares de textos almacenados en el dataframe.
- Como nuestro conjunto de resúmenes está separado en varias líneas, las juntamos en una con el método splitlines() para facilitar la comprensión al modelo.
- Input_text y target_text es la concatenación de los tokens especiales seguido por el artículo y terminado por 'Summary:' y el token de fin de línea. Esto se hace así para garantizar la estructura general del texto como debe tomar cada parte del texto.
- Input_encodings y target_encodings guardan las tokenizaciones de input_text y target_text, añadiendo truncamiento y padding para garantizar la misma longitud a todas las entradas si es necesario.
- Guardamos en input_ids, attention_mask y labels.
- Devolvemos los tensores de las anteriores variables.

Dividimos los datos en conjuntos de entrenamiento y validación y creamos también los dataloaders correspondientes. *Código 4.6.*

4.2. Fine-tuning del modelo GPT-2

En este punto se detallan los procedimientos específicos y las herramientas empleadas durante el entrenamiento para la tarea de generación de resúmenes de texto.

Si nos centramos en el ciclo de entrenamiento, deberemos hablar de la función de pérdida, el optimizador, y la lógica de entrenamiento y evaluación del modelo. En el *código 4.7* podemos ver la implementación del ciclo de entrenamiento usado en nuestro modelo.

Explicaremos paso a paso lo que vemos en el código para entenderlo con claridad:

- Tenemos comienzo unos cuantos parámetros de entrada para el optimizador, el scheduler y el bucle de entrenamiento. Estos pueden ser modificados dependiendo de los resultados que nos den, para mejorar nuestro modelo.
- AdamW es un optimizador, explicado en puntos anteriores, que implementamos para el modelo GPT-2 gracias a sus buenos resultados.
- Inicializamos también el scheduler para ajustar la tasa de aprendizaje durante el entrenamiento.
- Dentro del bucle tenemos:
  - *train():* configura el modelo en modo entrenamiento.
  - *Zero_grad():* resetea los gradientes acumulados a 0. Esto es muy importante ya que, si no se acumularían, además de ser todavía más costoso.
  - Los input_ids, attention_masks y labels que instanciamos en la clase GPT2Dataset los volcamos aquí. De esta manera el modelo tiene directamente los datos necesarios sin hacer cálculos previos.
  - En outputs guardamos los resultados del entrenamiento del modelo con los datos de entrada, etiquetas y objetivos.
  - En la variable 'loss' guardamos la pérdida por iteración de las salidas.
  - Calculamos los gradientes y actualizamos los parámetros del modelo y la tasa de aprendizaje

Una vez termina cada época de entrenamiento, se realiza la evaluación del modelo para monitorear su rendimiento en el conjunto de validación. Su funcionamiento es casi igual que el bucle de entrenamiento, pero con la diferencia que aquí tenemos el modelo en función de evaluación (*model.eval()*) , lo que impide al modelo seguir entrenando en un conjunto de datos. Esto lo hacemos para comprobar que tan bien está siendo entrenado nuestro modelo poniéndolo a prueba con otros conjuntos de datos diferente al usado anteriormente.

Además, imprimimos por pantalla la pérdida media tanto para el entrenamiento como para la evaluación al final del entrenamiento completo. En el *código 4.8* tenemos el código.

Una vez terminado nuestro entrenamiento, estamos preparados para usarlo y probarlo por nosotros mismos, no sin antes guardarlo para evitar tener que volver a realizar otro entrenamiento. Para ello tenemos en el *código 4.9* un método sencillo para hacerlo, dejándonos a continuación, el modelo preparado para utilizarlo en la aplicación que queramos.

En este capítulo, se ha detallado el enfoque metodológico adoptado para nuestra tarea, desde la configuración del entorno de desarrollo y la implementación del ciclo de entramiento y evaluación, hasta las técnicas de fine-tuning empleadas. Este enfoque meticuloso asegura que el modelo se entrene de manera eficiente y se evalúe rigurosamente, estableciendo una base sólida para los experimentos posteriores.

En el siguiente capítulo, se discutirá el corpus utilizado para entrenar y evaluar el modelo. Se analizarán las características de los datos, los criterios de selección y la preparación del corpus.

5. Corpus utilizados

5.1. Descripción de los corpus

En este trabajo hemos tenido que recabar información y recopilar varios corpus de datos para asegurar que nuestro modelo se entrene correctamente y sea capaz de realizar su tarea de generación de resúmenes. Para ello, todos los corpus que hemos encontrado tienen estas características en común:

- Conjuntos de pares de textos y resúmenes, en igual cantidad para ambos tipos.
- Gran volumen de datos con alrededor de 50 mil y 300 mil muestras de entrenamiento.
- Amplio campo de temas en noticias, incluyendo anuncios, premios, etc.
- La calidad de los resúmenes debe ser buena y concisa, sin excederse en longitud.

Esto son los principales puntos que hemos tenido en cuenta a la hora de elegir. Ahora debemos poner a prueba los corpus elegidos. En este caso nosotros, escogimos 4 que cumplían con los criterios mencionados, para finalmente quedarnos con uno de ellos y prepararlo adecuadamente para nuestro modelo específico.

De entre estos cuatro corpus, tres de ellos consisten en noticias del Daily-Mail y el CNN, medios que son importantes en temas de información debido a su tratamiento objetivo de temas diversos. El último corpus está compuesto por noticias de la BBC, igualmente extenso y amplio que los demás.

Corpus 1 (Penugonda, Shankar, & Gowri, 2021):

- *Nombre:* CNN-DailyMail News Text Summarization
- *Origen de descarga:* kaggle.com
- *Tamaño:* aproximadamente 500MB
- *Formato:* CSV
- *Separado en conjuntos:* train, validation, test
- *Idioma:* inglés

Corpus 2 (Sharif, 2018):

- *Nombre:* BBC News Summary
- *Origen de descarga:* kaggle.com
- *Tamaño:* 9MB
- *Formato:* TXT
- *Separado en conjuntos:* Articles y Summaries
- *Idioma:* inglés

Corpus 3 (Ahuir, 2023):

- *Nombre:* Dacsa
- *Origen de descarga:* HuggingFace.co
- *Tamaño:* aproximadamente 3GB
- *Formato:* JSON
- *Separado en conjuntos:* train, validation, test
- *Idioma:* español y catalán

Corpus 4 (harvardnlp, 2018):

- *Nombre:* Summarization CNN/DM
- *Origen de descarga:* opennmt.net
- *Tamaño:* aproximadamente 1,3GB
- *Formato:* SRC
- *Separado en conjuntos:* train, validation, test
- *Idioma:* inglés

Cada uno de estos corpus se somete a una serie de pruebas para comprobar cuál es el más cómodo y óptimo de utilizar. Primero, separamos estos corpus en conjuntos para el modelo: train, validation y test. A aquellos que ya venían preparados de esta manera, se les da prioridad en la selección. Posteriormente, comparamos los formatos, ya que cada uno está organizado de manera distinta: CSV, JSON, SRC y TXT. Nos damos cuenta de que, a la hora de cargar los datos en un modelo, los archivos en CSV y JSON son más rápido, por lo que también se les da prioridad. Por último, comprobamos el idioma en el que se encuentran nuestros corpus, ya que es crucial para el entrenamiento del modelo. Tenemos tanto corpus en español como en inglés.

5.2. Justificación y procesamiento de la selección

Después de evaluar estos cuatro corpus diferentes, seleccionamos el corpus 4 (Summarization CNN/DM) debido a sus características favorables: su estructura está ya dividida en conjuntos de entrenamiento, validación y prueba; su formato en CSV facilita la carga y el procesamiento de los datos; aunque su contenido no se encuentre en el idioma español, posteriormente veremos una alternativa para devolver resúmenes en español:

- **Formato y estructura:** El formato CSV es compatible con nuestras herramientas de procesamiento de datos del modelo que vamos a usar, permitiendo así, un uso fácil y una integración adecuada. Además, la estructura en conjuntos de train, validation y test, simplifica el trabajo y asegura que los datos estén listos para su uso inmediato.

- **Idioma:** Su idioma es el inglés, pero con métodos post-entrenamiento seremos capaces de trabajar y devolver resultados en español. Haremos uso de modelos pre-entrenados capaces de traducir el texto de entrada de español a inglés, para su uso en el nuestro modelo fine-tuned y viceversa a la hora de devolver un resumen generado.

- **Volumen de datos:** Con aproximadamente 1,3GB de datos, este corpus nos ofrece un gran volumen de muestras que es esencial para entrenar el modelo de manera robusta. Sin embargo, este gran volumen también presentó desafíos en términos de tiempo de entrenamiento y recursos computacionales.

Una vez seleccionado el corpus con el que vamos a trabajar, procedemos a su procesamiento para preparar los datos para el entrenamiento del modelo GPT-2. A continuación describimos los pasos clave:

- **Reducción del tamaño del conjunto de entrenamiento:** Aunque este corpus contenga un gran tamaño y sea una ventaja para la calidad del modelo, también puede ser una limitación debido a los altos requisitos de tiempo y recursos. Por ello, optamos por reducir el tamaño del conjunto de entrenamiento a una cantidad manejable sin sacrificar la representatividad de los datos (de 300 mil muestras a 50 mil, esto es, una sexta parte de los recursos del conjunto original). También nos aseguramos de que los textos seleccionados fueran suficientemente variados para entrenar el modelo eficazmente. Esto lo conseguimos con la función de filtrado vista anteriormente.

  En la *Figura 5.1* vemos el total de muestras para cada conjunto de datos (entrenamiento, validación y test). Hay que tener en cuenta que en el entrenamiento se harán 2 épocas, por lo que, aunque el número no sea tan elevado, mejorará mucho los resultados, así como el tiempo.

Conjunto	Muestras totales	Muestras recogidas
Entrenamiento	287.113	50.000
Validación	13.368	6.822
Test	11.490	500

Figura 5.1: conjuntos.

- **Tokenización:** Este es el proceso de convertir el texto original en una secuencia de tokens que el modelo pueda procesar. Para ello, utilizamos las herramientas integradas del modelo GPT-2, que incluyen un tokenizer optimizado. La tokenización correcta es fundamental porque afecta directamente la capacidad del modelo para comprender y generar texto de manera correcta.

  El funcionamiento del tokenizer de nuestro modelo se basa en el método de decodificación Byte Pair Encoding (BPE), que divide el texto en subpalabras en lugar de palabras completas. Este método tiene la ventaja de manejar tanto palabras comunes como palabras raras o nuevas de manera eficiente. El tokenizer convierte cada palabra en un identificador numérico que representa la subpalabra en el vocabulario del modelo. Este proceso sigue este orden:

- *Normalización del texto:* Esto es, convertir todo el texto a minúsculas y eliminar caracteres especiales no necesarios.
- *División en subpalabras:* Usa el algoritmo BPE para dividir palabras en unidades manejables.
- *Codificación:* Se asigna el identificador único a cada subpalabra.

Después de la tokenización, los datos se prepararon para el entrenamiento del modelo. Esto incluye:

- *Padding y truncación:* Nos aseguramos de que todas las secuencias tengan la misma longitud mediante el relleno (padding) de secuencias cortas y la truncación de secuencias más largas. Para ello, a la hora de cargar nuestros datos y recoger tanto los inputs como los targets, será necesario ajustar los parámetros 'truncation' y 'padding'. En nuestro trabajo hemos decidido poner estos valores para asegurar un correcto funcionamiento:

  Truncation = True
  Padding = 'max_length' (en este caso la longitud máxima será 768 por defecto)

- *Creación de conjunto de datos:* Dividimos el corpus tokenizado en lotes (batches) que el modelo pueda procesar en paralelo durante el entramiento.
- *Etiquetado:* Se asignan etiquetas a las secuencias de tokens para que el modelo pueda aprender a generar resúmenes a partir de los tokens.

6. Desarrollo experimental

En esta sección detallaremos el proceso experimental llevado a cabo para el entramiento y evaluación del modelo de resumen automático de textos. El objetivo principal es proporcionar una descripción exhaustiva de las etapas involucradas en el entrenamiento y análisis de los resultados obtenidos mediante el fine-tuning usando el corpus visto anteriormente.

6.1. Entrenamiento y validación

Explicaremos el procedimiento de entrenamiento y validación del modelo. Abordaremos aspectos como la selección de los datos, la configuración de hiperparámetros, así como las métricas usadas para evaluar su desempeño durante el entrenamiento. También describiremos los problemas y desafíos encontrados en el proceso, así como las soluciones implementadas para optimizar el rendimiento.

En cuanto a los datos utilizados, repasemos brevemente qué tenemos a nuestra disposición a la hora de empezar a entrenar:

- **Conjunto de entrenamiento:** es el conjunto que usaremos a la hora de ajustar nuestro modelo. Para ello, previamente habremos pasado la ruta del fichero CSV a nuestra función de lectura y filtrado para encargarnos de escoger aquellos datos que no excedan una longitud muy elevada. Posteriormente crearemos nuestros datasets y dataloaders para poder hacer uso de ellos en el entrenamiento. En consecuencia, obtendremos un total de **50.000 muestras** que el modeló usará para entrenarse. A estos datos se les aplicará truncamiento y padding, así como prepararlos para que el modelo sea capaz de entender su estructura.

  La estructura que tienen nuestros datos son tal que:

  - Artículo (texto de entrada): <|startoftext|> <|Resumen|> {texto} Summary: <|endoftext|>
  - Resumen (texto objetivo): {resumen} <|endoftext|>

  Hemos optado por estructurarlo de esta manera ya que el modelo entenderá como se forma un texto de entrada que es la concatenación de los tokens especiales seguido del texto del conjunto de datos y ‘Summary:’. El modelo sabe que lo que hay detrás del token ‘Resumen’ es el texto que debe tener en cuenta para resumir y a continuación tiene que ir el resumen. Por otro lado, el texto generado será simplemente un resumen terminado con el token de fin.

- **Conjunto de validación:** es el conjunto que usaremos una vez finalice cada época de entrenamiento, para comprobar el estado de éste. Realizaremos los mismos procesos que al conjunto de entrenamiento, pero está vez obtendremos un total de **6.822 muestras**, que serán suficientes para sacar resultados aceptables. Es posible obtener más muestras de este conjunto al aumentar el tamaño de tokens de entrada, pero disminuiremos la eficiencia.

Una vez tenemos los datos preparados para el entrenamiento, deberemos ajustar los hiperparámetros, tanto del optimizador como del scheduler:

- **Optimizador:** Estamos usando AdamW ya que es perfecto para modelos Transformer y produce buenos resultados. Los parámetros usados:
  - *Learning_rate:* Es un factor de escala que determina cuánto se ajustan los pesos del modelo. Tras muchas pruebas, lo ideal para nuestro caso es **4e-5**. Es posible disminuir un poco más este número, pero sería necesario aumentar los datos de entrada.
  - *Epsilon:* Asegura la estabilidad numérica durante el cálculo de la actualización de pesos. Hemos elegido el valor estándar **1e-8**, el cual funciona correctamente en nuestro modelo.
  - *Weight_decay:* Ayuda a prevenir el sobreajuste penalizando los grandes pesos. Usamos un valor de **1e-2**. Optamos por incluir este parámetro ya que hemos tenido bastantes problemas con sobreajustes en el modelo.

- **Scheduler:** Utilizamos el scheduler proporcionado por la biblioteca transformer, que ajusta la tasa de aprendizaje duramente el entrenamiento. Sus parámetros son:
  - *Optimizer (optimizador):* Escogemos el que hemos definido ahora para calcular los pesos del modelo.
  - *Num_warm_steps (pasos de calentamiento)*: Es el número de pasos durante los cuales la tasa de aprendizaje aumenta linealmente desde cero hasta el valor asignado. Nosotros hemos decidido aumentar un poco el valor estándar a **120** para garantizar que un buen comienzo de entrenamiento.
  - *Num_training_steps (pasos totales):* Es el número total de pasos de entrenamiento. Determina cuanto tiempo se reducirá la tasa de aprendizaje después del calentamiento. Usamos un valor calculado como la longitud del dataloader de entrenamiento por el número de épocas.

- **Epochs (épocas):** Este parámetro controla cuantas veces el algoritmo de aprendizaje ve el conjunto completo de datos de entrenamiento. Usaremos **2** para nuestro modelo. Si elevamos este número aumentaremos la probabilidad de sobre ajustar nuestro modelo, si disminuimos, lo contrario.

- **Batch:** Es el número de muestras de entrenamiento que se procesan juntas en una sola pasada a través de la red neuronal antes de actualizar los parámetros del modelo. Es uno de los parámetros más importantes y depende también del hardware accesible. En nuestro caso, tras varias pruebas, hemos comprobado que un batch_size de **4** es perfecto. Obtenemos buenos tiempo y resultados.

Ya ajustado nuestro modelo, estamos listos para empezar el bucle de entrenamiento. El funcionamiento lo hemos visto en puntos anteriores, por lo que aquí nos centraremos en interpretar las muestras que se van generando conforme avanza nuestro entrenamiento. Para ello, hemos creado una pequeña condición dentro del bucle, que generará muestras cada 100 batches, es decir, cada 400 muestras, ya que nuestro batch_size es 4. En el código *6.1* observamos el código. El código funciona tal que:

- Ponemos el modelo en modo evaluación (*model.eval()*).
- Se generan muestras y se guardan con los siguientes parámetros:
  - *Input_ids:* IDs de los tokens que representan la entrada del modelo.
  - *Attention_masks:* Máscaras que indican qué tokens deben ser atendidos.
  - *Do_sample=True:* Indica que se debe utilizar muestreo para la generación.
  - *Top_k=50:* Considera solo los 50 tokens más probables.
  - *Max_length=1024*: Longitud máxima del texto generado (también para GPT-2)
  - *Top_p=0.95:* Incluye tokens cuya probabilidad acumulada es menor a 0,95.
  - *Num_return_sequences=1:* Número de secuencias de texto que se devuelven.
  - *Num_beams=2:* Numero de haces de búsqueda.
  - *No_repeat_ngram_size=2:* Evita la repetición de n-gramas de tamaño 2.
  - *Early_stopping=True:* Detiene la generación si se alcanza una secuencia completa.
  - *Repetition_penalty=2.0:* Penaliza la repetición de tokens.
- Una vez tenemos las muestras, se separa el texto de entrada de los resúmenes generados. Esto es así por la estructura que hemos decidido, donde comienza con el artículo seguido de 'Summary:' y el resumen. Se guarda el resumen limpio en new_tokens.
- Imprimimos por pantalla para cada batch el artículo de entrada, su resumen esperado y el resumen generado por el modelo.
- Nos aseguramos de volver el modelo a modo entrenamiento (*model.train()*), para continuar con el bucle correctamente.

A continuación, mostraremos unas muestras conforme el entrenamiento avanza y comprobaremos que tanto el porcentaje de pérdida como los resúmenes generados mejoran:

Veamos el resultado al empezar el entrenamiento. Comprobamos que el resumen generado carece de sentido y está formado por palabras sueltas y símbolos extraños. Esto irá mejorando conforme avance el entrenamiento y se irá acercando al resumen esperado. Además, si comprobamos la pérdida, vemos que es mayor al 5%, lo cual es inaceptable:

Artículo:

Georgios Sanaras habló de su orgullo tras marcar el dramático penalti que envió a Grecia a los octavos de final del Mundial de Brasil. El exdelantero del Celtic convirtió con frialdad un penalti en el tiempo de descuento en Fortaleza para darle a su equipo una victoria por 2-1 sobre Costa de Marfil y el segundo lugar en el Grupo C. Los africanos parecían pasar a la fase eliminatoria después de que Wilfried Bony del Swansea cancelara el anuncio. Tras un gol de Andreas Samaris, Grecia, que estrelló tres veces en el palo, se llevó el premio final por su actuación positiva. Los jugadores de Grecia corren hacia Samaras después de que el delantero llevara al equipo a los octavos de final por primera vez. Samaras, liberado por el Celtic al final de la temporada pasada, fue derribado por Giovanni Sio en el área en los instantes finales y se levantó para dar la puntilla. Sanaras, cuyo gol fue su primer gol a nivel internacional en dos años, dijo a BBC Sport.

Resumen esperado:

Andreas Samaris adelantó a los Grooks antes de que Wilfried Bony empatara. El agente libre Samaras ganó un penalti polémico y acarició la manguera para dejar a Grecia segunda en el Grupo C. Se enfrentarán a Costa Rica en octavos de final.

Resumen generado:

Un hombre fue arrestado en la capital de India, Delhi, dijo la policía el domingo.

La policía ha dicho: "Un plan tiene a los funcionarios de la ciudad de Undon en casa en Twitter, foto india, menos, más ruso", tuiteó el presidente P He (Rusia Twitter y 1 South TV con National People, que An Ramhe Pr p. 3. PF ine No Re R L And 3 como After hours VN PH She - Me twout: There One Over Now England star D es para coofficial M As they Se United - Su jd Che ni siquiera EU Más Philadelphia r & Los 1 San durante la línea anterior Red 5 6 one 10 e 8 Mostrar o

Todo esto con Google Ver Reddit Go Ellos Tweet Imagen video noticias

A continuación, se muestra el resultado tras unas cuantas horas entrenando. Ya vamos viendo como el resumen generado va tomando coherencia y el mensaje se asemeja a lo que transmitir el artículo. El porcentaje de pérdida ronda el 2%.

Artículo:

“Cuatro personas murieron el jueves por la tarde y otras dos resultaron heridas en un ataque con pistola y cuchillo en una oficina tribal de nativos americanos en Alturas, California. El jefe de policía dijo que Sherrie Rhoades, de 44 años, asistía a una reunión de desalojo tribal en la oficina tribal de Cedarville Rancheria el jueves alrededor de las 3:30 p.m. cuando abrió fuego. Cuando Rhoades se quedó sin munición, agarró un cuchillo carnicero de la cocina y atacó a otra persona y la apuñaló. Las víctimas fallecidas fueron identificadas como una mujer de 19 años, una mujer de 45 años, una mujer de 30 años. un anciano y un hombre de 50 años. El tiroteo ocurrió en el condado de Modoc, en el norte de California, en la frontera con Oregón. Las autopsias de los fallecidos están previstas para el viernes”.

Resumen esperado:

“Cuatro víctimas fallecidas y dos personas están heridas y se encuentran en estado crítico”, afirma el jefe de policía. Las autoridades dijeron que una mujer disparó contra la gente antes de usar un cuchillo de carnicero. La sospechosa fue identificada como Sherrie Rhoades, una exlíder tribal. La policía dijo que el sospechoso se enfrentaba al desalojo”.

Resumen generado:

“Otros dicen o un pero que más no se trata del mes, dicen que también tendrá el año, victimas fallecidas la semana pasada, ese segundo, todo su uso puede ser la muerte, podría antes de que las autoridades de ayuda, usted, cinco, área única, hogar familiar. días cuatro utilizados ahora a través de la ley. El sitio de vida de la investigación pública tenía noticias del conocido sospechoso”.

Una vez termina una época, aparece por consola algo parecido a la *Figura 6.1*. Vemos la pérdida media del conjunto de entrenamiento y el conjunto de evaluación, así como el tiempo que ha tardado la época. Una pérdida del 0,56% es más que sobresaliente para un conjunto de datos tan complejo como el nuestro que abarca muchos tópicos y temas de noticias. Una pérdida del 0.68% para la validación también es aceptable teniendo en cuenta que el conjunto era bastante menor que el de entrenamiento, por lo que, si aumentamos los datos, probablemente mejore.

Una vez termina por completo el entrenamiento aparece el tiempo total en horas, minutos y segundos. Comprobamos que nuestro modelo ha tardado aproximadamente 7 horas, lo que es un tiempo algo elevado, pero que hubiese sido mayor si no hubiésemos filtrado y reducido nuestro conjunto de datos.

Pérdida media en entrenamiento:	**0.56 (sobre 100)**
Pérdida media en validación:	**0.62 (sobre 100)**
Tiempo de entrenamiento de la época:	**3:15:46**
Tiempo total de entrenamiento:	**7:14:36**

Figura 6.1: pérdida y tiempo transcurrido.

6.2. Evaluación

Este apartado lo dedicaremos a la evaluación final del modelo. Para ello hemos creado un script a parte donde podremos escribir nuestro propio texto en español y nos devolverá su resumen correspondiente.

Primero vamos a ver cómo funciona el código por partes, ya que ha sido necesario hacer una serie de pasos previos a la hora de generar nuestro resumen:

- **Función de traducción:** Como ya comentamos en puntos anteriores, el modelo ha sido entrenado con un corpus en inglés debido a sus ventajas respecto a los demás. Debido a esto, nuestro modelo solo será capaz de generar resúmenes coherentes si le pasamos como entrada un texto en inglés también.
  Para ello, hemos creado una función que toma el modelo pre-entrenado MarianMT para traducir el texto de entrada y el resumen de salida. En el *código 6.2* tenemos la implementación de esta función:
  - Cargamos el modelo y el tokenizer.
  - Tokenizamos el texto de entrada o el resumen
  - Generamos la traducción y la devolvemos decodificada.

- **Función de generación:** También necesitaremos crear una función con nuestro modelo que se encargará de generarnos nuestro resumen. La implementación es parecida a la que tuvimos en el bucle de entrenamiento a la hora de generar muestras, con la diferencia de que aquí no debemos usar los parámetros do_sample, top_k y top_p ya que solo son válidos en las muestras. En el *código 6.3* tenemos la implementación:
    - Tokenizaremos el texto de entrada y guardaremos tanto sus input_ids como sus máscaras de atención.
    - Generaremos un resumen de la misma manera que al generar muestras, con los mismos parámetros menos los mencionados anteriormente.
    - Separaremos el texto de entrada del resumen generado y lo devolveremos

Definiremos la función principal __main__ que se encargará de ejecutar todo. En el *código 6.4* tenemos las cargas de los modelos, para el generador de resúmenes cargamos nuestra carpeta donde lo guardamos previamente y para el traductor de textos usamos el pre-entrenado:

Una vez cargados los modelos, podemos generar los resúmenes y traducirlos. Hemos separado tanto el texto de entrada en inglés y en español, como los resúmenes generados en inglés y en español para mostrarlos por consola y confirmar que se han generado correctamente. En el *código 6.5* tenemos el código:

- Usaremos la función translate creada anteriormente pasándole como parámetros el prompt, el modelo de la traducción de español a inglés y el tokenizer correspondiente al modelo.
- Generaremos nuestro resumen en inglés con la función de generación creada pasándole por entrada la concatenación de los tokens especiales y el prompt traducido que es lo único que entiende nuestro modelo.
- Traduciremos nuestro resumen en ingles con la función translate y el modelo inglés a español, así como su tokenizer.

Ahora deberemos poner a prueba nuestro script y con ello, nuestro modelo. Utilicemos un prompt en español lo suficientemente largo como para que el modelo genere un resumen aceptable.

Prompt: “Según han informado a Europa Press fuentes del dispositivo de rastreo, participan en la búsqueda efectivos de Protección Civil de Verín y de la Guardia Civil, además de brigadas de Medio Rural del Distrito XIV de Verín. La búsqueda se centra este jueves en el pueblo de Oímbra y en la zona de monte de los alrededores, han señalado las mismas fuentes, que han destacado que por el momento no hay ninguna pista sobre el hombre. La familia del desaparecido denunció su ausencia este jueves por la mañana. Se trata de un ciudadano portugués de 62 años que vivió hasta hace un mes, han apuntado las mismas fuentes, en Oímbra”

Al ejecutar el script vemos estos resultados:

Texto traducido al inglés:

(El mismo prompt, pero en inglés. Confirmamos que traduce bien del español al inglés.)

Resumen generado en inglés:

“As for how it was, it was not the police, or they said: he also disappeared Thursday morning; (could be after sixty-two year up not to say used) in now or vera - everything I had found last since any use still time at home a policeman arrested”

Resumen traducido al español:

“De por como con era si su era la policía no o ellos un pero que dijo: también desaparecido jueves por la mañana; (podría ser después de sesenta y dos año para arriba no decir utilizado) en ahora o vera - todo lo que había encontrado último desde cualquier uso todavía tiempo en casa un policía arrestado”

Los resultados vistos en nuestro modelo son adecuados teniendo en cuenta la simplicidad del entrenamiento. Se pueden sacar las palabras clave y los puntos destacables del resumen y entender sobre que trata el texto de entrada. Para una mejor respuesta del modelo se debería entrenar con conjuntos de datos más extensos y mejor preparados, así como ir jugando con los hiperparámetros.

Tras haber observado algunos ejemplos de generación de nuestro modelo, debemos evaluar los resultados, para ello se han utilizado las siguientes métricas de evaluación:

- *ROUGE-1:* Que mide la superposición de unigramas entre el resumen generado y los resúmenes de referencia.
- *ROUGE-2:* Lo mismo que el ROUGE-1 pero para la superposición de bigramas.
- *ROUGE-L:* Mide la longitud de la subsecuencia común más larga.

Los resultados oscilan entre 0 y 1, las puntuaciones más altas indican una mayor similitud entre el resumen producido automáticamente y la referencia.

En el capítulo 8 observaremos mediante gráficas y números los resultados obtenidos.

Con esto concluiría el desarrollo experimental del modelo, y ahora nos centraremos en implementarlo en la aplicación móvil para un uso más adecuado e importante.

7. Aplicación móvil

7.1. Descripción de la aplicación

La aplicación móvil desarrollada tiene como objetivo principal ofrecer una herramienta que permita a un usuario generar automáticamente resúmenes de texto a partir de entradas proporcionadas, ya sea en formato de texto o archivos PDF. Esto busca facilitar la comprensión y el análisis rápido de grandes volúmenes de información, además de poner a prueba nuestro modelo entrenado para esta tarea.

Funcionalidades principales:

1. Entrada de texto:

- Los usuarios pueden escribir o pegar texto directamente en la aplicación para que sea resumido al instante.

- La interfaz de usuario proporciona un campo de texto claro y fácil de usar donde se puede ingresar el contenido.

2. Carga de archivos PDF:

- Los usuarios también tienen la opción de cargar archivos PDF desde sus dispositivos.

- La aplicación procesa estos archivos para extraer el texto contenido y generar un resumen a partir de él

3. Generación de resúmenes:

- La aplicación utiliza nuestro modelo basado en GPT-2, el cual hemos entrenado previamente, para generar el resumen.

- La generación se realiza de manera eficiente, utilizando la infraestructura de un servidor local para manejar el procesamiento.

4. Salida en formato texto y PDF:

- El resumen generado se muestra directamente en la aplicación para una revisión inmediata:

- Además, el usuario puede descargar el resumen en formato PDF, permitiendo su almacenamiento fácil.

Estas funcionalidades se integran en una aplicación móvil desarrollada con Flutter, garantizando una experiencia de usuario más fluida y teniendo, además, compatibilidad multiplataforma (Android, IOS, Windows, Linux, etc.). Firebase se utiliza para gestionar el almacenamiento y la autenticación, asegurado que los datos del usuario se manejen de manera segura.

7.2. Integración del modelo en la aplicación

7.2.1. Aspectos técnicos

Como bien dicho antes, la aplicación móvil está desarrollada con Flutter, un framework de desarrollo de aplicación que permite crear interfaces de usuario nativas para IOS y Android utilizando una única base de código. Elegimos Flutter por su facilidad de uso, capacidad para rápida iteración y su amplia comunidad de soporte.

El modelo entrenador se aloja en un servidor backend implementado en Python utilizando el framework web Flask. Este servidor maneja las solicitudes del resumen, procesa el texto o el contenido del PDF y devuelve el resumen generado. En el *código 7.1* podemos ver su código simplificado. Hablemos un poco de él:

- Primero cargamos tanto los tokenizadores y modelos tanto para traducir textos como el nuestro de generación de resúmenes.

- Tenemos dos tipos de entrada, texto y archivos.

- Si es texto, recoge el texto en español y lo traduce al inglés para su correcta interpretación en el modelo, luego se tokeniza y se genera un resumen. Se vuelve a traducir al español y se devuelve con codificación utf-8.
- Si es archivo, deberemos comprobar si es PDF primero. Es caso correcto leeremos las páginas y lo volcaremos en una variable. Luego, haremos lo mismo que para texto, traducimos al inglés, tokenizamos texto, generamos resumen y lo devolvemos traducido al español.

- Abrimos el servidor en el host 0.0.0.0:5000, en local.

Este servidor backend maneja las solicitudes de resumen y utiliza los modelos fine-tuned y traducción para el procesamiento del texto. El archivo ‘app.py’ contiene las rutas y la lógica del procesamiento, incluyendo la importación de nuestro modelo, la configuración de las rutas API, y la gestión de solicitudes y respuestas.

En cuanto al código frontend, el archivo summary_provider.dart (Flutter), gestiona las solicitudes al backend y actualiza la interfaz de usuario. Utiliza la librería ‘provider’ para manejar el estado de la aplicación y ‘http’ para realizar las solicitudes propias. En el código *7.2* tenemos el código frontend simplificado.

7.2.2. Interfaz de usuario

La interfaz de usuario está diseñada para ser intuitiva y fácil de usar, con una navegación básica que permite al usuario ingresar texto o cargar ficheros para obtener un resumen rápidamente. La interfaz consta de dos pantallas:

1. Pantalla principal. *Imagen 7.1*:

- Campo de entrada de texto para que los usuarios escriban o peguen contenido

- Botón para seleccionar y cargar archivo PDF desde el dispositivo.

- Botón para enviar la solicitud del resumen.

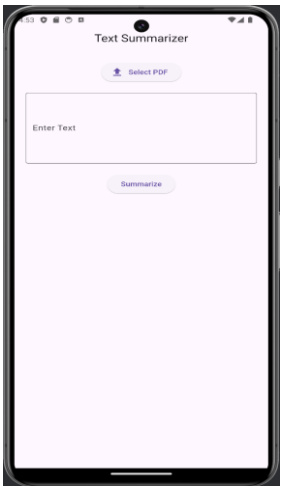

Imagen 7.1: interfaz pantalla principal.

2. Pantalla de resultados. *Imagen 7.2*:

- Área de visualización donde se muestra el resumen generado.

- Botón para descargar el resumen como archivo PDF.

Imagen 7.2: interfaz pantalla resultados.

El archivo ‘main.dart’ contiene la lógica de la interfaz de usuario. Incluye la estructura básica de la aplicación, la configuración del estado mediante ‘provider’ y la interacción con el usuario a través de formularios y botones. La funcionalidad principal incluye la captura del texto o PDF, el envío de la solicitud al backend y la visualización del resumen generado.

7.3. Pruebas y resultados

7.3.1. Casos de uso

En esta sección detallaremos las pruebas realizadas para evaluar la funcionalidad y precisión de la aplicación de móvil con la implementación del modelo ajustado para la tarea de resúmenes de texto, así como los resultados obtenidos.

Para probar la aplicación, se definieron varios casos de uso que cubren distintos escenarios posibles:

1. Texto corto:

- *Descripción:* Introducir manualmente un texto breve en la interfaz de la aplicación.

- *Procedimiento:* Ingresamos un párrafo corto (aproximadamente 100 palabras) y se solicitó un resumen mediante el botón 'Summarize'.

- *Resultado:* El modelo generó un resumen conciso y corto, manteniendo las palabras clave y los puntos importantes del texto original.

2. Texto largo:

- *Descripción:* Introducir manualmente un texto extenso en la interfaz de la aplicación.

- *Procedimiento:* Ingresamos un artículo (aproximadamente 800-1000 palabras) y se solicitó un resumen mediante el botón 'Summarize'.

- *Resultado:* El modelo generó un resumen conciso y de tamaño adecuado, destacando los puntos principales del texto.

3. Documento PDF:

- *Descripción:* Subir un documento PDF desde el dispositivo móvil.

- *Procedimiento:* Se cargo un PDF de varias páginas mediante el botón superior 'Select PDF' y se solicitó un resumen mediante el botón 'Summarize'.

- *Resultado:* La aplicación extrajo correctamente el texto del PDF y generó un resumen a partir de él.

4. Texto técnico:

- *Descripción:* Introducir un texto con terminología técnica específica.

- *Procedimiento:* Ingresar fragmento de texto o cargar documento PDF científico y solicitar resumen.

- *Resultado:* El modelo logró generar un resumen, aunque algunas limitaciones en la simplificación de terminologías técnicas.

7.3.2. Feedback, ajustes y resultados cuantitativos

Tras las pruebas iniciales, se recopiló feedback de usuarios para identificar áreas de mejora y realizar ajustes en la aplicación:

1. Interfaz de usuario:

- *Problema:* Algunos usuarios encontraron la interfaz confusa, especialmente a la hora de cargar documentos PDF.

- *Solución:* Se simplificó la interfaz de carga de archivos, añadiendo instrucciones claras y mejorando la accesibilidad de los botones.

2. Precisión del modelo:

- *Problema:* En algunos casos, el resumen generado no capturaba adecuadamente la intención del texto original.

- *Solución:* Se ajustaron los parámetros del modelo para mejorar la precisión de los resúmenes.

3. Tiempo de respuesta:

- *Problema:* Los usuarios notaron largos tiempos de espera para textos largos y documentos PDF.

- *Solución:* Se optimizó el código del procesamiento de texto y se implementaron mensajes de progreso para mejorar la experiencia del usuario.

También se realizaron pruebas cuantitativas para medir la efectividad del modelo en términos de precisión y tiempo de procesamiento.

1. Precisión:

- *Método:* Se utilizó la métrica ROUGE (Recall-Oriented Understudy for Gisting Evaluation) para evaluar la calidad de los resúmenes.

- *Resultados:* Los resúmenes generados alcanzaron un puntuaje buen puntuaje, lo cual es competitivo en comparación con otros modelos de generación automática de resúmenes. Lo veremos en el capítulo 8 con detalles.

2. Tiempo de procesamiento:

- *Método:* Se midió el tiempo de procesamiento para textos de diferentes longitudes y documentos PDF.

- *Resultados:* El tiempo promedio para generar un resumen de un teto corto fue de 3 segundos, mientras que para textos largos y documentos PDF fue de aproximadamente 10-15 segundos.

Conclusiones

Las pruebas demostraron que la aplicación es capaz de generar adecuadamente los resúmenes tanto para textos cortos como largos, así como para documentos PDF. Aunque se identificaron algunas áreas de mejora, los ajustes realizados en la interfaz de usuario y en los parámetros del modelo mejoraron la experiencia de usuario de manera significativa.

8. Resultados finales y discusión

8.1. Análisis de los resultados obtenidos

El análisis de los resultados obtenidos se enfoca en evaluar la efectividad del modelo implementado en la aplicación móvil comparándolo con algunos benchmarks establecidos en el campo de la generación automática de resúmenes.

Para contextualizar los resultados obtenidos con nuestro modelo, se realizaron comparaciones con otros modelos de generación de resúmenes existentes, utilizando métricas estándar como ROUGE (Wikipedia, ROUGE (métrica), 2024).

Para ello, se creó un script con el cual calcularemos las métricas obtenidas de varios modelos capaces de generar resúmenes de texto. Compararemos los resultados de estos modelos con el de este trabajo:

- *GPT-2 Original:* El modelo original sin fine-tuning, se utilizó como referencia para comparar la mejora obtenida tras el entrenamiento específico en nuestro corpus de datos.
- *BERTSUM:* Es un modelo basado en BERT optimizado para la tarea de generación de resúmenes, conocido por su alta precisión en benchmarcks de resúmenes, tanto extractivos como abstractivos.
- *PEGASUS:* Se trata de un modelo desarrollado por Google Research específicamente para la tarea de resúmenes, que ha demostrado resultados sobresalientes en diversos conjuntos de datos de evaluación.

Para comparar los resultados de estos modelos deberemos primeramente cargar en nuestro código todos y cada uno de los modelos que vayamos a analizar, así como sus respectivos tokenizers, usando la biblioteca Transformers. *Código 8.1*.

Usaremos en este caso un **conjunto de test** recogido anteriormente junto a los conjuntos de entrenamiento y validación, también contenido en formato CSV. Este conjunto está compuesto por 5.727 muestras, pero solo recogeremos **500 muestras** de ellas para esta tarea. El motivo de esto es que debemos filtrar nuestros datos previamente para adecuarlos a todos los modelos con los que vamos a trabajar. En este caso, los modelos GPT-2 original y el ajustado solo pueden recibir entradas máximas a 1024 tokens, en cambio, los modelos BERTURM y PEGASUS la mitad, 512 tokens.

Este obstáculo lo resolvemos utilizando la misma función de filtrado de datos usada previamente en el entrenamiento de nuestro modelo, pero adecuándolo a este caso, es decir, reduciendo la cantidad de tokens máximos a 400 aproximadamente (margen hasta 512 tokens) y utilizar el tokenizer adecuado para cada uno de los modelos. En el *código 8.2* tenemos un ejemplo de la función 'load_and_filter_data' adecuada a este contexto.

Habiendo cargado, filtrado y recogido los datos a usar, podemos generar resultados a partir de las generaciones de resúmenes de cada modelo. Para ello, generaremos resúmenes de la misma manera que hemos hecho durante el entrenamiento y evaluación, y calcularemos para cada modelo su métrica ROUGE pasándole el resumen de referencia y el resumen generado. El valor de cada modelo se guardará y se calculará la media de todo el conjunto de datos por separado.

Los datos recogidos son ROUGE-1, ROUGE-2 y ROUGE-L, explicados anteriormente en el apartado 6.2.

En cuanto a los resultados:

MODELO	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**
GPT-2 fine-tuned	0,24	0,14	0,17
GTP-2 original	0,21	0,12	0,15
BERTSUM	0,19	0,11	0,14
PEGASUS	0,26	0,10	0,19

Figura 8.1: tabla resultados métrica.

También recogidos en gráfico de barras:

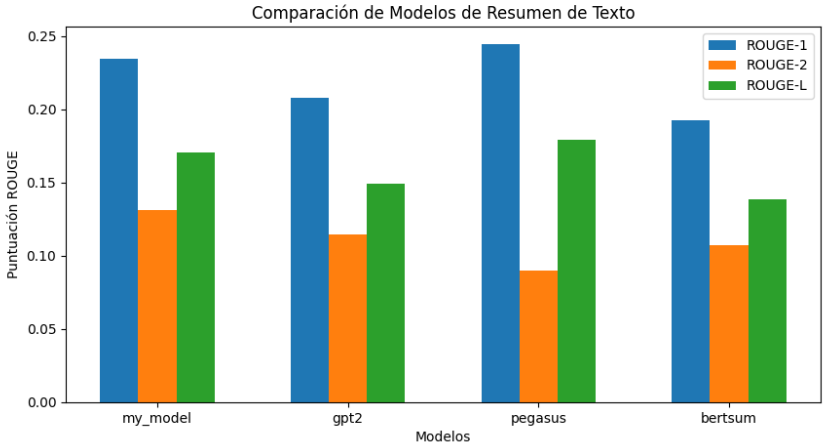

Figura 8.2: gráfica resultados métrica.

Si comparamos nuestro modelo fine-tuned con el original, el nuestro superó significativamente al GPT-2 original y BERTSUM en todas las métricas, destacando la importancia del fine-tuning en un conjunto de datos específico para mejorar así la calidad de los resúmenes generados.

Si lo comparamos con PEGASUS vemos que nuestro modelo no alcanzó los puntuajes de este modelo más avanzado, Aun así, los resultados obtenidos son competitivos, especialmente considerando la simplicidad y eficiencia del modelo implementado. La ligera desventaja de las métricas sugiere un área de mejora, como puede ser el ajuste de hiperparámetros y la incorporación de técnicas avanzadas de preprocesamiento y post procesamiento.

Los resultados obtenidos demuestran que nuestro modelo de generación automática de resúmenes es capaz de producir resúmenes de manera adecuada, con un desempeño ligeramente inferior a los modelos de última generación, pero siendo robusto y adecuado para aplicaciones prácticas como nuestra aplicación móvil.

8.2. Limitaciones y futuras líneas de investigación

En este apartado, discutiremos las limitaciones encontradas durante el desarrollo del proyecto y propondremos posibles mejoras y futuras líneas de investigación para superar estos desafíos y continuar avanzando en la generación automática de resúmenes de texto.

8.2.1. Desafíos encontrados

Limitaciones del modelo:

- *Capacidad de generalización:* Aunque nuestro modelo ajustado sobre el corpus específico mostró mejoras notables, su capacidad de generalización a textos fuera de su dominio aún es limitada. Esta restricción puede afectar la utilidad del modelo en aplicaciones que requieran versatilidad y adaptabilidad en diferentes contextos o estilos de escritura.
- *Tamaño y complejidad del modelo:* Aunque GPT-2 es potente, existen modelo más avanzados como GPT-4 que ofrecen capacidades superiores. La elección de GPT-2 fue una decisión basada en la eficiencia y disponibilidad a ser de código abierto, pero limita la calidad de los resúmenes en comparación con modelos más grandes.
- *Procesamiento de datos:* El preprocesamiento de los textos, especialmente en el caso de los archivos PDF, presentó desafíos significativos. La extracción de texto de PDFs puede ser inconsistente, lo que también afecta a la calidad del texto de entrada y, en consecuencia, al resumen generado.
- *Limitaciones computacionales:* La capacidad de procesamiento de una gráfica dedicada Nvidia RTX 3060 (usada para este proyecto), aunque sea adecuada para muchas tareas, impone restricciones en términos de ‘batch size’ y tiempo de entrenamiento. Modelos más complejos requerirían recursos mucho mayores.
- *Evaluación de resultados:* Las métricas de evaluación utilizadas, como ROUGE, proporcionan una buena medida, pero no capturan completamente aspectos como la coherencia, fluidez o relevancia del contenido resumido.

Desafíos técnicos y prácticos:

- *Integración en aplicaciones móviles:* La implementación de nuestro modelo en una aplicación móvil implicó desafíos relacionados con la gestión de recursos limitados y la optimización del tiempo de respuesta.
- *Dependencias y herramientas:* La configuración y gestión de dependencias, especialmente al integrar Firebase y Flutter en la aplicación móvil, presentó varios problemas técnicos que requirieron soluciones como conflictos de versiones o errores de configuración.
- *Manejo de errores y excepciones:* Asegurar que la aplicación maneje adecuadamente los errores, especialmente en escenarios donde se suben archivos PDF con formatos no soportados o textos que pueden ser procesados de manera incorrecta.

8.2.2. Sugerencias e investigaciones futuras

Mejoras en el modelo:

- Entrenamiento en conjuntos de datos más amplios y diversificados: Ampliar el conjunto de datos de entrenamiento para incluir una mayor variedad de dominios y estilos de escritura mejoraría la capacidad de generalización del modelo. Además, se podría incorporar técnicas de 'data argumentation' para ayudar a crear un conjunto más robusto.
- Adopción de modelos más avanzados: Considerar la utilización de modelos más avanzados como GPT-4 o BART podría mejorar significativamente la calidad. Sin embargo, esto requeriría una mayor capacidad computacional y optimización del uso de recursos.
- Técnicas de preprocesamiento avanzadas: Mejorar las técnicas de preprocesamiento para la extracción de texto, especialmente en archivos PDF, podría ser de gran ayuda. También, herramientas como OCR (Reconocimiento Óptico de Caracteres) y procesamiento de documentos se pueden incorporar para manejar mejor las variaciones en el formato de los archivos.
- Optimización de hiperparámetros: Realizar una búsqueda más exhaustiva de hiperparámetros podría llevar a mejorar el rendimiento del modelo. Métodos como 'grid search' o 'Bayesian' también son muy útiles en este sector.

Mejoras en la aplicación:

- *Interfaz de usuario mejorada:* Desarrollar una interfaz más fluida y funciona, que facilite la interacción del usuario con la aplicación. Además, se podría incluir guías y mensajes de error claros.
- *Soporte multi-idiomas:* Expandir la capacidad del modelo para manejar múltiples idiomas podría ampliar su aplicabilidad. Esto requeriría el fine-tuning del modelo en conjuntos de datos en diversos idiomas.
- *Optimización para dispositivos móviles:* Mejorar la eficiencia de la aplicación móvil para reducir el consumo de recursos y aumentar la velocidad de procesamiento. Técnicas como la compresión del modelo y optimización del código son una posible opción.
- *Manejo de archivos:* Ampliar las capacidades de manejo de archivos para soportar no solo PDFs sino también otros formatos de documentos comunes. Así como también implementar funciones de previsualización y edición de texto.

Investigaciones futuras:

- *Integración con otras herramientas y APIs:* Explorar la integración con otras herramientas para ampliar las funcionalidades de la aplicación. Por ejemplo, combinar la generación de generación de resúmenes con herramientas de análisis de sentimiento o extracción de entidades.
- *Evaluación cualitativa:* Involucrar a usuarios en pruebas de usabilidad y recolección de feedback detallado puede proporcionar información valiosa para futuras mejoras e implementaciones.
- *Exploración de nuevos algoritmos:* Investigar nuevos algoritmos y arquitecturas que puedan superar las limitaciones actuales de los modelos basados en Transformers. Los modelos híbridos que combinas técnicas simbólicas y de aprendizaje profundo podrían ofrecer nuevas perspectivas.

Aunque el modelo y la aplicación desarrolladas han demostrado resultados prometedores, existen varias áreas de mejora que pueden ser exploradas en futuras investigaciones. Abordar limitaciones actuales y aprovechar las oportunidades de mejorar, nos permitirá avanzar en nuestra tarea, así como ampliar su aplicabilidad en diferentes contextos y dominios.

9. Conclusiones

9.1. Resumen de hallazgos clave

En el desarrollo del trabajo hemos realizado importantes avances en la implementación y evaluación de un sistema de generación automática de resúmenes de texto utilizando aprendizaje automático, específicamente modelos de lenguaje pre-entrenados de tipo Transformer. A continuación, presentamos los hallazgos que han sido clave en esta investigación:

Efectividad del modelo GPT-2 fine-tuned:

- El modelo usado y ajustado demostró ser efectivo en la generación de resúmenes, tanto para textos cortos como para documentos PDF de mayor longitud.
- Además, los resultados obtenidos en las métricas indicaron que el modelo alcanzó un puntuaje adecuado, posicionándose competitivamente frente a otros modelos de generación de resúmenes.

Tipo de procesamiento:

- El tiempo promedio de las generaciones estuvo en un intervalo de 3 segundos y 15 segundos entre textos cortos, largos y documentos PDF. Esto demuestra la viabilidad del modelo para aplicaciones en tiempo real, como nuestra aplicación móvil creada desde 0, aunque con opciones para optimizaciones futuras.

Pruebas de usabilidad y feedback de usuarios:

- La aplicación móvil desarrollada permitió a los usuarios cargar documentos y obtener resúmenes. Las pruebas realizadas de usabilidad indicaron una buena aceptación de la interfaz de usuario tras algunos ajustes basados en el feedback recibido.
- Se mejoraron aspectos como la carga de archivos y la precisión del modelo mediante ajuste en los parámetros y optimización del código.

Casos de uso:

- Los casos de uso cubrieron la generación de resúmenes de texto extensos, documentos PDF y textos técnicos. En todos los casos, el modelo logró generar resúmenes adecuados, aunque se identificaron limitaciones en varios apartados.

Áreas de mejora e investigaciones futuras:

- Se identificaron varias áreas de mejora, incluyendo la optimización del modelo para reducir el consumo de recursos y aumentar la velocidad de procesamiento, así como la expansión de la capacidad para manejar múltiples idiomas.
- Se sugirió también la integración de la aplicación con otras herramientas y APIs para ampliar sus funcionalidades y realizar una evaluación más profunda.

9.2. Contribuciones del TFG

También hablaremos de las contribuciones que se han logrado en este ámbito. A continuación, se detallan las principales:

1. Desarrollo de un modelo de resumen automatizado

Se ha desarrollado y optimizado un modelo que demuestra ser efectivo en la generación de resúmenes. Además, hemos sido capaces de ajustarlo específicamente para su tarea con textos en español, superando desafíos relacionados con la adaptación de modelos originales entrenados en inglés.

2. Implementación de una aplicación móvil

El TFG ha incluido el diseño e implementación de una aplicación móvil que permite a resumir textos de manera más sencilla e intuitiva. Esta aplicación no solo facilita el acceso a la funcionalidad del modelo, sino que también mejora la experiencia del usuario.

3. Mejora en la interfaz de usuario

Se mejoró la interfaz de la aplicación, abordando problemas iniciales de usabilidad e incorporando guías y mensajes de error claros. Estas mejoras han permitido una interacción más eficiente y comprensible.

4. Contribuciones al conocimiento en el procesamiento de lenguaje natural (PLN)

Este TFG también ha aportado conocimientos valiosos sobre la adaptación y optimización de modelos de lenguaje pre-entrenados en idiomas distintos al inglés. Además, se han explorado técnicas de preprocesamiento y tokenización que mejoran la calidad del texto de entrada y, en consecuencia, los resúmenes. Estas contribuciones pueden servir como base para futuras investigaciones.

El proyecto no solo ha contribuido al conocimiento técnico en el área del lenguaje natural, sino que también ha implementado mejoras prácticas en la experiencia del usuario, optimización de tiempo y evaluación precisa. Todo esto subraya la importancia de continuar investigando y desarrollando soluciones innovadoras que faciliten el acceso a información resumida y mejoren la eficiencia en la gestión de grandes volúmenes de datos de texto. En resumen, el TFG ha demostrado ser una aportación valiosa tanto a nivel académico como en aplicaciones prácticas, abriendo así, camino para futuras investigaciones y desarrollos en este sector emergente.

Bibliografía

[1] Ahmed, S., Nielsen, I. E., Tripath, A., Siddiqu, S., Ramachandran, R. P., & Rasool, G. (2023). *Transformers in Time-series Analysis: A Tutorial*. arXiv:2205.01138v2

[2] Ahuir, V. (2023). HuggingFace. Obtenido de *dacsa*.

[3] Carmona, F. M., & Aldón, M. M. (2013). NeoInstrumenta. Obtenido de C*omparación de Sistemas de Extracción de Resúmenes*

[4] harvardnlp. (2018). Repositorio Github. Obtenido de *sent-summary*. *https://github.com/harvardnlp/sent-summary*

[5] Javi. (2023). Jacar.es. Obtenido de *Modelos pre-entrenados y transferencia de aprendizaje*

[6] jbagnato. (2022). Repositorio Github. Obtenido de *Gpt2 fine tuning. https://github.com/jbagnato/machine-learning/blob/master/gpt-2/Gpt2-fine-tuning.ipynb*

[7] Jin, H., Zhang, Y., Meng, D., Wang, J., & Tan, J. (2024). *A Comprehensive Survey on Process-Oriented Automatic Text Summarization with Exploration of LLM-Based Methods*. arXiv:2403.02901v1

[8] N.Moratanch, & S.Chitrakala. (2017). *A Survey on Extractive Text Summarization. ICCCSP-2017*.

[9] Penugonda, Shankar, & Gowri. Kaggle. (2021). Obtenido de *CNN-DailyMail News Text Summarization*

[10] Sharif, P. Kaggle. (2018). Obtenido de *BBC News Summary*

[11] Sutskever, I., Vinyals, O., & Le, Q. V. (2014). *Sequence to Sequence Learning with Neural Networks*. arXiv:1409.3215v3

[12] Trabado, M. Á. (2024). *La batalla de los transformes: BERT, GPT y Gemini*

[13] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . Polosukhin, I. (2023). *Attention Is All You Need*. arXiv:1706.03762v7

[14] Wikipedia. (2024). Obtenido de *GPT-2*

[15] Wikipedia. (2024). Obtenido de *Lenguaje Natural (LNP)*

[16] Wikipedia. (2024). Obtenido de *Mecanismos de Atención*

[17] Wikipedia. (2024). Obtenido de *ROUGE (métrica)*

[18] Alec Radford, Karthik Narasimham. (2018). *Improving Language Understanding by Generative Pre-Training*

Anexo

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**			X	
ODS 2. **Hambre cero.**			X	
ODS 3. **Salud y bienestar.**		X		
ODS 4. **Educación de calidad.**	X			
ODS 5. **Igualdad de género.**			X	
ODS 6. **Agua limpia y saneamiento.**			X	
ODS 7. **Energía asequible y no contaminante.**		X		
ODS 8. **Trabajo decente y crecimiento económico.**	X			
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades**.			X	
ODS 11. **Ciudades y comunidades sostenibles**.		X		
ODS 12. **Producción y consumo responsables.**		X		
ODS 13. **Acción por el clima.**			X	
ODS 14. **Vida submarina.**			X	
ODS 15. **Vida de ecosistemas terrestres.**			X	
ODS 16. **Paz, justicia e instituciones sólidas.**			X	
ODS 17. **Alianzas para lograr objetivos.**			X	

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

El TFG realizado se centra en el desarrollo y evaluación de modelos de resumen automático de textos. Este proyecto tiene, además, una relación alta con varios de los Objetivos de Desarrollo Sostenible, destacando en particular los ODS 4 (Educación de calidad), 8 (Trabajo decente y crecimiento económico) y 9 (Industria, innovación y estructura). Además, el trabajo tiene una relación media con los ODS 3 (Salud y bienestar), 7 (Energía asequible y no contaminante), 11 (Ciudades y comunidades sostenibles) y 12 (Producción y consumo responsables).

ODS 4: Educación de calidad

Este objetivo busca garantizar una educación inclusiva, equitativa y de calidad, promoviendo oportunidades de aprendizaje para todos. La relación del trabajo con este objetivo es clara en estos aspectos:

- *Acceso a la información y educación:* Los modelos de resumen automático de textos pueden facilitar el acceso a una información relevante y concisa, lo cual es fundamental en el contexto educativo. Estudiantes y profesores pueden beneficiarse de los resúmenes automáticos que les permitan asimilar rápidamente el contenido de artículos científicos, libros u otros recursos. Esto puede mejorar la eficiencia en la investigación y el aprendizaje.
- *Material educativo asequible:* Al utilizar modelos de resumen automáticos, es posible mejorar la calidad del material educativo al tener la posibilidad de generar resúmenes de alta calidad, ayudando así a regiones donde el acceso a libros y otros recursos es limitado. Esto contribuye a la democratización del conocimiento y a la reducción de las desigualdades educativas.
- *Herramientas de aprendizaje innovadoras:* El desarrollo de tecnologías basadas en Inteligencia Artificial para la educación fomenta nuevos métodos de enseñanza y aprendizaje.

ODS 8: Trabajo decente y crecimiento Económico

Este objetivo se centra en promover el crecimiento económico inclusivo y sostenible, el empleo pleno y productivo. Este trabajo contribuye a este objetivo de varias maneras:

- *Aumento de la productividad:* Los modelos de resúmenes automáticos pueden aumentar la productividad en distintas industrias al reducir el tiempo necesario para procesar y analizar grandes volúmenes de textos. Esto puede ayudar a liberar espacio para tareas más creativas y estratégicas.
- *Generación de nuevas oportunidades:* La implementación de la Inteligencia Artificial en diferentes sectores pueden crear nuevas oportunidades de empleo en áreas como desarrollo de software, análisis de datos, etc.
- *Mejora de la eficiencia:* Las empresas y organizaciones pueden utilizar resúmenes generados para mejorar la gestión de la información, lo que puede resultar en una mayor eficiencia y competitividad.

ODS 9: Industria, Innovación e Infraestructura

El objetivo 9 se centra en construir infraestructuras, promover la industrialización inclusiva y sostenible, y fomentar la innovación. El trabajo contribuye a este objetivo de varias maneras:

- *Innovación tecnológica:* La tecnología de generador de resúmenes de texto puede tener aplicaciones en diversas industrias, incluyendo medios de comunicación, investigación académica, y servicios empresariales, promoviendo la eficiencia y la productividad.
- *Desarrollo de infraestructura digital:* La creación de modelos avanzados de Inteligencia Artificial y su implementación en sistemas de resumen de textos contribuye a la infraestructura digital necesaria para una sociedad basada en el conocimiento.
- *Fomento del I+D:* Este trabajo no solo se basa en investigación de generación de texto, sino que también contribuye a este campo. La investigación y el desarrollo de tecnologías de IA son fundamentales para impulsar la innovación continua y el crecimiento económico sostenible.

ODS 3: Salud y Bienestar

El objetivo 3 busca garantizar una vida sana y promover el bienestar para todos en todas las edades. La relación del TFG con este objetivo es más indirecta, pero aún significativa:

- *Acceso a información sanitaria:* Los modelos de resumen automático pueden ser utilizados para procesar y resumir grandes volúmenes de literatura médica y científica, facilitando el acceso a información relevante para profesionales de la salud y pacientes.
- *Eficiencia en la investigación médica:* Los investigadores en el campo de la medicina pueden beneficiarse de resúmenes automáticos que les permitan revisar rápidamente la literatura existente, acelerando el proceso de descubrimiento y aplicación de nuevos tratamientos y medicamentos.

ODS 7: Energía Asequible y No Contaminante

El ODS 7 busca garantizar el acceso a una energía asequible, fiable, sostenible y moderna para todos. La relación del trabajo con este objetivo se manifiesta en:

- *Optimización del consumo de energía:* La implementación de modelos de IA en centros de datos y sistemas de procesamiento puede contribuir a optimizar el consumo de energía, haciéndolos más eficientes.
- *Investigación en energías renovables:* Los modelos de resumen automático pueden facilitar el acceso y análisis de la investigación en energías renovables.

ODS 11: Ciudades y Comunidades Sostenibles

El objetivo 11 se centra en hacer que las ciudades y los asentamientos humanos sean inclusivos, seguros y sostenibles. Este trabajo puede contribuir a este objetivo mediante:

- *Acceso a información para la planificación urbana:* Los resúmenes automáticos pueden ser utilizados por planificadores urbanos y responsables de políticas para acceder rápidamente a información relevante y actualizada.
- *Promoción de comunidades informadas:* El acceso a resúmenes automáticos de noticias y documentos puede empoderar a las comunidades, proporcionando información clara y concisa que fomente una participación ciudadana más activa y consciente en la planificación y desarrollo urbano.

ODS 12: Producción y Consumo Responsables

El objetivo 12 busca garantizar modalidades de consumo y producción sostenibles. La relación del trabajo con este objetivo incluye:

- *Difusión de prácticas sostenibles:* Los modelos de resumen automático pueden facilitar la difusión de prácticas sostenibles y estudios de caso en diversas industrias.
- *Reducción de residuos informativos:* Al proporcionar resúmenes claros y concisos, se puede reducir la cantidad de información redundante y poco relevante que los individuos y organizaciones deben procesar.

Apéndice

Apéndice A: Código capítulo 4

1. **Código 4.1.** Semillas y dispositivos de procesamiento.
   - Descripción: Se crea escoge una semilla para hacer posible la reproducción del entrenamiento y se elige el dispositivo con el que se va a trabajar, GPU o CPU. Se actualiza el tamaño del batch a 3 si es GPU o a 1 si es CPU.
   - Código:

```
np.random.seed(seed_val)

if torch.cuda.is_available():

    torch.manual_seed(seed_val)

    torch.cuda.manual_seed_all(seed_val)

    device = torch.device("cuda")

    batch_size = 3

else:

    device = torch.device("cpu")

    batch_size = 1
```

2. **Código 4.2.** Carga modelo y tokenizer.
   - Descripción: Aquí se cargan tanto el modelo como el tokenizer con los parámetros adecuados, como pueden ser los tokens especiales y la configuración preestablecida para el dropout.
   - Código:

```
tokenizer = AutoTokenizer.from_pretrained("openai-community/gpt2",
                                          bos_token='<|startoftext|>',
                                          eos_token='<|endoftext|>',
                                          pad_token='<|pad|>')

configuration = GPT2Config.from_pretrained('openai-community/gpt2', resid_pdrop=0.1,
embd_pdrop=0.1, attn_pdrop=0.1)

model = AutoModelForCausalLM.from_pretrained("openai-community/gpt2", config =
configuration)
```

3. **Código 4.3.** Token especial “Resumen”.
   - Descripción: Aquí estamos incluyendo un nuevo token especial para nuestro trabajo, en este caso llamado Resumen, que permitirá señalar al modelo qué parte debe resumir.
   - Código:

```
control_code = "Resumen"

special_tokens_dict = {

  "additional_special_tokens": [f"<|{control_code}|>"]

}

if f"<|{control_code}|>" not in tokenizer.all_special_tokens:

  num_added_toks = tokenizer.add_special_tokens(special_tokens_dict)

  model.resize_token_embeddings(len(tokenizer))
```

4. **Código 4.4.** Función de filtrado de datos.
   - Descripción: Esta es la función que permite cargar y filtrar los datos antes de ser usados para el entrenamiento por el model. Aquí estamos filtrando para recoger solamente aquellos datos que no superen la longitud máxima de tokens pasada como parámetro y además se escoge un número de muestras en caso de querer reducir aún más el tamaño.
   - Código:

```
def load_and_filter_data(file_path, tokenizer, max_length, num_samples):

  print('Loading files...')

  file = pd.read_csv(file_path)

  file['token_length'] = file['article'].apply(lambda x: len(tokenizer.encode(x)))

  filtered_file = file[file['token_length'] <= max_length]

  if len(filtered_file) > num_samples:

    filtered_file = filtered_file.sample(n=num_samples, random_state = 42)

  return filtered_file
```

5. **Código 4.5.** Clase GPT2Dataset.
   - Descripción: Aquí se muestra la implementación de la clase GPT2Dataset simplificada, en donde se crean las entradas, máscaras de atención y objetivos para cada dato del conjunto de datos para pasárselos posteriormente al modelo en el entrenamiento.
   - Código:

```
class GPT2Dataset(Dataset):

  def __init__(self, dataframe, control_code, tokenizer, max_length):
    (cargar los datos de entrada)

  def __len__(self):
    return len(self.dataframe)

  def __getitem__(self, idx):
    (recoger en variables llamadas ‘article’ y ‘summary’ los pares de textos y resumen del dataframe de entrada)

    input_text = f"<|startoftext|> {self.special_token} {article}   Summary: <|endoftext|>"
    target_text = f"{summary}" + '<|endoftext|>'
    input_encodings = self.tokenizer(input_text, truncation=True, padding='max_length', max_length=self.max_length, return_tensors='pt')
    target_encodings = self.tokenizer(target_text, truncation=True, padding='max_length', max_length=self.max_length, return_tensors='pt')

    input_ids = input_encodings["input_ids"].squeeze()
    attention_mask = input_encodings["attention_mask"].squeeze()
    labels = target_encodings["input_ids"].squeeze()

    return {
       'input_ids': input_ids,
       'attention_mask': attention_mask,
       'labels': labels
    }
```

6. **Código 4.6.** Rutas de datos y creación dataloaders.
   - Descripción: Aquí se muestra la creación de los Dataloaders mediante las rutas y funciones de filtrado de datos. Primeramente, se filtran los datos con la función creada anteriormente, luego se crean los Datasets correspondientes con la clase GPT2Dataset y finalmente se construyen los Dataloaders.
   - Código:

```
# Rutas de los ficheros CSV

train_csv_file = "./cnn_dailymail/train.csv"

val_csv_file = "./cnn_dailymail/validation.csv"
```

```
# Cargamos y filtramos datos

train_file = load_and_filter_data(train_csv_file, tokenizer, max_length=768, num_samples=50000)

val_file = load_and_filter_data(val_csv_file, tokenizer, max_length=768, num_samples=10000)

# Creamos datasets

train_dataset = GPT2Dataset(train_file, control_code, tokenizer, max_length=768)

val_dataset =   GPT2Dataset(val_file, control_code, tokenizer, max_length=768)

# Creamos dataloaders

train_dataloader = DataLoader(train_dataset, sampler = RandomSampler(train_dataset), batch_size = batch_size)

val_dataloader = DataLoader(val_dataset, sampler = RandomSampler(val_dataset), batch_size = batch_size)
```

7. **Código 4.7.** Ciclo de entrenamiento.
   - Descripción: Esta figura muestra el proceso de entrenamiento simplificado del modelo para la tarea de resúmenes de texto, donde se cargan las entradas, máscaras de atención y objetivos para pasárselos al modelo y devolver una salida (output). Posteriormente se recalculan las variables y la pérdida.
   - Código:

```
for step, batch in enumerate(train_dataloader):

    optimizer.zero_grad()

    input_ids = batch['input_ids'].to(device)

    attention_masks = batch['attention_mask'].to(device)

    target_ids = batch['labels'].to(device)

    output = model(input_ids,

                   attention_mask = attention_masks,

                   labels = target_ids,

                   token_type_ids=None)

    loss = output.loss

    loss.backward()

    optimizer.step()

    scheduler.step()

    total_train_loss += loss.item()
```

8. **Código 4.8.** Etapa de evaluación.
   - Descripción: Aquí se muestra el proceso de evaluación del conjunto de datos de evaluación. Es muy parecido al bucle de entrenamiento, pero el modelo no se pone en modo train(), si no que lo dejamos en modo eval(). Con esto, aseguramos que el modelo no se sigue entrenando con el conjunto de datos de evaluación.
   - Código:

```
t1 = time.time()

model.eval()

total_val_loss = 0.0

  for val_step, val_batch in enumerate(val_dataloader):

    with torch.no_grad():

      val_input_ids = val_batch['input_ids'].to(model.device)

      val_attention_mask = val_batch['attention_mask'].to(model.device)

      val_target_ids = val_batch['labels'].to(model.device)


      val_output = model(val_input_ids, attention_mask = val_attention_mask, labels =
val_target_ids)

      val_loss = val_output.loss

      total_val_loss += val_loss.item()
```

9. **Código 4.9.** Guardado del modelo.
   - Descripción: Aquí se muestra el proceso de guardado automático del modelo tras terminar el proceso de entrenamiento. Se proporciona un directorio y se guarda el modelo. Si no existe tal directorio se crea automáticamente para evitar errores.
   - Código:

```
output_dir = 'tu directorio aquí'

if not os.path.exists(output_dir):

  os.makedirs(output_dir)

model_to_save = model.module if hasattr(model, 'module') else model

model_to_save.save_pretrained(output_dir)

tokenizer.save_pretrained(output_dir)
```

Apéndice B: Código capítulo 6

1. **Código 6.1.** Generador de muestras de entrenamiento.
   - Descripción: Aquí se muestra el proceso de obtención de muestras durante el entrenamiento del modelo. Cada 100 pasos del bucle, se pone el modelo en modo evaluación para generar muestras con la función generate(), pasándole como parámetros los que vemos en el código. Posteriormente se imprimen por consola para comprobar los resultados.
   - Código:

```
if step % 100 == 0 and not step == 0:
        elapsed = format_time(time.time() - t0)
        model.eval()
        sample_outputs = model.generate(
                        input_ids = input_ids,
                        attention_mask = attention_masks,
                        do_sample=True,
                        top_k=50,
                        max_length = 1024,
                        top_p=0.95,
                        num_return_sequences=1,
                        num_beams=2,
                        no_repeat_ngram_size=2,
                        early_stopping=True,
                        repetition_penalty=2.0
        )
```

2. **Código 6.2**. Función de traducción.
   - Descripción: Aquí se muestra la implementación de la función que traduce el texto proporcionado. Se pasan los modelos y tokenizer de MarianMT, modelo que esta pre-entrenado para traducir textos. Se le pasa el texto por parámetro y lo devolvemos traducido según el idioma que hayamos escogido.
   - Código:

```
def translate(text, model_name, tokenizer_name):
    model = MarianMTModel.from_pretrained(model_name)
    tokenizer = MarianTokenizer.from_pretrained(tokenizer_name)
    inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True)
    translated = model.generate(**inputs)
    translated_text = tokenizer.batch_decode(translated, skip_special_tokens=True)[0]
    return translated_text
```

3. **Código 6.3.** Función de generación.
   - Descripción: Aquí muestra la implementación de la función que genera los resúmenes finales, es igual que cuando se generaban muestras que en el entrenamiento.
   - Código:

```
def generate_summary(text, model, tokenizer, device, 768, max_new_tokens=120):
    input_encodings = tokenizer(
       text, truncation=True, padding=True, max_length=max_input_length)
    input_ids = input_encodings["input_ids"].to(device)
    attention_mask = input_encodings["attention_mask"].to(device)
    sample_outputs = model.generate(
                                    input_ids,
                                    attention_mask=attention_mask,
                                    max_new_tokens=max_new_tokens,
                                    num_beams=2,
                                    no_repeat_ngram_size=2,
                                    early_stopping=True,
                                    repetition_penalty=2.0,
                                    length_penalty=1.0,
                                    pad_token_id=tokenizer.eos_token_id)
```

```
    input_text = tokenizer.decode(input_ids[0], skip_special_tokens=True)

    decoded_outputs = [tokenizer.decode(output, skip_special_tokens=True) for output in
sample_outputs]

    new_tokens = [decoded_output.replace(input_text, "") for decoded_output in decoded_outputs]

    return new_tokens[0]
```

4. **Código 6.4.** Carga de modelos.
   - Descripción: Aquí muestra la implementación del programa principal que carga los modelos necesarios (traducción y resumen). Hay que aclarar que para cada traducción hay que cargar un modelo y un tokenizer diferente, es decir, el modelo usado para traducir del inglés al español es distinto del que se usa para traducir del español al inglés.
   - Código:

```
# Modelo fine-tuned para resumir textos

    model_dir = 'tu directorio aquí'

    tokenizer = AutoTokenizer.from_pretrained(model_dir, bos_token='<|startoftext|>',
eos_token='<|endoftext|>', pad_token='<|pad|>', padding_side='left')

    model = AutoModelForCausalLM.from_pretrained(model_dir)

# Modelo pre-entrenado para traducir textos

    model_name_en_to_es = 'Helsinki-NLP/opus-mt-en-es'

    tokenizer_name_en_to_es = 'Helsinki-NLP/opus-mt-en-es'

    model_name_es_to_en = 'Helsinki-NLP/opus-mt-es-en'

    tokenizer_name_es_to_en = 'Helsinki-NLP/opus-mt-es-en'
```

5. **Código 6.5.** Generación de resúmenes.
   - Descripción: Aquí se muestra la implementación en el programa principal para generar e imprimir resúmenes. Simplemente se llaman a las funciones ‘translate’ y ‘generate’ implementadas anteriormente y se crean los resúmenes.
   - Código:

```
# Traducir el texto de español a inglés

    translated_text = translate(prompt, model_name_es_to_en, tokenizer_name_es_to_en)

# Generar el resumen en inglés

    summary_english = generate_summary(f"<|startoftext|> <|Resumen|> {translated_text}
<|endoftext|>", model, tokenizer, device)

# Traducir el resumen de inglés a español

    summary_spanish = translate(summary_english, model_name_en_to_es,
tokenizer_name_en_to_es)
```

Apéndice C: Código capítulo 7

1. **Código 7.1.** Implementación simplificada app.py
   - Descripción: Aquí muestra una implementación simplificada del script app.py utilizado en para la lógica de la aplicación de móvil. Como observamos, tiene muchas similitudes con los vistos anteriormente, ya que la traducción y la generación del resumen se hace igual, lo único que cambia es la forma en la se envían y se visualizan los resultados. En este caso, la app.py se conecta a la dirección proporcionada en __main__ y de ahí continúa dentro de la aplicación.
   - Código:

```
app = Flask(__name__)

(Se cargan todos los modelos, tanto los de traducción como el de resumen)

def translate(text, model, tokenizer):

    (Igual que hemos visto anteriormente)

def generate_summary(text, model, tokenizer, max_input_length=768, max_new_tokens=150):

    (Igual que hemos visto anteriormente)

@app.route('/summarize', methods=['POST'])

def summarize():

    data = request.form.get('text')

    translated_text = translate(data, model_es_to_en, tokenizer_es_to_en)

    summary_english = generate_summary(translated_text, model_summary, tokenizer_summary)

    summary_spanish = translate(summary_english, model_en_to_es, tokenizer_en_to_es)

    return summary_spanish

if __name__ == '__main__':

    app.run(host='0.0.0.0', port=5000)
```

2. **Código 7.2.** Implementación simplificada summary_privider.dart
   - Descripción: Aquí se muestra una implementación simplificada del script summary_provider para la conexión entre la aplicación móvil y el script app.py.
   - Código:

```
class SummaryProvider extends ChangeNotifier {

 String? summary;

 bool isLoading = false;

 Future<void> summarizeText(String text) async {

  _setLoading(true);

  try {

   final response = await http.post(

    Uri.parse('Aquí va la dirección creada anteriormente en app.py para conectarse'),

    body: {'text': text},);

   if (response.statusCode == 200) {

    _setSummary(response.body);}

  else {

    _setSummary('Error: ${response.statusCode}'); }

 } catch (e) {

  _setSummary('Error: $e'); }

 _setLoading(false); }
```

Apéndice D: Código capítulo 8

1. **Código 8.1.** Carga de modelos.
   - Descripción: Aquí se muestra la carga de los modelos usados para analizar las métricas respecto a nuestro modelo ajustado. Cada modelo tiene su propio tokenizer, y es importante respetarlo a la hora de generar un resumen.
   - Código:

```
# Cargar tu propio modelo

my_model_path = 'tu directorio aquí'

tokenizer = AutoTokenizer.from_pretrained(my_model_path, bos_token='', eos_token='', pad_token='', padding_side='left')

model_my_model = AutoModelForCausalLM.from_pretrained(my_model_path)

# Cargar GPT-2

tokenizer_gpt2 = AutoTokenizer.from_pretrained("gpt2")

model_gpt2 = GPT2LMHeadModel.from_pretrained("gpt2")

# Cargar Pegasus

tokenizer_pegasus = PegasusTokenizer.from_pretrained("google/pegasus-xsum")

model_pegasus = PegasusForConditionalGeneration.from_pretrained("google/pegasus-xsum")

# Cargar BERTSum

tokenizer_bertsum = BertTokenizer.from_pretrained("bert-base-uncased")

model_bertsum = BertLMHeadModel.from_pretrained("bert-base-uncased")
```

2. **Código 8.2.** Función ‘load_and_filter_data’ ajustado.
   - Descripción: Aquí se muestra la implementación de la función de carga y filtrado de datos ajustada al contexto de análisis de métricas. Se añade la función de cargar un diccionario de tokenizers, para que cada modelo use el suyo propio. Es importante ya que la calidad de los resúmenes generados puede variar.
   - Código:

```
def load_and_filter_data_ajustado(file_path, tokenizers, max_length):

    (se lee el archivo)

    for model_name, tokenizer in tokenizers.items():

        file[model_name + '_token_length'] = file['article'].apply(lambda x: len(tokenizer.encode(x)))

    filtered_file = file[file.apply(lambda row: all(row[model_name + '_token_length'] <= max_length for model_name in tokenizers.keys()), axis=1)

    return filtered_file
```