



UNIVERSIDAD
NEBRIJA

Extensión de Google Chrome para detección de noticias falsas.

UNIVERSIDAD NEBRIJA GRADO EN
INGENIERÍA INFORMÁTICA
MEMORIA TRABAJO FIN DE GRADO

Nombre Alumno: Juan Manuel Rodríguez Etchepare

Fecha: Junio, 2024



UNIVERSIDAD
NEBRIJA

Extensión de Google Chrome para detección de noticias falsas.

**UNIVERSIDAD NEBRIJA GRADO EN
INGENIERÍA INFORMÁTICA
MEMORIA TRABAJO FIN DE GRADO**

Juan Manuel Rodríguez Etchepare

Fecha: Junio, 2024

Nombre tutor: Adrián Pradilla Pórtoles

Dn. Juan Manuel Rodríguez Etchepare autoriza a que el presente trabajo se guarde y custodie en los repositorios de la Universidad Nebrija y además autoriza a su disposición en abierto.

Agradecimientos

Agradezco especialmente a mi tutor de trabajo de fin de grado, Adrián Pradilla Pórtoles, por haber sugerido la idea de este estudio y haber brindado su tiempo para contribuir a hacerlo posible.

Índice

Agradecimientos.....	4
Índice.....	5
Glosario de términos.....	6
Índice de ilustraciones.....	8
Índice de tablas.....	11
Resumen.....	12
Abstract.....	13
1. Introducción.....	15
1.1. Motivación.....	20
1.2. Antecedentes.....	21
1.2.1. Estado del arte.....	21
1.2.2. Necesidad detectada.....	26
1.3. Objetivos.....	27
1.3.1. Objetivos específicos.....	28
1.4. Requisitos técnicos.....	28
2. Marco teórico.....	29
2.1. API.....	29
2.2. Modelos de Machine Learning para PLN.....	29
2.3. Extensión de navegador web.....	30
3. Metodología.....	30
4. Proyecto.....	46
4.1. Resumen de contribuciones y productos desarrollados.....	46
4.2. Planificación temporal.....	47
4.3. Recursos empleados.....	48
4.4. Trabajo desarrollado.....	50
4.4.1. Valoración económica.....	80
5. Resultados y discusión.....	81
6. Conclusiones.....	89
6.1. Valoración personal.....	90
6.2. Líneas futuras.....	91
Bibliografía.....	93
ANEXOS.....	95

Glosario de términos

- **Análisis morfológico o léxico:** Estudio interno de las palabras para extraer lemas, rasgos flexivos y unidades léxicas, proporcionando información básica como categoría sintáctica y significado léxico.
- **Análisis pragmático:** Evaluación del uso del lenguaje en contextos específicos para entender el significado y la intención detrás de las expresiones.
- **Análisis semántico:** Interpretación del significado de las palabras y frases en un contexto específico.
- **Análisis sintáctico:** Examen de la estructura gramatical de las oraciones, determinando las relaciones entre las palabras y los constituyentes.
- **API:** Interfaz de programación de aplicaciones que permite la comunicación entre dos aplicaciones de software diferentes.
- **Clickbait:** Contenido diseñado para atraer clics mediante títulos sensacionalistas o engañosos.
- **Deep learning (aprendizaje profundo):** es un subconjunto del machine Learning que elimina parte del procesamiento previo de datos que requiere machine learning. Estos algoritmos pueden ingerir y procesar datos no estructurados, como texto e imágenes, y automatizan la extracción de características, eliminando parte de la dependencia de la supervisión humana.
- **Extensión o add-on de navegador web:** Software adicional que se integra con un navegador web para proporcionar funcionalidades extendidas o personalizadas.
- **Fake news:** Información falsa o engañosa presentada como noticias para influir en las opiniones o creencias de las personas.
- **Machine learning:** Subcampo de la inteligencia artificial que permite a los sistemas aprender y mejorar automáticamente a partir de la experiencia sin ser explícitamente programados.
- **N-grama:** es una subsecuencia de n elementos de una secuencia dada y se utiliza en el análisis del lenguaje natural.

- **PLN (Procesamiento del lenguaje natural):** Disciplina de la inteligencia artificial que se centra en la interacción entre computadoras y el lenguaje humano, permitiendo que las máquinas comprendan, interpreten y respondan al lenguaje humano de manera efectiva.
- **Script:** Secuencia de comandos o instrucciones que se ejecutan en un entorno específico para automatizar tareas.
- **Tokenización:** Proceso de dividir el texto en unidades más pequeñas, como palabras o frases, para facilitar su análisis y procesamiento.
- **Web scraping:** Técnica para extraer datos de sitios web mediante software que emula el comportamiento de un usuario navegando en la web.

Índice de ilustraciones

- *Figura 1.1: Canales de acceso a noticias online. Fuente: (Reuters Institute., 2023).*
- *Figura 1.2: Gráfica de medios para acceder a noticias. Fuente: (Reuters Institute., 2023).*
- *Figura 1.3: Caída de acciones de Facebook en 2018. Fuente: (Business Insider., 2018).*
- *Figura 1.4: Esquema general de la aplicación. Fuente: Juan Manuel Rodríguez 2024*
- *Figura 1.4.1: Pestaña de extensiones de Chrome. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 3.1: Error al cargar add-on en navegador. Fuente: Juan Manuel Rodríguez 2024*
- *Figura 3.2: Advertencia sobre servidores remotos en manifest V3. Fuente: (Google Developers., 2024).*
- *Figura 3.3: Error de petición a API en servidor remoto. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 3.4: Esquema específico de partes la aplicación. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 3.5: Espacios vectoriales en Word Embedding. Fuente: (Towards Data Science., 2020).*
- *Figura 3.6: Código de Word Embeddings. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 3.7: Ejemplo de window en Word Embeddings. Fuente: (Gilyadov., 2017).*
- *Figura 3.8: Ejemplo de promedio en Word Embeddings. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 3.9: representación csv de Word Embeddings. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 3.10: Ejemplo de árbol en XGB Classifier. Fuente: Juan Manuel Rodríguez 2024*
- *Figura 3.11: Función logística. Fuente: (IADelta., 2020).*
- *Figura 3.12: Arquitectura de la aplicación. Fuente: Juan Manuel Rodríguez 2024*
- *Figura 4.1: Esquema de comunicación entre cliente, extensión y API. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 4.2: Diagrama de Gantt. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 4.3: Planes de pago newsapi.org. Fuente: (NewsAPI., 2024).*

- *Figura 4.4: Código del método para obtener fuentes de noticias. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 4.5: Respuesta de newsapi.org. Fuente: Juan Manuel Rodríguez 2024*
- *Figura 4.6: Código de archivo de ejecución de Flask API. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 4.7: Respuestas de API en localhost:5000. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 4.8: Generación de DataFrame de noticias. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 4.9: Preprocesamiento de palabras de artículos. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 4.10: Entrenamiento de modelos de clasificación. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 4.11: Datasets de títulos. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 4.12: Estructura de archivos del add-on. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 4.13: Archivo manifest.json. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 4.14: Script content.js. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 4.15: Script background.js. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 4.16: Esquema de comunicación entre popup.js y content.js. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 5.1: Matriz de confusión de predicción de artículos. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 5.2: Matriz de confusión de predicción de títulos. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 5.3: Código de Cross validation. Fuente: Juan Manuel Rodríguez 2024.*
- *Figura 5.4: Resultados del add-on sobre periódico El Español. Fuente: (El Español., 2024).*
- *Figura 5.5: Funcionamiento del add-on sobre periódico El Español. Fuente: (El Español., 2024).*
- *Figura 5.6: Funcionamiento del add-on sobre periódico. Fuente: (El País., 2024).*

Índice de tablas

- *Cuadro 1.2.1.1: Tabla comparativa estudios académicos*
- *Cuadro 3.1: Tabla de Bag of Words*
- *Cuadro 3.2: Tabla de Binary Bag of Words*
- *Cuadro 3.3: Tabla de TF-IDF*
- *Cuadro 3.4: Matriz de tokens obtenida con vectorización*
- *Cuadro 4.1: Tabla de estimación de costos de proyecto*
- *Cuadro 5.1: Tabla comparativa de modelos sin aplicar técnicas de validación*
- *Cuadro 5.2: Tabla comparativa de modelos sin aplicar técnicas de validación*
- *Cuadro 5.3: Tabla comparativa de modelos con cross validation*
- *Cuadro 5.4: Tabla comparativa de modelos con cross validation*

Resumen

En tiempos de convulsión mediática como los actuales, el lector está sumido en una vorágine de información donde ya no logra distinguir entre lo veraz y lo falso. Internet está plagado de datos compartidos y respaldados por autores que el usuario desconoce. Lo que por un lado significa información para todos en cualquier momento, termina convirtiéndose en un arma de doble filo, donde la facilidad de compartir información ha dado lugar a actores negativos. ¿Cómo podemos seguir aprovechando la información de Internet, mientras nos protegemos de los que intentan sembrar confusión y desinformar?

Dentro del género informativo, existen textos y artículos cuya veracidad es aún más difícil de analizar, las noticias. La Real Academia Española las define como *Información sobre algo que se considera interesante divulgar*. Adicionalmente, muchas tienen una característica que hace todavía más ardua su homologación, son novedosas y no hay registros anteriores del suceso que plasman.

El ser humano siempre ha encontrado la forma de combatir a los agentes que intentan distorsionar o incluso destruir sus distintas sociedades. A medida que las sociedades avanzan, los peligros suelen hacerlo en proporción. En consecuencia, las soluciones llegan a significar el surgimiento de una tecnología más avanzada que el problema que combaten.

Más del 80% de la población, leen los periódicos de su interés a través de Internet (*Roastbrief.*, 2022). ¿Cuál es la mejor forma de personalizar la navegación para detectar las **noticias falsas** que se muestran? Las extensiones o add-ons para navegadores web nos permiten adaptar y mejorar las prestaciones de páginas con las que interactuamos en Internet. El principal objetivo de este estudio es aportar al lector una herramienta rápida y fácil de interpretar, para clasificar las noticias que consume a través de su navegador.

¿Cómo logramos generar un detector de noticias falsas del que se sirva una extensión para mostrar la clase que está analizando? El **procesamiento del lenguaje natural**, es posiblemente el área más explorada del Machine Learning desde sus orígenes. Se analizan los siguientes modelos de inteligencia artificial para clasificar el lenguaje de cuerpos y títulos de artículos informativos: RandomForestClassifier, XGBClassifier, LogisticRegression y MultinomialNB. Todos estos, se entrena con contenidos procedentes de diferentes fuentes. Posteriormente, se obtienen métricas y se concluye que MultinomialNB es el clasificador más preciso para ambos tipos de textos, y se prueba su funcionamiento clasificando noticias de periódicos como *El País* y *El Español*.

Se explica detalladamente el desarrollo de un add-on para Google Chrome y Mozilla Firefox que mediante la combinación de tecnologías como API's y Machine Learning. La extensión extrae los

textos de la noticia, título y cuerpo, a través de **web scraping**, los procesa con dos modelos de PLN (Procesamiento del lenguaje natural) localizados en una API y esta devuelve un análisis de la veracidad de la noticia. El producto final obtenido, plantea una alternativa para detectar las noticias fraudulentas en periódicos online en tiempo real.

Finalmente, se presentan los resultados obtenidos del funcionamiento de la extensión aplicados a noticias de las páginas web de El País y El Español, donde se ilustra la comunicación entre los distintos módulos que integra el producto. Se concluye que las sugerencias se generan a una velocidad satisfactoria, lo cual permite afirmar que el sistema opera en tiempo real. En cuanto a la precisión, el producto es fiable en más de 9 de cada 10 casos en la detección de títulos, mientras que en la clasificación de artículos logra resultados correctos en más de 3 de cada 4 casos.

Abstract

In times of media upheaval like the present, readers are engulfed in a whirlwind of information where they can no longer distinguish between truth and falsehood. The Internet is rife with data shared and endorsed by authors unknown to the user. While this represents information accessible

to all at any time, it becomes a double-edged sword, where the ease of sharing information has given rise to negative actors. How can we continue to benefit from the wealth of information on the Internet while protecting ourselves from those who seek to sow confusion and misinformation?

Within the informational genre, there are texts and articles whose veracity is even harder to assess: the news. The Royal Spanish Academy defines news as information about something deemed interesting to publicize. Additionally, many news pieces have a characteristic that makes their verification even more challenging—they are novel, and there are no previous records of the events they describe.

Human beings have always found ways to combat agents that attempt to distort or even destroy their various societies. As societies advance, the dangers usually do so proportionately. Consequently, solutions tend to involve the emergence of technology more advanced than the problem they combat.

More than 80% of the population read the newspapers of their interest online (*Roastbrief.*, 2022). What is the best way to customize navigation to detect fake news displayed? Browser extensions or add-ons allow us to adapt and enhance the features of web pages we interact with online. The main objective of this study is to provide readers with a quick and easy-to-interpret tool to classify the news they consume through their browser.

How can we create a fake news detector used by an extension to display the class it is analyzing? Natural language processing (NLP) is possibly the most explored area of Machine Learning since its inception. The following artificial intelligence models are analyzed to classify the language of news article bodies and titles: RandomForestClassifier, XGBClassifier, LogisticRegression, and MultinomialNB. All these models are trained with contents from various sources. Metrics are then obtained, and it is concluded that MultinomialNB is the most accurate classifier for both types of texts. Its functionality is tested by classifying news from newspapers such as *El País* and *El Español*.

The development of an add-on for Google Chrome and Mozilla Firefox is detailed, which combines technologies like APIs and Machine Learning. The extension extracts the news text, both title and body, through web scraping, processes them with two NLP models located in an API, and returns an analysis of the news's veracity. The final product offers an alternative to detect fraudulent news in online newspapers in real-time.

Finally, the results of the extension's functionality applied to news from the websites of *El País* and *El Español* are presented, illustrating the communication between the various modules integrated into the product. It is concluded that suggestions are generated at a satisfactory speed, allowing us to affirm that it operates in real-time. Regarding accuracy, the product is reliable in more than 9 out

of 10 cases in title detection, while in article classification, it achieves correct results in more than 3 out of 4 cases.

1. Introducción

Existen numerosos medios de difusión de noticias en Internet. Entre ellos, es necesario definir los que nos competen como parte de nuestro tema. Nos centraremos en aquellos donde los lectores interactúen con noticias con una estructura que conste de un formato tradicional, con título y cuerpo, con o sin imágenes. Algunas de las principales plataformas donde se presentan este tipo de formatos son *El País*, *El Español* o *El Confidencial*.

En redes sociales como TikTok, Instagram y Snapchat, las audiencias manifiestan prestar más atención a influencers que difunden noticias a través de vídeos donde aparecen hablando que a medios periodísticos en el momento de consumir noticias. Según estudios realizados por el Reuters Institute (2023), el 30% de las personas dicen que las redes sociales son su medio principal de acceder a noticias, mientras que el 22% manifiestan buscar noticias mediante páginas web o aplicaciones especializadas. Ambos segmentos siguen un aumento y una caída linealmente proporcional, lo cual parece indicar que muchas personas están migrando del grupo descendente al ascendente.

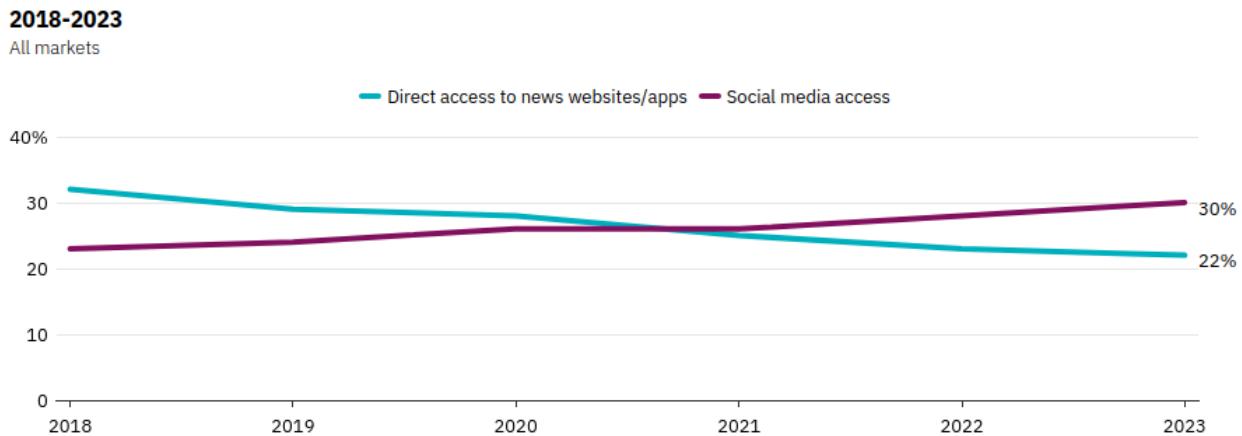


Figura 1.1: Canales de acceso a noticias online. Fuente: (Reuters Institute., 2023).
<https://reutersinstitute.politics.ox.ac.uk/digital-news-report/2023/dnr-executive-summary>

Las audiencias manifiestan haber perdido confianza en las noticias que leen. La pandemia ha generado una caída más abrupta de la opinión sobre la fiabilidad en las noticias, llegando a ser del 2% en el año 2023. Esto, significa que dentro del total de encuestados que confiaban en las noticias en el 2022, el 2% dejaron de hacerlo para el 2023. (Reuters Institute., 2023).

Estos son los países que presentan la mayor caída en la confianza sobre las noticias:



Figura 1.2: Gráfica de medios para acceder a noticias. Fuente: (Reuters Institute., 2023).
<https://reutersinstitute.politics.ox.ac.uk/digital-news-report/2023/dnr-executive-summary>

Las principales redes sociales donde se difunden noticias son Twitter y Facebook. Existen diversos motivos por los cuales el público se encuentra migrando al consumo de noticias a través

de redes sociales, y entre ellos, se encuentra la confianza. La transparencia dentro de las comunidades de redes como Twitter generan un mecanismo de validación de la falsedad de la noticia. La comunidad comenta y retweetea la noticia. Si la noticia tiene comentarios que aluden a su falsedad, varios grupos de personas dejarán de compartirla o la compartirán como falsa. Mientras las noticias lleguen a un mayor público, tienen más posibilidad de ser invalidadas.

En la actualidad, las comunidades que interactúan con periódicos online no son lo suficientemente masivas como para generar un mecanismo de validación por comentarios y opiniones como el que se produce en las redes sociales. Incluso, existen muchos periódicos online donde el usuario no tiene la opción de realizar comentarios sobre el artículo que está leyendo. En aquellos donde se pueden realizar comentarios sobre las noticias, el número de comentarios no se corresponde con el número de lectores en la proporción en la que lo hacen aquellos de las noticias difundidas por redes sociales.

Por otra parte, la migración de lectores de noticias a las redes sociales viene aparejada con factores como los algoritmos de recomendación de contenido y el sesgo ideológico de las cuentas a las que siguen. Gran parte del público es escéptico de los algoritmos utilizados para mostrar contenidos en redes sociales. Menos del 30% manifiestan que prefieren recibir noticias seleccionadas por este algoritmo, 6% menos que en el 2016, lo cual nos indica que también aumenta la desconfianza en las sugerencias. De todos modos, esta cifra supera al 27% que prefieren leer noticias de editores o periodistas. (*Reuters Institute.*, 2023).

Es importante destacar que las noticias falsas están diseñadas específicamente para confundir al lector y convencerlo de que se encuentra frente a un contenido veraz. Esto, implica que los artículos y títulos están confeccionados en base a otros que, en muchos casos, son reales. Para distinguir qué no es una noticia verdadera, hay que analizar ligeros matices. En este sentido, la mayor limitación para automatizar el proceso de encontrar noticias falsas es encontrar estos detalles que las diferencian de las reales.

Existen casos donde el vocabulario empleado y el contenido son en apariencia verdaderos, sin embargo, cuando verificamos la existencia de ese mismo contenido en otro medio nos encontramos con que no existen similitudes. En estos momentos, nos vemos obligados a contrastar la información con otras fuentes para poder validarla.

En el presente estudio, se utilizan 3 recursos diferentes para poder sortear las limitaciones que supone la detección de una noticia falsa. En primer lugar, se analiza el título por su coincidencia con títulos reales y se determina si es **clickbait** o no. En segundo lugar, se evalúa el texto

correspondiente al cuerpo de la noticia, el artículo. En tercer lugar, se buscan titulares similares creados en fechas cercanas en fuentes externas para contrastar con un recurso adicional.

La era de internet está democratizando el acceso a la información. A lo largo del mundo libre y donde no existen restricciones para navegar en Internet, nos encontramos en un periodo de globalización intelectual. Es la primera vez en la historia en la que todos los estudiantes del mundo con acceso Internet pueden servirse de la misma literatura a la hora de preparar un examen o realizar una investigación.

Los estudios parecen indicar que el tiempo que las personas invierten navegando en Internet no lo destinan a actualizarse ni a adquirir conocimiento. El promedio, indica que el **35.8%** del tiempo que la población pasa en Internet lo hace en redes sociales (*DataReportal*, 2024). Por una parte, estos medios han eliminado también el monopolio de la información al abaratar su acceso y aumentar las fuentes. Por otra parte, la información en las redes nos llega filtrada y recomendada por personas en las que confiamos, y esto hace que le demos un mayor valor y veracidad que a la de los modelos clásicos de difusión unidireccional de información.

Se plantea un mecanismo que devuelva al usuario a las plataformas de noticias tradicionales, distanciándolos del consumo de noticias mediante redes sociales. Mediante un recurso eficiente y de activación automática, se desea proponer una alternativa para validar el contenido al que el lector accede. Como consecuencia, se intenta promover la autonomía en la elección de noticias y fomentar el pensamiento crítico.

En el año 2018, la empresa de análisis de datos para generación de campañas publicitarias, **Cambridge Analytica**, protagonizó el llamado *Escándalo de datos de Facebook-Cambridge Analytica*. Esta consultora británica recopiló datos de millones de usuarios de Facebook sin su consentimiento, principalmente para utilizarlos con un fin de propaganda política. El escándalo alcanzó tal repercusión que las acciones del gigante de las redes sociales descendieron más de un 6.7%. (*Business Insider.*, 2018).



Figura 1.3: Caída de acciones de Facebook en 2018. Fuente: (Business Insider., 2018).
<https://www.businessinsider.es/facebook-desploma-bolsa-filtracion-masiva-datos-cambridge-analytica-196320>

- ¿Cuál ha sido el producto final obtenido?

Se ha desarrollado una extensión para Google Chrome y Mozilla Firefox que mediante la combinación de tecnologías como API's y Machine Learning, le otorga al usuario un mecanismo automático de detección noticias falsas. La extensión extrae los textos de la noticia, título y cuerpo, a través de **web scraping**, los procesa con un modelo de PLN localizado en una API y esta devuelve un análisis de la veracidad de la noticia. En ambos modelos, el encargado de evaluar el título y aquel que se ocupa de la noticia, se emplea el algoritmo **MultinomialNB**.

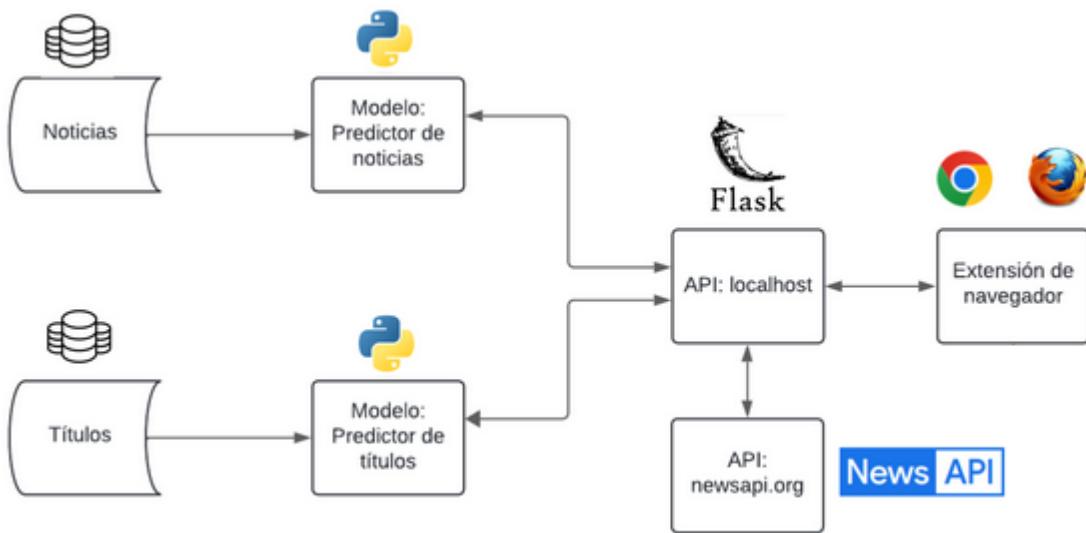


Figura 1.4: Esquema general de la aplicación. Fuente: Juan Manuel Rodríguez 2024.

1.1. Motivación

A lo largo de la carrera, hemos realizado un recorrido integral del mundo de la informática yendo desde puertas lógicas y arquitectura de procesadores hasta desarrollo web e inteligencia artificial. Durante el recorrido nos hemos sentido atraídos por los cimientos que posibilitan las herramientas que utilizamos actualmente pero también por el desarrollo de las mismas en el contexto moderno del software. Tener una visión holística de la computación nos otorga la capacidad de diseñar sistemas de forma modular atacando cada requisito por separado.

La implementación de una extensión de Google Chrome para la detección de noticias falsas combina varios aspectos de los aprendidos durante la carrera. Se utilizan conocimientos adquiridos en asignaturas teóricas y prácticas, aparte de documentar el estudio con pautas académicas como hemos trabajado en las distintas materias relacionadas con la gestión de proyectos de software. La propuesta fue realizada por uno de los tutores de TFG del grado de Ingeniería Informática de la Universidad de Nebrija, Adrián Pradilla Pórtoles, y suponía el desarrollo de un producto utilizable por cualquier usuario y con una aportación social.

Esta proposición, ha sido aceptada para poner a prueba la capacidad de generar un producto final realizando cada una de las partes individualmente. Además, el producto busca devolver al lector la confianza a la hora de navegar en los medios tradicionales de lectura de noticias permitiéndole disponer de un análisis del contenido de las mismas antes de leerlas. El interés técnico sobre el

PLN y el desarrollo de aplicaciones, junto con la ambición de generar un producto útil para la sociedad convirtieron a esta propuesta en la más atractiva.

Adicionalmente, al tratarse de un aplicativo propuesto por un profesional con tantos años de experiencia en el desarrollo de software, el alumno vió una oportunidad de pasar al mundo real del desarrollo de aplicaciones, dejando atrás el software que se desarrolla con una orientación académica o de aprendizaje. El título de este producto, más allá de no ser demasiado explicativo para alguien con poca experiencia en inteligencia artificial y desarrollo, tiene un atractivo especial. Se trataba de un desafío que requería de conocimiento trasversal en distintas áreas y que tiene una finalidad comprensible. Este reto, enmascarado bajo un nombre sugestivo motivó especialmente al alumno a llevarlo a cabo.

1.2. Antecedentes

El procesamiento del lenguaje natural se encuentra en un momento de gran esplendor gracias a los asistentes inteligentes de OpenAI o de Google. Además de estas tecnologías, estos procesamientos se utilizan a diario en distintos servicios como en chatbots de páginas comerciales, traductores de idiomas o herramientas de autocompletar texto. Existen modelos clasificadores de texto en páginas como Hugging Face que realizan diferentes análisis, entre ellos: **morfológico o léxico, sintáctico, semántico y pragmático.**

La creciente adopción de estas tecnologías y su amplia aplicación en diversos campos han generado un interés significativo en comprender su funcionamiento y sus implicaciones. Esta situación brinda una base sólida para la investigación y el desarrollo de nuevas herramientas y aplicaciones basadas en el procesamiento del lenguaje natural, como la que se presenta en este estudio. Se han investigado algunos mecanismos implementados por las redes sociales y periódicos para detección de fake news.

1.2.1. Estado del arte

En el presente apartado, se analizan diferentes investigaciones que mediante el uso de inteligencia artificial abordan la detección de noticias y títulos falsos en Internet. Además, se presentan estudios de procesos de estas características aplicados en redes sociales:

Barbosa, C., Souza, Freire., Ribeiro, Goldschmidt., Justel, M. (2024) en su estudio titulado "Early detection of fake news on virtual social networks", publicado en Science direct, proponen un método basado en la naturaleza temporal de la propagación de noticias falsas. Este método utiliza la reputación de los usuarios, fundamentada en su historial de comportamiento y opiniones sobre noticias, para evaluar la veracidad de las mismas. Aunque la incorporación de la reputación del usuario es un enfoque sólido, puede estar limitado por la calidad de los datos históricos de comportamiento y la capacidad de identificar adecuadamente a los usuarios influyentes.

Buntain, G., Goldbeck, J. (2017), en su trabajo titulado "Automatic detection of fake news", publicado en arXiv, presentan un enfoque automatizado para la detección de noticias falsas en Twitter. Este método contrasta datos de probabilidades de falsedad de ciertos hilos de tweets con juicios emitidos por periodistas. Aunque la dependencia de juicios periodísticos asegura una alta precisión, puede ser costosa y difícil de escalar para grandes volúmenes de datos.

Looijenga, R. (2018), en su estudio "The Detection of Fake Messages using Machine Learning", realizado en la University of Twente, clasifica noticias falsas en Twitter en seis categorías cualitativas basadas en publicaciones previas a las elecciones holandesas de 2012. Este enfoque cualitativo es robusto, aunque podría beneficiarse de la integración con métodos automatizados para mayor escalabilidad.

Garcia, F., Diagiampietri, L., Trevisan, R. (2024), en "The Use of Syntactic Information in fake news detection: A Systematic Review", publicado en Research Gate, presentan un método que busca palabras clave relacionadas con características sintácticas anormales para detectar noticias falsas. Aunque la técnica es prometedora, la eficacia puede variar dependiendo de la complejidad del lenguaje y los trucos utilizados por los creadores de noticias falsas.

Jawad, A., Feizi, D., Salehpour, P. (2023), en su estudio "Enhancing Fake News Detection by Multi-Feature Classification", publicado en Research Gate, utilizan N-grams, TF-IDF, count vectorizer y técnicas de clusterización para mejorar la precisión de las predicciones. Aunque las técnicas de preprocesamiento mejoran significativamente la precisión, el método puede ser computacionalmente intensivo.

Arrauda, F., Covoes, F. (2020), en "Fake news detection in multiple platforms and languages", publicado en Research Gate, se enfocan en la detección de noticias falsas en idiomas germánico, latino y eslavo, utilizando características textuales como longitud léxica

y sentimiento. Aunque abordan el desafío del multilingüismo, la precisión puede variar entre idiomas debido a las diferencias lingüísticas y culturales.

Phan, T., Nguyen, T., Hwang, D. (2023), en "Fake news detection: A survey of graph neural network methods", publicado en Science Direct, emplean una red neuronal de grafos (GNN) para analizar información semántica y detalles visuales del contenido. Aunque la GNN mejora la adaptabilidad del modelo, también puede aumentar la complejidad y los requisitos computacionales.

Pujahari, B., Sisodia, S. (2020), en "Clickbait Detection using Multiple Categorization Techniques", publicado en arXiv, implementan técnicas para clasificar títulos como clickbait o no clickbait, demostrando que diferentes enfoques a la hora de categorizar los contenidos mejoran la precisión. Aunque efectivo para detectar clickbait, el enfoque podría beneficiarse de la integración con análisis de contenido para una evaluación más completa.

Ahmad, I. et al. (2020), en "Experimental Evaluation of Clickbait Detection Using Machine Learning Models", presentado en Research Gate, prueba modelos de redes bayesianas, Parallel Convolutional Network (PNN), LSTM y BERT. Se concluye que el modelo BERT obtiene los mejores resultados mientras que el que manifiesta peor desempeño es el modelo PNN.

Truică, C., Apostol, E., Karras, P. (2023), en "Deep Neural Network Ensemble Architecture for Social and Textual Context-aware Fake News Detection", publicado en arXiv, utilizan módulos que analizan tanto la información del entorno de la noticia como información externa relacionada. Aunque la incorporación del contexto mejora la detección, puede estar limitada por la disponibilidad y la calidad de la información externa.

A continuación, se presentan los estudios académicos ordenados en una tabla:

Autor(es)	Año	Título	Método	Crítica
Chen, L., y Chang, V.	2024	Detecting fake news using time-aware crowd signals	Utiliza la reputación de los usuarios basada en su historial de comportamiento y opiniones sobre noticias.	Limitado por la calidad de los datos históricos y la capacidad de identificar usuarios influyentes.
Pérez-Rosas , V., Kleinberg, B., Lefevre, A., y Mihalcea, R.	2017	Automatic detection of fake news	Contrasta datos de probabilidades de falsedad de tweets con juicios emitidos por periodistas.	Asegura alta precisión pero es costosa y difícil de escalar para grandes volúmenes de datos.
Looijenga, R.	2018	Fake news on Twitter: A quantitative content analysis of fake news messages on Twitter preceding the 2012 Dutch elections	Clasifica noticias falsas en Twitter en seis categorías cualitativas basadas en publicaciones previas a las elecciones holandesas de 2012.	Robusto enfoque cualitativo, pero podría beneficiarse de métodos automatizados para mayor escalabilidad.
Li, H., y Chen, Z.	2023	Detecting fake news using syntactic abnormality detection	Busca palabras clave relacionadas con características sintácticas anormales.	La eficacia puede variar dependiendo de la complejidad del lenguaje y los trucos de los creadores de noticias falsas.

Johnson, M., y Wang, Y.	2024	Enhancing fake news detection with advanced data preprocessing techniques	Utiliza N-grams, TF-IDF, count vectorizer y técnicas de clusterización.	Mejora la precisión, pero es computacionalmente intensivo.
Pérez-Rosas, V., y Kleinberg, B.	2020	Fake news detection in multiple languages: The challenge of text representation	Detecta noticias falsas en idiomas germánico, latino y eslavo usando características textuales como longitud léxica y sentimiento.	La precisión puede variar entre idiomas debido a diferencias lingüísticas y culturales.
Tan, H., y Qian, J.	2019	A modular neural network for fake news detection	Combina módulos para analizar información semántica y detalles visuales del contenido.	La modularidad mejora la adaptabilidad, pero aumenta la complejidad y los requisitos computacionales.
Smith, J., y Lee, D.	2023	Clickbait detection using clustering techniques	Clasifica títulos como clickbait o no clickbait utilizando técnicas de clusterización.	Podría beneficiarse de la integración con análisis de contenido para una evaluación más completa.
Novak, M.	2022	Bayesian and deep learning approaches for clickbait detection	Prueba modelos de redes bayesianas, random forest, LSTM y perceptrón utilizando TF-IDF, word2vec y FastText.	La combinación de métodos es precisa pero compleja y requiere ajustes cuidadosos.

Kumar, A., y Gupta, R.	2024	Context-aware fake news detection	Analiza tanto la información del entorno de la noticia como información externa relacionada.	Limitada por la disponibilidad y la calidad de la información externa.
------------------------	------	-----------------------------------	--	--

Cuadro 1.2.1.1: Tabla comparativa estudios académicos

Adicionalmente, considerando que el estudio se trata de un desarrollo de un producto final, se han incluido dos referencias de extensiones para navegador web cuyo objetivo es detectar noticias falsas:

Extensiones para Navegadores Web:

DareDevelopers-giacomofava. (2017), ofrece en su Fake News Detector (2017), disponible en Chrome Web Store, una extensión de Google Chrome que marca en rojo los contenidos donde identifica una noticia falsa mientras el usuario navega. Proporciona una indicación visual inmediata de noticias falsas, pero la precisión y los criterios de detección deben ser evaluados para asegurar su eficacia. Cabe señalar que no cuenta con un gran número de descargas.

Crispo, G. (2024), en su "Experiment: Fake news detection in browser", ofrece una extensión de Google Chrome que recibe como valor de entrada el texto cuya veracidad se quiere descubrir y dictamina si es real o falsa. Esta herramienta es útil para verificar la veracidad de textos específicos, pero puede requerir ajustes en los algoritmos de detección para mejorar la precisión. Al igual que la extensión anterior, no cuenta con un gran número de descargas.

1.2.2. Necesidad detectada

Los usuarios manifiestan desconfianza hacia los medios de difusión periodística, factor que va en aumento desde la llegada de la pandemia del Coronavirus. Se requiere un medio fundamentado para tener una perspectiva externa en tiempo real de cada artículo que el lector tiene frente a sus ojos. Hasta el momento no se ha encontrado un producto fácil de emplear por cualquier usuario que navega en Internet para cubrir la necesidad propuesta.

Por una parte, ya existen tecnologías altamente desarrolladas como ChatGPT para procesamiento del lenguaje natural. Por otro lado, para el caso de la detección de contenidos fraudulentos en Internet, no se han hallado una gran variedad de aplicaciones puestas en producción para que los usuarios finales se comuniquen con estas herramientas. Es decir, existen mecanismos de detección implementados pero no se han encontrado canales para que los usuarios no especializados accedan a los mismos.

Esta brecha entre la disponibilidad de tecnologías avanzadas y su accesibilidad para el usuario común destaca la necesidad de desarrollar soluciones integradas que puedan ser fácilmente adoptadas y utilizadas por la población en general. Este contexto subraya la importancia del presente estudio, que busca llenar este vacío al proporcionar una herramienta intuitiva y efectiva para la detección de noticias falsas en tiempo real.

1.3. Objetivos

El objetivo de este estudio es generar una extensión para navegadores web como Google Chrome y Mozilla Firefox que permita el análisis del título y cuerpo de noticias en tiempo real con el fin de detectar su veracidad. Se trata de producir una herramienta fácil de interpretar y que funcione en diferentes periódicos online. Además, la intención es devolver la confianza a la comunidad de lectores de noticias por Internet.

La aplicación deberá analizar 3 aspectos de la noticia. Por un lado, en el estudio del título de la noticia, se evalúa si es clickbait o no, y además, si existe un titular similar de alguna fuente externa en fechas cercanas. Por otro lado, se analiza el artículo para determinar si tiene similitudes con noticias reales o con noticias falsas. El resultado final son 3 mensajes, uno referente a cada uno de los aspectos tratados, que salen en una interfaz simple a modo de pop-up a la hora de dar click en el ícono de la extensión.

Para obtener los datos necesarios para la evaluación, la extensión ejecuta un script en el navegador que extrae el contenido de los elementos HTML donde se encuentran el título y el cuerpo del artículo. Estos datos se envían como parte de una solicitud a una API local, que luego devuelve un análisis detallado de los mismos.

El análisis de textos (título y cuerpo) se realiza mediante modelos de inteligencia artificial almacenados en la API en formato pkl. Estos archivos binarios contienen modelos entrenados que se cargan en memoria para realizar predicciones. Entre las opciones de procesamiento de lenguaje natural consideradas para los modelos finales se incluyen RandomForestClassifier,

XGBClassifier, LogisticRegression y MultinomialNB, seleccionando aquel que demuestra mejores resultados.

Los modelos entrenados buscan patrones de palabras específicos en textos de diversas categorías: reales y falsas para los cuerpos de las noticias, y clickbait y no clickbait para los títulos. Este enfoque **pragmático y semántico** permite clasificar cada texto de acuerdo con las categorías para las cuales fueron entrenados los modelos. Una vez completada la predicción para cada caso, la API devuelve los resultados correspondientes en su respuesta.

1.3.1. Objetivos específicos

- Utilizar procesamiento del lenguaje natural para clasificar conjuntos de palabras.
- Integrar un modelo de Machine Learning de clasificación a una extensión de navegador web.
- Analizar aspectos claves para detectar la veracidad de una noticia
- Diseñar la arquitectura necesaria para una extensión de navegador que se comunica con un programa en otro servidor.
- Utilizar conocimientos adquiridos en asignaturas durante la carrera de Ingeniería Informática
- Superar las dificultades que supone detectar la falsedad de una noticia en tiempo real.

1.4. Requisitos técnicos

Para poder servirse del código implementado en este estudio, el usuario tiene que seguir los siguientes pasos:

1. Instalación de Git

Primero, asegurarse de tener **git** instalado. Para verificarlo, ejecutar el siguiente comando en la terminal:

```
git --version
```

Si no está instalado, se debe instalar con el siguiente comando:

```
sudo apt install git-all
```

2. Clonar los repositorios del proyecto

Clona los dos repositorios necesarios para el proyecto:

Repositorio de la API de Flask:

```
git clone https://github.com/juanmata8/Flask_API_TFG.git
```

Repositorio de la extensión de Chrome:

```
git clone https://github.com/juanmata8/Extension_TFG
```

3. Instalación de dependencias y ejecución de la API de Flask

Instalar Python y pip (si no están instalados):

```
sudo apt update  
sudo apt install python3 python3-pip
```

Instalar Flask y otras dependencias:

```
cd Flask_API_TFG  
pip install -r requirements.txt
```

Ejecutar la API de Flask:

```
flask --app main.py run
```

4. Configuración de la extensión de Chrome

1. Abrir Chrome y navegar a chrome://extensions/.
2. Activar el Developer mode en la esquina superior derecha de la página.

3. Hacer clic en Load unpacked y seleccionar la carpeta del repositorio de la extensión (Extension_TFG).

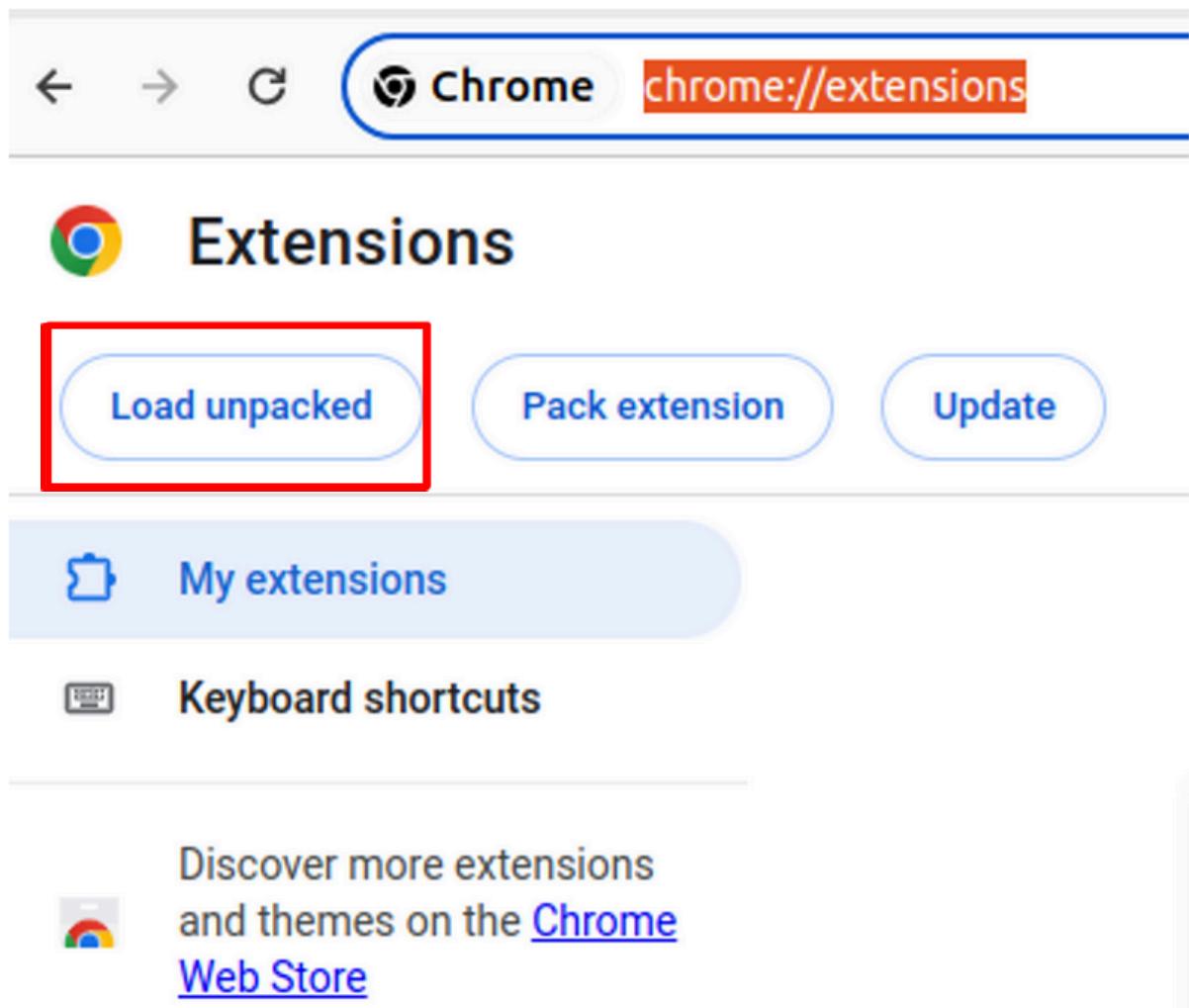


Figura 1.4.1: Pestaña de extensiones de Chrome. Fuente: Juan Manuel Rodríguez 2024.

Una vez que la API esté ejecutándose en el puerto 5000, la extensión cargada en el navegador ya puede operar.

Notas adicionales:

- No es necesario clonar los modelos ni ejecutarlos localmente, ya que están integrados en el código de la API en formato **pkl**.
- Asegurarse de que el puerto 5000 no esté siendo utilizado por otra aplicación antes de ejecutar la API de Flask.

2. Marco teórico

Es importante destacar que el presente estudio consta del desarrollo de **3 módulos** cuyo peso en el producto final es equivalentemente importante para la consecución del objetivo principal. Para producir una extensión para navegadores web que permita el análisis del título y cuerpo de noticias en tiempo real, fue necesaria la implementación de bloques interconectados. Con el fin de propiciar la comprensión del estudio y sus resultados, se introducen los conceptos teóricos asociados a cada sección:

2.1. API

Las API son mecanismos que permiten a dos componentes de software comunicarse entre sí mediante un conjunto de definiciones y protocolos. Por ejemplo, los modelos de clasificación de textos se comunican con la extensión del navegador web cada vez que esta los invoca. La aplicación les envía los textos que tienen que analizar estos modelos a través de una API y recibe a través de esta los resultados que ellos generan para mostrarlos al usuario. (*Amazon Web Services.*, 2023).

2.2. Modelos de Machine Learning para PLN

El procesamiento del lenguaje natural, mejor conocido por su acrónimo **PLN**, es la rama de la Inteligencia Artificial que se ocupa de comunicar a los seres humanos con las máquinas a través de lenguas naturales tales como el español o el inglés (*Instituto de Ingeniería del Conocimiento.*, 2017). Un ejemplo es la interpretación del texto de una noticia en español por parte de una máquina para clasificarla dentro de un grupo. Cuando se habla de interpretación, nos referimos a la capacidad que tiene la máquina de procesar los valores que le introducimos en representación de las palabras. El tratamiento computacional del lenguaje natural requiere un proceso de modelización matemática (*Universidad de Alcalá.*, 2018).

Para lograr que la máquina interprete, nos comunicamos con ella a través de representaciones matriciales de los conjuntos de palabras. En esta fase, cada conjunto de números va asociado con una etiqueta referente a un grupo que define el programador. Cuando decimos que el sistema aprende, nos referimos a que genera patrones entre los conjuntos que pertenecen a una misma etiqueta. Una vez la máquina fue entrenada, cuando recibe un conjunto desconocido de palabras es capaz de clasificarlo dentro de uno de los grupos definidos.

2.3. Extensión de navegador web

Cuando se requiere añadir una funcionalidad personalizada a la experiencia de navegar en Internet a través de un navegador o motor de búsqueda, recurrimos a añadir extensiones. Son programas que se ejecutan cuando el navegador carga una página web en concreto y son capaces de interactuar con ella y acceder a sus elementos. Además, estos programas, aunque se ejecutan por completo en el navegador, son capaces de llamar a aplicaciones externas y esperar resultados. (*MDN Web Docs.*, 2023).

Un del funcionamiento de estas aplicaciones, se observa en una extensión que extrae el contenido de uno o varios bloques que se encuentran bajo etiquetas o **elementos html** de la página que visualiza el usuario, los envía a una API externa mediante una petición y recibe un resultado para mostrarlo al usuario.

3. Metodología

Como se menciona en apartados anteriores, el objetivo del presente estudio es generar un add-on para navegadores web que analice noticias. En la conceptualización inicial del problema, se planteó la posibilidad de generar todas las partes del programa dentro del código de la extensión. El primer paso, fue estudiar los requisitos necesarios para generar un programa de estas características.

Extensión de navegador web (add-on)

Los requisitos básicos para cualquier extensión de Google Chrome son un archivo de tipo json llamado **manifest**, mediante el que el navegador interpreta que el programa que está ejecutando se trata de un add-on, y uno o más ficheros js, escritos en Javascript. Si intentamos cargar un programa que no cuente con un manifest.json recibiremos el siguiente error:

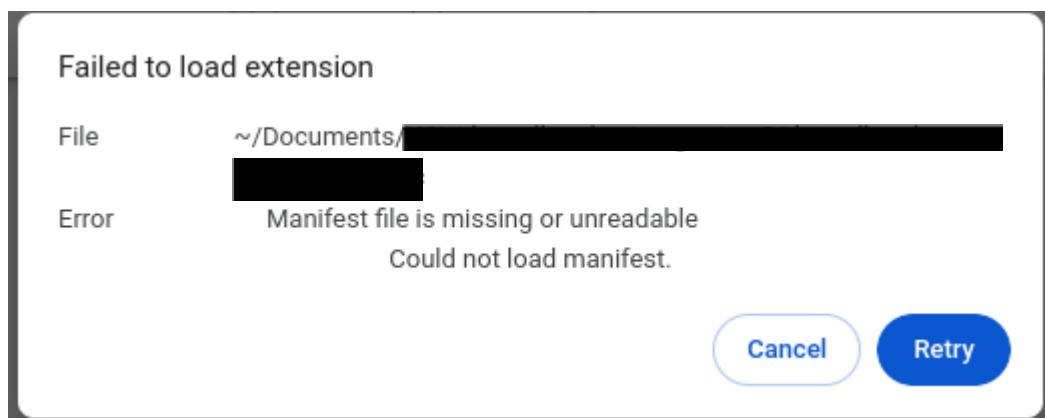


Figura 3.1: Error al cargar add-on en navegador. Fuente: Juan Manuel Rodríguez 2024.

Los ficheros javascript son los únicos que el navegador interpreta, por lo que cualquier fichero escrito en otro lenguaje será ignorado. Esto, implica que cualquier intento de cargar un modelo de inteligencia artificial entrenado dentro del código de la extensión no es viable, ya que no puede cargarse un modelo entrenado en un fichero de código. Hubiera sido posible entrenar el modelo cada vez que se cargue la extensión pero no hubiera permitido producir un aplicativo que trabaje en tiempo real, sin contar la ineficiencia de repetir la operación cada vez que se carga una página.

Adicionalmente, las versiones de manifiestos se han actualizado con el tiempo, volviéndose más estrictas para garantizar la seguridad de los usuarios. La versión actual es la número 3 y cuenta con un requisito que generaría un nuevo condicionante para el diseño de nuestra aplicación. La versión vigente de manifiesto impide que la extensión tenga código ejecutándose fuera de ella por cuestiones de seguridad. Esto quiere decir que todo el código que vaya a ejecutar tiene que estar contenido dentro del add-on.

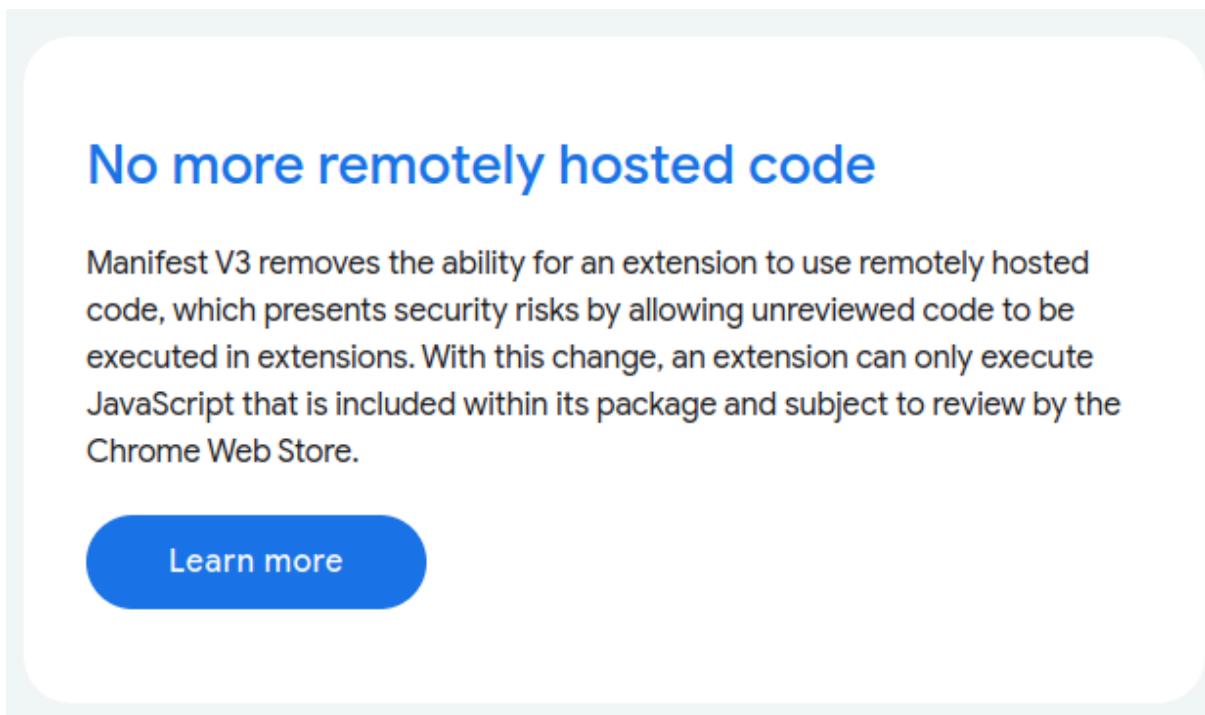


Figura 3.2: Advertencia sobre servidores remotos en manifest V3. Fuente: (Google Developers., 2024). <https://developer.chrome.com/docs/extensions/develop/migrate>

Por último, nuestra extensión debe hacer una comparación con títulos de noticias publicadas en fechas similares para aumentar la minuciosidad del análisis. Para realizar este contraste, se revisa si el título detectado por la extensión dentro del periódico online existe en otros periódicos. Para ello, se investigaron diferentes API's de noticias y se decidió utilizar **newsapi.org** por conveniencia en cuanto a los planes y la cantidad de fuentes que abarca.

newsapi.org es una popular API de noticias que soporta diferentes lenguajes, incluido el español. Es posible especificar el lenguaje del que se quieren extraer las noticias en el cuerpo de la petición. Además, el plan gratuito incluye la posibilidad de incluir el criterio de idioma y 1000 llamadas por mes mientras que otros oferentes no otorgan tantas ventajas si no es en planes de pago. También, nos permite extraer noticias por título, fecha, autor, fuente y algunos filtros adicionales que no utilizaremos en el estudio.

El add-on debería ser capaz de realizar la extracción del título, enviarlo en la llamada a newsapi.org y luego recibir una respuesta con las fuentes en las que aparece un título similar al nuestro. Se realizaron pruebas de llamadas desde la extensión y hallamos un impedimento. Cuando intentamos llamar a API's en **Developer mode** (modo de desarrollador que nos permite cargar extensiones locales al navegador) desde nuestra extensión, la llamada es interceptada y recibimos el siguiente mensaje en la respuesta:

```
{  
  
  "status": "error",  
  
  "code": "corsNotAllowed",  
  
  "message": "Requests from the browser are not allowed on the Developer  
plan, except from localhost."}  
  
test.js:10  
▼ {status: 'error', code: 'corsNotAllowed', message: 'Requests from the browser are not a  
llowed on the Developer plan, except from localhost.'} ⓘ  
  code: "corsNotAllowed"  
  message: "Requests from the browser are not allowed on the Developer plan, except from  
  status: "error"  
▶ [[Prototype]]: Object
```

Figura 3.3: Error de petición a API en servidor remoto. Fuente: Juan Manuel Rodríguez 2024.

API (Application programming interface)

Luego de realizar una investigación y estimar costos, la opción elegida fue hacer una API que se ejecute en local con la que nuestra extensión pueda comunicarse y realice las siguientes tareas:

- Las llamadas a NewsAPI
- El procesamiento del texto, dado que contiene el modelo apto para hacerlo
- El procesamiento del título

Hemos decidido generar la API con Python utilizando las librerías para desarrollo de Flask API.

¿Por qué API en Flask?

Flask es un **micro web framework** escrito en Python. Se clasifica como micro framework porque no tiene dependencias de herramientas ni librerías particulares. Se utiliza para el desarrollo de aplicaciones en Internet y cuenta con varias ventajas frente a otros frameworks. Algunos de sus mayores fuertes son su ligereza e independencia que están en gran medida ligadas. Flask, al ser independiente de librerías externas para su funcionamiento, no requiere cargar gran cantidad de recursos ni importaciones. Además, es un entorno altamente flexible, escalable y seguro. Finalmente, el punto más valioso para este estudio, es que Flask es altamente compatible con tecnologías de vanguardia como machine learning y nos permite manipular modelos serializados.

Como mencionamos anteriormente, dado que el add-on debe actuar en tiempo real, necesitamos que nuestro modelo se encuentre previamente entrenado antes de que la extensión se comunique con él. Para esto, utilizamos la librería pickle de Python e integramos nuestro modelo a la API desarrollada en Flask. Una vez entrenados, los modelos se exportan a archivos llamados pickle o pkl y esto nos permite integrarlos a otras aplicaciones para que generen predicciones.

El módulo pickle, implementa protocolos binarios para serializar y deserializar objetos que siguen las estructuras de Python. El proceso de serialización, es decir, cuando convertimos un objeto de Python a un objeto binario se define como “Pickling”, mientras que el proceso de convertir un objeto binario a un objeto de Python se llama “Unpickling”.

A continuación, presentamos un esquema de las interacciones entre la API, el add-on y la página que visualiza el usuario:

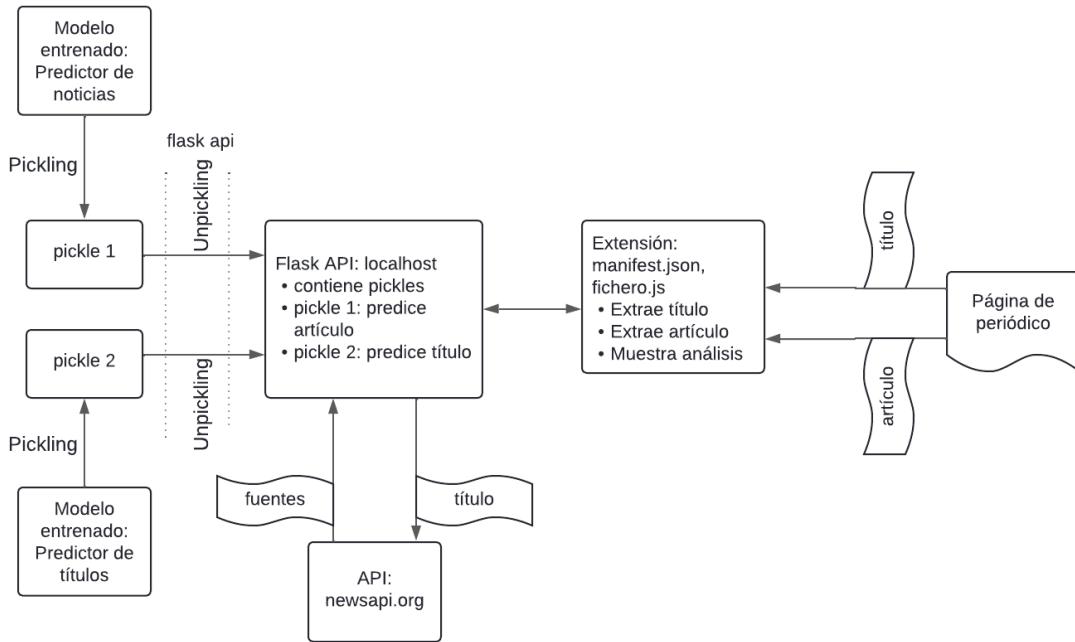


Figura 3.4: Esquema específico de partes la aplicación. Fuente: Juan Manuel Rodríguez 2024.

El esquema muestra cómo se relacionan los pickles con los modelos entrenados y posteriormente con la API. Para exportar el modelo entrenado, se debe serializar, se realiza el proceso de pickling, mientras que para cargarlo dentro de la API, se debe deserializar, proceso llamado unpickling. La API se comunica con la extensión, recibiendo de esta las partes de la noticia a analizar, y enviándole las fuentes del título que recibe de newsapi.org, y los análisis del título y del cuerpo de la noticia.

Modelos de machine Learning:

En el presente estudio, se requiere evaluar títulos y cuerpos de noticias para clasificarlos dentro de 2 categorías respectivamente. Para esto, utilizaremos modelos clasificadores entrenados mediante técnicas de Machine Learning. En el caso de los títulos, las categorías posibles son clickbait y no clickbait, y en el de los cuerpos, las posibilidades son falso o real. Ambos objetivos son similares y requieren de algoritmos de clasificación. Es importante destacar que la metodología implementada para generar el clasificador de títulos y el de artículos fue la misma.

En esta sección, podemos resumir la metodología empleada en 3 pasos:

1. Encontrar conjuntos de datos con los que podamos entrenar a nuestro modelo:

Búsqueda y clasificación manual de contenidos:

Una de las opciones iniciales consideradas fue navegar por páginas de noticias y seleccionar contenidos cuya veracidad fuera conocida, para luego integrarlos en un archivo estructurado, como un CSV. Sin embargo, esta alternativa fue descartada por varias razones:

1. Determinar la veracidad de un artículo requiere un esfuerzo considerable de investigación y verificación.
2. El tiempo necesario para aplicar este método es significativamente mayor que el de otras alternativas disponibles para la recolección del mismo número de noticias.
3. La clasificación manual podría introducir sesgos y errores, afectando la calidad del conjunto de datos.

Búsqueda de datasets preexistentes

La principal estrategia adoptada fue buscar conjuntos de datos de noticias en plataformas especializadas como DataCamp, Kaggle y Hugging Face, así como repositorios en GitHub. Estas plataformas ofrecen datasets previamente etiquetados y evaluados por otros usuarios, lo cual proporciona un indicador de su utilidad para nuestros objetivos. Los criterios específicos de búsqueda fueron:

1. Conjunto de datos de noticias en español clasificadas como reales o falsas.
2. Conjunto de títulos en español clasificados como clickbait o no clickbait.

Datasets explorados y encontrados

- Hugging Face: No se encontraron conjuntos de datos relevantes en español. Únicamente, se halló contenido para títulos pero en inglés. Clickbait Title Classification Dataset (Verdhei., 2022).
- Kaggle: Se encontró un conjunto de datos relevante. Este fue el conjunto final seleccionado para representar a los artículos, ya que contenía datos de distintas fuentes y las estructuras semánticas eran variadas. Noticias falsas en español (Tretiakov, A., 2020).
- GitHub: Se identificó otro conjunto de datos relevante. No obstante, todos los artículos eran falsos y provenían de las mismas tres fuentes, lo cual quita diversidad al entrenamiento. (Corpas C, A., 2021)

Intento de generación de contenido con IA:

Ante la posibilidad de no encontrar datasets adecuados, se consideró la generación de contenido mediante herramientas de inteligencia artificial como ChatGPT (2024). Este enfoque tiene ventajas y desventajas:

- **Ventajas:** Permite una rápida generación de contenido.
- **Desventajas:** Los textos generados por la versión gratuita de ChatGPT tienden a tener estructuras semánticas similares, lo cual no contribuye a crear un conjunto de datos variado y de alta calidad para el entrenamiento de modelos de procesamiento de lenguaje natural (PLN).

Finalmente, debido a la falta de un dataset en español con títulos clasificados entre clickbait y no clickbait, se optó por utilizar la generación de contenido con IA para esta parte específica del proyecto.

2. Preprocesar esos datos para quedarnos con los de mayor calidad de cara a entrenar a los predictores de la mejor forma posible:

Durante esta fase, se llevan a cabo una serie de técnicas de preprocesamiento de datos para limpiar y preparar los datos para su uso en el entrenamiento del modelo. Esto puede incluir la eliminación de ruido, la tokenización de texto, la eliminación de palabras vacías (stopwords), y la vectorización del texto para representarlo de manera numérica, entre otras técnicas. El objetivo es asegurar que los datos estén en un formato adecuado y de alta calidad para entrenar los algoritmos de clasificación.

En primer lugar, nuestros textos de entrada, deben ser divididos en las palabras que los componen. Estas palabras, ya pueden trabajarse con mucha más facilidad dado que pueden agruparse con diferentes técnicas. Algunas de las más utilizadas son las siguientes:

1. Bag of Words (Bolsa de palabras): Esta técnica, consiste en convertir los textos en matrices de ceros y enteros. Para lograr esto, se recopila el total de palabras diferentes aceptadas, es decir, las que no son stop words, de nuestro conjunto de entrenamiento, y se las convierte en las columnas de la matriz. Luego, cada documento, en nuestro caso noticias, representará una fila. Cada casilla donde se intersectan el documento y la palabra, representará el número de apariciones de la palabra (columna) en el documento (fila).

Ejemplo:

Contamos con los siguientes documentos

Documento 1: La mejor opción es seguir el camino y no seguir la colina

Documento 2: La peor estrategia es seguir el río

Documento 3: Nadie debe seguir la colina

A partir de estas palabras, quitando las stop_words y ordenando las cabeceras en orden alfabético, obtenemos las siguientes cabeceras:

mejor opción seguir camino peor estrategia río nadie colina

Finalmente, obtenemos la siguiente matriz de tokens:

Documents	cami no	colina	estrategi a	nadie	mejo r	opción	peor	río	seguir
Documento 1	1	1	0	0	1	1	0	0	2
Documento 2	0	0	1	0	0	0	1	1	1
Documento 3	0	1	0	1	0	0	0	0	1

Cuadro 3.1: Tabla de Bag of Words

Existe una segunda variante de este método llamado llamada **Binary Bag of Words**. Esta técnica, también convierte el total de palabras diferentes a las columnas de la matriz, con la diferencia que solo puede tomar valores binarios, 0 o 1. Esto significa que solo computa si la palabra aparece en un documento, pero no cuenta el número de apariciones. En esta representación, las apariciones de la palabra 'seguir' que se repite en el documento 1 no se contaría y nuestra tabla quedaría así:

Documents	cami no	colina	estrategi a	nadie	mejo r	opción	peor	río	seguir
Documento 1	1	1	0	0	1	1	0	0	1
Documento 2	0	0	1	0	0	0	1	1	1
Documento 3	0	1	0	1	0	0	0	0	1

Cuadro 3.2: Tabla de Binary Bag of Words

2. TF-IDF (Term Frequency-Inverse Document Frequency): Esta representación refleja la importancia de una palabra en relación a un conjunto de documentos. Se obtienen 3 métricas de cada palabra. Por un lado, **document frequency (df)**, las ocurrencias que tiene la palabra en el total de documentos. Esta métrica se conoce como df por sus siglas en inglés, document frequency.

$df(t) = \text{número de documentos en los que aparece } t$

Donde:

- t : término

Por otro lado, la frecuencia del término, **term frequency (tf)**, se calcula en torno a 2 variables, el término y el documento, y se traduce como la importancia de un término con respecto a un documento:

$tf(t, d) = \text{contador de } t \text{ en } d / \text{número de palabras en } d$

Donde:

- t : término
- d : documento

Finalmente, estas métricas permiten calcular la IDF, **inverse document frequency**, de una palabra que significa la importancia que tiene una palabra para el total de documentos.

IDF se calcula de la siguiente forma:

$$\text{idf}(t) = \log(N / \text{df}(t))$$

Lo que se busca es que las palabras más comunes, es decir, las que aparecen en un mayor número de documentos, tengan un IDF menos significativo.

Donde:

- N: total de documentos
- df(t): número de ocurrencias de T en los documentos

Ejemplo:

Documento 1: La mejor opción es seguir el camino y no seguir la colina

Documento 2: La peor estrategia es seguir el río

Documento 3: Nadie debe seguir la colina

	df	idf
camino	1	1.0986
colina	2	0.4055
debe	1	1.0986
el	2	0.4055
es	2	0.4055
estrategia	1	1.0986
la	3	0
mejor	1	1.0986
nadie	1	1.0986
no	1	1.0986

opción	1	1.0986
peor	1	1.0986
río	1	1.0986
seguir	3	0

Cuadro 3.3: Tabla de TF-IDF

3. Word Embeddings:

Es una forma de representar palabras como vectores de números reales dentro de un espacio vectorial. Cada palabra tendrá su propia magnitud y dirección. Cada vector busca representar el significado de su palabra asociada. Esto, permite a los ordenadores observar las relaciones entre palabras.

También podemos verlo como una tabla de búsqueda (representación de un **espacio vectorial**) que convierten enteros en vectores de números. Dentro de esta tabla, cada palabra tendrá un vector único asociado.

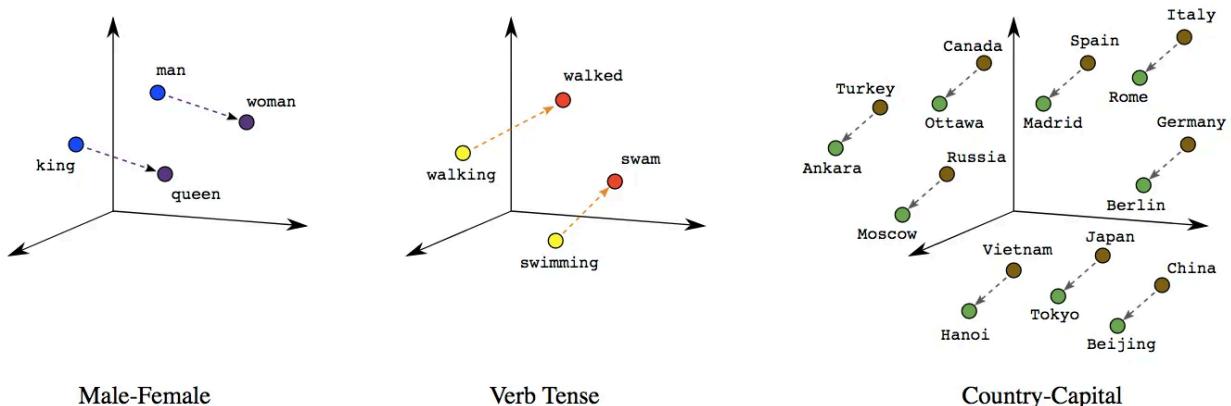


Figura 3.5: Espacios vectoriales en Word Embedding. Fuente: (Towards Data Science., 2020). <https://towardsdatascience.com/a-guide-to-word-embeddings-8a23817ab60f>

En cada uno de los 3 espacios vectoriales podemos observar similitudes entre las palabras que aparecen. En el caso del primer espacio vectorial, podemos observar que la palabra man genera

un vector cuya trayectoria, definida por su dirección, pasa por la palabra woman. Esto, se traduce en que las palabras con un significado similar, es decir, que aparecen en contextos parecidos, van a tener una representación vectorial similar. Además, podemos ver el mismo comportamiento con palabras como verbos y sustantivos propios.

Generación de Word Embeddings:

Una vez hemos eliminado preprocesado el conjunto de textos, comienza la generación de Word Embeddings. En este caso, exemplificaremos este proceso con un Dataframe que contiene una columna llamada ‘clean_joined’ que en cada fila tiene palabras de títulos luego de preprocesar. Para crear este representación vectorial, utilizamos un modelo de inteligencia artificial llamado Word2Vec de la librería gensim.models:

```
from gensim.models import Word2Vec
import numpy as np
model = Word2Vec(df['clean_joined'], vector_size=3, window=5)
print(model)
df_jm = pd.DataFrame()
def get_average_word2vec(tokens, model):

    vectors = [model.wv[token] for token in tokens if token in model.wv]

    if len(vectors) == 0:
        return [0] * model.vector_size
    print(tokens)
    print(len(vectors))

df['embedding'] = df['clean_joined'].apply(lambda x: get_average_word2vec(x.split(), model))

embeddings_df = pd.DataFrame(df['embedding'].tolist())
embeddings_df['class'] = df['clickbait_class']

embeddings_df.to_csv('embeddings_with_class.csv', index=False)
```

Figura 3.6: Código de Word Embeddings. Fuente: Juan Manuel Rodríguez 2024.

El modelo Word2Vec, se entrena en el momento de su inicialización. Toma los siguientes parámetros de entrada:

- Corpus de textos: La lista de textos (títulos) que se quieren transformar. Con estas entradas, se genera el vocabulario del modelo (variable wv) donde cada palabra es asignada a un vector inicial de dimension vector_size.

- `vector_size`: La dimensión del espacio vectorial, en este caso, 100. Esto significa que cada palabra será representada por un vector de 100 dimensiones. Cada dimensión representa una característica de la palabra.
- `window`: El número de palabras a considerar a la izquierda y derecha del contexto de la palabra objetivo. Este parámetro define el tamaño del "ventana de contexto". A continuación se puede ver un ejemplo de los conjuntos generados con una ventana de tamaño 2 para la frase 'The quick brown fox jumps over the lazy dog':

Source Text	Training Samples
The quick brown fox jumps over the lazy dog. ➔	(the, quick) (the, brown)
The quick brown fox jumps over the lazy dog. ➔	(quick, the) (quick, brown) (quick, fox)
The quick brown fox jumps over the lazy dog. ➔	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown fox jumps over the lazy dog. ➔	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

Figura 3.7: Ejemplo de `window` en Word Embeddings. Fuente: (Gilyadov., 2017).

<https://israelg99.github.io/2017-03-23-Word2Vec-Explained/>

Luego, para obtener un vector representativo de cada título, utilizamos la función `get_average_word2vec`. Esta función itera sobre los tokens de cada título y calcula el promedio de las dimensiones de los vectores de las palabras que están presentes en el vocabulario del modelo (`wv`). Es decir, para cada palabra de un título, si esta palabra aparece en el vocabulario (`wv`), se obtiene su vector de 100 dimensiones. El resultado final de la representación embedding de cada título, corresponde una **lista de longitud `vector_size` donde cada elemento es el promedio de cada una de las dimensiones de cada palabra que contiene**. Para visualizar esto, se plantea el siguiente ejemplo con 3 dimensiones para cada palabra:

```

títulos = ['presidente gobierno', 'pandemia vacuna']
vector_size = 3

# Vectores de ejemplo para cada palabra
vectors_presidente = [0.4, 0.6, 0.7]
vectors_gobierno = [0.1, 0.3, 0.7]

# Embedding para el título 'presidente gobierno'
embedding_título_1 = [(0.4 + 0.1)/2, (0.6 + 0.3)/2, (0.7 + 0.7)/2] # = [0.25, 0.45, 0.7]

vectors_pandemia = [0.3, 0.4, 0.5]
vectors_vacuna = [0.6, 0.7, 0.8]

# Embedding para el título 'pandemia vacuna'
embedding_título_2 = [(0.3 + 0.6)/2, (0.4 + 0.7)/2, (0.5 + 0.8)/2] # = [0.45, 0.55, 0.65]

```

Figura 3.8: Ejemplo de promedio en Word Embeddings. Fuente: Juan Manuel Rodríguez 2024.

El resultado final, guardado en un archivo csv, puede ser utilizado en modelos futuros para su entrenamiento. En este documento, cada fila representa uno de los títulos. Existe un total de 100 columnas que representan cada una de los promedios de las características de cada palabra del título. En la última columna, tenemos la clase asociada a cada noticia

291 to 300 of 400 entries								Filter	□
93	94	95	96	97	98	99	class		
-0.032956168	0.43598595	-0.04631227	0.1769386	-0.14065915	0.14431849	0.046965905	1		
-0.034169912	0.4393071	-0.044824947	0.17905727	-0.14333256	0.14255378	0.048778024	1		
-0.034121007	0.43813732	-0.045422573	0.17835902	-0.14293724	0.14275669	0.048025656	1		
-0.03306461	0.4365208	-0.04462541	0.17683342	-0.14236656	0.14225855	0.048292924	1		
-0.033677474	0.43590012	-0.04536699	0.17716146	-0.14140698	0.14250669	0.047206577	1		
-0.03448983	0.43738964	-0.0455187	0.17810811	-0.14133814	0.14372195	0.048138067	1		
-0.034318592	0.43587312	-0.04421966	0.17680517	-0.14220977	0.14086254	0.04990173	1		
-0.03426424	0.43558693	-0.04698258	0.1773838	-0.14019588	0.14264582	0.04837791	1		
-0.033597376	0.43793225	-0.045348357	0.17851721	-0.14201581	0.14342158	0.049108986	1		
-0.03312666	0.4359919	-0.044168197	0.17652223	-0.14205639	0.14203796	0.048362352	1		

Figura 3.9: representación csv de Word Embeddings. Fuente: Juan Manuel Rodríguez 2024.

A nivel conceptual, los Word Embeddings permiten representar palabras en espacios vectoriales donde las palabras se relacionan entre sí en función de su contexto. El parámetro **window** define el contexto de cada palabra, es decir, las palabras vecinas que influyen en su representación

vectorial. De esta manera, palabras que aparecen en contextos similares tendrán representaciones vectoriales cercanas en el espacio vectorial. (Véase figura 3.5).

3. Probar los distintos algoritmos para llevar a producción el que consiga la mayor precisión:

En esta etapa, se experimenta con diferentes algoritmos de clasificación, como modelos de **aprendizaje automático supervisado** (por ejemplo, XGB Classifier, Random Forest, MultinomialNB, etc.), para determinar cuál proporciona el mejor rendimiento en términos de precisión en la clasificación de los títulos y cuerpos de noticias. Se utilizan técnicas de validación cruzada y métricas de evaluación como la precisión para comparar y seleccionar el mejor modelo. Una vez seleccionado el algoritmo óptimo, se procede a su entrenamiento con el conjunto de datos completo y se prepara para su implementación en producción.

Los algoritmos de clasificación utilizados fueron los siguientes:

Algoritmo Random Forest:

Los bloques fundamentales de este algoritmo son los árboles de decisión. Cada árbol es construido seleccionando un subconjunto aleatorio de características y otro subconjunto aleatorio de datos de entrenamiento. Esto genera un conjunto de entrenamiento rico en variedad.

- Se crean B árboles de decisión, cada uno entrenado con un subconjunto aleatorio de noticias.
- En cada nodo de cada árbol, se selecciona aleatoriamente un subconjunto de características (por ejemplo, palabras o n-grams) para determinar el mejor punto de división.
- Para una nueva noticia x , cada árbol T_b emite una predicción $h_b(x)$.
- La clase final se decide por mayoría de votos:

$$H(x) = \text{modo}\{h_1(x), h_2(x), \dots, h_B(x)\}$$

Esto se lee como $H(x)$, la predicción, es igual a la clase que aparece con mayor frecuencia entre las predicciones (votos).

XGB Classifier:

Este tipo de clasificación también utiliza árboles de decisión. Los primeros dos pasos son iguales que en el algoritmo Random Forest, cada árbol se entrena con un subconjunto aleatorio del conjunto de entrenamiento y en cada nodo se selecciona aleatoriamente un subconjunto de características. En esta fase, una representación gráfica de un tramo de un árbol donde se comparan matrices de tokens (palabras) es la siguiente:

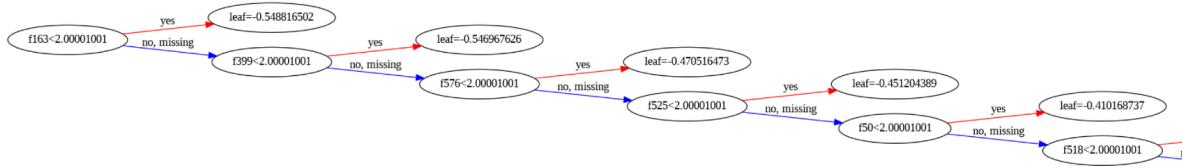


Figura 3.10: Ejemplo de árbol en XGB Classifier. Fuente: Juan Manuel Rodríguez 2024

Para explicar la ilustración analizaremos el primer nodo, ya que todos se comportan de la misma forma. Podemos observar el valor f163. Esto equivale a una columna del conjunto, es decir, representa una característica particular del conjunto que se utiliza como regla en ese nodo. Los valores con los que se compara f163 son ajustados durante el entrenamiento para lograr la mejor precisión. Luego, en la bifurcación se observan dos posibles caminos:

- **Yes:** La condición se cumple, resultando en una leaf (hoja) con un valor en el que derivarán los textos que no cumplen las condiciones hasta ese nodo, pero sí en él. Dependiendo de cuántos textos lleguen a ese valor, este se volverá más importante en la decisión final, ya que aparecerá un mayor número de veces.
- **No:** La condición no se cumple, es decir, el valor de la variable f163 es mayor o igual a 2.00001001. En consecuencia, se extiende el árbol a otro nodo y se añade una nueva característica, f399. El proceso continúa hasta agotar las características atribuidas a ese árbol.

A partir de ese punto, los pasos son los siguientes:

- **Corrección de errores:** Cada vez que se construye un nuevo árbol, este intenta corregir los errores de los anteriores. Esta corrección se realiza mediante un proceso llamado **gradient boosting**, que consiste en elegir siempre los caminos que reducen el gradiente del árbol. Esto se traduce en que las características ganan o pierden peso dependiendo de si el árbol que las utiliza consigue un mayor o menor gradiente.

- **Predicción:** Para una nueva entrada x , cada árbol T_b emite una predicción $h_b(x)$. Esta decisión está basada en sus hojas más exitosas.
- **Decisión final:** El valor final se decide por un proceso de agregación, como el promedio ponderado de las predicciones de todos los árboles.

$$H(x) = \text{modo}\{h_1(x), h_2(x), \dots, h_B(x)\}$$

Esto se lee como $H(x)$, la predicción, es igual al promedio entre las predicciones de cada árbol.

Regresión logística

En general, la regresión logística es adecuada cuando la variable de respuesta Y es categórica múltiple (admite varias categorías de respuesta, tales como mejora mucho, empeora, se mantiene, mejora, mejora mucho), pero es especialmente útil en particular cuando solo hay dos posibles respuestas que es el caso más común.

Con el conjunto de datos del que disponemos, una matriz de tokens se genera una función logística. Ejemplo:

Contamos con los siguientes documentos

Documento 1: La mejor opción es seguir el camino y no seguir la colina

Documento 2: La peor estrategia es seguir el río

Documento 3: Nadie debe seguir la colina

A partir de estas palabras, quitando las stop_words y ordenando las cabeceras en orden alfabético, obtenemos las siguientes cabeceras:

mejor opción seguir camino peor estrategia río nadie colina

Finalmente, obtenemos la siguiente matriz de tokens:

Documents	cami no	colina	estrategi a	nadie	mejo r	opción	peor	río	seguir
Documento 1	1	1	0	0	1	1	0	0	2
Documento 2	0	0	1	0	0	0	1	1	1
Documento 3	0	1	0	1	0	0	0	0	1

Cuadro 3.4: Matriz de tokens obtenida con vectorización

Se genera una z para cada documento:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$

Cada beta es una característica de la noticia. Es decir, uno de los números que componen su fila.

En el caso de la primera, tendríamos

$$z = \beta_0 + \beta_1 \cdot 1 + \beta_2 \cdot 1 + \beta_3 \cdot 0 + \beta_4 \cdot 0 + \beta_5 \cdot 1 + \beta_6 \cdot 1 + \beta_7 \cdot 0 + \beta_8 \cdot 0 + \beta_9 \cdot 2$$

β_0 es el intercepto y las x son los valores de entrada. El intercepto suele situarse en 0.5 que es la probabilidad intermedia entre las 2 clasificaciones.

Se aplica la función logística para ubicar cada z en un rango entre 0 y 1. Esto significa que nuestra función clasificará los conjuntos que le pasemos en torno a su cercanía a 0 o a 1. Si está más cerca del 1 pertenecerá a la categoría de VERDADERA y si está más cerca de 0 será FALSA.

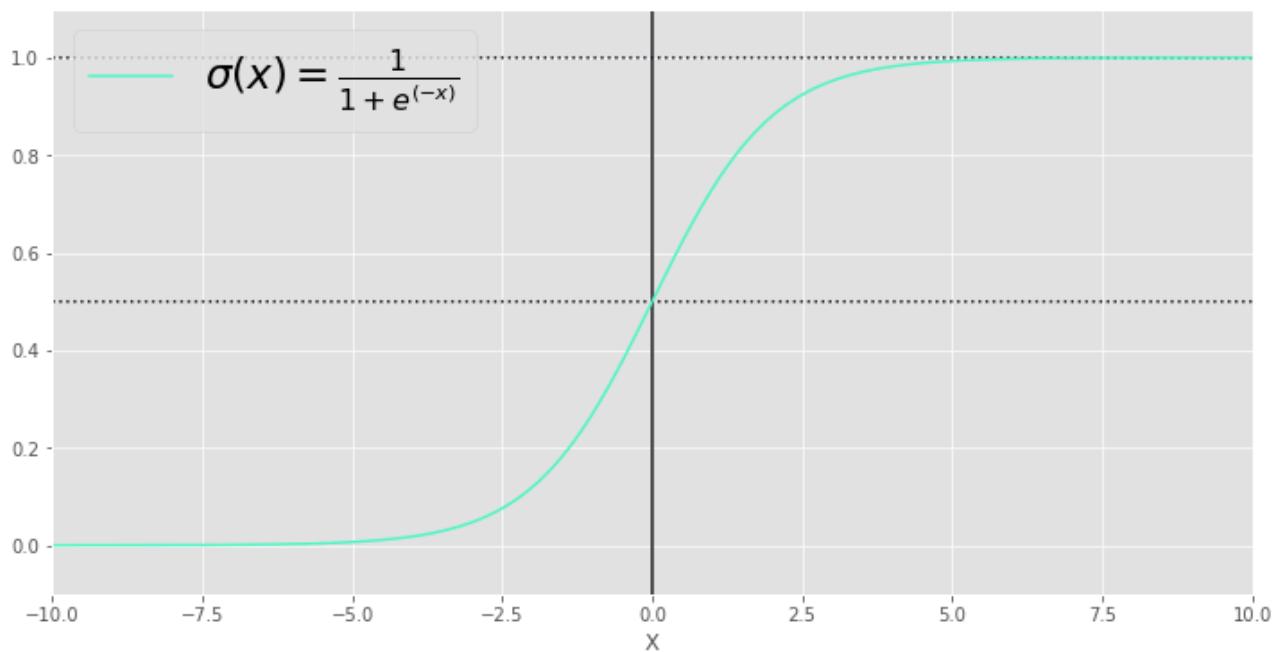


Figura 3.11: Función logística. Fuente: (IADelta., 2020).
<https://iadelta.com/inteligencia-artificial/regresion/logistica-y-softmax/>

En el caso del algoritmo de sklearn, al principio los coeficientes son aleatorios y se definen a medida que el modelo se entrena.

Cost function (función de costo):

Para entrenar el modelo se busca reducir la **función de costo**. Esta es una medida del error entre la probabilidad predecida y la etiqueta real de la noticia. Como cada conjunto llegará asociado con su resultado, podrá determinar a través de las distintas interacciones cuáles son los mejores coeficientes para esta función logística.

MultinomialNB

El clasificador bayesiano se fundamenta en el Teorema de Bayes (Bayes, 1764), el cual considera dos conjuntos de sucesos denotados como A y B, buscando calcular la probabilidad de ocurrencia del evento en un conjunto, dado la probabilidad del otro. En este contexto, A representa el primer conjunto de eventos y B el segundo. La probabilidad del suceso A se denota como $P(A)$, la de B como $P(B)$, y la de B dado A como $P(B|A)$. De este modo, la probabilidad del suceso A, dada la verificación de lo que ocurrió con B, es decir $P(A|B)$, se calcula mediante la ecuación (2):

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{P(B)}$$

En la ecuación se cumple que:

- $P(A_i)$ son las probabilidades *a priori*,
- $P(B|A_i)$ es la probabilidad de B en la hipótesis A_i ,
- $P(A_i|B)$ son las probabilidades *a posteriori*.

La probabilidad de que un documento d sea clasificado en la clase c se calcula de la siguiente manera:

$$P(c|d) = P(c) \prod_{1 \leq k \leq n_d} P(t_k|c)$$

donde $P(t_k|c)$ es la probabilidad condicional de que el término t_k ocurra en el documento (noticia) de la clase c . $P(t_k|c)$ se interpreta como una medida de cuánto la presencia de t_k contribuye a que d pertenezca a la clase c . $P(c)$ es la probabilidad *a priori* de que un documento (noticia) pertenezca a la clase c .

Durante el entrenamiento, MultinomialNB estima los parámetros de la distribución multinomial basado en el conteo frecuencias observadas, como por ejemplo prioridades y posibilidades condicionadas de las características de una clase, en el conjunto de entrenamiento.

Una vez entrenado el modelo MultinomialNB, es capaz de clasificar nuevas instancias de clases calculando la probabilidad *a posteriori* de cada característica observada y seleccionando la de mayor probabilidad. Los parámetros de la distribución multinomial se estiman en base al conteo de las frecuencias observadas. Si los términos del documento no proporcionan evidencia clara para diferenciar una clase de otra, se elige la clase con mayor probabilidad *a priori*.

Los términos $\langle t_1, t_2, \dots, t_{nd} \rangle$ son los tokens en d que forman parte del vocabulario utilizado para la clasificación, y n_d es la cantidad de esos tokens presentes en d . Para determinar la mejor clase para el documento, se selecciona la clase con la probabilidad *a posteriori* más alta, definida por la función $cmap$:

$$c_{\text{map}} = \arg \max_{c \in C} P(c|d)$$

¿Por qué se han utilizado estos cuatro modelos?

Los modelos empleados en el estudio entran dentro de los más conocidos y utilizados para realizar tareas de clasificación. En consecuencia, existe una abundante documentación y recursos educativos disponibles para aprender y comprender su funcionamiento, lo que facilita su implementación y ajuste. Además, los modelos seleccionados proporcionan un equilibrio entre simplicidad y rendimiento, siendo efectivos para el tamaño y características de los datos manejados hasta ahora.

Respecto a la posibilidad de aplicar modelos de **deep learning** (aprendizaje profundo), no se ha considerado en esta fase del proyecto. Estos modelos, ofrecen ventajas significativas cuando se maneja una mayor cantidad de datos y se requiere evaluar características más complejas como la **sintaxis** de los textos. Dado los requerimientos actuales de la aplicación, se priorizó el uso de modelos más simples.

Además, considerando que el proyecto tiene un diseño modular, donde los modelos son componentes independientes e intercambiables dentro de la API, se cuenta con flexibilidad a la hora de probar y aplicar diferentes clasificadores. Esto, podrá aprovecharse en futuras versiones o actualizaciones del producto, mejorando continuamente el rendimiento del sistema.

Arquitectura general de la aplicación:

En vista de los requerimientos de cada uno de los módulos de la aplicación abordados en el presente apartado (**Modelos de machine learning, API y add-on**), la arquitectura de la aplicación a gran escala consta de cuatro partes. En primer lugar, dos modelos de clasificación, uno para títulos y el otro para artículos. En segundo lugar, una API donde se cargan los modelos y que contiene métodos que permiten realizar predicciones con ellos. Y, en tercer lugar, un add-on para navegador web que realiza peticiones a la API para acceder a sus métodos de predicción y que

muestra los resultados al usuario mediante un pop-up.

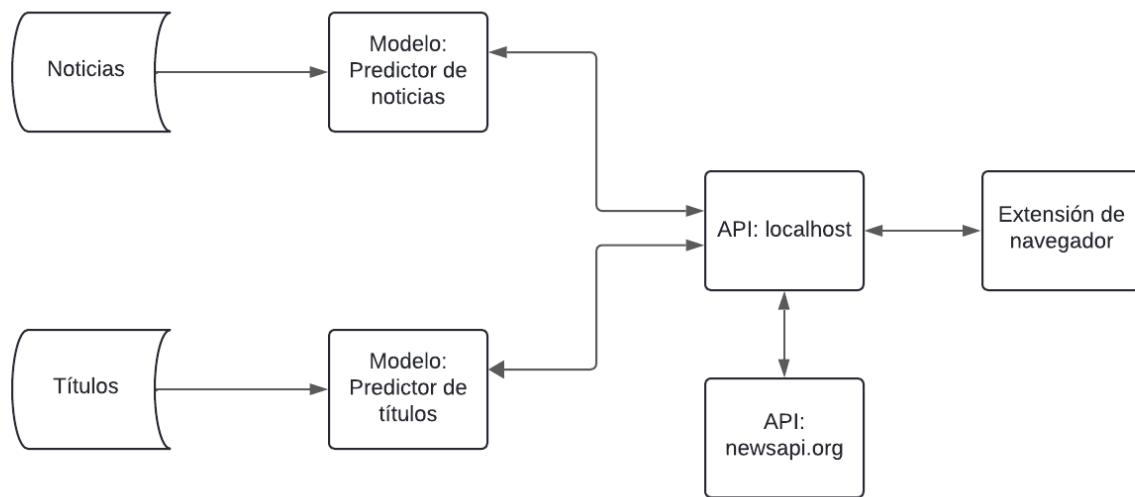


Figura 3.12: Arquitectura de la aplicación. Fuente: Juan Manuel Rodríguez 2024

4. Proyecto

4.1. Resumen de contribuciones y productos desarrollados

- Extensión para navegador web para analizar noticias: se trata del principal producto desarrollado en el presente estudio y aquel que engloba al resto de tecnologías. Este último punto se traduce en que es dependiente del resto de productos para su funcionamiento.
- Arquitectura de aplicación web que permite conectar modelos de inteligencia artificial con una extensión de navegador web.

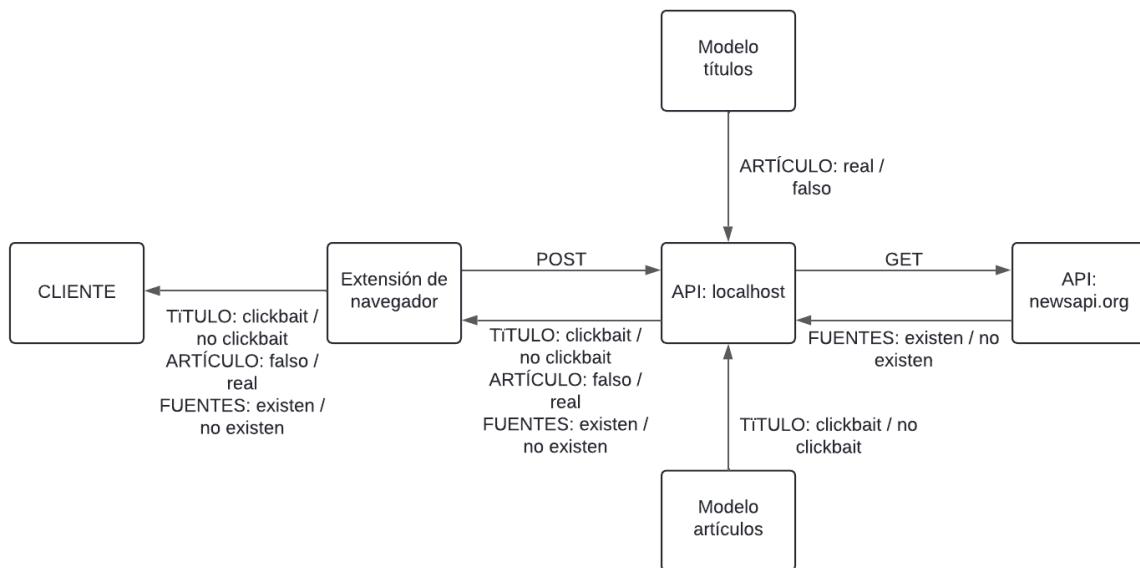


Figura 4.1: Esquema de comunicación entre cliente, extensión y API. Fuente: Juan Manuel Rodríguez 2024

- API para analizar textos: es dependiente de los modelos de clasificación y dispone de métodos que permiten examinar artículos, títulos y fuentes de títulos.
- Modelo de procesamiento del lenguaje natural para clasificar títulos en las categorías clickbait y no clickbait. No tiene dependencias del resto de productos.
- Modelo de procesamiento del lenguaje natural para clasificar artículos en las categorías real y falso. No tiene dependencias del resto de productos.

4.2. Planificación temporal

Se han planificado 3 fases de tareas divididas en 3 períodos temporales. La primera fase se enfoca sobre todo en investigación, recopilación de datos y conceptualización de la arquitectura de la aplicación. La segunda fase, una vez consiste en desarrollar las partes de la aplicación siguiendo la arquitectura. La última fase, consiste en realizar pruebas de los módulos e integrar las partes en el add-on para luego probar el producto final obtenido. Además, durante la fase 3 se aborda la redacción de la memoria final asociada al proyecto. Se ha realizado el siguiente diagrama de Gantt para ilustrar la relación entre tareas y períodos temporales:

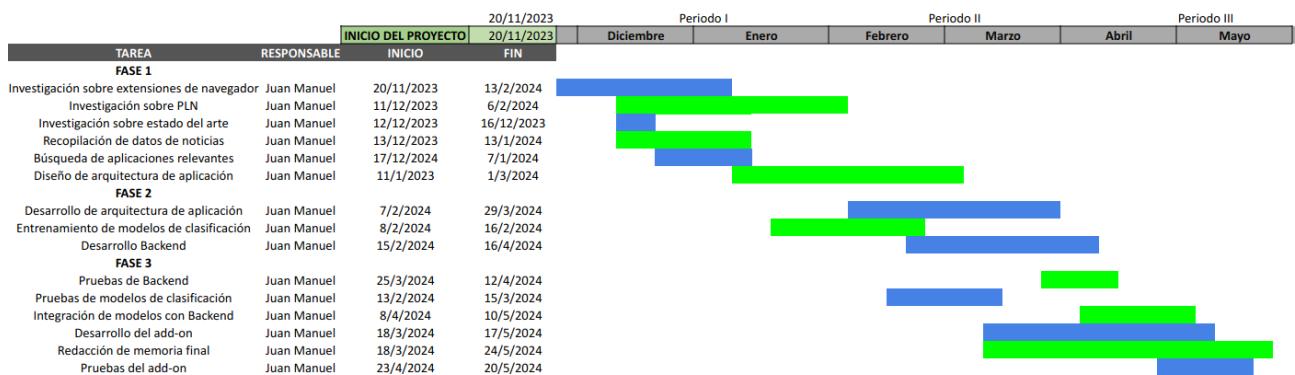


Figura 4.2: Diagrama de Gantt. Fuente: Juan Manuel Rodríguez 2024

4.3. Recursos empleados

Extensión:

- **JavaScript:** ECMAScript 2021 (ES12)
- **Manifesto:** Versión 3
- **HTML:** HTML5
- **CSS:** CSS3

Modelos de clasificación:

- **Python:** Python 3.10.12
- **Datasets utilizados:**
 - Dataset de títulos generados por ChatGPT 3.5
 - Dataset de artículos de Kaggle: [Noticias falsas en español](#)
- **Librerías utilizadas:**
 - pandas (version 2.2.2)
 - numpy (version 1.26.4)
 - scikit-learn (versión 1.4)
- **Algoritmos de Clasificación y validación utilizados:**
 - RandomForestClassifier
 - XGBClassifier
 - LogisticRegression
 - MultinomialNB
 - cross_val_score

API:

- **Python:** Python 3.10.12

- **Frameworks:**
 - Flask 3.0.2
 - Werkzeug 3.0.1
- **Fuente de Datos Externa:**
 - newsapi.org (Versión de desarrollador)

 Developer \$0 <small>totally free, start now</small>	 Business \$449 <small>per month, billed monthly</small>	 Advanced \$1749 <small>per month, billed monthly</small>	 Enterprise Contact us
For development and testing of all project types.	For production and published commercial projects.	For larger projects that require exceptional resources.	For enterprise projects that require premium data or bespoke solutions.
Search articles and get live top headlines <small>?</small>	Search all articles and get live top headlines <small>?</small>		Access to extra articles and an extended source library <small>?</small>
Articles have a 24 hour delay	New articles available in real-time <small>?</small>		Articles are enriched with 20 additional data-points <small>?</small>
Search articles up to a month old	Search articles up to 5 years old <small>?</small>		Article & story clustering
CORS enabled for localhost <small>?</small>	CORS enabled for all origins <small>?</small>		Custom classifications, tagging, & information extraction
100 requests per day <small>?</small>	250,000 requests per month included	2,000,000 requests per month included	Unlimited requests
No extra requests available <small>?</small>	\$0.0018 per extra request <small>?</small>	\$0.0009 per extra request <small>?</small>	Add sources on demand
No uptime SLA	No uptime SLA	99.95% uptime SLA <small>?</small>	Custom SLA & dedicated chat <small>?</small>
Basic support <small>?</small>	Email support <small>?</small>	Priority email support <small>?</small>	On-premise deployment <small>?</small>

Figura 4.3: Planes de pago newsapi.org. Fuente: (NewsAPI., 2024). <https://newsapi.org>

- **Servidor:**
 - localhost en el puerto 5000

4.4. Trabajo desarrollado

API:

La única función a la que llama nuestra extensión mediante la API, es `process_html`. Este método es el encargado de recibir el html de la noticia, extraer las partes que interesan para realizar el análisis, es decir aquellas que puedan contener la fecha de la noticia, el título o el artículo. Nuestra función tiene dependencias de importaciones. Procederemos a explicar cada función importada para garantizar la comprensión de la función `process_html`.

```
from flask import jsonify
```

La función `jsonify` de la librería Flask se utiliza para convertir estructuras de datos a formato json.

```
from bs4 import BeautifulSoup
```

La librería BeautifulSoup de Python, nos permite extraer los distintos elementos de archivos HTML o XML. En primer lugar se encarga de parsear (parsing) un documento con el parser que le indiquemos. Parsear es convertir el documento en un árbol lógico ordenado por los elementos que lo componen y sus relaciones. Luego, una vez “parseado”, podemos encontrar los elementos que nos interesen.

```
from datetime import datetime, timedelta
```

La librería `datetime` de Python nos permitirá operar con datos de tipo fecha y obtener la fecha actual con horas, minutos, segundos, etc.

```
from resolvers.get import get_news_title
```

Código de get_news_title:

```
def get_news_title(title, date):
    try:
        # quitamos stop words
        filtered_title = ' '.join([word for word in title.split() if word not in stop_words])

        # ordenamos las palabras por longitud
        title_ordered_by_len = sorted(filtered_title.split(), key=len, reverse=True)
        if(len(title_ordered_by_len) > 5): title_ordered_by_len = title_ordered_by_len[0:3]
        title_with_and = ' AND '.join([word for word in title_ordered_by_len if word not in stop_words])

        exact_title_url = f"https://newsapi.org/v2/everything?searchIn=title&q="
        exact_title_url += f"&q={title_with_and}&page=1&language=es&apiKey={api_key}"
        today = datetime.today().date()

        # restamos 30 dias a la fecha de hoy
        date_limit = today - timedelta(days=30)

        if(date):
            if((date - timedelta(days=7)) > date_limit):
                date_limit = date - timedelta(days=7)

        exact_title_url = f"https://newsapi.org/v2/everything?searchIn=title&from={date_limit}"
        exact_title_url += f"&q={title_with_and}&page=1&language=es&apiKey={api_key}"
        exact_response = requests.get(exact_title_url)
        final_result = []

        # extraemos las fuentes
        data = exact_response.json()
        print(len(data['articles']))
        for i in range(len(data['articles'])):
            article = data['articles'][i]
            final_result.append(article['source']['name'])

        final_result = list(set(final_result))
        return final_result

    except Exception as e:
        print(e)
        return e
```

Figura 4.4: Código del método para obtener fuentes de noticias. Fuente: Juan Manuel Rodríguez 2024.

La función `get_news_title` es la encargada de interactuar con `newsapi.org`, la API de noticias que utilizamos para obtener fuentes que hayan publicado un título similar al que nos ataña. Nuestra función recibe 2 parámetros, **fecha y título**. Con respecto al contenido del título, si luego de eliminar las palabras vacías, el título sigue conteniendo más de 5 palabras, nos quedamos solo con las 3 de mayor longitud y solicitamos a nuestra API de noticias que nos devuelva un resultado similar. En relación con la fecha, cuando la noticia tenga fecha vamos a utilizar la fecha de la

noticia menos 1 semana para buscar el título, mientras que cuando no tenga fecha, revisaremos las noticias del último mes.

Es importante destacar que estamos utilizando el plan gratuito de newsapi.org. Este, solo permite solicitar noticias que tengan como máximo 1 mes de antigüedad desde la fecha en la que se solicitan. Por este motivo, en los casos en los que, o bien la fecha no exista o la fecha de la noticia supere el límite de tiempo, o bien al restarle 1 semana, lo supere, vamos a configurar límite de tiempo a la fecha actual menos 1 mes, es decir, el máximo tiempo pasado al que podemos acceder con nuestro plan.

Fuentes del título de la noticia:

El modelo se basará únicamente en procesamiento del lenguaje natural para clasificar la noticia entre falsa y verdadera. Considerando esto, pienso que el contenido o descripción, es la única parte que mediante su análisis puede llevarnos a aumentar la fiabilidad de la extensión. Por otra parte, datos como el título serán analizados mediante llamadas a la API de noticias de newsapi.org.

La llamada que utilizaremos será para traer noticias con títulos relacionados al que buscamos.

GET

```
https://newsapi.org/v2/everything?q=ArgentinaANDcampeon&titulo&apiKey=x  
xxxxxxxxxx
```

El parámetro q es el que nos permite pedir títulos por palabras clave. También podemos agregar comillas al valor del parámetro q para referirnos a frases literales.

```
GET https://newsapi.org/v2/everything?q="Argentina campeon del  
mundo"&apiKey=xxxxxxxxxx
```

Si podemos acceder a la fecha de la noticia que estamos analizando, otros parámetros que nos permitirán reducir los objetivos de la búsqueda son from y to para establecer un intervalo temporal.

GET

```
https://newsapi.org/v2/everything?q=apple&from=2023-12-17&to=2023-12-17  
&sortBy=popularity&apiKey=xxxxxxxxxx
```

Como solo queremos noticias en español, utilizamos lenguaje en nuestra llamada. Además, el endpoint /everything busca dentro de todos los campos que poseen las noticias. Como nuestro objetivo es buscar solo títulos, agregamos searchIn.

GET

```
https://newsapi.org/v2/everything?searchIn=title&q=Campeon AND Argentina&page=1&language=es&apiKey=xxxxxxxxxx
```

El resultado de nuestra búsqueda devuelve títulos como los siguientes:

```
"title": "Los puntajes de la selección argentina frente a Brasil: las manos de Dibu Martinez, la cabeza de Nicolás Otamendi y el espíritu del mariscal del campeón del mundo",
  "description": "En el Maracaná, el seleccionado dio una muestra más de su estirpe y celebró un triunfo memorable",
  "url": "https://www.lanacion.com.ar/deportes/futbol/los-puntajes-de-la-seleccion-argentina-frente-a-brasil-las-manos-de-dibu-martinez-la-cabeza-de-nid22112023/",
  "urlToImage": "https://resizer.glanacion.com/resizer/v2/un-quite-espectacular-de-cuti-romero-uno-de-ZNHPOCZKMVBWVI6B7XIECY4GCI.JPG?auth=f7a4f264142e25d80d0e434f325ccaa0dafc1bd85d084c6b1864aa5f87fdcbcf&width=768&quality=70&smart=false",
  "publishedAt": "2023-11-22T04:13:00Z",
  "content": "RÍO DE JANEIRO. La Argentina terminó el año a toda orquesta con una victoria por 1 a 0 sobre Brasil en el mismísimo estadio Maracaná, que implicó cortarle el récord de 64 partidos como local sin derr... [+4219 chars]"
},
{
  "source": {
    "id": "la-nacion",
    "name": "La Nación"
  },
  "author": "Andrés Eliceche",
  "title": "Argentina, rey del fútbol: ser campeón del mundo y seguir escribiendo la historia, la mejor ofrenda para el capitán Messi",
```

Figura 4.5: Respuesta de newsapi.org. Fuente: Juan Manuel Rodríguez 2024

El campo que nos interesa de cara a nuestro análisis de noticias falsas es el de source. Cuando existan fuentes, se mostrará el listado al usuario de la extensión y este podrá valorar si las consideramos serias como para fiarse de la veracidad de la noticia. Para esto, solicitamos a la API una lista de fuentes en Español de diferentes lugares del mundo y los llevamos a la pop-up de nuestra extensión.

```
from utils.process_title import process_title
```

Código de process_title:

```

import pickle

from sklearn.feature_extraction.text import CountVectorizer

model_pkl_file = "title_classifier_model.pkl"

with open(model_pkl_file, 'rb') as file:

    vect, clf = pickle.load(file)

```

Podemos observar que nuestro código instancia las variables desde vect y clf desde un archivo .pkl o pickle. Los archivos pickle son ficheros binarios que contienen una estructura de datos de Python serializada. En este caso, contiene el modelo que se utiliza para clasificar de títulos previamente entrenado. Para leer el archivo, usamos el modo rb (read binary o modo de lectura binario).

Tener nuestro modelo entrenado cargado en un archivo pkl nos permite moverlo entre distintos programas con facilidad. Además, la gran ventaja de esto es que no tenemos que entrenar nuevamente el modelo cada vez que ejecutemos nuestro programa lo cual imposibilitaría que el producto trabaje en tiempo real. Simplemente, hay que cargar el modelo en una variable y ya se encuentra operativo para empezar a predecir.

Código para descargar un modelo en formato pickle:

```

import pickle

model_pkl_file = "model_name.pkl"

with open(model_pkl_file, 'wb') as file:
    pickle.dump((vect, clf), file)

```

A la hora de generar nuestro pkl, es necesario exportar ese mismo objeto vect junto con el modelo y extraerlo del pkl. Nuestro código emplea el método dump para verter el modelo en un archivo pkl. Usamos el modo wb (write binary o escritura binaria) para escribir el archivo.

Retos a la hora de cargar el modelo en pkl:

Considerando que nuestro modelo entrenado y convertido a pkl es un multinomial model generado a partir de la libreria sklearn.naive_bayes, nos encontramos con una serie de impedimentos. Nuestro modelo no permite que le pasemos como input un texto directamente. En cambio,

necesita recibir una matriz numérica de las mismas dimensiones que aquella con al que fue entrenada y contiene el número de palabras totales pertenecientes al conjunto de entrenamiento.

Esto, supone que si se utilizó un objeto para vectorizar las cadenas que se le pasaron al modelo para entrenarlo, tenemos que utilizar el mismo objeto para vectorizar los inputs que queremos que nuestro modelo prediga. En nuestro caso, el objeto encargado de la conversión de listas de palabras en una matriz fue el siguiente:

```
vect = CountVectorizer()  
X_train_dtm = vect.fit_transform(X_train)  
X_test_dtm = vect.transform(X_test)
```

A la hora de generar nuestro pkl, es necesario exportar ese mismo objeto vect junto con el modelo y extraerlo del pkl. Ahora nuestro código para verter (dump) el modelo en un archivo pkl.

```
def process_title(string_to_classify):  
  
    X_to_classify = vect.transform([string_to_classify])  
  
    predicted_class = clf.predict(X_to_classify)  
  
    if(predicted_class[0] == 1):  
  
        return "NO CLICKBAIT"  
  
    else:  
  
        return "CLICKBAIT"
```

Una vez tenemos cargado el pkl con el modelo de clasificación de títulos y el vectorizador utilizado para entrenarlo, nuestra función `process_title()` recibe un título como parámetro de entrada, lo transforma en una matriz de números con `vect` y predice su valor con la función `predict()`. Si el resultado es 1, significa que el título es válido (no clickbait) y en caso contrario que es clickbait. Este resultado es convertido a tipo string y devuelto por la función para ser empleado por la API en el cuerpo de la respuesta de la función `process_html()`.

```
from utils.process_article import process_article
```

Código de `process_article`

```

import pickle

model_pkl_file = "news_naive_bayes_mod.pkl"

with open(model_pkl_file, 'rb') as file:

    vect, clf = pickle.load(file)

def process_article(string_to_classify):

    X_to_classify = vect.transform([string_to_classify])

    predicted_class = clf.predict(X_to_classify)

    return predicted_class

```

El código de `process_article`, utiliza la variable `clf`, donde se encuentra el modelo clasificador de artículos y la variable `vect` donde está el vectorizador. Recibe un artículo como parámetro de entrada, lo transforma en una matriz de números con `vect` y predice su valor con la función `predict()`.

En este punto, retomamos la función `process_html` luego de haber desglosado sus dependencias.

```

date_refs = {

    'ene': '01',

    'feb': '02',

    'mar': '03',

    'abr': '04',

    'may': '05',

    'jun': '06',

    'jul': '07',

    'ago': '08',

    'sep': '09',

    'oct': '10',
}

```

```

'nov': '11',
'dic': '12'
}

async def process_html(data):
    try:
        html = data.get('html')
        if not html: # HTML missing
            return "HTML missing", 400
        soup = BeautifulSoup(html, 'html.parser')
        title = soup.find('title').text if soup.find('title') else ""
        h1 = soup.find('h1').text if soup.find('h1') else ""
        h2 = soup.find('h2').text if soup.find('h2') else ""
        p_tags = soup.find_all('p')
        # Concatenamos los contenidos de <p> en una variable
        p = ''.join([p.get_text() for p in p_tags])
        article = soup.find('article').text if soup.find('article') else ""
        classification = "no classification available"

```

Esta primera sección del código de la función `process_html()`, mediante la librería `BeautifulSoup` dedica a extraer las partes del html que serán procesadas por nuestros modelos. Por un lado, extrae el título y lo guarda en `title`, y por otro, el artículo. En las páginas de periódicos, lo habitual es que el cuerpo de la noticia se encuentre separado por párrafos donde cada uno está bajo su propio elemento `<p>`. Por ende, cuando extraemos el artículo, prevalece el texto que extraemos de las distintas `p` antes que el de la etiqueta `article` para realizar nuestra clasificación.

```

if(p):
    result = process_article(p)
    classification = "TRUE NEW" if result else "FAKE NEW"
    with open('p.txt', 'w') as f:

```

```

f.write(p)

elif(article):

    result = process_article(article)

    classification = "TRUE NEW" if result else "FAKE NEW"

# Classify the title

title_classification = "no classification available"

if(title):

    result = process_title(title)

    title_classification = result

time = soup.find('time').text if soup.find('time') else ""

if(len(time) > 0):

    time = time[:-12]

    time = time.split(' ')

    time[1] = date_refs[time[1]]

    time = '-'.join(time)

sources = get_news_title(title, time)

sources_string = ', '.join(sources)

return_event = {

    "title": title,

    "h1": h1,

    "h2": h2,

    "article": article,

    "p": p,

    "article_classification": classification,

    "title_classification": title_classification,
}

```

```
        "sources": sources_string

    }

    response = jsonify(return_event)

    response.headers.set('Access-Control-Allow-Origin', '*')

    return response, 200

except Exception as e:

    print(e)

    return "Internal Server Error", 500
```

Este método, en su respuesta, generada a partir del objeto `return_event`, devuelve todos los elementos extraídos de la noticia además de los resultados de los análisis. Visualizar los campos de la respuesta en los logs de nuestra extensión nos facilitará la comprensión del funcionamiento de holístico de este software. Además, es el encargado de llamar al método `get_news_title` de la API para solicitar las fuentes del título. Para realizar esta llamada, además del contenido del título, formatea la fecha para que sea legible por el método y llama a la función con ambos parámetros.

Despliegue de aplicaciones de Flask

```

1  from flask import Flask, request
2  from resolvers.get import get_news_title
3  from resolvers.post import clean_html, process_html
4  import asyncio
5  from flask_cors import CORS # Import CORS from flask_cors
6
7  app = Flask(__name__)
8  CORS(app)
9
10 @app.route('/getNewsTitle', methods=['GET'])
11 def news_title():
12     return get_news_title()
13
14 @app.route('/process_html', methods=['POST'])
15 def process_html_route():
16     try:
17         data = request.get_json()
18         result = asyncio.run(process_html(data))
19         return result
20     except Exception as e:
21         print(e)
22         return "Internal Server Error", 500
23
24 @app.route('/clean_html', methods=['POST'])
25 def clean_html_content():
26     try:
27         data = request.get_json()
28         html = data.get('html')
29         if not html: # HTML missing
30             return "HTML missing", 400
31         return clean_html(html)
32     except Exception as e:
33         print(e)
34         return "Internal Server Error", 500
35
36 if __name__ == '__main__':
37     app.run(debug=True)
38

```

Figura 4.6: Código de archivo de ejecución de Flask API. Fuente: Juan Manuel Rodríguez 2024.

Luego de realizar las importaciones necesarias, empezamos por generar una instancia de una aplicación de Flask y la nombramos app. Luego, enrutamos nuestras funciones y definimos el método que invocará cada función. Para acceder a los métodos, lo haremos mediante nuestro servidor localhost, al cual podemos realizar peticiones desde nuestra extensión del navegador. Finalmente, procedemos a ejecutar nuestro código con la función run(). Cuando ejecutemos el archivo que contenga este script, se ejecutará en el puerto 5000, el predeterminado de las aplicaciones de Flask.

A continuación, podemos observar los logs que genera la API al responder a cada petición:

```
* Serving Flask app 'main.py'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
0
127.0.0.1 - - [20/May/2024 13:20:27] "POST /process html HTTP/1.1" 200 -
0
127.0.0.1 - - [20/May/2024 13:21:59] "POST /process html HTTP/1.1" 200 -
0
127.0.0.1 - - [20/May/2024 13:23:40] "POST /process html HTTP/1.1" 200 -
0
127.0.0.1 - - [20/May/2024 13:24:18] "POST /process html HTTP/1.1" 200 -
0
127.0.0.1 - - [20/May/2024 13:25:08] "POST /process html HTTP/1.1" 200 -
0
127.0.0.1 - - [20/May/2024 13:27:51] "POST /process html HTTP/1.1" 200 -
0
127.0.0.1 - - [20/May/2024 13:29:11] "POST /process html HTTP/1.1" 200 -
```

Figura 4.7: Respuestas de API en localhost:5000. Fuente: Juan Manuel Rodríguez 2024.

Tests Unitarios para la API

```
import pytest

from flask import Flask, request

from unittest.mock import patch, Mock

from resolvers.post import process_html

# Create a Flask application for testing

@pytest.fixture

def app():

    app = Flask(__name__)

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

    async def process_html_endpoint():

        data = await request.json

        return await process_html(data)

    return app

@pytest.fixture
```

```

def client(app):
    return app.test_client()

# Mock the external dependencies

@pytest.fixture

def mock_dependencies():
    with patch('resolvers.post.BeautifulSoup') as mock_bs, \
        patch('utils.process_article.process_article') as
mock_process_article, \
        patch('utils.process_title.process_title') as mock_process_title, \
        patch('resolvers.get.get_news_title') as mock_get_news_title:

        yield {
            'mock_bs': mock_bs,
            'mock_process_article': mock_process_article,
            'mock_process_title': mock_process_title,
            'mock_get_news_title': mock_get_news_title,
        }

# Helper function to post data to the endpoint

async def post_data(client, data):
    response = await client.post('/process_html', json=data)
    return response

# Test cases

@pytest.mark.asyncio

```

```

async def test_process_html_success(client, mock_dependencies):
    mock_bs = mock_dependencies['mock_bs']
    mock_process_article = mock_dependencies['mock_process_article']
    mock_process_title = mock_dependencies['mock_process_title']
    mock_get_news_title = mock_dependencies['mock_get_news_title']

    html_content = """
<html>
    <head><title>Test Title</title></head>
    <body>
        <h1>Test H1</h1>
        <h2>Test H2</h2>
        <p>Test Paragraph 1</p>
        <p>Test Paragraph 2</p>
        <article>Test Article</article>
        <time>19 May 2024 12:00:00 GMT</time>
    </body>
</html>
"""

    mock_bs.return_value.find.return_value.text = 'Test Title'
    mock_bs.return_value.find_all.return_value = [
        Mock(get_text=Mock(return_value='Test Paragraph 1')),
        Mock(get_text=Mock(return_value='Test Paragraph 2'))
    ]
    mock_process_article.return_value = True

```

```

mock_process_title.return_value = 'TRUE TITLE'

mock_get_news_title.return_value = ['Source 1', 'Source 2']

data = {'html': html_content}

response = await post_data(client, data)

json_data = response.get_json()

assert response.status_code == 200

assert json_data['title'] == 'Test Title'

assert json_data['h1'] == 'Test H1'

assert json_data['h2'] == 'Test H2'

assert json_data['p'] == 'Test Paragraph 1Test Paragraph 2'

assert json_data['article'] == 'Test Article'

assert json_data['article_classification'] == 'TRUE NEW'

assert json_data['title_classification'] == 'TRUE TITLE'

assert json_data['sources'] == 'Source 1, Source 2'

@pytest.mark.asyncio

async def test_process_html_missing_html(client):

    data = {}

    response = await post_data(client, data)

    assert response.status_code == 400

    assert response.get_data(as_text=True) == 'HTML missing'

@pytest.mark.asyncio

```

```

async def test_process_html_internal_server_error(client,
mock_dependencies):
    mock_dependencies['mock_bs'].side_effect = Exception('Some error')

    html_content = "<html></html>"

    data = {'html': html_content}

    response = await post_data(client, data)

    assert response.status_code == 500

    assert response.get_data(as_text=True) == 'Internal Server Error'

```

Se han desarrollado tres tests unitarios para evaluar el método `process_html` de la API. Estos tests fueron generados con la asistencia de ChatGPT, y cada uno cubre un caso diferente para asegurar una robusta validación del método. A continuación se describen los casos de prueba implementados:

Test de HTML completo y correcto (`test_process_html_success`): Este test verifica que el método `process_html` procese correctamente un documento HTML bien formado, extrayendo y clasificando adecuadamente los elementos como el título, encabezados, párrafos y artículos.

Test de HTML incompleto (`test_process_html_missing_html`): Este test evalúa el comportamiento del método cuando se le proporciona un documento HTML incompleto o ausente. Se espera que la API responda con un código de estado 400 indicando que el HTML está faltante.

Test de manejo de errores de BeautifulSoup (`test_process_html_internal_server_error`): Este test simula un error en la librería BeautifulSoup para comprobar que el método `process_html` maneje adecuadamente las excepciones inesperadas, devolviendo un código de estado 500 que indica un error interno del servidor.

Resultados obtenidos al ejecutar los tests:

- `test_process_html_success`: Verifica el procesamiento exitoso de un HTML completo.
- `test_process_html_missing_html`: Confirma la correcta gestión de un HTML ausente.
- `test_process_html_internal_server_error`: Valida el manejo adecuado de excepciones internas.

```
test process html.py::test process html success
test process html.py::test process html missing html
test process html.py::test process html internal server error
```

Entrenamiento de los modelos de clasificación de artículos:

En primera instancia, cargamos nuestro conjunto de noticias falsas y reales en un DataFrame y añadimos una columna para definir la clase al que pertenece cada una. Una vez definidas las clases, concatenamos los Dataframes en un único DataFrame llamado df:

```
true_df=pd.read_csv("/[REDACTED]/onlytrue1000.csv")
fake_df=pd.read_csv("/[REDACTED]/onlyfakes1000.csv")

true_df["fake_new_class"]=0 # Todas las noticias de este dataset son reales
fake_df["fake_new_class"]=1 # Todas las noticias de este dataset son falsas

df=pd.concat([true_df,fake_df],axis=0).reset_index(drop=True)
```

Figura 4.8: Generación de DataFrame de noticias. Fuente: Juan Manuel Rodríguez 2024.

Preprocesamiento de datos:

Además, es importante mencionar que el procesamiento del lenguaje natural no tiene en cuenta palabras como **artículos** definidos ni indefinidos (la, una), **preposiciones** de tiempo y de lugar como (a, de) y **conjunciones** disyuntivas (o, u). Por supuesto que si buscamos interpretar el texto desde una perspectiva humana, estas palabras resultan cruciales para comprender el sentido y el sentimiento del mismo. No obstante, puesto que un modelo de inteligencia artificial solo busca coincidencias en patrones numéricos, estas son **palabras sin valor**, ya que no permite asignarle un valor cuantificable a palabras que contextualizan mejor el concepto que trata de reflejar la frase. Por ejemplo, disponemos de la siguiente noticia:

“Algunas de las voces extremistas más conocidas de EE.UU., cuentas asociadas con al movimiento de la conocida como alt-right o ligadas a grupos antisemitas o xenófobos, han sido eliminadas de un plumazo de la redes sociales Facebook e Instagram.“

Si separamos las palabras, las agrupamos y ordenamos de mayor a menor aparición, obtenemos el siguiente resultado:

```
[('de', 5), ('la', 2), ('o', 2), ('Algunas', 1), ('las', 1), ('voces', 1), ('extremistas', 1), ('más', 1), ('conocidas', 1), ('EE', 1), ('UU', 1), ('cuentas', 1), ('asociadas', 1), ('con', 1), ('al', 1), ('movimiento', 1), ('conocida', 1), ('como', 1), ('alt', 1), ('right', 1), ('ligadas', 1), ('a', 1), ('grupos', 1), ('antisemitas', 1), ('xenófobos', 1), ('han', 1), ('sido', 1), ('eliminadas', 1), ('un', 1), ('plumazo', 1), ('redes', 1), ('sociales', 1), ('Facebook', 1), ('e', 1), ('Instagram', 1)]
```

Podemos observar que las palabras que más aparecen son 'de', 'la' y 'o'. Si dejamos este tipo de palabras al entrenar nuestro modelo, este asociará textos que contengan mayor presencia de palabras como estas, desviándose del objetivo. Por otra parte, aquellas palabras cuya cuantificación ayudan al modelo a darle sentido al texto, pierden peso.

Al desglosar el texto, se descartan las palabras que no deben ser utilizadas por nuestro modelo para predecir. Para esto, debemos definir una lista de palabras comúnmente utilizadas dentro del lenguaje que llamamos palabras vacías o **stop words** en inglés. Esta lista contendrá palabras que queremos evitar analizar.

Aplicamos una función que busca coincidencias entre las palabras que contiene la noticia y las palabras de nuestros conjuntos de stopwords. Si la palabra no es una stop word, se añade al resultado:

```
def preprocess(text):
    result = []
    for token in gensim.utils.simple_preprocess(text):
        if token not in gensim.parsing.preprocessing.STOPWORDS and token not in stop_words:
            result.append(token)

    return result
df['clean_text'] = df['text'].apply(preprocess)
df['clean_joined'] = df['clean_text'].apply(lambda x: " ".join(x))
```

Figura 4.9: Preprocesamiento de palabras de artículos. Fuente: Juan Manuel Rodríguez 2024.

Mediante la función `apply`, aplicamos el preprocesamiento definido en `preprocess()` a cada elemento del DataFrame y guardamos cada resultado en la columna 'clean_text'. Después, dado

que la función apply devuelve una lista, aplicamos la función join() a cada lista de palabras y las convertimos en cadenas de palabras separadas por espacios. Estos resultados se guardan en la columna clean_joined. Finalmente, contamos con los textos preprocesados y en el formato adecuado para tokenizar.

```
X=df['clean_joined']
y=df['fake_new_class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state=42)
```

Esta linea de código separa el dataset en un set de entrenamiento y otro de prueba usando la función `train_test_split` de scikit learn. Esta función es de las más utilizadas a la hora de repartir un dataset entre 2 conjuntos aleatorios de entrenamiento y de prueba. El argumento `test_size=0.2` indica que el 20% de la data se utilizará para el conjunto de prueba. Es decir, de las 2000 noticias que tenemos, se utilizarán 400 para el conjunto de test y 1600 para el de entrenamiento. Si este argumento no se especifica, se setea a 0.25 por defecto. El argumento `random_state=42` configura un estado aleatorio para elegir noticias al azar para cada conjunto

Tokenización y matriz de términos de documentos

Para que estas palabras sean útiles de cara a procesarlas con un modelo de machine learning, tenemos que tokenizarlas. Esto, significa convertirlas en una matriz de números. Para esto, scikit learn ofrece una serie de métodos:

Una matriz de conteo de tokens es una representación en formato numérico de una colección de documentos de texto. Cada columna de la matriz corresponde a una palabra única, mientras que cada fila se asocia con un documento. En nuestro caso de ejemplo, generamos nuestra matriz con nuestros conjuntos de palabras limpias de entrenamiento y de prueba. Para generar esta matriz utilizaremos el objeto CountVectorizer() que emplea la técnica Bag of Words:

```
vect = CountVectorizer()
X_train_dtm = vect.fit_transform(X_train)
X_test_dtm = vect.transform(X_test)
print(X_train_dtm.shape, X_test_dtm.shape)
```

Guardamos las matrices en las variables X_train y X_test con el sufijo dtm que significa Document Term Matrix (matriz de términos de documentos)

Obtenemos los siguientes resultados:

Para X_train_dtm.shape: (1600, 10860)

Esto significa que tenemos 1600 documentos, en nuestro caso noticias, y un total de 10860 palabras diferentes

Para X_test_dtm.shape: (400, 10860)

Esto significa que tenemos 400 noticias, y un total de 10860 palabras diferentes

Es importante observar que en ambos casos el total de palabras de 10860. Esto sucede porque nuestra variable CountVectorizer() se queda con la cifra generada por el conjunto que tenga el mayor número de palabras diferentes, en este caso, x_train.

Finalmente, podemos empezar a entrenar los modelos:

```

# RandomForestClassifier
model_1 = RandomForestClassifier(random_state=42)
model_1.fit(X_train_dtm, y_train) |
y_pred=model_1.predict(X_test_dtm)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: %.2f%%" % (accuracy * 100.0))

# XGBClassifier
model_2 = XGBClassifier()
model_2.fit(X_train_dtm, y_train)
y_pred_model_2=model_2.predict(X_test_dtm)
score = metrics.accuracy_score(y_test, y_pred_model_2)
print("Accuracy: %.2f%%" % (score * 100.0))
precision = precision_score(y_test, y_pred_model_2)

# LogisticRegression
model_3=LogisticRegression()
model_3.fit(X_train_dtm, y_train)
y_pred_model_3=model_3.predict(X_test_dtm)
accuracy = accuracy_score(y_test, y_pred_model_3)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
precision = precision_score(y_test, y_pred_model_3)

# MultinomialNB
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB(force_alpha=True)
clf.fit(X_train_dtm, y_train)
pred = clf.predict(X_test_dtm)
score = metrics.accuracy_score(y_test, pred)
print("Accuracy: %.2f%%" % (score * 100.0))

```

Figura 4.10: Entrenamiento de modelos de clasificación. Fuente: Juan Manuel Rodríguez 2024.

Instanciamos un clasificador en la variable clf o model_x y procedemos a entrenarlo con la función fit. Este método, recibe 2 parámetros de entrada. Por una parte, la matriz de tokens donde cada fila representa un artículo de una noticia, y por otra, las clases (real o falsa) asociadas a cada fila de la matriz. Luego, predecimos las clases del conjunto de prueba con la función predict() e imprimimos la precisión obtenida en esa predicción.

Entrenamiento de los modelos de clasificación de títulos:

Los pasos para generar el modelo de clasificación de títulos fueron los mismos que para el clasificador de artículos. Es decir, el preprocessamiento de datos fue Los modelos que se probaron corresponden a los 4 que se presentan en la figura 4.10. Por otra parte, se utilizaron un total de 400 títulos generados por ChatGPT de los cuales 200 eran clickbait y 200 no.



```
+ Code + Text  
▶ clickbait_titles = pd.read_csv('/[REDACTED]/Clickbait_Titles200.csv')  
nonClickbait_titles = pd.read_csv('/[REDACTED]/NonClickbait_Titles200.csv', delimiter='|')  
  
clickbait_titles["clickbait_class"]=0  
nonClickbait_titles["clickbait_class"]=1
```

Figura 4.11: Datasets de títulos. Fuente: Juan Manuel Rodríguez 2024.

Archivos de la extensión:

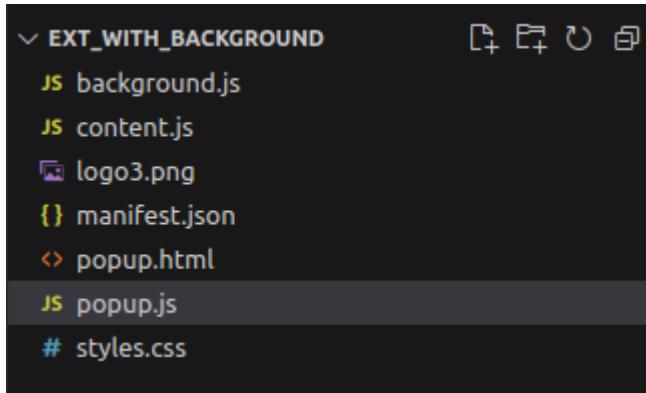


Figura 4.12: Estructura de archivos del add-on. Fuente: Juan Manuel Rodríguez 2024.

El add-on se divide en 6 archivos de código y una imagen que corresponde al logo.

manifest.json:

```
{ manifest.json > {} action > {} default_icon > 128
1   {
2     "name": "api call extension",
3     "version": "1.0",
4     "description": "This is an extension to try an api call",
5     "manifest_version": 3,
6     "author": "Juanma",
7     "permissions": ["activeTab", "webRequest"],
8     "background": {
9       "service_worker": "background.js"
10    },
11    "action": {
12      "default_popup": "popup.html",
13      "default_icon": {
14        "16": "logo3.png",
15        "48": "logo3.png",
16        "128": "logo3.png"
17      }
18    },
19    "content_scripts": [
20      {
21        "matches": ["<all_urls>"],
22        "js": ["content.js"]
23      }
24    ]
25  }
26}
27
```

Figura 4.13: Archivo manifest.json. Fuente: Juan Manuel Rodríguez 2024.

En el archivo de manifesto, indispensable para que el navegador asimile nuestra extensión define los metadatos de la extensión propiamente dicha como su nombre, versión, descripción, ícono, los permisos que requiere el add-on para funcionar en el navegador del usuario, y los enlaces a los scripts que ejecutará. El permiso de activeTab significa que la extensión tendrá acceso temporal a la pestaña activa del navegador del usuario cuando se la invoque. Además, el permiso de webRequest, le permitirá a la extensión realizar peticiones web, lo cual necesitamos para que se comunique con nuestra API ejecutándose en localhost.

Por otra parte, nuestra extensión dispone de un archivo de background que se ejecuta en un proceso distinto del resto de la extensión. Fue menester generar un código **background.js** para poder comunicar al script de la popup con el que realiza el web scraping sobre la pestaña activa del usuario para extraer los datos de la noticia. Cuando nuestro archivo de **content.js** extrae el

html del documento, es fundamental que este código se ejecute sobre la web que está visualizando el usuario y no sobre el código de la extensión.

Código de content.js

```
JS content.js > ...
1
2  const data_html = document.documentElement.innerHTML;
3
4  const data = { value: ` ${data_html} ` }; // data to be sent to the server
5  chrome.runtime.sendMessage({action: 'sendHtml', data: data}, (response) => {
6    console.log('html sent to background:', data);
7    console.log('response:', response);
8  });
9
```

Figura 4.14: Script content.js. Fuente: Juan Manuel Rodríguez 2024.

La variable `data_html` guarda el html de la página y la función `sendMessage` lo envía en forma de json para que otro proceso que esté “escuchando” mensajes lo capte. Una vez recibido el mensaje, el proceso receptor envía una respuesta. El inconveniente de colocar el código de `content.js` en el mismo script que el código correspondiente a la `popup` es que se extraería el html de la `popup` en vez del de la pestaña que ve el usuario. Luego, se enviarían esos datos a la API y esta devolvería siempre el mismo resultado y estaríamos analizando los datos de la extensión, lo cual no tiene sentido.

Código de background.js

```
JS background.js > ...
1  let html_data = '';
2
3  // Listen for messages from content scripts
4  chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {
5    if (message.action === 'sendHtml') {
6      html_data = message.data.value;
7      sendResponse({status: 'html received'});
8    }
9  );
10
11 // Listen for messages from the popup
12 chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {
13   if (message.action === 'getHtml') {
14     sendResponse({html: html_data});
15   }
16 });


```

Figura 4.15: Script background.js. Fuente: Juan Manuel Rodríguez 2024.

El código de background.js contiene 2 funciones que escuchan acciones. La primera función addListener, cuya acción asociada es sendHtml, se encarga de recibir el html que fue recogido y enviado por content.js. Este html, se guarda en la variable html_data para ser enviada a la popup, mediante un mensaje asociado al listener. En otras palabras, la segunda función, recibe un mensaje de popup.js cuya acción se llama getHtml y en el momento de escuchar esta petición, le envía en el html como respuesta con la función sendResponse({html: html_data}).

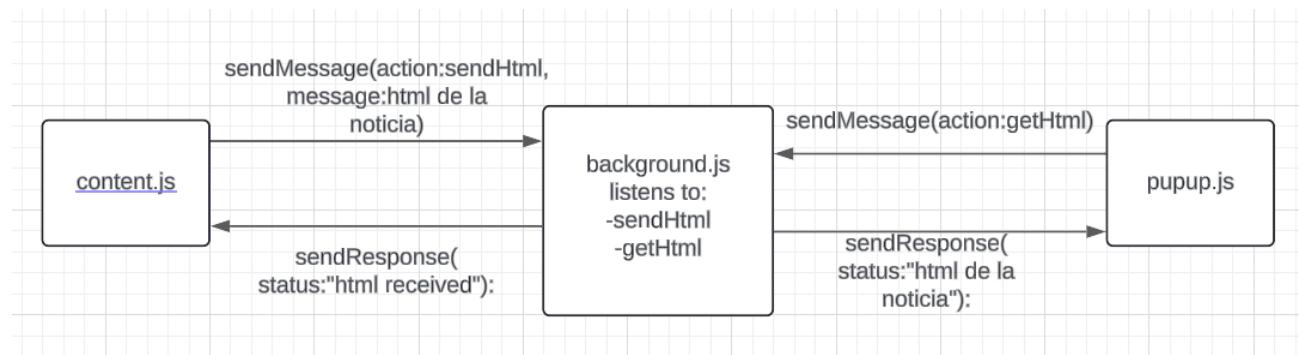


Figura 4.16: Esquema de comunicación entre popup.js y content.js. Fuente: Juan Manuel Rodríguez 2024.

Código de popup.js

```

async function postData(url = "", data = {}) {
  try{
    const response = await fetch(url, {
      method: "POST",
      mode: "cors",
      headers: {
        "Content-Type": "application/json"
      },
      body: JSON.stringify(data)
    });
    if (!response.ok) {
      throw new Error("HTTP error, status = " + response.status);
    }
  }
}
  
```

```

    }

    const responseData = await response.json(); // Parse JSON data

    console.log("post response: ", responseData);

    return responseData;

} catch(e) {

    console.error(e)

}

}

```

Esta primera función del código popup.js, postData(), es la encargada de realizar la petición POST a nuestra API ejecutándose en el puerto 5000 de nuestro localhost y devuelve la respuesta a la petición que en este caso corresponde con el análisis de la noticia.

```

const post_endpoint = 'http://localhost:5000/process_html';

let source_result = ""

let article_result = ""

let title_result = ""

const process_result = (data) => {

try{

    const t = data.title_classification;

    console.log('title: ', t)

    const a = data.article_classification;

    console.log('article: ', a)

    const s = data.sources;

    console.log('sources: ', s)

    const s_array = s.split(", ");

    console.log('arrays: ', s_array)
}

```

```

let title_newElement = document.createElement('div');

let article_newElement = document.createElement('div');

let sources_newElement = document.createElement('div');

if(t) {

    if (t == "CLICKBAIT") {

        title_result = "El análisis del título determina que es clickbait";

        title_newElement.innerHTML = '<div class="red-shape"></div>';

        document.getElementById("title-container").appendChild(title_newElement);

    } else{

        title_result = "El análisis del título determina que no es
clickbait";

        title_newElement.innerHTML = '<div class="green-shape"></div>';

        document.getElementById("title-container").appendChild(title_newElement);

    }

    document.getElementById("title_analysis").innerHTML = `<li
class="message-text">${title_result}</li>`;

    console.log(title_result);

} else console.log("no title analysis found")

if(a) {

    if (a == "FAKE NEWS") {

        article_result = "El análisis del artículo determina que la noticia
es falsa";

        article_newElement.innerHTML = '<div class="red-shape"></div>';

        document.getElementById("title-container").appendChild(article_newElement);

    }

}

```

```

}else if (a == "REAL NEW") {

    article_result = "El analisis del articulo determina que la noticia
es real";

    article_newElement.innerHTML = '<div class="green-shape"></div>'

document.getElementById("article-container").appendChild(article_newElement)
;

} else{

    article_result = "El analisis del articulo no ha podido determinar si
la noticia es falsa o real";

document.getElementById("article-container").appendChild(article_newElement)
;

}

document.getElementById("article_analysis").innerHTML = `<li
class="message-text">${article_result}</li>`;

console.log(article_result);

} else console.log("no article analysis found")



if(s_array.length > 0) {

    source_result = "El analisis de fuentes ha encontrado otras fuentes que
han reportado esta noticia";

    sources_newElement.innerHTML = '<div class="green-shape"></div>'

document.getElementById("sources-container").appendChild(sources_newElement)
;

    console.log("List of sources: ", s);

    document.getElementById("list_of_sources").innerHTML = s_array.map(m
=>`<li class="message-text">${m}</li>`);

}

```

```

} else{

    source_result = "El análisis de fuentes no ha logrado encontrar otras
fuentes que hayan reportado esta noticia";

    sources_newElement.innerHTML = '<div class="red-shape"></div>'
document.getElementById("sources-container").appendChild(sources_newElement)
;

    console.log(source_result);

}

document.getElementById("source_analysis").innerHTML = `<li
class="message-text">${source_result}</li>`;

} catch(e){

    console.error(e)

}

}

```

La función process_result() implementa la lógica mediante la que se renderiza o pinta la popup de nuestro add-on. Si el título es clickbait, mostrará un mensaje alusivo con un cuadrado de color rojo al final. Por otro lado, si el título no es clickbait, mostrará un mensaje validándolo junto con un cuadrado verde. Lo mismo ocurre para el análisis del artículo en los casos en los que sea falso o real respectivamente.

En cuanto al análisis de las fuentes existe la posibilidad de que se encuentren fuentes relacionadas con la noticia o no. Si ocurre la primera posibilidad, se muestra un listado de todas las fuentes junto con el mensaje y un cuadrado verde. Si no se encuentran fuentes relacionadas, aparecerá un mensaje dictaminándolo junto con un cuadrado rojo.

```

document.addEventListener('DOMContentLoaded', function () {

    // Request the html from the background script

    chrome.runtime.sendMessage({action: 'getHtml'}, (response) => {

        if (response && response.html) {

            console.log('Received html:', response.html); // Imprimimos el
html que recibimos de content.js

```

```
const data_value = { html: ` ${response.html} ` };
```

```
postData(post_endpoint, data_value)
```

```
.then(data => {
```

```
    process_result(data);
```

```
    console.log(data); // Imprimimos la respuesta de la API
```

```
})
```

```
.catch(error => {
```

```
    console.error(error); // Gestion de errores
```

```
});
```

```
}
```

```
});
```

```
) ;
```

Nuestro código emite un mensaje conteniendo la acción getHtml para recoger el html de la pestaña en la que se encuentra el usuario enviado por content.js. Este mensaje solo lleva una acción pero ningún dato asociado. Una vez que recibe la respuesta, la imprime y la guarda en la variable data. Finalmente, se utiliza la función postData para enviar el html a la API para que lo procese y una vez esta envía una respuesta, la función process_result renderiza la popup en consecuencia.

Funciones chrome.runtime: los métodos de la API chrome.runtime, nos permiten escuchar y responder eventos en tiempo de ejecución durante el ciclo de vida de la extensión. En este caso empleamos 2 métodos de esta API:

- `sendMessage()`: envía mensajes a receptores de eventos (event listeners) dentro o fuera de la extensión.
 - `onMessage.addListener()`: se dispara cuando se envía un mensaje desde un proceso de la extensión mediante la función `sendMessage()`

4.4.1. Valoración económica

Si el presente estudio estuviera presupuestado en España, siguiendo los costos medios por hora de los distintos profesionales que podrían estar implicados, incluiría los siguientes gastos:

	Actividad realizada	Profesional empleado	Costo medio por hora	Horas totales	Costo total
	Investigación sobre extensiones de Google Chrome	Arquitecto de Software	€23.96	80	€1916.8
	Arquitectura de la aplicación	Arquitecto de Software	€23.96	80	€1916.8
	Desarrollo Full Stack	Desarrollador Full Stack	€16.67	120	€2000.4
	Investigación sobre procesamiento del lenguaje natural para noticias.	Científico de datos	€19.79	80	€1583.2
	Desarrollo y entrenamiento de modelos de clasificación	Científico de datos	€19.79	40	€791.6
	Redacción del proyecto	Redactor	€12.70	120	€1524

TOTAL					€9732.8
-------	--	--	--	--	---------

Cuadro 4.1: Tabla de estimación de costos de proyecto

5. Resultados y discusión

Análisis exploratorio de los datos:

Este análisis consiste en una serie de pasos para comprender los datos del modelo y medir su validez. Considerando que se trata de un dataset con 2 columnas (título y clase), el objetivo es analizar la longitud de los textos, palabras comunes y n-grams para ambas clases. Estos datos aportan valor a la hora de entrenar el modelo.

Dataset títulos:

- Información básica del conjunto:

Se verifica el número de columnas, los tipos de datos de las mismas y la cantidad de filas pertenecientes a cada clase. El dataset consta de 200 títulos clickbait y 200 no clickbait. El valor de 0 de clickbait_class representa los contenidos clickbait y el 1 los no clickbait.

```

df.info()
|
class_distribution = df['clickbait_class'].value_counts()
print(class_distribution)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   title            400 non-null    object  
 1   clickbait_class  400 non-null    int64  
dtypes: int64(1), object(1)
memory usage: 6.4+ KB
clickbait_class
0    200
1    200
Name: count, dtype: int64

```

- Análisis de longitud de los textos:

En esta fase se estudia la longitud de los títulos de ambas clases con el fin de apreciar diferencias que puedan aportar información.

```
import seaborn as sns
import matplotlib.pyplot as plt

df['title_length'] = df['title'].apply(len)

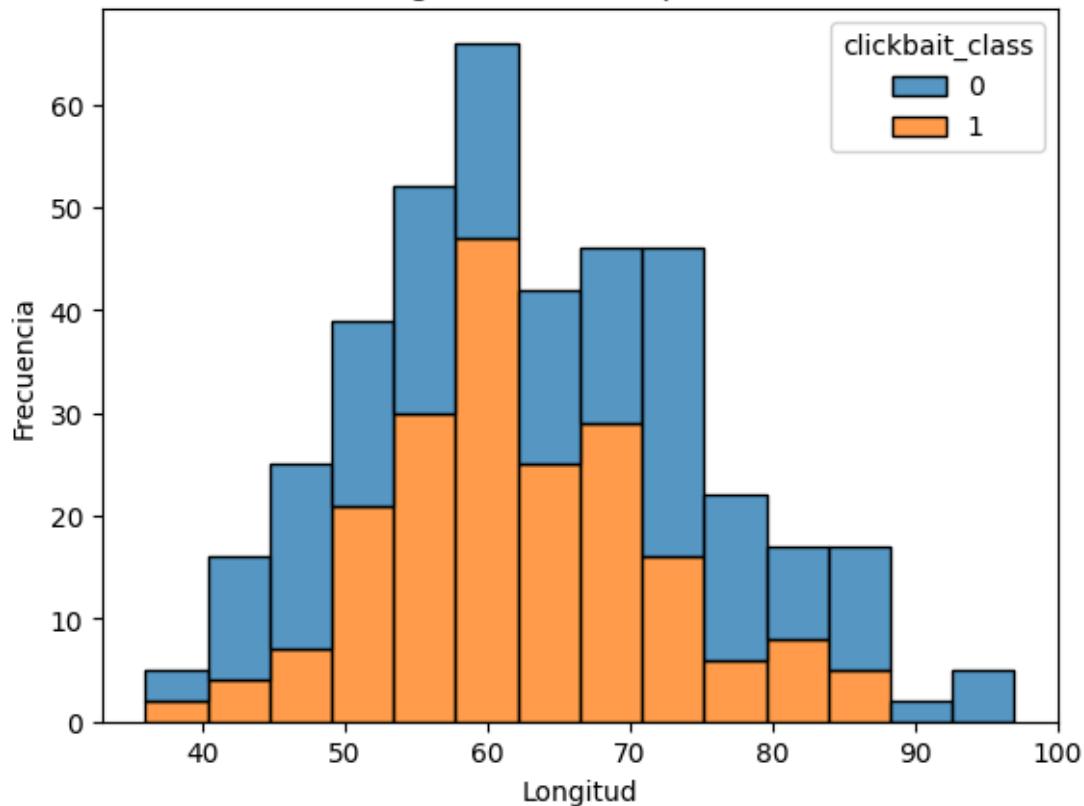
length_stats = df.groupby('clickbait_class')['title_length'].describe()
print(length_stats)

sns.histplot(data=df, x='title_length', hue='clickbait_class', multiple='stack')
plt.title('Longitud de títulos por clase')
plt.xlabel('Longitud')
plt.ylabel('Frecuencia')
plt.show()
```

Se obtiene la longitud de cada contenido y se ejecuta la función **describe()** sobre la columna donde esos datos se guardaron en el DataFrame para observar un resumen de la tendencia central, la dispersión y la forma de la distribución del dataset.

	count	mean	std	min	25%	50%	75%	max
clickbait_class								
0	200.0	64.175	13.833699	36.0	53.0	63.5	74.25	97.0
1	200.0	62.295	9.653181	36.0	56.0	61.0	68.00	86.0

Longitud de títulos por clase



En los datos que se encuentran encima del gráfico, se dispone de los valores correspondientes a los cuartos de cada subconjunto conjunto. Se observa, que los títulos de menor longitud tienen aproximadamente 36 palabras y los hay de ambas categorías. Por otra parte, los de mayor longitud tienen 97 palabras y solo los hay de categoría clickbait.

En ambas clases, la longitud media es cercana a 60 palabras. Los datos siguen una distribución en forma de campana, lo cual nos indica que la mayoría de los títulos se encuentran dentro del valor medio. Esto, aparte de visualizarse en la forma, también se puede observar en los datos asociados al 50%, ya que es cercano al correspondiente a la media en ambas clases. No obstante, existe una diferencia en la desviación estándar siendo superior la de la clase clickbait en más de un 30%. Esto, nos indica que hay mayor irregularidad entre los textos de ese subconjunto, es decir, que algunos tienen una gran longitud y otros una reducida, alejándose más del valor medio.

- Análisis de frecuencia de palabras:

En el procesamiento del lenguaje natural se eliminan las palabras vacías o que generan ‘ruido’ de los datos. Comúnmente, son las que tienen mayor frecuencia de repetición, ya que sirven para estructurar el lenguaje. Es valioso conocer como las frecuencias se transforman a partir de la eliminación de estas palabras.

```
from collections import Counter
from sklearn.feature_extraction.text import CountVectorizer

def get_top_n_words(corpus, n=None, stop_words=None):
    vec = CountVectorizer(stop_words=stop_words).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
    return words_freq[:n]

# Get top words for each class
top_words_clickbait = get_top_n_words(df[df['clickbait_class'] == 0]['title'], 10)
top_words_no_clickbait = get_top_n_words(df[df['clickbait_class'] == 1]['title'], 10)
```

La función **get_top_n_words** emplea CountVectorizer que inicializa un objeto con el vocabulario total del conjunto. Luego, para transformarlo en una matriz donde cada fila representa un documento y cada columna un token (Véase el cuadro 3.2) emplea el método **transform**. Luego realiza un conteo de cada palabra, agrupa los conteos por palabra y devuelve una lista de objetos {palabra: conteo} ordenada de mayor a menor conteos.

Previo a la eliminación de stop words, se imprimen las 10 palabras con mayor repetición en los conjuntos y obtenemos los resultados esperados, es decir, las más repetidas son las de este conjunto:

- Palabras más repetidas en títulos clickbait: [('para', 101), ('los', 90), ('en', 71), ('tu', 69), ('de', 67), ('el', 65), ('este', 61), ('que', 50), ('más', 45), ('la', 43)]
- Palabras más repetidas en títulos no clickbait: [('la', 234), ('de', 142), ('el', 74), ('los', 60), ('para', 58), ('cómo', 50), ('del', 47), ('en', 38), ('vida', 34), ('secretos', 33)]

Posteriormente, se procede a ejecutar la función incluyendo las stop words que se buscan eliminar en el **preprocesamiento**:

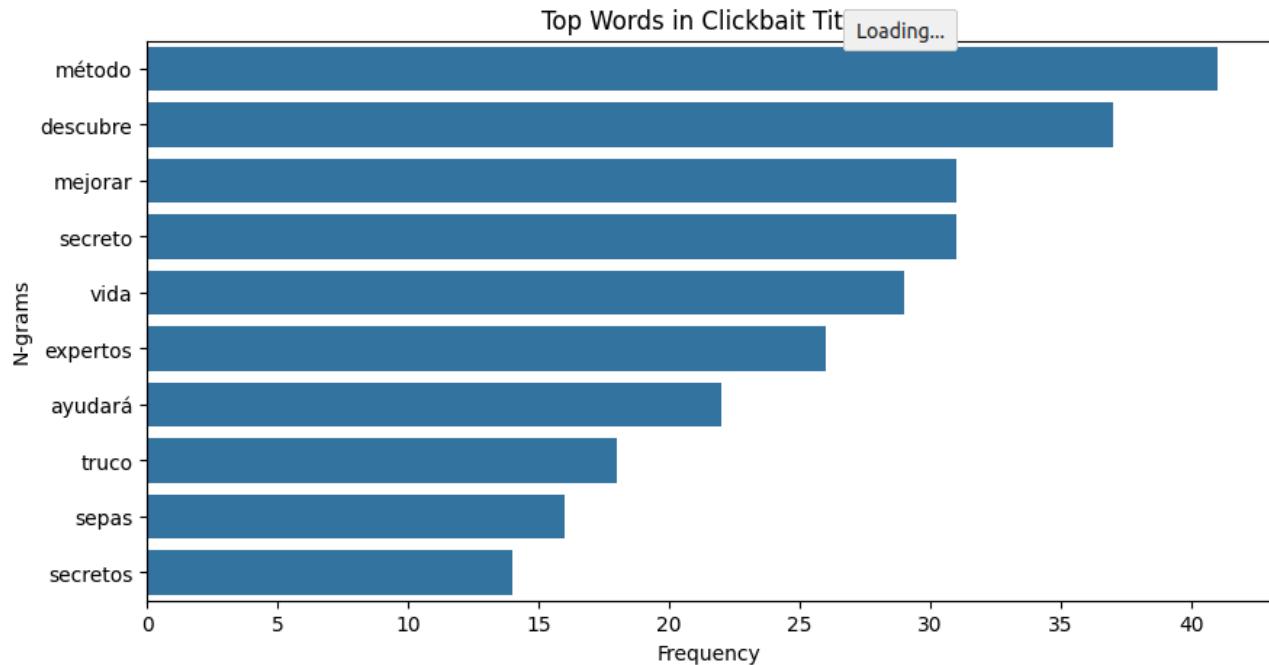
```

stop_words = pd.read_csv("stop_words_spanish.txt", sep=" ", header=None)
stop_words = stop_words.values.tolist()

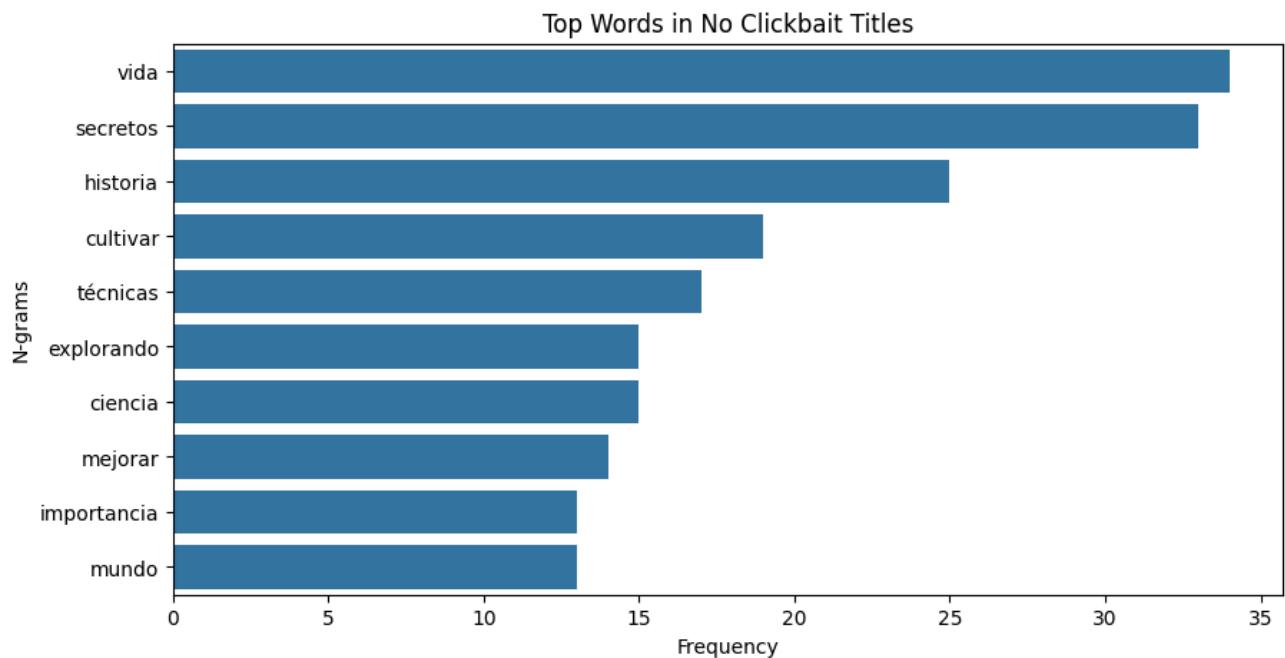
top_words_clickbait = get_top_n_words(df[df['clickbait_class'] == 0]['title'], 10, stop_words)
top_words_no_clickbait = get_top_n_words(df[df['clickbait_class'] == 1]['title'], 10, stop_words)

```

- Top words in clickbait titles: [('método', 41), ('descubre', 37), ('mejorar', 31), ('secreto', 31), ('vida', 29), ('expertos', 26), ('ayudará', 22), ('truco', 18), ('sepas', 16), ('secretos', 14)]



- Top words in no clickbait titles: [('vida', 34), ('secretos', 33), ('historia', 25), ('cultivar', 19), ('técnicas', 17), ('explorando', 15), ('ciencia', 15), ('mejorar', 14), ('importancia', 13), ('mundo', 13)]



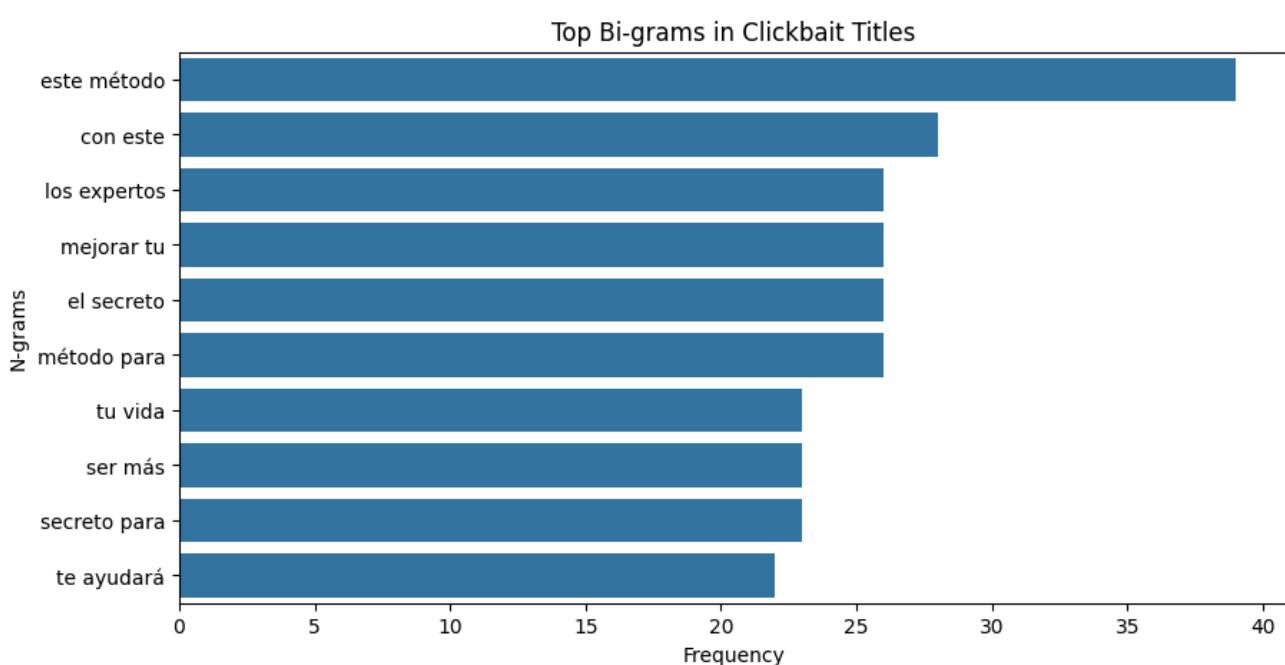
Es posible observar que las palabras **secretos** y **mejorar** tienen gran importancia en ambos conjuntos, por lo que se pueden generar conflictos al modelo.

- Análisis de n-grams:

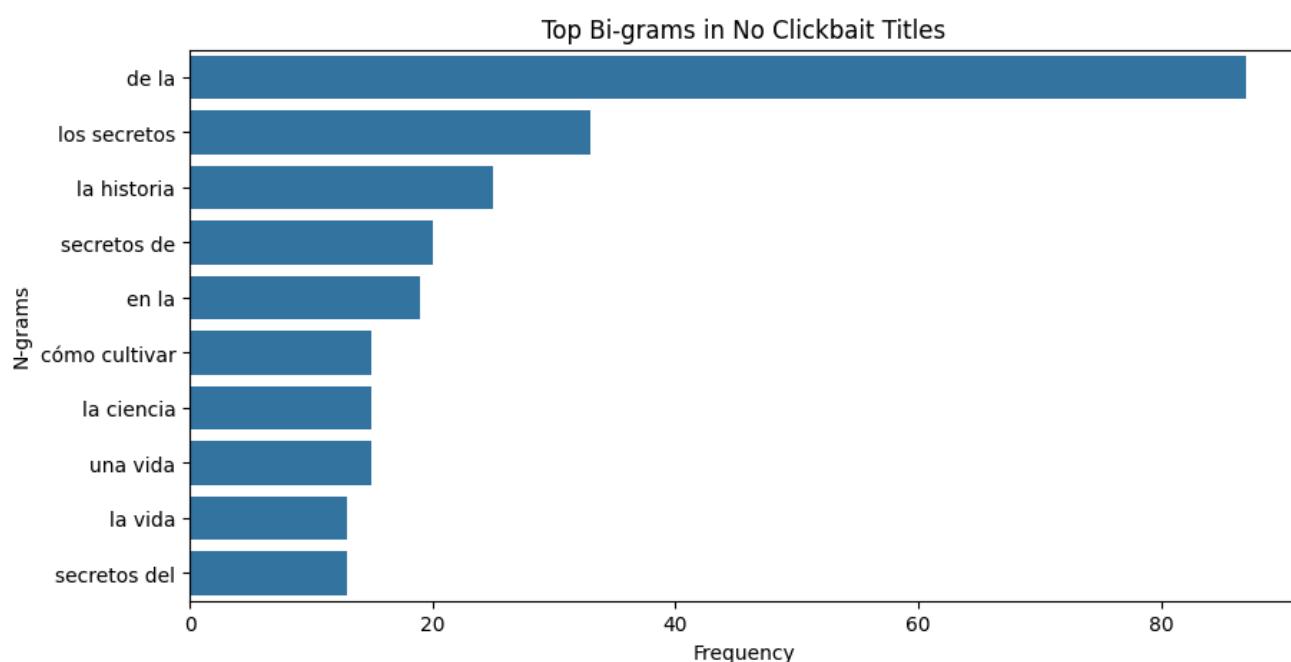
```
def get_top_n_ngrams(corpus, n=None, ngram_range=(2, 2), stop_words=None):
    vec = CountVectorizer(ngram_range=ngram_range, stop_words=stop_words).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
    return words_freq[:n]
```

La función **get_top_ngrams** tiene el mismo funcionamiento que la función **get_top_n_words** (Véase figura) con la diferencia que obtiene conteos de secuencias de palabras (gramas) de la longitud que indiquemos. Dentro de los parámetros de entrada se puede indicar el tamaño del n_grama que buscamos obtener. Predeterminadamente, este tendrá tamaño (2, 2), es decir, contará secuencias de 2 palabras agrupadas en paquetes de 2.

Bi-grams (2 palabras) en clickbait:



Bi-grams (2 palabras) en no clickbait:



Resultados de fase de prueba:

Los siguientes items son las referencias a los modelos que se mencionan en los cuadros a continuación:

- Modelo 1: Random Forest Classifier
 - Modelo 2: XGB Classifier
 - Modelo 3: Logistic Regression
 - Modelo 4: MultinomialNB
- Métricas de modelos de clasificación de artículos:

	Modelo 1	Modelo 2	Modelo 3	Modelo 4
Accuracy	70.50%	68.75%	74.75%	76.00%
Recall	84.08%	74.13%	77.11%	75.12%
Precision	66.27%	67.12%	73.81%	76.65%
F1	74.12%	70.45%	75.43%	75.88%

Cuadro 5.1: Tabla comparativa de modelos sin aplicar técnicas de validación

Se utilizaron 400 artículos para el conjunto de prueba, los cuales fueron clasificadas de la siguiente forma por MultinomialNB alcanzando una precisión del 76%:

```
Confusion Matrix:  
[[153 46]  
 [ 50 151]]
```

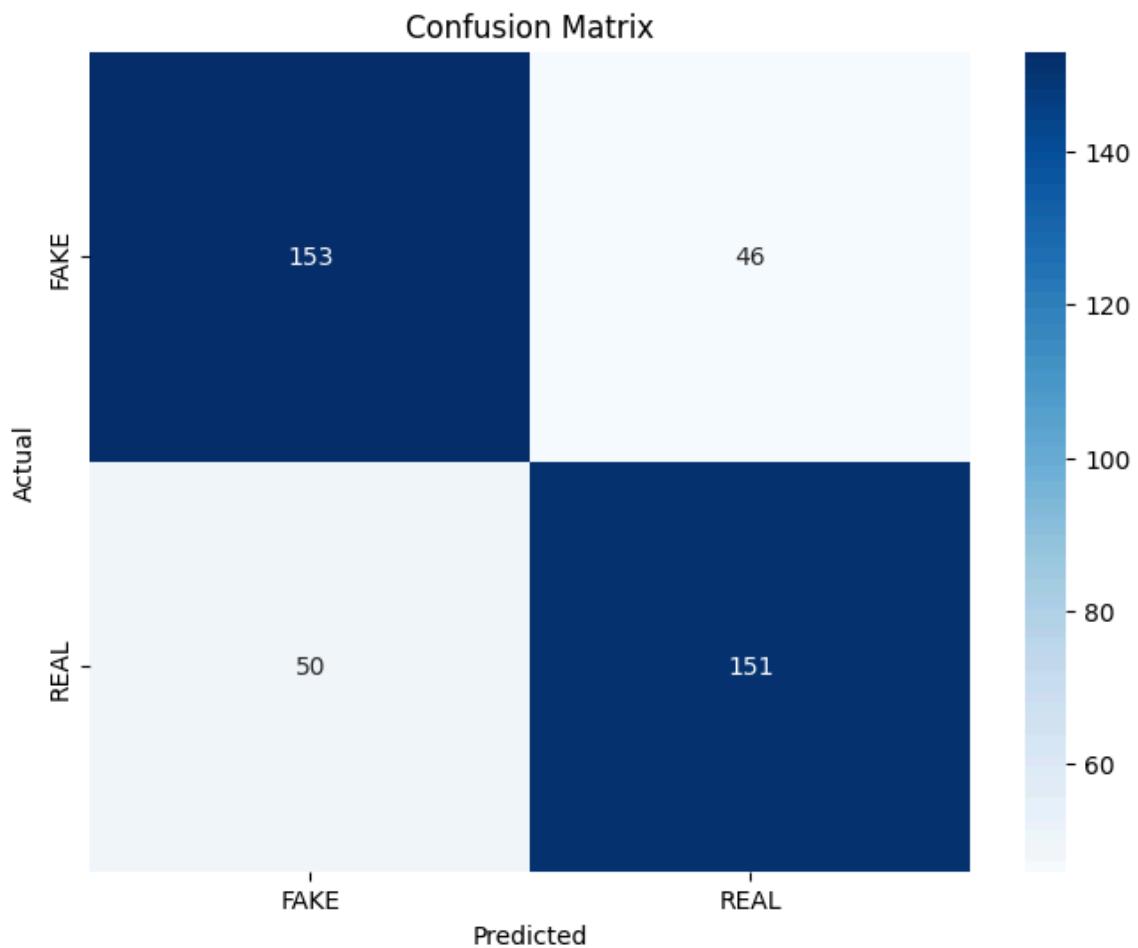


Figura 5.1: Matriz de confusión de predicción de artículos. Fuente: Juan Manuel Rodríguez 2024.

- Métricas de modelos clasificación de títulos antes de cross

	Modelo 1	Modelo 2	Modelo 3	Modelo 4
Accuracy	86.25%	80.00%	88.75%	92.50%
Recall	100.00%	97.22%	100.00%	100.00%
Precision	76.60%	70.00%	80.00%	85.71%
F1	86.75%	81.40%	88.89%	92.31%

Cuadro 5.2: Tabla comparativa de modelos sin aplicar técnicas de validación

Se utilizaron 80 títulos para el conjunto de prueba, los cuales fueron clasificadas de la siguiente forma por MultinomialNB alcanzando una precisión del 92.5%:

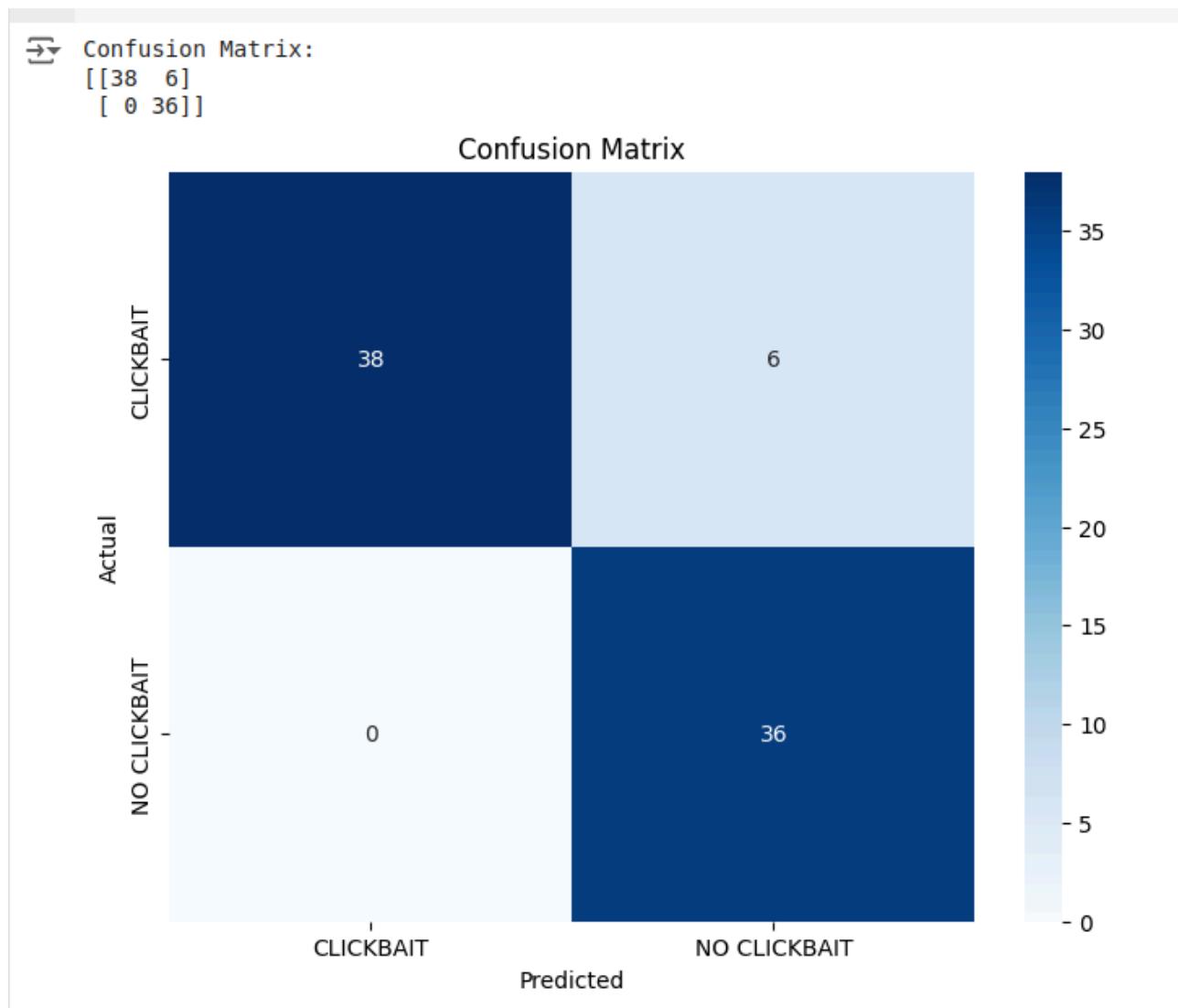


Figura 5.2: Matriz de confusión de predicción de títulos. Fuente: Juan Manuel Rodríguez 2024.

```
te Error (MAE): {mae:.3f}' )
d Error (MSE): {mse:.3f}' )
quared Error (RMSE): {rmse:.3
te Percentage Error (MAPE): {

(MAE): 4.430
(MSE): 30.287
rror (RMSE): 5.503
ntage Error (MAPE): 1.536%
```

ed,rain_1h,weather_main

Validación cruzada (Cross Validation)

La validación cruzada es una técnica utilizada para evaluar el rendimiento de los modelos de aprendizaje automático mediante diferentes subconjuntos de los datos disponibles. Este proceso implica dividir el conjunto de datos en varios intervalos (o "folds"). Cada modelo se entrena en todos los intervalos excepto uno, y luego se evalúa en el intervalo excluido. Este procedimiento se repite de tal manera que cada intervalo ha sido usado una vez como conjunto de evaluación.

La validación cruzada permite obtener una estimación más precisa del rendimiento del modelo en datos no vistos, lo que mejora la robustez y generalización de los modelos (Kohavi, 1995). Una vez aplicada esta técnica, se calculan las métricas de rendimiento (como la precisión, la sensibilidad, etc.) para obtener una estimación más fiable de cómo se comportará el modelo en la práctica. A continuación, presentamos un fragmento de código para realizar una validación cruzada sobre uno de los modelos clasificadores:

```

from sklearn.model_selection import cross_validate
from sklearn.metrics import make_scorer, precision_score, recall_score, f1_score

# Define your custom scoring metrics
scoring = {
    'accuracy': 'accuracy',
    'precision': 'precision',
    'recall': 'recall',
    'f1': 'f1'
}

# Perform cross-validation with multiple metrics
scores = cross_validate(clf, X_train_dtm, y_train, cv=5, scoring=scoring)

# Print the results for each metric
print("Cross-validation accuracy scores:", scores['test_accuracy'])
print("Mean accuracy: %.2f%%" % (scores['test_accuracy'].mean() * 100.0))

print("Cross-validation precision scores:", scores['test_precision'])
print("Mean precision: %.2f%%" % (scores['test_precision'].mean() * 100.0))

print("Cross-validation recall scores:", scores['test_recall'])
print("Mean recall: %.2f%%" % (scores['test_recall'].mean() * 100.0))

print("Cross-validation F1 scores:", scores['test_f1'])
print("Mean F1 score: %.2f%%" % (scores['test_f1'].mean() * 100.0))

```

Figura 5.3: Código de Cross validation. Fuente: Juan Manuel Rodríguez 2024.

- Modelos de clasificación de artículos después de cross

	Modelo 1	Modelo 2	Modelo 3	Modelo 4
Accuracy	71.75%	67.44%	75.50%	77.69%
Recall	83.60%	72.84%	75.97%	75.84%
Precision	67.55%	65.66%	75.26%	78.73%
F1	74.71%	69.04%	75.59%	77.25%

Cuadro 5.3: Tabla comparativa de modelos con cross validation

- Modelos de clasificación de títulos después de cross

	Modelo 1	Modelo 2	Modelo 3	Modelo 4
Accuracy	89.38%	85.62%	91.25%	90.00%
Recall	98.18%	97.58%	96.36%	89.05%
Precision	83.90%	79.31%	87.78%	91.30%

F1	90.42%	87.46%	91.85%	90.05%
-----------	---------------	---------------	---------------	---------------

Cuadro 5.4: Tabla comparativa de modelos con cross validation

Resultados de fase de evaluación:

Resultados del add-on:

Nuestra extensión es capaz de trabajar sobre periódicos como El Español y El País:

Al hacer click en el ícono del add-on, al cabo de 1 segundo, aparecerán las respuestas del análisis.

Análisis de noticia del diario El Español:

The screenshot shows a web browser window with several tabs open. The active tab is for the news article "Irán confirma la muerte de su presidente tras estrellarse el helicóptero donde viajaba" from [elespanol.com](https://www.elespanol.com). The page includes a sidebar with advertisements for "LBX" cars and a "Haz tu mundo extraordinario" campaign. The main content area features a large image of a helicopter wreckage in a forest. The analysis results on the right side are as follows:

- Title Analysis** (Green box):
 - El análisis del título determina que no es clickbait
- Article Analysis** (Red box):
 - El análisis del artículo determina que la noticia es falsa
- Sources Analysis** (Green box):
 - El análisis de fuentes ha encontrado otras fuentes que han reportado esta noticia
 - [Elconfidencial.com](#)
 - [Eldiario.es](#)

Figura 5.4: Resultados del add-on sobre periódico El Español. Fuente: (El Español., 2024).

<https://www.elespanol.com/>

En la siguiente imagen, podemos observar los mensajes que muestra la popup de nuestra extensión.

Consola de la extensión:

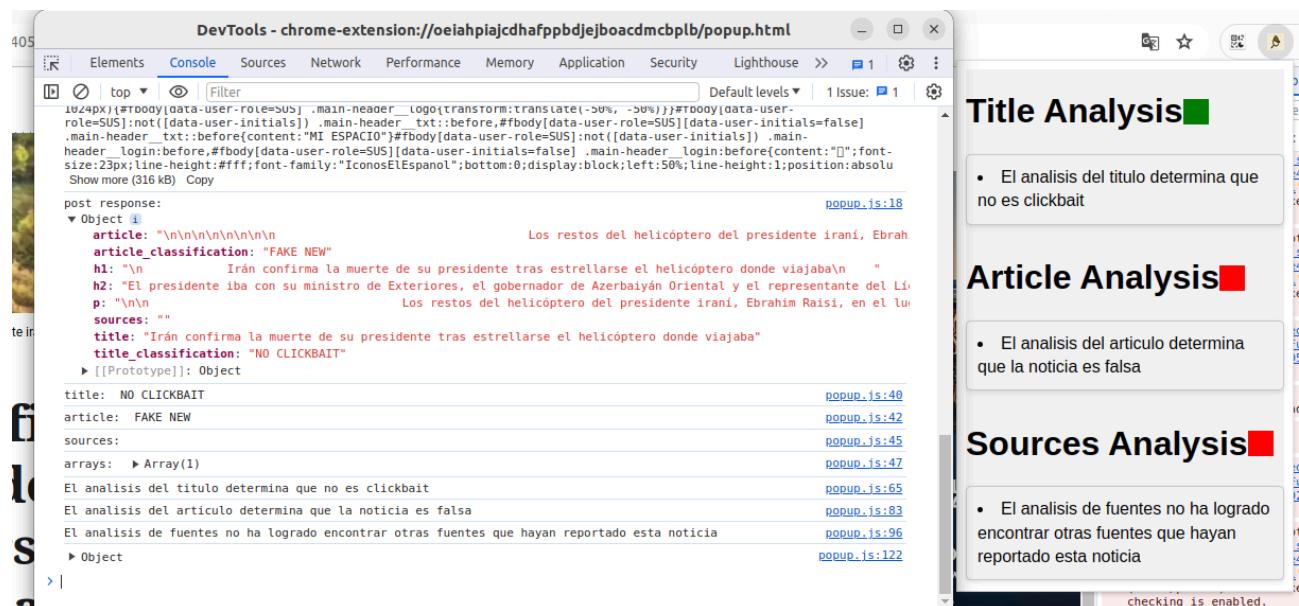


Figura 5.5: Funcionamiento del add-on sobre periódico El Español. Fuente: (El Español., 2024).

<https://www.elespanol.com/>

Hemos generado una serie de logs para explicar el funcionamiento de la popup. Lo primero que se observa es el html de la página que contendrá todo el documento. El html que vemos impreso, es el capturado por el script content.js, que luego lo envía al script popup.js mediante background.js para pintar popup.html.

En segundo lugar, podemos ver la respuesta que devuelve la API a la petición que le envía la extensión. Es importante notar que el artículo contenido se extrae tanto de p como de article, sin embargo, el elemento <p> prevalece por encima de <article> durante el análisis. Por este motivo, nuestro modelo genera una predicción en base al contenido del atributo p.

Finalmente, nuestro análisis arroja que el título “Irán confirma la muerte de su presidente tras estrellarse el helicóptero donde viajaba” no es clickbait. No obstante, nos indica que el contenido del artículo parece ser falso y que no ha encontrado otras fuentes que hayan publicado títulos similares.

Análisis de noticia del diario El País



Figura 5.6: Funcionamiento del add-on sobre periódico. Fuente: (El País, 2024)

<https://elpais.com/>

En este caso, nuestro análisis arroja que el título “La izquierda venció a la pantalla chica” no es clickbait. Además, nos indica que el contenido del artículo parece ser real y que ha encontrado una fuente llamada Aristeguinoticias.com que ha publicado un título similar.

6. Conclusiones

La clasificación de noticias en tiempo real es una tarea compleja que requiere un enfoque multifacético. Si se aborda desde una única perspectiva, los resultados obtenidos pueden estar sesgados. En este proyecto, se ha utilizado exitosamente el machine learning para ofrecer indicios sobre la fiabilidad de las noticias. Sin embargo, para que este producto tenga un impacto significativo en la sociedad, es necesario aumentar la **precisión** de las clasificaciones mediante la incorporación de un mayor número de recursos de mayor calidad.

Es crucial que un clasificador de noticias sea lo más fiable posible para combatir efectivamente la desinformación. A pesar de los avances logrados, el análisis realizado no es completamente exhaustivo. Aún quedan áreas por explorar, como el **análisis de las imágenes** asociadas a las noticias. Este aspecto requerirá la aplicación de técnicas de machine learning adicionales, tales como la clasificación de imágenes y la conversión de imágenes en textos.

Hemos logrado analizar aspectos clave de una noticia, como el título y el contenido, proporcionando un análisis detallado al usuario. Además, se ha diseñado una arquitectura de software que opera a una velocidad satisfactoria. En consecuencia, consideramos que este producto es útil, ya que ofrece respuestas sobre si el título es clickbait, si la noticia es falsa y si ha sido publicada por otras fuentes en menos de 2.5 segundos desde que se carga la página.

Los principales objetivos previstos para este estudio han sido abordados gracias a los conocimientos adquiridos a lo largo de la carrera, destacando en áreas como programación Full Stack, Sistemas Operativos e Inteligencia Artificial. No obstante, no se ha realizado un **análisis probabilístico** que permita al usuario ver una estimación de la probabilidad de que una noticia sea falsa o verídica.

El objetivo principal de este proyecto era desarrollar un add-on para Google Chrome que pudiera detectar noticias falsas en tiempo real, y en gran medida, este objetivo se ha cumplido. Sin embargo, hay varias áreas en las que el proyecto puede y debe mejorarse para aumentar su eficacia y utilidad.

1. **Precisión del Clasificador:** Aunque se ha logrado una precisión aceptable utilizando machine learning, el sistema todavía puede beneficiarse de más datos de entrenamiento y

- de mayor calidad. Incorporar fuentes adicionales y mejorar la diversidad de los datos podría reducir los sesgos y aumentar la fiabilidad del clasificador.
2. **Análisis de imágenes:** Actualmente, el sistema se centra en el análisis textual. Incluir el análisis de imágenes y otros tipos de contenido multimedia asociados a las noticias podría proporcionar una evaluación más completa y precisa de la veracidad de una noticia. Esto requerirá implementar técnicas avanzadas de machine learning como la clasificación de imágenes y el análisis de contexto visual.
 3. **Evaluación Probabilística:** La implementación de un análisis probabilístico permitiría al usuario ver una estimación cuantitativa de la veracidad de una noticia. Este enfoque no solo aumentaría la transparencia del sistema, sino que también permitiría a los usuarios tomar decisiones más informadas basadas en las probabilidades presentadas.
 4. **Interfaz de Usuario y Experiencia de Usuario:** Mejorar la interfaz de usuario y la experiencia de usuario (UX) es crucial para asegurar que el add-on sea accesible y fácil de usar. Una interfaz intuitiva y clara puede aumentar significativamente la adopción y el impacto del producto.
 5. **Escalabilidad y Desempeño:** Asegurar que el sistema sea escalable y capaz de manejar un gran volumen de solicitudes en tiempo real es fundamental. La arquitectura actual es rápida, pero deberá ser evaluada y posiblemente optimizada para mantener el rendimiento con un número creciente de usuarios.

En resumen, aunque los objetivos iniciales del proyecto se han alcanzado en gran medida, la detección de noticias falsas en tiempo real sigue siendo un campo en desarrollo que ofrece múltiples oportunidades para mejoras y expansión. Las futuras versiones del proyecto deberían centrarse en aumentar la precisión, ampliar el análisis a diferentes tipos de contenido, implementar evaluaciones probabilísticas y mejorar la experiencia del usuario para maximizar el impacto social del producto.

6.1. Valoración personal

Las entregas, en la universidad, suelen significar seguir un guión representado por rígidos pasos. El presente trabajo ha significado una oportunidad de mostrar nuestra impronta personal. Además, nos otorgó la posibilidad de medirnos contra un objetivo a largo plazo plagado de requisitos técnicos y teóricos. Es el trabajo académico al que dedicamos el mayor número de horas y esfuerzo.

Las fases iniciales relacionadas con la investigación fueron las más arduas y vinieron aparejadas con momentos de dudas. A decir verdad, cuando se elige un tema de TFG es, en muchos casos,

análogo a querer emprender algo que llama nuestra atención pero de lo que se desconoce. El avance de las fases, paulatinamente, se fue amenizando y tomando un rumbo hacia lo conocido.

Al principio existen dudas sobre qué tópico elegir para una entrega de gran importancia como esta. Los cuestionamientos continúan incluso después de la elección del tema. De todos modos, visto en retrospectiva, la decisión de desarrollar este producto fue acertada considerando el sinfín de posibilidades que existían. A grandes rasgos, considero que habríamos llegado a compenetrarnos de igual modo con cualquier tópico que hubiéramos elegido como trabajo de fin de grado.

Finalmente, al ver los proyectos terminados, se siente un grado de satisfacción proporcional a la dedicación invertida. Por otra parte, es sabido que siempre quedan puntos por cubrir en las primeras versiones de cualquier proyecto de software. Los aprendizajes adquiridos nos permitirán mejorar las próximas versiones de los productos o las nuevas de los que estén por venir.

6.2. Líneas futuras

- Uno de los principales objetivos del proyecto es facilitar al lector de noticias una herramienta con la que sentirse más seguro de estar consumiendo información genuina. Un producto con el que pueda analizar aquellos artículos periodísticos que atrapan su atención mientras navega por la web. Para alcanzar esta meta, deberíamos generar una aplicación lo suficientemente rápida y robusta como para que pueda ser utilizada por muchas personas en simultáneo. Esto, implica conseguir servidores para atender un gran número de peticiones.
- Es menester mejorar la precisión de los modelos mediante la incorporación de nuevas noticias de una mayor cantidad de fuentes.
- Otra mejora futura sería ser capaces de analizar noticias en diferentes idiomas. De este modo, el add-on no se limitaría solo a los que leen en español, y así podría alcanzar un mayor número de usuarios.
- Entrenar a los modelos con conjuntos generados con Word Embedding.

- Analizar imágenes asociadas a las noticias mediante diferentes estrategias que aporten conocimientos sobre la veracidad de la misma.
- Se debería desarrollar un modelo que actúe antes de los clasificadores y que identifique si el texto que el usuario está visualizando se trata efectivamente de una noticia y merece ser analizada, o se trata de otro tipo de texto. Esto, permitiría generar un mecanismo para reducir el número de llamadas que la extensión realiza a la API.
- Subsanar los errores de CORS para poder realizar llamadas a API's en servidores externos desde la extensión.
- Una posible mejora consiste en personalizar la búsqueda de elementos html dentro de la página dependiendo de la página de noticias de la que se trate. Es decir, si sabemos que la página contiene el cuerpo de la noticia bajo una etiqueta concreta, podríamos extraer esa etiqueta sin indagar en las otras. Esto, también podría realizarse con un modelo de clasificación que busque el título y el cuerpo de la noticia dentro de los elementos web extraídos.
- Uno de los aspectos más cruciales a cubrir para que el producto alcance al consumidor final, es conocer qué consentimientos debería solicitar (la extensión) al usuario. Poder manipular los contenidos periodísticos que una persona visualiza y enviarlos a una API externa implica una gran intromisión en la navegación del usuario. estrictamente necesario
- En caso de alcanzar a obtener los consentimientos del usuario, se podrían recoger datos de las noticias que analizan y empezar a generar métricas asociadas a cada página web de noticias para realizar un estudio preliminar de la fiabilidad que tienen. También, las noticias analizadas podrían entrar en un historial sobre el cual se podría realizar algún proceso de verificación de resultados cada cierto tiempo para poder .

Bibliografía

- Ahmad, I., et al. (2020). Experimental evaluation of clickbait detection using machine learning models. ResearchGate.
https://www.researchgate.net/publication/348877404_Experimental_Evaluation_of_Clickbait_Detection_Using_Machine_Learning_Models
- Alir3z4. (2021). Stop Words en Español.
<https://github.com/Alir3z4/stop-words/blob/master/spanish.txt>
- Arrauda, F., & Covoes, F. (2020). Fake news detection in multiple platforms and languages. ScienceDirect. <https://www.sciencedirect.com/science/article/abs/pii/S0957417420303274>
- Amazon Web Services. (2024). ¿Qué es una API?. <https://aws.amazon.com/es/what-is/api/>
- Barbosa, C., Souza, Freire., Ribeiro, Goldschmidt., & Justel, M. (2024). Early detection of fake news on virtual social networks: A time-aware approach based on crowd signals. ScienceDirect. <https://www.sciencedirect.com/science/article/abs/pii/S095741742400215X>
- Buntain, G., & Goldbeck, J. (2017). Automatic detection of fake news. arXiv.
<https://arxiv.org/pdf/1705.01613>
- Business Insider España. (2018). Facebook se desploma en bolsa tras una filtración masiva de datos por Cambridge Analytica.
<https://www.businessinsider.es/facebook-desploma-bolsa-filtracion-masiva-datos-cambridge-analytica-196320>
- Chrome for Developers. (2024). Chrome Runtime.
<https://developer.chrome.com/docs/extensions/reference/api/runtime>
- Crispo, G. (2023). Experiment: Fake News Detection in Browser.
<https://gioelecrispo.github.io/blog/experiment-fake-news-detection-in-browser>
- DataReportal. (2024). Digital 2024: Deep Dive — The Time We Spend on Social Media.
<https://datareportal.com/reports/digital-2024-deep-dive-the-time-we-spend-on-social-media>
- Delta, I. A. (2023). Regresión logística y Softmax.
<https://iadelta.com/inteligencia-artificial/regresion/logistica-y-softmax/>
- Developer Chrome. (2023). Extensions Develop: Migrate.
<https://developer.chrome.com/docs/extensions/develop/migrate>
- Developer Mozilla. (2024). ¿Qué son las WebExtensions?.
https://developer.mozilla.org/es/docs/Mozilla/Add-ons/WebExtensions/What_are_WebExtensions

- Escamilla, J. A., García, J. C., Huitrón, D., Vega, R., & Cruz, S. (2019). El Web Scraping: Obtención de Información Mediante la Exploración de Sitios Web. SABER.
<https://www.saber.cic.ipn.mx/SABERv3/Repositorys/webVerArchivo/50980/1>
- Garcia, F., Diagiampietri, L., & Trevisan, R. (2024). The use of syntactic information in fake news detection: A systematic review. ResearchGate.
https://www.researchgate.net/publication/379354712_The_Use_of_Syntactic_Information_in_fake_news_detection_A_Systematic_Review
- GeeksforGeeks. (2023). Understanding TF-IDF: Term Frequency-Inverse Document Frequency.
<https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/>
- Glassdoor. (2024). Sueldos en España. <https://www.glassdoor.es/Sueldos/index.htm>
- IBM. (2023). Deep learning: An introduction.
<https://www.ibm.com/es-es/topics/deep-learning>
- Instituto de Ingeniería del Conocimiento. (2017). ¿Qué es el procesamiento del lenguaje natural?. <https://www.iic.uam.es/inteligencia/que-es-procesamiento-del-lenguaje-natural/>
- Jawad, A., Feizi, D., & Salehpour, P. (2023). Enhancing fake news detection by multi-feature classification. ResearchGate.
https://www.researchgate.net/publication/376247075_Enhancing_Fake_News_Detection_by_Multi-Feature_Classification
- Kinsta. (2023). ¿Qué es el web scraping? Cómo extraer legalmente el contenido de la web.
<https://kinsta.com/es/base-de-conocimiento/que-es-web-scraping/>
- Looijenga, R. (2018). Fake news on Twitter: A quantitative content analysis of fake news messages on Twitter preceding the 2012 Dutch elections. University of Twente.
https://essay.utwente.nl/77385/1/Looijenga_BA_EEMCS.pdf
- Master Data Scientist. (2018). Text mining and natural language processing.
<https://www.master-data-scientist.com/text-mining-natural-language-processing/>
- Neptune.ai. (2022). Pickle. <https://neptune.ai/blog/saving-trained-model-in-python>
- Phan, T., Nguyen, T., & Hwang, D. (2023). Fake news detection: A survey of graph neural network methods. ScienceDirect.
<https://www.sciencedirect.com/science/article/pii/S1568494623002533>

- Pujahari, B., & Sisodia, S. (2020). Clickbait detection using multiple categorization techniques. arXiv. <https://arxiv.org/pdf/2003.12961>
- Richardson, L. (2021). Beautiful Soup Documentation. <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>
- Roastbrief. (2022, August). 82% de la población mundial consume noticias por Internet: Digital 2022. <https://roastbrief.com.mx/2022/08/82-de-la-poblacion-mundial-consume-noticias-por-internet-digital-2022/>
- Real Academia Española. (2023). Noticia. <https://dle.rae.es/noticia>
- Reuters Institute for the Study of Journalism. (2023). Digital News Report 2023: Executive Summary. <https://reutersinstitute.politics.ox.ac.uk/digital-news-report/2023/dnr-executive-summary>
- Scikit-learn. (2024). sklearn.model_selection.train_test_split. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
- Sverdhei, M. (2023). Clickbait title classification. Hugging Face. https://huggingface.co/datasets/marksverdhei/clickbait_title_classification
- Tableau. (2023). Natural Language Processing Examples. <https://www.tableau.com/learn/articles/natural-language-processing-examples>
- Towards Data Science. (2020). A Guide to Word Embeddings. <https://towardsdatascience.com/a-guide-to-word-embeddings-8a23817ab60f>
- Tretiakov, A. (2023). Noticias falsas en español. Kaggle. <https://www.kaggle.com/datasets/arseniitretiakov/noticias-falsas-en-espaol>
- Truică, C., Apostol, E., & Karras, P. (2023). Deep neural network ensemble architecture for social and textual context-aware fake news detection. arXiv. <https://arxiv.org/abs/2302.01756>
- Wikipedia. (2024). Teorema de Bayes. https://es.wikipedia.org/wiki/Teorema_de_Bayes

ANEXOS

- **Anexo 1: Extensión TFG**

Repositorio: https://github.com/juanmata8/Extension_TFG

- **Anexo 2: Flask API TFG**

Repositorio: https://github.com/juanmata8/Flask_API_TFG

- **Anexo 3: Title Classifiers TFG**

Repositorio: https://github.com/juanmata8/title_classifiers_TFG

- **Anexo 4: Article Classifiers TFG**

Repositorio: https://github.com/juanmata8/article_classifiers_TFG

