Universidad Zaragoza
1542

Trabajo Fin de Grado

Análisis de textos para la validación de citas y referencias

Text analysis for citation and reference validation

Autora

Paula Ezpeleta Castrillo

Directores

Lucía Pitarch Ballesteros

Carlos Bobed Lisbona

Escuela de Ingeniería y Arquitectura
2023-2024

Agradecimientos

A mis padres y mi familia, por pensar siempre en mi.

A Lucía y Carlos, por su compromiso, orientación y dedicación.

A Vero y al resto de mis amigos, por estar siempre cuando necesito despejarme.

A Natalia, por su cariño incondicional, por apoyarme y confiar siempre en mi.

A Horacio, por su compañía inigualable.

A mi abuelo, por repetirme siempre que estudiase mucho durante el curso, para que así pasase un buen verano.

Índice general

Resumen del proyecto

En un contexto donde la información, tanto científica como no científica, se puede encontrar de manera online y el volumen de estos datos es muy amplio, surge la necesidad de verificar la autenticidad y relevancia de las fuentes y citas utilizadas en los artículos académicos. Este proyecto se enfoca en desarrollar una herramienta para evaluar la adecuación entre citas y referencias en los artículos científicos, dada la importancia de garantizar la veracidad en este ámbito.

La herramienta propuesta permitirá a los usuarios determinar si las citas reflejan adecuadamente el contenido de las fuentes citadas y evaluar la polaridad de las mismas. Les permitirá subir un artículo en formato en formato PDF o buscarlo en arXiv y seleccionar una cita de este. Los resultados con respecto a la similitud y polaridad se mostrarán mediante gráficos en los que se reflejarán estos resultados para que sean más fáciles de entender para los usuarios, además de más atractivos.

El proyecto se ha llevado a cabo mediante la integración de estos servicios de PLN en una plataforma Web accesible y fácil de usar. Con cada fase del proyecto, se ha buscado mejorar y ajustar la funcionalidad de la herramienta en respuesta a las necesidades y comentarios de los usuarios.

1 Introducción

1.1. Objetivos del proyecto

El objetivo de este proyecto es desarrollar una herramienta que permita analizar la similitud y la polaridad entre citas y los artículos a los que estas hacen referencia. Este análisis se llevará a cabo mediante la implementación de varios modelos de Procesamiento del Lenguaje Natural (PLN) entrenados y preentrenados.

En primer lugar, se diseñará e implementará una interfaz de usuario intuitiva y fácil de usar que permita a los usuarios introducir citas y artículos. Esta interfaz integrará herramientas para cargar y gestionar textos de citas y artículos de manera eficiente, facilitando así el acceso y manipulación de los datos por parte de los usuarios.

El análisis de similitud entre citas y artículos se realizará utilizando distintos modelos preentrenados y realizando el entrenamiento de otros, que permiten calcular la similitud entre las citas y los artículos citados.

Además, se determinará la polaridad de los textos mediante un modelo de análisis de sentimiento, que permitirá identificar la polaridad (positiva, negativa, neutra) de las citas. Este análisis ayudará a comprender mejor el tono y la intención detrás de los textos.

Para mejorar la precisión en el análisis de similitud y polaridad, se entrenarán modelos específicos con conjuntos de datos relevantes. El conjunto de datos de entrenamiento para el modelo de simlitud consta de un total de 30,9 GB de artículos provenientes de arXiv, en formato JSON, proporcionado por Zenodo y obtenidos a través del artículo de Saier et al. (2023) [1]. Este proceso de entrenamiento y ajuste es crucial para asegurar la efectividad y exactitud del sistema. Por otra parte, el conjunto de datos de entrenamiento para el modelo de la polaridad, utiliza el conjunto de datos SCICite, propuesto por Arman et al. (2019) [2].

La visualización de los resultados será una parte importante del proyecto, ya que es de gran importancia que los usuarios los comprendan fácilmente. Se desarrollarán herramientas de visualización que permitan mostrar de manera clara y comprensible los resultados del análisis de similitud y polaridad. Gráficos interactivos facilitarán la interpretación de los datos por parte de los usuarios, proporcionando una experiencia visual rica y accesible.

Es importante destacar que, aunque el desarrollo inicial del proyecto se centra en artículos de arXiv, en el futuro se podría ampliar para utilizar artículos de diferentes fuentes, como puede ser Google Scholar, aumentando así su versatilidad y aplicabilidad.

Finalmente, se creará una documentación completa y detallada del sistema, incluyendo instrucciones para su uso, asegurando así una adopción efectiva y satisfactoria de la herramienta.

1.2. Gestión del proyecto

La gestión del proyecto se ha realizado a través de reuniones periódicas con los directores. Durante el primer mes, estas reuniones se llevaban a cabo cada dos semanas, permitiendo así el tiempo necesario para alcanzar los objetivos iniciales, ya que los comienzos de los proyectos suelen ser costosos y requieren una fase de adaptación. Después de esta fase, las reuniones pasaron a ser semanales, estableciendo distintos objetivos que debían cumplirse antes de la siguiente reunión programada.

En cuanto al código, se ha utilizado Git para llevar un control de versiones eficiente. Git ha facilitado el seguimiento de los cambios y la posibilidad de revertir a versiones anteriores cuando ha sido necesario. Además, se ha trabajado con una sola rama, manteniendo el código organizado y estable durante todo el desarrollo.

Para la redacción de la memoria del proyecto, se ha utilizado Overleaf. Esta herramienta ha facilitado la gestión y edición del documento, resultando especialmente útil para la organización de la bibliografía y las citas.

En la Figura 1.1 se muestra un diagrama de Gantt que ilustra cómo se ha organizado el tiempo a lo largo del proyecto, proporcionando una visión clara de las diferentes fases y tareas realizadas.

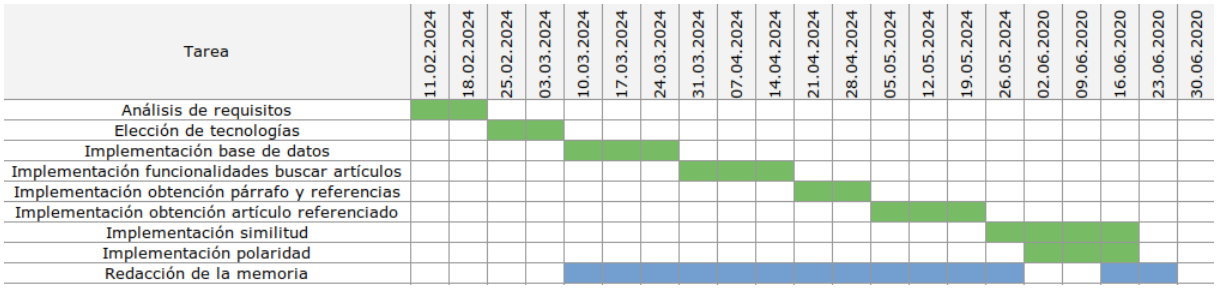

Figura 1.1: Diagrama de Gantt del proyecto

Además, en la Figura 1.2 se puede observar la dedicación en horas a diferentes tareas del proyecto.

Tarea	Horas dedicadas
Estudio del estado del arte y herramientas	30
Implementación interfaz	30
Implementación funcionalidad	250
Reuniones con directores	25
Redacción de la memoria	60
	Total: 395

Figura 1.2: Dedicación en horas al proyecto

1.3. Organización de la memoria

En este primer capítulo se ha realizado una breve introducción del proyecto además de la metodología seguida para llevarlo a cabo. En el segundo capítulo se presenta el contexto tecnológico y algunas de las herramientas utilizadas. En el tercer capítulo se explica el diseño de las dos partes del proyecto; *frontend* y *backend*, además de ilustrar cómo se hace un uso normal de la aplicación por parte de un usuario. En el cuarto capítulo se procede a describir cómo ha sido la implementación de estas dos partes, y además toda parte correspondiente a los distintos entrenamientos realizados en el proyecto. En el quinto capítulo se hacen algunos comentarios respecto a la evaluación de los resultados. En el sexto y último capítulo se exponen las conclusiones del proyecto y el trabajo futuro a realizar.

2 Contexto tecnológico

2.1. Estado del arte

Tal y como se expone en Zhang et al. (2023) [3], las citas en trabajos académicos tienen una función crucial al reconocer y otorgar crédito a las fuentes originales de conocimiento mencionadas. Además de este reconocimiento, las citas establecen relaciones entre los distintos artículos científicos, lo que facilita la difusión del conocimiento especializado y permite a los autores y lectores realizar referencias precisas en diversos contextos. Esta interconexión entre documentos crea una red literaria que puede ser aprovechada para una comprensión más profunda del panorama de la investigación en un campo específico.

Los Modelos de Lenguaje de Gran Escala (LLMs) están surgiendo como herramientas prometedoras para mejorar el análisis de citas. Estos modelos, como ChatGPT/GPT-4, proporcionan características de alta calidad para comprender el contexto de las citas. Por un lado, los LLMs pueden capturar información detallada sobre las citas a partir del texto que las rodea, lo que permite un análisis más exhaustivo y preciso. Por otro lado, las citas también pueden enriquecer el conjunto de datos de entrenamiento de los LLMs al proporcionar conexiones entre documentos que reflejan el conocimiento humano y la interrelación entre los artículos académicos. Sin embargo, es crucial abordar el desafío de la "halucinación" al emplear estos modelos en el análisis bibliográfico. Aunque los LLMs son poderosos, su capacidad para generar información precisa no está exenta de riesgos. Los LLMs pueden generar predicciones erróneas o basadas en correlaciones superficiales en lugar de una comprensión profunda del contenido. Por ejemplo, podrían asociar automáticamente la palabra "great" con polaridad positiva, sin considerar el contexto completo de la frase. Por lo tanto, mientras se explora el potencial de los LLMs en el análisis de citas, también se debe abordar de manera crítica sus limitaciones y la necesidad de utilizarlos con precaución y discernimiento. Esta tarea de investigación y reflexión continua es fundamental para aprovechar al máximo el potencial de los LLMs en el análisis bibliográfico y garantizar resultados precisos y significativos.

En Te et al. (2022) [4], se discute la clasificación de contextos de citas en críticos y no críticos. Se introduce el concepto de "citas críticas", que son aquellas que señalan debilidades, comparaciones o dudas sobre el trabajo citado. Sobre esto, Bordignon (2022) [5] expone que el estudio de las citas críticas es un fenómeno vital para el desarrollo científico. Se sugiere que clasificar los contextos de citas en categorías críticas y no críticas podría ser esencial para procesos posteriores, como identificar afirmaciones científicas u observar artículos controvertidos.

En el panorama actual, Nicholson et al. [6], explican cómo se ha llevado a cabo la herramienta Scite. Scite utiliza avances recientes en Inteligencia Artificial para producir "citas inteligentes". Estas citas revelan cómo se ha citado un artículo científico al proporcionar el contexto de la cita y un sistema de clasificación que describe si proporciona evidencia de apoyo o contrastante para la afirmación citada, o si simplemente la menciona.

En cuanto a lo que la similitud respecta, en La Quatra et al. (2021) [7], se destaca la importancia de explorar la similitud entre los distintos apartados de un artículo citado y el contexto local donde se encuentra la cita en el artículo que la menciona. Se menciona que, en muchos casos, el contenido del título y el resumen del artículo citado es altamente similar al contexto de la cita en el artículo que la menciona.

En último lugar, Yu y Hua (2023) [8], destacan la importancia de estudiar y analizar las funciones y los sentimientos de las citas científicas para evaluar de manera más efectiva un artículo y descubrir su información subyacente. Además, la clasificación del sentimiento de la cita científica puede ayudar a los investigadores a comprender mejor las actitudes y perspectivas de los demás hacia campos de investigación específicos, lo que ayuda a determinar la calidad y confiabilidad de la investigación existente, así como a evaluar las tendencias de investigación en el campo. La polaridad de la cita se refiere a la actitud emocional del autor hacia el artículo citado, como aprobación, oposición o neutralidad. El análisis del sentimiento de la cita revela esta actitud emocional a través de varios métodos, como SVM, Naive Bayes, TextCNN, BERT, etc.

Actualmente no existe una herramienta abierta que integre estas técnicas de manera unificada. Herramientas como Scite han avanzado en la producción de citas inteligentes, pero su acceso es limitado debido a su carácter comercial. La creación de una herramienta de este tipo no solo ayudaría a reconocer y evaluar la calidad y el impacto de los artículos de manera más efectiva, sino que también fomentaría una mayor colaboración y transparencia en el intercambio de conocimientos.

2.2. Contexto tecnológico y herramientas utilizadas

El Procesamiento del Lenguaje Natural (PLN) es una disciplina dentro del campo de la Inteligencia Artificial que se centra en la interacción entre los ordenadores y el lenguaje humano, con el objetivo de permitir que los ordenadores comprendan, interpreten y generen lenguaje humano de manera efectiva. En este contexto, el Aprendizaje Profundo y los *word embeddings*[1], son herramientas utilizadas para abordar desafíos específicos en el Procesamiento del Lenguaje Natural.

Como se trata en Kulkani y Shivanda (2019) [9], los *word embeddings* son fundamentales para capturar las relaciones semánticas y contextuales entre las palabras en el Procesamiento del Lenguaje Natural. Estas representaciones vectoriales de palabras se utilizan en una variedad de tareas de PLN, como la traducción automática, la recuperación de información y la generación de texto. Al representar las palabras como vectores de número

[1]Se va a utilizar el término inglés *word embeddings* en vez de sus traducciones comunes al español: *vector de incrustaciones de palabras*, *vector de representación*

reales nos podemos aproximar lo suficiente a capturar su significado como para mejorar la precisión y efectividad de los modelos de PLN.

El aprendizaje de representaciones continuas de palabras ha sido un tema central en el Procesamiento del Lenguaje Natural (PLN) durante décadas. Tradicionalmente, estas representaciones se derivan de grandes corpus no etiquetados utilizando estadísticas de co-ocurrencia, que miden la frecuencia con la que dos o más palabras ocurren juntas en un texto o corpus. Métodos como el propuesto por Collobert et al. (2008) [10], basado en redes neuronales *feed-forward*, han sido utilizados para entrenar *embeddings* de palabras. Sin embargo, la mayoría de estas técnicas asignan un vector único a cada palabra del vocabulario, sin considerar la estructura interna de las palabras ni la variabilidad semántica de estas según el contexto en el que se encuentra, suponiendo un problema el hecho de que no puedan codificar la polisemia. Esta limitación es especialmente notable en lenguajes morfológicamente ricos, como el turco, el finlandés, el francés y el español, donde las palabras pueden tener muchas formas diferentes debido a la conjugación y declinación, dificultando el aprendizaje de representaciones efectivas.

En los últimos años, han surgido modelos que lidian con este problema principalmente utilizando mecanismos de atención, como BERT [11], que usa una arquitectura basada en transformers y esto le permite considerar el contexto tanto antes como después de una palabra en una oración, por tanto, las palabras pueden tener diferentes significados según el contexto en el que aparecen.

Por último, el Aprendizaje Profundo (BERT, por ejemplo, entraría en esta categoría de arquitecturas), con sus capacidades para modelar la estructura y complejidad del lenguaje, se utiliza en una amplia gama de aplicaciones de PLN. Las redes neuronales convolucionales (CNN), por ejemplo, se utilizan para tareas como la clasificación de documentos, el etiquetado de partes del discurso y el análisis de sentimientos, lo que permite a estos modelos aprender características complejas del lenguaje y realizar predicciones precisas sobre el texto de entrada.

2.3. Modelos estáticos

2.3.1. Word2Vec

Mikolov et al. (2013) [12], explica el enfoque de Word2Vec, desarrollado por Google, el cual se ha convertido en un marco de referencia en el aprendizaje de representaciones de palabras mediante el uso de redes neuronales poco profundas. Este método se basa en la predicción de palabras cercanas en un corpus de texto, utilizando dos arquitecturas principales, esquematizadas en la Figura 2.1:

- **Skip-Gram**: El modelo skip-gram (Mikolov et al. (2013) [12]) se predice la probabilidad de una palabra dado su contexto de palabras circundantes. Cada oración en el corpus de datos genera una palabra objetivo y su contexto, que son las palabras cercanas.

- **CBOW (Continuous Bag of Words)**: En este método, el modelo predice la palabra objetivo basada en un contexto de palabras vecinas. Se utiliza la información del contexto para predecir la palabra objetivo, lo que permite capturar las relaciones semánticas y sintácticas entre las palabras.

En ambas arquitecturas, hay 3 parámetros importantes:

Ventana (*Window*) : este concepto hace referencia al número de palabras que definen el contexto.

Época (*Epoch*): se define como el número de iteraciones que se van a realizar sobre el corpus de entrenamiento, para ir ajustando los parámetros del modelo e ir así mejorando los *embeddings*.

Dimensión (*Dimension*): es la dimensión de los *embeddings*.

En los tres parámetros, cuanto mayor es el valor, mayor coste computacional y mayor precisión. Una detallada descripción de los distintos elementos para word2vec, se puede encontrar en Rong (2016) [13].

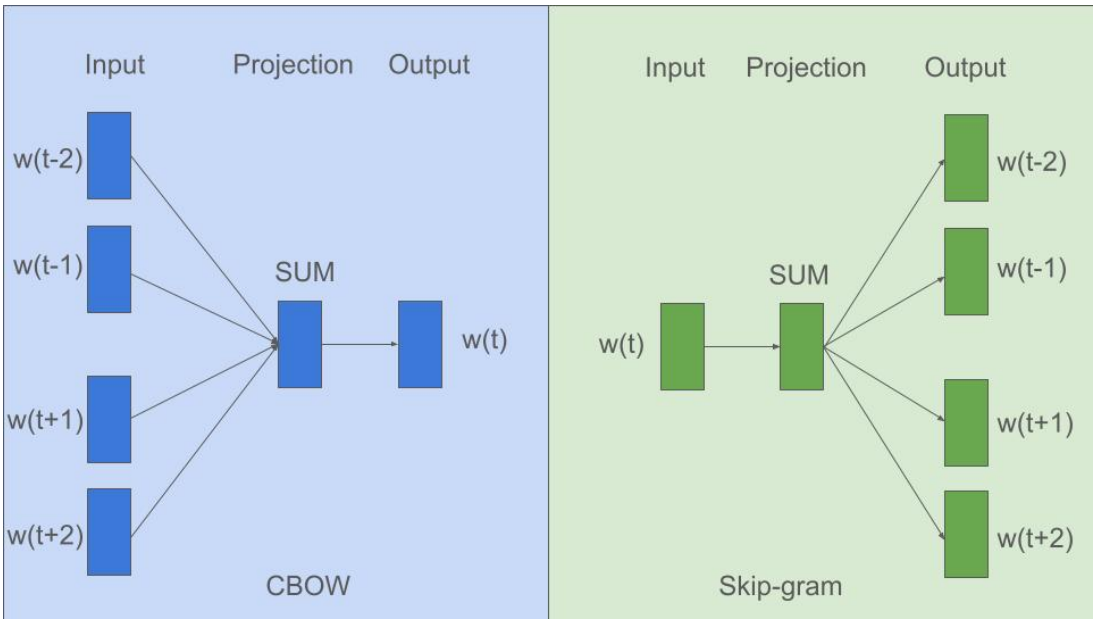

Figura 2.1: Arquitecturas de word2vec: CBOW y Skip-gram

Ambos enfoques tienen como objetivo principal la creación de *word embeddings* que capturen el contexto en el que aparecen en el corpus. Se ha demostrado que Word2Vec supera a otras metodologías en estas áreas, como se puede apreciar en los artículos de Mikolov et al. (2013) [12] y Allen y Hospedales (2019) [14].

Esta afirmación también está respaldada por el artículo Fonseca et al. (2015) [15]. En él, evaluaron el desempeño de tres modelos distintos de espacios vectoriales empleados en el etiquetado POS con un etiquetador neuronal. Utilizaron Word2Vec Skip-Gram, HAL y el método neuronal de Collobert et al. (2011) [16]; Skip-Gram demostró ser el más efectivo en todas las pruebas.

Una limitación de este modelo es que cada palabra en el vocabulario está representada por un solo vector, lo que significa que la representación de una palabra en el espacio vectorial es independiente de su contexto. Por ejemplo, en las frases "he contratado un modelo para la sesión de fotos" y "he entrenado un nuevo modelo para calcular la polaridad de la cita", en ambos casos, "modelo" tendrá el mismo *embedding*, aún si el significado y el referente en ambos casos es muy distinto.

Otra limitación es que en términos de OOV (*out of vocabulary*), generalmente no puede producir representaciones vectoriales para palabras que no están presentes en su vocabulario durante el entrenamiento. Esto es algo que Fasttext sí intenta mitigar.

Modelo preentrenado Word2Vec de Google

Entrenar estos modelos requiere grandes recursos y capacidades computacionales. Afortunadamenete, Google ha hecho público un modelo preentrenado, que ha sido entrenado con más de 100 mil millones de palabras. Los datos utilizados por Google para el entrenamiento de este modelo, se obtuvieron utilizando un enfoque basado en datos, como se describe en Mikolov et al. (2013) [12]. Según se expone en este artículo, se han realizado varias mejoras al modelo original Skip-gram. Una de ellas es el submuestreo de palabras frecuentes durante el entrenamiento, lo cual resulta en una aceleración significativa (entre 2 y 10 veces más rápido) y mejora la precisión de las representaciones de palabras menos frecuentes.

2.3.2. Fasttext

Tal y como se explica en Bojanowski et. al (2017) [17], FastText propone una extensión del modelo Skip-gram continuo (modelo que intenta predecir el contexto de una palabra dada; o lo que es lo mismo, las palabras que la rodean, en lugar de la palabra en sí misma) que incorpora información a nivel de sub-palabra. En lugar de representar cada palabra con un único vector, FastText descompone las palabras en n-gramas de caracteres (secuencia continua de n caracteres dentro de una palabra) y suma los vectores de estos n-gramas para obtener la representación de la palabra completa. Esta técnica permite capturar la morfología interna de las palabras, mejorando la calidad de las representaciones, sobre todo para palabras extrañas o no vistas durante el entrenamiento.

Al utilizar información a nivel de n-gramas, FastText es capaz de producir representaciones precisas y robustas para lenguajes complejos, aprovechando las reglas morfológicas subyacentes.

2.4. Modelos dinámicos

En los modelos estáticos nombrados, existe la limitación de que a cada palabra se le asigna únicamente un vector. En cambio, en modelos como BERT, se usan *contextual*

embeddings, que capturan el significado de una palabra en su contexto, lo que le permite comprender mejor el significado polisémico y las relaciones entre palabras en un texto.

2.4.1. Bidirectional Encoder Representations from Transformers (BERT)

Como se expone Devlin et al. (2019) [11] BERT es un modelo avanzado para el Procesamiento del Lenguaje Natural, también desarrollado por Google. Su arquitectura y funcionamiento se basan en dos fases principales: *pre-entrenamiento* y *fine-tuning*.

En la fase de pre-entrenamiento, BERT se entrena con datos no etiquetados. Durante esta fase, BERT se entrena en dos tareas principales: el Modelo de Lenguaje Enmascarado (MLM, por sus siglas en inglés Masked Language Model) y la Predicción de la Siguiente Oración (NSP, por sus siglas en inglés Next Sentence Prediction). En la tarea MLM, BERT aprende a predecir palabras enmascaradas en una oración utilizando el contexto circundante. Mientras tanto, en NSP, BERT se entrena para determinar si una oración sigue lógicamente a otra en un corpus de texto.

Una vez pre-entrenado, BERT se adapta a tareas específicas mediante la fase de *fine-tuning*. Aquí, el modelo pre-entrenado se inicializa con los parámetros aprendidos y luego se ajusta utilizando datos etiquetados específicos de la tarea, como clasificación de texto, preguntas y respuestas, o análisis de sentimientos. Cada tarea específica tiene su propio modelo ajustado, aunque todos se basan en los mismos parámetros.

Para manejar una variedad de tareas específicas, BERT utiliza representaciones de entrada capaces de representar tanto una sola oración como un par de oraciones en una secuencia de tokens. Se aprenden *embeddings* de WordPiece con un vocabulario de 30,000 tokens. El primer token de cada secuencia es siempre un token especial de clasificación ([CLS]), y el estado oculto final correspondiente a este token se usa como la representación agregada de la secuencia para tareas de clasificación. Las parejas de oraciones se empaquetan juntas en una sola secuencia, separadas por un token especial ([SEP]) y se añade un *embedding* aprendido a cada token para indicar a qué oración pertenece.

2.4.2. Sentence transformers + BERT

Según se explica en el artículo de Reimers y Gurevych (2019) [18], Sentence-BERT (SBERT) es una adaptación del modelo BERT que utiliza redes siamesas y tripletas para obtener *embeddings* de oraciones que son semánticamente significativas. Esto permite que BERT se use para nuevas tareas como la comparación de similitud semántica a gran escala, la agrupación de oraciones y la búsqueda semántica, algo que BERT no podía hacer de manera eficiente.

SBERT mapea cada oración a un espacio vectorial, donde las oraciones semánticamente similares están cercanas entre sí. En lugar de usar directamente las salidas de BERT, SBERT utiliza una arquitectura de red siamesa que permite derivar vectores de tamaño fijo para las oraciones de entrada. Con medidas de similitud como la del coseno o las

distancias Manhattan o Euclídea, se pueden encontrar oraciones semánticamente similares de manera muy eficiente en hardware actual.

2.4.3. DictSentiBert

DictSentiBERT es un modelo desarrollado para clasificar el sentimiento en citas científicas, utilizando una arquitectura BERT modificada y mejorada con un diccionario de sentimientos, lo que permite captar mejor las emociones y opiniones en el texto académico. Este diccionario contiene puntuaciones de sentimiento para una amplia gama de palabras y frases, y su integración permite que el modelo priorice aquellos términos con puntuaciones de sentimiento significativas. Esto es crucial para captar las sutilezas y matices del lenguaje académico y científico, donde el sentimiento puede ser expresado de maneras más complejas. Se explica detalladamente en Yu y Hua (2023) [8].

3 Diseño

El sistema desarrollado consta de dos partes principales: **frontend** y **backend**.

La arquitectura del sistema ha sido diseñada para ser modular y permitir de esta manera mayor escalabilidad en cuanto a la adición de módulos y funciones. Esto también permite que sea fácil de mantener. Este sistema tiene una arquitectura de dos capas, donde cada una de ellas, es responsable de una parte específica del procesamiento de los diferentes datos.

- En primer lugar, se encuentra la capa de *frontend*, responsable de la interacción con el usuario en sí.
- En segundo lugar, se encuentra la capa de *backend*. En esta capa se produce la mayor parte del procesamiento de los datos, así como la lógica de la aplicación y las solicitudes enviadas por el *frontend* y también se encuentra la base de datos, que gestiona todas aquellas operaciones relacionadas con el acceso a los datos persistentes.

Esto se puede observar mediante un diagrama de despliegue en la Figura 3.1. Por otra parte, en la Figura 6.1 del Anexo 3, se muestra un diagrama de secuencia de alto nivel, en el que se describe cómo sería el flujo de uso normal y completo de un usuario en la web.

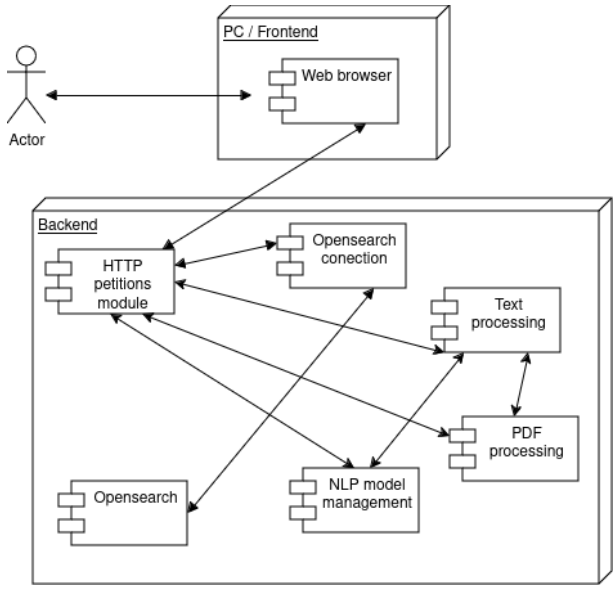

Figura 3.1: Diagrama de despliegue de la aplicación

En la Sección 4 se describe de manera intuitiva el flujo de control de un uso habitual.

3.1. Diseño del *frontend*

Para el desarrollo de la interfaz de usuario de la aplicación se definieron los requisitos funcionales y no funcionales. Se describen a continuación:

Requisitos funcionales
1. El usuario debe poder introducir el ID de un artículo de arXiv y buscarlo, además de permitirle subirlo desde el ordenador. 2. El sistema debe mostrar el artículo en un visualizador de PDFs. 3. El usuario debe poder seleccionar una cita o cualquier texto en el PDF visualizado. 4. El texto seleccionado debe aparecer en un recuadro de texto con un botón que al presionarlo, se analice la cita. 5. Al presionar el botón, el sistema debe mostrar el párrafo completo al que pertenece el texto seleccionado. 6. El sistema debe mostrar todas las referencias del párrafo de la selección en forma de lista. 7. El usuario debe poder hacer clic en una referencia de esta lista que sea un artículo de arXiv. 8. Al hacer clic en la referencia, el sistema debe mostrar el artículo referenciado. 9. El usuario debe poder calcular la similitud entre el *abstract* del artículo referenciado y la cita seleccionada. También debe poder calcular la similitud entre los párrafos del artículo referenciado y la cita seleccionada. 10. El usuario debe poder obtener la probabilidad de que la cita seleccionada pertenezca a las clases positiva, negativa o neutra. 11. Los resultados de estos cálculos deben mostrarse al usuario en forma de gráficos interactivos.

Requisitos no funcionales
1. La aplicación debe ser compatible con los navegadores más utilizados, incluyendo Chrome, Firefox, Safari y Edge, en sus dos últimas versiones y terner un código con buenas prácticas de desarrollo

3.2. Diseño del *backend*

El *backend* se divide en dos partes fundamentales, la base de datos y el servidor.

3.2.1. Base de datos

Como se ha nombrado previamente, los datos elegidos para la posterior implementación de la base de datos provienen del artículo Saier et al. (2023) [1]. En este artículo se puede

encontrar referencias a dos conjuntos de datos, proporcionados por Zenodo[1]. Por un lado, está el conjunto completo de datos [19]. Dado su enorme tamaño, se decidió trabajar con el subconjunto distribuido [20]. Este consta de 4.8 GB compactados en formato `xz`, que una vez descomprimido ocupaban 33.6 GB. Está compuesto por el contenido y la información extraída de 165000 artículos provenientes de arXiv[2].

Estos artículos han sido preprocesados específicamente para facilitar tareas de PLN. Cada artículo está en formato JSON, lo que permite manipular y extraer información de los artículos con facilidad. La descripción detallada del formato de la información correspondiene a los artículos se puede consultar en el Anexo 1.

Debido a la decisión de utilizar un conjunto de datos ya procesados, el diseño de la base de datos para el proyecto fue guiado por el propio conjunto. Los datos determinaron el diseño de la base de datos, aprovechando las estructuras y campos definidos en el artículo original.

Analizada la estructura de los ficheros JSONs, se decidió que los campos utilizados serían los siguientes:

- `paper_id`: Identificador único del artículo.
- `discipline`: Disciplina a la que pertenece el artículo.
- `title`: Título del artículo.
- `category`: Categoría del artículo.
- `authors`: Autores del artículo.
- `json_data`: JSON completo del artículo, serializado.

Cabe destacar que no se diseñó un diagrama entidad-relación ya que únicamente se iba a crear una tabla. Esto se hizo prioritariamente para asegurar que la búsqueda fuese rápida.

3.2.2. Diseño del servidor

Se tomó la decisión de fragmentar el código utilizado en el servidor en varios módulos distintos, cada uno dedicado a una función específica. Estos módulos y las relaciones entre ellos se pueden ver en la Figura 3.1.

Módulo de gestión de Opensearch: se encarga de gestionar el acceso a la base de datos, asegurando una comunicación eficiente y confiable para la búsqueda y análisis de datos.

Módulo de procesamiento de PDF: se encarga de la manipulación precisa y optimizada de estos PDFs en el flujo de datos de la aplicación.

Módulo de procesamiento de texto: en este módulo se llevan a cabo las operaciones necesarias para analizar y extraer información de manera efectiva.

[1] `https://zenodo.org/`
[2] `https://arxiv.org/`

Módulo del análisis de contenido: módulo encargado de la gestión de modelos preentrenados y entrenados, asegurando la implementación coherente y eficiente de estos recursos en el sistema. En este módulo, se llevan a cabo todas las operaciones necesarias para calcular la similitud y la polaridad. Es necesario recalcar que, para calcular la similitud entre dos frases, se obtienen las representaciones vectoriales promediando los vectores de palabras correspondientes a cada frase. Luego, se determina el coseno del ángulo entre los vectores normalizados, lo que proporciona una medida de la similitud semántica entre las frases. También proporciona las funcionalidades necesarias para obtener la polaridad, utilizando un modelo entrenado específicamente para esta tarea.

Módulo de peticiones y respuestas HTTP: dedicado a la gestión de solicitudes y respuestas HTTP, abarcando tanto las peticiones como las respuestas, garantizando un manejo completo y estructurado de las interacciones entre el servidor y los clientes.

Esta división del código no solo proporciona organización y facilidad a la hora del mantenimiento del código, sino que también facilita la escalabilidad del *software*.

4 Implementación

4.1. Análisis y elección de tecnologías

La elección de tecnologías es una decisión muy importante a tomar antes de comenzar a desarrollar un proyecto informático. A continuación, se expondrá dividida en 3 partes la elección de tecnologías: para la parte de *frontend*, para el *backend* y para la base de datos.

En primer lugar, para el desarrollo del *frontend* de la aplicación, se decidió utilizar JavaScript con React-Bootstrap[1], debido, principalmente, a su amplia gama de componentes predefinidos. La utilización de esta biblioteca de componentes permitió construir una interfaz consistente y atractiva, además de intuitiva.

Para el desarrollo del *backend* de la aplicación se optó por utilizar Python con Flask[2]. Una de las razones para esta elección fue la oportunidad de adquirir nuevas habilidades, dado que Python no se había aprendido durante la carrera. Flask es un framework ligero y flexible que simplifica el desarrollo de aplicaciones web en Python, ofreciendo una estructura mínima pero potente para construir servicios Web. Su simplicidad y modularidad lo hacen ideal para proyectos de diversos tamaños. Otra de las razones para el uso de python fue la existencia de múltiples librerias para el dominio del problema de este proyecto, ampliamente utilizadas y testeadas, como pueden ser spacy, re, gensim, PyPDF2, etc.

En último lugar, la elección de las tecnologías para la implementación de la base de datos resultó ser un proceso complejo, caracterizado por la consideración de múltiples opciones y la posterior necesidad de reevaluación. Inicialmente, se consideraron diversas alternativas para satisfacer las necesidades específicas del proyecto, como fue por ejemplo la utilización de una base de datos NoSQL como MongoDB[3]. Se barajó esta opción en primer lugar ya que para la implementación del proyecto era necesario almacenar en la base de dato documentos JSON (JavaScript Object Notation), y MongoDB está basada en un modelo de datos flexible en forma de documentos de este tipo, por lo que era la opción perfecta ya que está orientada al almacenamiento de documentos. Además, existe PyMongo, la biblioteca oficial de Python para utilizar MongoDB. Es una API muy completa que ofrece numerosas operaciones para gestionar la BD. Sin embargo, esta opción tuvo que ser descartada por el hecho de que MongoDB únicamente ofrece un espacio de

[1] `https://react-bootstrap.netlify.app/`
[2] `https://flask.palletsprojects.com/en/3.0.x/`
[3] `https://www.mongodb.com/`

512MB de manera gratuita utilizando su servicio de base de datos en la nube. Se realizaron varias pruebas para comprobar cuántos documentos JSON podrían ser insertados en este espacio, obteniendo un resultado de 2600 aproximadamente. Esto no era algo factible de utilizar, ya que había más de 30GB de documentos JSON que era necesario almacenar. Por tanto, fue necesario buscar una solución alternativa. Tras investigar acerca de las distintas alternativas y lo que su uso supondría para el desarrollo del proyecto, se dio con OpenSearch[4]. OpenSearch es un motor de búsqueda y análisis, de código abierto, diseñado para trabajar de manera eficiente con grandes volúmenes de datos. Presenta gran capacidad para indexar y buscar datos en tiempo real, lo cual, era necesario en este proyecto ya que el volumen de datos utilizado iba a ser de tamaño considerable. Además, como ya se ha dicho previamente, la escalabilidad era algo a considerar en este proyecto, y OpenSearch está diseñado para escalar tanto vertical como horizontalmente.

4.2. Implementación del *frontend*

Mediante las herramientas nombradas en la Sección 4.1, se implementó el código del *frontend*. Este código debía permitir a los usuarios utilizar el servicio web tal y como se explica en la Sección 3.1.

En la Figura 4.1 se muestra la interfaz.

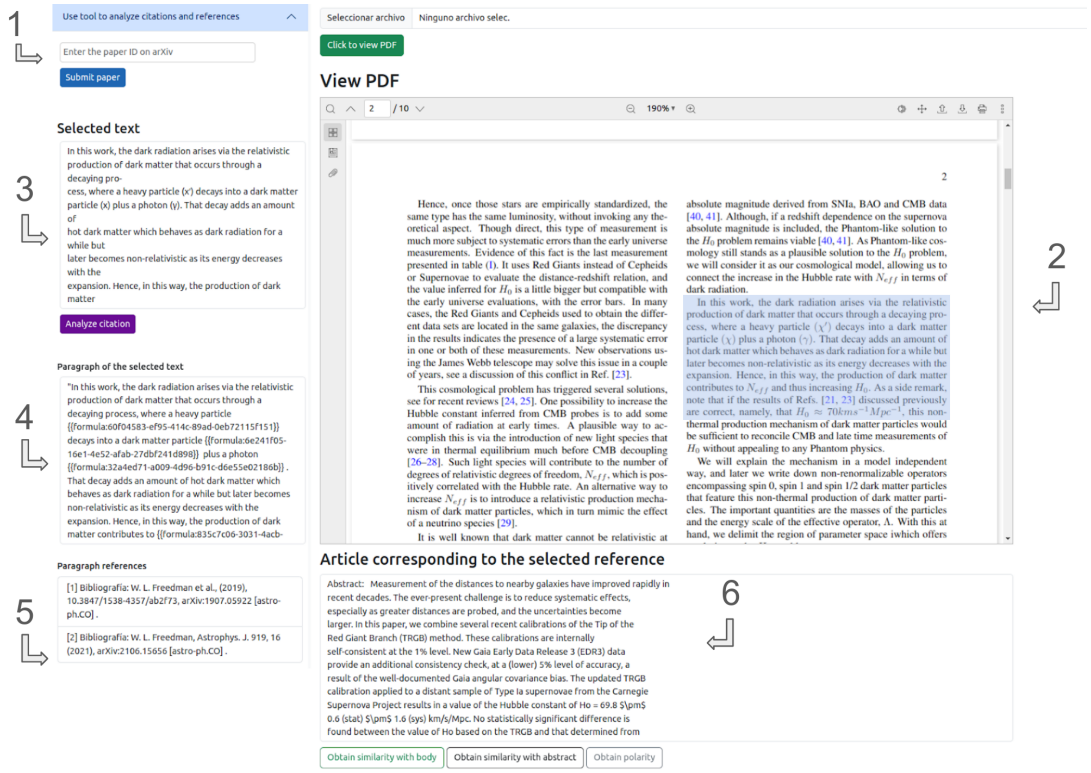

Figura 4.1: Interfaz de la aplicación con las diferentes áreas numeradas

[4] `https://opensearch.org/`

Siguiendo la Figura 4.1, se va a explicar cómo debe ser el uso de la interfaz:

- En primer lugar, el usuario introduce el ID de un artículo de arXiv en el área de texto señalada por el **1** y presiona el botón que hay debajo. Tras hacer click en el botón, aparecerá el artículo correspondiente en formato PDF en la zona marcada por el **2**.
- En este PDF visualizado, el usuario selecciona texto (puede seleccionar texto que no contenga una referencia, pero realmente la parte de interés requiere seleccionar un fragmento del PDF que la contenga). Tras seleccionar el texto tal y como se muestra en la Sección **2** en azul, el texto seleccionado aparecerá automáticamente en el área de texto marcada por el **3**. Tras aparecer este texto, el botón que se encuentra debajo pasará a activarse y por tanto a poder hacer click en él. Tras hacer click en este botón, tras unos segundos, aparecerá en el área de texto marcada por el **4** el párrafo correspondiente al texto seleccionado por el usuario. Además, a la vez aparecerá en el área marcada por el **5** todas las referencias de ese párrafo.
- Una vez aparezca la lista de referencias, el usuario hará click en aquella que desea analizar (esta referencia deberá pertenecer a arXiv, si no, aparecerá un mensaje de aviso en la esquina superior derecha de la pantalla). Tras hacer click en una referencia de arXiv, en el área marcada por el **6** aparecerá el *abstract* y el cuerpo completo del documento correspondiente a la referencia.
- Debajo del área **6** aparecen tres botones, que pasarán a estar activos cuando aparezca el documento en el área **6**.
- El primero de ellos, "Obtain similarity with body", tras presionarlo, calculará la similitud entre el texto seleccionado por el usuario en el área **2** y cada uno de los párrafos del cuerpo del documento referenciado, que ahora aparece en el área **6**. Aquel que obtenga mayor similitud, se mostrará en un gráfico, con las similitudes calculadas por cada uno de los cinco modelos utilizados. Además, se mostrará el párrafo completo debajo del gráfico. Se puede observar en el Anexo 4 una captura de cómo se muestra esto en la herramienta.
- En caso de presionar el segundo botón; "Obtain similarity with *abstract*", se mostrará en un gráfico la similitud calculada por cada uno de los cinco modelos entre el *abstract* del documento referenciado en el área marcada con un **2**.
- En caso de presionar el tercer botón; "Obtain polarity", se mostrará un gráfico con la polaridad de la cita, calculada por el modelo creado específicamente para obtener la polaridad. En este gráfico, se muestran las probabilidades de que la cita pertenezca a cada una de estas tres clases: positiva, neutra o negativa. Esto se puede observar en el Anexo 4, en la Figura 4.3.4.

4.3. Implementación del *backend*

En primer lugar se va a describir la implementación del servidor y en segundo la de la base de datos.

4.3.1. Implementación del servidor

El servidor está compuesto por un conjunto de módulos tal y como se ha descrito en el diseño.

- **Módulo de procesamiento de PDF:** Para la implementación de este módulo se han empleado varias bibliotecas, como `PyPDF2` para la extracción de texto de PDFs, y `requests` para la descarga de archivos desde la web. A través del módulo se ofrecen diferentes funcionalidades. En primer lugar, la capacidad de extraer el ID de un artículo de arXiv a partir del texto completo del artículo o de una referencia específica dentro del texto. Posteriormente, proporciona la funcionalidad de descargar archivos PDF de arXiv utilizando el ID del artículo proporcionado, permitiendo especificar el directorio de destino para guardar el archivo descargado. También incluye la capacidad de extraer el texto completo de un PDF descargado, facilitando así el procesamiento del contenido del artículo.
- **Módulo de peticiones y respuestas HTTP:** Este módulo facilita la comunicación entre el servidor y los clientes mediante el protocolo HTTP. Ofrece una variedad de funcionalidades para manejar solicitudes y respuestas HTTP, utilizando rutas para gestionarlas. Se basa en el framework Flask para Python y se integra con Elasticsearch para el almacenamiento y recuperación eficientes de datos. También gestiona todo lo relacionado con las sesiones de usuario.
- **Módulo de procesamiento de texto:** Este módulo está dedicado a la manipulación y procesamiento de texto utilizando herramientas de PLN. En él se emplean varias bibliotecas de Python, como `re`, para la manipulación de cadenas y búsqueda mediante expresiones regulares, `spaCy` para el procesamiento de texto en inglés, y `time` para la medición de tiempos de ejecución.

  La clase principal, `TextProcessor`, proporciona una serie de métodos diseñados para filtrar y procesar textos, extraer citas y comparar contenidos textuales. Entre sus funcionalidades principales se encuentran la limpieza de texto para eliminar caracteres no deseados, la extracción de citas mediante expresiones regulares y la obtención de listas de palabras alfabéticas de textos y citas utilizando el modelo de lenguaje de `spaCy`. También se implementa en él el código necesario para comparar conjuntos de palabras y poder determinar si son suficientemente similares como para poder decir que coinciden.

  Problema de comparación de texto

  Un problema importante a considerar surge en el inicio del proceso desatado por el usuario en el momento en que, tras haber cargado un artículo, selecciona una parte del texto del mismo con el fin de buscar y calcular la adecuación del propio texto con los contenidos de las citas que aparecen el párrafo que lo contiene. Esto supone que hay que recorrer todos los párrafos del artículo y ver cuál de ellos encaja mejor con el texto seleccionado. Es necesario tener en cuenta que el texto de un archivo en formato PDF puede contener caracteres de control, caracteres extraños que correspondan a símbolos provenientes de fórmulas, etc., de forma que el *string*

correspondiente a la selección no coincidirá, en general, con el texto en que se ve en el PDF. Por lo tanto, el problema a resolver es ver en qué párrafo del artículo encaja mejor el string obtenido del PDF (el texto seleccionado). Encontrar el mejor párrafo requiere recorrer todos los párrafos del artículo. Por lo tanto, es importante que la operación de comparación sea eficiente. Por ello se estudiaron posibles alternativas. Básicamente, si `text` es el texto seleccionado, y `par` es un párrafo, el problema consiste en encontrar la subcadena de `text` más larga que esté contenida en `par`.

Se implementaron dos soluciones a partir de los métodos descritos en Cormen et al. (2001) [21]. Por un lado, una adaptación del que denominan como solución *naive*: para cada posición de `text`, y cada posición de `par`, contar coincidencias consecutivas. Esto lleva a una solución de coste $O(n^3)$, con n la longitud más larga de ambas cadenas. Alternativamente, se implementó una solución basada en el algoritmo de búsqueda de patrones Knuth-Morris-Pratt (KMP) [21]. Con este se obtiene una solución de coste $\Theta(n^2)$.

En las pruebas realizadas no se constató una mejora real con la segunda solución. Posiblmente sea debido a que el algoritmo KMP funciona mejor con textos largos; pero este no es el caso, pues aquí se ha de comparar el texto seleccionado por el usuario (que será corto) con un párrafo del artículo (que tampoco será largo). Finalmente, el sistema utiliza algoritmo KMP, por su mejor coste teórico.

- **Módulo de gestión de Opensearch:** Este módulo proporciona una serie de funciones que permiten interactuar con un servidor OpenSearch para realizar operaciones específicas relacionadas con la búsqueda y el procesamiento de documentos. Permite establecer conexiones con el servidor OpenSearch utilizando los detalles de host, puerto y credenciales de autenticación. Ofrece también métodos para verificar la conexión al servidor y para obtener información específica de documentos utilizando sus identificadores únicos, como el identificador de un artículo (`paper_id`).

  Una de las funciones clave del módulo es la capacidad de recuperar el texto completo y la bibliografía de un documento dado su `paper_id`. Este proceso implica realizar una consulta al servidor OpenSearch para obtener el documento específico, luego deserializarlo y extraer el texto y la información de la bibliografía del documento. Además, el módulo ofrece funcionalidades para obtener los tokens de palabras del texto de un documento, lo que será útil para el análisis de texto y la extracción de características.

  Este módulo va a ser utilizado en todas aquellas operaciones en las que sea necesario hacer algún tipo de consulta a la base de datos, como puede ser la búsqueda de documentos, la obtención de las citas contenidas en un párrafo concreto y más.

- **Módulo del análisis de contenido:** Este módulo proporciona una serie de funcionalidades para realizar operaciones relacionadas con la comparación de texto entre citas y artículos. En él, se obtiene la similitud entre una cita de un artículo y el cuerpo de un artículo, utilizando los modelos pre-entrenados descritos en la Sec-

ción 2.2, además de un modelo propio entrenado como se detalla más adelante en la Sección 4.3.3.

4.3.2. Implementación de la base de datos

Tras tomar la decisión con respecto a qué campos de los JSON indexar, tal y como se ha señalado en la Sección 3.2.1, se procedió a la creación de un índice para los documentos en Opensearch.

Tras la creación del índice, se procedió a la inserción de todos los documentos en Opensearch. Estos documentos se encontraban organizados en un total de 26 directorios, conteniendo los archivos JSON. Cada archivo JSON puede contener uno o varios artículos. En total, había 165.000 artículos. Se realizó una pequeña prueba de carga, para calcular una aproximación al tiempo requerido para la inserción de todos los artículos. La prueba consistió en la inserción de los artículos contenidos en uno de los directorios. Este directorio constaba de 1596 artículos. El tiempo dedicado a la indexación fue de 178.07 segundos. Se extrapoló que el tiempo aproximado que iba a ser necesario para la inserción del dataset parcial sería de aproximadamente 5,11 horas.

Es necesario recalcar que, tras examinar varios campos a indexar en diferentes JSONs, se observó la presencia de caracteres especiales como acentos diacríticos y otros símbolos especiales en los nombres de muchos autores y, por tanto, se decidió utilizar una lista de conversión de caracteres especiales de LaTeX a Unicode, para realizar una especie de preprocesamiento de los campos a indexar de los artículos. Esta lista asigna a cada carácter Unicode su correspondiente representación en LaTeX. Ver Anexo 2. El propósito de esta lista es normalizar los nombres de autores y los títulos de los documentos antes de su inserción en Opensearch, asegurando así que no haya problemas al buscar documentos por título o por nombre de autor, independientemente de los caracteres especiales que contengan.

4.3.3. Entrenamiento del modelo para calcular similitudes

El entrenamiento se ha hecho utilizando la técnica Word2Vec, la cual, es una técnica de Procesamiento del Lenguaje Natural que utiliza redes neuronales para aprender representaciones vectoriales de palabras. Estos vectores (*embeddings*) capturan las relaciones semánticas entre las palabras, de manera que palabras con significados similares tendrán vectores cercanos en el espacio vectorial. El conjunto de datos utilizado para el entrenamiento del modelo ha sido el mismo que el utilizado para la implementación de la base de datos. De estos artículos, el campo usado para el entrenamiento del modelo ha sido `body_text`, el cual contiene el cuerpo del artículo (el contenido del artículo exceptuando el título, *abstract*, referencias, ...).

La implementación se ha llevado a cabo utilizando la biblioteca Gensim[5], ya que se trata de una biblioteca ampliamente reconocida por resultar muy eficaz a la hora de la creación de modelos de PLN.

[5] `https://radimrehurek.com/gensim/`

Antes de comenzar con el entrenamiento se implementaron funcionalidades para realizar el pre-procesamiento del texto, eliminando secuencias de números y carácteres no alfanuméricos y además, la conversión del texto a minúsculas. A continuación, se construyó el vocabulario del modelo. Para ello, se recorren todos los artículos, extrayendo las palabras.

Una vez obtenido el vocabulario, se pasó a la fase de entrenamiento. Esta se llevó a cabo de manera incremental. El entrenamiento se realizó utilizando la arquitectura Skip-gram. Las razones detrás de esta decisión en lugar de CBOW son varias y se extraen de artículos como el de Mikolov et al. (2013) [12] y Rong (2016) [13]. Skip-gram ofrece mayor flexibilidad y robustez en comparación con CBOW al manejar vocabularios extensos. Está diseñado para funcionar eficazmente incluso cuando los datos de entrenamiento son limitados, lo que lo hace adecuado para conjuntos de datos más grandes y diversos.

Cuando se combina con técnicas como el muestreo negativo (según lo define Rong (2016) [13], donde "para lidiar con la dificultad de tener demasiados vectores de salida que necesitan ser actualizados por iteración, solo se actualiza una muestra de ellos"), Skip-gram puede ser entrenado eficientemente en conjuntos de datos muy grandes, mejorando aún más su capacidad para aprender *embeddings* de palabras de alta calidad en comparación con CBOW.

Como parámetros fundamentales del entrenamiento se tomaron los siguientes: `window = 5`, `dimension = 200`, `epochs = 2`, `workers = 15`. Estos parámetros se eligieron como un compromiso entre la calidad del modelo y el tiempo de computación y recursos requeridos, dado que se iba a realizar en un ordenador personal.

Inicialmente, se contempló la posibilidad de ejecutar varios procesos en paralelo, cada uno encargado de procesar un subconjunto de artículos. El objetivo era generar un vocabulario y un modelo para cada subconjunto, con la intención de luego fusionar estos modelos en uno único. Sin embargo, esta estrategia resultó inviable debido a la incompatibilidad de los vectores de palabras generados en diferentes conjuntos de datos. En otras palabras, los vectores de palabras entrenados en diferentes conjuntos podrían no estar en el mismo espacio vectorial, lo que podría afectar negativamente a la preservación de las relaciones semánticas entre las palabras. Este problema se discute en detalle en el artículo de Mikolov et al. (2013) [12], donde se destaca la importancia de contar con un espacio vectorial compartido y alinear los vectores en el contexto de Word2Vec. Se optó entonces por un enfoque de entrenamiento incremental y por fases. Se comenzó seleccionando un conjunto inicial de directorios y generando el vocabulario y el modelo correspondiente. En cada fase subsiguiente, se reutilizó el modelo previamente generado como punto de partida y se lo reentrenó con el vocabulario de un nuevo conjunto de directorios. Este proceso se repitió hasta completar el procesamiento de todos los artículos deseados. Este enfoque se implementó para minimizar el riesgo de pérdida de trabajo en caso de fallos durante el entrenamiento, garantizando así la conservación del progreso y las horas invertidas en el proceso de entrenamiento de los modelos.

Durante los experimentos iniciales se observó que a pesar de establecer 15 *workers* como parámetro, realmente no llegaba a usar más de 3 en paralelo. La causa era que el pre-procesamiento de los ficheros que se llevaba a cabo antes de alimentar a los *workers* era costoso y no daba tiempo a alimentar a más. Por eso, se optó, en una segunda implementación en la cual todo el pre-procesamiento de los ficheros se hizo como paso previo al

entrenamiento, de manera que la alimentación a los *workers* era inmediata.

El tiempo de entrenamiento fue de 664.06 segundos para la primera fase, de 1445.23 segundos para la segunda, de 203346.87 segundos para la tercera y de 237549.73 segundos para la última. En total, el tiempo aproximado fue de 123.06 horas. Es importante recalcar que el entrenamiento se llevó a cabo en un ordenador con las siguientes características: 12th Gen Intel@ Core i7-12700x20, utilizando un sistema operativo Ubuntu 24.04 LTS, con 32 GB de RAM.

4.3.4. Entrenamiento del modelo para calcular polaridad

Para llevar a cabo el entrenamiento del modelo para calcular la polaridad de una cita, se ha seguido el artículo de Yu y Hua (2023) [8], que además proporciona un GitHub [6] con el código implementado para ello. El conjunto de datos se ha obtenido de SCICite, propuesto por Arman et al. (2019) [2]. Este conjunto está formado por una gran cantidad de artículos científicos sobre medicina y ciencias de la computación. La razón por la que se ha usado este conjunto de datos y no el mismo que para el modelo de similitud, es que el conjunto de datos de arXiv no está anotado para polaridad, mientras que este segundo es el que se usa en Yu y Hua (2023) [8], y además se considera que puede ser una aproximación razonable debido al solapamiento de documentos existente al menos en la categoría de informática.

En ese artículo, utilizan un conjunto de datos para entrenamiento y otro para pruebas. Ambos pertenecientes a SCICite. Al igual que en este artículo, se han utilizado 50 *epochs*. Se reconoce que las Feedforward Neural Networks (FNN) no constituyen el método óptimo para entrenar el modelo, sin embargo, debido a limitaciones de tiempo y recursos computacionales disponibles, se ha optado por utilizarlas igualmente. El entrenamiento con el conjunto de datos de SCICite, llevó un total de 15.12 horas de *elapsed time*, que corresponden a 174,5 horas de tiempo de usuario, ya que trabajaban 12 hilos en paralelo. Se obtuvieron los siguientes resultados:

```
50/50 - 100.00%
[train] loss: 0.5566, acc: 99.49, micro f1: 0.99, macro f1: 0.98
[test] loss: 0.6125, acc: 93.81, micro f1: 0.94, macro f1: 0.83
best loss: 0.6048, best acc: 94.63

50/50 - 100.00%
[train] loss: 0.5546, acc: 99.68, micro f1: 1.00, macro f1: 0.99
[test] loss: 0.6059, acc: 94.38, micro f1: 0.94, macro f1: 0.84
best loss: 0.5979, best acc: 95.45
```

[6] https://github.com/UFOdestiny/DictSentiBERT

4.4. Evaluación de la adecuación de los modelos a los textos del dataset

El cálculo de las palabras fuera del vocabulario es una práctica importante para medir los resultados de un modelo en el Procesamiento del Lenguaje Natural (PLN). Estas palabras deben ser tratadas adecuadamente para no degradar la calidad de las aplicaciones de procesamiento de lenguaje natural (NLP). Cuanto mayor sea este conjunto, peor será la calidad del modelo, como se explica en [22].

Las palabras OOV son aquellas que aparecen en un conjunto de datos pero no están presentes en el vocabulario del modelo, lo cual puede ocurrir por razones como que el modelo ha sido entrenado con un conjunto de datos diferente, o porque las palabras contienen errores tipográficos o nombres propios. También pueden aparecer palabras inexistentes debido a que estas "palabras" se corresponden en realidad con una url, dirección de correo electrónico, palabras extranjeras, etc.

Contar las palabras OOV es crucial para comprobar el posterior rendimiento de un modelo por distintas razones. En primer lugar, un alto porcentaje de palabras OOV indica que el modelo no está bien adaptado al dominio específico del conjunto de datos, lo que puede llevar a un rendimiento deficiente en tareas de PLN como por ejemplo el análisis de sentimientos. En segundo lugar, ayuda a evaluar la cobertura del vocabulario del modelo: un vocabulario amplio y bien representado generalmente mejora el rendimiento del modelo. En tercer lugar, la capacidad de un modelo para manejar palabras OOV puede indicar su capacidad para generalizar a nuevos datos, ya que los modelos que pueden inferir el significado de palabras desconocidas o adaptarse a ellas tienden a ser más robustos.

Para obtener el número de palabras OOV de los distintos modelos, se utilizó el mismo conjunto de datos para entrenar el modelo propio que para comprobar qué palabras de estas están OOV, este es el que debería tener menor número de palabras OOV. Los resultados obtenidos para los diferentes modelos utilizados en el proyecto, se pueden observar en la tabla de la Figura 4.1. Tanto BERT como SBERT utilizan el tokenizador de BERT, por lo que solamente aparece una vez en la tabla.

Tal y como se puede observar en la tabla 4.1, aquel modelo con menor número de palabras OOV es el propio. Esto es debido a haber realizado su entrenamiento con el mismo conjunto de datos con el que se han comprobado estas palabras OOV.

Fichero de Texto	Total de Palabras	Modelo	OOV
`Fase 1`	67.588.747	fasttext	3.296.488
		Google	11.525.280
		BERT	7.619.017
		propio	367.749
`Fase 2`	154.636.595	fasttext	7.235.363
		Google	25.839.695
		BERT	17.178.156
		propio	775.366
`Fase 3`	323.264.834	fasttext	13.459.129
		Google	51.616.057
		BERT	34.132.153
		propio	1.452.619
`Fase 4`	367.524.793	fasttext	15.097.768
		Google	58.348.127
		BERT	38.853.595
		propio	1.648.672

Tabla 4.1: Cálculos de OOV de cada modelo

5 Evaluación de los modelos

Para evaluar la calidad de los modelos, se ha optado por un análisis cualitativo en lugar de uno cuantitativo, ya que un análisis cuantitativo requeriría la anotación manual de cientos de artículos.

Para el análisis cualitativo realizado se ha utilizado una muestra de 10 artículos. La selección de estos artículos se ha basado en la necesidad de que las citas contenidas en ellos referencien a artículos presentes en la base de datos y que las referencias sean válidas (es decir, que no se trate simplemente de una referencia a una imagen, fórmula o demostración matemática). Para identificar los artículos que contienen referencias a otros artículos dentro de la base de datos, se ha desarrollado un pequeño *script* que extrae los identificadores de arXiv tanto de los artículos citados como de los artículos que los referencian.

Los datos recopilados se han plasmado en un formulario de Google, que incluye las citas, los *abstracts* de los artículos referenciados por dichas citas y el párrafo con mayor similitud con cada cita en el artículo referenciado. Para obtener una evaluación objetiva, se ha solicitado la opinión de un pequeño grupo de 8 personas compuesto por 2 conocedores de los temas tratados y 6 no concedores de estos. Los evaluadores han tenido la tarea de determinar, según su criterio, la polaridad de cada cita (*positiva, negativa o neutra*). Además han evaluado el grado de similitud entre la cita y el *abstract*, así como entre la cita y el párrafo seleccionado del artículo, categorizándolas como alta, media o baja. Por otro lado, para comparar cómo de similares son las respuestas de los evaluadores con las de la herramienta, se han establecido distintos rangos en los resultados de los distintos modelos para poder clasificar los porcentajes obtenidos por ellos en similitud *alta, media* o *baja*. Para la evaluación de la concordancia entre los múltiples evaluadores, se ha usado la medida Kappa de Fleiss [1]: es una medida estadística del acuerdo entre un número de evaluadores sobre un número de elementos en base a un conjunto de categorías. En el Anexo 5 se pueden ver las tres tablas correspondientes a la evaluación de la polaridad de la cita, de la similitud entre cita y *abstract* y la similitud entre cita y párrafo. Los resultados obtenidos para los anotadores son tres. La Kappa de Fleiss de los evaluadores con la polaridad es de `0.38`, lo que indica que hay un "acuerdo justo" entre ellos. La Kappa de Fleiss en cuanto a la similitud entre cita y *abstract* es de `0.058`, lo que indica un "acuerdo ligero". Por último, el valor obtenido para el acuerdo entre cita y párrafo es de `0.095`, lo que también indica un "acuerdo ligero".

Dados los resultados obtenidos, se podría considerar las opiniones de los anotadores como

[1] `https://en.wikipedia.org/wiki/Fleiss'_kappa`

válidas con respecto a la polaridad, mientras que para las de similitud, sería necesario revisar con otra medida que capturase más información, aunque para tener un Kappa más fiable, sería necesario utilizar un número mucho mayor de artículos, sin embargo, el coste manual de la anotación de cientos de artículos se quedaba fuera de los objetivos de este trabajo.

También se considera necesario recalcar que el perfil de los distintos anotadores es muy heterogéneo y que en una futura iteración, sería necesario trazar unas guías de anotación más completas, como cómo se entiende la polaridad y la similitud en el trabajo. Para calcular la polaridad, por ejemplo, en la cita "The limit of the detailed balance for systems which include some irreversible elementary processes (without reverse processes) was recently studied in detail [9, 10]", que se ha utilizado en el formulario, se habla de una limitación del sistema, lo cual podría ser marcado como polaridad negativa, pero al mismo tiempo, utiliza las referencias de la cita a favor de su argumento, como ejemplo de otros artículos que recalcan esta limitación. Por tanto, ejemplos como este, hacen muy complicada su anotación.

La validez de los modelos debería comparar sus medidas con un conjunto de soluciones correctas. La anotación y el bajo acuerdo entre anotadores ha demostrado sin embargo que llegar a única solución correcta en tareas subjetivas como la anotación de polaridad es sino imposible, sí muy difícil, sobre todo teniendo en cuenta el diverso perfil de los anotadores y falta de preparación para la tarea de anotación en particular. En cualquier caso, se va a considerar para cada artículo la polaridad obtenida por la mayoría de los anotadores. Para poder comparar las respuestas de los notadores con respecto a la similitud (*baja*, *media*, *alta*), ya que los modelos simplemente daban un porcentaje, se observó el mayor y menor porcentaje obtenido por cada uno de los modelos y se dividió en terciles, para así poder establecer qué porcentajes quedaban dentro de cada una de estas respuestas.

En el caso de la polaridad, hay coincidencia en un 60 %. En los modelos para similitud, también se ha observado la solución de cada modelo frente al covenio de la mayoría de anotadores. En caso de similitud para cita y *abstract*, en el modelo de Fasttext, obtiene un 50 % de coincidencia, el modelo de obtiene un 50 %, el modelo propio un 40 %, BERT un 50 % y el de sentence transformers + BERT un 50 %. En la medida de similitud entre cita y párrafo, los resultados de los modelos frente a los obtenidos por la mayoría de anotadores se comentan a continuación. En el modelo de Fasttext, Google, BERT y sentence transformers + BERT, existe una coincidencia del 20 %, mientras que la del modelo propio es de un 10 %.

6 Conclusiones y trabajo futuro

A nivel personal, me ha gustado mucho hacer este trabajo de fin de grado. He aprendido mucho y el tema me ha parecido muy interesante. Ha sido una experiencia enriquecedora que me ha permitido ampliar mis conocimientos sobre este tema, que además, al no haber cursado ciertas asignaturas, no era muy amplio en el inicio del proyecto.

Aunque la implementación de la parte del servidor utilizando Python ha permitido realizar diversas tareas relacionadas con el entrenamiento de modelos, este lenguaje presenta ciertas limitaciones en cuanto al paralelismo. Si se hubiese utilizado otro lenguaje con mejor soporte para el paralelismo, posiblemente se podría haber reducido el tiempo requerido para algunos procesamientos.

El trabajo a desarrollar en un futuro abarca diferentes áreas. En primer lugar, uno de los objetivos del trabajo futuro sería permitir el uso de la herramienta no solo para artículos presentes en arXiv, sino también para aquellos que proceden de otros orígenes como puede ser Google Scholar.

También se propone la ampliación de la herramienta desarrollada para incluir la verificación de afirmaciones científicas, una tarea que implica seleccionar resúmenes de la literatura científica que contengan evidencias que apoyen o refuten una afirmación dada, como se lleva a cabo en Wadden et al. (2020) [23].

Además, una vez calculadas las similitudes entre los diferentes artículos mediante modelos textuales, se entrenaría un modelo de aprendizaje automático para predecir la probabilidad de que un documento cite a otro, de manera análoga a la propuesta de La Quatra et al. (2021) [7].

También es necesario validar los modelos. Para ello, se requeriría hacer un estudio mediante la anotación manual de cientos o miles de artículos, con interpretaciones hechas por personas conocedoras del dominio y su correlación con los resultados obtenidos por los métodos.

Referencias

[1] T. Saier, J. Krause y M. Färber, «unarXive 2022: All arXiv Publications Pre-Processed for NLP, Including Structured Full-Text and Citation Network,» en *2023 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, IEEE, jun. de 2023. dirección: http://dx.doi.org/10.1109/JCDL57899.2023.00020.

[2] A. Cohan, W. Ammar, M. van Zuylen y F. Cady, «Structural Scaffolds for Citation Intent Classification in Scientific Publications,» en *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran y T. Solorio, eds., Minneapolis, Minnesota: Association for Computational Linguistics, jun. de 2019, págs. 3586-3596. DOI: 10.18653/v1/N19-1361. dirección: https://aclanthology.org/N19-1361.

[3] Y. Zhang et al., *When Large Language Models Meet Citation: A Survey*, 2023. arXiv: 2309.09727.

[4] S. Te, A. Barhoumi, M. Lentschat, F. Bordignon, C. Labbé y F. Portet, «Citation Context Classification: Critical vs Non-critical,» en *Proceedings of the Third Workshop on Scholarly Document Processing*, A. Cohan et al., eds., Gyeongju, Republic of Korea: Association for Computational Linguistics, oct. de 2022, págs. 49-53. dirección: https://aclanthology.org/2022.sdp-1.6.

[5] F. Bordignon, «Critical citations in knowledge construction and citation analysis: from paradox to definition,» *Scientometrics*, vol. 127, n.º 2, págs. 959-972, 2022. DOI: 10.1007/s11192-021-04226-0. dirección: https://enpc.hal.science/hal-03468402.

[6] J. M. Nicholson et al., «scite: A smart citation index that displays the context of citations and classifies their intent using deep learning,» *Quantitative Science Studies*, vol. 2, n.º 3, págs. 882-898, nov. de 2021, ISSN: 2641-3337. DOI: 10.1162/qss_a_00146. eprint: https://direct.mit.edu/qss/article-pdf/2/3/882/1970740/qss_a_00146.pdf. dirección: https://doi.org/10.1162/qss%5C_a%5C_00146.

[7] M. La Quatra, L. Cagliero y E. Baralis, «Leveraging full-text article exploration for citation analysis,» *Scientometrics*, vol. 126, n.º 10, 2021, ISSN: 1588-2861. DOI: 10.1007/s11192-021-04117-4. dirección: https://doi.org/10.1007/s11192-021-04117-4.

[8] D. Yu y B. Hua, «Sentiment Classification of Scientific Citation Based on Modified BERT Attention by Sentiment Dictionary,» en *Proceedings of the Joint Workshop of the 4th Extracting and Evaluation of Knowledge Entities from Scientific Documents (EEKE2023) and the 3rd AI + Informetrics (AII2023)*, C. Zhang et al., eds.,

vol. 3451, CEUR-WS.org, 2023, págs. 59-64. dirección: https://ceur-ws.org/Vol-3451/paper10.pdf.

[9] A. Kulkarni y A. Shivananda, *Natural Language Processing Recipes: Unlocking Text Data with Machine Learning and Deep Learning using Python*. Apress, 2019.

[10] R. Collobert y J. Weston, «A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning,» en *Proceedings of the 25th International Conference on Machine Learning*, ép. ICML '08, Helsinki, Finland: ACM, 2008, págs. 160-167. dirección: http://doi.acm.org/10.1145/1390156.1390177.

[11] J. Devlin, M.-W. Chang, K. Lee y K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019. arXiv: 1810.04805 [cs.CL].

[12] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado y J. Dean, «Distributed Representations of Words and Phrases and their Compositionality,» en *Advances in Neural Information Processing Systems*, C. Burges, L. Bottou, M. Welling, Z. Ghahramani y K. Weinberger, eds., vol. 26, Curran Associates, Inc., 2013. dirección: https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

[13] X. Rong, *word2vec Parameter Learning Explained*, 2016. arXiv: 1411.2738 [cs.CL].

[14] C. Allen y T. Hospedales, *Analogies Explained: Towards Understanding Word Embeddings*, 2019. arXiv: 1901.09813 [cs.CL].

[15] E. R. Fonseca, J. L. G. Rosa y S. M. Aluísio, «Evaluating word embeddings and a revised corpus for part-of-speech tagging in Portuguese,» *Journal of the Brazilian Computer Society*, vol. 21, n.º 1, pág. 2, feb. de 2015, ISSN: 1678-4804. DOI: 10.1186/s13173-014-0020-x. dirección: https://doi.org/10.1186/s13173-014-0020-x.

[16] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu y P. Kuksa, «Natural Language Processing (Almost) from Scratch,» *Journal of Machine Learning Research*, vol. 12, n.º 76, págs. 2493-2537, 2011. dirección: http://jmlr.org/papers/v12/collobert11a.html.

[17] P. Bojanowski, E. Grave, A. Joulin y T. Mikolov, «Enriching Word Vectors with Subword Information,» *Transactions of the Association for Computational Linguistics*, vol. 5, L. Lee, M. Johnson y K. Toutanova, eds., págs. 135-146, 2017. dirección: https://aclanthology.org/Q17-1010.

[18] N. Reimers e I. Gurevych, «Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,» en *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, nov. de 2019. dirección: http://arxiv.org/abs/1908.10084.

[19] T. Saier, J. Krause y M. Färber, *unarXive: All arXiv Publications Pre-Processed for NLP, Including Structured Full-Text and Citation Network (full)*, Zenodo, mar. de 2023. DOI: 10.5281/zenodo.7752754. dirección: https://doi.org/10.5281/zenodo.7752754.

[20] T. Saier, J. Krause y M. Färber, *unarXive: All arXiv Publications Pre-Processed for NLP, Including Structured Full-Text and Citation Network (open subset)*, Zenodo, mar. de 2023. DOI: 10.5281/zenodo.7752615. dirección: https://doi.org/10.5281/zenodo.7752615.

[21] T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein, *Introduction to Algorithms*, 2nd. The MIT Press, 2001.

[22] J. V. Lochter, R. M. Silva y T. A. Almeida, «Deep Learning Models for Representing Out-of-Vocabulary Words,» en *Intelligent Systems*, R. Cerri y R. C. Prati, eds., Cham: Springer International Publishing, 2020, págs. 418-434.

[23] D. Wadden et al., «Fact or Fiction: Verifying Scientific Claims,» en *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, B. Webber, T. Cohn, Y. He e Y. Liu, eds., Online: Association for Computational Linguistics, nov. de 2020, págs. 7534-7550. DOI: 10.18653/v1/2020.emnlp-main.609. dirección: https://aclanthology.org/2020.emnlp-main.609.

Anexo 1

Formato de los datos proporcionados por Zenodo.

```
unarXive
========
arXiv source data used:
    all LaTeX sources up until Dec 31 2022
arXiv metadata used:
    snapshot from Jan 1 2023
OpenAlex data used:
    snapshot from Nov 28 2022

contents
========
-README                                  // this file
-LICENSE                                 // data license
-unarXive_<yymmdd>.tar.xz
    +-<yy>                               // year directory
        +-arXiv_src_<yymm>_<num>.jsonl   // unarXive data

links
=====

GitHub: https://github.com/IllDepence/unarXive
Zenodo: https://doi.org/10.5281/zenodo.7752754 (full)
        https://doi.org/10.5281/zenodo.7752615 (open subset)
Huggingface: https://huggingface.co/datasets/saier/unarXive_citrec
             https://huggingface.co/datasets/saier/unarXive_imrad_clf

usage
=====

(See GitHub repository for more detailed information.)

1. unpack
---------
unarXive is distributed as an XZ compressed TAR archive. You can unpack
```

it using either of the following methods.
    - On the command line using tar -xJf.
    - Graphically with 7zip (https://www.7-zip.org/).

2. load

Each decompressed file arXiv_src_<yy><mm>_<num>.jsonl is in the JSON Lines format (https://jsonlines.org/). This means each line is a JSON object and can, for example, be loaded in Python with json.loads(some_line).

3. use paper data

Papers are represented as shown in the example below, which is an excerpt from the paper 2105.05862 from the file arXiv_src_2105_034.jsonl. A full documentation of data fields is given further down.

Paper object:
```
{
  "paper_id": "2105.05862",
  "_pdf_hash": None,
  "_source_hash": "b7d5f27b5c8abc3bd8a44d875899fdc0d945a604",
  "_source_name": "2105.05862.gz",
  "metadata": {...},
  "discipline": "Physics",
  "abstract": {...},
  "body_text": [...],
  "bib_entries": {...},
  "ref_entries": {...}
}
```

One example paragraph in "body_text" is:
```
{
  "section": "Memory wave form",
  "sec_number": "2.1",
  "sec_type": "subsection",
  "content_type": "paragraph",
  "text": "The gauge choice leading us to this solution does not fix "
          "completely all the gauge freedom and an additional constraint "
          "should be imposed to leave only the physical degrees of freedom. "
          "This is done by projecting the source tensor {{formula:7fd88bcd-"
          "9013-433d-9756-b874472530d9}}  into its transverse-traceless (TT) "
          "components (see for example {{cite:80dbb6c8b9c12f561a8e585faceac5f"
          "4e104d60d}}). Doing this and without loss of generality, we will "
```

```
          "use the following very well known ansatz for the source term "
          "proposed in {{cite:bc9a8ca19785627a087ae0c01abe155c22388e16}}\n",
  "cite_spans": [...],
  "ref_spans": [...]
}
```

where "{{formula:7fd88bcd-9013-433d-9756-b874472530d9}}" refers in "ref_entries" to

```
{
  "latex": "S_{\\mu \\nu }",
  "type": "formula"
}
```

and "{{cite:bc9a8ca19785627a087ae0c01abe155c22388e16}}", for example, refers in "bib_entries" to

```
{
  "bib_entry_raw": "R. Epstein, The Generation of Gravitational Radiation by "
                   "Escaping Supernova Neutrinos, Astrophys. J. 223 (1978) "
                   "1037.",
  "contained_arXiv_ids": [],
  "contained_links": [
    {
      "url": "https://doi.org/10.1086/156337",
      "text": "Astrophys. J. 223 (1978) 1037.",
      "start": 87,
      "end": 117
    }
  ],
  "discipline": "Physics",
  "ids" {...}
}
```

data format
===========

root object (paper)
----

paper_id: arXiv ID of the paper
_pdf_hash: always None
_source_hash: SHA1 hash of the arXiv source file
_source_name: name of the arXiv source file

```
metadata: paper metadata from kaggle.com/datasets/Cornell-University/arxiv
discipline: scientific discipline of the paper
abstract: paper abstract copied from metadata
body_text: list of paper content sections (paragraphs, listings, etc.)
bib_entries: list of bibliographic references
ref_entries: list of non-textual content (figures, formulas, etc.)

  body_text list element
  ----------------------
  section: section name
  sec_number: section number
  sec_type: section type (section, subsection, etc.)
  content_type: content type (paragraph, listing, etc.)
  text: text content
  cite_spans: list of citation markers
  ref_spans: list of referenced non-textual content (figures, formulas, etc.)

    cite_spans list element
    -----------------------
    start: starting character offset in text
    end: ending character offset in text
    text: surface text
    ref_id: dictionary key for linked content in bib_entries

    ref_spans list element
    ----------------------
    start: starting character offset in text
    end: ending character offset in text
    text: surface text
    ref_id: dictionary key for linked content in ref_entries

  bib_entries element
  -------------------
  bib_entry_raw: raw bibliographic reference string
  contained_arXiv_ids: list of linked arXiv papers
  contained_links: list of embedded links
  discipline: scientific discipline of the cited paper
  ids: matched identifiers of referenced paper

    contained_arXiv_ids element
    ---------------------------
    id: ID of linked arXiv paper
    text: text segment in reference that the link was attached to
    start: starting character offset in bib_entry_raw
    end: ending character offset in bib_entry_raw
```

```
  contained_links_ids element
  ---------------------------
  url: URL of link
  text: text segment in reference that the link was attached to
  start: starting character offset in bib_entry_raw
  end: ending character offset in bib_entry_raw

  ids element:
  ------------
  open_alex_id: referenced paper's OpenAlex ID
  sem_open_alex_id: referenced paper's  SemOpenAlex ID
  pubmed_id: referenced paper's PubMed ID
  pmc_id: referenced paper's PMC ID
  doi: referenced paper's DOI
  arxiv_id: referenced paper's arXiv ID

ref_entries dictionary entry (tables and figures)
-------------------------------------------------
type: content type
caption: table/figure caption

ref_entries dictionary entry (mathematical notation)
----------------------------------------------------
type: always "formula"
latex: content of LaTeX math mode
```

Anexo 2

Lista de conversión de caracteres especiales de LaTeX a Unicode.

```
latexAccents = [
  [ u"à", "\\‘a" ], # Grave accent
  [ u"è", "\\‘e" ],
  [ u"ì", "\\‘\\i" ],
  [ u"ò", "\\‘o" ],
  [ u"ù", "\\‘u" ],
  [ u"ỳ", "\\‘y" ],
  [ u"À", "\\‘A" ],
  [ u"È", "\\‘E" ],
  [ u"Ì", "\\‘\\I" ],
  [ u"Ò", "\\‘O" ],
  [ u"Ù", "\\‘U" ],
  [ u"Ỳ", "\\‘Y" ],
  [ u"á", "\\’a" ], # Acute accent
  [ u"é", "\\’e" ],
  [ u"í", "\\’\\i" ],
  [ u"ó", "\\’o" ],
  [ u"ú", "\\’u" ],
  [ u"ý", "\\’y" ],
  [ u"Á", "\\’A" ],
  [ u"É", "\\’E" ],
  [ u"Í", "\\’\\I" ],
  [ u"Ó", "\\’O" ],
  [ u"Ú", "\\’U" ],
  [ u"Ý", "\\’Y" ],
  [ u"â", "\\^a" ], # Circumflex
  [ u"ê", "\\^e" ],
  [ u"î", "\\^\\i" ],
  [ u"ô", "\\^o" ],
  [ u"û", "\\^u" ],
  [ u"ŷ", "\\^y" ],
  [ u"Â", "\\^A" ],
  [ u"Ê", "\\^E" ],
```

```
[ u"Î", "\\^\\I" ],
[ u"Ô", "\\^O" ],
[ u"Û", "\\^U" ],
[ u"Ŷ", "\\^Y" ],
[ u"ä", "\\\"a" ],    # Umlaut or dieresis
[ u"ë", "\\\"e" ],
[ u"ï", "\\\"\\i" ],
[ u"ö", "\\\"o" ],
[ u"ü", "\\\"u" ],
[ u"ÿ", "\\\"y" ],
[ u"Ä", "\\\"A" ],
[ u"Ë", "\\\"E" ],
[ u"Ï", "\\\"\\I" ],
[ u"Ö", "\\\"O" ],
[ u"Ü", "\\\"U" ],
[ u"Ÿ", "\\\"Y" ],
[ u"ç", "\\c{c}" ],   # Cedilla
[ u"Ç", "\\c{C}" ],
[ u"œ", "{\\oe}" ],   # Ligatures
[ u"Œ", "{\\OE}" ],
[ u"æ", "{\\ae}" ],
[ u"Æ", "{\\AE}" ],
[ u"å", "{\\aa}" ],
[ u"Å", "{\\AA}" ],
[ u"{", "--" ],   # Dashes
[ u"|", "---" ],
[ u"ø", "{\\o}" ],    # Misc latin-1 letters
[ u"Ø", "{\\O}" ],
[ u"ß", "{\\ss}" ],
[ u"¡", "{!‘}" ],
[ u"¿", "{?‘}" ],
[ u"\\", "\\\\" ],    # Characters that should be quoted
[ u"~", "\\~" ],
[ u"&", "\\&" ],
[ u"$", "\\$" ],
[ u"{", "\\{" ],
[ u"}", "\\}" ],
[ u"%", "\\%" ],
[ u"#", "\\#" ],
[ u"_", "\\_" ],
[ u"", "$\\ge$" ],   # Math operators
[ u"", "$\\le$" ],
[ u"", "$\\neq$" ],
[ u"©", "\copyright" ], # Misc
[ u"ı", "{\\i}" ],
[ u"µ", "$\\mu$" ],
```

```
  [ u"°", "$\\deg$" ],
  [ u"‘", "‘" ],    #Quotes
  [ u"’", "’" ],
  [ u"\", "‘‘" ],
  [ u"", "’’" ],
  [ u",", "," ],
  [ u"„", ",," ],
]
```

Anexo 3

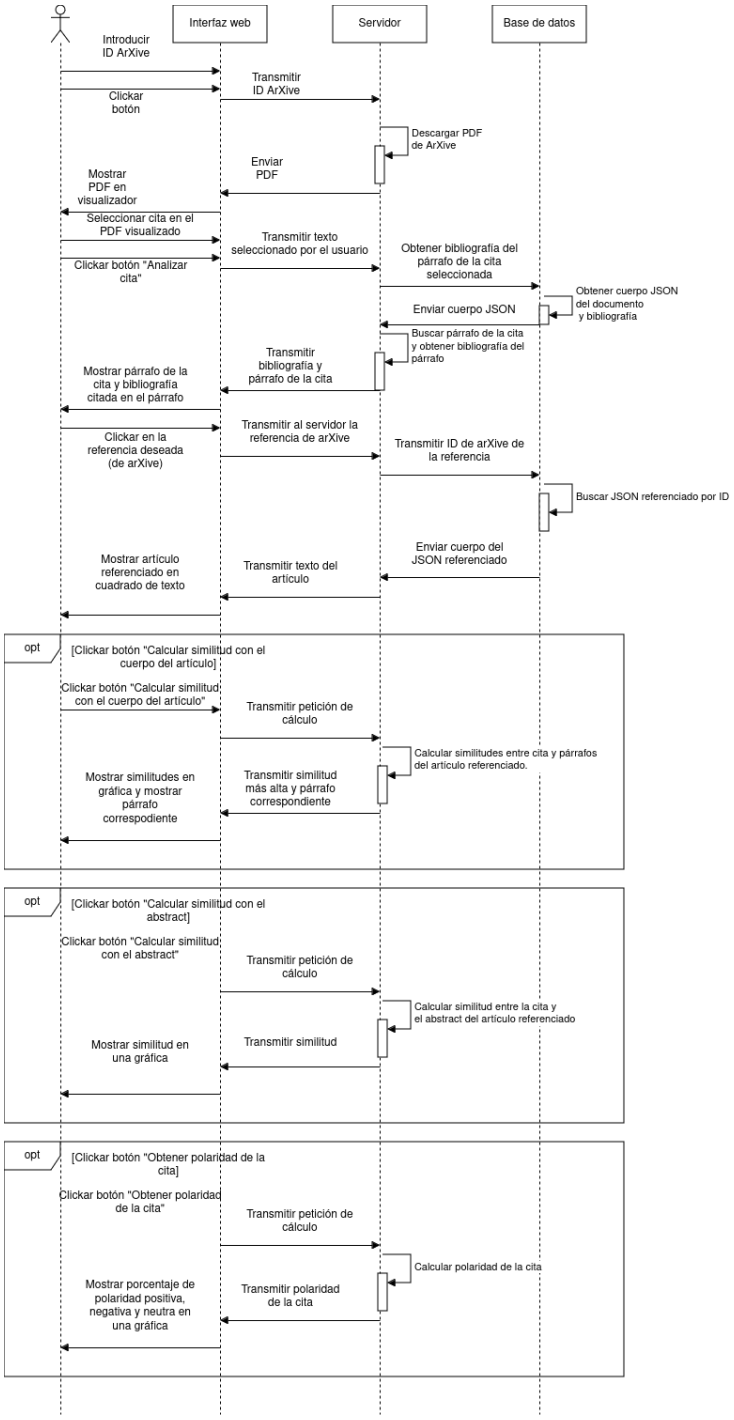

Figura 6.1: Diagrama de secuencia de alto nivel del flujo normal de la aplicación

Anexo 4

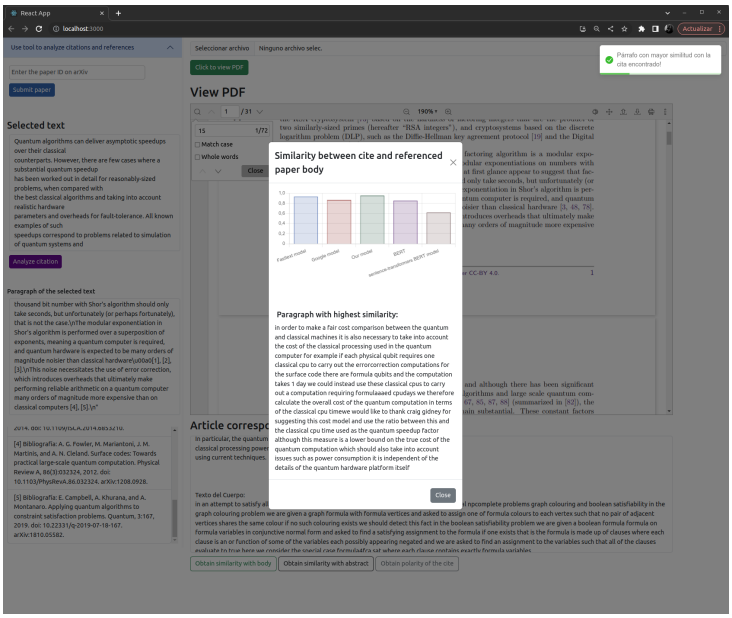

Figura 6.2: Gráfico de la similitud entre la cita y el párrafo con mayor similitud con esta del artículo referenciado

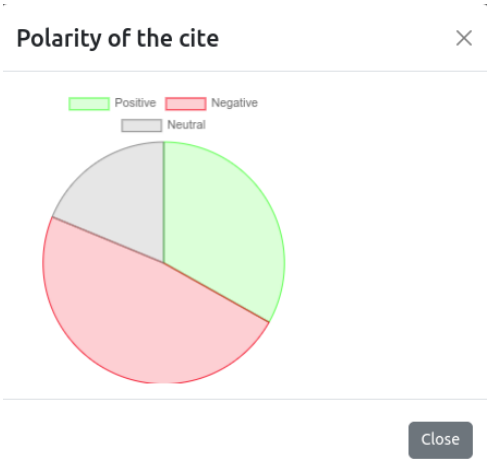

Figura 6.3: Gráfico de la probabilidad de que la cita pertenezca a las clases positiva, negativa o neutra

Anexo 5

Polaridad	Anotadores								
	Anotador1	Anotador2	Anotador3	Anotador4	Anotador5	Anotador6	Anotador7	Anotador8	Resultados herramienta
Artículo1	Neutra	Neutra	Neutra	Neutra	Neutra	Neutra	Neutra	Neutra	Neutra
Artículo2	Neutra	Negativa	Negativa	Negativa	Neutra	Neutra	Negativa	Negativa	Neutra
Artículo3	Neutra	Negativa	Neutra	Negativa	Neutra	Neutra	Neutra	Neutra	Neutra
Artículo4	Neutra	Neutra	Neutra	Positiva	Neutra	Neutra	Neutra	Neutra	Positiva/Neutra
Artículo5	Neutra	Neutra	Neutra	Positiva	Neutra	Positiva	Neutra	Neutra	Neutra
Artículo6	Negativa	Negativa	Negativa	Negativa	Negativa	Negativa	Negativa	Negativa	Positiva
Artículo7	Neutra	Neutra	Neutra	Negativa	Neutra	Neutra	Neutra	Neutra	Positiva
Artículo8	Neutra	Positiva	Neutra	Negativa	Neutra	Neutra	Neutra	Positiva	Neutra
Artículo9	Neutra	Neutra	Neutra	Neutra	Neutra	Neutra	Positiva	Neutra	Negativa
Artículo10	Negativa	Negativa	Negativa	Positiva	Neutra	Negativa	Negativa	Negativa	Negativa

Figura 6.4: Tabla con los resultados obtenidos de la evaluación de la polaridad

Cita/Abstract	Anotadores												
	Anotador1	Anotador2	Anotador3	Anotador4	Anotador5	Anotador6	Anotador7	Anotador8	Modelo1	Modelo2	Modelo3	Modelo4	Modelo5
Artículo1	Alta	Alta	Media	Baja	Media	Media	Alta	Media	Medio	Medio	Medio	Medio	Medio
Artículo2	Baja	Alta	Baja	Baja	Media	Baja	Baja	Alta	Medio	Medio	Medio	Medio	Medio
Artículo3	Media	Alta	Media	Baja	Media	Baja	Media	Baja	Medio	Medio	Medio	Medio	Medio
Artículo4	Media	Alta	Alta	Alta	Media	Baja	Baja	Media	Medio	Medio	Medio	Medio	Bajo
Artículo5	Alta	Alta	Alta	Alta	Alta	Baja	Media	Alta	Medio	Medio	Medio	Medio	Medio
Artículo6	Alta	Alta	Media	Baja	Media	Baja	Baja	Baja	Medio	Medio	Medio	Medio	Bajo
Artículo7	Alta	Alta	Media	Baja	Alta	Alta	Baja	Media	Medio	Medio	Medio	Medio	Medio
Artículo8	Baja	Baja	Baja	Baja	Baja	Baja	Baja	Media	Bajo	Bajo	Bajo	Bajo	Bajo
Artículo9	Media	Alta	Media	Alta	Media	Baja	Media	Baja	Medio	Medio	Alto	Medio	Medio
Artículo10	Baja	Baja	Baja	Alta	Media	Baja	Baja	Alta	Medio	Medio	Medio	Medio	Medio

Figura 6.5: Tabla con los resultados obtenidos de la evaluación de la similitud entre cita y *abstract*

Cita/Párrafo	Anotadores												
	Anotador1	Anotador2	Anotador3	Anotador4	Anotador5	Anotador6	Anotador7	Anotador8	Modelo1	Modelo2	Modelo3	Modelo4	Modelo5
Artículo1	Alta	Media	Alta	Alta	Alta	Media	Media	Alta	Medio	Medio	Medio	Medio	Medio
Artículo2	Baja	Alta	Baja	Alta	Alta	Alta	Baja	Alta	Medio	Medio	Medio	Medio	Medio
Artículo3	Alta	Media	Alta	Baja	Media	Baja	Baja	Baja	Medio	Medio	Medio	Medio	Medio
Artículo4	Alta	Alta	Alta	Alta	Alta	Alta	Baja	Media	Medio	Medio	Medio	Medio	Medio
Artículo5	Alta	Alta	Alta	Media	Alta	Alta	Alta	Media	Medio	Medio	Medio	Medio	Medio
Artículo6	Alta	Media	Media	Baja	Alta	Alta	Alta	Baja	Medio	Medio	Medio	Medio	Medio
Artículo7	Alta	Alta	Baja	Baja	Alta	Baja	Baja	Alta	Medio	Bajo	Medio	Medio	Medio
Artículo8	Alta	Baja	Alta	Alta	Alta	Alta	Alta	Alta	Medio	Medio	Medio	Medio	Medio
Artículo9	Media	Alta	Media	Baja	Media	Media	Baja	Media	Medio	Medio	Alto	Medio	Medio
Artículo10	Baja	Baja	Alta	Alta	Media	Media	Media	Media	Medio	Medio	Medio	Medio	Medio

Figura 6.6: Tabla con los resultados obtenidos de la evaluación de la similitud entre cita y párrafo