TRABAJOS FIN DE GRADO

Universidad de Jaén

Escuela Politécnica
Superior de Jaén

Prototipo de Monitor de Detección de Lenguaje Ofensivo

Autor: Francisco Cano Cubillo

Grado: Ingeniería Informática

Directores: Eugenio Martínez Cámara, Luis Alfonso Ureña López
Departamento de los directores: Informática

Fecha: 24/06/2024

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Eugenio Martínez Cámara y Don Luis Alfonso Ureña López, tutores del Proyecto Fin de Carrera titulado: Prototipo de Monitor de Detección de Lenguaje Ofensivo, que presenta Francisco Cano Cubillo, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, junio de 2024

El alumno:

CANO CUBILLO FRANCISCO - 77690139A
Firmado digitalmente por CANO CUBILLO FRANCISCO - 77690139A
Fecha: 2024.06.24 18:02:01 +02'00'

Francisco Cano Cubillo

Los tutores:

Firmado digitalmente por MARTINEZ CAMARA EUGENIO - 77355503V
Fecha: 2024.06.24 20:07:59 +02'00'

Eugenio Martínez Cámara

UREÑA LOPEZ LUIS ALFONSO - 52552054K
Firmado digitalmente por UREÑA LOPEZ LUIS ALFONSO - 52552054K
Fecha: 2024.06.24 22:19:41 +02'00'

Luis Alfonso Ureña López

Agradecimientos

A mis padres y mi hermana, por nunca parar de creer en mí y de apoyarme, aun cuando ni yo mismo podía hacerlo. Por ser el impulso y la motivación para levantarme tras cada caída.

A Zuri, por amarme sin importar el momento ni el lugar. Por hacerme sentir la persona más única, especial y afortunada del mundo estando a tu lado, porque cada momento contigo se convierte en un recuerdo amarillo radiante.

A Eugenio y a Alfonso, por ser los *debuggeadores* de este proyecto y asegurarse de que consiga llegar al *return 0* del final del *main*.

A Carmen, por estar en primera fila para verme crecer, celebrar todos mis éxitos y presumir de ello. Por seguir siempre uno junto al otro, a pesar de todos los males que hemos tenido que soportar.

A Juanjo y a Nati, por ser la alegría personificada, porque cada día con vosotros se graba en el corazón y me hace tener resaca emocional al día siguiente.

Índice

1. Introducción

En la actualidad, debido al crecimiento del número de redes sociales y al de su uso, es necesario aplacar uno de los problemas más urgentes que las envuelve: el discurso de odio en Internet. En estas circunstancias, la implementación de un detector de lenguaje ofensivo emerge como una respuesta crítica para combatir esta tendencia tan preocupante.

Las redes sociales, siendo lugares donde ocurren millones y millones de interacciones entre usuarios al día, han democratizado el intercambio de ideas a un nivel nunca visto en la historia. Como parece obvio, este acceso aparentemente sin control ha producido un aumento de comportamientos negativos, nocivos y perjudiciales, incluyendo la difusión de discursos de odio y el acoso cibernético *(ciberbullying).*

Una de las consecuencias más alarmantes de este auge es el impacto negativo en la salud mental y el bienestar emocional de las víctimas. El ciberacoso y la exposición permanente a contenido hiriente pueden producir secuelas asoladoras e irreversibles, que abarcan desde la ansiedad y la depresión hasta el suicidio. Para evitarlo, es fundamental y urgente crear espacios seguros, positivos y constructivos con el objetivo de proteger la integridad psicológica de los individuos.

Del mismo modo, el filtrado y acordonamiento de contenido ofensivo e impulsar la creación de entornos positivos y constructivos hacen que afloren espacios en los medios digitales donde la inmensa variedad de opiniones puede ser expresada de manera respetuosa y sin represalias. Con estos ambientes, no únicamente mejora la satisfacción del usuario, sino que se fomenta una cultura mucho más inclusiva y empática.

1.1. Hipótesis de Investigación

Se estima que, mediante la implementación de un monitor detector de lenguaje ofensivo, se logrará brindar una ayuda para filtrarlos o encontrar a sus autores para, en ese caso, conseguir una reducción de la cantidad de lenguaje ofensivo en los entornos de las redes sociales.

En primer lugar, se antepone que la capacidad para detectar los mensajes permitirá una acción rápida de los administradores de las plataformas, los cuales podrán tomar medidas para moderar el contenido ofensivo y emprender las acciones correspondientes, antes de que el alcance del contenido ofensivo sea mayor.

Además, como se ha expuesto anteriormente, el filtrado de este tipo de temas de manera eficaz y eficiente reforzará los entornos en línea positivos y respetuosos. De esta forma, se espera conseguir un efecto de contagio positivo, en el cual se cree un círculo virtuoso que fomente una cultura digital más inclusiva, respetuosa.

1.2. Objetivo

El propósito de este proyecto es el diseño e implementación de un prototipo de monitor de detección de lenguaje ofensivo para moderar la ascendente problemática de los discursos de odio y el ciberacoso en medios digitales. Este prototipo está concebido como una primera iteración de un sistema más completo de detección y moderación de contenido ofensivo.

El trabajo se enfocará en el desarrollo de un prototipo capacitado para la detección del lenguaje ofensivo, tanto en fragmentos de texto introducidos por el usuario, como en redes sociales. Para ello, se utilizarán técnicas de Procesamiento del Lenguaje Natural (PLN) para el entrenamiento del modelo en detección de patrones lingüísticos que contengan alguna relación con el lenguaje ofensivo. Estos modelos serán sometidos a un estudio experimental en el cual se evaluarán su rendimiento, siguiendo criterios predefinidos.

El objetivo principal del citado estudio es comprobar si, mediante la combinación de distintos conjuntos de datos en un mismo modelo, se logra mejorar el rendimiento y la capacidad de detección de lenguaje ofensivo del modelo. Incluso se pretende ir un paso más allá buscando la detección de subtipos específicos de odio, como el sexista, de forma que el modelo sea capaz de detectar esa ofensividad hasta en los comentarios más sutiles que pasan desapercibidos por otros modelos.

Posteriormente, los mejores con mayor desempeño en el estudio serán embebidos en una aplicación web, en la que los usuarios podrán introducir texto o

contenido de redes sociales para su análisis. Tras el análisis, se mostrarán los resultados de la detección de lenguaje ofensivo realizada por los modelos.

En resumen, el objetivo del proyecto se fundamenta en crear un prototipado inicial, funcional y estable de un monitor de detección de lenguaje ofensivo, mediante la generación de modelos, su integración en una aplicación y su ensayo en entornos reales. A través de este proceso, se intenta demostrar la viabilidad y utilidad de la detección de lenguaje ofensivo.

1.3. Estructura

El capítulo actual trata de mostrar una visión global y detallada de la organización y del contenido que presenta este Trabajo de Fin de Grado. Para ello, a continuación, se explica el contenido de cada uno de los cuatro capítulos que componen este trabajo, excluyendo al actual.

Para empezar, estará el capítulo 2. Estado del Arte, un apartado de investigación previa para ahondar y alcanzar un grado de comprensión mayor en el campo del PLN, centrándose en técnicas y algoritmos relevantes para la detección de lenguaje ofensivo.

A continuación, se encuentra la sección de 3. Análisis, dedicada a detallar los requisitos, ámbito y estructura de la aplicación propuesta en este trabajo. Se describen las funcionalidades, los casos de uso previstos y las consideraciones técnicas y de diseño.

En el siguiente capítulo, 4. Experimentación y Evaluación de Modelos, se expone la metodología empleada para entrenar y evaluar distintos modelos de detección de lenguaje ofensivo. Se detallan los conjuntos de datos utilizados, las métricas consideradas para la evaluación y los procedimientos llevados a cabo. Se analizan los resultados devueltos por los modelos para seleccionar los mejores e integrarlos en la aplicación.

Seguidamente se encuentra el capítulo 5. Diseño de la aplicación, que describe el diseño de la aplicación a desarrollar. Este diseño incluye la estructura de clases que

contiene el proyecto, la estrategia para desplegar la aplicación en Internet y la descripción detallada de la interfaz de usuario de cada una de las páginas web.

Acto seguido, se trata la parte técnica del proyecto en 6. Implementación del proyecto, donde se detalla la programación utilizada, tanto para la fase experimental, en la cual se entrenan y evalúan los distintos modelos, como la propia estructura del prototipo desarrollado. En él se explican las bibliotecas utilizadas, la organización del código y las funcionalidades de cada uno de los componentes.

Para terminar, se incluye un apartado de 7. Anexos, en el cual se incluyen un manual de instalación y otro de usuario con instrucciones detalladas para la instalación y el uso de la aplicación. En esta sección se explica cómo configurar y obtener la aplicación en el equipo del usuario, así como una guía completa de uso, explicando las funcionalidades disponibles, el flujo de trabajo y las mejores prácticas para optimizar el rendimiento de la herramienta.

2. Estado del Arte

En el presente capítulo se plasma el análisis previo realizado para adquirir los conocimientos necesarios para realizar este proyecto. Este apartado proporciona un contexto fundamental para comprender el estado actual del PLN y los avances en la detección y mitigación del lenguaje ofensivo en entornos digitales.

2.1. Procesamiento del Lenguaje Natural

El lenguaje ha sido, desde el inicio de los tiempos, la herramienta imprescindible de comunicación entre humanos. Es por ello por lo que, en los últimos años, se ha convertido en un campo de estudio que abarca numerosas áreas como la lingüística, informática, la psicología o la inteligencia artificial, entre otros (Balayn et al., 2021). Tanto ha sido el caso de este último, que el objetivo de esta disciplina ha sido el diseño de algoritmos que conviertan a las máquinas actuales en agentes independientes capaces de resolver, de forma solvente, situaciones cotidianas en las que las personas nos vemos involucradas. Una de las tareas en la que es crucial e indispensable que estos agentes sean resolutivos y eficientes es en la de comunicación con otros seres humanos.

Es con este objetivo que nace el PLN, un área de la inteligencia artificial que busca dotar a los ordenadores, a través del uso de algoritmos y métodos computacionales, la capacidad para entender, interpretar el lenguaje humano y generar una respuesta en el mismo lenguaje (Goldberg, 2022). Realmente, el PLN se podría considerar como la combinación de dos amplios subcampos: la Comprensión del Lenguaje Natural (CLN, por sus siglas) y la Generación de Lenguaje Natural (GLN, por sus siglas) (Pilehvar & Camacho-Collados, 2020).

La Comprensión del Lenguaje Natural se centra en ahondar en el lenguaje para entender el significado de este a través de la comunicación humana, ya sea a través de textos escritos, o de conversaciones habladas. Sin embargo, no está exenta de complicaciones, tales como la naturaleza ambigua del lenguaje. La ambigüedad puede manifestarse a distintos niveles, desde el nivel léxico hasta un nivel anafórico, lo que la convierte en uno de los mayores desafíos a la hora de entender el lenguaje humano. También tenemos el lenguaje figurado, tales como las metáforas o el

sarcasmo, que es usado por todas las personas en cualquier ámbito, hablado o escrito (Pilehvar & Camacho-Collados, 2020). Esta forma de comunicación se convierte en un verdadero reto para los algoritmos de comprensión porque, normalmente, el significado de las expresiones sarcásticas y metafóricas no suelen estar relacionadas con las palabras que conforman la propia expresión.

La Generación de Lenguaje Natural plantea el enfoque en la generación de textos por parte de un ordenador. En otras palabras, se trata de responder a los seres humanos usando el lenguaje natural (Pilehvar & Camacho-Collados, 2020). De la misma forma que el CLN, esta también tiene una serie de dificultades a la hora de generar las respuestas, tales como el extenso vocabulario para referirse a un mismo concepto, el orden dinámico de las palabras en una oración o la concordancia dentro de la oración.

Se han hallado numerosas aplicaciones para el PLN en diversas áreas. Por ejemplo, en el ámbito de las redes sociales, tenemos el análisis de opiniones, el cual es utilizado para comprender las opiniones de la gente. Por ejemplo, tenemos la detección del lenguaje ofensivo para ayudar a establecer un entorno seguro y respetuoso para los usuarios. Por otro lado, en el ámbito industrial, el PLN es aplicable en el servicio de atención al cliente para obtener una comprensión mayor de las necesidades de los consumidores, y en traducción, para derruir barreras lingüísticas. Además, en el ámbito médico, se utiliza también para analizar historiales médicos y proveer asistencia en los diagnósticos.

Aun con todos los avances realizados en este campo, el PLN sigue enfrentándose a grandes desafíos. La clave para combatir estos retos reside en mejorar la interpretación del contexto y la generación del lenguaje natural. También es importante mencionar la esencialidad de la ética, sobre todo en temas de privacidad y sesgos de datos, haciendo que los modelos que se desarrollen en el futuro sean los más equitativos y equivalentes posibles.

2.2. Detección de Lenguaje Ofensivo

La detección del lenguaje ofensivo es una subárea del PLN enfocada en identificar expresiones lingüísticas que se pueden llegar a considerar agresivas, discriminatorias, hirientes o inapropiadas.

Las técnicas empleadas en la detección del lenguaje ofensivo abarcan desde el uso de listados de palabras y reglas gramaticales hasta modelos de aprendizaje automático que han sido previamente entrenados con corpus etiquetados. Estos permiten identificar patrones lingüísticos que desembocan en expresiones y comportamientos que pueden resultar ofensivos o agresivos para un entorno seguro digital. Inicialmente, algunos autores propusieron un método consistente en dos fases: la primera consiste en obtener características léxico-semánticas utilizando técnicas de minería de datos y PLN. Durante la segunda fase, se añaden características de los usuarios mediante análisis de patrones de comportamiento (Molero et al., 2023). Este procedimiento permitió crear un conjunto de reglas sintácticas y palabras ofensivas para identificar el nivel de odio de una oración. A su vez, permitió medir el grado de toxicidad de un usuario agregándole al nivel de odio del historial de mensajes del usuario otras características del propio usuario, como la forma de escribir.

Con estas características reunidas, realizaron una serie de experimentos (Chen et al., 2012). En el primero de ellos, utilizaron tanto términos ofensivos obvios como sutiles, pero el resultado no fue tan positivo como utilizar solamente palabras ofensivas obvias. Tras ello, decidieron realizar un segundo experimento en el que solamente incluyeron las sutiles, con el sorprendente resultado de ser superior a las proposiciones anteriores. Esto derivó en la conclusión de que, a falta de términos que resultasen inequívocamente ofensivos, era esencial interpretar el contexto del texto para detectar la ofensividad del mensaje.

Actualmente, la gran mayoría de aplicaciones incluyen herramientas para la moderación de contenido, identificación de comportamientos de acoso o discriminación y la implementación de medidas preventivas para salvaguardar la seguridad y el confort de la comunidad. Estas medidas pueden ser aplicadas de manera automáticas por un sistema entrenado para detectar estos comportamientos, o bien, tras analizar los reportes de los usuarios indicando las razones o motivos por

los cuales consideran que algún comentario que se ha publicado en la aplicación es ofensivo (Balayn et al., 2021). Estos reportes también ayudan al sistema automático a aprender y descubrir nuevos matices que, desde el punto de vista humano, sí es ofensivo, pero que una máquina no puede llegar a captar. Esta área va muy ligada a otros ámbitos, como el análisis emocional y el análisis sentimental, ya que puede beneficiarse de ellos. Concretamente, ayuda a comprender el tono emocional detrás del mensaje que, normalmente, se pierde al estar escrito y no hablado. Gracias a esto, se pueden proporcionar respuestas más efectivas en la moderación de contenido en medios digitales.

Aunque se trate de un tema muy presente hoy en día, la detección de lenguaje ofensivo aun presenta diversas debilidades, tanto con los matices del discurso, como con la tarea de clasificación, lo que impide a los sistemas alcanzar los resultados óptimos que se desean. Uno de los principales desafíos se centra en la complejidad que conlleva definir el discurso de odio y la ambigüedad extendida en el uso de términos relacionados, tales como lenguaje abusivo, asertivo, tóxico o peligroso, ya que de forma frecuente se superponen y dan pie a interpretaciones altamente subjetivas (Poletto et al., 2021).

En conclusión, aunque se han logrado avances significativos en la detección del lenguaje ofensivo, todavía hoy en día persisten varios retos que limitan su eficacia. Algunos de ellos, como se ha comentado anteriormente, consisten en la complejidad para definir el discurso de odio o en la ambigüedad en el uso de términos relacionados. No obstante, gracias a la continua evolución de técnicas de PLN, junto a un enfoque en el contexto y la subjetividad humana, harán posible superar estas dificultades, haciendo que se mejore la precisión y efectividad de las herramientas de moderación de contenido en el futuro.

3. Análisis

En este capítulo, se realiza una exploración detallada de la base que se utilizará para el diseño y desarrollo de la aplicación propuesta en este TFG. Se empezará con un análisis completo de los requisitos del proyecto, detallando las necesidades y expectativas del usuario, así como los criterios de calidad y rendimiento que debe superar. A continuación, se presentarán los casos de uso que proporcionan una visión global de las funcionalidades y escenarios de uso de la aplicación. Además, se mostrarán también los diagramas de actividad, de secuencia y de clases de análisis, ofreciendo una representación visual de la estructura y el comportamiento del sistema, haciendo que la comprensión de los aspectos técnicos y funcionales del sistema sea más fácil. Por último, se expondrán consideraciones y reflexiones sobre ideas que podrían haber sido introducidas, pero debido a impedimentos fuera del alcance del proyecto, no se han podido incluir. Este análisis permite entender y justificar las decisiones tomadas durante el desarrollo del proyecto, así como posibles áreas de mejora o futuras investigaciones.

3.1. Análisis de requisitos

El análisis de requisitos es el proceso de comprender, definir y documentar las necesidades y características de un producto para cumplir los objetivos del proyecto que lo abarca. Este proceso es de gran importancia en el desarrollo de sistemas software, debido a que sienta las bases para el diseño, desarrollo, implementación y pruebas de este.

En este apartado, se especificarán los requisitos funcionales y no funcionales recabados para este TFG. Los requisitos funcionales son especificaciones detalladas que describen las funciones o acciones específicas de un sistema, es decir, definen qué tiene que hacer la aplicación. En cambio, los requisitos no funcionales especifican el comportamiento de la aplicación en términos de calidad, restricciones y limitaciones, en vez de describir las funciones del sistema.

3.1.1. Requisitos funcionales

- **Analizar texto insertado manualmente**:
  - Proporcionar una interfaz al usuario para permitir la entrada de datos en forma de texto para ser analizado.
- **Analizar comentarios extraídos de YouTube**:
  - Establecer una conexión con la API de YouTube para obtener y analizar los comentarios de un vídeo elegido por el usuario.
- **Visualización de resultados**:
  - Mostrar los resultados y las explicaciones de manera clara, concisa y comprensible para el usuario.
  - Acompañar los resultados con gráficos y tablas para condensar la información.

3.1.2. Requisitos no funcionales

- **Precisión:** Capacidad detectar y clasificar los mensajes entrantes con una tasa de error mínima.
- **Rendimiento:** Capacidad de procesar los mensajes eficientemente y en tiempo real.
- **Robustez y seguridad:** Garantizar la seguridad y privacidad de los datos y del usuario, mediante buenas técnicas de programación.
- **Interpretabilidad:** Proporcionar explicaciones claras y comprensibles para todo tipo de usuarios, desde los más noveles a usuarios expertos.
- **Compatibilidad:** Capacidad de obtener los datos de diferentes fuentes, como enlaces, redes sociales o ficheros de texto.
- **Documentación:** Proporcionar una documentación clara y comprensible para entender su uso y su mantenimiento.
- **Interfaz accesible:** Suministrar al usuario una interfaz con la que interactuar de forma que sea lo más accesible posible
- **Eficiencia:** Obtener una respuesta en el mínimo tiempo posible.

3.2. Casos de Uso

Los casos de uso describen, utilizando la forma de acciones y reacciones, el comportamiento que tiene un sistema desde el punto de vista del usuario, es decir, detalla la funcionalidad independientemente de la implementación desarrollada. Se documentan con una lista numerada de pasos que sigue el actor al interactuar con el sistema y con otra lista alternativa con errores que pueden aparecer durante la ejecución del caso de uso. Todos los pasos de las listas están escritos en lenguaje natural.

Para comprender y averiguar qué casos de uso se detallan en la aplicación, es imprescindible mostrar un diagrama de historias de caso. Con él se busca plasmar los servicios que el prototipo desarrollado debe brindarle al usuario final.

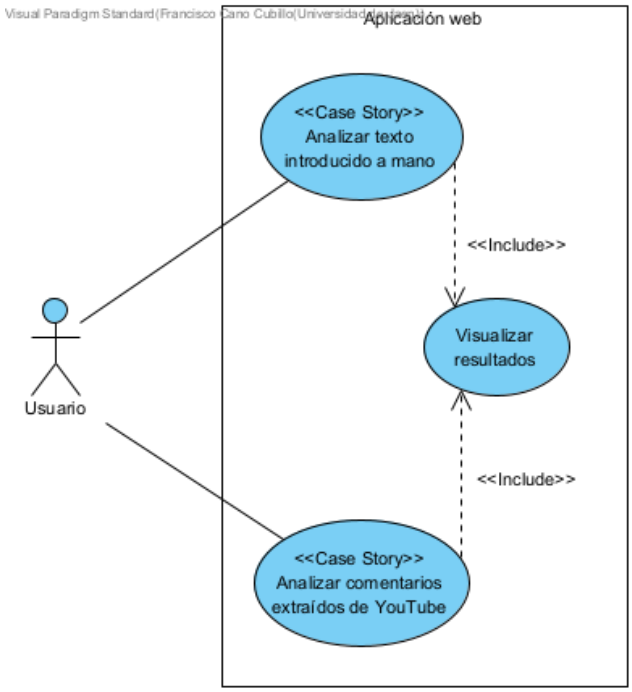


Figura 1 – Diagrama de historias de caso de la aplicación

En el caso del proyecto, el primer caso de uso contemplado () consiste en el análisis de mensajes tecleados por el propio usuario en el apartado dedicado para ello dentro de la aplicación.

Analizar texto insertado manualmente	
Descripción	Los usuarios pueden ingresar un mensaje cualquiera. Tras ello, el sistema de la aplicación recopila el contenido del mensaje, lo analiza y dictamina si es ofensivo o no, mostrando la respuesta al usuario.

Secuencia Normal	**Paso**	**Acción**
	1	El usuario ingresa un mensaje.
	2	El sistema analiza el contenido del mensaje.
	3	El sistema muestra al usuario el resultado para visualizarlo.
Secuencia Alternativa	**Paso**	**Acción**
	1A	Si el usuario introduce un mensaje vacío, devolver error "Mensaje vacío". Después, ir a 1.

Tabla 1 – Definición del caso de uso "*Analizar texto insertado manualmente*"

De forma gráfica, el caso de uso puede visualizarse siguiendo el siguiente diagrama:

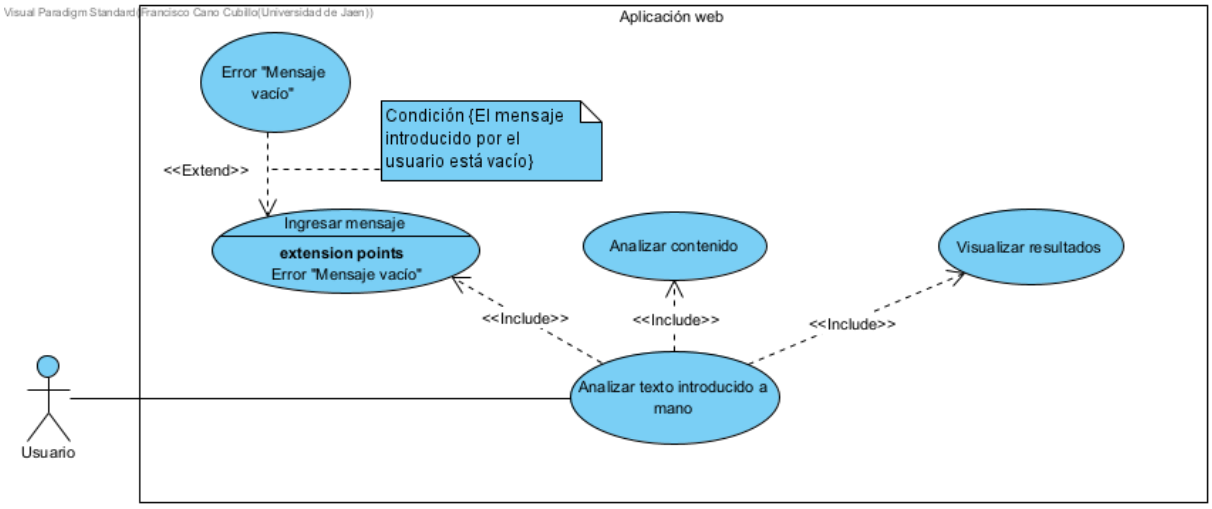


Figura 2 – Diagrama de caso de uso "*Analizar texto introducido a mano*"

Otro caso de uso que se aprecia se trata de la representación gráfica de un análisis de los comentarios de un vídeo de YouTube, el cual es indicado por el usuario con la URL del mismo.

Analizar comentarios extraídos de YouTube		
Descripción	Los usuarios pueden ingresar una URL de un vídeo de YouTube y seleccionar qué modelos analizarán los comentarios. Tras ello, el sistema dichos comentarios, realiza el análisis de todos ellos por cada modelo indicado y devuelve una representación gráfica del resultado.	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario ingresa una URL de un vídeo de YouTube.
	2	El usuario selecciona qué modelos analizarán los comentarios
	3	El sistema analiza los comentarios.
	4	El sistema muestra al usuario los resultados para visualizarlos.
	Paso	**Acción**

Secuencia Alternativa	1A	Si el usuario introduce una URL vacía o incorrecta, devolver error “URL inválida”. Después, ir a 1.
	2A	Si el usuario no selecciona ningún modelo para analizar los comentarios, devolver error “Modelos no seleccionados”. Después, volver a 2.

Tabla 2 – Definición del caso de uso “*Analizar comentarios de un vídeo de YouTube*”

Para comprender mejor la situación de este caso de uso, se puede contemplar, a continuación, de forma gráfica:

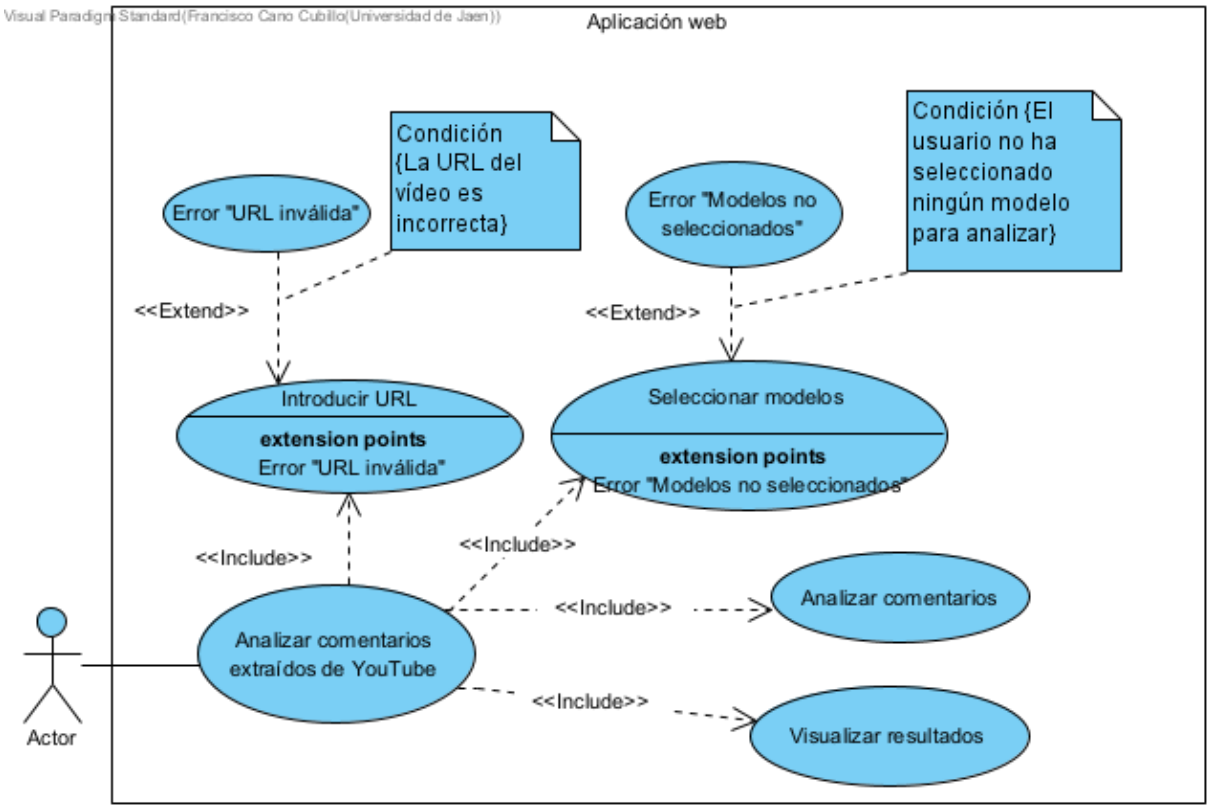


Figura 3 – Diagrama de caso de uso “*Analizar comentarios de YouTube*”

El último caso de uso contemplado en el proyecto es el correspondiente a la visualización de los resultados devueltos por el análisis.

Visualizar resultados		
Descripción	Una vez el sistema ya ha analizado el contenido que el usuario quiere que se clasifique, se prepara la información para establecerla en un formato específico y es devuelta al usuario para su interpretación.	
Secuencia Normal	**Paso**	**Acción**
	1	El sistema formatea los resultados para una configuración específica.

	2	El sistema devuelve la respuesta al usuario en el formato correspondiente.

Tabla 3 – Definición del caso de uso "*Visualizar resultados*"

Visto desde un punto de vista gráfico, se presenta el siguiente diagrama:

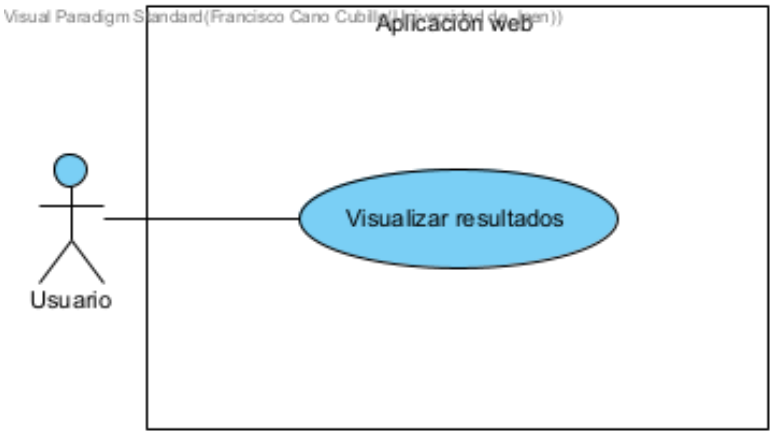


Figura 4 – Diagrama de caso de uso "*Visualizar resultados*"

3.3. Modelado y Representación de los Casos de Uso

Como ya se ha explicado al inicio de este capítulo, a continuación, se muestran los diagramas de actividad, de secuencia y de clases software. Estos ofrecen un mayor grado de comprensión del comportamiento del sistema de cara al usuario y de la estructura del proyecto.

3.3.1. Modelado del Ciclo de Vida. Diagramas de Actividad

Un diagrama de actividad es un tipo de diagrama de comportamiento UML en el que se representa el flujo de trabajo de un sistema utilizando actividades. Estas pueden ser tareas simples o complejas del sistema, y el diagrama de actividad plasma la interconexión de estas tareas utilizando flujos de control.

En el contexto de este proyecto, el siguiente diagrama de flujo describe las acciones descritas en el caso de uso *"Analizar texto insertado manualmente"* y cómo se interconectan sus tareas:

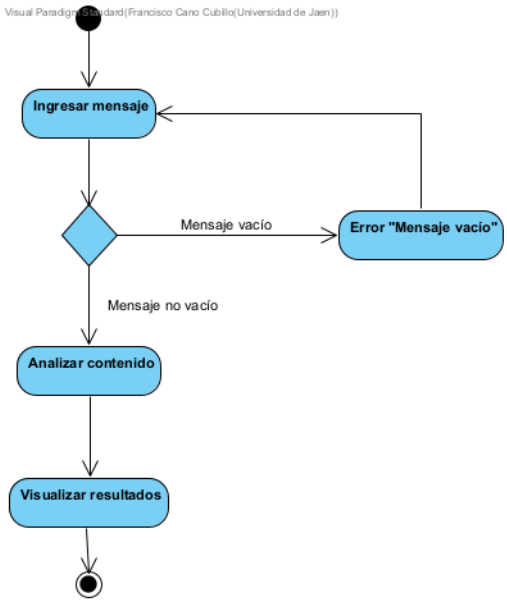


Figura 5– Diagrama de actividad para el caso de uso "*Analizar mensajes escritos por el usuario*"

Para el siguiente caso de uso, llamado *"Analizar comentarios extraídos de YouTube"*, se detallan las tareas que la componen en el siguiente diagrama de actividad:

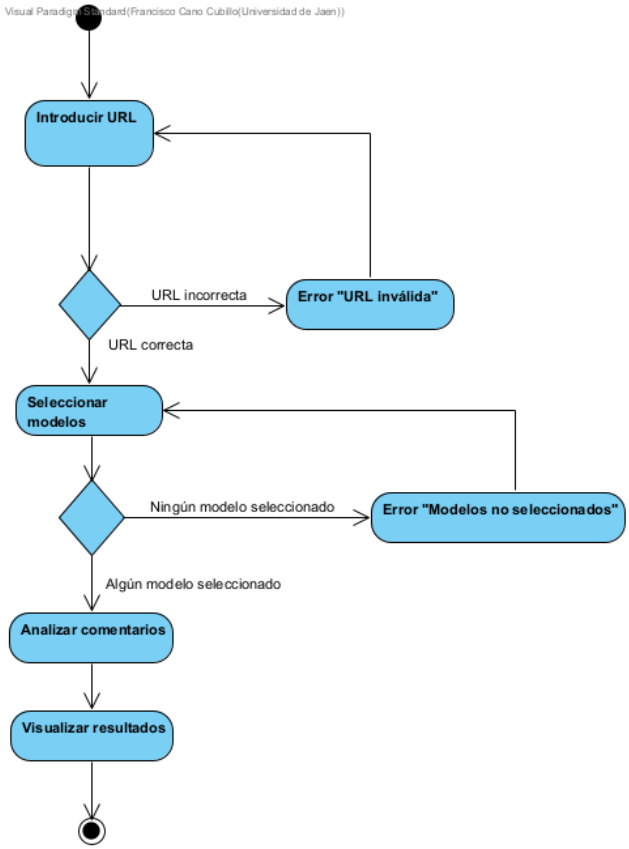


Figura 6– Diagrama de actividad para el caso de uso "*Analizar comentarios de un vídeo de YouTube*"

Para el último caso de uso, de nombre *"Visualizar resultados"*, se detallan las actividades que lo forman en el siguiente diagrama de actividad:

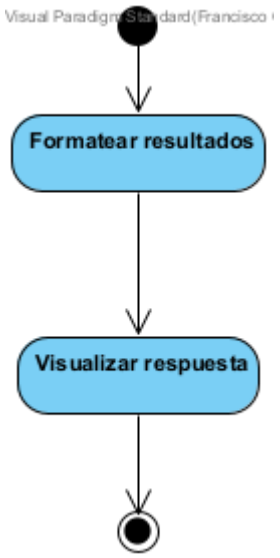


Figura 7 – Diagrama de actividad para el caso de uso "*Visualizar resultados*"

3.3.2. Modelado de la Dinámica. Diagramas de Secuencia

El diagrama de secuencia es otro tipo de diagrama que detalla la interacción entre los objetos involucrados en un caso de uso concreto y en un orden temporal específico. Este tipo de diagramas se centra en el intercambio de mensajes a lo largo del tiempo durante la ejecución de un escenario o una función del sistema.

En este trabajo, el diagrama de secuencia que se muestra a continuación detalla el intercambio de mensajes entre el usuario, la interfaz del sistema y el propio sistema durante el caso de uso *"Analizar texto insertado manualmente"*.

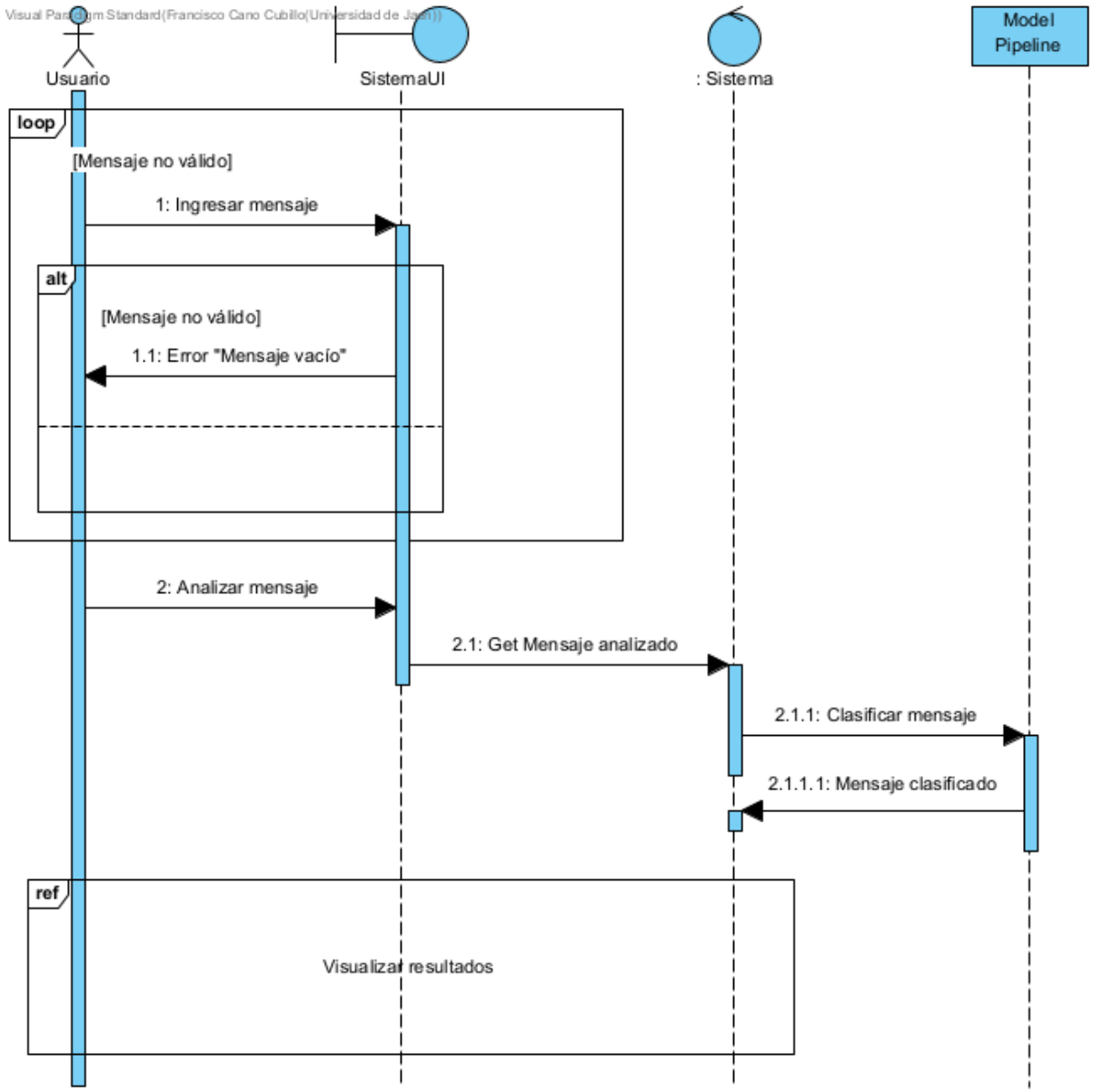


Figura 8 – Diagrama de secuencia para el caso de uso "*Analizar mensajes escritos por el usuario*"

Para el caso de uso *"Analizar comentarios extraídos de YouTube"*, también contemplado anteriormente, se muestra a continuación el paso de mensajes entre las clases involucrados, que coinciden con las del anterior caso de uso, añadiendo las clases límites que representan la API para conectarse a YouTube y el módulo de la aplicación que se conecta con dicha API:

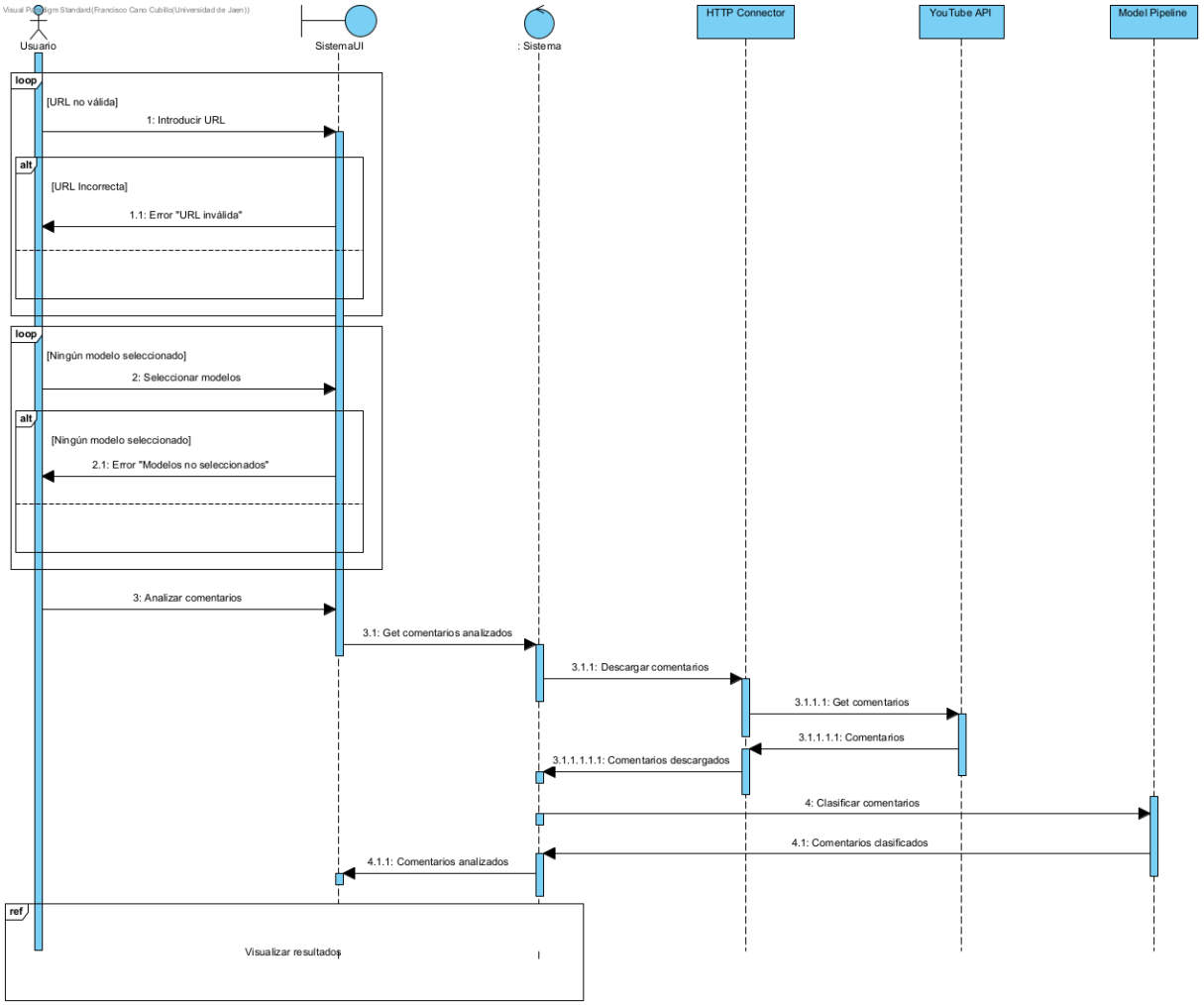


Figura 9– Diagrama de secuencia para el caso de uso "*Analizar comentarios de un vídeo de YouTube*"

Por último, para mostrar el flujo entre los distintos componentes del último caso de uso, *"Visualizar resultados"*, se muestra el siguiente diagrama de secuencia:

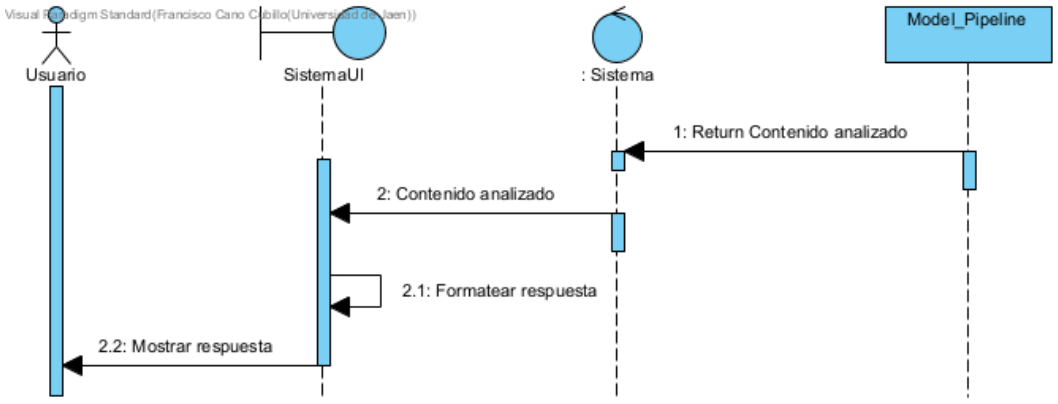


Figura 10 – Diagrama de secuencia para el caso de uso "*Visualizar resultados*"

3.3.3. Realización de Casos de Uso. Diagramas de Clases de Análisis

Un diagrama de clases de análisis ayuda a presentar y analizar las relaciones existentes entre los actores, clases y entidades en cada caso de uso contemplado en la aplicación. Es una herramienta que ilustra cómo interactúan las diferentes clases y qué acciones realiza cada una en el contexto que la engloba.

En este proyecto, se han contemplado una serie de casos de uso, a los cuales se les ha creado un diagrama de clases de análisis acorde. Para el caso de uso *"Analizar texto introducido a mano"*, este es su diagrama:

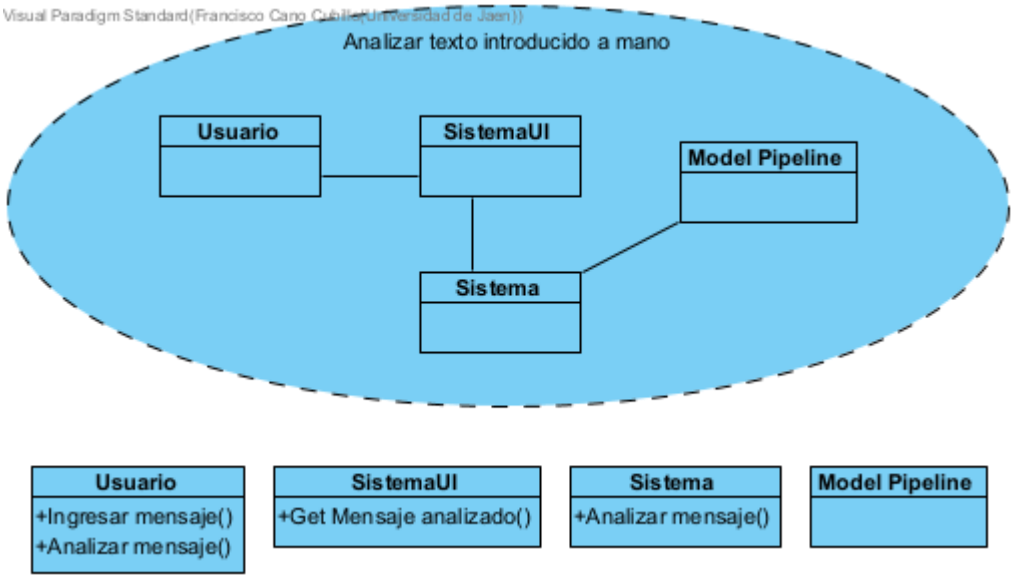


Figura 11– Diagrama de clase de análisis para el caso de uso "*Analizar mensajes escritos por el usuario*"

Para el segundo caso de uso, llamado *"Analizar comentarios extraídos de YouTube"*, se ha realizado otro diagrama siguiendo la misma lógica:

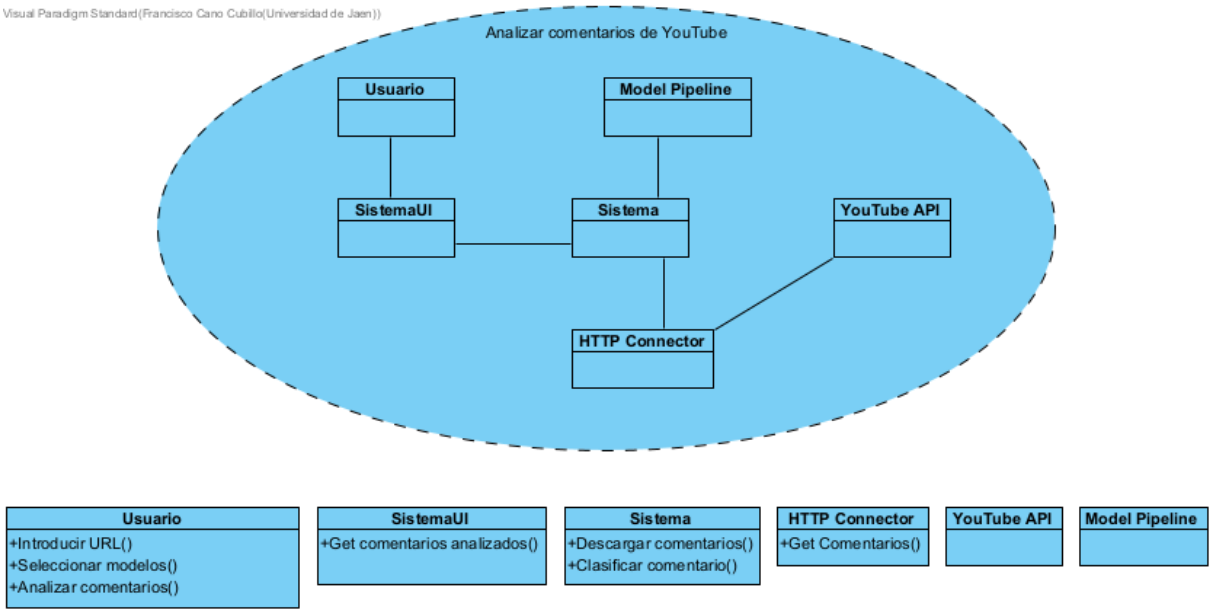


Figura 12 – Diagrama de clase de análisis para el caso de uso "*Analizar comentarios de un vídeo de YouTube*"

Para el último caso de uso, *"Visualizar resultados"*, también se ha creado un diagrama de clases de análisis con las relaciones entre las clases involucradas:

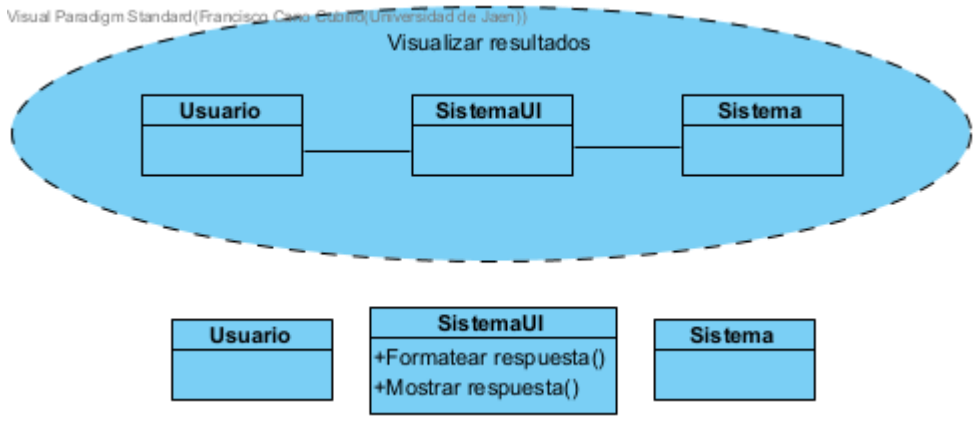


Figura 13 – Diagrama de clase de análisis para el caso de uso "*Visualizar resultados*"

3.4. Análisis de Redes Sociales

La idea inicial de este proyecto era construir una aplicación que recogiera comentarios escritos por los usuarios en diversas redes sociales, los analizará utilizando técnicas de PLN y mostrase los resultados de estos análisis en una interfaz para que el usuario pudiera visualizarlo.

Lamentablemente, no se ha podido conseguir llevar a cabo esta idea al completo, debido a que varias redes sociales de las consideradas inicialmente, o bien las APIs

que proporcionan escapan del presupuesto con el que cuenta este proyecto, o bien las APIs no permiten llevar a cabo esta idea, o bien, para tener acceso a la API, es necesario proporcionar cierta información que no es posible proporcionar.

Se consideraron las que, a priori, parecen ser las redes sociales más frecuentadas. A continuación, en Tabla 4 se muestra una descripción detallada de cada red social contemplada inicialmente en el proyecto:

 **X (antigua Twitter)**	**La versión gratuita de la API[1] solamente permite ser usada para la publicación de tweets o posts. Para el resto de las aplicaciones de la API, hay que realizar una suscripción.**
 **Reddit**	**No existe versión gratuita para la API[2] de esta red social desde el pasado 1 de julio de 2023. Es un precio poco elevado, ya que factura 0.24$ cada 1000 llamadas realizadas a la API.**
 **TikTok**	**Tienen un apartado para desarrolladores[3] en el cual, tras registrarse gratuitamente, se pueden realizar llamadas a la API y obtener información de vídeos, usuarios y comentarios, pero se necesita crear una empresa y una aplicación, y esta aplicación tiene que llevar la URL de la aplicación que va a integrar la API.**
 **Instagram**	**La API Graph[4] está dedicada a empresas y creadores de contenido. Permite obtener métricas de la propia cuenta o de otras cuentas empresariales, pero no puede acceder a la información de cuentas de usuarios corrientes.** **La API de visualización básica[5] permite obtener solamente información de los perfiles de los usuarios, lo que no incluye a los comentarios de las publicaciones. Del mismo modo que la API Graph, solamente se puede acceder a datos de usuarios corrientes y no de cuentas empresariales.**
 **YouTube**	**Permite acceder a información relativas a vídeos, canales y comentarios. Es gratuita[6], pero tiene una limitación de carga diaria, representada en puntos.**

Tabla 4 – Tabla comparativa de distintas APIs de redes sociales

[1] *Developer Platform*
[2] *Reddit API*
[3] *TikTok for Developers*
[4] *Meta for Developers – API Graph*
[5] *Meta for Developers – API de visualización básica*
[6] *YouTube Data API*

Tras haber contemplado las opciones, la óptima sería la de YouTube, por lo que el apartado de analizar comentarios extraídos por la API de las redes sociales extraerá, analizará y mostrará comentarios extraídos de los vídeos de YouTube.

4. Experimentación y Evaluación de Modelos

En este capítulo, se aborda en detalle la metodología empleada para entrenar y evaluar un conjunto de modelos con el fin de utilizarlos como detectores de lenguaje ofensivo. Para ello, se comenzará describiendo los conjuntos de datos utilizados, tanto para el entrenamiento como la evaluación, así como las métricas a medir durante el proceso de evaluación. A continuación, se detallará el procedimiento experimental llevado a cabo, explicando la configuración de los modelos y los parámetros de entrenamiento. Para finalizar, se analizarán y discutirán los modelos con mayor rendimiento para utilizarlos dentro de la aplicación desarrollada en este trabajo.

En resumen, este apartado proporciona una visión profunda de la experimentación realizada, permitiendo evaluar el rendimiento y la eficacia de los modelos de detección de lenguaje ofensivo en el contexto específico de este proyecto.

4.1. Conjuntos de datos utilizados

En este apartado, se realizará un análisis de los conjuntos de datos utilizados en el contexto de este trabajo. Estos conjuntos de datos representan la piedra angular sobre la que construir y evaluar los modelos de detección de lenguaje ofensivo. Todos ellos contienen, como mínimo, un conjunto de mensajes obtenidos de redes sociales y un campo de etiqueta que identifica el tipo de ofensividad que contiene el mensaje.

El primer *dataset* de todos ellos se llama **OffendES**[7] (Plaza-Del-Arco et al., 2021). Como parece evidente por el nombre, este conjunto se centra en la ofensividad y el objetivo de esa ofensividad, es decir, a quién se busca atacar. Las etiquetas que contiene OffendES son:

- **NO**: El mensaje no contiene ningún tipo de ofensividad

[7] *OffendES – Hugginface hub*

- **NOE**: El mensaje no contiene ningún tipo de ofensividad, con la salvedad de que contiene alguna palabra o conjuntos de palabras que podrían resultar ofensivas, pero por el contexto no lo son.
- **OFP**: El mensaje sí es ofensivo y busca atacar a una persona en concreto.
- **OFE**: El mensaje sí contiene ofensividad y va dirigida a un grupo de personas con una característica común.

El segundo conjunto de datos se trata del **EXIST**[8] (Rodríguez-Sánchez et al., 2021). Este está dirigido a la detección de comentarios sexistas. Contiene más etiquetas que el anterior y son:

- **non-sexist**: El mensaje no es sexista.
- **misogyny-non-sexual-violence**: El texto expresa odio y violencia hacia las mujeres.
- **sexual-violence**: Contienen sugerencias sexuales o se produce acoso de naturaleza sexual.
- **ideological-inequality**: Los mensajes buscan desacreditar movimientos e ideas feministas, rechazar la desigualdad entre hombres y mujeres o poner a los hombres en el papel de víctimas de la opresión de género de las mujeres.
- **stereotyping-dominance**: Expresa la falsa idea de que las mujeres no están capacitadas para realizar ciertas tareas u ocupar ciertos cargos, y que los hombres son superiores en ese aspecto.
- **objectification**: Presentan a las mujeres como objetos, despojándolas de su dignidad y aspectos personales. Aparte, describen ciertas cualidades físicas que las mujeres deben poseer para cumplir con los roles de género tradicionales.

El último de todos es el conjunto de datos llamado **HaterNet**[9] (Pereira-Kohatsu et al., 2019). Este *dataset* se utiliza para buscar y detectar lenguaje de odio. Es el que menos etiquetas contiene, porque es el más genérico de los tres, ya que

[8] *EXIST – NLP UNED*
[9] *HaterNet – Zenodo*

solamente distingue los mensajes que no contienen lenguaje de odio de los que sí lo tienen:

- **0**: El mensaje no contiene lenguaje de odio.
- **1**: El mensaje sí contiene lenguaje de odio.

Una vez vistos todos los conjuntos de datos, es conveniente mostrar una visión global de todos los *datasets* que se han empleado en este proyecto. Es por ello que se encuentra la siguiente tabla comparativa:

	***Datasets* incluidos**	**Número de filas por *dataset***	**Número de clases**
OffendES	Entrenamiento, Evaluación, Validación	16710, 13606, 100	4
EXIST	Entrenamiento, Evaluación	6977, 4368	6
HaterNet	Entrenamiento	6000	2

Tabla 5 – Tabla comparativa de los distintos conjuntos de datos

La razón por la cual se han elegido estos conjuntos de datos es por su diversidad al reconocer si un mensaje se considera ofensivo, sexista, de odio o no. Es por ello por lo que en esta fase de experimentación se ha buscado la manera de combinar los distintos *datasets* para concluir si, al combinarlos y entrenar un modelo con ellos, ayuda a realizar una detección más precisa que si se usaran cada uno por separado. Incluso se puede ir un paso más adelante, se puede integrar también conjuntos de datos como EXIST, que están enfocados a la detección de un tipo específico de ofensividad, con el fin de verificar si también incrementa la detección de mensajes de este subtipo de ofensividad o si, por el contrario, la enturbia y dificulta la identificación.

4.2. Métricas por analizar

Las métricas son medidas utilizadas para evaluar el rendimiento y eficacia de los modelos durante el proceso de entrenamiento. Proporcionan información

cuantitativa sobre diversos aspectos del modelo. Hay una gran variedad de métricas[10], pero esta experimentación se va a centrar en recopilar los resultados de tres de ellas específicamente, las cuales son:

- **Precisión *(accuracy)***: Calcula el número de clasificaciones correctas realizadas por el modelo en función del número total de clasificaciones realizadas. La fórmula para el accuracy sería:

$$accuracy = \frac{Clasifiaciones\ correctas}{Clasificaciones\ totales}$$

Fórmula 1 – Cálculo de la precisión

- **Recuperación *(recall)***: Es la proporción de ejemplos positivos que fueron correctamente clasificados con respecto al total de ejemplos positivos, es decir, la división entre el número de verdaderos positivos y la suma de verdaderos positivos y falsos negativos. La manera de calcular el *recall* sería:

$$recall = \frac{Verdaderos\ positivos}{Verdaderos\ positivos + Falsos\ negativos}$$

Fórmula 2 – Cálculo de la recuperación

- **F1 *(f1 score)***: Combina la precisión y el *recall* en una sola métrica, consistente en una media armónica de ambas métricas y se define como:

$$F1 = \frac{2 \cdot accuracy \cdot recall}{accuracy + recall}$$

Fórmula 3 – Cálculo de la F1

  - **F1 macro**: Es una especificación de la F1 y consiste en la media aritmética de todas las F1 de cada clase individual. Se calcularía como:

$$F1\ Macro = \sum_{i=1}^{n} \frac{F1_i}{n}, siendo\ n\ el\ número\ de\ clases\ del\ conjunto\ de\ datos$$

Fórmula 4 – Cálculo de la F1 macro

[10] *Scikit-learn*

4.3. Proceso de entrenamiento

En este apartado se busca describir el proceso de entrenamiento llevado a cabo. Para ello, se detallarán las principales características y pasos que sigue el código para realizar un entrenamiento completo. Este código, concretamente, ha sido desarrollado en Python utilizando la biblioteca PyTorch como base para el entrenamiento, así como otras bibliotecas auxiliares para la lectura de los archivos de los conjuntos de datos o para la creación de los objetos que encapsulan las métricas, por ejemplo. El código completo se puede visualizar en Google Colaboratory, o si lo prefiere, puede descargarse en fichero de Python:

1. **Computación de métricas y tokenizador**: Se inicializan los objetos que encapsulan las métricas a medir y el tokenizador, encargado de convertir los mensajes de los conjuntos de datos en entradas válidas para el modelo.
2. **Procesamiento de los conjuntos de datos**: Utilizando la biblioteca **pandas**, se cargan en memoria los ficheros que componen los conjuntos de datos del entrenamiento correspondiente y se tratan y preparan para ser utilizados a continuación.
3. **Creación de los *datasets* de entrenamiento y evaluación**: Una vez tengamos los ficheros procesados, se combinan para crear los conjuntos de datos que servirán como conjuntos de entrenamiento y de evaluación para nuestro modelo.
4. **Configuración de los parámetros de entrenamiento**: Se establecen los valores de las variables que definirán el transcurso del entrenamiento, tales como la tasa de aprendizaje, el número de etapas o *epochs* o el directorio en el cual se crearán los ficheros necesarios para utilizar el modelo.
5. **Iniciar el entrenamiento**: Una vez se tenga todo listo, se importa el modelo que hemos seleccionado, escogiendo la versión sin entrenar, y se procede a entrenarlo utilizando los conjuntos de datos que hemos creado y los parámetros que se han configurado anteriormente.

Para una explicación más detallada de este proceso, véase en 6.1 Implementación de la experimentación.

4.4. Resultados de la experimentación

Una vez implementado el código empleado para realizar los entrenamientos de esta fase de experimentación, se han realizado numerosas combinaciones con los conjuntos de datos posibles para entrenar una serie de modelos, obtener los resultados de sus entrenamientos y decidir cuáles son que mejor rendimiento han tenido.

A continuación, se explicarán cuáles han sido esas combinaciones de *datasets* para los entrenamientos, los nombres que han recibido, los resultados que han obtenido y las conclusiones a las que se han llegado.

4.4.1. Entrenamientos realizados

Como ya se ha mencionado anteriormente, el propósito de esta experimentación es verificar si la combinación de dos o más conjuntos de datos ayudan, o no, a la identificación de mensajes ofensivos, así como al reconocimiento de comentarios ofensivos de tipos de odio específicos. Siguiendo este objetivo, se han realizado una serie de entrenamientos de modelos realizando cambios en los conjuntos de entrenamiento y evaluación, en los cuales se han realizado las citadas combinaciones. A continuación, se detallan todas las combinaciones realizadas, así como la abreviación de los conjuntos de datos. Estas abreviaciones ayudan a identificar rápidamente las composiciones de los conjuntos de entrenamiento y evaluación utilizadas en el entrenamiento:

- **Off**: Conjunto de datos OffendES original.
- **Ex**: Conjunto de datos EXIST original.
- **Hn**: Conjunto de datos HaterNet.
- **OfO**: Conjunto de datos OffendES transformado a odio/no odio.
- **ExO**: Conjunto de datos EXIST transformado a odio/no odio.

Número experimento	Entrenamiento	Evaluación	Nombre
1	OffendES	OffendES	Off-Off
2	EXIST	EXIST	ExEx
3	OffendES (odio)	OffendES (odio)	OfO-OfO

4	EXIST (odio)	EXIST (odio)	ExO-ExO
5	OffendES (odio)	HaterNet	OfO-Hn
6	EXIST (odio)	HaterNet	ExO-Hn
7	HaterNet	OffendES (odio) + EXIST (odio)	Hn-OfOExO
8	OffendES (odio)	EXIST (odio)	OfO-ExO
9	OffendEs (odio) + HaterNet	EXIST (odio)	OfOHn-ExO
10	OffendES (odio) + EXIST (odio)	OffendES (odio) + EXIST (odio)	OfOExO-OfOExO
11	OffendES (odio) + EXIST (odio)	HaterNet	OfOExOHn-OfOExO
12	OffendES (odio) + EXIST (odio) + HaterNet	OffendES (odio) + EXIST (odio)	OfOExOHn-OfOExO

Tabla 6 – Combinaciones de los conjuntos de datos de entrenamiento y evaluación de los experimentos

4.4.2. Resultados de los entrenamientos

Tras definir los entrenamientos que se van a realizar, implementar el código de entrenamiento y realizar los mismos, estos son los mejores resultados de las métricas del mejor *epoch* del entrenamiento de cada entrenamiento:

Número experimento	Nombre	Accuracy	Recall	F1 macro
1	Off-Off	0,8816	0,7037	0,7332
2	ExEx	0,6699	0,5555	0,58
3	OfO-OfO	0,9120	0,8164	0,8429
4	ExO-ExO	0,7921	0,7933	0,7921
5	OfO-Hn	0,7667	0,7263	0,7134
6	ExO-Hn	0,6938	0,5441	0,5439
7	Hn-OfOExO	0,8033	0,6246	0,6447
8	OfO-ExO	0,6477	0,5352	0,5315
9	OfOHn-ExO	0,649	0,5392	0,5365

10	OfOExO-OfOExO	0,8843	0,8156	0,8303
11	OfOExOHn-OfOExO	0,735	0,6837	0,6736
12	OfOExOHn-OfOExO	0,8787	0,7818	0,8109

Tabla 7 – Resultados numéricos de las métricas para cada modelo entrenado

Desde un punto de vista más gráfico, los resultados de la fase de experimentación serían así:

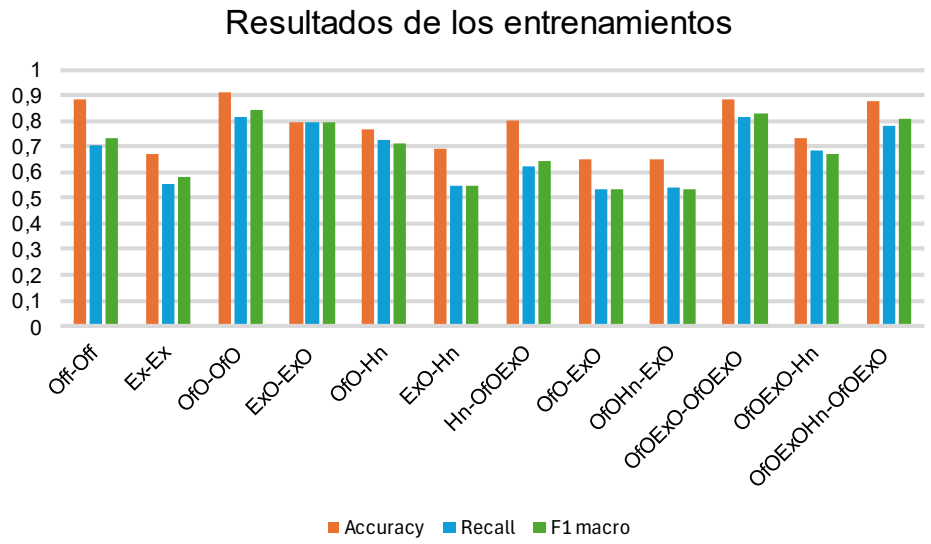
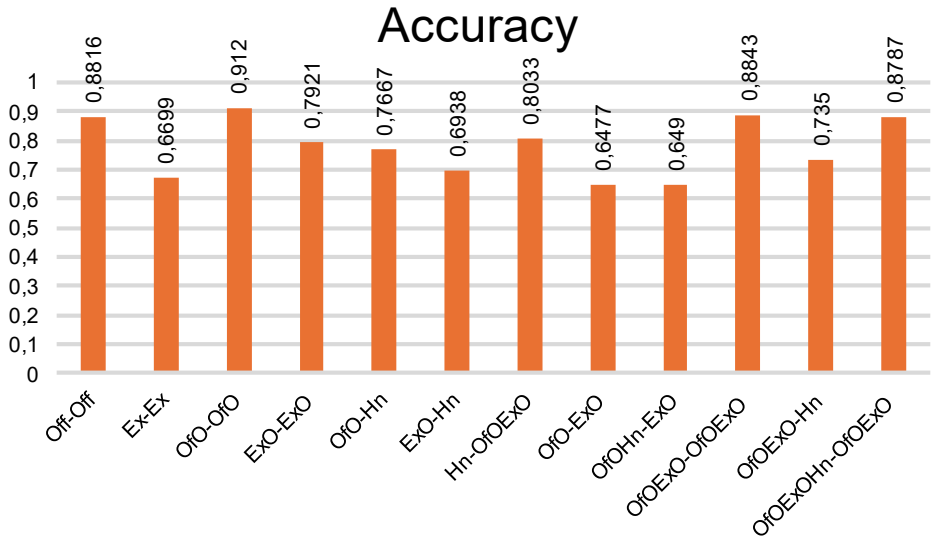
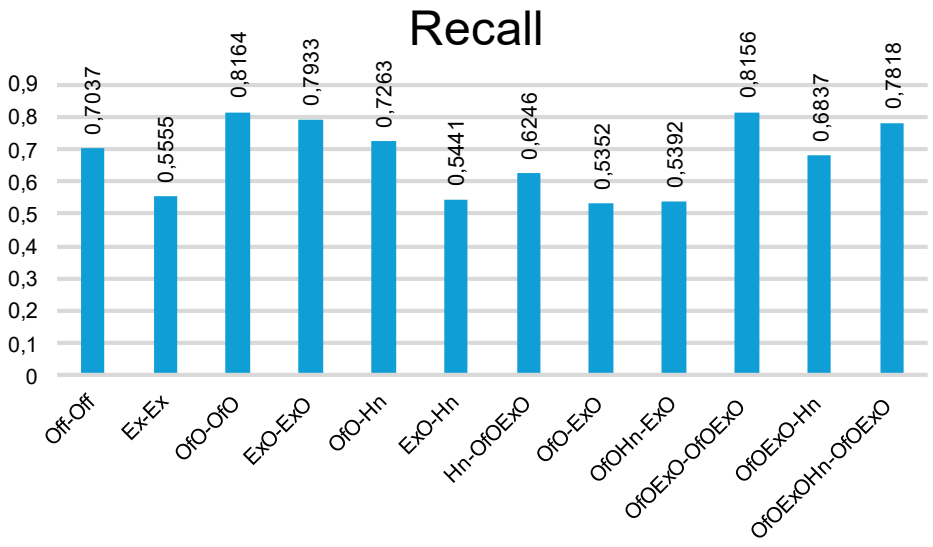


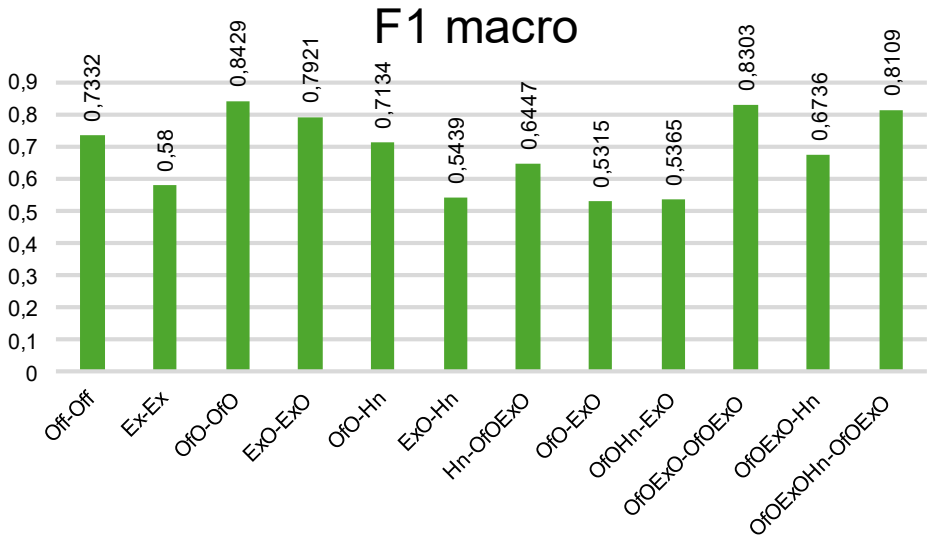
Figura 14 – Resumen gráfico de los resultados de los entrenamientos de los modelos



Resultados gráficos de la precisión de los entrenamientos de los modelos



Resultados gráficos de la recuperación de los entrenamientos de los modelos



Resultados gráficos de la F1 macro de los entrenamientos de los modelos

Tras recabar los datos y realizar un análisis exhaustivo de ellos, se han obtenido una serie de conclusiones y consideraciones:

- El modelo que ha obtenido el rendimiento más alto es el experimento número 3 (**OfO-OfO**). Todos sus valores son bastantes altos, superando el 80%, destacando su precisión superior al 90% (91,2%). Estas métricas demuestran una clasificación general bastante sólida.
- Los modelos número 10 (**OfOExO-OfOExO**) y 12 (**OfOExOHn-OfOExO**) también han obtenido resultados superiores al 80%, pudiendo

considerarlos como modelos con un rendimiento bastante consistente, aunque menor que el número 3.

- 4 de los 9 experimentos en los que ha participado el *dataset* EXIST en cualquiera de sus transformaciones han obtenido resultados bastantes pobres, con unos *recalls* y unos *f1 macro* con valores apenas superando el 50% y un *accuracy* entre el 60% y el 70%. De estos modelos se pueden extraer dos afirmaciones:
    - Estos modelos, si se plantea usarlos para futuras iteraciones del modelo, es muy aconsejable realizar mejoras en su entrenamiento o en su arquitectura para lograr que su desempeño sea mejor.
    - El conjunto de datos EXIST parece estar lastrando el entrenamiento y hace que empeoren los resultados. Revisando los mensajes que contiene, se han detectado numerosos casos en los que el mensaje se considera de una categoría sexista pero que, al leerlo, el mensaje en sí no es sexista, sino que se trata de una denuncia hacia una actitud sexista. Es por esto por lo que se considera que el *dataset* EXIST empeora los resultados, porque puede tender una tendencia de considerar mensajes ofensivos, sexistas o de odio aquellos que denuncien estas actitudes, en lugar de considerar aquellos que sí lo son. Los siguientes comentarios son algunos ejemplos de este suceso:
        - *“Sí, todo ello lamentable y deberían contar con terapia y otras ayudas profesionales como parte de cualquier seguro, público al menos. El tema es la constante invisibilización de las mujeres y la cultura misógina que acarrean y no parece que les interese cuestionarse”*
        - *“¡Quieren mujeres sumisas por que a una mujer libre no la pueden controlar!”*
        - *“Me da una felicidad enorme que finalmente sea ley. Es la mejor noticia del año. Pero no puedo evitar pensar en todas esas mujeres que murieron por culpa de la clandestinidad y que hoy no están para festejar con todas nosotras. Esto es x todas las que ya no están. Se logró.”*

Tras haber obtenido estas conclusiones, y recogiendo los objetivos de esta fase de experimentación, se puede considerar que el reconocimiento de odio por parte de los modelos ha mejorado considerablemente al combinar los conjuntos de datos iniciales, como se puede observar en los modelos con mejor rendimiento, siendo 3 (**OfO-OfO**), 10 (**OfOExO-OfOExO**) y 12 (**OfOExOHn-OfOExO**). En cuanto al otro objetivo de la experimentación, no todos los modelos han obtenidos grandes resultados en cuanto a la detección de subtipos de odio, como el sexismo en este caso. Aun así, el modelo 10 sí ha conseguido obtener resultados favorables en esta tarea y lograr una detección de comentarios sexistas que otros modelos no han podido lograr. De ellos, los dos mejores modelos serán los que se han integrado en la aplicación desarrollada, siendo:

Número experimento	Nombre	Accuracy	Recall	F1 macro
3	OfO-OfO	0,9120	0,8164	0,8429
10	OfOExO-OfOExO	0,8843	0,8156	0,8303

Figura 15 – Modelos seleccionados para ser integrados en la aplicación

La elección de estos modelos radica en el estupendo rendimiento que han logrado en la evaluación, despuntando sobre el resto. Si bien que 3 no destaca en la labor de la detección de subtipos de odio, lo compensa con su gran acierto en la detección de mensajes ofensivos. En cuanto a 10, también logra identificar los mensajes ofensivos de forma correcta, pero también compensa la carencia del otro modelo seleccionado y reconoce ese matiz sexista que pueden llegar a contener los mensajes.

5. Diseño de la aplicación

En el presente capítulo, se explica la estructura interna de la aplicación, sus clases principales, la organización de la aplicación tras ser desplegada y la interfaz de usuario, explicando el contenido de cada una de las distintas páginas web que componen la aplicación.

5.1. Estructura de la aplicación

La aplicación consta de tres módulos, o clases, principales, que son los que posibilitan que la aplicación funciones y sea operativa. Se encargan principalmente de recibir las peticiones que se envían desde el cliente, procesar la información de la solicitud y preparar y devolver la correspondiente respuesta. A estos módulos hay que incluir la clase principal, que es la encargada de iniciar el arranque de la aplicación.

Para realmente vislumbrar la estructura interna que posee la aplicación, a continuación, se muestra un diagrama de clases software:

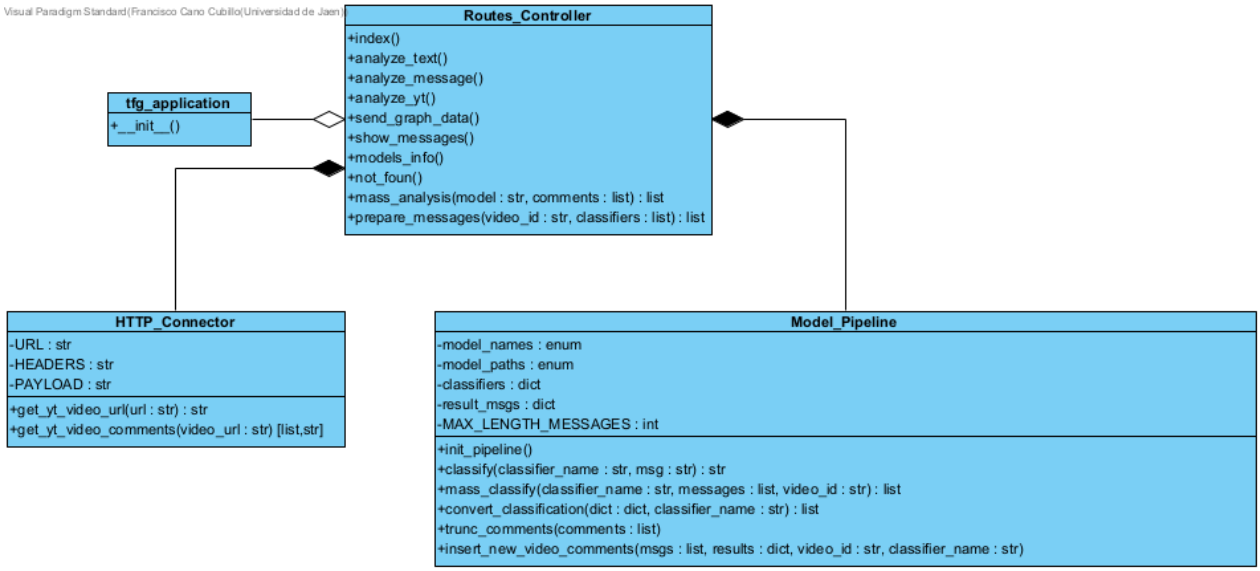


Figura 16 – Diagrama de clases software de la aplicación

En este esquema se observa la clase principal de la aplicación (*tfg_application*), la encargada de recibir las peticiones y enviar las peticiones (*Routes_Controller*), siendo el eje central, la clase que contiene los modelos seleccionados previamente y que son usados para analizar los mensajes entrantes y obtener y formatear las

respuestas (*Model_Pipeline*) y la que establece la conexión con la API de YouTube para descargar los comentarios del vídeo solicitado. Para una descripción más detallada y en profundidad de estas clases, véase Código back-end.

5.2. Estructura de despliegue

Ahora que todas las clases ya están completamente programadas, todas las páginas web diseñadas y construidas y, en general, la aplicación está terminada, es conveniente plantear la estrategia de salida al mundo real, es decir, de desplegar el proyecto en Internet para poder acceder a él y utilizarlo. Aunque el despliegue no se haga efectivo, debido a que es un prototipo, sí se ha planteado cuál sería la forma de hacerlo. La manera de realizarlo consiste en replicar el modo del propio *framework* de ejecutar la aplicación en un entorno local, consistente en montar una arquitectura de tipo cliente-servidor. Dicha arquitectura, adaptada a este proyecto, se puede contemplar en el siguiente diagrama de despliegue:

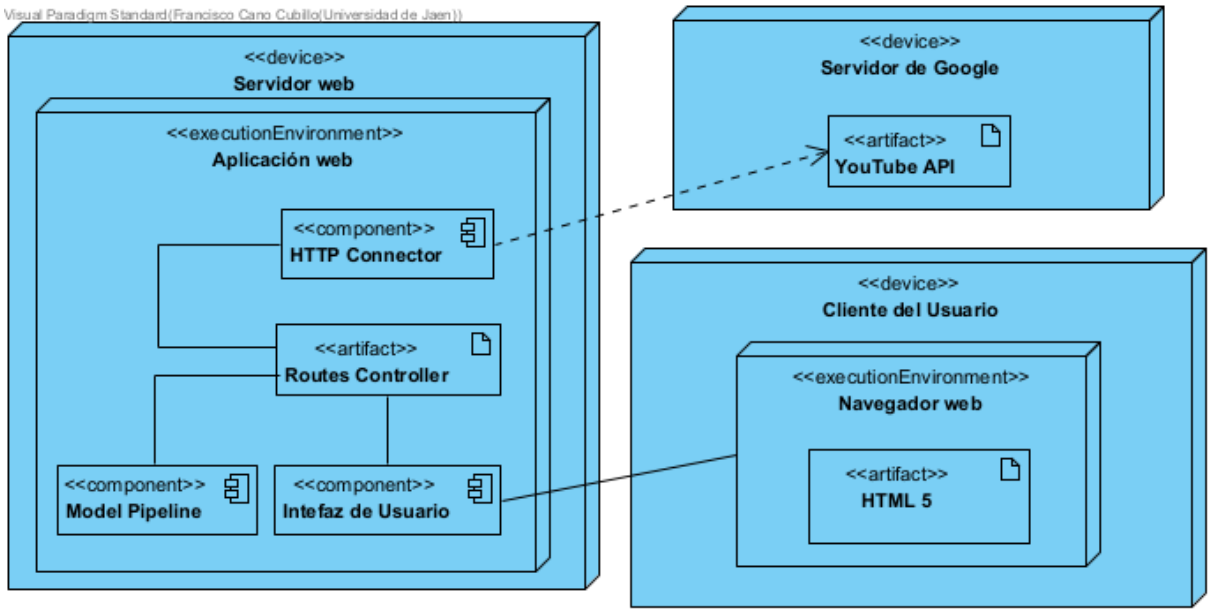


Figura 17 – Diagrama de despliegue de la aplicación

En el esquema, se muestra que tenemos tres nodos principales, siendo estos el servidor web que aloja a la aplicación, el dispositivo que controla el usuario y los servidores de Google en los cuales se encuentra la API de YouTube. Dentro del entorno de ejecución de la aplicación web, se encuentran las clases descritas en el apartado anterior, añadiéndose la interfaz de usuario. Todas ellas establecen una relación de asociación con la que es el eje principal de la aplicación (*RoutesController*).

Además, la interfaz de usuario también se asocia con el navegador web que utiliza el usuario, puesto es este el que se la muestra al usuario; y el componente *HTTP_Connector* que depende de la API de YouTube para desempeñar su labor.

5.3. Diseño de la Interfaz de Usuario

Para terminar con la sección dedicada al diseño de la aplicación desarrollada, se encuentra el capítulo dedicado a la interfaz de usuario, elemento imprescindible en cualquier aplicación, ya que ayuda a aportar información al usuario y lo guía durante su uso. En los siguientes apartados, se detallan cada una de las páginas que componen el proyecto, así como el objetivo principal que tienen cada una de ellas. Para una descripción detallada de los ficheros HTML de cada página web, véase Código front-end.

5.3.1. Página principal

Al iniciar la aplicación, la primera página con la que nos encontramos es la página de bienvenida o página principal. Esta página se caracteriza por mostrar al usuario el objetivo y las funcionalidades que posee la aplicación, así como de proporcionarle la posibilidad de ir a cualquiera de las páginas que la componen.

Los objetos principales de la aplicación son las tarjetas que describen brevemente el objetivo de las páginas a las que dirigen. Cada una cuenta con un título, una descripción de la funcionalidad y el botón para realizar la redirección. Todo esto se puede visualizar a continuación:

Ilustración 1 – Diseño de la página principal de la aplicación

5.3.2. Página "Analizar Texto"

En esta segunda página, se le habilita al usuario la posibilidad de introducir mensajes de texto para ser analizados.

Sus principales características son, en primer lugar, el cuadro de texto en el que el usuario puede escribir sus mensajes y el botón para que se analice y, en segundo lugar, la leyenda que explica cuáles son los modelos que están siendo utilizados para entrenar. Tras el análisis, los elementos destacados son los cuadros de texto de color que indican la respuesta de cada uno de los modelos integrados. Estos indican si el mensaje es ofensivo o no, gracias al color del cuadro y al propio mensaje, y el porcentaje de confianza en esa respuesta. Visualmente se puede apreciar a continuación:

Ilustración 2 – Diseño de la página "Analizar Texto" de la aplicación

5.3.3. Página "Analizar YouTube"

La otra página con la otra gran funcionalidad es la página encargada de analizar comentarios extraídos de vídeos de YouTube.

Para ello, se pone a disposición del usuario un campo para introducir el enlace al vídeo de YouTube y unos botones de tipo radio para que elija cuál o cuáles modelos quiere que se empleen para analizar los comentarios del vídeo que ha introducido. Tras introducir los parámetros y hacer el análisis, en la página se muestra un gráfico en el que se recopila la cantidad de mensajes ofensivos y no ofensivos para cada modelo que se ha seleccionado. Justo debajo de este gráfico se encuentra también una tabla en la que se muestra cuál ha sido la clasificación de los modelos para cada mensaje, de forma que los colores de las clasificaciones coincidan con los del gráfico. De igual forma que en la anterior página, incluye la leyenda que explica cada uno de los modelos disponibles. Visualmente la página se vería así:

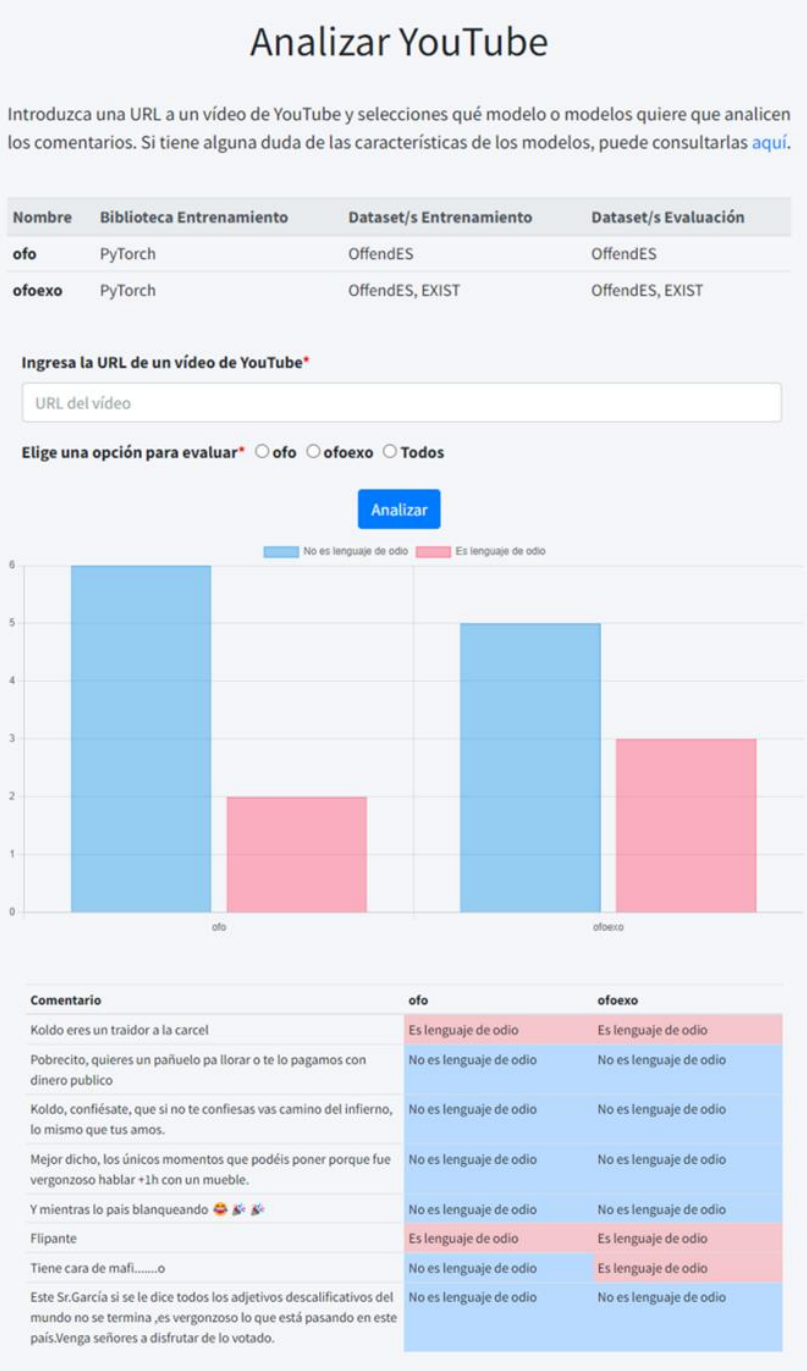


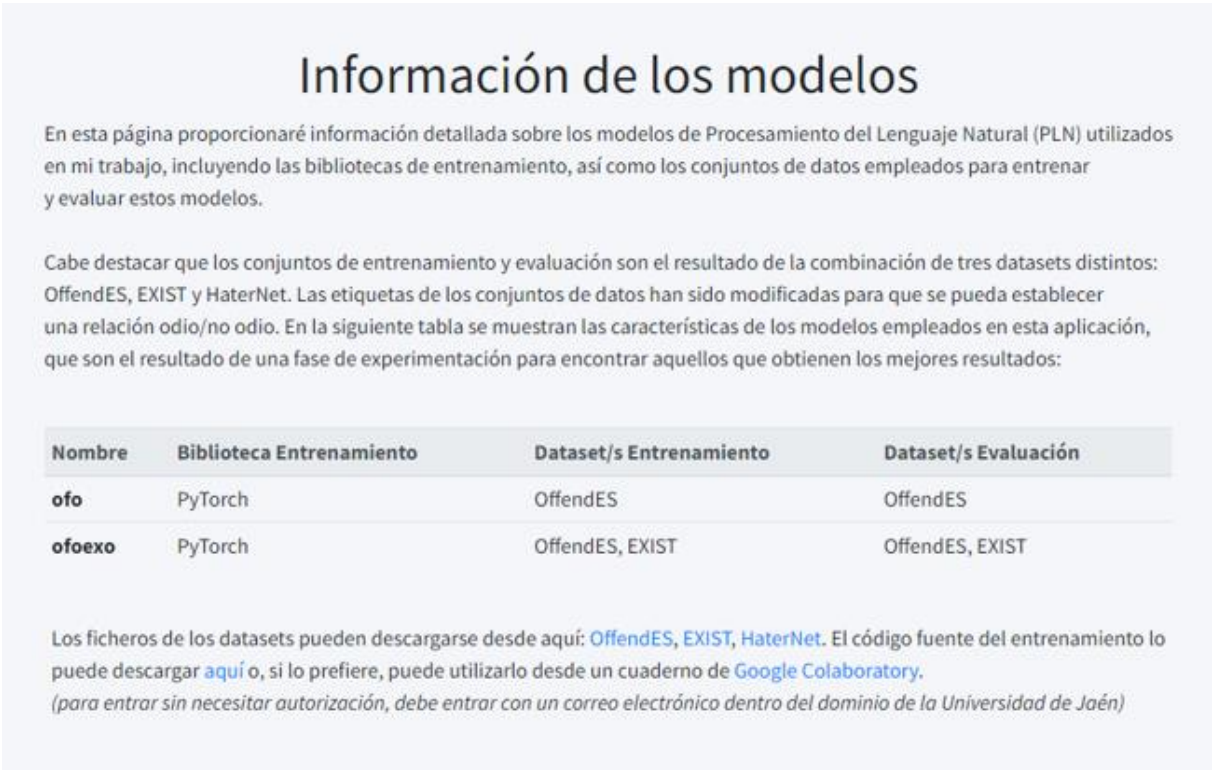
Ilustración 3 – Diseño de la página "Analizar YouTube" de la aplicación

5.3.4. Página "Sobre los modelos"

Para comprender de forma más detallada la configuración y conjuntos de datos utilizados en los modelos que se han incorporado a la aplicación, se encuentra esta página.

Detalla cómo se han generado los conjuntos de datos para el entrenamiento y la evaluación de los modelos, con qué biblioteca se ha utilizado y el nombre que se

les ha puesto. Al final, se encuentran los enlaces para descargar o visualizar el código utilizado para entrenar los modelos y los enlaces de descarga de los ficheros de los *datasets*. Cabe destacar que, nuevamente, se refleja en la página la leyenda para condensar la información de la página.



Información de los modelos

En esta página proporcionaré información detallada sobre los modelos de Procesamiento del Lenguaje Natural (PLN) utilizados en mi trabajo, incluyendo las bibliotecas de entrenamiento, así como los conjuntos de datos empleados para entrenar y evaluar estos modelos.

Cabe destacar que los conjuntos de entrenamiento y evaluación son el resultado de la combinación de tres datasets distintos: OffendES, EXIST y HaterNet. Las etiquetas de los conjuntos de datos han sido modificadas para que se pueda establecer una relación odio/no odio. En la siguiente tabla se muestran las características de los modelos empleados en esta aplicación, que son el resultado de una fase de experimentación para encontrar aquellos que obtienen los mejores resultados:

Nombre	Biblioteca Entrenamiento	Dataset/s Entrenamiento	Dataset/s Evaluación
ofo	PyTorch	OffendES	OffendES
ofoexo	PyTorch	OffendES, EXIST	OffendES, EXIST

Los ficheros de los datasets pueden descargarse desde aquí: OffendES, EXIST, HaterNet. El código fuente del entrenamiento lo puede descargar aquí o, si lo prefiere, puede utilizarlo desde un cuaderno de Google Colaboratory.
(para entrar sin necesitar autorización, debe entrar con un correo electrónico dentro del dominio de la Universidad de Jaén)

Ilustración 4 – Diseño de la página "Sobre los modelos"

5.3.5. Página "Error 404"

Cuando el usuario intenta acceder a una página web que no está contemplada en el ámbio de la aplicación, suele dar lugar a error *"404: Not Found"*, que indica que la página no existe. Es por ello por lo que en todas las aplicaciones web existe una página que se muestra al usuario en estos casos. Esta página advierte al usuario que la dirección web a la que intenta acceder no existe, e invita al usuario a volver al ámbito de la aplicación. En este caso es igual, se muestra un mensaje que indica que la página no existe y ofrece un enlace al usuario para volver a la página de inicio.

Ilustración 5 – Diseño de la página "Error 404"

6. Implementación del proyecto

En este capítulo del proyecto, se presenta una descripción detallada del código utilizado para el entrenamiento y evaluación de los modelos de la anterior fase de experimentación y de la aplicación desarrollada en este TFG. Se muestran distintos aspectos como el lenguaje y el *framework* utilizado para la implementación, el procedimiento detallado para entrenar los modelos, los distintos módulos que componen la aplicación y las conexiones con las distintas APIs.

Se pretende mostrar una visión detallada y precisa del procedimiento que se ha seguido para conseguir los modelos entrenados, así como de la organización y estructura del prototipo desarrollado, de forma que sea fácilmente legible y comprensible.

6.1. Implementación de la experimentación

Para lograr obtener los modelos entrenados que se han mostrado previamente, se han tenido que seguir una serie de pasos para obtener toda la información requerida, procesada y en el formato correcto. Es por ello por lo que, a continuación, se explicará en detalle los principales pasos, desarrollando cada uno de ellos, las bibliotecas utilizadas para ello y el uso que han tenido.

6.1.1. Bibliotecas y módulos empleados

Para empezar, las bibliotecas que han sido utilizadas en esta implementación son:

- **os**: Comprobación y creación de directorios
- **evaluate**: Cargar los módulos de evaluación de las métricas.
- **numpy**: Operaciones matemáticas sobre *arrays*.
- **pandas**: Lectura y manipulación de ficheros. En este caso, de ficheros csv, tsv y parquet.
- **datasets**: Encapsulamiento de los conjuntos de datos de entrenamiento y evaluación. Concretamente se ha importado el módulo **Dataset**.
- **transformers**: Todo lo relativo al entrenamiento, desde importar el modelo a entrenar y su correspondiente *tokenizer*, hasta el propio

proceso de entrenar siguiendo una serie de parámetros configurables. De esta biblioteca, se han utilizado los módulos **BertTokenizer**, **AutoModelForSequenceClassification**, **DataCollatorWithPadding**, **TrainingArguments** y **Trainer**.

6.1.2. Código de implementación

El proceso de entrenar los modelos vistos con anterioridad no resulta complejo, pero cuenta con una serie de pasos que garantizan su correcto funcionamiento. Cada uno juega un papel fundamental en el entrenamiento, ya que los primeros pasos son los que habilitan a los siguientes, ya sea porque se tratan de declaraciones de funciones, o porque se inicializan objetos que son utilizados posteriormente.

1. **Computación de métricas y *tokenizador***. Para comenzar, se crean las variables que encapsulan las métricas a valorar durante la evaluación de los modelos. Esto es posible con la función *load* de la biblioteca **evaluate**. Tras esto, se crea la función *compute_metrics* que devuelve los valores de las métricas para cada predicción realizada por el modelo durante la evaluación. Para finalizar este apartado, se crean la variable *tokenizer*, encargada posteriormente de convertir los mensajes de los conjuntos de datos en entradas válidas para el modelo, y la función *tokenize_data*, que realiza esta transformación al mensaje que reciba como parámetro de entrada.
2. **Procesamiento de los conjuntos de datos**. A continuación, se definen las funciones que cargan en memoria, usando la biblioteca **pandas**, los conjuntos de datos de entrenamiento y evaluación de cada *dataset* y los procesa de forma que solamente queden dos columnas, llamadas text (mensaje en sí) y *label* (clase del mensaje). Una vez terminado el procesamiento, se devuelven ambos objetos, el de entrenamiento y el de evaluación, aunque en caso de **HaterNet** solamente devuelve el de entrenamiento.
3. **Creación de los *datasets* de entrenamiento y evaluación**. Una vez ya tenemos los conjuntos de datos procesados y listos para su uso, es hora de crear los *datasets*. Para ello, se crean dos variables en las que se

almacenan dos *arrays* en el que se añaden aquellos conjuntos de datos que se desea que sean los que formen el *dataset* de entrenamiento, ídem para el *dataset* evaluación. Una vez tengamos en dos *arrays* los conjuntos de datos, se concatenan creando un gran conjunto de datos utilizando la función *concat* de **pandas**. Tras esto, se crean los *datasets* con la utilidad *from_pandas* de **Dataset**. Una vez hecho esto, solamente falta aplicar la anterior función *tokenize_data* para convertir los mensajes en entradas válidas del modelo. Este último paso se realiza con la función **map** de los *datasets*.

4. **Definición de los parámetros de entrenamiento**. Una vez preparados los *datasets* para utilizarlos en el entrenamiento, ahora toca configurar el propio entrenamiento. Para ello se crea un objeto de tipo **TrainingArguments**, encargado de recopilar la configuración del entrenamiento. Los argumentos que configuran el entrenamiento son:
    a. *output_dir*: Directorio en el que se escribirán los archivos del modelo tras cada iteración del entrenamiento.
    b. *learning_rate*: Tasa de aprendizaje a aplicar durante el entrenamiento. Controla el ajuste de los pesos del modelo basándose en el error calculado, estableciéndose el valor en $2e^{-5}$.
    c. *per_device_train_batch_size*: Tamaño de los lotes a procesar durante el entrenamiento. En este caso, los lotes son de 16 comentarios.
    d. *per_device_eval_batch_size*: Tamaño de los lotes a procesar durante la evaluación. En este caso, los lotes son de 16 comentarios.
    e. *num_train_epochs*: Número de iteraciones completas (*epochs*) a realizar durante el entrenamiento. En las ejecuciones realizadas se ha establecido el valor en 3.
    f. *weight_decay*: Regularización para controlar la penalización aplicada a los pesos para evitar el sobreajuste. Su valor es de 0.01.
    g. *evaluation_strategy*: Indica cuándo se evalúa el modelo. En este caso, se realiza al finalizar cada *epoch*.
    h. *save_strategy*: Indica cada cuánto tiempo se guarda en el directorio el modelo. Para estos entrenamientos, se ha indicado que se guarde tras cada *epoch*.

i. *load_best_model_at_end*: Establece si, cuando se guarda el modelo, se guarda siempre la mejor versión encontrada del modelo o la última registrada. En este caso, se guarda la mejor.
j. *push_to_hub*: Parámetro que indica si, al finalizar el entrenamiento, se sube el modelo al repositorio en la nube de *HugginFace*. Dado que en este proyecto solamente se busca que estén subidos aquellos modelos que se van a integrar en la aplicación, esta opción está deshabilitada.

5. **Ejecución del entrenamiento**. Para finalizar, se crea un objeto **Trainer**, que va a ser el encargado de realizar todo el entrenamiento. Para inicializarlo correctamente, se deben configurar el modelo[11] que se va a entrenar, los argumentos de entrenamiento que se han configurado en el paso anterior, *los datasets* de entrenamiento y evaluación, el *data_collator*, encargado de procesar los mensajes de forma que todos tengan la misma longitud; y la función *compute_metrics* del primer paso. Tras esto, se llama a la función *train* del propio objeto y dará comienzo el entrenamiento.

Durante el transcurso de la ejecución, aparecerá en la salida de la consola una barra de progreso y, tras cada evaluación, aparecerán los valores de las métricas que hayamos definido. Para consultar este código, se puede encontrar en *Google Colaboratory*[12] o en fichero de Python[13]. Es necesario acceder con cuenta de correo electrónico que esté dentro del dominio de la Universidad de Jaén.

6.2. Implementación de la aplicación

Este prototipo es el resultado final de la investigación y experimentación realizadas en este trabajo, ofreciendo una primera versión de una herramienta práctica y funcional para la detección del lenguaje ofensivo. La aplicación web desarrollada, como ya se explicó anteriormente, está implementada en Python y utilizando el *framework* de desarrollo Flask. Gracias a él, se facilita en gran medida la implementación y definición de la estructura de la aplicación, permitiendo agilizar el proceso.

[11] *dccuchile/bert-base-spanish-wwm-uncased*
[12] *Google Colaboratory*
[13] *Training.py*

6.2.1. Código back-end

Los principales archivos Python que componen la parte servidor de la aplicación son:

- **tfg_application.py**: Clase principal de la aplicación, y a la que se llama para iniciarla.
- **routes_controller.py**: Almacena todas las funciones que se ejecutan cuando el cliente realiza una petición a una URL. Se pueden condensar más de una dirección a la misma función o implementar dos funciones con la misma dirección pero que el método sea distinto. En este proyecto, estas funciones, o mandar renderizar archivos HTML para cambiar de página, o devuelven respuestas a peticiones realizadas por los formularios de la aplicación. Todo esto es posible gracias a las herramientas que el propio *framework* proporciona.
- **model_pipeline.py**: Es el encargado de conectarse con *Hugginface* usando su propia librería ***transformers***, para obtener los modelos con los que analizar los mensajes, y de analizar los propios mensajes. Implementa varias funciones para clasificar uno o varios mensajes, todas ellas usando objetos de la clase ***pipeline*** de ***transformers***. También, guarda un registro de los vídeos que han sido analizados y del resultado devuelto por los modelos al analizar los comentarios.
- **http_connector.py**: Realiza la conexión con la API de YouTube y, mediante peticiones HTTP con la biblioteca ***requests***, recupera los comentarios de los vídeos, utilizando el identificador del vídeo que se encuentra en el enlace del mismo. Debido al límite de peticiones diarias con el que cuenta la API, la función encargada de devolver los mensajes cuenta con una gestión de errores para, en caso de ocurrir algún fallo, poder solventarlo de la mejor forma posible.

6.2.2. Código front-end

Por otro lado, tenemos la parte cliente. Dicha parte está constituida por los ficheros HTML, CSS y JS, y son la parte visible de la aplicación. Las páginas que constituyen esta aplicación son:

- **index.html**: Constituye la página de inicio o bienvenida de la aplicación. En ella, se explica brevemente en qué consiste la aplicación y las distintas opciones que el usuario puede realizar. Estas opciones se representas mediante tarjetas, las cuales contienen el título de la sección a la que redirigen, un resumen de la funcionalidad de ese apartado y un botón que dirige al usuario a dicha página.
- **analyze_text.html**: Permite introducir al usuario un texto y analizarlo en busca de lenguaje de odio. Tras el análisis, se muestran los resultados de los distintos modelos integrados en la aplicación.
- **analyze_yt.html**: Da la posibilidad al usuario de introducir el enlace a un vídeo de YouTube y elegir con qué modelos clasificar los comentarios de dicho vídeo. Tras finalizar la clasificación, se muestran un gráfico con la distribución de comentarios dependiendo de las clasificaciones, y una tabla con los comentarios y la clase que cada modelo le ha colocado.
- **models_info.html**: Muestra al usuario la configuración básica de los modelos integrados en la aplicación. Explica cuáles son las clases con las que los modelos clasifican los mensajes, con qué bibliotecas se han entrenado y cuáles han sido los conjuntos de datos utilizados en el entrenamiento y en la evaluación. También da la opción de descargar los propios *datasets* y el código utilizado para el entrenamiento tanto en formato *notebook* con Google Colaboratory, como en fichero de Python.

Para asegurar la correcta navegación entre páginas web de este proyecto, se ha incluido también una barra de navegación a modo de cabecera con el escudo de la Universidad de Jaén, el logo de la Escuela Politécnica Superior de Jaén, que redirigen a sus respectivas páginas web, y un enlace por cada página de la aplicación (Ilustración 6).

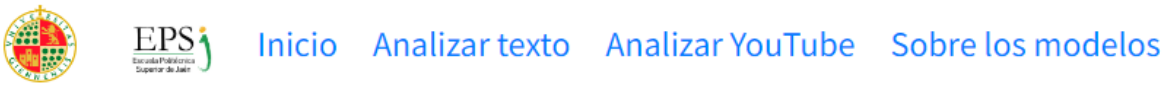


Ilustración 6 – Barra de navegación de la aplicación

6.2.3. Consideraciones propias del framework

Cabe destacar que también, para casos como este en el que el proyecto se trata de un prototipo, Flask permite configurar el entorno en modo desarrollo de forma que, al realizar cualquier cambio en cualquiera de los archivos que componen la aplicación y se guarden esos cambios, la aplicación automáticamente realizará un reinicio rápido.

Este modo resulta muy útil para agilizar la implementación y no obligar al desarrollador a reiniciar la aplicación con cada modificación realizada. Para conseguir esto, basta con crear en el directorio raíz de la aplicación un fichero llamado *.flaskenv*, en el que se introducen dos variables: ***FLASK_APP*** y ***FLASK_DEBUG***. La primera indica cuál es el fichero de Python que contiene a la clase principal, por lo que en nuestro caso se le asignará **tfg_application.py**. El segundo indica si la aplicación ha de ejecutarse en modo desarrollador o no. Este modo ayuda a optimizar el tiempo de visualización de cambios, ya que realiza un reinicio de la aplicación cuando esta detecta algún cambio. Como en este caso sí lo deseamos, establecemos esta variable con el valor 1.

7. Anexos

Para finalizar, se encuentran dos de los apartados imprescindibles cuando se desarrolla una aplicación, tratándose de los manuales de instalación y de usuario. Ambos ayudan al usuario final a conseguir obtener y configurar correctamente la aplicación, indicando las posibles dependencias que pueda tener, así como a la utilización óptima de la herramienta, con el fin de obtener una experiencia de usuario y optimización del uso máxima.

7.1. Manual de instalación

Para poder obtener la aplicación de este proyecto y utilizarla, se deben seguir una serie de pasos:

1. Instalar una versión de Python, descargando el fichero ejecutable desde el sitio web oficial[14]. Para mayor compatibilidad, la aplicación ha sido desarrollada en la versión 3.11.8, con fecha de actualización a 6 de febrero de 2024.
2. Una vez instalada la distribución de Python, hay que descargar la aplicación[15]. Para hacerlo, hay que tener creada una cuenta en el dominio de la Universidad de Jaén en *GitLab*.
3. Tras haber descargado y descomprimido el fichero, abrimos un símbolo del sistema y cambiamos el directorio de trabajo al directorio raíz de la aplicación descargada:

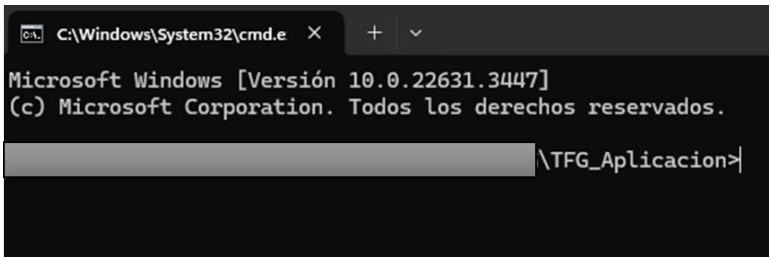


Ilustración 7 – Símbolo del sistema indicando que el directorio de trabajo es el de la aplicación

[14] *Python Downloads*
[15] *Prototipo de Monitor de Detección de Lenguaje Ofensivo – GitLab*

4. Ejecutamos el siguiente comando, el cual leerá el fichero **requirements.txt** situado en el mismo directorio e instalará todas las bibliotecas que incluye:

```
python -m pip install -r requirements.txt
```

Comando 1: Orden utilizada para instalar las bibliotecas que se indican en el fichero

5. Si desea comprobar que las bibliotecas están instaladas exitosamente, puede comprobarlo con la siguiente orden, la cual listará todas las bibliotecas instaladas en el equipo:

```
python -m pip list
```

Comando 2: Orden para listar las bibliotecas instaladas en el equipo

6. Tras haberse asegurado de que las bibliotecas están instaladas, solamente resta iniciar la aplicación. Para ello, utilice la siguiente instrucción:

```
python -m flask run
```

Comando 3: Instrucción para ejecutar una aplicación *FLASK*

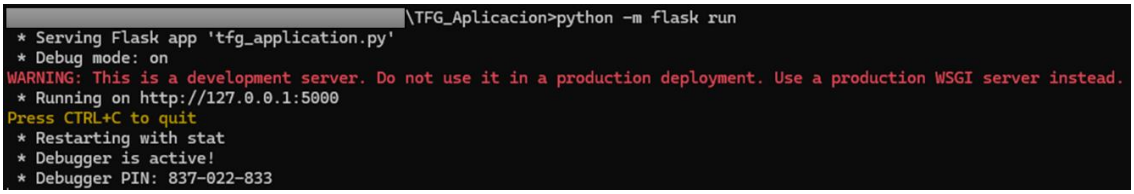


Ilustración 8 – Símbolo del sistema con mensaje de confirmación de que la aplicación se inició correctamente

Una vez iniciada la aplicación Ilustración 8, basta con abrir el navegador y escribir en la barra de búsqueda ***127.0.0.1:5000***, o en su defecto, ***localhost:5000***.

7.2. Manual de usuario

En esta sección, se presenta una guía completa por cada página web de la aplicación en la que se explica todas las posibles interacciones que el usuario puede realizar, los casos de éxito y los casos de error al interactuar con la aplicación.

7.2.1. Página principal

En la página de inicio de la aplicación se muestra una breve descripción del propósito de este proyecto, las funcionalidades que proporciona e información adicional sobre las herramientas utilizadas para llevar a cabo estas funcionalidades. Además, encontramos una tarjeta por cada sección adicional de la aplicación,

excluyéndose ella misma, en la que se muestra una escueta explicación de su propósito y su uso. Todo esto se puede visualizar en Ilustración 9:

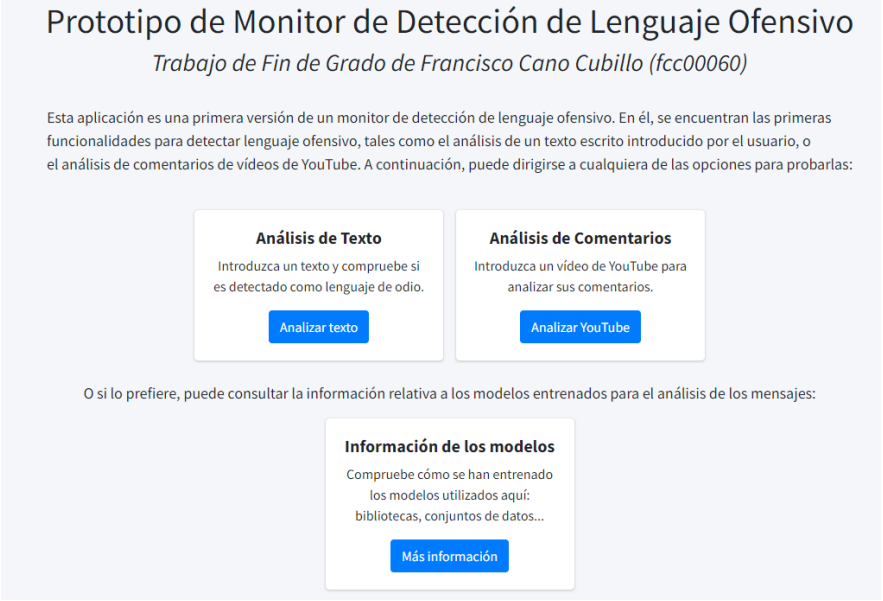


Ilustración 9 – Página de inicio de la aplicación

7.2.2. Página de "Analizar Texto"

En esta sección, se le presenta al usuario un cuadro de texto ampliable en el que podrá redactar un texto, y los modelos que se encuentran integrados en la aplicación lo analizarán y devolverán su clasificación, de forma que el texto puede ser considerado como contenedor de lenguaje de odio o estar exento de este.

Inicialmente, la página se muestra como en Ilustración 10:

Ilustración 10 – Página de "Analizar Texto"

Una vez que el texto está escrito se pulsa en el botón *Analizar*, el cual mandará el texto insertado por el usuario a los modelos para su clasificación. Una vez el análisis está completo, se devuelve las respuestas y se muestran en bloques, en función de si es contenido de odio o no, de alerta de éxito o de peligro. Los primeros se utilizan para los mensajes que no contienen lenguaje de odio y los de peligro, para los que sí lo contienen. Unos ejemplos de este funcionamiento se pueden contemplar en Ilustración 11 e Ilustración 12 :

Ilustración 11 – Página "Analizar Texto" con un mensaje escrito y la respuesta no ofensiva de los modelos

Ilustración 12 – Página "Analizar Texto" con un mensaje escrito y la respuesta ofensiva de los modelos

Aun con todo, existe la posibilidad de que el usuario intente, intencionalmente o por accidente, mandar a analizar un texto que esté vacío. En este caso, se contempla un mensaje de error que obliga al usuario a introducir algún texto en el área destinada para ello antes de poder analizar (Ilustración 13).

Ilustración 13 – Página "Analiar Texto" con error de mensaje vacío

7.2.3. Página de "Analizar YouTube"

En la siguiente pestaña tenemos la página dedicada al análisis y clasificación en masa de comentarios provenientes de vídeos de YouTube. Para esto, se ha ideado la siguiente interfaz:

Ilustración 14 – Estado inicial de la página "Analizar YouTube"

Con ella, se pide al usuario que introduzca un enlace a un vídeo de YouTube y que elija qué modelos quiere que analicen este mensaje, si solamente uno de ellos, o ambos. Cuando se rellenan los dos campos, el usuario tiene que pulsar en el botón *Analizar*.

Ilustración 15 – Página "Analizar YouTube" con los campos requeridos rellenados

Tras esto, la aplicación se conecta al servicio correspondiente de la API de YouTube para obtener los comentarios, utilizando el identificador del vídeo. Una vez

se completa el análisis, se muestra en conteo de comentarios con lenguaje de odio y sin él en un gráfico realizado con *ChartJS*[16].

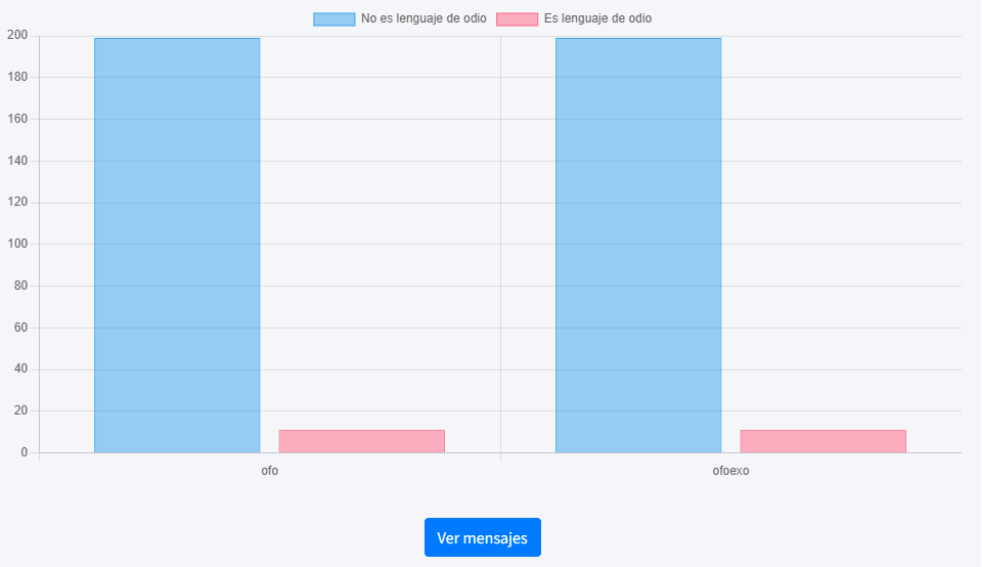


Ilustración 16 – Gráfico mostrado tras el análisis de los comentarios

Además del gráfico con los resultados contabilizados, se da la opción de visualizar el resultado de los modelos comentario a comentario. Para mostrarlos, basta con pulsar en el botón *Ver mensajes* situado debajo de gráfico.

[16] *Chart.js | Open source HTML5 Charts for your website*

Comentario	ofo	ofoexo
Un gustazo de partida ✨ El sueño lo terminaré de contar en otra ocasión 👍	No es lenguaje de odio	No es lenguaje de odio
Se buscaron una igual que loca que olivia para jugar su deck	No es lenguaje de odio	No es lenguaje de odio
Me declaro fan de Gakian, hacía tiempo que no me reía tanto. De las mejores partidas del canal. Necesito ver otra partida, pero esta vez con Gakian, Jagger, Ania e Isma. 😂😂😂	No es lenguaje de odio	No es lenguaje de odio
Maravillosas las palabras finales de Gakian al perder, tenéis que hacer una partida de rol con ambas, son puro caos xD	No es lenguaje de odio	No es lenguaje de odio
No se ustedes pero la pelirroja se me hizo muy insoportable, fuera de eso buena partida me gustó	No es lenguaje de odio	No es lenguaje de odio
45:22 y ahora te voy a tirar algo que te vas a quedar calvo	Es lenguaje de odio	No es lenguaje de odio
Jajajajajaja menuda partida tan random estaaaa, sería genial ver más partidas asiiii. Judge! Duda: ¿Hay missplay en 1:13:25? ¿Alesya puede llevarse la recompensa del botín al comienzo de su paso final? Como Panda ha muerto, ¿no se checkea la State-Based Action de jugador muerto y el jugador de la izquierda de Alesya pasa a ser Isma?	No es lenguaje de odio	No es lenguaje de odio
Las miradas de Panda a isma cuando la chica de su lado juega son la ostia jajajaja Está en plan: Isma tio, que nos gana. Le has dado el mejor deck cabron	No es lenguaje de odio	Es lenguaje de odio
Gakian es un autentico agente de Hastur, un agente de la entropía. Ismael estaba perdiendo cordura por segundo!!! XDDDD	No es lenguaje de odio	No es lenguaje de odio
Quiero jugar comander solo por esta partida! Maximo 😂😂😂	No es lenguaje de odio	No es lenguaje de odio
50:34 Pipo? a caso veo una Alexelcapo reference? PD: por favor convenzan a Alex de ir a jugar una partida	No es lenguaje de odio	No es lenguaje de odio
Es increíble que si Ania hubiera estado aquí sería la persona más cuerda de la partida. No tengo pruebas, tampoco dudas.	No es lenguaje de odio	No es lenguaje de odio
Es buenísimo. Les pido una partida con Gakian y Jagger asddasdjasjd	No es lenguaje de odio	No es lenguaje de odio
Insisto... el mejor "mansplainning" de la historia 😂😂	No es lenguaje de odio	No es lenguaje de odio
🤠🤡🤠🤠🤡🤠🤡	No es lenguaje de odio	Es lenguaje de odio
La otra diciendo su masó esta roto... Yo tipo: pero si lo hubieras matado, en tu turno.	No es lenguaje de odio	No es lenguaje de odio

Ilustración 17 – Tabla informativa con la clasificación de cada modelo para cada mensaje

Como se puede observar, se recogen en una tabla donde la primera columna es el propio comentario y las sucesivas son los distintos modelos con los resultados para cada comentario, cambiando también el color en función de si se consideran o no lenguaje de odio.

De la misma forma que ocurre en la anterior sección, aquí también casos en los que el usuario puede insertar datos incorrectos, por lo que también se realiza una gestión de errores para evitar situaciones indeseadas.

El primero consiste en intentar realizar un análisis de una URL vacía o no indicar con qué modelo o modelos se quiere analizar los mensajes. Para solventar esto, se implementa de la misma forma que en el apartado anterior un mensaje de error indicando que los campos señalados con * han de estar rellenos obligatoriamente.

Ilustración 18 – Página "Analizar YouTube" con error de campos obligatorios no rellenados

El segundo tipo de error que puede ocurrir, y siguiendo con la temática del anterior, consiste en introducir un enlace no válido. Para ello, se obliga al usuario a utilizar enlaces correctos provenientes de YouTube. Se entiende por enlace válido aquel que vídeo cuyos dominios coinciden con ***www.youtube.com*** o con ***youtu.be***. También debe incluir el identificador del vídeo. En el caso del primer dominio, el identificador del vídeo se encuentra en los parámetros de la URL, cuyo nombre es *v*, mientras que, en el segundo caso, es el primer subdirectorio del enlace. Todos estos errores se pueden visualizar en Ilustración 19 e Ilustración 20.

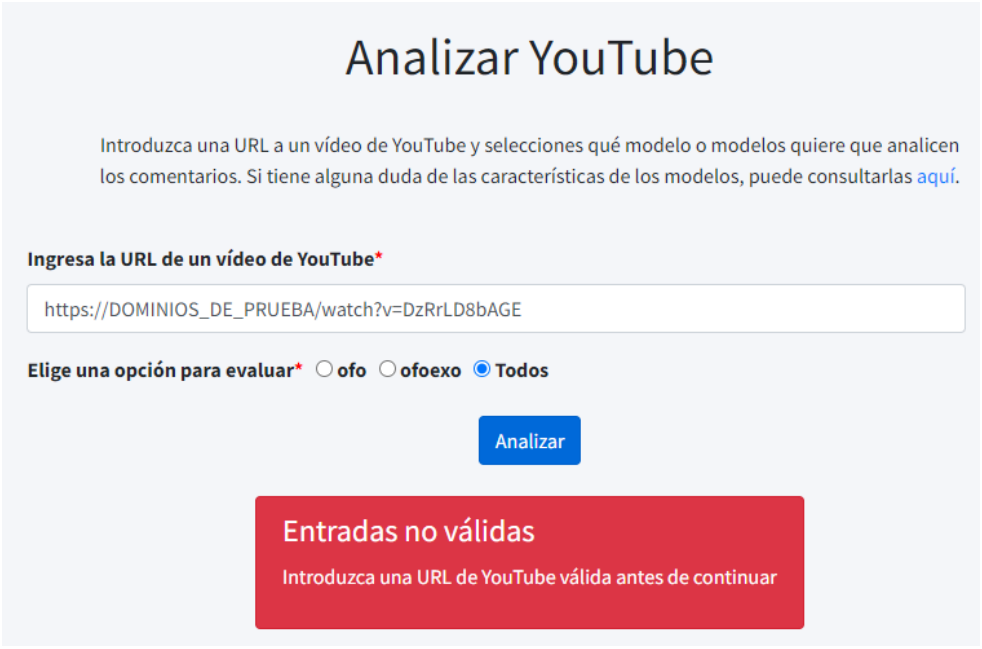


Ilustración 19 – Página "AnalizarYouTube" con mensaje de URL no válida por dominio de YouTube incorrecto

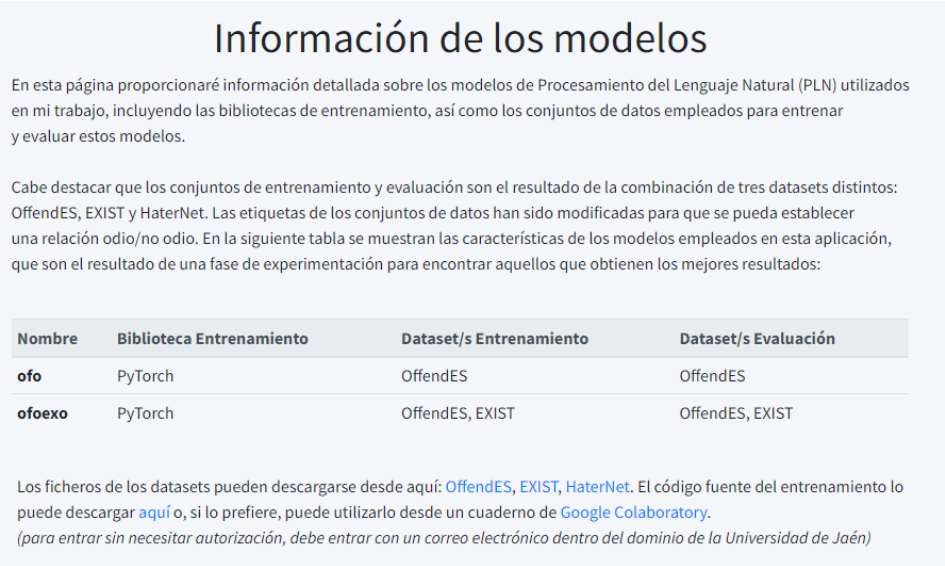
Ilustración 20 – Página "Analizar YouTube" con mensaje de URL no válida por falta de identificador de vídeo

También pueden darse situaciones no deseadas, como que los comentarios del vídeo estén deshabilitados, se haya alcanzado el cupo máximo de peticiones diarias, etc. Para advertir de esto, se ha implementado también una alerta que advierte al usuario de un fallo interno ocurrido durante el proceso de petición a la API:

Ilustración 21 – Página "Analizar YouTube" con mensaje de error durante la recogida de mensajes

7.2.4. Página "Sobre los Modelos"

La penúltima sección de la aplicación web es una vista informativa sobre cómo se han entrenado los modelos embebidos en la parte servidor. Explican cómo se han obtenido los conjuntos de datos utilizados en los entrenamientos y evaluaciones de los modelos, las bibliotecas utilizadas para ello y enlaces de descarga, tanto de los *datasets*, como del código fuente.



Información de los modelos

En esta página proporcionaré información detallada sobre los modelos de Procesamiento del Lenguaje Natural (PLN) utilizados en mi trabajo, incluyendo las bibliotecas de entrenamiento, así como los conjuntos de datos empleados para entrenar y evaluar estos modelos.

Cabe destacar que los conjuntos de entrenamiento y evaluación son el resultado de la combinación de tres datasets distintos: OffendES, EXIST y HaterNet. Las etiquetas de los conjuntos de datos han sido modificadas para que se pueda establecer una relación odio/no odio. En la siguiente tabla se muestran las características de los modelos empleados en esta aplicación, que son el resultado de una fase de experimentación para encontrar aquellos que obtienen los mejores resultados:

Nombre	Biblioteca Entrenamiento	Dataset/s Entrenamiento	Dataset/s Evaluación
ofo	PyTorch	OffendES	OffendES
ofoexo	PyTorch	OffendES, EXIST	OffendES, EXIST

Los ficheros de los datasets pueden descargarse desde aquí: OffendES, EXIST, HaterNet. El código fuente del entrenamiento lo puede descargar aquí o, si lo prefiere, puede utilizarlo desde un cuaderno de Google Colaboratory.
(para entrar sin necesitar autorización, debe entrar con un correo electrónico dentro del dominio de la Universidad de Jaén)

Ilustración 22 – Página "Sobre los modelos"

7.2.5. Página "Error 404"

La última sección de la aplicación web se trata de una página de gestión de errores, concretamente, del error 404 *"Page not found"*. Con esto, se busca hacer ver al usuario que utilice el prototipo que el enlace que ha introducido no está dentro del alcance esperado y que, por lo tanto, la página web asociada no existe. Se trata de una vista muy sencilla donde aparecen los logos de la Universidad de Jaén y de la Escuela Politécnica Superior de Jaén, un mensaje descriptivo del error y un enlace para volver a la página de inicio.

Ilustración 23 – Página "Error 404"

8. Bibliografía

Balayn, A., Yang, J., Szlavik, Z., & Bozzon, A. (2021). Automatic Identification of Harmful, Aggressive, Abusive, and Offensive Language on the Web: A Survey of Technical Biases Informed by Psychology Literature. *ACM Transactions on Social Computing*, *4*(3), 1–56. https://doi.org/10.1145/3479158

Chen, Y., Zhou, Y., Zhu, S., & Xu, H. (2012). *Detecting Offensive Language in Social Media to Protect Adolescent Online Safety*.

Goldberg, Y. (2022). *Neural Network Methods in Natural Language Processing*.

Molero, J. M., Pérez-Martín, J., Rodrigo, A., & Peñas, A. (2023). *Offensive Language Detection in Spanish Social Media: Testing from Bag-of-Words to Transformers Models*. https://doi.org/10.1109/ACCESS.2017.DOI

Pereira-Kohatsu, J. C., Quijano-Sánchez, L., Liberatore, F., & Camacho-Collados, M. (2019). Detecting and monitoring hate speech in twitter. *Sensors (Switzerland)*, *19*(21). https://doi.org/10.3390/s19214654

Pilehvar, M. T., & Camacho-Collados, J. (2020). *D R A F T Embeddings in Natural Language Processing Theory and Advances in Vector Representation of Meaning*.

Plaza-Del-Arco, F. M., Casavantes, M., Escalante, H. J., Martín-Valdivia, M. T., Montejo-Ráez, A., Montes-Y-Gómez, M., Jarquín-Vásquez, H., & Villaseñor-Pineda, L. (2021). Overview of MeOffendEs at IberLEF 2021: Offensive Language Detection in Spanish Variants. *Procesamiento Del Lenguaje Natural*, *67*, 183–194. https://doi.org/10.26342/2021-67-16

Poletto, F., Basile, V., Sanguinetti, M., Bosco, C., & Patti, V. (2021). Resources and benchmark corpora for hate speech detection: a systematic review. In *Language Resources and Evaluation* (Vol. 55, Issue 2, pp. 477–523). Springer Science and Business Media B.V. https://doi.org/10.1007/s10579-020-09502-8

Rodríguez-Sánchez, F., Carrillo-De-Albornoz, J., Plaza, L., Gonzalo, J., Rosso, P., Comet, M., & Donoso, T. (2021). Overview of EXIST 2021: sEXism Identification in Social neTworks. In *Procesamiento del Lenguaje Natural* (Vol. 67, pp. 195–207). Sociedad Espanola para el Procesamiento del Lenguaje Natural. https://doi.org/10.26342/2021-67-17