TRABAJOS FIN DE GRADO

Universidad de Jaén

Escuela Politécnica Superior de Jaén

Aplicación Web para el uso y mejora de sistemas clasificatorios de lenguaje ofensivo

Autor: Daniel Carrasco de la Chica

Grado: Ingeniería Informática

Directoras: Salud María Jiménez Zafra y María Dolores Molina González
Departamento de las directoras: Informática y Telecomunicaciones

Fecha: 30/06/2024

CREA

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Doña Salud María Jiménez Zafra y Doña María Dolores Molina González, tutoras del Trabajo Fin de Grado titulado: *Aplicación Web para el uso y mejora de sistemas clasificatorios de lenguaje ofensivo*, que presenta Daniel Carrasco de la Chica, autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, JUNIO de 2024

El alumno:

CARRASCO DE LA CHICA DANIEL - 77647963D
Firmado digitalmente por CARRASCO DE LA CHICA DANIEL - 77647963D
Fecha: 2024.07.02 02:25:15 +02'00'

Daniel Carrasco de la Chica

Las tutoras:

JIMENEZ ZAFRA SALUD MARIA - 77370897R
Firmado digitalmente por JIMENEZ ZAFRA SALUD MARIA - 77370897R
Fecha: 2024.07.01 14:28:42 +02'00'

Salud María Jiménez Zafra

MOLINA GONZALEZ MARIA DOLORES - 26009191D
Firmado digitalmente por MOLINA GONZALEZ MARIA DOLORES - 26009191D
Fecha: 2024.07.01 14:34:20 +02'00'

Mª Dolores Molina González

Agradecimientos

A mis tutoras Salud y Mª Dolores, por guiarme durante todos estos meses y por sus consejos, que han sido vitales para la realización de este trabajo.

A mi familia, por creer en mí y darme ánimos cuando más lo necesitaba.

A mis amigos, por haber sido un pilar fundamental en esta etapa, dándome energía tras largas jornadas de trabajo y esfuerzo. Esta motivación ha sido esencial para alcanzar el nivel de calidad del proyecto.

Especialmente a mi madre, cuya imagen, aun no presente, sigue iluminando mi camino. Este trabajo es un tributo a tu eterno cariño.

Índice de contenidos

Índice de ilustraciones

Índice de tablas

Fragmentos de código

1. Introducción

En la era digital, la comunicación en línea ha trascendido las barreras físicas, permitiendo el intercambio de ideas y opiniones en una escala sin precedentes. Sin embargo, esta libertad de expresión viene acompañada de desafíos importantes, uno de los cuales es el incremento del lenguaje ofensivo y el discurso de odio en plataformas digitales.

Según las Naciones Unidas [42], aunque a día de hoy no existe una definición universal de discurso de odio de acuerdo al derecho internacional en materia de derechos humanos, se propone una definición que abarca un sentido más amplio que "una instigación a la discriminación, la hostilidad o violencia". El discurso de odio se plantea como un tipo de comunicación que puede ser discriminatoria tanto para un individuo como un colectivo, así como basado en factores de identidad reales o percibidos como la religión, etnia, nacionalidad, raza…

Este Trabajo Fin de Grado (TFG) tiene como objetivo desarrollar varios sistemas de clasificación automática basados en técnicas avanzadas de aprendizaje profundo, capaces de reconocer el lenguaje ofensivo y determinar el tipo de víctima al que este va dirigido, ya sea un individuo o un colectivo. El proyecto se centra en la creación de varios sistemas clasificatorios que participen en las subtareas 1 y 2 de la tarea "MeOffendEs Offensive Language Detection in Spanish Variants" en la campaña de evaluación IberLEF 2021" [1]

A lo largo de este TFG, se abordan varios aspectos cruciales para la implementación de estos sistemas, incluyendo el desarrollo de una arquitectura de servicios distribuidos en el backend, el diseño de una capa de presentación moderna e intuitiva, y la integración de varios de los modelos de aprendizaje profundo desarrollados. Este enfoque no solo busca obtener al menos un modelo óptimo entre los desarrollados, sino también proporcionar una herramienta práctica y accesible para que cualquier usuario pueda acceder a ellos y al mismo tiempo pueda mejorarlos.

1.1. Motivación

Desde el comienzo de la era digital, la proliferación de dispositivos conectados ha transformado radicalmente nuestra manera de comunicarnos. Según datos de Marketing4Ecommerce, el número de usuarios de Internet y dispositivos conectados crece exponencialmente cada año, una tendencia que, aunque ha democratizado el acceso a la información, también ha ampliado el escenario para la extensión del discurso de odio.

Ilustración 1. Uso de dispositivos conectados en la actualidad

Este TFG se motiva por la paradoja de nuestra hiperconectividad, ya que mientras la tecnología ha potenciado la libertad de expresión, también ha facilitado la emergencia de discursos dañinos que atentan contra la dignidad humana. En todas las sociedades modernas prevalece un principio fundamental que es la libertad de expresión. Este principio inherente a la dignidad humana permite a los individuos compartir ideas, opiniones y críticas sin temor a censura o represión. Sin embargo, esta libertad no es absoluta y encuentra su límite cuando se convierte en un vehículo para el discurso de odio, es decir, cuando las palabras traspasan la crítica para fomentar la discriminación o la violencia contra individuos o colectivos en base a atributos como la religión, orientación sexual… entre otros.

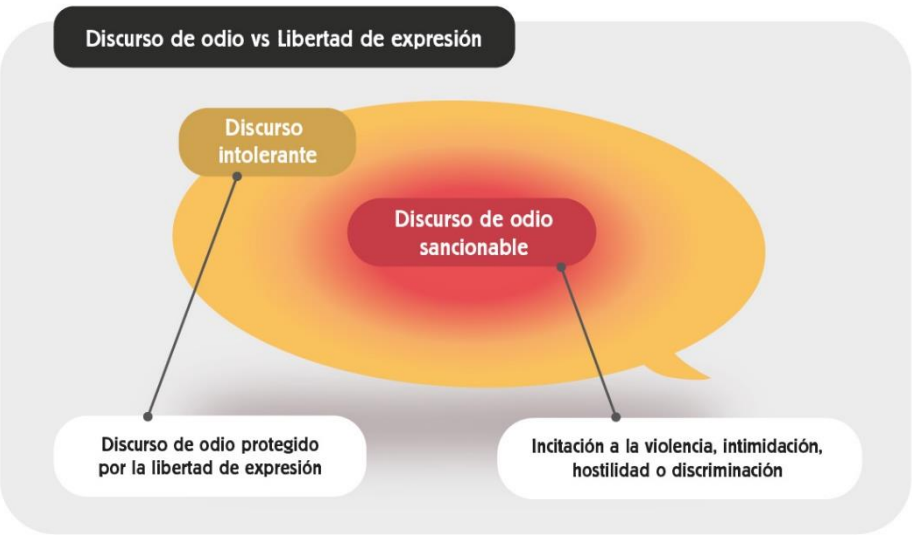


Ilustración 3. Diferencia entre libertad de expresión y discurso de odio

Las consecuencias derivadas de la proliferación de este tipo de discursos pueden plasmarse en la siguiente pirámide de odio. Como puede apreciarse en la **Ilustración 2**, unos simples prejuicios que no son visibles para la mayoría de la sociedad pueden escalonar en comportamientos y actitudes cada vez más graves. En su base, se encuentran los estereotipos que forman actitudes negativas. Subiendo en la pirámide, encontramos los actos de prejuicio como el acoso y el aislamiento social. En última instancia estos pueden intensificarse en discriminación, después en violencia física y en el extremo, pueden culminar en el genocidio, que busca la destrucción de un grupo.

Ilustración 2. Diferentes escalas de odio representadas en forma piramidal

1.2. Propósitos basados en las carencias del entorno actual

El campo de la detección de discurso de odio no es algo nuevo y ha existido desde hace muchos años. Sin embargo, uno de los mayores problemas reside en que las técnicas basadas en PLN, hasta no hace mucho, se centraban en métodos simples, a menudo dependientes únicamente de librerías externas como Spacy o Nltk, que se apoyan en el análisis estadístico y algoritmos de aprendizaje automático básicos. Estas aproximaciones, aunque son útiles, presentan limitaciones en su capacidad para comprender y procesar las sutilezas del lenguaje natural y el contexto en el que se producen los discursos de odio.

Otro de los mayores problemas encontrados, es la carente accesibilidad pública a estas herramientas avanzadas. La mayor parte de este tipo de soluciones quedan incrustadas en un artículo académico, no quedando claro ni su funcionamiento ni sus resultados para el usuario común, creando por tanto una brecha entre la tecnología y su aplicación práctica en la vida cotidiana.

Con el objetivo de superar esta barrera, en este TFG se propone el desarrollo de una herramienta capaz de clasificar el lenguaje ofensivo siendo accesible en una plataforma web completa. Esta plataforma no solo servirá como un punto de acceso a los modelos creados, sino como un laboratorio en vivo para estudiar y mejorar continuamente los modelos en un entorno real por cualquier usuario, potenciando su aplicabilidad y eficiencia.

Además, esta plataforma tendrá como objetivo proporcionar herramientas que fomenten la interacción social entre distintos usuarios para probar los modelos que publican otros usuarios, evaluarlos, mejorar sus datos de entrenamiento, así como proporcionar feedback.

El objetivo de esta interactividad es mucho mayor que proporcionar una interfaz amena para experimentar con distintos modelos. Para entender completamente el fin del proyecto y cómo proporciona un aspecto diferenciador respecto a numerosos modelos clasificatorios que existen, es necesario echar la vista no muy atrás a una de las mayores irrupciones dentro el ámbito tecnológico: ChatGPT.

ChatGPT, desarrollado por OpenAI, es una plataforma para probar modelos de lenguaje basados en la arquitectura GPT. Si bien es cierto, estos modelos otorgaban un gran rendimiento, este factor no fue aquel que hizo a OpenAI distanciarse con tanta diferencia de sus competidores. ChatGPT ofrecía una manera ingeniosa para acceder a modelos de lenguaje de gran escala, eliminando cualquier tipo de barrera y logrando 100 millones de usuarios activos mensuales en solo 2 meses, convirtiéndose en la aplicación de consumo de más rápido crecimiento de la historia.

Una de las innovaciones que ofrecía ChatGPT para fomentar el acceso a sus modelos de lenguaje fue proporcionar una interfaz sencilla y de uso gratuito, permitiendo que cualquier persona pudiera interactuar con estos avanzados modelos de Inteligencia Artificial (IA) sin necesidad de tener conocimientos técnicos profundos. Esto ha democratizado el acceso a tecnologías aún más avanzadas, permitiendo a usuarios de todo el mundo utilizar y experimentar con IA.

Esta analogía ha sido extrapolada al proyecto actual, las funcionalidades propuestas tienen como objetivo final fomentar el acceso y mejora de los modelos por parte de los propios clientes, gracias al uso de herramientas interactivas que permiten actualizar en vivo los datos con los que las IA son entrenadas. Además, se fomenta un feedback entre usuarios objetivo y los creadores de los modelos para que estos últimos puedan tener en cuenta casos específicos donde no se obtienen los resultados esperados con sus modelos.

Por ende, en resumen, este proyecto tendrá en cuenta la creación de avanzados modelos que clasifiquen el lenguaje ofensivo y, además, sean encapsulados en una interfaz disruptiva que permita su constante mejora gracias a una trabajada interactividad.

1.3. Objetivos

Los objetivos de este trabajo son:

1. Realizar un estudio sobre las técnicas más básicas para el Procesamiento del Lenguaje Natural que puedan ser aplicadas a la clasificación de lenguaje ofensivo.
2. Investigar acerca de las técnicas de vanguardia basadas en aprendizaje profundo que puedan ser aplicadas al Procesamiento del Lenguaje Natural.
3. Seleccionar un corpus de calidad para poder entrenar óptimamente los algoritmos de aprendizaje profundo.
4. Comparar los resultados obtenidos con los diferentes modelos de aprendizaje profundo y realizar un análisis de errores de los mismos.
5. Diseñar y desarrollar una plataforma web que sirva como punto de acceso a los modelos creados y que permita su estudio y mejora.
6. Redactar una memoria que recoja todo el trabajo realizado.

1.4. Resultados esperados

Los resultados esperados tras la finalización del trabajo son:

- Adquirir un conocimiento lo suficientemente asentado para ser capaz de encontrar los límites de las técnicas empleadas hasta ahora.
- Un sistema de clasificación de textos para clasificar lenguaje ofensivo usando técnicas de vanguardia.
- Una aplicación web completamente funcional, con una base de datos, backend y frontend que faciliten el acceso al sistema.
- Memoria final del proyecto donde se detalle todo el proceso seguido.
- Manual de instalación para explicar cómo probar el conjunto de las aplicaciones en local.

1.5. Metodologías software

Una metodología es un sistema de prácticas, técnicas, procedimientos y reglas que se utilizan para organizar el trabajo en la realización de proyectos de investigación o en el desarrollo de software. En este apartado se define el enfoque estructurado para abordar y resolver los problemas planteados.

1.5.1. Metodologías tradicionales vs metodologías ágiles

Las metodologías tradicionales en la gestión de proyectos, pero sobre todo en el desarrollo de software, incluyen principalmente modelos como el modelo en cascada. Este modelo ha quedado prácticamente en desuso en muchos proyectos de la actualidad por culpa de su carácter tan restrictivo. En un modelo en cascada es necesario terminar cada una de las fases antes de concurrir a la siguiente, por tanto, aunque se previenen los errores en los proyectos, su desarrollo es demasiado rígido.

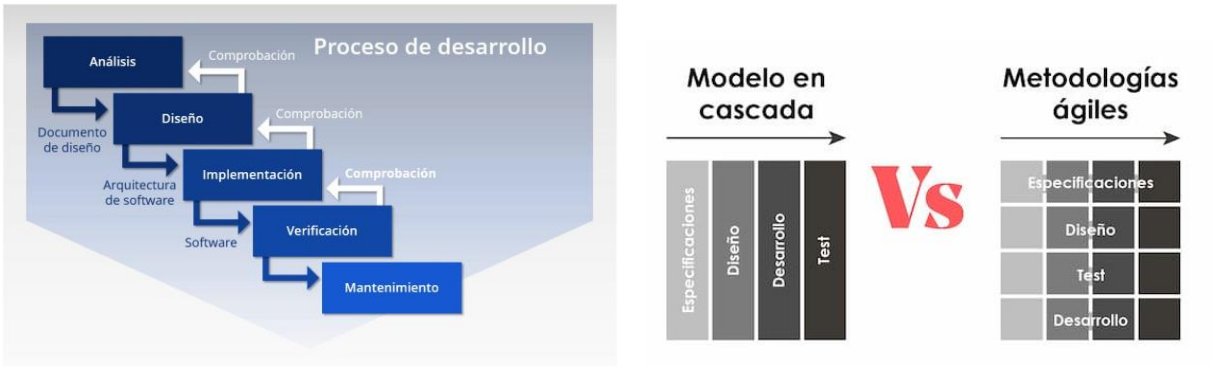


Ilustración 4. Etapas de un modelo en cascada y comparativa de modelos en cascada vs metodologías ágiles

Esta rigidez supone un enorme problema en el ámbito de desarrollo de software por muchos motivos:

- En la gran mayoría de ocasiones **los requerimientos no son estables**, bien porque los requisitos cambien o porque estos sean inciertos desde el inicio, por tanto, en muchas ocasiones se requiere volver a etapas previas. Sin embargo, el modelo en cascada no permite volver atrás a una etapa fácilmente una vez ésta ha sido completada.
- El software en la actualidad se actualiza reiteradamente en un intervalo de tiempo muy corto, al usar modelos tan rígidos, los **proyectos finalizados** pueden quedar **totalmente desfasados**.

- En muchas ocasiones, el producto final solo se ve al final del proyecto, lo cual puede resultar en productos que no se alinean totalmente con las necesidades del cliente provocando un **retraso en la entrega.**

Las metodologías ágiles suponen un gran avance para paliar los defectos encontrados en las tradicionales. Estas tienen como objetivo principal conseguir una entrega temprana y continua de software con valor.

Las principales metodologías ágiles son:

- SCRUM
- Kanban
- Lean
- XP

1.5.2. Kanban

Kanban es una metodología centrada en la gestión visual del trabajo mediante un sistema de tarjetas (kanban). Esta metodología es eficaz en proyectos que requieren una flexibilidad constante y flujo continuo, facilitando la gestión de trabajo.

Ilustración 5. Metodología Kanban

Sin embargo, cuenta con algunas desventajas que me han impedido usarlo:

- A diferencia de otras metodologías, **Kanban no tiene Sprints**, por lo que es menos ideal en proyectos como este que se benefician de revisiones y ajustes periódicos
- Se requiere una disciplina rigurosa para gestionar y priorizar eficazmente las tareas, **dependiendo más de la auto organización** y menos de las estructuras temporales fijas. Por tanto, aunque a simple vista Kanban es más flexible, es potencialmente menos predecible en términos de progreso del proyecto

1.5.3. Lean

Lean es una metodología enfocada en la eficiencia. Su objetivo es maximizar el valor del cliente mientras se minimiza el desperdicio, es decir, cualquier recurso que no contribuya al valor final del producto. Se centra en la mejora continua y en la eliminación de procesos que no generan valor.

Ilustración 6. Metodología Lean

Lean, sin embargo, también cuenta con algunos inconvenientes:

- Lean no se centra en los ciclos de desarrollo iterativos y evaluativos sino en la **eficiencia y la eliminación de desperdicios**.
- Aunque se promueve la eficiencia, **podría no proporcionar suficiente estructura para la exploración y experimentación** que requieren el desarrollo y validación de los modelos de IA desarrollados. En el contexto del desarrollo de clasificadores avanzados, procesos como la experimentación e iteración (ajuste de hiperparámetros) son cruciales para optimizar y validar estos sistemas. Sin embargo, estas actividades podrían ser vistas como desperdicios desde una perspectiva Lean tradicional.

1.5.4. XP

XP (Extreme Programming) es una metodología que se enfoca en la mejora de la calidad del software y la capacidad de respuesta a los cambios en los requisitos del cliente. XP enfatiza la comunicación continua, el feedback y la simplicidad en el diseño. Se basa en principios como programación en pareja, desarrollo impulsado por pruebas, y lanzamientos frecuentes que permiten cambios rápidos y eficaces.

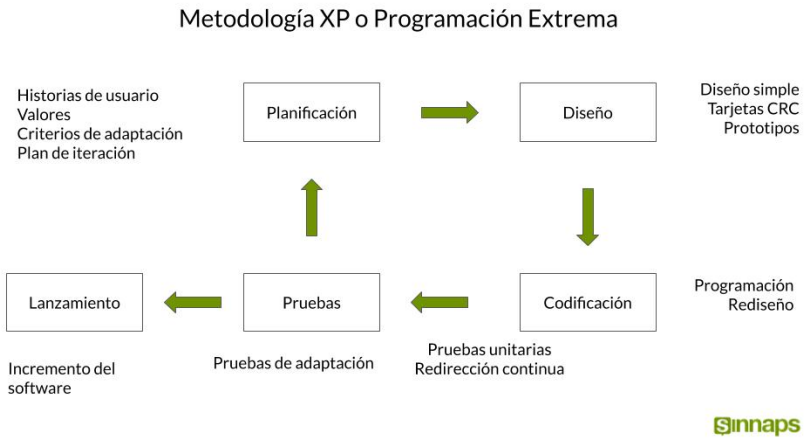


Ilustración 7. Metodología XP

De nuevo, esta metodología presenta problemas:

- Esta metodología se centra mucho en las **buenas prácticas de codificación** y no se centra tanto en la gestión del proyecto en sí.

1.5.5. SCRUM

La metodología ágil empleada en este proyecto ha sido SCRUM. SCRUM es un marco de trabajo que permite entregar productos complejos mediante ciclos de desarrollo iterativos y entregas incrementales, conocidos como Sprints. Esta metodología es especialmente adecuada en proyectos como este debido a su naturaleza iterativa e incremental.

- **Enfoque incremental:** Consiste en dividir el trabajo en módulos que puedan ser construidos independientemente, integrándose junto a los demás una vez el proyecto esté acabado. Este proyecto está dividido en una serie de componentes individuales como son el modelo de detección de lenguaje ofensivo, el backend y finalmente el frontend de la plataforma web. Esto permite concentrarse en cada parte de manera paralela y ver progresos concretos, integrando y probando cada módulo de manera sistemática.
- **Enfoque iterativo:** Consiste en dividir cada módulo en pequeñas iteraciones o ciclos, cada iteración es un proceso de planificación, diseño, implementación y pruebas en un ciclo de tiempo específico en el que se pueden aplicar mejoras. En cada iteración, se evalúan y perfeccionan los componentes, lo que define el sistema.

Para comprender SCRUM es vital comprender bien cuáles son sus roles principales: Scrum Master, Product Owner y Equipo de desarrollo. En este proyecto yo he asumido la totalidad de los roles.

- **Scrum Master:** Es el responsable de facilitar el proceso de Scrum, ayudando al equipo a trabajar de manera eficiente, eliminando obstáculos y asegurando que se sigan las prácticas de Scrum.
- **Product Owner:** Actúa como el enlace entre el equipo y las partes interesadas. Es responsable de definir los requisitos del proyecto, priorizar la lista de tareas del sprint y asegurar que el trabajo entregado maximice el valor para el negocio.
- **Equipo de desarrollo:** Un grupo autoorganizado y multifuncional que realiza el trabajo técnico. No hay roles fijos dentro del equipo; todos colaboran para completar el conjunto de tareas seleccionadas para cada Sprint.

2. Fundamentos del PLN y aplicación para clasificación del lenguaje ofensivo

2.1. Introducción al PLN

El Procesamiento del Lenguaje Natural (PLN) es un subcampo de la informática y de la IA que se centra en la interacción entre las computadoras y el lenguaje humano. El objetivo principal del PLN es permitir que las máquinas comprendan, interpreten y generen lenguaje humano de manera útil.

El PLN permite que computadoras y dispositivos digitales comprendan y generen información a partir de texto o voz, entre otros, combinando la lingüística computacional junto al modelado estadístico, el aprendizaje automático y el aprendizaje profundo.

El campo del PLN ha evolucionado significativamente desde su origen en la década de 1950. Los primeros sistemas de PLN se basaban en reglas lingüísticas estrictas, pero a lo largo de las décadas, el enfoque ha cambiado hacia métodos estadísticos y de aprendizaje automático. Con la llegada del aprendizaje profundo, el PLN ha experimentado avances notables, permitiendo el desarrollo de sistemas mucho más precisos.

Los primeros sistemas de PLN, nacidos en la década de los 60s, se inspiraron en la teoría de la gramática transformacional de Chomsky y en algoritmos basados en árboles de análisis sintáctico. La teoría de la gramática transformacional de Noam Chomsky se centraba en cómo las estructuras sintácticas del lenguaje podían ser generadas y transformadas mediante reglas gramaticales específicas.

Esta teoría distingue entre la estructura profunda y superficial de una oración. La estructura profunda representa el significado abstracto, mientras que la estructura superficial es la forma final de la oración que se pronuncia o se escribe. Además, Chomsky propuso que un conjunto finito de reglas gramaticales puede generar una cantidad infinita de oraciones gramaticalmente correctas. Este concepto fue trascendental para entender cómo los humanos pueden producir y comprender oraciones nuevas.

La teoría de la gramática transformacional de Chomsky influyó directamente en el desarrollo de algoritmos para el análisis sintáctico, también conocidos como parsing. Los primeros sistemas de PLN desarrollados incorporaron estos conceptos para analizar y generar lenguaje natural. Uno de los sistemas más influyentes fue ELIZA, desarrollado por Joseph Weizenbaum en 1966, que utilizaba reglas simples basadas en patrones para simular una conversación con un psicoterapeuta.

En la década de los 80 y 90, el PLN experimentó un renacimiento estadístico. Estas décadas estuvieron marcadas por el cambio hacia enfoques basados en datos a través de grandes corpus de texto y capacidades computacionales mejoradas surgiendo los primeros modelos estadísticos de traducción automática, los modelos de Markov Ocultos y los modelos n-gram.

Sin embargo, el mayor avance dentro del campo del PLN data de la década del 2010 a la actualidad, gracias al aprendizaje automático y profundo. En unos pocos años hemos sido testigos de arquitecturas altamente sofisticadas como las redes neuronales recurrentes y las redes de memoria a corto plazo. Sin embargo, los mayores avances en este campo han ido de la mano de grandes modelos de lenguaje preentrenados como son GPT, BERT o T5 que han surgido en el último lustro. Estos modelos no solo han batido récords en diversas tareas de PLN, sino que han revolucionado la forma en la que abordamos el lenguaje.

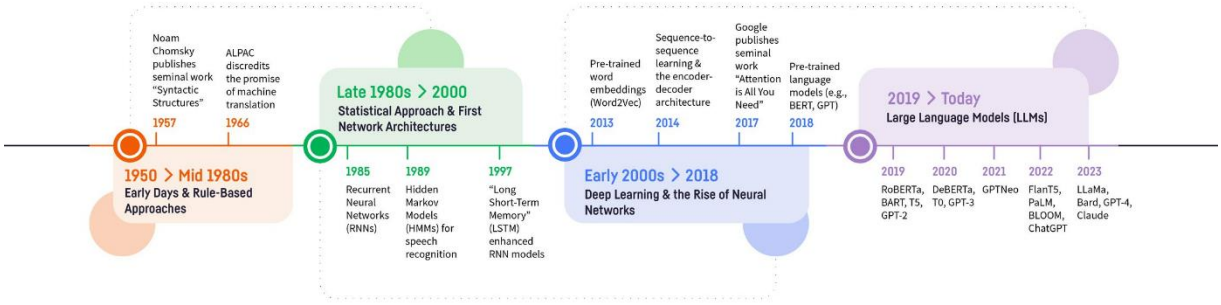


Ilustración 8. Evolución del Procesamiento del Lenguaje Natural

2.2. Aplicaciones del PLN

Los grandes avances experimentados en el campo del PLN han permitido su aplicación directa en una amplia gama de áreas debido a su capacidad para comprender y generar lenguaje humano. El uso de técnicas de procesamiento del lenguaje natural no solo se ha masificado por la comodidad que nos ofrecen, sino también por el paradigma actual de la información, conocido como Big Data. En la actualidad, nos enfrentamos a una ingente cantidad de información no estructurada, que resulta demasiado compleja para ser manejada por los sistemas de procesamiento de datos tradicionales.

La aplicación del PLN en el procesamiento de estos datos, permite extraer conocimientos útiles y realizar tareas sobre grandes volúmenes de datos textuales de forma eficiente. A continuación, se exponen algunos de los usos más prácticos del uso de estas técnicas.

- **Traducción automática:** La traducción automática fue una de las primeras aplicaciones que se le dio al PLN, su aplicación es especialmente útil para las empresas gracias a que facilita la comunicación permitiendo llegar a audiencias más amplias.
- **Resumen de textos:** El resumen automático de textos permite extraer la información más importante. Su objetivo es simplificar el proceso de revisión de grandes cantidades de datos como artículos científicos.
- **Estudios de mercado:** Los especialistas en marketing pueden beneficiarse del PLN para aprender más sobre sus potenciales clientes y utilizar estos conocimientos para crear estrategias más efectivas.
- **Clasificación textual:** Las técnicas de PLN permiten desarrollar herramientas capaces de analizar sentimientos o clasificar textos en diferentes categorías. Para este trabajo, se han elegido técnicas de PLN que permitan clasificar textos en varias categorías de ofensividad.

2.3. Elementos en los sistemas clasificatorios

En los sistemas de PLN, existen varios conceptos clave fundamentales para su funcionamiento. A continuación, se detallan algunos de los elementos más importantes y su aplicación directa en los sistemas clasificatorios.

- **Token:** Un token es una unidad básica de texto que el sistema considera para el procesamiento de textos. Un token puede ser una palabra, subpalabra, símbolo, carácter o incluso una combinación de estos. La estructura de un token no está estrictamente definida y su efectividad dependerá directamente del tipo de tarea asociada.
- **Tokenización:** La tokenización constituye el proceso de dividir una secuencia de texto en unidades más pequeñas, los tokens. Este paso es absolutamente crucial independientemente de la arquitectura usada en el modelo clasificador, ya que transforma el texto en unidades procesables para dichos modelos.

Como se ha descrito, la definición de los tokens y el proceso de tokenización en un sistema clasificador resulta un paso crucial para determinar la calidad de los modelos. Sin embargo, su elección depende de muchos factores externos a la arquitectura del sistema, como por ejemplo el idioma de los textos, la longitud de la información representada, entre otros.

2.4. Introducción a los modelos de clasificación

La elección del modelo clasificador óptimo es una tarea ardua que requiere un largo proceso de investigación. Por este motivo, a lo largo de este TFG se detalla con todo lujo de detalles el proceso para seleccionar el modelo óptimo para clasificar el lenguaje ofensivo. Sin embargo, es importante esbozar los aspectos más básicos de los principales modelos clasificadores para mostrar los conceptos principales que deberemos abordar para la gran mayoría de ellos, así como los impedimentos que encontramos en algunos.

- **Modelos basados en reglas:** Los primeros modelos de PLN, utilizan conjuntos de reglas gramaticales y sintácticas. Aunque son modelos muy básicos y actualmente desfasados, sentaron las bases para desarrollos posteriores.
- **Modelos estadísticos:** Utilizan técnicas estadísticas para modelar lenguaje, dependiendo de la probabilidad de ocurrencia de secuencias de palabras. Fueron modelos muy utilizados hasta hace poco tiempo, sin embargo, la irrupción de modelos más avanzados en los últimos años los ha dejado en desuso.
- **Modelos de aprendizaje automático:** Utilizan algoritmos como Naive Bayes o Máquinas de Vectores de Soporte (SVM), que es uno de los más utilizados. Estos modelos ofrecen buenos resultados a un bajo coste computacional, pero han sido superados por modelos más avanzados.
- **Modelos de aprendizaje profundo:** Utilizan redes neuronales profundas para modelar el lenguaje, incluyendo Redes Neuronales Recurrentes (RNN), Memoria a largo plazo (LSTM), unidades recurrentes cerradas (GRU) y Transformers. Son modelos mucho más sofisticados que los anteriores, pero también mucho más costosos computacionalmente.

Los 3 últimos modelos, aun con sus diferencias, tienen un elemento común extremadamente importante, la necesidad de un corpus de procesamiento.

Un corpus es una colección grande y estructurada de textos que se utiliza en el campo de PLN para entrenar y evaluar modelos de lenguaje. Los corpus son fundamentales para desarrollar estos sistemas porque proporcionan los datos necesarios para que los algoritmos aprendan patrones y características del lenguaje humano.

Un corpus presenta las siguientes características:

- Dependiendo de su propósito, un corpus puede contener desde **cientos hasta millones de palabras o documentos**. Un corpus más grande suele dar mejores resultados ya que proporciona más datos para el entrenamiento. Sin embargo, no solo importa el tamaño, un corpus debe tener suficiente calidad para ofrecer los resultados deseados. Un corpus grande con muchos errores gramaticales, puede proporcionar peores resultados que un corpus de menor tamaño y mejor calidad.
- Un corpus puede contener **información de noticias, libros, discursos, publicaciones en redes sociales, correos electrónicos**, entre otros.
- Los corpus se almacenan en **distintos formatos**, pudiendo ser archivos de texto plano, XML, TSV, JSON, etc.
- Un corpus **puede estar etiquetado o no**, un corpus etiquetado incluye metadatos adicionales que proporcionan información sobre el texto. Los modelos basados en aprendizaje automático utilizan corpus no etiquetados mientras que los basados en aprendizaje profundo usan corpus con datos etiquetados. Por esto, los modelos de aprendizaje profundo suelen ofrecer mejores resultados siempre que el corpus cumpla unos mínimos estándares de calidad.
- Un corpus debe ser **suficientemente diverso**, esto significa que debe cubrir una amplia gama de estilos, registros, jerga e información relevante para la aplicación específica.

Por tanto, la elección y extracción de un corpus de calidad supone un proceso igual o más importante que la elección de un buen modelo clasificador.

2.5. Limitaciones de los modelos clasificatorios más avanzados

A lo largo de los últimos años, el campo del PLN ha crecido exponencialmente en los resultados que ofrecen sus modelos, así como en su complejidad y coste asociado. Para muchos desarrolladores individuales, entrenar modelos de estas características es imposible debido a varias limitaciones. A continuación, se detallan algunas de estas limitaciones a través de varias ilustraciones de interés.

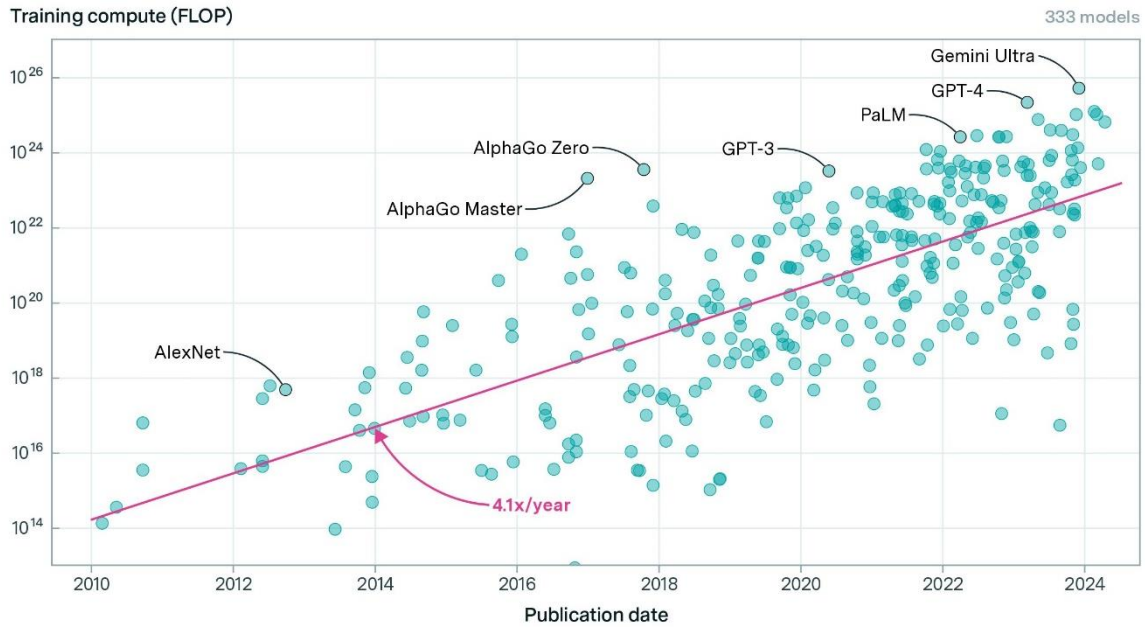


Ilustración 9. Evolución coste computacional LLM

En la **Ilustración 9** se muestra el avance de la potencia computacional requerida para entrenar algunos de los grandes modelos de lenguaje (LLM) más importantes a lo largo del tiempo. El entrenamiento de modelos como GPT-3 requirió un costo computacional alrededor de $3{,}14^{24}$ flops, lo que implicó el uso de miles de tarjetas gráficas NVIDIA[1] V100 para su entrenamiento durante meses. Para este trabajo, aun contando con una tarjeta gráfica bastante potente, solo tenemos el equivalente a aproximadamente $1{,}59^{13}$ flops, lo que representa aproximadamente un 0,0000000022 % de la potencia necesaria para entrenar GPT-3.

Sin embargo, vemos que la tendencia no se detiene, surgiendo modelos aún más grandes que GPT-3 que crecen a un ritmo de 4,1 veces en coste computacional anual.

[1] https://www.nvidia.com/es-es/

Aunque es evidente que el enorme coste computacional del entrenamiento de los LLM es inviable para equipos pequeños, este no es el único factor limitante.

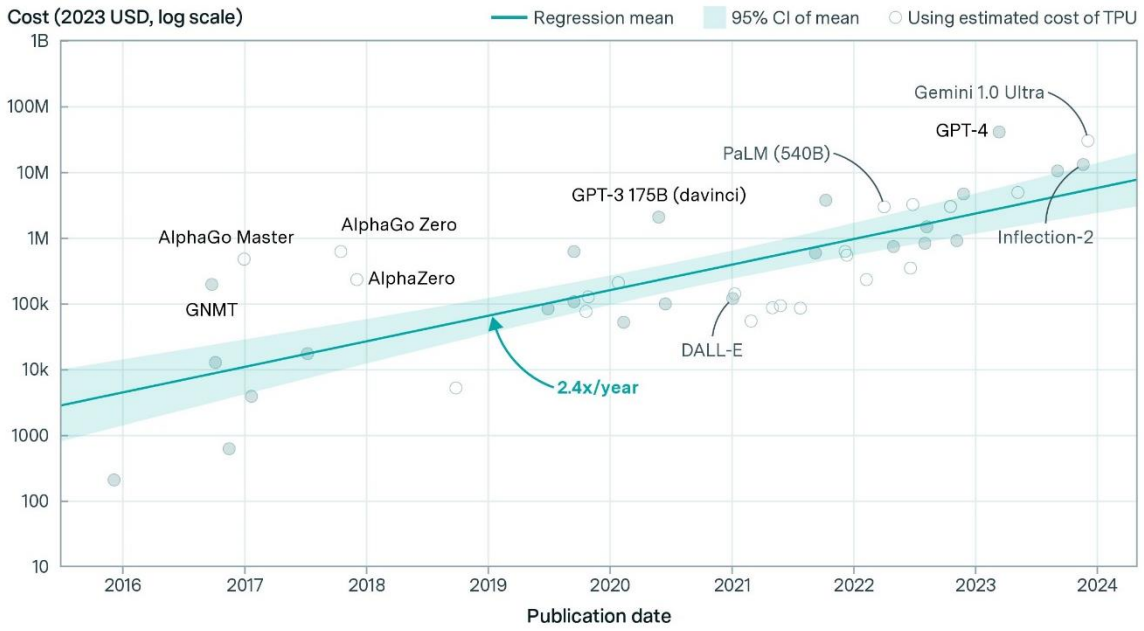


Ilustración 10. Evolución coste de entrenamiento de LLM

En la **Ilustración 10** podemos apreciar que el coste económico es otro gran problema intrínseco a los LLM. Modelos muy recientes como GPT-4 rozan la cifra de 100 millones de dólares exclusivamente para entrenarlos, rivalizando en costes con las grandes producciones de Hollywood o de videojuegos.

Dadas estas limitaciones, muchos desarrolladores optan por utilizar modelos más accesibles y menos costosos. En este proyecto se ha optado por probar como modelos más avanzados los derivados de BERT, que son más manejables en términos de coste computacional y tiempo de entrenamiento. Estos modelos, que serán descritos posteriormente con mucho más detalle, utilizan solo encoders, lo cual los hace menos costosos en comparación con los modelos más complejos como GPT, que incluyen tanto encoders como decoders. Además, estos modelos pueden otorgar una efectividad muy similar en tareas específicas como la clasificación de lenguaje ofensivo, frente a los LLM que son generalistas.

2.6. Competiciones en PLN

En el contexto del PLN, las competiciones juegan un papel crucial para evaluar diferentes modelos. Estas competiciones proporcionan un entorno controlado para que los investigadores y desarrolladores puedan probar sus modelos utilizando un conjunto común de datos y métricas de evaluación, permitiendo por tanto una comparación objetiva del rendimiento de los diferentes sistemas.

Las competiciones suelen estar organizadas en torno a tareas específicas de PLN, como la clasificación de textos, la traducción automática, el análisis de sentimientos, entre otras. Cada competición incluye un conjunto de datos de entrenamiento y otro de prueba, además de unas reglas claras de cómo deben evaluarse los resultados.

Para este proyecto, se ha utilizado la competición MeOffendEs organizada en IberLEF 2021. Esta competición se centra en la detección de lenguaje ofensivo en variantes del español y cuenta con varias subcategorías:

- **Subtarea 1 y 2:** Se dedica a la clasificación multicategoría de textos en español genérico. Estas categorías se detallan en el apartado **6.1**
- **Subtarea 3 y 4:** Se dedica a la clasificación binaria de textos ofensivos en español mexicano con y sin contexto adicional.

Para este TFG se ha optado por elaborar modelos que sean capaces de competir con los vencedores en la subtarea 1 y 2.

3. Tecnologías y herramientas utilizadas

Para el desarrollo del proyecto, la elección de las tecnologías y herramientas es un factor clave para asegurar la construcción del sistema. Este apartado detalla las diversas tecnologías empleadas para implementar los algoritmos de clasificación de lenguaje ofensivo, así como para el desarrollo del frontend y backend de la aplicación que los alberga.

La correcta elección de las tecnologías no solo facilita el desarrollo del sistema, sino que garantiza que las funcionalidades requeridas se implementen de manera efectiva. En este contexto, se han utilizado diferentes frameworks, librerías y servicios en la nube cuya elección no se rige por abordar los desafíos que se presentan en este trabajo, sino también por contar con un amplio soporte por parte de la comunidad de desarrolladores. Esto las convierte en herramientas software modernas, versátiles y actualmente utilizadas y demandadas.

3.1. Clasificación de lenguaje ofensivo

3.1.1. Python

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general. Python fue creado por Guido van Rossum y su primera versión fue lanzada en 1991, desde ahí ha adquirido una gran popularidad gracias a su simplicidad y legibilidad.

Python es un lenguaje de programación compatible con la gran mayoría de sistemas operativos y que además cuenta con una sintaxis fácil de leer y escribir, lo cual permite que sea un lenguaje de programación altamente accesible y con una curva de aprendizaje menor en comparación a otros lenguajes de programación no interpretados como pueden ser C++ o Java.

Gracias a esta gran accesibilidad, Python es un lenguaje con una comunidad de desarrolladores inmensa que han creado una enorme biblioteca de módulos y paquetes. Estas bibliotecas permiten a los desarrolladores construir herramientas para manipulación de archivos, desarrollo web, ciencia de datos y tareas de Machine Learning (ML) entre otros de manera mucho más sencilla y efectiva que sus competidores.

Esto puede ejemplificarse si atendemos al siguiente gráfico de la popular herramienta de ML Anaconda, la cual hizo una encuesta a sus usuarios sobre los lenguajes de programación que utilizaban.

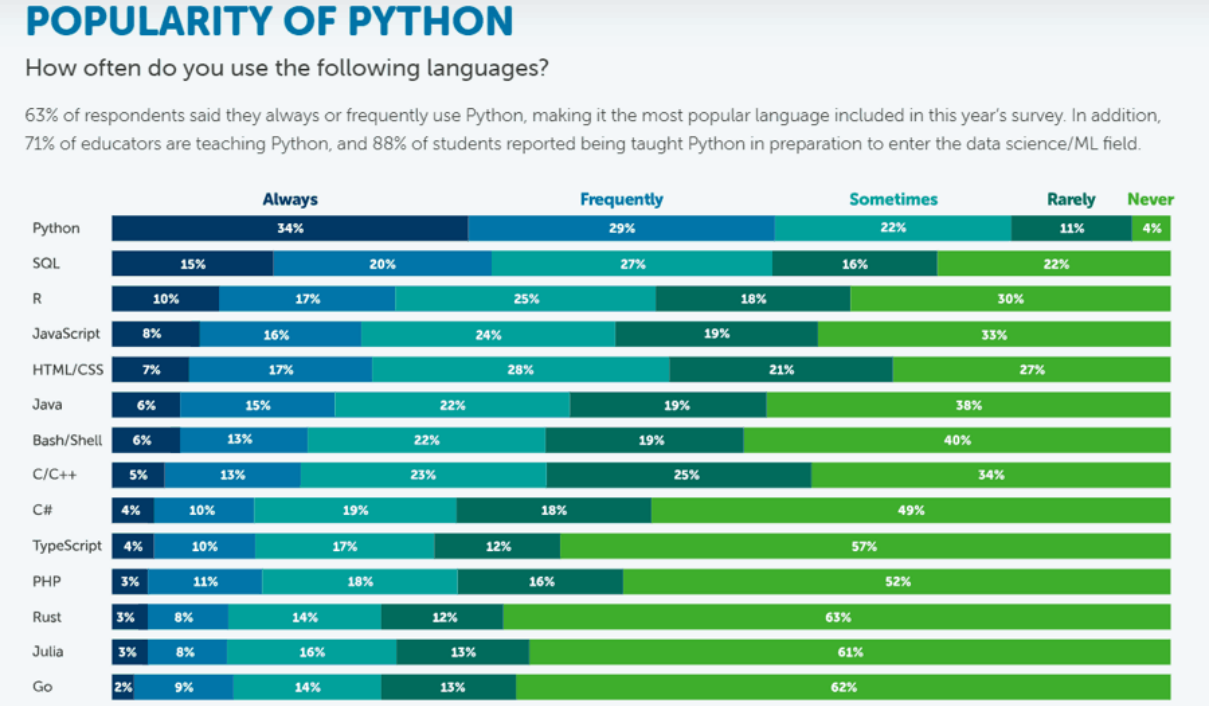


Ilustración 11. Popularidad de Python

Muchas de estas librerías utilizadas como Sklearn, PyTorch o Flask han sido un factor diferencial para el correcto desarrollo de gran parte de las funcionalidades principales de este TFG, las cuales no solo incluyen la creación de modelos clasificatorios del lenguaje ofensivo, sino que además han permitido exponer estas funcionalidades a los clientes para poder ser utilizadas en la web.

Sin embargo, aunque Python constituye las bases para el desarrollo de software de manera muy efectiva y en poco tiempo, su uso se ha limitado a algunas funciones del backend debido a las siguientes desventajas:

- Al ser un lenguaje interpretado, **Python es más lento que algunos lenguajes compilados como Java** ya que el código se ejecuta en un intérprete en tiempo de ejecución, añadiendo una sobrecarga adicional ya que en cada ejecución existen traducciones de código fuente a instrucciones ejecutables. Esta sobrecarga se traduce no sólo en términos de rendimiento sino en consumo energético.

- **Las estructuras de datos en Python son más pesadas** en términos de memoria que algunas de la competencia.
- La gran flexibilidad de Python **dificulta las optimizaciones a bajo nivel** durante la compilación y ejecución.
- **Muchas librerías están escritas en C** y aunque están altamente optimizadas, la interoperabilidad entre Python y C añade una sobrecarga adicional.

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

Ilustración 12. Comparativa coste energía de distintos lenguajes de programación

Debido a esta ineficiencia, se ha usado Python para lo mínimo e indispensable. Como lo descrito anteriormente, para la creación de modelos de clasificación o la exposición a la web de estas funcionalidades. Para conseguirlo, se han usado los siguientes módulos.

3.1.2. Flask

Flask es un microframework web escrito en Python utilizado en el desarrollo de aplicaciones web ligeras y sencillas, fue creado por Armin Ronacher como parte del proyecto Pocoo y su primera versión data del año 2010. Flask se destaca por su simplicidad, flexibilidad y facilidad de uso.

Esta sencillez ha sido clave para elegir a Flask como framework, si bien es sabido que existen otros framework mucho más potentes en Python como Django, usar estos frameworks tan complejos sería un enorme desperdicio considerando que el sistema no requiere de una alta complejidad para exponer al público las funcionalidades de los sistemas clasificadores, más aun teniendo en cuenta la intrínseca ineficiencia en la que se sustentan las aplicaciones web desarrolladas en Python, lo cual fue expuesto en la **Ilustración 12**.

Flask se considera como microframework debido a que mantiene un núcleo pequeño pero extensible, es decir, contiene lo esencial para iniciar una aplicación web sin imponer dependencias ni decisiones de diseño innecesarias.

Este principio de contener “lo mínimo para empezar” permite a los desarrolladores estructurar sus aplicaciones de la manera que prefieren, no imponiendo una arquitectura específica y otorgando por ende una gran flexibilidad para adaptar la estructura a las necesidades de este proyecto.

Otra de las ventajas que contiene Flask es permitir definir rutas de manera sencilla a cualquier método integrado en la aplicación, lo que facilita exponer en una API pública todas las funcionalidades de los modelos clasificadores aun sin haber sido concebidos en un principio para ser albergados en una aplicación web.

```
1. @app.route('/prediccion', methods=['POST'])
2. def predict_route():
3.     if not current_model:
4.         return jsonify({'error': 'No hay modelo cargado'}), 400
5.
6.     json_input = request.get_json()
7.     text = json_input['text']
8.     print("USANDO MODELO: "+str(current_model['path']))
9.     predicted_label_index, _ = predecir(text, current_model['tokenizer'],
current_model['modelo'], current_model['device'])
10.     etiqueta_a_indice_inverso = {0: "OFP", 1: "OFG", 2: "NO", 3: "NOE"}
11.     predicted_label = etiqueta_a_indice_inverso[predicted_label_index]
12.
13.     return jsonify({'prediction': predicted_label}), 200
```

Código 1. Método expuesto por Flask

Tal y como puede apreciarse, en el anterior código se ha expuesto a una API pública la funcionalidad de predecir una frase únicamente incluyendo la línea nº1, que contiene el endpoint deseado.

3.1.3. CSV

Csv es una librería ampliamente utilizada en Python que permite leer y escribir archivos CSV. En este proyecto, csv se utiliza para cargar los datos de entrada (comentarios y etiquetas) desde archivos en formato TSV para su posterior procesamiento y clasificación.

3.1.4. Pytorch y Torch

Torch es posiblemente una de las librerías más potentes junto a Tensorflow que se utilizan en la actualidad para tareas de ML. Esta librería es la principal del framework Pytorch, utilizado para el cálculo de tensores y el desarrollo de modelos de aprendizaje profundo, permite una flexibilidad y eficiencia significativas para construir los modelos propuestos de aprendizaje profundo.

Esta biblioteca de código abierto desarrollada por Facebook ofrece una sintaxis más intuitiva y amigable para los desarrolladores que su principal competidora, Tensorflow. Esto la hace especialmente popular para proyectos iniciales de investigación y de introducción al ML.

Pytorch proporciona un modelo de ejecución imperativa, permitiendo a las operaciones ejecutarse inmediatamente lo que facilita una depuración mucho más sencilla y en tiempo real. Sin embargo, la mayor ventaja que ofrece Pytorch es sin duda el aprovechamiento de la unidad de procesamiento gráfico (GPU) con CUDA. CUDA es una plataforma de computación paralela y una API desarrollada por NVIDIA en 2009 que permite a los desarrolladores utilizar la potencia de procesamiento de las tarjetas gráficas para realizar cálculos generales más allá de la típica rasterización de gráficos.

Para entender el porqué de la utilización de este framework, es vital aclarar las diferencias más básicas entre una GPU o una CPU en el procesamiento de este tipo de operaciones. Una CPU contiene una pequeña cantidad de núcleos extremadamente potentes, mientras que una GPU cuenta con miles de núcleos menos

potentes. Las operaciones de aprendizaje profundo como la multiplicación de matrices y las convoluciones pueden ser fácilmente paralelizadas. Tener un gran número de núcleos permite dividir estas operaciones en tareas menores que se ejecutan simultáneamente. Estas tareas se clasifican en batches y cada batch puede ser dividido y distribuido entre múltiples núcleos.

Por tanto, para tareas intrínsecamente secuenciales, es preferible utilizar una CPU. Sin embargo, la naturaleza de las operaciones de Deep Learning (DL) es concurrente y no secuencial, lo que materializa la necesidad de entrenar estos modelos en GPUs.

Gracias a Pytorch los modelos clasificatorios pueden ser entrenados en tensores de precisión simple (FP32) o precisión media (FP16). La elección de la precisión del tensor depende de varios factores y determina la velocidad y la calidad del entrenamiento de un modelo de aprendizaje profundo. Es importante tener en cuenta que, aunque el incremento de precisión a menudo mejora la calidad del modelo resultante, también es inversamente proporcional a la velocidad de entrenamiento. Por tanto, en modelos de DL, optar por precisión doble (FP64) es algo completamente innecesario, utilizando en su lugar precisiones más bajas para acelerar el entrenamiento de modelos cada vez más grandes.

Un tensor es una estructura de datos que generaliza los conceptos de escalares, vectores y matrices a dimensiones superiores. Es un componente fundamental en el campo del Procesamiento del Lenguaje Natural. Estos tensores pueden ser de orden 0 (escalares), orden 1 (vectores), orden 2 (matrices) y de dimensiones superiores.

En este trabajo se ha optado por utilizar tensores de precisión en punto flotante de 32 bits, lo cual ofrece un buen equilibrio entre precisión y rendimiento bajo una dimensionalidad de orden 2.

3.1.4. Transformers

Esta librería de HuggingFace permite usar modelos preentrenados basados en la arquitectura Transformer para realizar tareas de PLN. Esta herramienta además proporciona tokenizadores adaptados para cada modelo transformer.

3.1.5. Matplotlib

Matplotlib es una librería de visualización de datos que permite crear gráficos estáticos, animados e interactivos en Python. Su uso es de vital importancia para generar gráficos AUC ROC y las matrices de confusión de los modelos entrenados

3.2. Desarrollo de la aplicación

Para poder cumplir los objetivos de este trabajo, es crucial desarrollar una aplicación con una arquitectura bien estructurada que garantice la escalabilidad, mantenibilidad y rendimiento del sistema. Para poder cumplir los estándares de calidad deseados en este TFG se ha optado por desarrollar una aplicación empresarial bajo una arquitectura de 3 capas por el lado del servidor, integrando tecnologías modernas como React para el frontend y Spring Boot para el backend.

3.2.1. Introducción a las aplicaciones empresariales

El desarrollo de aplicaciones web ha evolucionado drásticamente en las últimas décadas. Antes de que se masificara el uso de las aplicaciones empresariales, existían 2 arquitecturas comúnmente utilizadas.

- **Arquitectura cliente-servidor de 2 capas:** En esta arquitectura se dividía el sistema en 2 componentes principales, el cliente que interactuaba directamente con el usuario, y el servidor, que era el encargado de gestionar y almacenar los datos. Esto tenía muchos inconvenientes, para empezar, el servidor solo almacenaba datos sin realizar ningún procesamiento de la lógica de negocio, esto significaba que la lógica de negocio se gestionaba directamente en el lado del cliente, generando redundancia y dificultad en su mantenimiento. Además, esta arquitectura no estaba diseñada para soportar nuevos tipos de clientes web, requiriendo cambios significativos en la arquitectura y lógica de negocio.
- **Arquitectura mainframe:** Esta arquitectura más avanzada que la cliente-servidor de 2 capas se basa en la interacción de los usuarios con el sistema a través de dispositivos simples sin capacidad de procesamiento propio que capturaban la entrada del usuario mostrando su salida al mainframe. Este mainframe se encarga de la ejecución de todas las aplicaciones, manejando la lógica de negocio y la base de datos. Debido a esto, un mainframe es un punto

extremadamente crítico en la arquitectura ya que, si falla, todo el sistema cae derrumbando el procesamiento distribuido.

La arquitectura de 3 capas es una evolución de las arquitecturas previamente descritas. Se compone de las siguientes 3 capas:

- **Capa de la lógica de negocio**: Esta capa contiene la lógica de negocio de la aplicación, procesando las solicitudes de los clientes, se utilizan tecnologías como Java bajo frameworks como Spring Boot.
- **Capa de servicios**: Esta capa actúa como intermediario entre la capa de la lógica de negocio y los clientes, proporcionando interfaces para interactuar con la lógica de negocio. Esta capa expone la API REST a través de protocolos universales y estandarizados, así como políticas de seguridad.
- **Capa de persistencia:** Esta capa simplifica el acceso a la base de datos, eliminando las características particulares de cada Sistema de Gestión de Base de Datos (SGBD) así como el uso de lenguajes de consulta específicos.

Además de las capas por el lado del servidor encontramos 2 capas ajenas a él:

- **Capa de presentación:** Esta es la capa que interactúa directamente con el usuario, se utilizan tecnologías como HTML, CSS, JavaScript además de frameworks como React.
- **Capa de Datos:** Esta capa maneja el almacenamiento y recuperación de datos a través de bases de datos relacionales y no relacionales.

3.2.2. Spring Boot

Para la creación de aplicaciones empresariales, es vital elegir un framework que se ajuste a las características del proyecto. Un framework proporciona la estructura necesaria para la implementación de la arquitectura de 4 capas.

Spring Framework es el principal framework de Java, aparecido en el 2004 como una versión mejorada de J2EE para el desarrollo de aplicaciones empresariales distribuidas. Su popularidad se debe a ser extremadamente potente, estar en permanente mejora y contar con un amplio soporte lo que ha originado una enorme comunidad de desarrolladores.

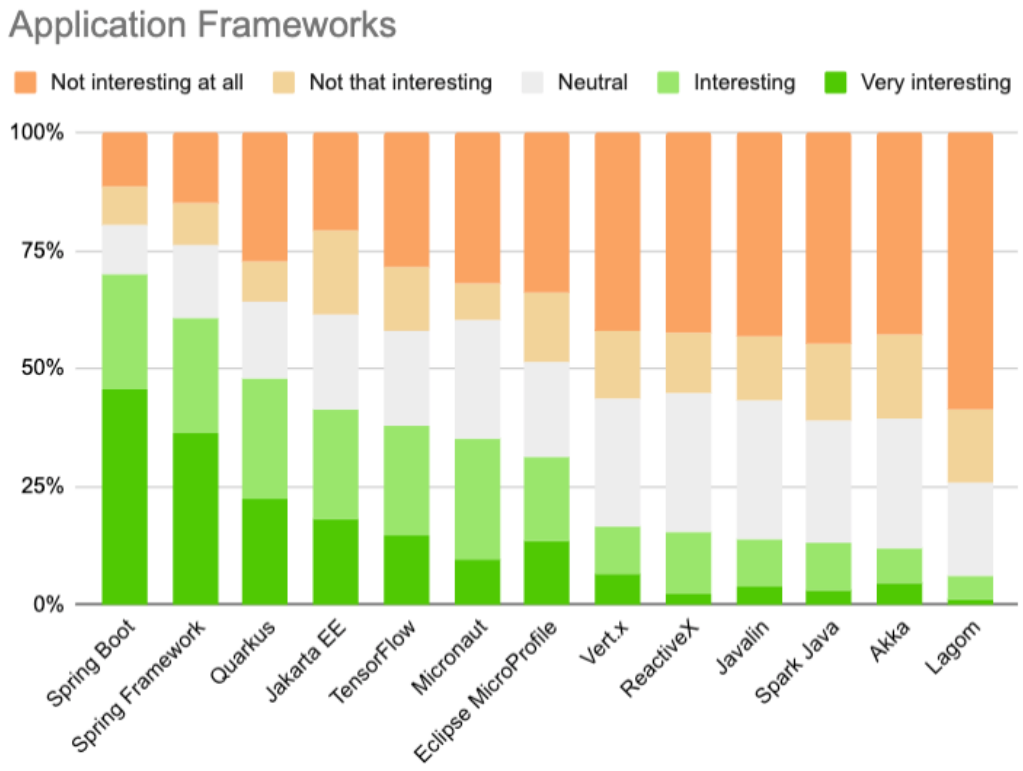


Ilustración 13. Frameworks más populares de Java

Aunque Spring es un framework extremadamente potente y cuenta con muchas ventajas, la gran cantidad de módulos y dependencias que contiene puede dificultar su uso en algunos proyectos. Bajo estos inconvenientes es donde destaca Spring. Boot, un marco de trabajo basado en Spring que facilita y simplifica la creación de aplicaciones Java. Spring Boot permite modular la integración de cada una de las funcionalidades de Spring, de forma que el proyecto pueda ir escalando en complejidad en función de sus necesidades facilitando la curva de aprendizaje.

3.2.3. Java

Para el desarrollo de aplicaciones empresariales existen una gran variedad de lenguajes de programación, así como frameworks asociados a ellos, sin embargo, en este proyecto ha elegido Java por las ventajas que ofrece frente a sus competidores.

Categoría	Java	C#	Python
Framework asociado	Spring	.NET	Django
Madurez del framework	Alta madurez, con una larga historia de mejoras y soporte	Moderada madurez, pero con cambios significativos recientemente	Baja madurez y menor robustez para aplicaciones empresariales complejas
Ecosistema y comunidad	Enorme comunidad con un ecosistema rico con numerosos proyectos y extensiones	Gran comunidad, pero inferior a la de Spring	La comunidad de Python es grande, pero la gran cantidad de frameworks y bibliotecas hace más difícil la búsqueda específica de soluciones
Rendimiento	Alto rendimiento	Alto rendimiento	Menor rendimiento por su naturaleza interpretada
Escalabilidad	Spring Boot facilita la creación de aplicaciones escalables con arquitectura de microservicios o modular	.NET es escalable, pero es más complejo de configurar fuera del ecosistema de Microsoft	Se puede escalar, pero requiere mayor esfuerzo para alcanzar el nivel de Spring Boot o .NET

Compatibilidad	Alta compatibilidad con múltiples sistemas	Alta compatibilidad dentro del ecosistema de Microsoft, aunque con limitaciones fuera de él	Alta compatibilidad, pero se requieren de más configuraciones manuales

Tabla 1. Comparativa lenguajes programación backend

En resumen, para el desarrollo de una aplicación empresarial compleja y escalable, Java con Spring Boot es la mejor opción debido principalmente a su madurez, robustez y amplia comunidad.

3.2.4. MySQL

MySQL es un sistema de gestión de bases de datos relacional que utiliza SQL como su lenguaje principal para gestionar y manipular datos. Desarrollado en un principio por MySQL AB y actualmente mantenido por Oracle Corporation, MySQL es una de las bases de datos más populares y ampliamente utilizadas en el mundo de aplicaciones web.

MySQL permite organizar los datos estructurados en entidades a través de tablas, que pueden ser relacionadas entre sí mediante claves primarias y foráneas, facilitando la normalización de estas.

Este sistema utiliza SQL como lenguaje para realizar consultas, lo cual es ampliamente conocido por administradores de bases de datos.

Además, MySQL soporta transacciones cumpliendo con las propiedades ACID[2], garantizando fiabilidad y consistencia de datos. Además, es capaz de manejar grandes volúmenes de datos otorgando un gran rendimiento, sobre todo en aplicaciones de lectura intensiva.

Si bien es cierto que existen recientes tecnologías que compiten con SQL como, por ejemplo, NoSQL, su uso se ha descartado para este proyecto por varios motivos.

[2] (Atomicidad, Consistencia, Aislamiento, Durabilidad)

- En aplicaciones donde las entidades tienen relaciones bien definidas, MySQL ofrece un modelo relacional que facilita la representación de estas relaciones, sin embargo, **NoSQL no es eficiente** para estos escenarios ya que está optimizado **para gestionar modelos no relacionales** basados en clave-valor.
- Generalmente **no cumple con las propiedades ACID** en su totalidad.

3.2.5. React

React es una librería de JavaScript de código abierto, desarrollada por Facebook que se utiliza para construir interfaces de usuario en entornos de aplicaciones web. Se lanzó originalmente en el año 2013 y actualmente es ampliamente utilizada gracias a su eficiencia y flexibilidad.

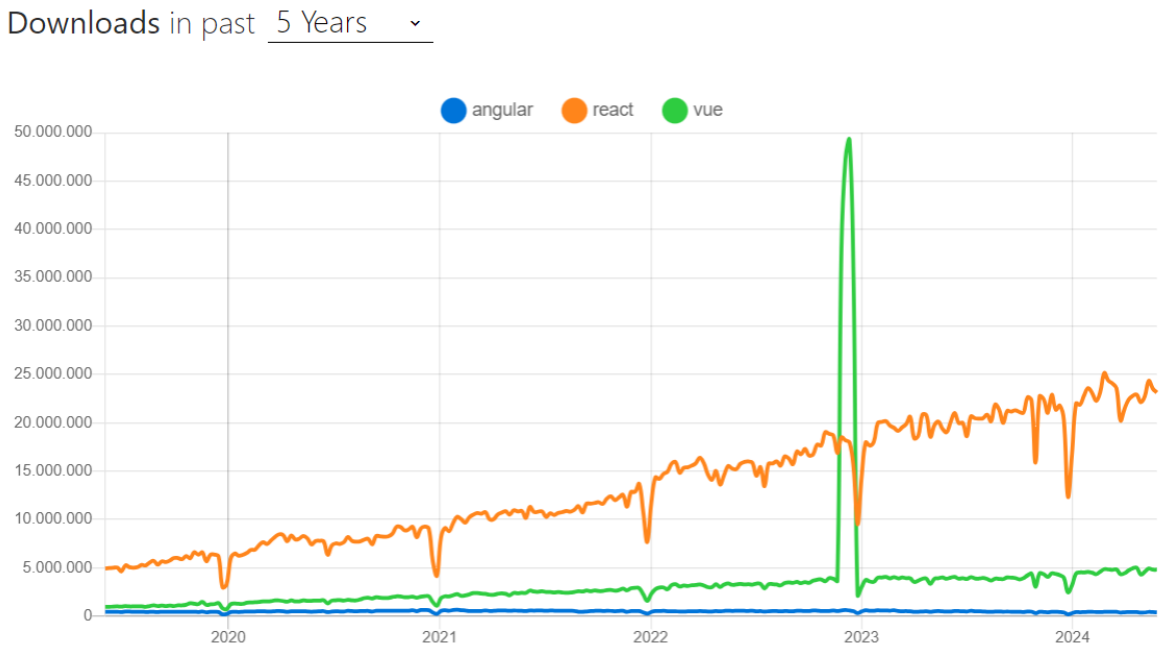


Ilustración 14. Comparativa de uso de React, Angular y Vue

Tal y como puede comprobarse en la **Ilustración 14**, usando npm, uno de los principales gestores de dependencias de bibliotecas de JavaScript, React tiene una gran ventaja que va en aumento respecto a sus competidores.

Esta librería se basa en el uso de componentes reutilizables que encapsulan la lógica y la presentación de la interfaz de usuario. Cada componente se compone a su vez de otros componentes hijos que tienen argumentos que los distinguen unos de otros. Un componente es una pequeña zona encapsulada que realiza una actividad específica y puede tener estado o no.

Sin embargo, la principal característica que proporciona React es su extrema eficiencia. Cuando existe algún cambio que impacta en el DOM, React realiza el cambio únicamente sobre ese elemento. Esto se consigue gracias a la naturaleza declarativa y a su Virtual DOM, cuando se realiza un cambio, React identifica las diferencias entre el Virtual DOM anterior y el nuevo, volviendo a renderizar únicamente las partes de la interfaz que presentan cambios.

Otro punto fuente de React es su comportamiento predecible en una única dirección, esto significa que los datos siempre fluyen desde el componente padre a sus componentes hijos. Este enfoque se traduce en una mayor previsibilidad y facilidad para rastrear el estado y los datos de la aplicación.

Los datos que se gestionan dentro de un componente se conocen como estados, que en resumen son un conjunto de datos que solo pueden ser modificados por ese mismo componente. Estos datos pueden ser pasados a sus componentes hijos mediante el uso de Props. Al pasar los datos en un único flujo se mantiene una jerarquía y transparencia haciendo que cualquier cambio en la interfaz de usuario sea rastreable hasta su origen.

React proporciona otra característica muy importante, desde su versión 16.8 esta librería introdujo un antes y después en la manera en la que se usa el estado. Los hooks permiten a los desarrolladores usar el estado y otras características de React a través de componentes funcionales, sin necesidad de escribir clases. Los hooks en resumen son funciones que representan una manera sencilla y extremadamente poderosa de manejar el estado, efectos secundarios y otros aspectos del ciclo de vida de los componentes de React. Algunos de los hooks más comúnmente utilizados en React son useState, useEffect, useContext.

En la siguiente tabla se muestran las características clave de React frente a sus competidores que han propiciado su elección.

Categoría	React	Angular	Vue
Curva de aprendizaje	Moderada, solo requiere conocimientos de JavaScript	Alta. Requiere aprender su extensa API	Baja. Sencillo, especialmente para principiantes
Rendimiento	Alto, utiliza un DOM virtual	Medio/Alto, Change Detection eficiente pero más pesado	Alto, usa un DOM virtual similar a React
Ecosistema y comunidad	Enorme, respaldada por Facebook	Grande, respaldada por Google.	Media
Flexibilidad	Alta. Biblioteca ligera que permite integración con otras tecnologías	Baja. Framework completo con arquitectura rígida	Alta. Framework progresivo que puede integrarse en proyectos existentes
Uso en grandes empresas	Amplio. Utilizado por Facebook, Instagram, Airbnb, Netflix	Amplio. Utilizado por Google, Microsoft, Upwork	Bajo

Tabla 2. Comparativa de herramientas frontend

Tal y como puede apreciarse en la tabla, Vue es sin duda la herramienta menos interesante ya que principalmente tiene poco uso en grandes aplicaciones. Respecto a React y Angular, se ha optado elegir React por varios motivos:

- Las **dimensiones del frontend no son demasiado grandes**, por lo que utilizar un framework como Angular puede ser algo excesivo. Una biblioteca como React es capaz de suplir todas las necesidades actuales y si en un futuro se necesitasen herramientas más avanzadas, podría integrarse el framework Next, especialmente pensado para React.
- Hasta hace pocos años, **Angular no incorporó herramientas que demandaban desde hace mucho tiempo los desarrolladores**, por este motivo muchos de ellos se pasaron a React. Aunque en las últimas versiones de Angular se han ido incorporando mejoras como una interfaz declarativa, componentes más modulares y reutilizables o conceptos parecidos a los hooks de React, gran parte de la comunidad ha perdido la credibilidad en el soporte de Angular.

3.2.6. TypeScript

TypeScript es un superset tipado de JavaScript que compila a JavaScript puro. Esto significa que todo código JavaScript válido es también código TypeScript válido, pero TypeScript añade una capa adicional de seguridad y productividad a través de su sistema de tipos estáticos opcionales. Fue desarrollado y es mantenido por Microsoft desde octubre del 2012.

Una de los motivos para usar TypeScript en este proyecto ha sido mi experiencia previa con lenguajes de programación tipados, como C, C++, C# y Java, adquirida durante el grado. Estos lenguajes, al igual que TypeScript, hacen uso del tipado estático, lo que proporciona un entorno de desarrollo más seguro y robusto.

TypeScript es ampliamente utilizado por numerosos frameworks o librerías de JavaScript, por ejemplo, React.

3.2.7. TailwindCSS

TailwindCSS es un framework CSS de utilidad que permite a los desarrolladores construir rápidamente interfaces de usuario modernas y responsivas a través de clases predefinidas. A diferencia de otros frameworks CSS, TailwindCSS se centra en ofrecer una serie de clases de utilidad que pueden crear cualquier tipo de diseño.

La principal ventaja que nos ofrece TailwindCSS es disponer de una amplia gama de clases de utilidad para estilos comunes como márgenes, rellenos, colores, fuentes y mucho más.

Además, TailwindCSS facilita el diseño responsivo con clases de utilidad que pueden aplicarse en distintos puntos de interrupción. Esto permite construir rápidamente interfaces que se adapten a diferentes tamaños de pantalla. En este proyecto esta utilidad se ha implementado para crear una interfaz para ordenadores y otra para dispositivos móviles.

3.2.8. Docker

Docker es una plataforma de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores software. Los contenedores son entornos ligeros, portátiles y autosuficientes que permiten empaquetar una aplicación y sus dependencias de manera uniforme y consistente. Docker fue lanzado en marzo de 2013 por Docker, INC. Y desde entonces ha revolucionado la manera en la cual las aplicaciones se desarrollan.

Un contenedor es una instancia en ejecución de una imagen de Docker, a diferencia de las máquinas virtuales tradicionales, los contenedores comparten el sistema operativo de la máquina host, lo que los hace más ligeros y eficientes en términos de uso de recursos. Cada contenedor tiene su propio sistema de archivos, memoria, CPU y espacio de red, lo que garantiza el aislamiento de la aplicación.

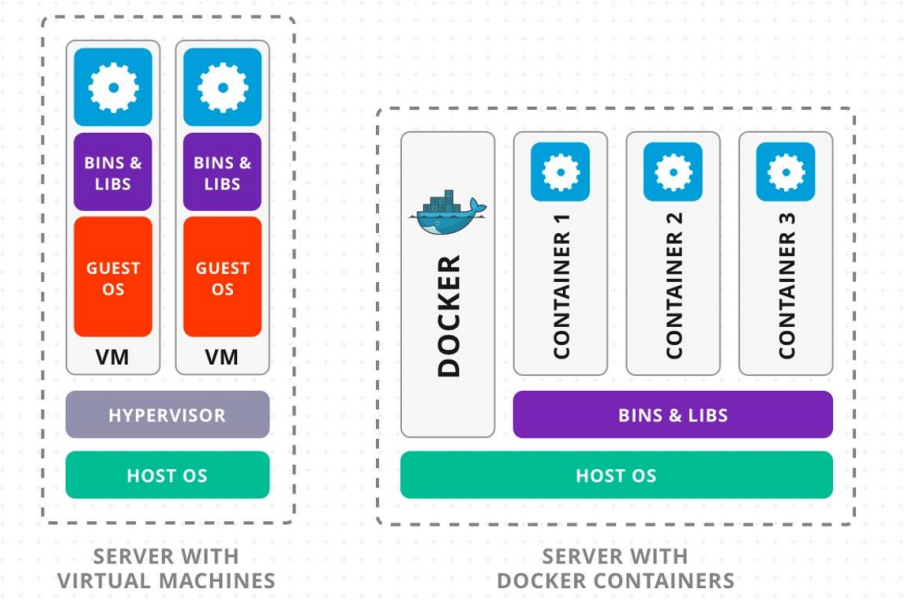


Ilustración 15. Comparativa de servidor con máquinas virtuales vs servidor con contenedores de imágenes

Una imagen de Docker es una plantilla inmutable que define todo lo necesario para ejecutar una aplicación: código fuente, bibliotecas, dependencias, herramientas y configuraciones. Las imágenes se crean a partir de archivos Dockerfile, que almacenan una serie de instrucciones para ensamblar la imagen paso a paso.

En este proyecto, se ha utilizado un archivo Dockerfile que contiene algunas acciones como, por ejemplo:

- Se ha utilizado *nvidia/cuda:11.8.0-cudnn8-devel-ubuntu22.04* como imagen base, que proporciona un entorno de desarrollo CUDA con cuDNN sobre Ubuntu 22.04.
- A través de comandos apt-get, se han instalado Python 3.11, pip y otras utilidades.
- Se ha configurado la zona horaria a Europa/Madrid para asegurar la consistencia en los registros de tiempo.
- Se ha creado y activado un entorno virtual de Python dentro del contenedor para instalar todas las dependencias definidas en requirements.txt.

Por otro lado, Docker Compose se ha utilizado para definir y ejecutar múltiples contenedores Docker. En este proyecto se han definido los siguientes servicios en el docker-compose.yml:

- **db:** Un servicio de la base de datos MySQL.
- **flask-app:** La aplicación backend de Flask.
- **spring-boot-app:** La aplicación backend de Spring Boot.

Se ha configurado Docker Compose para gestionar la red y los volúmenes compartidos entre los contenedores, y para asegurar que los contenedores dependientes se inicien en el orden correcto.

Las imágenes Docker se han subido a Docker Hub, lo que permite a otros usuarios descargarlas y ejecutar el proyecto sin necesidad de construir las imágenes localmente.

3.2.9. Microsoft Azure

Microsoft Azure es una plataforma de servicios en la nube ofrecida por Microsoft, que proporciona una variedad de servicios como computación, almacenamiento, bases de datos, redes y más.

En este proyecto, Microsoft Azure se ha utilizado para el almacenamiento y gestión del corpus de entrenamiento y de las métricas de evaluación. A continuación, se detalla el servicio Azure Blob Storage utilizado.

Azure Blob Storage es un servicio de almacenamiento de objetos en la nube que permite almacenar grandes cantidades de datos no estructurados. En este proyecto, Azure Blob Storage se ha utilizado para albergar el corpus y las métricas. El almacenamiento de los datos se realiza mediante la API de Azure Storage Blob, siendo los datos cargados y descargados desde Azure Blob Storage a través de esta API.

4. Análisis y diseño del sistema

4.1. Requisitos del sistema

Una vez aclarecidos los objetivos por satisfacer, en este apartado se detallan los requisitos funcionales y no funcionales del sistema. Los requisitos establecen las bases para el diseño y el desarrollo del sistema.

La definición clara y precisa de los requisitos es un paso crucial en el desarrollo de cualquier sistema de software. Los requisitos funcionales especifican las acciones que el sistema debe ser capaz de realizar, describiendo las funcionalidades y comportamientos esperados. Estos requisitos son esenciales para asegurar que el sistema cumpla con las necesidades del usuario final y soporte todas las funcionalidades previstas.

Por otro lado, los requisitos no funcionales establecen los criterios que determinan la calidad y eficiencia del sistema. Estos requisitos incluyen aspectos como el rendimiento, la usabilidad, la seguridad y la escalabilidad del sistema, asegurando que el software no solo funciona correctamente, sino que también sea seguro, eficiente y fácil de usar.

A continuación, se presentan los requisitos funcionales y no funcionales del sistema:

4.1.1. Requisitos funcionales

ID	Requisito	Descripción
RF01	Selección de modelo	El usuario debe ser capaz de elegir entre varios modelos de clasificación de ofensividad
RF02	Clasificación de lenguaje ofensivo	El sistema debe ser capaz de clasificar frases en varias categorías de ofensividad o no ofensividad
RF03	Actualización del corpus	El usuario debe ser capaz de actualizar el corpus de entrenamiento con nuevas entradas etiquetadas
RF04	Testeo de modelos	El usuario debe ser capaz de probar la precisión de un modelo

RF05	Generación de métricas	El usuario debe ser capaz de visualizar métricas de testeo para comprobar la calidad de los modelos entrenados
RF06	Registro y login de usuarios	El usuario debe poder registrarse e iniciar sesión utilizando sus datos personales
RF07	Descarga del corpus	El usuario debe poder descargar el corpus de entrenamiento actualizado en caso de existir nuevas entradas por cualquier otro usuario
RF08	Ver datos de interés	El usuario debe ser capaz de visualizar datos de interés como un top de los modelos con mayor valoración o los nombres de los usuarios que suben más modelos
RF09	Puntuar los modelos	El usuario podrá valorar en un ratio del 1/5 los resultados obtenidos para cada modelo durante el testeo de frases sueltas
RF10	Dar feedback sobre los modelos	El usuario podrá escribir una valoración para los modelos que considere oportunos
RF11	Alternar entre modos de visualización	El usuario podrá cambiar entre 2 modos (claro y oscuro) según sus preferencias
RF12	Enviar solicitudes de acceso	El usuario debe ser capaz de enviar una solicitud de acceso en aquellos modelos que no tienen un acceso público
RF13	Gestionar las solicitudes	El usuario creador de un modelo específico podrá aceptar o rechazar las peticiones de acceso para cada usuario solicitante
RF14	Filtrar modelos	El usuario podrá filtrar el listado de modelos por su puntuación, número de parámetros en orden ascendiente o descendiente o nombre del modelo

Tabla 3. Requisitos funcionales

4.1.2. Requisitos no funcionales

ID	Requisito	Descripción
RNF01	Tiempos de respuesta	El sistema debe responder a las solicitudes de clasificación en menos de 5 segundos
RNF02	Eficiencia del uso de memoria	El sistema debe evitar saturar el máximo de memoria VRAM provocado por un acceso simultáneo de varios usuarios
RNF03	Disponibilidad	El sistema debe estar disponible al menos el 99,5% del tiempo
RNF04	Seguridad	Los datos de los usuarios deben estar protegidos contra accesos no autorizados mediante mecanismos de autenticación y cifrado
RNF05	Escalabilidad	El sistema debe ser capaz de escalar para manejar un aumento en el número de usuarios y en el volumen de datos
RNF06	Usabilidad	La interfaz del sistema debe ser intuitiva y fácil de usar para usuarios con conocimientos básicos de informática. Además, la interfaz deberá contar con un máximo de 3 colores simultáneos para un uso ameno y estos deberán ser lo más neutros posibles para personas que padezcan afecciones visuales como, por ejemplo, daltonismo
RNF07	Diseño responsivo	La interfaz debe ser responsiva, adaptándose a una amplia gama de dispositivos, tanto ordenadores como dispositivos móviles
RNF08	Consistencia	Tanto el backend como el frontend deberán estar lo suficientemente depurados para evitar frustrar al usuario con errores ajenos a él

Tabla 4. Requisitos no funcionales

4.2. Casos de uso

Partiendo de los requisitos obtenidos previamente, en este apartado se detallan los diferentes casos de uso que conforman la aplicación.

Estos casos de uso son una herramienta esencial para comprender cómo los usuarios interactúan con el sistema. Describen de manera detallada los escenarios en los que los usuarios utilizarán el sistema, proporcionando una visión clara y estructurada de las interacciones entre el sistema y sus usuarios.

Los casos de uso ayudan a identificar los flujos de trabajo, los actores involucrados, las precondiciones y postcondiciones de cada funcionalidad, asegurando así que todos los aspectos relevantes sean considerados durante el desarrollo

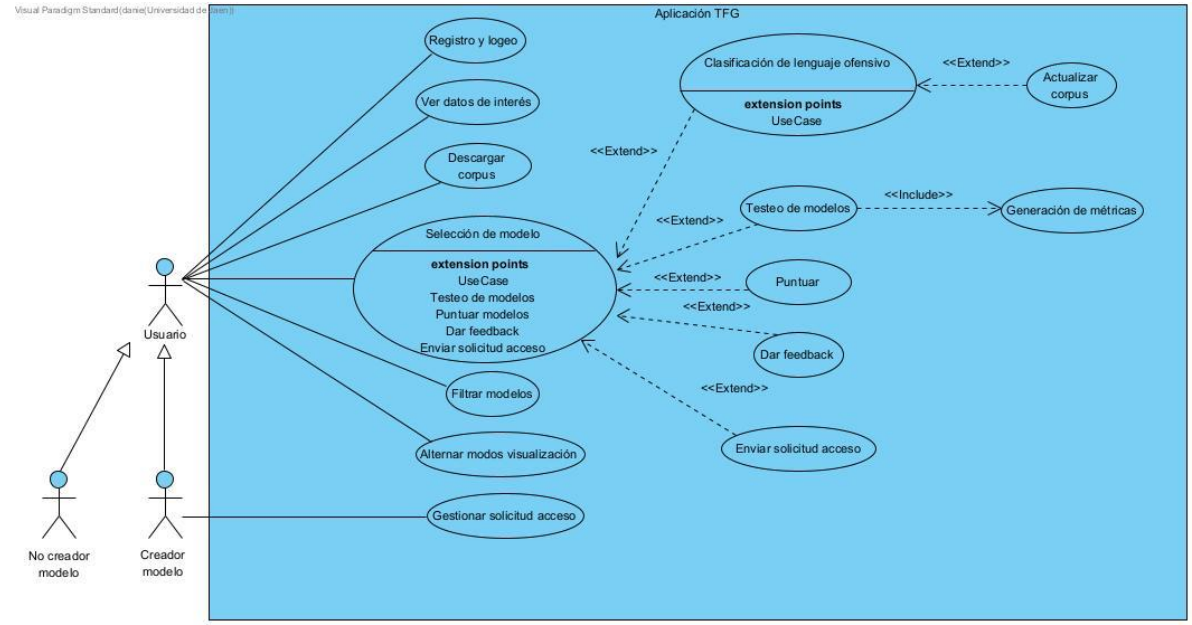


Ilustración 16. Diagrama de casos de uso

En el anterior diagrama se ven representados todos los casos de uso de la aplicación, asociados a sus respectivos actores y con los mecanismos que denotan las relaciones entre ellos, las etiquetas <<*include*>> y <<*extend*>>.

La relación <<*include*>> se utiliza para mostrar que un caso de uso incluye el comportamiento de otro caso de uso, <<*extend*>> por el contrario se utiliza para mostrar que un caso de uso puede extender el comportamiento de otro caso de uso bajo ciertas condiciones.

4.2.1. Desarrollo de los casos de uso

En este apartado se van a desarrollar cada uno de los casos de uso mencionados en la **Ilustración 16** de forma que se visualice con mayor detalle los actores implicados, así como las precondiciones, postcondiciones y el flujo principal de cada caso de uso.

Caso de uso	Descripción
ID	CU01
Título	Logueo de usuarios
Actor principal	Usuario
Sistema	Aplicación TFG
Participantes	Usuario
Nivel	Objetivo de usuario
Precondición	
Operaciones básicas	
1	Acceder a la página de login
2	Completar el formulario de credenciales de login
3	Pulsar el botón para finalizar el logueo
Alternativas	
2.A	Si las credenciales no cumplen un formato adecuado en determinados campos, se muestra un mensaje de error
3.A	Si las credenciales son incorrectas o algunas de estas faltan, el sistema muestra un mensaje de error y solicita reingresarlas

Tabla 5. CU-01

Caso de uso	Descripción
ID	CU02
Título	Registro de usuarios
Actor principal	Usuario
Sistema	Aplicación TFG
Participantes	Usuario
Nivel	Objetivo de usuario
Precondición	
Operaciones básicas	
1	Acceder a la página de registro
2	Completar el formulario de credenciales de registro
3	Pulsar el botón para finalizar el registro
Alternativas	
2.A	Si las credenciales no cumplen un formato adecuado en determinados campos, se muestra un mensaje de error
3.A	Si las credenciales se corresponden con un usuario ya existente o algunas de estas faltan, el sistema muestra un mensaje de error y solicita reingresarlas

Tabla 6. CU-02

Caso de uso	Descripción
ID	CU03
Título	Clasificación de texto
Actor principal	Usuario
Sistema	Aplicación TFG
Participantes	Usuario
Nivel	Objetivo de usuario
Precondición	
Operaciones básicas	
1	El usuario pulsa sobre el menú home para visualizar la página principal
2	El sistema muestra varias tablas y gráficos

Tabla 7. CU-03

Caso de uso	Descripción
ID	CU04
Título	Descargar corpus
Actor principal	Usuario
Sistema	Aplicación TFG
Participantes	Usuario
Nivel	Objetivo de usuario
Precondición	
Operaciones básicas	
1	El usuario debe pulsar sobre el icono de descarga ubicado en el navbar
2	El sistema establece conexión con los servicios en la nube para descargar la última versión del corpus

Tabla 8. CU-04

Caso de uso	Descripción
ID	CU05
Título	Seleccionar modelo
Actor principal	Usuario
Sistema	Aplicación TFG
Participantes	Usuario
Nivel	Objetivo de usuario
Precondición	El usuario debe encontrarse en una página que permita cambiar de modelo
Operaciones básicas	
1	El usuario pulsa sobre el dropdown que contiene todos los modos a elegir
2	El usuario pincha sobre el modelo elegido
3	El sistema carga en la VRAM el modelo seleccionado por el usuario, liberando los modelos anteriores de esta en caso de desbordamiento de memoria

Tabla 9. CU-05

Caso de uso	Descripción
ID	CU06
Título	Clasificación de lenguaje ofensivo
Actor principal	Usuario
Sistema	Aplicación TFG
Participantes	Usuario
Nivel	Objetivo de usuario
Precondición	Un tipo de modelo debe haber sido seleccionado
Operaciones básicas	
1	El usuario introduce una frase en el input
2	El usuario pulsa sobre el botón “Clasificar”
3	El sistema devuelve la clasificación de esa frase sobre un modal y pregunta al usuario si es correcta
4	El usuario cierra el modal pulsando sobre la “x”
Alternativas	
3.A	Si la clasificación no es correcta, el sistema pregunta al usuario cual sería la forma correcta de clasificar la frase y se actualiza el corpus de entrenamiento. Salta al punto 4

4.A	El usuario cierra el modal pulsando “sí” sobre la pregunta “¿Ha clasificado bien la frase el modelo?” que el sistema le hace

Tabla 10. CU-06

Caso de uso	Descripción
ID	CU07
Título	Actualizar corpus
Actor principal	Sistema
Sistema	Aplicación TFG
Participantes	Usuario, Sistema
Nivel	Objetivo de sistema
Precondición	El usuario debe haber clasificado una frase determinada
Operaciones básicas	
1	El sistema añade una nueva entrada al corpus de entrenamiento incorporando tanto la frase clasificada como la etiqueta correcta que asoció el usuario

Tabla 11. CU-07

Caso de uso	Descripción
ID	CU08
Título	Testeo de modelos
Actor principal	Usuario
Sistema	Aplicación TFG
Participantes	Usuario
Nivel	Objetivo de usuario
Precondición	Un tipo de modelo debe haber sido seleccionado
Operaciones básicas	
1	El usuario pincha sobre la página “Tests” ubicada en el sidebar
2	El usuario pulsa el botón de “Comenzar Test”

Tabla 12. CU-08

Caso de uso	Descripción
ID	CU09
Título	Generar métricas
Actor principal	Sistema
Sistema	Aplicación TFG
Participantes	Usuario, Sistema
Nivel	Objetivo de sistema
Precondición	El test de un modelo debe de haber finalizado
Operaciones básicas	
1	El sistema genera 4 métricas, una muestra los datos de precisión micro, macro, etc… para cada clase, otra métrica detalla la matriz de confusión para cada clase, otra muestra un gráfico AUC-ROC para clasificación multiclase y finalmente se genera un gráfico AUC-ROC para clasificación binaria (Ofensivo/no ofensivo)

Tabla 13. CU-09

Caso de uso	Descripción
ID	CU10
Título	Puntuar
Actor principal	Usuario
Sistema	Aplicación TFG
Participantes	Usuario
Nivel	Objetivo de usuario
Precondición	Un tipo de modelo debe haber sido seleccionado
Operaciones básicas	
1	El usuario pulsa sobre las estrellas que se muestran en el menú valoración de cada modelo
2	Comprobar que la valoración añadida por el usuario ha influido en la valoración promedio del modelo, ubicada en la misma pantalla

Tabla 14. CU-10

Caso de uso	Descripción
ID	CU11
Título	Dar feedback
Actor principal	Usuario
Sistema	Aplicación TFG
Participantes	Usuario
Nivel	Objetivo de usuario
Precondición	Un tipo de modelo debe haber sido seleccionado
Operaciones básicas	
1	El usuario pincha sobre el editor WYSIWYG ubicado justo debajo de cada modelo
2	El usuario inserta un texto
3	El usuario pulsa el botón “Enviar”
Alternativas	
2.A	El usuario pone el texto en algún formato
2.B	El usuario añade una tabla

Tabla 15. CU-11

Caso de uso	Descripción
ID	CU12
Título	Enviar solicitud de acceso
Actor principal	Usuario
Sistema	Aplicación TFG
Participantes	Usuario, Sistema
Nivel	Objetivo de usuario
Precondición	El usuario debe encontrarse en una página que permita cambiar de modelo
Operaciones básicas	
1	El usuario pulsa sobre el botón de “Solicitar Acceso” del modelo privado que le interesa
2	El sistema envía una solicitud al usuario creador

Tabla 16. CU-12

Caso de uso	Descripción
ID	CU13
Título	Gestionar solicitud de acceso
Actor principal	Usuario
Sistema	Aplicación TFG
Participantes	Usuario, Sistema
Nivel	Objetivo de usuario
Precondición	Deben existir peticiones de acceso a algún modelo que haya creado el usuario
Operaciones básicas	
1	El usuario pincha sobre el icono de la campana
2	El usuario acepta una petición para un modelo en concreto
3	El sistema elimina la petición en cuestión
Alternativas	
2.A	El usuario rechaza la petición

Tabla 17. CU-13

Caso de uso	Descripción
ID	CU14
Título	Filtrar modelos
Actor principal	Usuario
Sistema	Aplicación TFG
Participantes	Usuario, Sistema
Nivel	Objetivo de usuario
Precondición	El usuario debe encontrarse en una página que permita cambiar de modelo
Operaciones básicas	
1	El usuario pincha el filtro de orden por puntuación
2	El usuario pincha orden ascendiente o descendiente
Alternativas	
1.A	El usuario pincha el filtro de orden por número de parámetros

Tabla 18. CU-14

4.3. Diseño del sistema

En este apartado se define la estructura y comportamiento de los componentes que lo conforman. En esta sección se detalla el diseño del sistema desarrollado para la aplicación enfocada en la clasificación de lenguaje ofensivo.

4.3.1. Diseño del backend

Tal y como se ha detallado en el apartado **3.2.1**, el backend de la aplicación utiliza una arquitectura de 3 capas, teniendo en cuenta la considerable magnitud de cada una de ellas, a continuación, se detallan los diagramas que detallan la estructura para confeccionar la capa de la lógica de negocio, servicios y de persistencia.

4.3.1.1. Diagrama de clases

En la siguiente ilustración se presenta el diseño de la capa de servicios y de la lógica de negocio. Este diagrama ha contado con distintas modificaciones a lo largo del proyecto por la naturaleza iterativa e incremental de la metodología SCRUM utilizada.

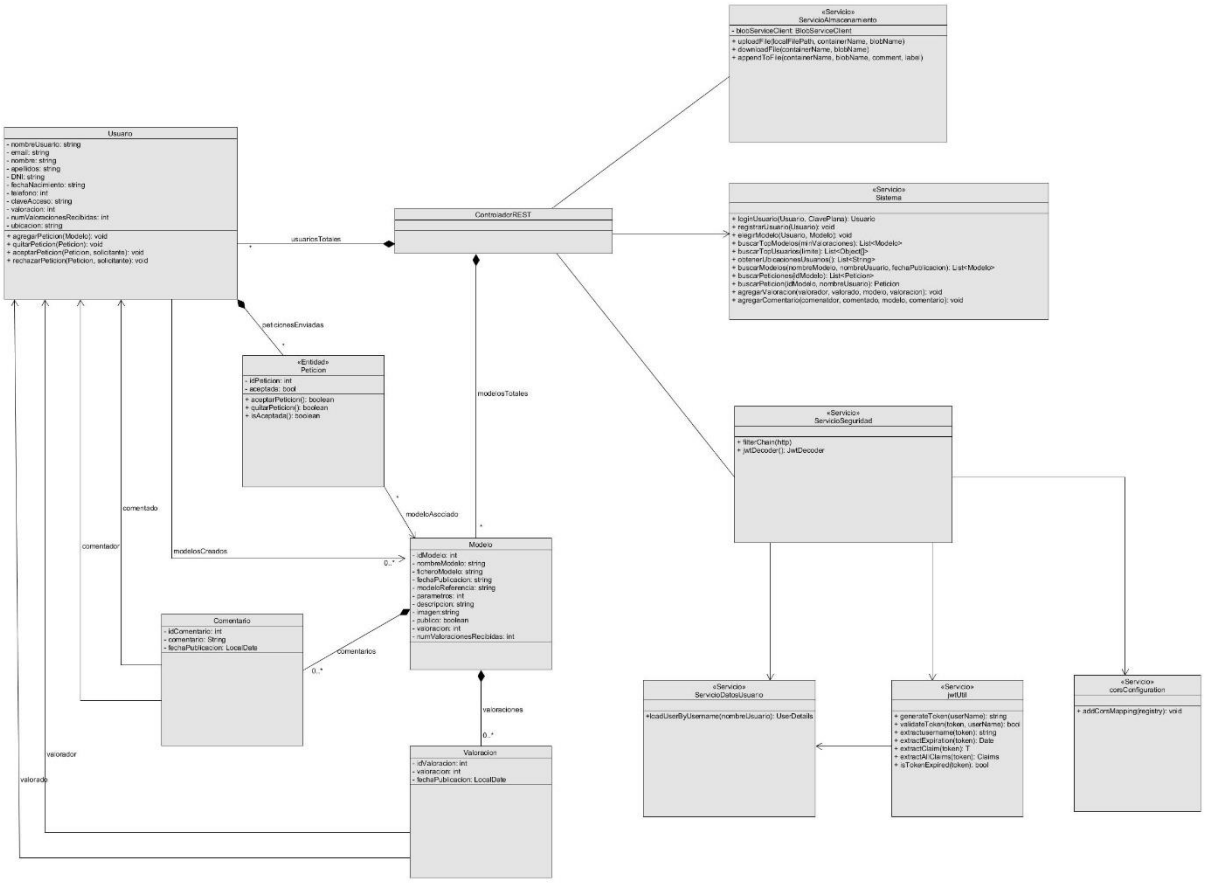


Ilustración 17. Diagrama de clases de la capa de servicios y lógica de negocio

Tal y como puede apreciarse, el proyecto incorpora una gran cantidad de Entidades y Servicios que permiten mantener una estructura organizada cumpliendo siempre un mínimo de estándares de seguridad.

4.3.1.2. Diagrama ORM

Una vez detalladas las clases que componen la lógica de negocio de la aplicación empresarial, podemos mostrar con detalles la capa de persistencia. Las bases de datos relacionales son ampliamente utilizadas para almacenar datos de forma segura, sin embargo, interactuar directamente con ellas mediante consultas SQL puede ser tedioso. Aquí es donde entra en juego el mapeo relacional de objetos (ORM) que se ejemplifica en el siguiente diagrama:

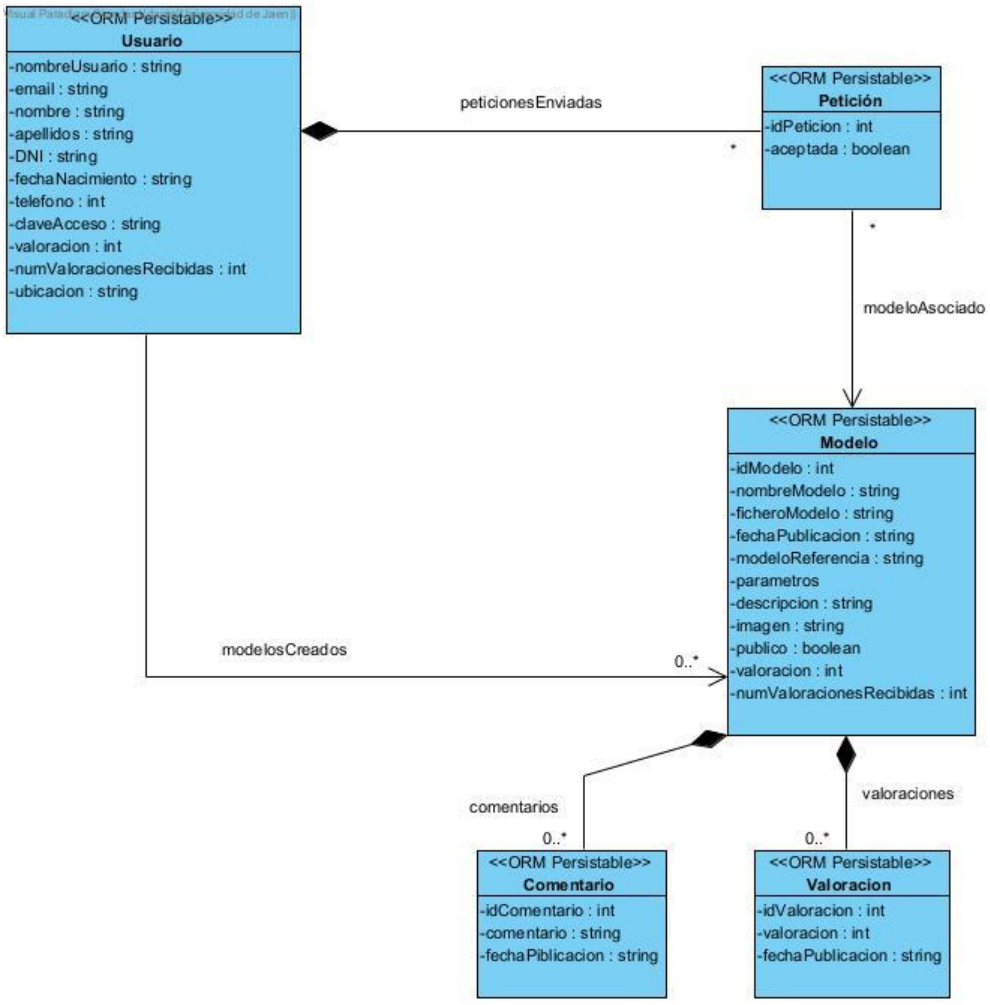


Ilustración 18. Diagrama ORM de la capa de persistencia

En el primer diagrama ORM, vemos una representación de clases como “Usuario”, “Peticion”, “Modelo”, “Comentario” y “Valoracion”. Cada clase contiene atributos que se mapean a las columnas de una tabla. Además, se definen las

relaciones de estas clases, indicando asociaciones como “peticionesEnviadas” y “modelosCreados”.

4.3.1.3. Diagrama ERD

El diagrama Entidad-Relación (ERD) proporciona una representación visual de las entidades en la base de datos y las relaciones entre ellas. Este tipo de diagrama es fundamental para el correcto diseño de la base de datos, ya que muestra cómo se organizan los datos y cómo interactúan las diferentes entidades.

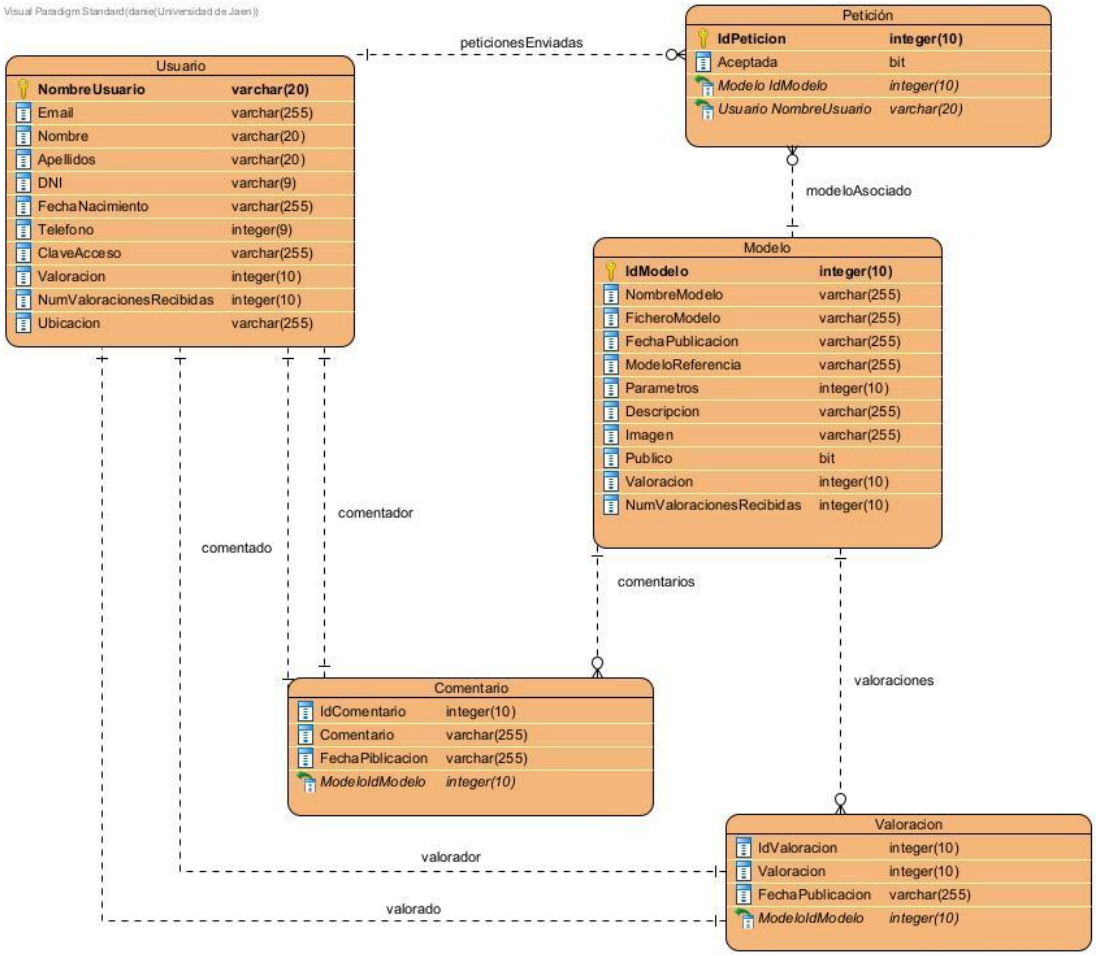


Ilustración 19. Diagrama ERD

En este diagrama, vemos la representación de las mismas entidades del ORM, pero ahora como tablas y relaciones en la base de datos. Cada entidad ORM se ha traducido a una tabla en el ERD, y las asociaciones se representan como relaciones entre tablas. La transformación del ORM al ERD se puede realizar sin demasiados problemas a través de herramientas como Visual Paradigm.

4.3.1.4. Diagrama final

Una vez detallado el desarrollo de cada una de las capas principales del lado del servidor, podemos detallar el diagrama de clases final de la aplicación. Sin embargo, antes es necesario indicar que la interacción entre la aplicación y la base de datos se ha seguido gracias al uso del patrón de diseño Data Access Object (DAO).

El patrón DAO es una estrategia de diseño que define para cada clase a persistir una clase especializada conocida como DAOs. Estos objetos implementan un esquema de operaciones CRUD en la base de datos, ofreciendo una abstracción que oculta los detalles del acceso a la base de datos. Esto permite que los desarrolladores trabajen con objetos de negocio sin preocuparse por la complejidad de las bases de datos.

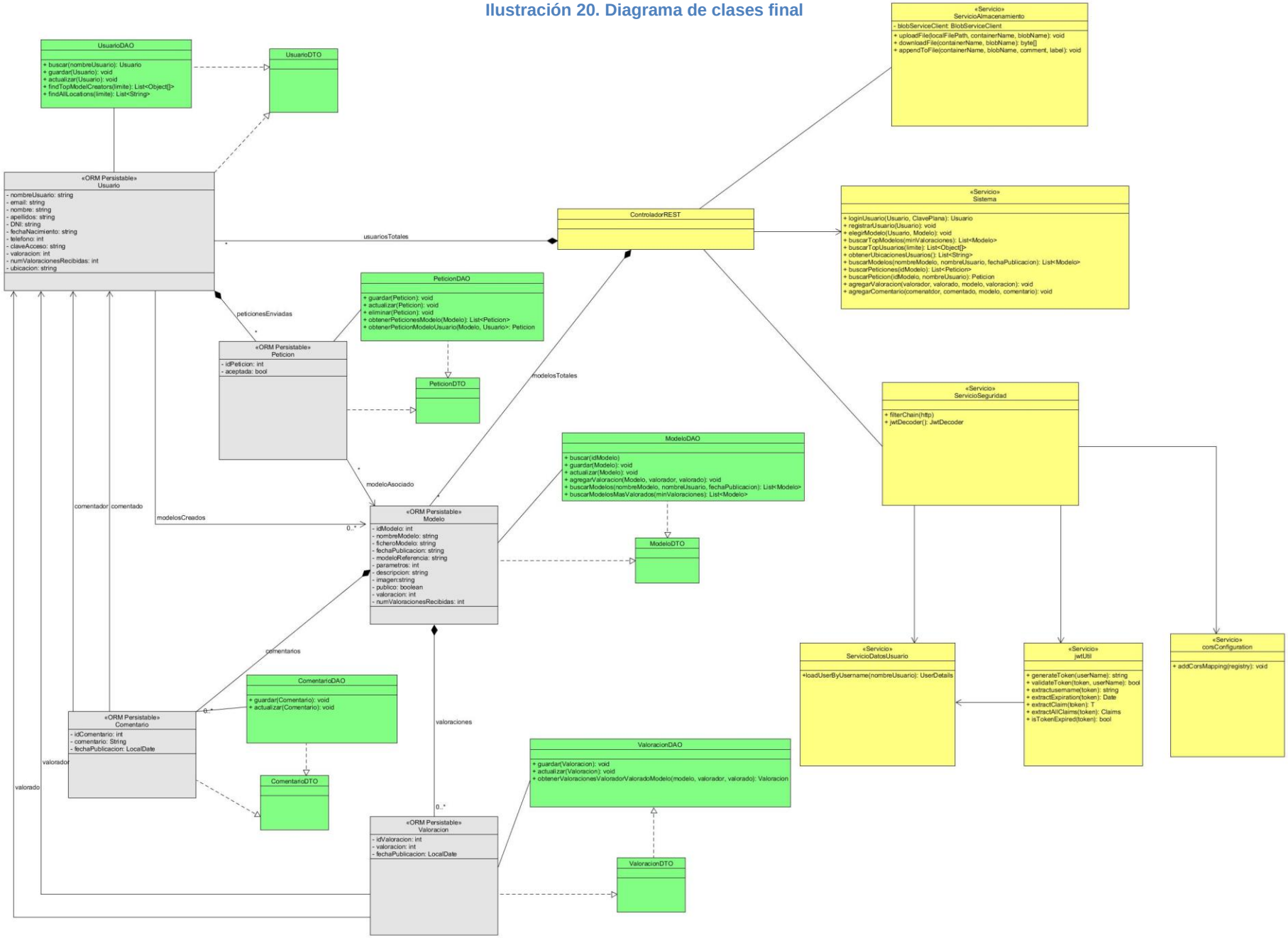
Los DAOs otorgan varios beneficios:

- **Separar la lógica de acceso a datos de la lógica de negocio**, obteniendo un código más limpio y modular.
- **Reutilizar código** ya que los DAOs pueden ser reutilizados en numerosas partes de la aplicación, evitando duplicación de código.
- **Facilidad en los tests unitarios y de integración** gracias a que los DAOs pueden ser fácilmente simulados para verificar el comportamiento de la lógica de negocio sin necesitar una base de datos real.
- **Flexibilidad en el acceso a los datos** ya que los DAOs permiten cambiar la tecnología de persistencia sin afectar a la lógica de negocio.

Estos DAOs, normalmente singletons también se conocen como repositorios según el DDD.

Una vez esquematizados los DAOs así como las clases previamente definidas podemos obtener el diagrama de clases final donde se asignará el color gris para las clases pertenecientes a la capa de dominio, amarillo para las clases pertenecientes a la capa de servicios y verde para las clases pertenecientes a la capa de persistencia.

Ilustración 20. Diagrama de clases final



4.3.1.5. Diseño de la API REST

Con el diagrama de clases establecido, podemos proceder a diseñar la API REST que permitirá la interacción de la aplicación cliente con la aplicación servidor.

Una interfaz de programación de aplicaciones (API) es un conjunto de definiciones y protocolos que permiten que diferentes aplicaciones se comuniquen entre sí. REST es un estilo de arquitectura para diseñar redes de aplicaciones en hipermedia. En resumen, una API RESTful permite que los sistemas se comuniquen usando el protocolo HTTP de manera predefinida.

Las API RESTful son ampliamente utilizadas debido a su simplicidad y escalabilidad. Utilizan métodos HTTP estándar como GET, POST, PUT y DELETE para realizar operaciones sobre los servicios que expone el sistema:

- **GET:** Recupera presentaciones del recurso sin modificarlo.
- **POST:** Crea un nuevo recurso.
- **PUT:** Actualiza un recurso.
- **DELETE:** Elimina un recurso.

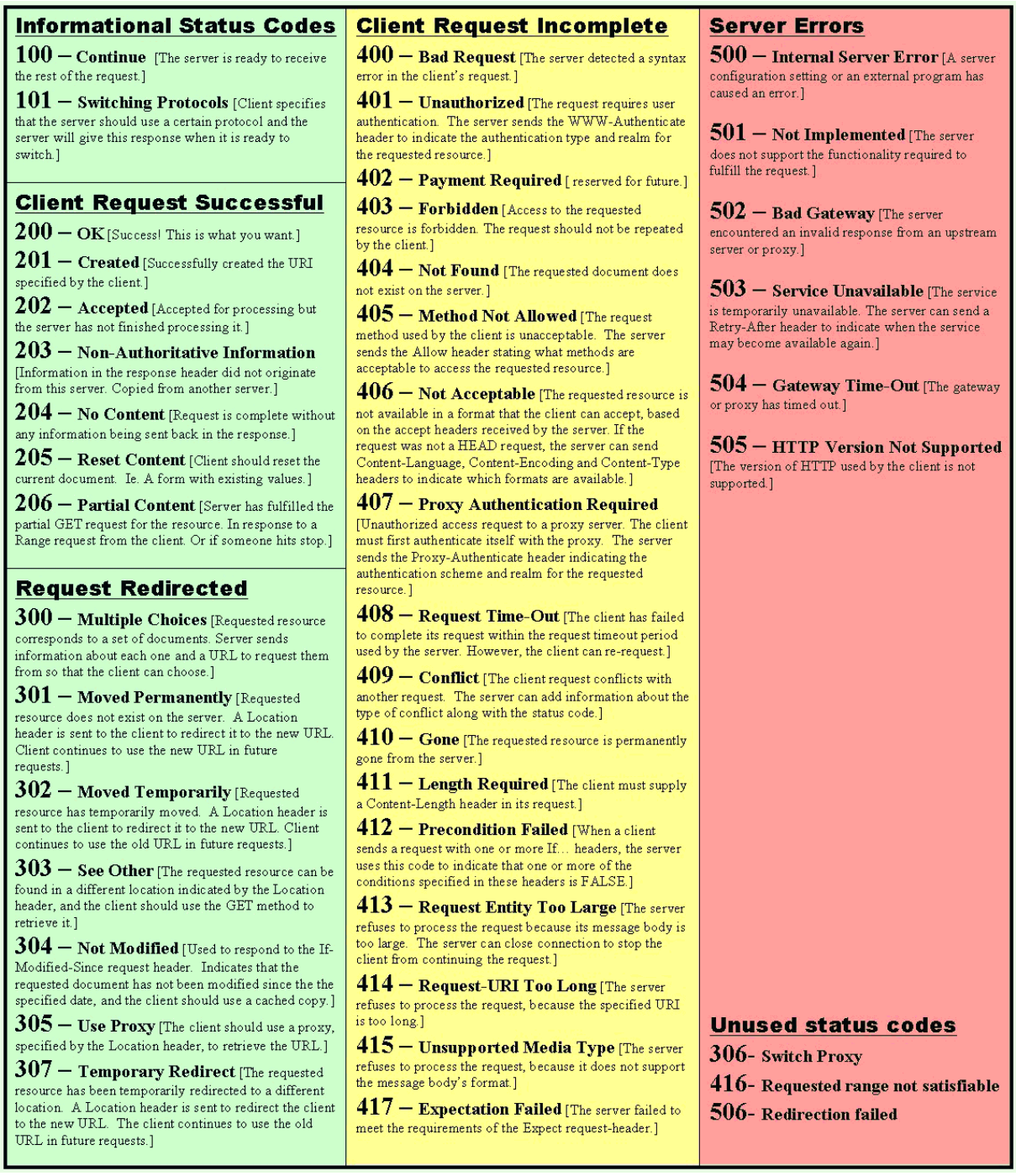
Las API RESTful deben de cumplir los principios REST. Algunos de los más importantes son:

- **En REST, todo es un recurso**. Un recurso puede ser cualquier cosa que necesite ser almacenada o representada, como un usuario, un modelo, un comentario, etc. Cada recurso se identifica de manera única a través de una URL.
- Cada solicitud del cliente al servidor debe contener toda la información necesaria para entender y procesar la solicitud. **El servidor no debe almacenar ninguna información sobre el estado del cliente entre solicitudes**. Esto facilita la escalabilidad de la aplicación, ya que el servidor no necesita gestionar solicitudes.
- **REST proporciona una interfaz uniforme y bien definida** entre un cliente y un servidor, lo que facilita la comunicación y la evolución independiente de ambos. Una interfaz uniforme debe incluir los siguientes principios:

- Identificación de recursos en las solicitudes.
- Manipulación de recursos mediante representaciones.
- Mensajes autodescriptivos
- Hipermedios como motor del estado de la aplicación (HATEOAS)

Durante el diseño de una API REST, es fundamental definir los códigos de respuesta adecuados, ya que proporcionan información sobre el resultado de una solicitud al cliente. Utilizar códigos de respuesta adecuados no solo mejora la comunicación entre cliente y servidor, sino que también ayuda en la gestión de errores y en la optimización de la experiencia del usuario.

A continuación, se presenta una descripción de los códigos de respuesta más comunes y su uso en el contexto de una API REST:



Informational Status Codes

100 – Continue [The server is ready to receive the rest of the request.]

101 – Switching Protocols [Client specifies that the server should use a certain protocol and the server will give this response when it is ready to switch.]

Client Request Successful

200 – OK [Success! This is what you want.]

201 – Created [Successfully created the URI specified by the client.]

202 – Accepted [Accepted for processing but the server has not finished processing it.]

203 – Non-Authoritative Information [Information in the response header did not originate from this server. Copied from another server.]

204 – No Content [Request is complete without any information being sent back in the response.]

205 – Reset Content [Client should reset the current document. Ie. A form with existing values.]

206 – Partial Content [Server has fulfilled the partial GET request for the resource. In response to a Range request from the client. Or if someone hits stop.]

Request Redirected

300 – Multiple Choices [Requested resource corresponds to a set of documents. Server sends information about each one and a URL to request them from so that the client can choose.]

301 – Moved Permanently [Requested resource does not exist on the server. A Location header is sent to the client to redirect it to the new URL. Client continues to use the new URL in future requests.]

302 – Moved Temporarily [Requested resource has temporarily moved. A Location header is sent to the client to redirect it to the new URL. Client continues to use the old URL in future requests.]

303 – See Other [The requested resource can be found in a different location indicated by the Location header, and the client should use the GET method to retrieve it.]

304 – Not Modified [Used to respond to the If-Modified-Since request header. Indicates that the requested document has not been modified since the the specified date, and the client should use a cached copy.]

305 – Use Proxy [The client should use a proxy, specified by the Location header, to retrieve the URL.]

307 – Temporary Redirect [The requested resource has been temporarily redirected to a different location. A Location header is sent to redirect the client to the new URL. The client continues to use the old URL in future requests.]

Client Request Incomplete

400 – Bad Request [The server detected a syntax error in the client's request.]

401 – Unauthorized [The request requires user authentication. The server sends the WWW-Authenticate header to indicate the authentication type and realm for the requested resource.]

402 – Payment Required [reserved for future.]

403 – Forbidden [Access to the requested resource is forbidden. The request should not be repeated by the client.]

404 – Not Found [The requested document does not exist on the server.]

405 – Method Not Allowed [The request method used by the client is unacceptable. The server sends the Allow header stating what methods are acceptable to access the requested resource.]

406 – Not Acceptable [The requested resource is not available in a format that the client can accept, based on the accept headers received by the server. If the request was not a HEAD request, the server can send Content-Language, Content-Encoding and Content-Type headers to indicate which formats are available.]

407 – Proxy Authentication Required [Unauthorized access request to a proxy server. The client must first authenticate itself with the proxy. The server sends the Proxy-Authenticate header indicating the authentication scheme and realm for the requested resource.]

408 – Request Time-Out [The client has failed to complete its request within the request timeout period used by the server. However, the client can re-request.]

409 – Conflict [The client request conflicts with another request. The server can add information about the type of conflict along with the status code.]

410 – Gone [The requested resource is permanently gone from the server.]

411 – Length Required [The client must supply a Content-Length header in its request.]

412 – Precondition Failed [When a client sends a request with one or more If... headers, the server uses this code to indicate that one or more of the conditions specified in these headers is FALSE.]

413 – Request Entity Too Large [The server refuses to process the request because its message body is too large. The server can close connection to stop the client from continuing the request.]

414 – Request-URI Too Long [The server refuses to process the request, because the specified URI is too long.]

415 – Unsupported Media Type [The server refuses to process the request, because it does not support the message body's format.]

417 – Expectation Failed [The server failed to meet the requirements of the Expect request-header.]

Server Errors

500 – Internal Server Error [A server configuration setting or an external program has caused an error.]

501 – Not Implemented [The server does not support the functionality required to fulfill the request.]

502 – Bad Gateway [The server encountered an invalid response from an upstream server or proxy.]

503 – Service Unavailable [The service is temporarily unavailable. The server can send a Retry-After header to indicate when the service may become available again.]

504 – Gateway Time-Out [The gateway or proxy has timed out.]

505 – HTTP Version Not Supported [The version of HTTP used by the client is not supported.]

Unused status codes

306- Switch Proxy

416- Requested range not satisfiable

506- Redirection failed

Ilustración 21. Guía de códigos de respuesta HTTP

Al desarrollar una API REST, es importante ser consciente de los posibles errores que pueden surgir. Estos errores pueden afectar muy negativamente a la usabilidad de la API.

Uno de los errores más comunes es utilizar inconvenientemente los métodos HTTP (GET, POST, PUT, DELETE) como, por ejemplo, utilizar POST para actualizar un recurso.

Otro error muy común es la construcción de URLs con poca consistencia e ineficientes. Las URLs deben ser claras y descriptivas, URLs confusas pueden dificultar la comprensión o el uso de la API.

Por ejemplo, carecería de sentido usar el endpoint *GET /usuarios/{nombreUsuario}* para obtener un usuario específico y al mismo tiempo utilizar el método *GET /valoracion/{nombreUsuario/* para obtener la valoración promedio de un usuario. La solución está en crear URL consistentes, obteniendo *GET /usuarios/{nombreUsuario}/valoracion* para conseguir la valoración de un usuario.

Otro de los errores más repetidos es no proporcionar mensajes de error claros y adecuados, como por ejemplo usar el error 404 Not Found para solicitudes malformadas, para este caso sería necesario utilizar el error 400 Bad Request.

Para garantizar que la API REST sea fácil de entender y utilizar, se ha integrado Swagger UI en la aplicación. Swagger UI es una herramienta de código abierto que permite visualizar los endpoints de la API de manera interactiva. Esta herramienta facilita la documentación de la API y permite a los desarrolladores explorar las diferentes operaciones de manera sencilla. Se puede acceder a Swagger UI una vez desplegado el servicio Spring Boot del backend a través de la siguiente URL: http://localhost:8080/swagger-ui/index.html.

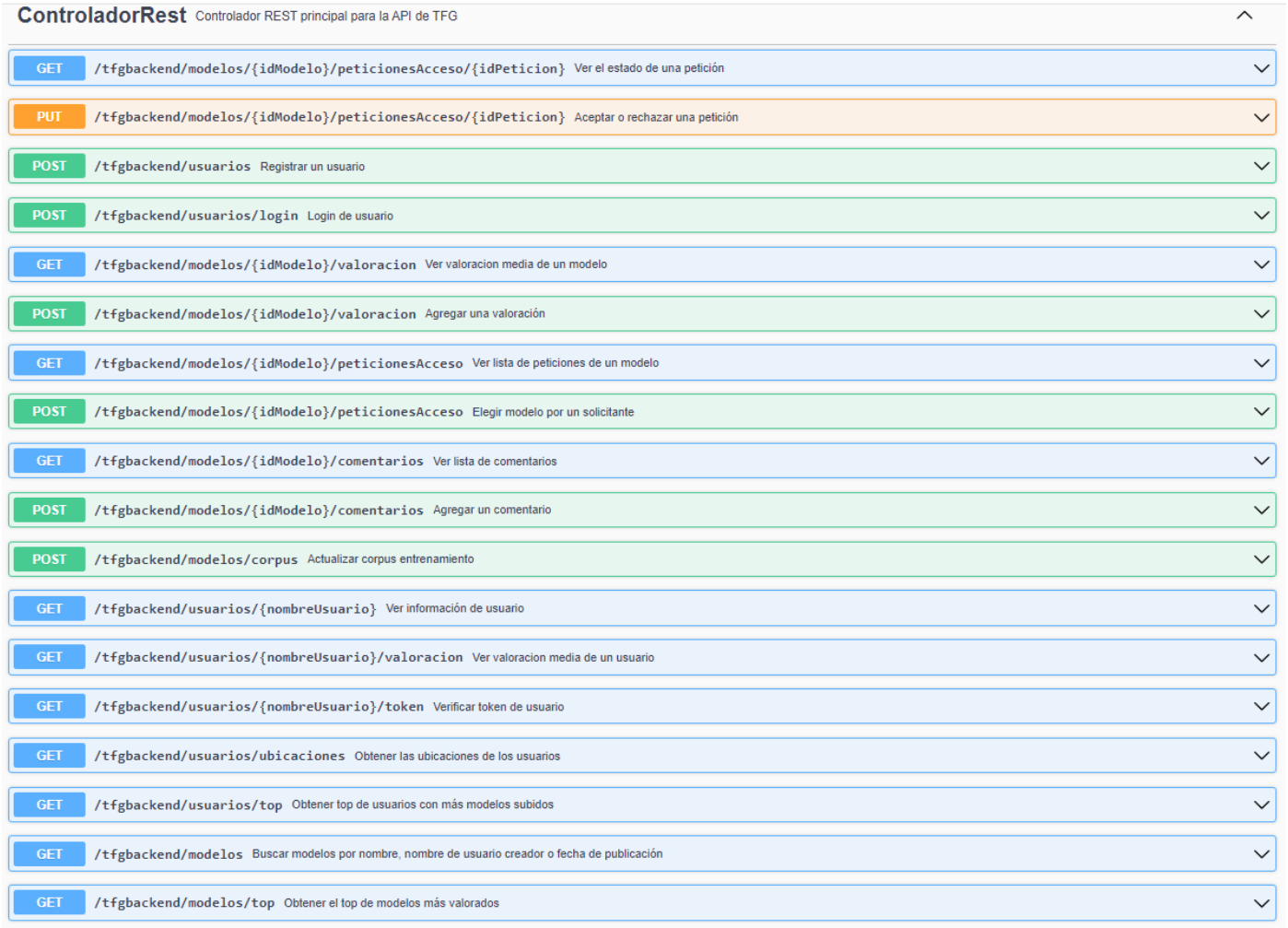


Ilustración 22. Métodos del controlador REST del servicio Spring Boot

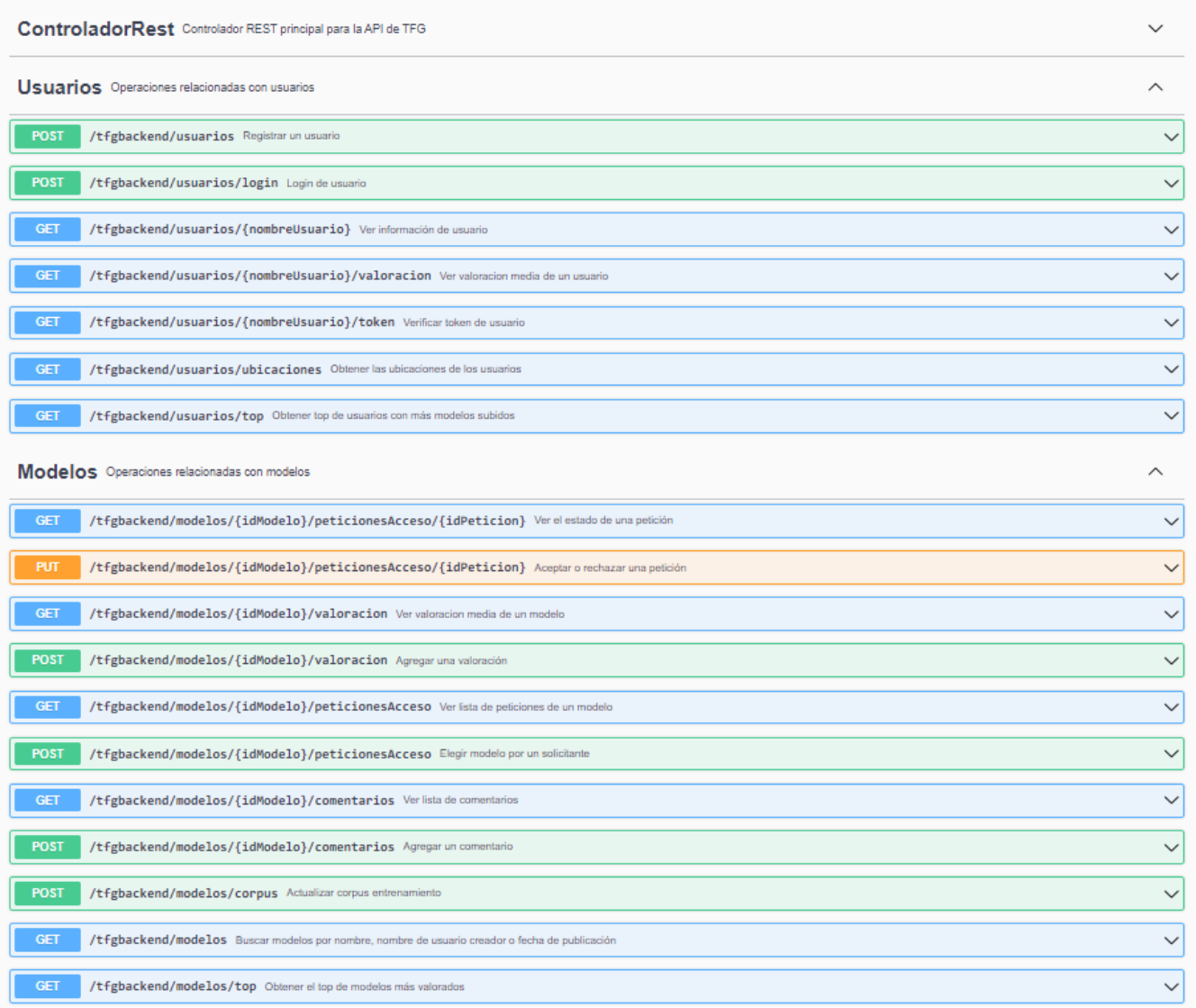


Ilustración 23. Métodos del controlador REST del servicio Spring Boot organizados por entidad relacionada

Ilustración 24. Data Transfer Objects del controlador REST del servicio Spring Boot

A continuación, se muestran los detalles de una operación del servicio a través de la interfaz de Swagger UI, concretamente la operación para obtener la valoración promedio de un usuario.

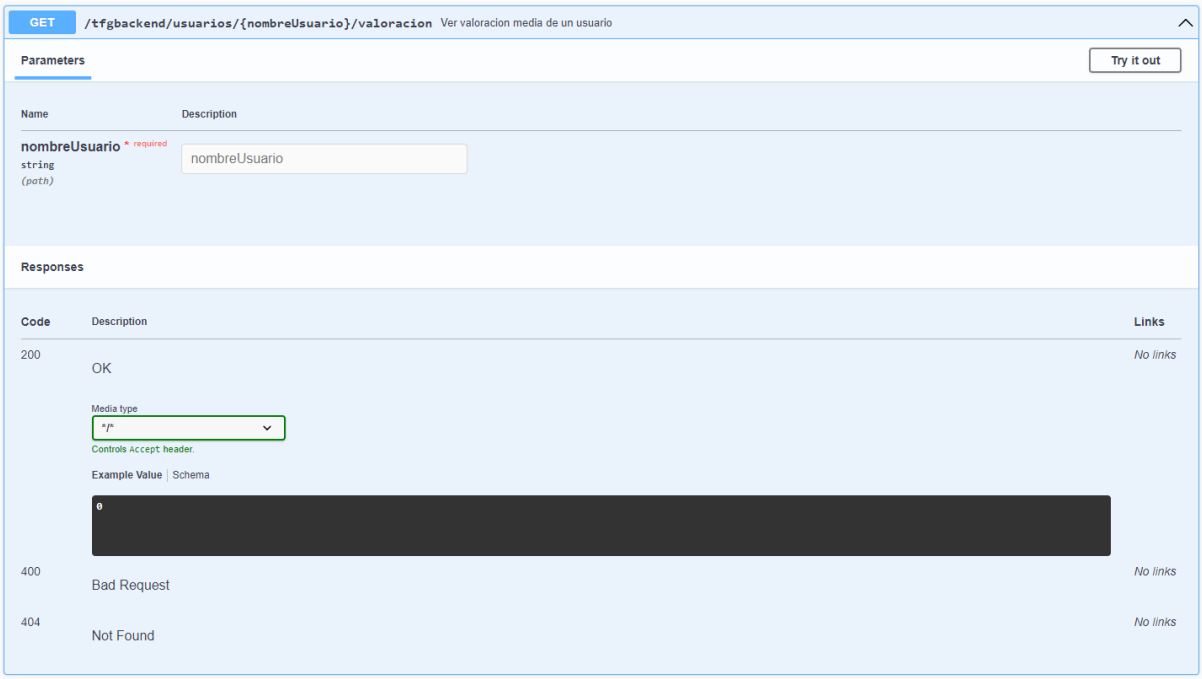


Ilustración 25. Detalles del método GET

4.3.2. Diseño del frontend

4.3.2.1. Introducción al patrón Redux

Para el desarrollo del frontend se ha elegido la biblioteca de JavaScript React detallada en el apartado **3.2.5**. Esta biblioteca no sigue estrictamente patrones de diseño convencionales como el patrón modelo-vista-controlador (MVC), esto se debe a que React se centra en la composición de componentes adoptando un enfoque declarativo para construir interfaces de usuario. Sin embargo, para tener una visión más clara de ese concepto, debemos entender cómo funciona el patrón MVC.

Dentro del patrón MVC encontramos 3 componentes principales dentro de la aplicación:

- **Modelo:** Representa la lógica y gestión de datos de la aplicación
- **Vista:** Presenta los datos del modelo a los usuarios de manera específica sin preocuparse de la lógica de negocio
- **Controlador:** Actúa como intermediario entre el modelo y la vista recibiendo entradas del usuario para posteriormente realizar las operaciones en el modelo y devolver los resultados a la vista

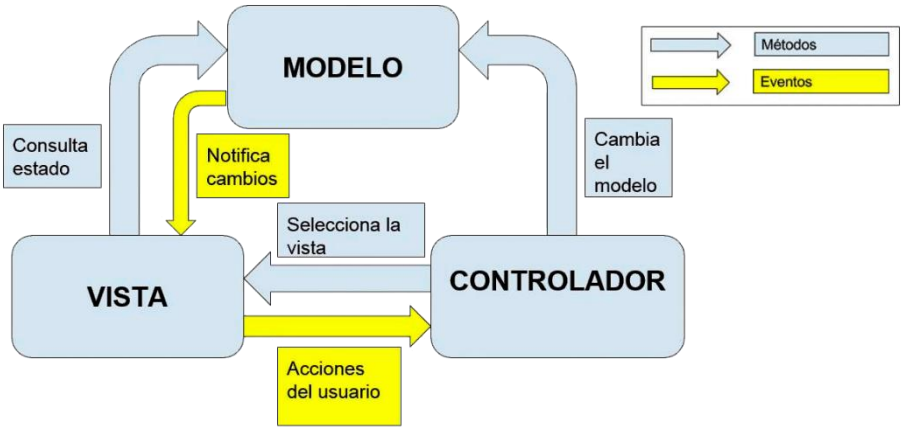


Ilustración 26. Arquitectura MVC

El flujo de datos en el MVC sigue estos pasos: primero el usuario interactúa con la vista; el controlador recibe esa entrada, la procesa y realiza las llamadas al modelo; el modelo realiza las operaciones solicitadas actualizando el estado; finalmente, el controlador toma el resultado del modelo y actualiza la vista para mostrar los cambios al usuario.

React, por el contrario, no utiliza una capa Controlador específica. En su lugar, maneja la interacción y los cambios de estado dentro de los componentes requieriendo bibliotecas como Redux para la gestión global del estado. Además, React presenta otras características que difieren del patrón MVC.

Característica	MVC	React
Modelo	Gestión centralizada de datos	Estado encapsulado dentro de componentes o gestión centralizada usando Redux
Vista	Generación de interfaz basada en el modelo	Componentes declarativos
Controlador	Intermedio entre Modelo y Vista	No existe una capa Controlador específica
Flujo de datos	Bidireccional	Unidireccional
Arquitectura	Separación estricta de capas	Componentes autocontenidos

Tabla 19. Arquitectura React vs MVC

Podemos apreciar que, si bien React no sigue estrictamente las reglas del patrón MVC, presenta ciertas similitudes. Los componentes en React pueden combinar renderizado con lógica de renderizado; sin embargo, la mayoría de estos suelen encontrarse en un punto intermedio, tienen un poco de salida de renderizado y lógica de negocio.

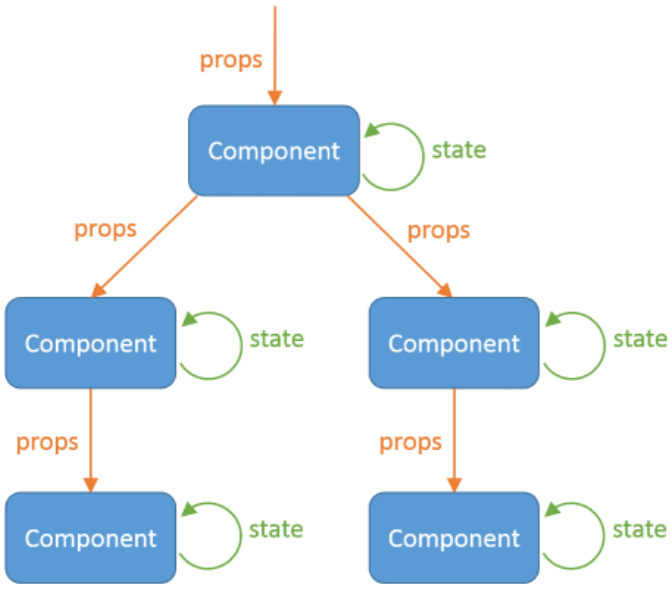


Ilustración 27. Arquitectura de componentes de React

Una de las decisiones más acertadas trabajando con React es usar Redux, que sigue un enfoque conocido como arquitectura Flux. Esta arquitectura permite seguir implementando un flujo de datos unidireccional desde componentes de nivel superior a inferior. Esto significa que los datos siempre fluyen desde la acción al reducer y finalmente al store, facilitando el seguimiento y depuración de datos del estado.

Redux permite además centralizar todo el estado de la aplicación en un único estado. Esto permite a React mantener las propiedades que lo hacen tan interesante como su flujo unidireccional, aplicando la gestión centralizada del estado como se hacía en el patrón MVC.

Característica	MVC	React + Redux
Modelo	Gestión centralizada de datos	Gestión centralizada de datos a partir de los reducers
Vista	Generación de interfaz basada en el modelo	Componentes declarativos
Controlador	Intermedio entre Modelo y Vista	Los Actions gestionan la interacción
Flujo de datos	Bidireccional	Unidireccional
Manejo del estado	Distribuido entre el Modelo y Controlador	Centralizado en el Store

Tabla 20. Arquitectura React + Redux vs MVC

Para ilustrar las ventajas de React + Redux frente al tradicional MVC consideramos el flujo necesario para añadir un comentario dentro de un modelo.

Usando MVC:

1. **Vista:** Se envía la acción de añadir comentario.
2. **Controlador:** Procesa la acción y llama al modelo.
3. **Modelo:** Actualiza la lista de comentarios.
4. **Controlador:** Actualiza la vista con el nuevo estado del tablón de comentarios del modelo.

Usando React + Redux:

1. **Componente Modelo**: Maneja el evento del clic del botón enviar.
2. **Función de acción:** Actualiza el estado de los comentarios.
3. **React:** Vuelve a renderizar el componente del Modelo con el nuevo estado.

Por tanto, esta arquitectura ayuda a mantener la claridad y simplicidad en la gestión del estado, lo cual es esencial para aplicaciones complejas. Estos principios, similares a los del patrón MVC, permiten manejar la separación de preocupaciones, pero de una manera que se adapta mejor al desarrollo de aplicaciones web interactivas y reactivas.

4.3.2.2. Diagrama de clases

Para la implementación del frontend de la aplicación, se ha seguido una arquitectura bien organizada que facilita la gestión del estado y la reutilización de componentes. La estructura del diagrama de clases posteriormente ilustrado refleja esta organización y proporciona una vista clara de cómo interactúan los diferentes módulos de la aplicación en el lado del cliente.

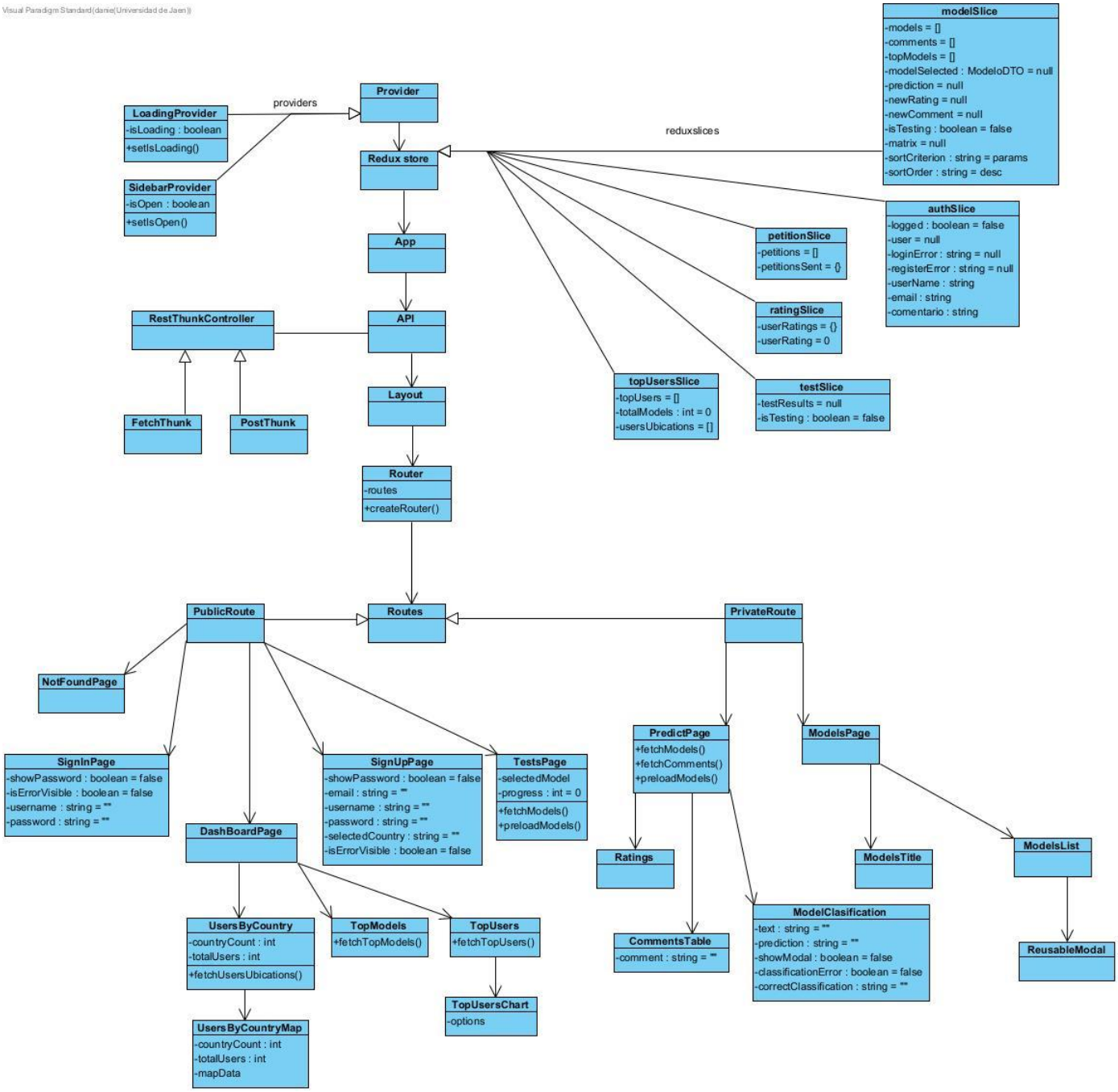


Ilustración 28. Diagrama de clases del frontend

En la anterior imagen podemos observar el esquema de la aplicación, destacando los distintos módulos que la compone. Muchos de estos módulos son conceptos triviales en otras librerías o frameworks como Vue o Angular. Sin embargo, en este proyecto es especialmente relevante la existencia de varios providers que envuelven a la aplicación.

Estos providers, incluyendo el provider principal y otros específicos como LoadingProvider, proporcionan acceso al store de Redux y permiten que cualquier componente de la aplicación acceda al estado global de manera eficiente. Esto facilita la gestión centralizada del estado, una característica esencial en aplicaciones de React con Redux.

El Redux Store se organiza mediante varios slices, cada uno encargado de gestionar una parte específica del estado, como la autenticación (authSlice), los modelos (modelSlice), las peticiones (petitionSlice), entre otros. Esta modularidad mejora la mantenibilidad y escalabilidad de la aplicación.

El módulo Router, del cual derivan PublicRoute y PrivateRoute, controla el acceso a las rutas públicas y privadas respectivamente, asegurando que solo los usuarios autenticados puedan acceder a ciertas áreas de la aplicación.

Además, encontramos los denominados thunk dentro del módulo API, un thunk es una función de Redux que permite escribir lógica para despachar múltiples acciones asíncronas accediendo al estado de la aplicación. Estos thunks son fundamentales para la comunicación entre el frontend y los servicios expuestos por la API REST del backend.

Finalmente, los providers adicionales como LoadingProvider, gestionan estados específicos, como el estado de carga, desde cualquier componente hijo.

5. Planificación y gestión del proyecto

5.1. Estimación del tamaño y esfuerzo

La estimación del tamaño y el esfuerzo es una práctica fundamental en la gestión de proyectos ya que permite planificar de manera realista el trabajo necesario para alcanzar los objetivos propuestos. En el contexto de este TFG, la estimación precisa es crucial para asegurar que todas las fases del proyecto se completen dentro de los plazos establecidos y garantizar la calidad y funcionalidad del sistema desarrollado.

Para este proyecto, se han descompuesto las tareas en historias de usuario, cada una de las cuales describe una funcionalidad específica a desarrollar. A cada historia se le asignan unos puntos de historia que reflejan el esfuerzo requerido para completarla.

Los puntos de historia se asignan en función de la complejidad, riesgo y esfuerzo estimado para cada tarea. Además, se priorizan aquellas que aseguran que las funcionalidades más importantes se desarrollen primero, alineando el desarrollo del proyecto con las expectativas del usuario final.

A continuación, se presenta la tabla de puntos de historia, que incluye la identificación y descripción de cada historia, su prioridad, puntos de historia asignados y dependencias:

ID	Descripción de la historia	Prioridad	Puntos de historia
H01	Configuración del entorno de desarrollo	Alta	4
H02	Implementación de modelos de referencia	Alta	5
H03	Pruebas unitarias para el modelo SVM	Media	2
H04	Implementación de modelos con arquitectura Transformer	Alta	10
H05	Mejoras en los modelos con arquitectura Transformer	Media	8
H06	Integración de la funcionalidad al backend	Alta	6
H07	Desarrollo de la interfaz de usuario	Media	5
H08	Testeo de los modelos a través del frontend	Alta	2
H09	Desarrollo de otro servicio general para el backend	Media	9
H10	Pruebas unitarias y de integración	Baja	4
H11	Actualizar corpus de entrenamiento	Media	3
H12	Obtener métricas de evaluación	Alta	4
H13	Mejoras de la interfaz de usuario	Media	8
H14	Interacción social a través de comentarios y puntuaciones	Baja	5

Tabla 21. Tabla de puntos de historia

5.2. Planificación temporal

Una vez establecido el tamaño y el esfuerzo para cada tarea, podemos proceder a la planificación temporal. En este apartado usaremos un diagrama de Gantt para visualizar el cronograma del proyecto, asegurando que cada tarea se complete en el orden y tiempo adecuados.

El diagrama de Gantt es una herramienta de gestión de proyectos que muestra las tareas programadas a lo largo de un tiempo determinado. Cada tarea se representa como una barra en el gráfico, con la longitud de la barra indicando la duración en días de la tarea. Las dependencias entre tareas también se muestran permitiendo una comprensión clara de cómo se interrelacionan las distintas partes del proyecto.

	Nombre de tarea	Fecha de inicio	Fecha final
		2023/11/13	**2024/05/31**
1	Investigación situación actual	2023/11/13	2023/11/26
2	Estudio de mejoras	2023/11/27	2023/12/10
3	Configuración del entorno de desarrollo	2023/12/11	2023/12/17
4	Implementación de modelos de referencia	2023/12/18	2023/12/31
5	Pruebas unitarias del modelo SVM	2024/01/15	2024/01/21
6	Implementación de modelos con arquitectura Transformer	2024/01/22	2024/02/18
7	Mejoras de los modelos con arquitectura transformer	2024/02/19	2024/03/03
8	Integración de la funcionalidad al backend	2024/03/04	2024/03/10
9	Desarrollo de la interfaz de usuario	2024/03/11	2024/03/17
10	Testeo de los modelos a través del frontend	2024/03/18	2024/03/20
11	Desarrollo de otro servicio general para el backend	2024/03/21	2024/04/07
12	Pruebas de integración	2024/04/08	2024/04/11
13	Actualizar corpus de entrenamiento mediante el front	2024/04/12	2024/04/14
14	Obtener métricas de evaluación	2024/04/15	2024/04/17
15	Mejoras de la interfaz de usuario	2024/04/18	2024/04/28
16	Interacción social a través de comentarios y puntuaciones	2024/04/29	2024/04/30
17	Mejoras finales	2024/05/01	2024/05/31
18	Elaboración de la memoria	2024/03/03	2024/05/31

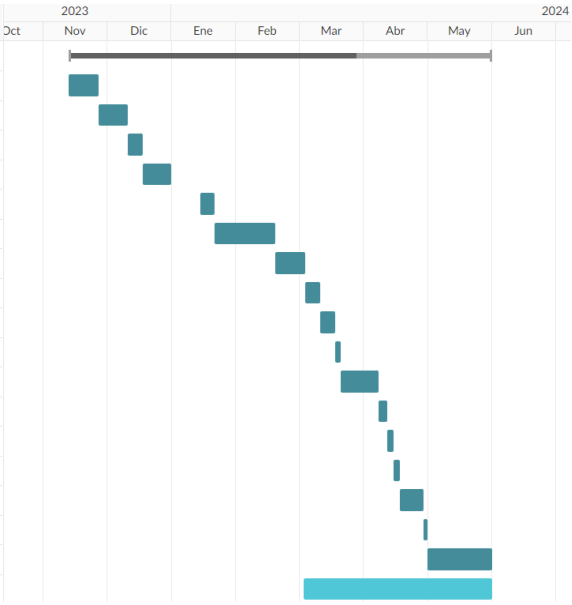


Ilustración 29. Planificación temporal con diagrama de Gantt

5.3. Presupuesto

El presupuesto de este proyecto es un componente esencial para su correcta planificación y ejecución ya que permite evaluar los recursos necesarios y sus costes asociados. En este TFG se han considerado tres tipos principales de costes: Los costes de software, los costes de materiales y los costes derivados del personal. A continuación, se desglosan estos costes para proporcionar una visión clara y detallada del presupuesto total del proyecto.

5.3.1. Costes de software

En este apartado se detallan los costes derivados del uso del software en todo el proyecto.

En la **Tabla 22**, se detalla cada software utilizado, así como el motivo por el que se ha utilizado además del coste asociado a cada programa.

Software	Motivo de uso	Coste (€)
Windows 11 Home	Sistema operativo usado durante el desarrollo	145
Office 365	Suite de aplicaciones de productividad utilizada para documentación y gestión	99
Visual Studio Code	Editor de código utilizado para el desarrollo del frontend y el servicio del backend de la funcionalidad de los modelos	0
NetBeans	Entorno de desarrollo integrado (IDE) utilizado para programar en Java el servicio general del backend	0
Visual Paradigm	Herramienta de modelado UML para diseñar los diagramas y la arquitectura del sistema	0
Microsoft Azure	Plataforma en la nube utilizada para almacenar el corpus de entrenamiento, así como para las métricas de evaluación en contenedores separados	0
TOTAL		**244**

Tabla 22. Costes asociados al software

Tal y como puede apreciarse, la mayoría de programas elegidos son gratuitos. Respecto al software de pago como es el caso de Office 365, aunque existen variantes gratuitas como, por ejemplo, OpenOffice, se ha preferido utilizar su variante de pago debido a la incorporación de funcionalidades más avanzadas, así como un soporte por la comunidad mucho mayor.

5.3.2. Costes de material

Una vez detallados los costes de software en el apartado **5.3.1**, es esencial especificar los costes derivados del material usado para la elaboración de este trabajo.

En este apartado se detallan los costes originados por el equipo informático utilizado, fundamental para elaborar los complejos modelos de aprendizaje profundo que se detallarán en el apartado **6.2.1**, así como los periféricos esenciales para su adecuado desarrollo. Los siguientes materiales han sido elegidos cautelosamente, buscando un equilibrio entre rendimiento y precio capaces de otorgar un hardware lo suficientemente potente para ejecutar tareas avanzadas de Procesamiento de Lenguaje Natural sin necesidad de requerir un alto coste.

Material	Características clave	Necesidades satisfechas	Coste (€)
Portátil Gigabyte Aorus 15P XD	Tarjeta gráfica NVIDIA RTX 3070 mobile de 8GB	Entrenamiento de modelos transformer gracias a una gran potencia de cómputo en FP32, así como una suficiente cantidad de VRAM	1299
Ratón Razer Deathadder Essential	Diseño ergonómico, sensor óptico de alta precisión	Precisión y comodidad durante largas sesiones de programación y desarrollo	30
Teclado Newskill Hanshi Spectrum	Switches kailh brown y teclas RGB	Baja sonoridad gracias a los switches y visibilidad durante sesiones nocturnas gracias al RGB	80
Monitor Acer vg270up	Resolución QHD (2560x1440), 27”, IPS, 99% sRGB, 144hz	Mayor productividad gracias a la segunda pantalla, gran fidelidad del frontend gracias a la alta cobertura del espacio sRGB	329
TOTAL			**1738**

Tabla 23. Costes asociados al material

Tal y como puede apreciarse en la **Tabla 23**, uno de los elementos que más repercuten en el coste del material es el equipo utilizado. Si bien existen portátiles mucho más baratos que también proporcionan tarjetas gráficas dedicadas, para este proyecto era necesario elegir un portátil que tuviera un mínimo de 8GB de VRAM

debido a la alta demanda de este componente hardware por parte de los modelos de aprendizaje profundo. Dentro del ámbito de los ordenadores portátiles, la NVIDIA RTX 3070m es la tarjeta gráfica de menor gama y, por tanto, de menor coste que ofrece los ansiados 8GB de VRAM, necesarios tal y como se detallará posteriormente para realizar las tareas requeridas en este TFG.

Además, para la elección de la GPU se han descartado completamente las opciones de AMD[3] ya que si bien es cierto ofrecen una mayor cantidad de VRAM, sufren serias incompatibilidades con librerías de enorme importancia como Pytorch o Tensorflow, las cuales precisan de la arquitectura CUDA exclusiva a día de hoy en las tarjetas NVIDIA.

5.3.3. Costes de personal

Una vez concluidos los costes de los elementos necesarios para la elaboración de este proyecto, es de vital importancia definir los costes derivados del tiempo y esfuerzo de los diferentes perfiles que colaboran en el desarrollo del sistema propuesto.

En la siguiente tabla se definen los siguientes roles asociados al desarrollo, así como los gastos asociados a la realización de sus respectivas tareas.

[3] https://www.amd.com/es.html

Rol	Salario Medio Bruto por Hora (€)	Horas Estimadas	Coste Total (€)
Analista y programador de modelos de aprendizaje profundo	35	120	4250
Programador experto en extracción de datos	30	20	600
Programador backend en Flask	32	30	960
Programador backend en Java Spring Boot	32	60	1920
Programador frontend en React	29	60	1740
Diseñador gráfico de interfaces web	27	10	270
TOTAL			**9740**

Tabla 24. Costes asociados al personal

Para obtener tales cifras se han utilizado portales como SalaryXplorer[4] o ERI Economic Research Institute[5] que recopilan cifras medias aproximadas de los salarios brutos que ven reflejados dichos roles por sus esfuerzos. Los cálculos de los costes finales han sido ajustados en función del número de horas requeridas para el desempeño de cada tarea.

5.3.4. Costes totales

Tras exponer los costes desglosados en función de su propia naturaleza, en este apartado se define la suma total que acumulan todos esos costes en conjunto. Además, se incluyen algunos gastos indirectos que, si bien no se pueden atribuir directamente a una tarea específica del proyecto, son esenciales para el desarrollo y la gestión del mismo. Dentro de estos costes indirectos se incluyen algunos como la luz, el agua, la tarifa de Internet o el alquiler de la zona de trabajo.

[4] https://www.salaryexplorer.com/
[5] https://www.erieri.com/salaryreport/researcher

Basándonos en fuentes contrastadas como el Project Management Institute, podemos contrastar que de media los costes indirectos oscilan entre el 10 y el 20% del presupuesto total de un sistema. Por tanto, en este proyecto se considerará un punto intermedio entre los 2 extremos mencionados, lo cual constituye unos gastos indirectos del 15%.

Concepto	Coste (€)
Costes de software	244
Costes materiales	1738
Costes de personal	9740
SUBTOTAL	**11722**
Costes indirectos (15%)	1758,3
TOTAL	**13480,3**

Tabla 25. Costes finales desglosados

Finalmente podemos concluir que el proyecto tiene un coste final de **13480,3 €** incluyendo todos los elementos descritos.

5.4. Ejecución de los Sprints

En este apartado, se detalla el proceso de planificación y ejecución de los Sprints de acuerdo a la metodología Scrum que ha sido utilizada para el desarrollo del proyecto. Estos Sprints se estructuran a partir del diagrama de puntos de historia expuesto en el apartado **5.1**. Cada sprint se enfoca en un conjunto específico de tareas, distribuidas según su prioridad y esfuerzo requerido. Los Sprints están diseñados para abordar de manera gradual las distintas funcionalidades principales, las mejoras y pruebas necesarias.

A continuación, se presenta una descripción detallada de cada sprint. Cada Sprint vendrá acompañado de la tabla de puntos de historia de referencia donde se colorean en verde aquellas historias que pertenezcan a cada Sprint.

5.4.1. Sprint 1

El objetivo principal del Sprint 1 es establecer una base sólida para el proyecto configurando el entorno de desarrollo e implementando los modelos de referencia iniciales. Esto asegura tener un punto de partida sólido para implementar funciones más complejas en Sprints posteriores.

ID	Descripción de la historia	Prioridad	Puntos de historia
H01	Configuración del entorno de desarrollo	Alta	4
H02	Implementación de modelos de referencia	Alta	5
H03	Pruebas unitarias para el modelo SVM	Media	2
H04	Implementación de modelos con arquitectura Transformer	Alta	10
H05	Mejoras en los modelos con arquitectura Transformer	Media	8
H06	Integración de la funcionalidad al backend	Alta	6
H07	Desarrollo de la interfaz de usuario	Media	5
H08	Testeo de los modelos a través del frontend	Alta	2
H09	Desarrollo de otro servicio general para el backend	Media	9
H10	Pruebas unitarias y de integración	Baja	4
H11	Actualizar corpus de entrenamiento	Media	3
H12	Obtener métricas de evaluación	Alta	4
H13	Mejoras de la interfaz de usuario	Media	8
H14	Interacción social a través de comentarios y puntuaciones	Baja	5

Tabla 26. Historias del Sprint 1

La ejecución de este sprint no fue tan compleja como los Sprints siguientes, sin embargo, fue muy importante cerciorarse de que los modelos de referencia estuvieran bien construidos para tener una base sólida en el proyecto.

5.4.2. Sprint 2

El objetivo principal del Sprint 2 es avanzar en la implementación de modelos más complejos utilizando arquitecturas Transformer. Este Sprint tiene como meta elaborar sistemas de clasificación con arquitecturas sólidas que proporcionen resultados superiores a los obtenidos en el Sprint anterior.

ID	Descripción de la historia	Prioridad	Puntos de historia
H01	Configuración del entorno de desarrollo	Alta	4
H02	Implementación de modelos de referencia	Alta	5
H03	Pruebas unitarias para el modelo SVM	Media	2
H04	Implementación de modelos con arquitectura Transformer	Alta	10
H05	Mejoras en los modelos con arquitectura Transformer	Media	8
H06	Integración de la funcionalidad al backend	Alta	6
H07	Desarrollo de la interfaz de usuario	Media	5
H08	Testeo de los modelos a través del frontend	Alta	2
H09	Desarrollo de otro servicio general para el backend	Media	9
H10	Pruebas unitarias y de integración	Baja	4
H11	Actualizar corpus de entrenamiento	Media	3
H12	Obtener métricas de evaluación	Alta	4
H13	Mejoras de la interfaz de usuario	Media	8
H14	Interacción social a través de comentarios y puntuaciones	Baja	5

Tabla 27. Historias del Sprint 2

A diferencia de otros Sprint, éste cuenta únicamente con una sola tarea debido a la enorme complejidad que puede verse reflejada en sus 10 puntos de historia. Tal y como se detallará en el apartado **6.2.1**, existen numerosos modelos que se basan en la arquitectura Transformer, por tanto, la tarea de implementar y comparar cada uno de estos modelos supone un desafío considerable.

5.4.3. Sprint 3

El objetivo principal de este sprint es mejorar los modelos implementados en el Sprint 2 e integrar la funcionalidad de estos modelos en el backend. Esto permitirá asegurar que los modelos sean más eficientes y se puedan usar desde un cliente.

ID	Descripción de la historia	Prioridad	Puntos de historia
H01	Configuración del entorno de desarrollo	Alta	4
H02	Implementación de modelos de referencia	Alta	5
H03	Pruebas unitarias para el modelo SVM	Media	2
H04	Implementación de modelos con arquitectura Transformer	Alta	10
H05	Mejoras en los modelos con arquitectura Transformer	Media	8
H06	Integración de la funcionalidad al backend	Alta	6
H07	Desarrollo de la interfaz de usuario	Media	5
H08	Testeo de los modelos a través del frontend	Alta	2
H09	Desarrollo de otro servicio general para el backend	Media	9
H10	Pruebas unitarias y de integración	Baja	4
H11	Actualizar corpus de entrenamiento	Media	3
H12	Obtener métricas de evaluación	Alta	4
H13	Mejoras de la interfaz de usuario	Media	8
H14	Interacción social a través de comentarios y puntuaciones	Baja	5

Tabla 28. Historias del Sprint 3

La ejecución de este Sprint ha sido una de la más tediosas, debido a que las mejoras en los modelos implican ajustar hiperparámetros, incorporar técnicas de post procesado, entre otros. Por otro lado, la integración de la funcionalidad de estos modelos como su testeo, clasificación… necesitaron la elaboración de un servicio usando Flask.

5.4.4. Sprint 4

El objetivo principal del Sprint 4 es desarrollar una interfaz de usuario básica que facilite la interacción con el sistema y la prueba de los modelos de clasificación.

ID	Descripción de la historia	Prioridad	Puntos de historia
H01	Configuración del entorno de desarrollo	Alta	4
H02	Implementación de modelos de referencia	Alta	5
H03	Pruebas unitarias para el modelo SVM	Media	2
H04	Implementación de modelos con arquitectura Transformer	Alta	10
H05	Mejoras en los modelos con arquitectura Transformer	Media	8
H06	Integración de la funcionalidad al backend	Alta	6
H07	Desarrollo de la interfaz de usuario	Media	5
H08	Testeo de los modelos a través del frontend	Alta	2
H09	Desarrollo de otro servicio general para el backend	Media	9
H10	Pruebas unitarias y de integración	Baja	4
H11	Actualizar corpus de entrenamiento	Media	3
H12	Obtener métricas de evaluación	Alta	4
H13	Mejoras de la interfaz de usuario	Media	8
H14	Interacción social a través de comentarios y puntuaciones	Baja	5

Tabla 29. Historias del Sprint 4

Este sprint es uno de los que ha requerido menos tiempo y esfuerzo, sin embargo, supone un gran avance para empezar a evaluar y mejorar la usabilidad de la aplicación. Aunque las tareas de este Sprint son menos complejas, la creación de una interfaz funcional es el requisito indispensable para la versión final.

5.4.5. Sprint 5

El objetivo principal del Sprint 5 es construir un servicio que permita gestionar la interacción social de la aplicación web, lo cual es clave para que los modelos sean fácilmente utilizados y valorados por la comunidad.

ID	Descripción de la historia	Prioridad	Puntos de historia
H01	Configuración del entorno de desarrollo	Alta	4
H02	Implementación de modelos de referencia	Alta	5
H03	Pruebas unitarias para el modelo SVM	Media	2
H04	Implementación de modelos con arquitectura Transformer	Alta	10
H05	Mejoras en los modelos con arquitectura Transformer	Media	8
H06	Integración de la funcionalidad al backend	Alta	6
H07	Desarrollo de la interfaz de usuario	Media	5
H08	Testeo de los modelos a través del frontend	Alta	2
H09	Desarrollo de otro servicio general para el backend	Media	9
H10	Pruebas unitarias y de integración	Baja	4
H11	Actualizar corpus de entrenamiento	Media	3
H12	Obtener métricas de evaluación	Alta	4
H13	Mejoras de la interfaz de usuario	Media	8
H14	Interacción social a través de comentarios y puntuaciones	Baja	5

Tabla 30. Historias del Sprint 5

Tal y como se puede comprobar en la tabla, este sprint incluye pocas tareas, pero cada una requiere una gran cantidad de tiempo y esfuerzo, reflejados en puntos de historia. La creación de un servicio que gestione la interactividad de la plataforma representa el segundo mayor esfuerzo para la elaboración del proyecto. Esto se debe a la necesidad de crear un backend lo suficientemente robusto, eficiente y escalable para manejar miles de transacciones simultáneas.

5.4.6. Sprint 6

El objetivo principal de este Sprint final es añadir funcionalidades no tan importantes como las previamente descritas, pero que juegan un papel importante a la hora de que los usuarios muestren interés por la plataforma.

ID	Descripción de la historia	Prioridad	Puntos de historia
H01	Configuración del entorno de desarrollo	Alta	4
H02	Implementación de modelos de referencia	Alta	5
H03	Pruebas unitarias para el modelo SVM	Media	2
H04	Implementación de modelos con arquitectura Transformer	Alta	10
H05	Mejoras en los modelos con arquitectura Transformer	Media	8
H06	Integración de la funcionalidad al backend	Alta	6
H07	Desarrollo de la interfaz de usuario	Media	5
H08	Testeo de los modelos a través del frontend	Alta	2
H09	Desarrollo de otro servicio general para el backend	Media	9
H10	Pruebas unitarias y de integración	Baja	4
H11	Actualizar corpus de entrenamiento	Media	3

H12	Obtener métricas de evaluación	Alta	4
H13	Mejoras de la interfaz de usuario	Media	8
H14	Interacción social a través de comentarios y puntuaciones	Baja	5

Tabla 31. Historias del Sprint 6

Podemos comprobar en la tabla que este sprint final se compone de una gran cantidad de historias, la mayoría de baja complejidad, con la excepción de H13. En esta etapa se añaden más funcionalidades al backend construido en el sprint anterior, como la obtención de métricas de evaluación al pasar los test sobre un determinado modelo, la actualización del corpus con nuevo conocimiento etiquetado y mejoras en la interfaz de usuario para aumentar la usabilidad de la aplicación.

6. Implementación

6.1. Extracción del corpus

La extracción del corpus tal y como se ha detallado supone una fase fundamental para entrenar y evaluar los modelos realizados en este trabajo. Un corpus de calidad debe ser representativo del tipo de lenguaje que se quiere analizar y estar adecuadamente etiquetado.

En este trabajo, se ha utilizado el corpus OffendES[6] para la detección de lenguaje ofensivo. Este corpus ha sido utilizado en varias tareas de investigación de PLN, especialmente en el contexto de la clasificación de textos ofensivos.

Uno de los principales motivos que ha propiciado la elección de este corpus es el tipo de información que contiene. Para la implementación de un sistema capaz de clasificar el lenguaje ofensivo, es crucial obtener textos de entrenamiento que contengan insultos, connotaciones negativas o sarcasmo. La gran virtud de este corpus es haber incorporado información sobre comentarios publicados en distintas redes sociales, que son una fuente importante del discurso de odio.

La estructura del corpus es relativamente simple, contiene un conjunto de información distribuida en varias columnas: el id del comentario, el comentario, el influencer que recibió el comentario, el género del comentador, la red social donde se comentó y una etiqueta manual que representa la connotación del comentario. Las categorías que puede tener un comentario son las siguientes:

- **0 u OFP:** Representan comentarios **of**ensivos hacia una sola **p**ersona.
- **1 u OFG:** Representan comentarios **of**ensivos hacia **g**rupos o colectivos.
- **2 o NO:** Representan comentarios **n**o **o**fensivos.
- **3 o NOE:** Representan comentarios **n**o **o**fensivos, pero con contenido **e**xplícito o soez.

El corpus contiene una gran cantidad de información para cada frase, lo cual permite que sea usado para una gran variedad de tareas además de clasificar el lenguaje ofensivo como, por ejemplo, detectar posible misoginia entre otros casos. Sin

[6] https://huggingface.co/datasets/fmplaza/offendes

embargo, la característica más importante que nos ofrece este corpus en comparación con otros es la presencia de múltiples categorías para clasificar el lenguaje ofensivo. Muchos otros corpus presentan simplemente comentarios etiquetados como ofensivos o no ofensivos, sin prestar atención a detalles tan importantes como si la ofensividad va dirigida a una persona o varias, o si se incorpora contenido soez aun sin ser ofensivo.

Esto, combinado con los más de 30 mil comentarios agrupados en los distintos conjuntos de datos de OffendES, proporciona un punto de partida excelente para construir modelos clasificatorios de lenguaje ofensivo altamente precisos.

Los conjuntos de datos que nos proporciona el corpus son los siguientes:

- **Datos de entrenamiento:** El corpus de entrenamiento es el conjunto de datos utilizado para ajustar los parámetros del modelo. En esta fase, el modelo aprende a identificar patrones y características del lenguaje basándose en los ejemplos proporcionados. Este es el conjunto de datos más grande, ocupando un total de 16710 ejemplos, ya que se necesita una gran cantidad de datos para que el sistema generalice correctamente.
- **Datos de desarrollo:** El corpus de desarrollo, también conocido como conjunto de validación, se utiliza durante el entrenamiento del modelo para evaluar su rendimiento y ajustar los hiperparámetros. Permite prever cómo el modelo se comportaría con datos no vistos en el entrenamiento. Este conjunto de datos tiene 100 ejemplos.
- **Datos de prueba:** El corpus de prueba se utiliza exclusivamente para evaluar el rendimiento final del modelo después de haber completado el entrenamiento. Proporciona una estimación imparcial de la capacidad del modelo para generalizar datos completamente nuevos. Este corpus tiene 13607 ejemplos.

A continuación, se muestra el fragmento de código en Python necesario para extraer el corpus.

```
 1. from datasets import load_dataset
 2. import csv
 3.
 4. # Cargar el dataset
 5. dataset = load_dataset("fmplaza/offendes")
 6.
 7. # Guardar el split de entrenamiento en un TSV
 8. datos_entrenamiento = dataset['dev_set']
 9.
10. nombre_archivo = 'offendes_dev.tsv'
11.
12. # Abrir el archivo para escribir con el modo 'w' para escribir y 'newline='' para evitar
líneas en blanco
13. with open(nombre_archivo, 'w', newline='', encoding='utf-8') as archivo:
14.     # Crear un objeto writer de csv que usará tabulaciones como delimitadores
15.     escritor = csv.writer(archivo, delimiter='\t')
16.
17.     # Nombres de las columnas como la primera fila del archivo TSV
18.     escritor.writerow(datos_entrenamiento.column_names)
19.
20.     # Iterar sobre los registros del conjunto de datos y escribir cada uno en el archivo
21.     for registro in datos_entrenamiento:
22.         escritor.writerow([registro[columna] for columna in
datos_entrenamiento.column_names])
```

Código 2. Carga de los dataset

El **Código 2** utiliza la librería datasets para cargar el conjunto de datos OffendES desde el repositorio de datasets "fmplaza/offendes". Una vez cargado el conjunto de datos, se crea un objeto csv.writer para iterar todos los registros del conjunto de datos y poder escribirlos en el archivo, columna por columna.

En este ejemplo se muestra la extracción del corpus de desarrollo, sin embargo, la extracción de los 2 corpus restantes es trivial. Únicamente es necesario cambiar el valor asociado a la línea 8 de código y poner ‘train_set’ o ‘test_set’ en lugar de ‘dev_set’.

6.2. Desarrollo de los modelos clasificatorios

6.2.1 Estudio de técnicas para la clasificación del lenguaje ofensivo

En este capítulo se detallan los conceptos y aspectos técnicos que se han usado como base para la creación de un sistema capaz de clasificar e identificar posibles textos que incorporan lenguaje ofensivo. En primera instancia se describen los conceptos de los métodos sobre la representación del texto que se han utilizado tradicionalmente, así como los más actuales. Posteriormente se detallan las diferentes arquitecturas de clasificadores basados en aprendizaje no supervisado y aprendizaje supervisado que han surgido gracias a la representación de ese texto.

6.2.1.1. Técnicas de representación del texto

La representación textual es un paso crítico en cualquier tarea de Procesamiento del Lenguaje Natural. La idea consiste en convertir texto, que es intrínsecamente no estructurado y variado, en algún tipo de información estructurada, generalmente bolsas de palabras o vectores de números que permitan a los algoritmos trabajar entre ellos de manera más efectiva.

6.2.1.1.1. Técnicas sin embedding

Para entender bien esta representación de texto es de vital importancia entender el concepto básico que sirvió como piedra angular: El One-Hot-Encoding. Esta técnica consiste en representar cada palabra como un vector en un espacio de alta dimensionalidad donde cada dimensión corresponde a una palabra única en el vocabulario del corpus. Si el corpus tiene X palabras, entonces cada palabra se ve representada por un vector de X elementos, donde un elemento es “1” indicando la presencia de una palabra, y los X-1 elementos restantes son “0”, indicando la ausencia de las demás palabras.

Por ejemplo, si consideramos un vocabulario simple {el, chico, juega, al, fútbol}. La palabra “juega” se representaría como [0,0,1,0,0] donde cada posición representa una palabra específica del vocabulario. Esta representación es intuitiva, pero a la vez, peca de ser sumamente ineficiente para vocabularios de gran tamaño, ya que resulta en vectores esparcidos con una gran cantidad de ceros, lo cual no es solo un enorme desperdicio de memoria, sino que además carece de capacidad para capturar relaciones semánticas entre palabras.

Por tanto, encontramos en One-Hot-Encoding una gran limitación, la incapacidad de capturar el contexto y la semántica de todas las palabras puesto que todas ellas son equidistantes entre sí en este espacio geométrico de representación. Esto llevó al desarrollo de técnicas ligeramente más avanzadas como los modelos Bag of Words (BoW). Un modelo BoW es una forma simple de extraer características de texto, pero con una sutil diferencia respecto a los modelos precedentes, BoW representa la frecuencia de aparición de una palabra en el documento lo que permite reducir en parte la inmensa dimensionalidad que padecían los modelos anteriores.

La representación básica de BoW puede entenderse como un vector $\vec{d}$ donde cada elemento d_i representa la frecuencia de la i-ésima palabra del vocabulario en el documento. Considerando un vocabulario V con N palabras entonces un documento D se representa como:

$$\vec{d} = [d_1, d_2, \dots, d_N]$$

Donde d_i es el número de veces que la palabra i aparece en el documento

Este modelo si bien es cierto mitiga algunos de los defectos de los sistemas clásicos, One-Hot-Encoding preserva significativas limitaciones. En primer lugar, BoW sigue sin capturar la estructura lingüística o el contexto en el cual las palabras aparecen por lo que es deficiente a la hora de distinguir palabras polisémicas. Además, para grandes corpus, BoW aún puede llevar a vectores de una gran dimensionalidad siendo ineficiente.

Por otro lado, encontramos modelos similares a BoW que implementan algunas técnicas más avanzadas como Term Frequency-Inverse Document Frecuency (TF-IDF). TF-IDF pondera la frecuencia de las palabras, otorgando más importancia a las palabras que son frecuentes en un documento, pero no en todo el corpus. TF-IDF utiliza 2 componentes:

- **TF:** Funciona de forma parecida a BoW, donde se calcula la frecuencia de una palabra en un documento. Esto refleja cuán relevante es una palabra dentro del documento individual. Hay varias maneras de calcular TF. Una forma simple es contar el número de veces que el término aparece en el documento y dividirlo por el número total de términos en el documento.

$$TF(t,d) = \frac{\text{Número de veces que el término } t \text{ aparece en el documento } d}{\text{Número total de términos en el documento } d}$$

- **IDF:** Esta componente reduce la importancia (peso) de las palabras que aparecen en muchos documentos del corpus, considerándolas menos relevantes para cualquier documento específico. La idea es que las palabras que aparecen en muchos documentos no son buenos discriminadores, incluyendo probablemente menos información útil sobre el contenido específico del documento. Se obtiene dividiendo el número total de documentos por el número de documentos que contienen el término, y luego tomando el logaritmo del coeficiente.

$$IDF(t,D) = \log\left(\frac{\text{Número total de documentos en el corpus } D}{\text{Número de documentos donde el término } t \text{ aparece}}\right)$$

La combinación de TF e IDF (TF-IDF) resulta en una puntuación que refleja la importancia de una palabra para un documento en relación a un corpus de documentos. Las palabras comunes como “el”, ”y” y ”en” que aparecen en muchos documentos tendrán un valor IDF bajo, mientras que términos más específicos y relevantes para el tema de un documento tendrán un valor IDF alto. La puntuación TF-IDF de un término en un documento específico se obtiene como el producto de su TF e IDF

$$TFIDF(t,d,D) = TF(t,d) \times IDF(t,D)$$

TF-IDF ofrece una forma elegante y potente para ponderar los términos de los documentos, destacando la importancia de los términos específicos del documento en relación con el corpus general.

Sin embargo, aunque los modelos Bag of Words y sobre todo los modelos TF-IDF ofrecen un punto de partida sólido focalizando algunos de los problemas críticos de las técnicas tradicionales, sus limitaciones como la necesidad de entender el contexto de la representación textual, destacan la necesidad de usar técnicas de representación más sofisticadas, como Word Embedding que superan estas restricciones proporcionando representaciones más densas y semánticamente ricas del texto.

6.1.1.1.2. Técnicas con embedding

Las técnicas de representación vectorial del texto, como los Word Embedding marcan una evolución significativa respecto a métodos más simples como BoW o TF-IDF. Estos modelos representan palabras en vectores de menor dimensionalidad, capturando similitudes semánticas y relacionales. Esto se logra aprendiendo estos vectores a partir de grandes conjuntos de datos de texto, de modo que el modelo captura contextos y relaciones semánticas complejas.

Algunos de los mayores precursores en el campo del embedding se pueden encontrar en el modelo Word2vec propuesto en el artículo [5] que marcó un hito en el campo del PLN para la representación de datos vectoriales de forma embedida. Una de las arquitecturas más importantes fue Continuous Bag of Words (CBOW) el cual predice la palabra actual basándose en un contexto. Este contexto se define como las palabras circundantes entorno a la palabra evaluada.

La función objetivo de CBOW se centra en maximizar la probabilidad condicional de la palabra objetivo dado un contexto.

$$P(w_t|\text{context}) = \frac{\exp\left(v'_{w_t}{}^{\top} v_{\text{context}}\right)}{\sum_{w \in V} \exp\left(v'_{w}{}^{\top} v_{\text{context}}\right)}$$

Donde $v_{context}$ es el valor promedio de los vectores de contexto y V es el vocabulario.

También encontramos Skip-Gram como arquitectura inversa a CBOW donde se utiliza una palabra para predecir su contexto. Esta arquitectura es muy eficiente para detectar palabras raras. La función objetivo de Skip-gram busca maximizar la probabilidad condicional del contexto dada una palabra objetivo.

$$P(\text{context}|w_t) = \prod_{-c\leq j\leq c, j\neq 0} \frac{\exp\left({v'_{w_{t+j}}}^{T} v_{w_t}\right)}{\sum_{w\in V}\exp\left({v'_w}^{T} v_{w_t}\right)}$$

En ambas fórmulas, v'_w y v'_{w_t} son los vectores de la palabra objetivo y las palabras de contexto, respectivamente, y c es el tamaño del contexto.

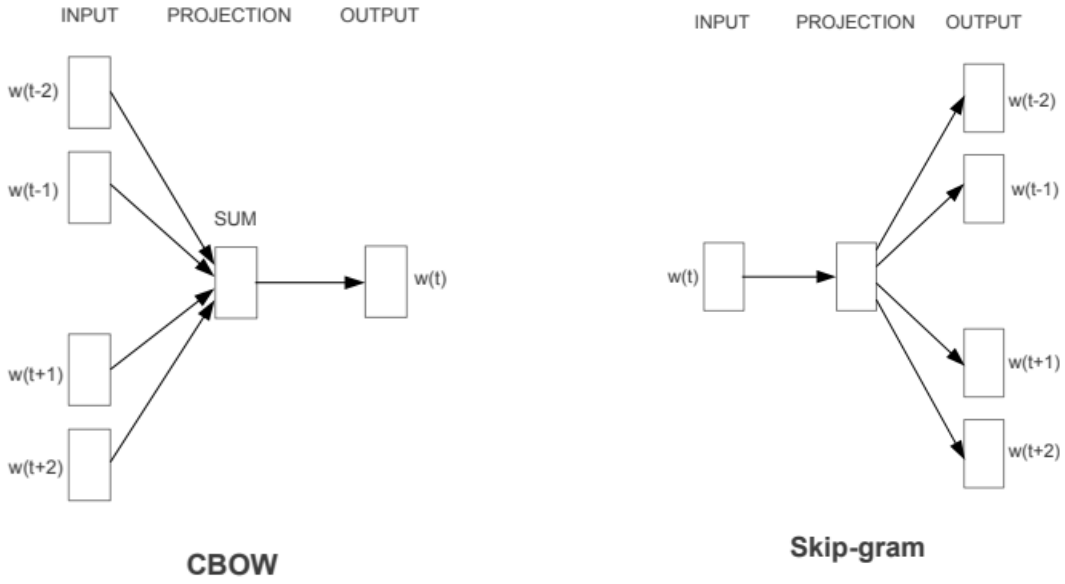


Ilustración 30. Función FBOW y Skip-gram

Las innovaciones introducidas por Word2Vec, especialmente en términos de eficiencia computacional y calidad de los embeddings, allanaron el camino para un uso más amplio del DL en PLN. Sin embargo, los embeddings de Word2Vec son estáticos y no capturan variaciones de significado en diferentes contextos.

6.2.1.2. Modelos de clasificación

En este apartado se describen los distintos modelos clasificadores de texto basados en aprendizaje supervisado surgidos gracias a la investigación de las técnicas de representación textuales descritas anteriormente. Se detallan los modelos que utilizan técnicas de aprendizaje automático, así como los que utilizan técnicas más avanzadas basadas en aprendizaje profundo. Finalmente se mostrarán sus ventajas e impedimentos que propiciaron con el paso de los años la aparición de una de las mejores arquitecturas existentes para la clasificación textual, la arquitectura Transformer.

6.2.1.2.1. Modelos de clasificación supervisados

La clasificación supervisada es uno de los pilares fundamentales del aprendizaje automático. Estos modelos se entrenan sobre conjuntos de datos etiquetados, donde el objetivo es aprender a predecir la etiqueta o categoría de nuevas instancias basándose en las características aprendidas durante el entrenamiento. En el contexto del PLN, la clasificación supervisada permite a las máquinas realizar tareas como la que viene reflejada en este trabajo, la clasificación de mensajes de odio.

Estos modelos clasificatorios precisan de datos etiquetados, un conjunto de datos de entrenamiento donde cada ejemplo se asocia a una categoría correcta. Además, presentan una etapa de aprendizaje donde durante el entrenamiento, el modelo aprende a asociar patrones respecto a los datos de entrada y sus respectivas etiquetas, ajustando sus parámetros internos para minimizar el error en sus predicciones.

Algunas de las técnicas más comunes de la clasificación supervisada son:

- **Regresión Logística:** Es un modelo clasificatorio que como bien describe su nombre, estima probabilidades utilizando una función logística. Es particularmente útil para la clasificación binaria.
- **Árboles de decisión:** Modelan la decisión de asignar etiquetas a través de una estructura de árbol, donde cada nodo representa una decisión basada en una característica.
- **Bosques Aleatorios:** Modelan la decisión de asignar etiquetas a través de una estructura de árbol, donde cada nodo representa una decisión basada en una característica.
- **Máquinas de Vectores de Soporte:** Como se detallará en el apartado **6.2.1**, las SVM buscan el hiperplano que mejor separa las clases en el espacio de características, siendo eficaces incluso en espacios de alta dimensión.

Las SVM no incorporan ningún concepto actual, estos sistemas componen un conjunto de métodos de aprendizaje supervisado utilizados para la clasificación, regresión y detección de valores atípicos cuya fundamentación se origina a principios de la década de los 60’s. En el contexto del PLN, las SVM se han usado ampliamente en tareas para la clasificación de textos o análisis de emociones debido a su capacidad para manejar espacios de alta dimensionalidad y su eficacia en situaciones donde el número de dimensiones supera al número de muestras.

Las SVM trabajan mapeando datos de entrada a un espacio de características de alta dimensión donde es posible realizar una separación lineal entre las clases. La clave de su éxito radica en la solución de dos grandes problemas principales. El primero (conceptual) es causado por la necesidad de seleccionar un hiperplano óptimo que garantice una buena generalización debido a la existencia de multitud de hiperplanos que podrían no generalizar bien a datos no vistos, aunque puedan separar los datos de entrenamiento.

El segundo problema (técnico) se fundamenta en que tratar computacionalmente con espacios de alta dimensión puede ser algo desafiante. Por ejemplo, construir un polinomio de grado 4 o 5 en un espacio de 200 dimensiones podría requerir trabajar en un espacio de características de mil millones de dimensiones, algo completamente inviable computacionalmente.

Para resolver el problema conceptual se propuso el concepto de hiperplano óptimo para clases separables, que es aquel con el margen máximo entre los vectores entre las dos clases. Se observó que para construir tales hiperplanos óptimos solo se requería considerar un subconjunto de los datos de entrenamiento, los vectores de soporte, que determinan este margen. Este hiperplano óptimo contribuye a una mejor generalización en espacios de alta dimensionalidad.

Para solucionar el problema de la alta dimensionalidad, se introduce el truco del kernel. En lugar de transformar los datos de entrada y luego calcular los productos en el espacio de características, se utilizan funciones de kernel que permiten calcular directamente el producto en el espacio de entrada, seguido de una transformación no lineal de este resultado. Esto permite construir hiperplanos de alta dimensión sin tener que calcular dicha transformación de manera explícita.

Por tanto, el objetivo de un SVM es maximizar el margen entre las clases, lo cual se logra minimizando una función objetivo que incluye un término de regularización (norma cuadrática de los pesos) y una suma de las violaciones del margen.

La formulación de una SVM lineal puede expresarse como el problema de optimización:

$$\min_{w,b,\xi} \frac{1}{2}|w|^2 + C\sum_{i=1}^{n} \xi_i$$

Sujeto a la restricción:

$$y_i(w^T\phi(x_i) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, n$$

Donde:

- w es el vector de pesos.
- b es el término del sesgo.
- ξ_i son las variables de holgura que permiten violaciones del margen.
- C es el es el parámetro de regularización que controla el trade-off entre aumentar el margen y asegurar que los x_i yacen en el lado correcto de él.
- $\phi(x_i)$ es la función de mapeo a un espacio de características de alta dimensión.
- y_i son las etiquetas de clase.

Sin embargo, aunque los sistemas SVM proporcionan buenos resultados y suelen ser el punto de partida de referencia de muchos sistemas clasificadores, al operar en un espacio de características que se deriva directamente de los datos de entrada, hacen que las características se basen en sistemas rudimentarios como BoW o TF-IDF, que no capturan la semántica de la misma manera que los embeddings.

Además, aunque parece que se resuelve el problema de la alta dimensionalidad sin la necesidad de incorporar embedding, a medida que el tamaño del conjunto de datos de entrenamiento aumenta, el costo computacional puede crecer rápidamente ya que la optimización resultada se trata de un problema cuadrático. Esto sumado a la fragilidad del kernel junto a la sensibilidad a los datos ruidosos y atípicos (el hiperplano óptimo depende críticamente de los vectores de soporte más cercanos al margen) hace necesario obtener clasificadores más avanzados que además de mantener la connotación semántica de la oración, incorporen embedding para

hacerlos más eficientes, más inteligentes y capaces de proporcionar clasificaciones multiclase.

6.2.1.2.1. Modelos de clasificación basados en aprendizaje profundo

Las redes neuronales simples, conocidas también como Perceptrones Multicapa (MLP), son la piedra angular del aprendizaje profundo. Compuestas por una serie de capas de neuronas interconectadas, estas redes pueden aprender representaciones complejas de datos al ajustar los pesos sinápticos entre neuronas a través del proceso de entrenamiento.

El concepto de red neuronal no es para nada novedoso pues data de aproximadamente la década de los 50's. Sin embargo, su practicidad recién se ha visto reflejada debido a la falta de potencia de cómputo de la época.

Una red neuronal simple consiste en una capa de entrada, una o más capas ocultas y una capa de salida. La capa de entrada recibe el texto transformado en vectores numéricos (de la misma forma que se representaba con los demás métodos), las capas ocultas aprenden representaciones no lineales y la capa de salida proporciona la clasificación final.

A pesar de la capacidad que otorga para modelar relaciones no lineales, las redes neuronales simples no son óptimas para captar la estructura inherente del texto, como por ejemplo la relevancia relativa de diferentes palabras y la localidad de características relevantes. Además, carecen de la capacidad para procesar secuencias de datos de manera efectiva.

Estas carencias propiciaron no mucho después la aparición de unos modelos más avanzados como son las RNN.

Esta arquitectura, a diferencia de las redes neuronales simples, tienen conexiones cíclicas que permiten procesar secuencias de datos. Esto las hace extremadamente útiles para tareas como el procesamiento del lenguaje natural o la predicción de series temporales, donde es importante tener en cuenta la información previa.

Las RNN pueden representarse con la siguiente fórmula para el estado oculto en el tiempo t, que incorpora la entrada actual x_t y el estado oculto h_{t-1}

$$h_t = f(W \cdot h_{t-1} + U \cdot x_t + b)$$

Donde:

- h_t es el estado oculto en el tiempo, t
- h_{t-1} es el estado oculto en el tiempo t-1
- $W\ y\ U\ son\ matrices\ de\ pesos$
- x_t es la entrada en el tiempo t
- b es el vector de sesgo
- f es una función de activación no lineal, como la tangente hiperbólica o la función de activación sigmoidea

En resumen, una RNN puede interpretar una oración separada por tokens, donde cada palabra procesada previamente por el modelo se introduce en el análisis del nuevo token. Es decir, el output de un proceso constituye el input del siguiente, formando un generador secuencial.

Aunque este proceso parece óptimo, en realidad padece serios problemas. Para empezar, encontramos el problema del gradiente desvaneciente y explosivo. La RNN estándar tiende a olvidar rápidamente los efectos de los elementos de entrada anteriores, especialmente en secuencias largas, suceso conocido como falta de memoria. Otro de los grandes problemas es inherente a su estructura, las RNN deben procesar la información secuencialmente, lo cual no se presta bien a la paralelización.

Debido a estas limitaciones, las Redes Neuronales Convolucionales (CNN) comenzaron a aplicarse masivamente muchos campos de la IA, especialmente en la computación de visión. Este enfoque tiene como objetivo buscar secuencias o patrones en las imágenes de manera eficiente.

Una CNN utiliza filtros llamados capas de convolución para poder extraer características locales (como n-gramas) y capturar patrones en el texto de forma paralela justo antes de las capas ocultas. Estos filtros por naturaleza tienen 2 dimensiones (ancho y alto), sin embargo, para poder extrapolar estas técnicas al procesamiento de texto se utilizan filtros de una sola dimensión.

Para ejemplificar las CNN supongamos un ejemplo con imágenes, imaginemos que tenemos como tarea diferenciar los números del 1-9 a partir de imágenes. Para poder realizarla, una de las opciones más viables es la extracción de características.

Antes de entrar en los conceptos técnicos que hacen posible la extracción de características necesitamos hacer énfasis en el significado de su uso. Supongamos que queremos diferenciar arbitrariamente 2 números como, por ejemplo, un 9 y un 1. Para diferenciarlos podríamos extraer características básicas como comprobar el número de círculos que ambos números presentan (1 y 0 respectivamente) o en caso de querer diferenciar dígitos como el 9 y 0 podríamos comprobar el número de círculos y palos que contienen (1 palo, 1 círculo y 0 palos, 0 círculos respectivamente).

Esta extracción de características se realizaría de la siguiente forma. Primero, se dividiría la imagen en sus 3 canales (en caso de ser a color), posteriormente se aplicaría una convolución por canal con los núcleos o kernel que fueran necesarios, por ejemplo, con los kernel que se especializan en resaltar los ejes en las imágenes. En el caso de usar 16 núcleos diferentes tendríamos un total de 48 convoluciones que posteriormente nos darían un total de 16 resultados a partir de una sola imagen como input. Gracias a esta capa de convolución ya no solo obtenemos simples píxeles, sino que aparecen patrones y ejes. Finalmente, encontramos una capa de agrupación que tiene 2 grandes objetivos, reducir el tamaño de la imagen y finalmente resaltar las características más importantes. Este proceso se repetiría hasta encontrar la precisión deseada.

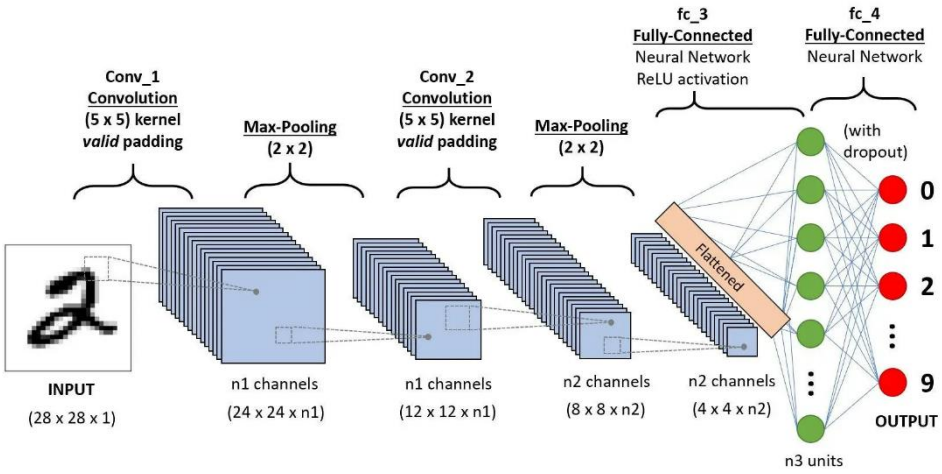


Ilustración 31. Esquema red neuronal convolucional

Esta arquitectura como hemos podido comprobar es excepcionalmente buena encontrando características que permiten reconocer imágenes que incluso presentan variaciones respecto a los inputs originales (imágenes rotadas, con fallos), algo que sería impensable para un modelo MLP. Sin embargo, la extracción de características de texto es bastante más complicada pues hay que tener en cuenta entre otras cosas, el contexto. Es en este punto donde encontramos las primeras costuras de las CNN, estas no son tan efectivas a la hora de dar importancia a la secuencialidad y a las dependencias a largo plazo entre palabras. Las CNN pueden perder el contexto más amplio o las relaciones entre elementos separados por distancias largas en el texto.

Por tanto, se requieren de mecanismos más tradicionales como las RNN, pero intentando solucionar el problema de la preservación del contexto, es aquí donde entra un factor clave para entender las arquitecturas posteriores como la arquitectura Transformer, las máscaras o mecanismos de atención.

Este concepto detalla que la atención se calcula usando 3 conjuntos de vectores para cada palabra (o token) en una secuencia de entrada:

- **Vectores de claves:** Estos vectores representan las “preguntas” en el contexto del mecanismo de atención. Estos vectores se utilizan para identificar los elementos relevantes a los cuales se debe prestar atención.
- **Vectores de valores:** Contienen la “respuesta” o el contenido al que se presta atención. Una vez identificados los elementos relevantes en la secuencia, los vectores de valores proporcionan la información que se utiliza para la salida.
- **Vectores de consulta:** Son los vectores que buscan en los vectores de claves para encontrar coincidencias o relevancias. Estos vectores actúan como la herramienta con la que se sondean los vectores de clave para determinar qué valores son relevantes en el contexto dado.

Como ya se ha detallado anteriormente, la mejor manera de poder manipular la información textual es representándola a través de vectores, concepto que persiste también en estos mecanismos de atención.

Estos mecanismos de atención, especialmente la atención autodirigida (self-attention) en los Transformers, utilizan estos vectores de la siguiente manera para encontrar vínculos entre palabras:

Primero, cada palabra se proyecta en tres espacios diferentes, vectores de consulta, clave y valor a través de matrices de pesos entrenables específicas. Posteriormente, para calcular la atención, el modelo mide la similitud entre cada vector de consulta y todos los vectores de clave. Esto se hace típicamente mediante el producto escalar, seguido de una operación softmax para obtener pesos de atención normalizados. Los pesos de atención determinan cuánto enfoque se debe poner en otros tokens para cada token en particular. Los vectores de valor se ponderan según estos pesos de atención, lo que significa que los tokens relevantes tienen más influencia.

Finalmente, los vectores de valor ponderados se suman para producir la salida del mecanismo de atención para cada token, reflejando un nuevo vector que es una combinación de información relevante de toda la secuencia.

Para entender correctamente el funcionamiento del mecanismo de atención autodirigida se expone la siguiente frase de ejemplo:

“El chico no cogió la bolsa porque estaba cansado”

Primero, se convierte la frase en vectores de consulta (Q), clave (K), y valor (V) a través de matrices de peso entrenables. Imaginemos una simplificación donde solo prestamos atención a la palabra “cansado” y tratamos de entender a quién se refiere.

Para dicha palabra generamos su vector de consulta (Q) y lo comparamos con los vectores de clave (K) de todas las palabras que componen la frase, incluida la misma. La similitud entre “cansado” como consulta y las otras palabras como claves nos da los pesos de atención. Dicha similitud se calcula como el producto escalar seguido de una operación softmax. Estos pesos indican cuánto enfoque poner en cada palabra (valor) cuando tratemos de entender el token “cansado”.

Suponiendo que después del cálculo el modelo asigna mayor peso de atención a “El chico” y menos a “la bolsa”, significaría que el modelo reconoce que cansado está más relacionado con “El chico” en este contexto.

Finalmente, se suman los vectores de valor (V) de todas las palabras, ponderados por los pesos de atención obtenidos para producir un nuevo vector para “cansado” que ahora lleva información agregada que indica que “El gato” es el sujeto cansado.

Con este mecanismo podemos finalmente contextualizar cualquier palabra respecto a otra que se pueda encontrar a cualquier distancia solucionando uno de los principales problemas de las Redes Neuronales Recurrentes, la falta de memoria.

Gracias a la combinación de estos mecanismos de atención con las RNN se han podido obtener nuevos modelos más potentes del PLN, sin embargo, el artículo “attention is all you need” propuso un concepto muy importante para estos modelos, que las RNN son perfectamente prescindibles y únicamente son necesarios los mecanismos de atención.

Este artículo menciona además un nuevo modelo para clasificación de texto para obtener rendimientos aún superiores, el modelo Transformer.

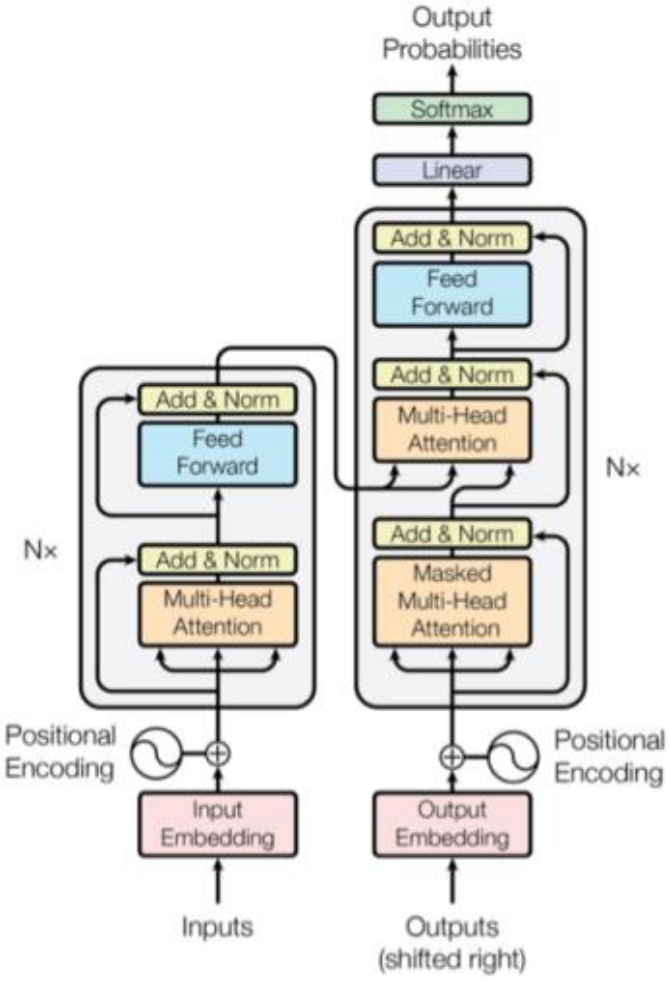


Ilustración 32. Arquitectura Transformer

La arquitectura Transformer como ya se ha detallado, ofrece soluciones para preservar contexto en documentos de gran tamaño gracias al entrenamiento de la frase en su conjunto (usando atención autodirigida) en lugar de procesar secuencialmente varios tokens que la conforman, sin embargo, el deshacerse de las RNN deriva en un nuevo problema, mantener el orden de los tokens. Este orden es fundamental para entender el significado de una frase, pero dado que esta arquitectura procesa todas las palabras en paralelo y no una seguida de la otra, no tiene una manera inherente de captar la posición relativa de las palabras. Para solucionar esto, el modelo incorpora una etapa llamada “Positional Encoding” que inyecta información de cada token en la secuencia.

Positional Encoding incorpora una representación que se suma a los embeddings de entrada para proporcionar información sobre la posición de cada palabra en la secuencia. El objetivo es diferenciar la posición de las palabras y permitir al modelo detectar patrones relacionados con la secuencia entre palabras.

Aunque el Positional Encoding puede implementarse de varias maneras, el método original presentado en “Attention is all you need” utiliza funciones trigonométricas para generar codificadores únicos para cada posición. La codificación posicional se calcula usando funciones seno y coseno con frecuencias que varían según la progresión geométrica en diferentes dimensiones del vector de codificación.

$$PE_{(\mathrm{pos},2i)} = \sin\left(\frac{\mathrm{pos}}{10000^{2i/d_{\mathrm{model}}}}\right)$$

$$PE_{(\mathrm{pos},2i+1)} = \cos\left(\frac{\mathrm{pos}}{10000^{2i/d_{\mathrm{model}}}}\right)$$

Donde:

- pos es la posición de la palabra en la secuencia
- i es la dimensión dentro del vector de embedding
- d_{model} es la dimensión total del embedding

La presentación del modelo Transformer en el artículo “Attention is all you need” marcó un punto de inflexión en el panorama del PLN estableciendo las bases para la creación de sistemas aún más avanzados.

Este es el caso de Bidirectional Encoder Representations from Transformers (BERT). La principal innovación de BERT tal y como indica su nombre es su preentrenamiento bidireccional. Mientras que los Transformers originales utilizan una atención autodirigida que no es inherentemente bidireccional (Los encoders tratan la información de izquierda a derecha para cada traducción), BERT utiliza una arquitectura que se preentrena de manera bidireccional y que además no utiliza los decoder de la arquitectura Transformer. Esto se explica ya que BERT está diseñado especialmente para tareas del PLN donde se deben generar representaciones textualmente ricas, y no para producir secuencias de texto como salida (como en la traducción automática)

Para cada token BERT observa tanto el contexto anterior como el siguiente dentro de una secuencia.

Para preentrenar BERT se utilizan dos tareas del PLN:

- **Modelado de Lenguaje Enmascarado (MLM):** Algunas de las palabras de la entrada se enmascaran aleatoriamente (se ocultan al modelo), y la tarea de BERT durante el preentrenamiento es predecir estas palabras enmascaradas basándose en su contexto. Esto contrasta con el preentrenamiento estándar del Transformer que se basa en predecir la siguiente palabra en una secuencia.
- **Predicción de la Siguiente Oración (NSP):** Se le dan a BERT pares de oraciones como entrada y debe predecir si la siguiente oración es continuación lógica de la primera. Esto ayuda al modelo a comprender las relaciones entre pares de oraciones, muy útil para tareas como la comprensión del texto.

La estructura de un modelo BERT es la siguiente:

- **Entradas del modelo:** BERT toma de entrada una secuencia de tokens que ha sido convertida en embeddings y les suma las siguientes informaciones:
    - **Token embeddings:** Representan los tokens transformados en vectores, cada token se mapea primero a un índice y luego a un embedding aprendible.
    - **Segment embedding**: Diferencian entre dos secuencias diferentes, lo cual es útil para tareas que requieren el entendimiento de la relación entre pares de secuencias
    - **Positional embeddings:** Descritos anteriormente, BERT utiliza también embeddings posicionales para capturar la posición de cada token dentro de la secuencia debido a la pérdida de la noción del orden de los tokens
- **Capas del encoder:** Tiene 2 subcomponentes
    - **Mecanismos de autoatención:** Permite a cada token atender a todos los tokens de la secuencia, lo que permite que sea bidireccional.
    - **Feed-Forward Neural Networks:** Después de la atención auto dirigida, la información procesada se pasa a través de una red feed-forward, la cual es la misma para cada posición, pero con diferentes parámetros aprendidos para cada capa del codificador.

Además de estos componentes, cada subcapa en cada capa del codificador tiene una conexión residual y es seguida por una normalización de capa. Esto significa que la salida de cada subcapa es la suma de sus entradas más la suma de una función de transformación aplicada a estas, seguido por una normalización.

BERT al igual que otros modelos que derivan de él, se presenta en 2 tamaños con diferencias en número de parámetros que implican el número de capas (encoders), tamaño de los encoders así como las cabezas de atención:

- **BERT-Base:** 12 capas de encoders, 12 cabezas de atención por capa, 768 capas ocultas y 168 millones de parámetros (para su versión multilingüe)
- **BERT-Large:** 24 capas de transformadores, 16 cabezas de atención por capa, 1024 capas ocultas y 335 millones de parámetros (para su versión en inglés)

Otro de los modelos más interesantes es BETO, BETO representa un hito en el avance de las técnicas del PLN debido a su construcción teniendo en cuenta nuestro idioma en un ámbito donde la mayoría de modelos monolingües están en inglés. BETO ha sido preentrenado exclusivamente en un vasto corpus de español, abarcando una rica variedad de géneros y estilos textuales. Este modelo al igual que BERT utiliza aprendizaje profundo bidireccional, pero afinado para reflejar las estructuras lingüísticas y usos idiomáticos del español.

Gracias a estar preentrenado en un solo idioma, BETO puede superar en teoría a BERT en tareas que requieren una comprensión contextualizada del español. BETO se puede entrenar en las mismas variantes que BERT:

- **BETO-Base:** 12 capas de encoders, 12 cabezas de atención por capa, 768 capas ocultas y 110 millones de parámetros.
- **BETO-Large:** 24 capas de transformadores, 16 cabezas de atención por capa, 1024 capas ocultas y 340 millones de parámetros.

Existen otros modelos que parten de la base de BERT como RoBERTa, acrónimo de “Robusty optimized BERT approach”. Este es un modelo del PLN diseñado por Facebook AI. RoBERTa es una modificación y optimización del modelo BERT original, diseñado para mejorar su rendimiento a través de un entrenamiento con más datos, más largo y la eliminación de la tarea de predicción de la siguiente oración durante el preentrenamiento.

Las configuraciones de RoBERTa son similares a las de BERT en términos de arquitectura del modelo, pero con algunas diferencias clave en el entrenamiento. Las especificaciones estándar de RoBERTa son:

- **RoBERTa-Base**: 12 capas de encoders, 12 cabezas de atención por capa, 768 capas ocultas y 125 millones de parámetros.
- **RoBERTa-Large:** 24 capas de encoders, 16 cabezas de atención por capa, 1024 capas ocultas y 355 millones de parámetros.

Aunque la estructura de BERT y RoBERTa son similares, RoBERTa tiene algunas diferencias clave:

- **Datos de entrenamiento**: RoBERTa se entrenó con un dataset 10 veces más grande.
- **Entrenamiento más largo con Batch Size más grande:** Esto permite que más conjuntos de ejemplos se procesen en una misma iteración.
- **Se omite la tarea de predicción de la siguiente oración**
- **Dynamic masking:** RoBERTa aplica masking de forma dinámica en el texto de entrada, lo que se traduce en que las palabras a enmascarar para el entrenamiento MLM se seleccionan de manera diferente en cada época.

Continuando con la descripción de los modelos de la familia de Transformers, vamos a profundizar en el modelo XLM-RoBERTa, que representa una evolución de los modelos multilingües como BERT y RoBERTa, específicamente diseñado para mejorar la comprensión de múltiples lenguas de manera simultánea.

Este modelo, desarrollado por Facebook AI, combina las ventajas de RoBERTa con técnicas avanzadas para el aprendizaje multilingüe. XLM-RoBERTa se basa en la arquitectura RoBERTa, pero se entrena en un corpus multilingüe masivo que abarca 100 idiomas diferentes. Esta característica lo hace especialmente robusto para tareas de procesamiento del lenguaje natural (PLN) en diferentes lenguas.

XLM-RoBERTa se puede entrenar en las mismas variantes que BERT:

- **XLM-RoBERTa Base:** 12 capas de encoders, 12 cabezas de atención por capa, 768 dimensiones en las capas ocultas, y aproximadamente 270 millones de parámetros.
- **XLM-RoBERTa Large:** 24 capas de encoders, 16 cabezas de atención por capa, 1024 dimensiones en las capas ocultas, y aproximadamente 550 millones de parámetros.

6.2.2. Métricas de evaluación

Antes de adentrarnos en una descripción detallada de la aplicación de los experimentos realizados, es esencial establecer un marco riguroso para el análisis de los resultados. Este apartado tiene como objetivo proporcionar una comprensión profunda de cómo las métricas seleccionadas nos permiten evaluar y comparar la efectividad de las diversas técnicas de clasificación aplicadas, incluyendo tanto los modelos tradicionales como las actuales arquitecturas basadas en Transformers.

En este contexto se han seleccionado meticulosamente un conjunto de métricas estándar en la industria para no solo mostrar la eficacia general de los modelos para clasificar el lenguaje ofensivo, sino también para revelar sutilezas en su desempeño como la capacidad para equilibrar la sensibilidad y la especificidad frente a clases desbalanceadas.

Una de las métricas más influyentes es la matriz de confusión que muestra características importantes como:

- **Precisión:** La precisión indica la proporción de predicciones correctas para una clase determinada con respecto a todas las predicciones que el modelo hizo para esa clase. Es la proporción de verdaderos positivos frente a la suma de verdaderos positivos y falsos positivos.

$$\text{Accuracy} = \frac{TP}{TP + FP}$$

  Donde:
  - TP es la suma de los verdaderos positivos de una determinada clase.
  - FP es la suma de los falsos positivos de una determinada clase.

- **Recall:** El recall o sensibilidad mide la proporción de los verdaderos positivos identificados correctamente por el modelo con respecto al número total de casos reales de esa clase. Indica cuán bueno es el modelo para encontrar todas las instancias relevantes en una determinada categoría.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Donde:

- FN es la suma de los falsos negativos de una determinada clase.

- **Accuracy:** La exactitud (accuracy) es la proporción de predicciones correctas (verdaderos positivos y verdaderos negativos) respecto al total de predicciones realizadas.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Donde:

- TN es la suma de los verdaderos negativos de una determinada clase.

- **F1-Score**: Es una media armónica de la precisión y el recall. Esta es una buena variable para entender el balance entre la precisión y el recall de un modelo, especialmente cuando las clases están desequilibradas.

$$\text{F1-Score} = 2 \times \frac{\text{Precisión} \times \text{Recall}}{\text{Precisión} + \text{Recall}}$$

- **Macro Avg**: El promedio macro de una métrica (precisión, recall…) se calcula tomando el promedio aritmético de esa métrica para cada clase, sin tener en cuenta el soporte para cada clase. Por tanto, todas las clases contribuyen por igual al promedio final.

$$\text{Macro Average X} = \frac{1}{N}\sum_{i=1}^{N} \text{F1-Score}_i$$

Donde:

- X es la métrica a medir.
- N es el número total de clases.

- **Weighted Avg:** Esta variable indica el promedio de una métrica para cada clase, pero a diferencia del Macro AVg, tiene en cuenta el soporte para cada clase. Por tanto, las clases con más instancias incluyen más en el promedio final que las clases minoritarias.

$$\text{Weighted Average X} = \sum_{i=1}^{N} w_i \times \text{F1-Score}_i$$

Donde:

- X es la métrica a medir.
- w_i es el peso o soporte de cada clase.

- **Support**: Indica el número de ocurrencias de cada clase en el conjunto de datos. Es una representación de la distribución de las clases en el corpus de prueba.
- **Micro Avg:** En problemas de clasificación multiclase, el promedio micro considera la suma de todos los verdaderos positivos y falsos positivos a través de todas las clases, tratando todas las instancias igualmente.

$$\text{Micro Average Precision} = \frac{\sum_{i=1}^{N} TP_i}{\sum_{i=1}^{N} (TP_i + FP_i)}$$

$$\text{Micro Average Recall} = \frac{\sum_{i=1}^{N} TP_i}{\sum_{i=1}^{N} (TP_i + FN_i)}$$

Donde:

- $\sum TP$ es la suma de los verdaderos positivos de todas las clases.
- $\sum FP$ es la suma de los falsos positivos de todas las clases.
- $\sum FN$ es la suma de los falsos negativos de todas las clases.

En una matriz de confusión cada fila representa las instancias de una clase real mientras que cada columna representa las instancias de una clase predicha.

Otra de las métricas que se han incorporado es el AUC-ROC para mostrar la capacidad de clasificación del modelo. AUC significa “Área bajo la curva” y ROC es la “curva de características operativas del receptor”. Esta métrica se usa sobre todo en problemas de clasificación binaria, sin embargo, puede también extrapolarse a estos sistemas multiclase.

AUC ROC emplea dos conceptos, ROC es una curva de probabilidad que marca el ratio de verdaderos positivos (TPR) frente al ratio de falsos positivos (FPR) para cada clase individual. El área bajo cada curva ROC (AUC) es una medida de la capacidad del modelo para distinguir entre la clase específica y las demás clases. Se interpreta como la capacidad de que el modelo clasifique mejor que el azar, siendo 0.5 este y 1 un modelo perfecto.

6.2.3. Experimentos realizados

Tras una extensa revisión teórica del funcionamiento de las técnicas de representación actual y de los avances en modelos de clasificación supervisada, nos embarcamos en la fase experimental del trabajo. Este segmento es crucial para depositar el conocimiento teórico en el aplicado. En este apartado se detallan las diversas técnicas que se han utilizado para poder crear un modelo lo más óptimo posible capaz de clasificar el lenguaje ofensivo en varias categorías.

6.2.3.1. Sistemas de clasificación basados en SVM

Las máquinas de vectores de soporte (SVM) tal y como se han descrito en el apartado **5.2.1** contienen múltiples diferencias respecto a modelos más avanzados basados en aprendizaje profundo, concretamente la falta de contexto. Sin embargo, estas constituyen un muy buen punto de referencia para construir modelos más avanzados gracias a su simplicidad y a un rendimiento razonable en numerosas tareas de clasificación textual.

Antes de entrenar el modelo basado en vectores de soporte es necesario realizar una etapa de preprocesamiento. Esta etapa es crucial debido a la falta de embedding de estos modelos que derivan en una alta dimensionalidad y por tanto en la necesidad de aplicar técnicas para procesar la información textual eficientemente.

Primero, tiene lugar una fase de tokenización, proceso que convierte el texto en unidades más pequeñas. Ayuda a identificar las palabras que serán útiles para el análisis y la clasificación.

La elección del tokenizador es un factor clave para elaborar un buen clasificador y por tanto esta deberá estar vinculada al tipo de tarea que se busque. En este caso uno de los tokenizadores que mejor resultado ofrecen es WordPuncTokenizer que permite dividir la cadena basándose en los espacios en blanco y los signos de puntuación, los cuales pueden ser un factor clave en la identificación de lenguaje ofensivo.

También es necesario eliminar todas las stopwords presentes, las stopwords son palabras que se filtran antes o después del procesamiento de datos. Son palabras comunes que generalmente no contribuyen al significado de una frase, por lo menos para estas tareas básicas de procesamiento. Algunas de las stopwords más básicas en castellano son “el”, “la”, “una”, etcétera…

Otro paso clave para el preprocesamiento textual es el stemming. Este proceso consiste en reducir las palabras a su raíz para disminuir la variabilidad de las palabras, agrupando diferentes formas de una palabra como una única característica mejorando por tanto la generalización del modelo y mitigando el riesgo de sobreajuste.

```
1. # Preparación del entorno de NLTK
2. import nltk
3. nltk.download('stopwords')
4. nltk.download('punkt')
```

Código 3. Preparación del entorno de NLTK

```
1. # Inicialización de herramientas de procesamiento de texto
2. spanish_stops = set(stopwords.words('spanish'))
3. stemmer = SnowballStemmer("spanish")
4. tk = WordPunctTokenizer()
```

Código 4. Inicialización de herramientas de procesamiento de texto

Una vez definidos los tokenizadores y el stemmer, primero se ponen todos los caracteres de cada frase en minúscula para posteriormente tokenizar sus palabras y eliminar stopwords así como signos de puntuación. Finalmente, en esta etapa de preprocesamiento se almacenan las frases resultantes eliminando también caracteres vacíos.

```
 1. # Mapeo de los valores numéricos a las categorías textuales
 2. label_mapping = {0: "OFP", 1: "OFG", 2: "NO", 3: "NOE"}
 3.
 4. with open('./Desktop/TFG2024/datasets/offendes/offendes_train.tsv', 'r', encoding='utf-8',
errors='replace') as tsv_file:
 5.     tsv_reader = csv.reader(tsv_file, delimiter='\t')
 6.     next(tsv_reader)  # Saltar la primera fila (headers)
 7.     for row in tsv_reader:
 8.         linea = row[1].lower()
 9.         tokens = tk.tokenize(linea)
10.         filtered = [stemmer.stem(word) for word in tokens if word not in spanish_stops and
not all(char in string.punctuation for char in word)]
11.         if filtered:
12.             frase = ' '.join(filtered)
13.             arrayFrases.append(frase)
14.             tags.append(int(row[5]))
```

Código 5. Procesamiento general

A continuación, tiene lugar la vectorización del texto preprocesado en una representación numérica mediante TF-IDF. El entrenamiento del modelo se puede realizar mediante varios tipos de kernel, sin embargo, se ha optado por usar el predeterminado, el kernel RBF (Radial Basis Function). Este Kernel es una opción muy popular porque puede manejar casos en los que la relación de clases no es lineal.

El modelo precisa únicamente de dos variables para su entrenamiento, un array de etiquetas, así como el conjunto de frases preprocesadas.

```
1. # Vectorización con TF-IDF
2. vectorizer = TfidfVectorizer()
3. X = vectorizer.fit_transform(arrayFrases)
4.
5. # Entrenamiento del modelo SVM
6. model = svm.SVC()
7. model.fit(X, tags)
```

Código 6. Entrenamiento del modelo

6.2.3.1.1. Análisis de resultados

Etiqueta\Categoría	Precission	Recall	F1-Score	Support
OFP	0.8227	0.3808	0.5206	2340
OFG	0.0000	0.0000	0.0000	211
NO	0.8361	0.9911	0.9070	9651
NOE	0.7839	0.6047	0.6828	1404
Micro avg	0.8309	0.8309	0.8309	13606
Macro avg	0.6107	0.4941	0.5276	13606
Weighted avg	0.8154	0.8309	0.8034	13606

Tabla 32. Métricas SVM

Actual\Predicción	OFP	OFG	NO	NOE
OFP	891	0	1261	188
OFG	53	0	144	14
NO	54	0	9565	32
NOE	85	0	470	849

Tabla 33. Matriz confusión SVM

Tal y como puede apreciarse en la imagen, el modelo SVM presenta unos resultados a simple vista buenos centrándonos sobre todo en la micro precisión o accurary (0.8309). Sin embargo, encontramos una macro avg precision y en especial un macro avg recall bajos. Esto se debe seguramente a la incapacidad del modelo para clasificar una frase en la categoría OFG, que es casualmente aquella que tiene con diferencia, la menor representación entre todas las clases existentes.

Una de las posibles soluciones aplicadas ha sido aplicar sobremuestreo mediante SMOTE (Synthetic Minority Over-sampling Technique) al conjunto de entrenamiento para poder manejar el desbalance de clases. Esta técnica genera ejemplos sintéticos para la clase minoritaria basándose en los que ya existen, con el fin de alcanzar un equilibrio entre las clases.

Para utilizar SMOTE únicamente se modifican las líneas de código que afectan al entrenamiento de las SVM:

```
1. # Aplicar SMOTE para sobremuestreo
2. smote = SMOTE(random_state=42)
3. X_res, y_res = smote.fit_resample(X_train, y_train)
```

Código 7. Sobre muestreo con SMOTE

Una vez aplicado el sobremuestreo, los resultados son los siguientes:

Etiqueta\Categoría	Precission	Recall	F1-Score	Support
OFP	0.7738	0.4064	0.5329	2340
OFG	0.0000	0.0000	0.0000	211
NO	0.8341	0.9901	0.9054	9651
NOE	0.8233	0.5377	0.6506	1404
Micro avg	0.8276	0.8276	0.8276	13606
Macro avg	0.6078	0.4836	0.5222	13606
Weighted avg	0.8097	0.8276	0.8010	13606

Tabla 34. Métricas sobre muestreo

Actual\Predicción	OFP	OFG	NO	NOE
OFP	951	2	1264	123
OFG	53	0	145	13
NO	70	0	9555	26
NOE	155	3	491	755

Tabla 35. Matriz confusión sobre muestreo

Se puede apreciar que, si bien es cierto, gracias al sobre muestreo el modelo es ahora capaz de identificar algunas frases en la categoría “OFG”, todos estos casos son falsos positivos.

6.2.3.2. Sistemas de clasificación basados en Transformers

Los sistemas basados en Transformers han marcado una revolución en el ámbito del NLP. En este apartado se detalla el proceso analítico seguido para evaluar cómo modelos basados en Transformer como BERT, RoBERTa, XML-RoBERTa, han sido especializados para la clasificación de lenguaje ofensivo, siguiendo varias estrategias, entre ellas una de ajuste fino.

En este apartado se aplica el concepto de “transfer learning” que permite adaptar conocimientos generales del lenguaje a tareas específicas como la de este trabajo.

6.2.3.2.1. Modelos BERT

El entrenamiento de un modelo BERT para la clasificación de lenguaje ofensivo permite obtener resultados muy buenos gracias a la atención autodirigida bidireccional.

La aplicación de este modelo lleva implícita la ejecución de las siguientes subtareas:

- **Preparación del conjunto de Datos:** Primero, se cargan un conjunto de datos desde un archivo TSV en un DataFrame de Pandas. Luego, se extraen las columnas de comentarios y etiquetas necesarias para el entrenamiento.

```
# 1. Preparar el conjunto de datos

comments = []

# Cargar el archivo TSV como un DataFrame de pandas
df = pd.read_csv('./Desktop/TFG2024/datasets/offendes/offendes_train.tsv', delimiter='\t', encoding='utf-8')
labels = df['label'].tolist()

with open('./Desktop/TFG2024/datasets/offendes/offendes_train.tsv', 'r', encoding='utf-8', errors='replace') as tsv_file:
    tsv_reader = csv.reader(tsv_file, delimiter='\t')
    next(tsv_reader)
    for row in tsv_reader:
        comments.append(row[1])
```

Código 8. Preparación del conjunto de datos

- **Carga del tokenizador y el Modelo:** El tokenizador es uno de los elementos clave más aún si tenemos en cuenta que BERT es un modelo preentrenado originalmente con datos en inglés. Debido a que la finalidad de este proyecto es construir un clasificador tomando de base un corpus de datos en español, se precisa de un tokenizador que se amolde a las características lingüísticas de este idioma. En este caso se ha optado por usar un tokenizador por defecto de la librería BertTokenizer, concretamente el “bert-base-multilingual-cased”.

```
# 2. Cargar el tokenizador y el modelo
tokenizer = BertTokenizer.from_pretrained('bert-base-multilingual-cased')
```

Código 9. Carga del tokenizador y del modelo

- **Tokenización:** La tokenización divide el texto en palabras, subpalabras o índices a partir de un tokenizador WordPiece. Esta tokenización se aplica a todos los comentarios para obtener los inputs_ids y las attention_masks. Estas attention masks permiten al modelo saber qué tokens deben considerarse y cuáles son de relleno.

```python
# Función de tokenización y creación del dataset
def encode_comments(comments, labels):
    input_ids = []
    attention_masks = []

    for comment in comments:
        encoded_dict = tokenizer.encode_plus(
            comment,
            add_special_tokens=True,  # Agrega '[CLS]' y '[SEP]'
            max_length=64,           # Longitud máxima para padear/truncar
            padding='max_length',  # Parámetro de padding
            return_attention_mask=True,  # Construye la attention mask
            return_tensors='pt',     # Retorna tensores de PyTorch
            truncation=True
        )

        input_ids.append(encoded_dict['input_ids'])
        attention_masks.append(encoded_dict['attention_mask'])

    input_ids = torch.cat(input_ids, dim=0)
    attention_masks = torch.cat(attention_masks, dim=0)
    labels = torch.tensor(labels)

    return input_ids, attention_masks, labels
```

Código 10. Tokenización y creación del dataset

- **Creación del objeto Dataset:** Esta clase es vital para almacenar los inputs_ids así como las attention_masks. Este Dataset no es más que una variación de la clase propia de PyTorch Dataset. Esta clase personalizada debe de ir acompañada de los métodos __len__ y __getitem__ que son necesarios para que el objeto Dataset pueda ser utilizado por un DataLoader de PyTorch.

```python
# 3. Crear el objeto Dataset
class CommentsDataset(Dataset):
    def __init__(self, input_ids, attention_masks, labels):
        self.input_ids = input_ids
        self.attention_masks = attention_masks
        self.labels = labels

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

    def __getitem__(self, idx):
        return {
            'input_ids': self.input_ids[idx],
            'attention_mask': self.attention_masks[idx],
            'labels': self.labels[idx]
        }
```

Código 11. Creación del objeto dataset

- **División del conjunto de datos:** Se utiliza train_test_split de sklearn para dividir el conjunto de datos en conjuntos de entrenamiento y validación.

```
# Dividir los datos en conjuntos de entrenamiento y validación
from sklearn.model_selection import train_test_split

# Primero, dividir input_ids y labels
train_inputs, validation_inputs, train_labels, validation_labels = train_test_split(
    input_ids, labels, test_size=0.1, random_state=42)

# Dividir attention_masks de la misma manera, usando los mismos índices
train_masks, validation_masks, _, _ = train_test_split(
    attention_masks, labels, test_size=0.1, random_state=42)

train_dataset = CommentsDataset(train_inputs, train_masks, train_labels)
val_dataset = CommentsDataset(validation_inputs, validation_masks, validation_labels)
```

Código 12. División del conjunto de datos

- **Definición de argumentos de entrenamiento:** En esta parte se definen los TrainingArguments para configurar los parámetros de entrenamiento como el directorio de salida, el número de épocas, los tamaños del batch, pasos de calentamiento, decaimiento de peso y la configuración de los registros. Esta parte es fundamental para ajustar la precisión del modelo aumentando su precisión a través de fine-tuning

```
# 4. Definir los argumentos de entrenamiento
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
)
```

Código 13. Definición de los argumentos de entrenamiento

- **Entrenamiento y Evaluación del modelo:** Para poder entrenar el modelo BERT y transferir su conocimiento al corpus de datos deseado es necesario indicar el número de etiquetas que se desean clasificar, así como la versión utilizada del modelo (en este caso bert-base-multilingual-cased). También encontramos dos variables que indican si se desea obtener los pesos de atención de cada capa de atención del modelo (output_attentions) así como si se deben devolver los estados ocultos de las capas del modelo, es decir, las representaciones intermedias (output_hidden_states).

```
# 5. Entrenar y evaluar el modelo
model = BertForSequenceClassification.from_pretrained(
    "bert-base-multilingual-cased",
    num_labels=4,
    output_attentions=False,
    output_hidden_states=False,
)
```

Código 14. Entrenamiento del modelo

- **Guardar el modelo entrenado:** Después de su entrenamiento, se guarda el modelo para su posterior análisis de resultados

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset
)

trainer.train()

# 6. Guardar el modelo entrenado
model.save_pretrained('./my_bert_modelOffendEsV1')
tokenizer.save_pretrained('./my_bert_modelOffendEsV1')
```

Código 15. Guardado del modelo

6.2.3.2.1.1. Análisis de resultados

Una vez entrenado el modelo y comparadas las predicciones con el conjunto de etiquetas verdaderas obtenemos estos resultados:

Etiqueta\Categoría	Precission	Recall	F1-Score	Support
OFP	0.7808	0.5449	0.6418	2340
OFG	0.5042	0.2844	0.3636	211
NO	0.8878	0.9703	0.9272	9651
NOE	0.7383	0.6873	0.7119	1404
Micro avg	0.8573	0.8573	0.8573	13606
Macro avg	0.7278	0.6217	0.6612	13606
Weighted avg	0.8480	0.8573	0.8472	13606

Tabla 36. Métricas BERT

Actual\Predicción	OFP	OFG	NO	NOE
OFP	1275	24	787	254
OFG	37	60	93	21
NO	199	21	9364	67
NOE	122	14	303	965

Tabla 37. Matriz confusión BERT

En esta imagen se aprecia cómo a diferencia de modelos más simple como las máquinas de vectores de soporte, BERT consigue paliar la escasez de ejemplos de la clase minoritaria OFG obteniendo una precisión micro de 0.8573 y un P-macro avg de 0.7278, un R-macro avg de 0.6217 y un f1-macro avg de 0.6612. Estos datos demuestran que si bien es cierto existe una mejora abismal respecto a las SVM, el recall macro sigue siendo bajo, lo cual indica que las clases minoritarias como OFG aún no se reconocen correctamente.

Los resultados del análisis usando las métricas AUC-ROC son los siguientes para clasificación multiclase y binaria

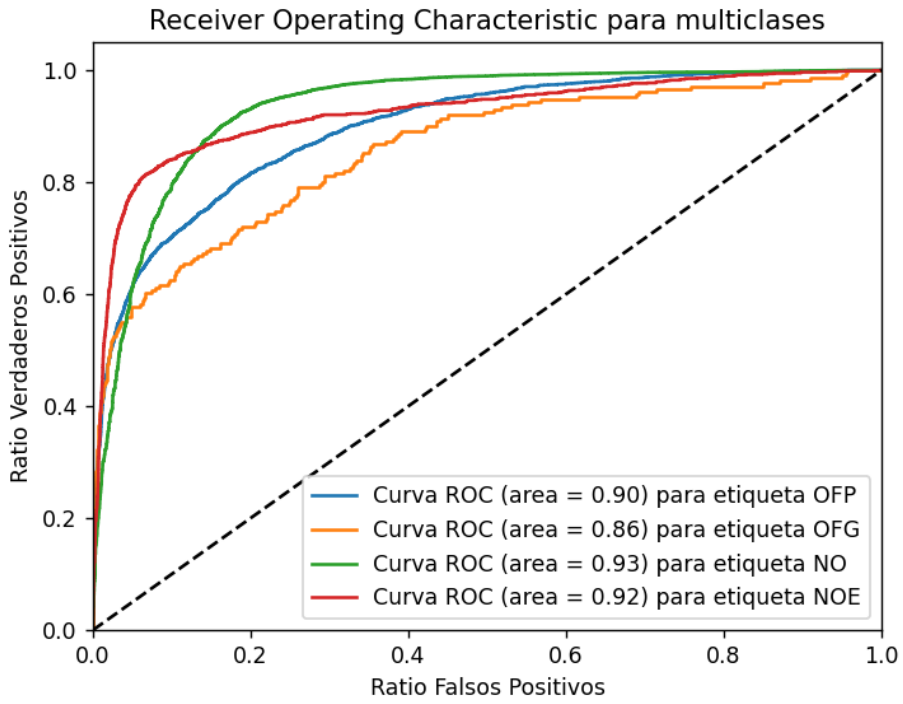


Ilustración 33. AUC-ROC BERT multiclase

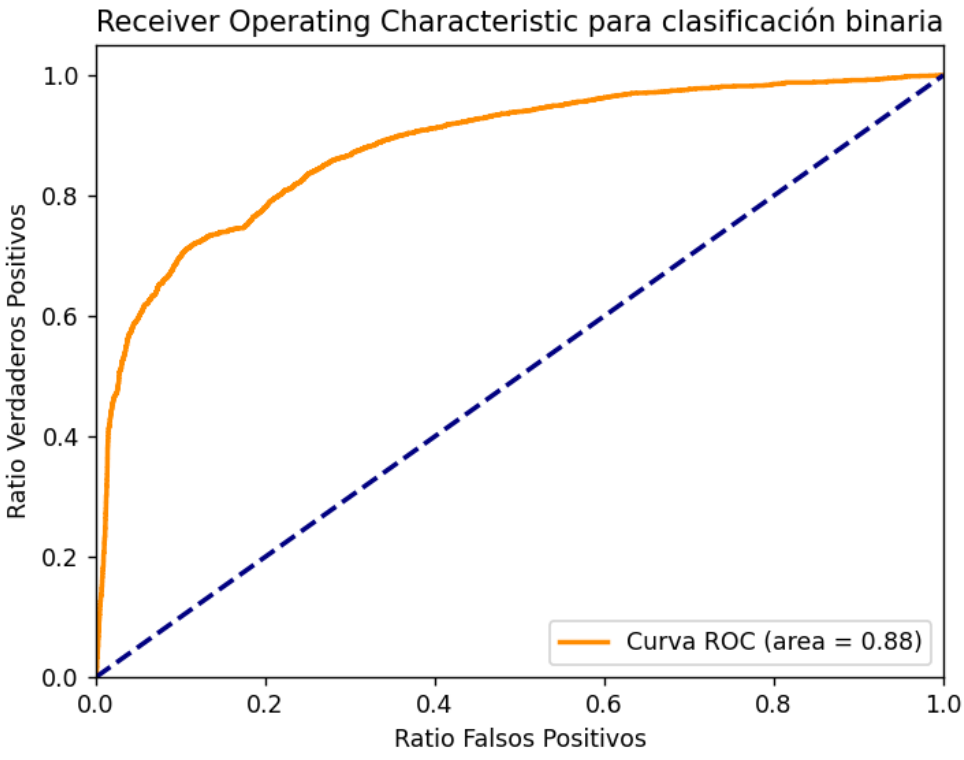


Ilustración 34. AUC-ROC BERT binaria

6.2.3.2.1.2. Técnicas de post procesamiento aplicadas

Existen diversas formas de mejorar la precisión de un modelo. Una de las más obvias es aumentar el número de epochs o épocas de aprendizaje, proceso que es eficaz, pero a la vez costoso computacionalmente ya que implica volver a entrenar el modelo en un proceso aún más largo.

Debido a esto, en este trabajo se ha optado primero por probar qué técnicas se pueden aplicar tras el resultado del clasificador (tanto SVM, BERT…) que permitan aumentar la precisión del modelo sin perjudicar en exceso al rendimiento de este.

Una de las soluciones más intuitivas es tratar de filtrar los falsos positivos del lenguaje No Ofensivo (NO). Un modelo, por muy preciso que sea siempre va a cometer errores, sin embargo, gracias a la comprensión contextual del castellano, se pueden elaborar lexicones del idioma, así como diversas técnicas que permitan identificar lenguaje soez en estas.

En la siguiente imagen se muestra un diagrama de flujo que detalla la creación de un sistema que aplica las siguientes técnicas sobre una frase que ha sido predecida a priori como NO

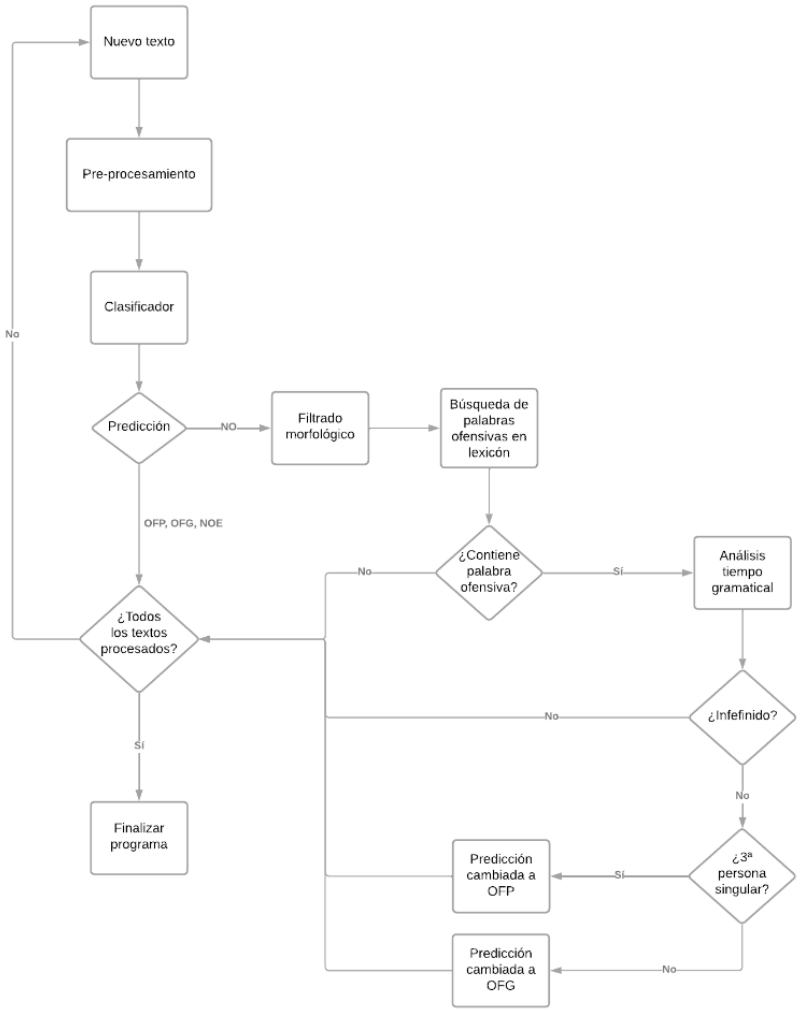


Ilustración 35. Técnicas de post procesamiento

Primero, una vez el clasificador en cuestión muestra un resultado, si la predicción es No Ofensiva se realiza un filtrado morfológico para obtener los tokens más importantes (adjetivos, sustantivos e interjecciones) y comprobar si contienen algún insulto que se encuentre dentro de un lexicón de insultos en español. En caso de no incluirlos, se mantiene el resultado previo y si los incluye, se comprueba el tiempo gramatical de la frase para clasificar el contenido como OFG (Ofensivo hacia un grupo) u OPF (Ofensivo hacia una sola persona).

```python
def postProcesamiento(linea,prediction):
    etiqueta_a_indice = {"OFP": 0, "OFG": 1, "NO": 2, "NOE": 3}

    if prediction == "NO":
        doc = nlp(linea)
        adj_list=[stemmer.stem(token.text) for token in doc if token.text not in spanish_stops and token.text not in string.punctuation if token.pos_ == "ADJ" or token.pos_=="NOUN" or token.pos_=="INTJ"]

        if any(word in adj_list for word in insultos_stem):
            tiempoVerbal=comprobadorTiempoGramatical(linea)
            if(tiempoVerbal=="3rdSing"):
                prediction="OFP"
            elif(tiempoVerbal=="3rdPlur"):
                prediction="OFG"

    return etiqueta_a_indice[prediction]
```

Código 16. Función general postprocesamiento

```python
def comprobadorTiempoGramatical(frase):
    doc = nlp(frase.lower())
    # Inicializa un diccionario para contar las ocurrencias de cada combinación de persona y número
    combinaciones = {
        '1stSing': 0, '2ndSing': 0, '3rdSing': 0,
        '1stPlur': 0, '2ndPlur': 0, '3rdPlur': 0
    }

    for token in doc:
        if token.pos_ == "VERB":
            persona = token.morph.get("Person")
            numero = token.morph.get("Number")
            # Comprueba si el token tiene atributos de persona y número
            if persona and numero:
                key = f'{persona[0]}{"st" if persona[0] == "1" else "nd" if persona[0] == "2" else "rd"}{"Sing" if "Sing" in numero else "Plur"}'
                combinaciones[key] += 1

    # Devuelve la combinación más frecuente
    if all(val == 0 for val in combinaciones.values()):
        return 'Indefinido'
    else:
        return max(combinaciones, key=combinaciones.get)
```

Código 17. Función comprobadorTiempoGramatical

Tras haber aplicado este postprocesado obtenemos los siguientes resultados:

Etiqueta/Categoría	Precission	Recall	F1-Score	Support
OFP	0.6413	0.6060	0.6232	2340
OFG	0.3239	0.3270	0.3255	211
NO	0.9054	0.9264	0.9158	9651
NOE	0.7383	0.6873	0.7119	1404
Micro avg	0.8374	0.8374	0.8374	13606
Macro avg	0.6523	0.6367	0.6441	13606
Weighted avg	0.8337	0.8374	0.8353	13606

Tabla 38. Resultados postprocesamiento

Actual\Predicción	OFP	OFG	NO	NOE
OFP	1418	35	633	254
OFG	56	69	65	21
NO	553	90	8941	67
NOE	184	19	236	965

Tabla 39. Matriz confusión postprocesamiento

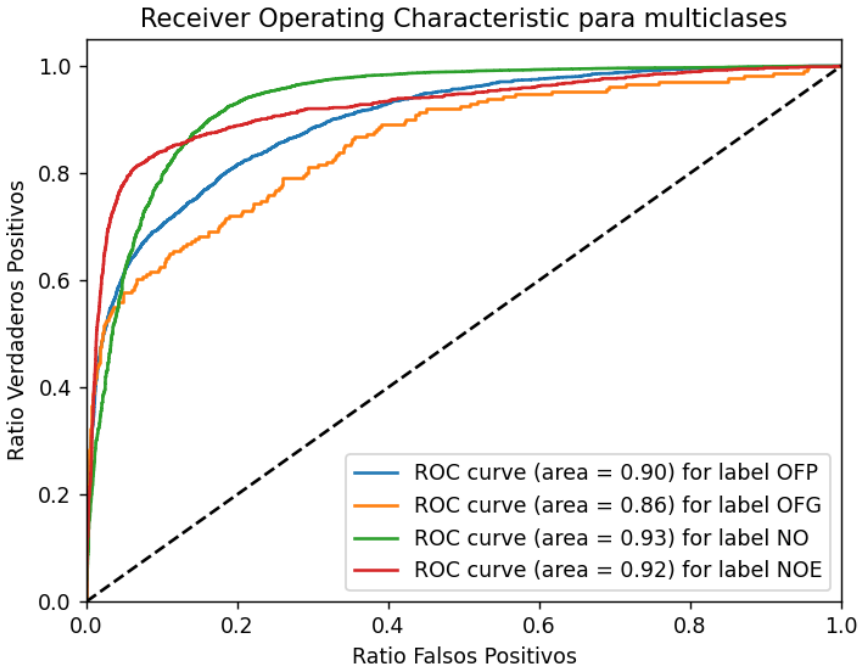


Ilustración 36. AUC-ROC post procesamiento multiclases

Los resultados obtenidos son buenos, de hecho, podemos observar como el recall de casi todas las clases ha aumentado, sin embargo, esto no compensa la pérdida de precisión provocada por el incremento de falsos positivos.

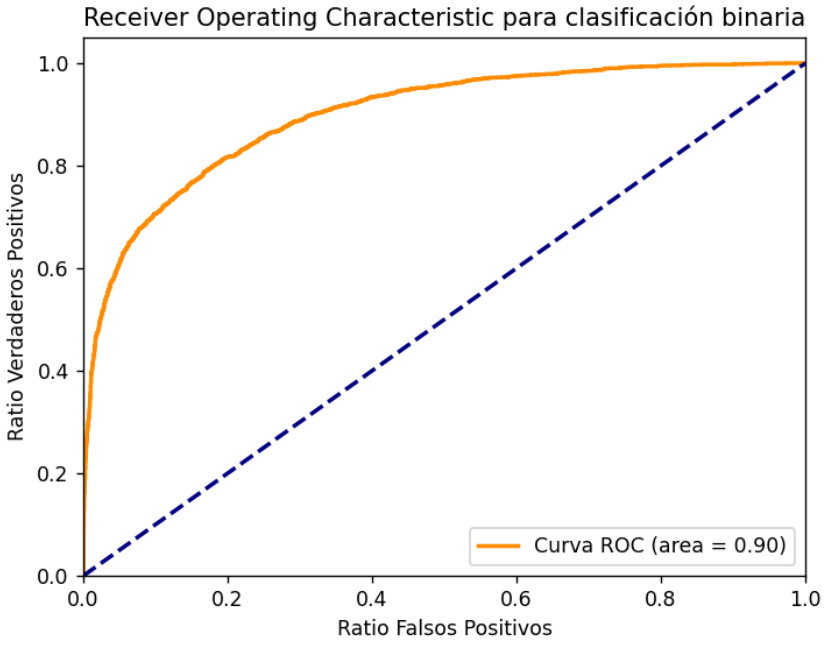


Ilustración 37. AUC-ROC post procesamiento binaria

6.2.3.2.1.3. Ajuste de hiperparámetros

Debido a la incapacidad de los sistemas propios integrados para mejorar la comprensión textual como las etapas de procesamiento, es necesario incorporar técnicas más sofisticadas que permitan entrenar una red profunda que haga más énfasis en el corpus.

En este contexto es donde surge el ajuste de hiperparámetros, el ajuste de hiperparámetros no cambia la arquitectura subyacente del modelo (como el número de capas o la estructura de conexiones entre las neuronas), sino que optimiza los parámetros de control que gobiernan cómo el modelo aprende durante su entrenamiento. Los valores por defecto del constructor TrainingArguments son:

Argumento	Valor por defecto	Descripción
Output_dir	None	El directorio donde se guardarán los resultados del entrenamiento.
Overwrite_output_dir	False	Si True, sobrescribe el contenido del directorio de salida si ya existe.
Num_train_epochs	3.0	Número de épocas de entrenamiento.
Output_dir	None	El directorio donde se guardarán los resultados del entrenamiento.
Overwrite_output_dir	False	Si True, sobrescribe el contenido del directorio de salida si ya existe.
Per_device_train_batch_size	8	Tamaño del lote por dispositivo durante el entrenamiento.
Per_device_eval_batch_size	8	Tamaño del lote por dispositivo durante la evaluación.
Warmup_steps	0	Número de pasos de calentamiento para el scheduler del ratio de aprendizaje.
Weight_decay	0.0	Decaimiento de peso para la regularización.
Logging_dir	None	Directorio donde se guardarán los logs.

Evaluation_strategy	No	La estrategia para evaluar durante el entrenamiento (No, Steps, Epoch).
Save_strategy	Epoch	La estrategia para guardar los checkpoints (No, Steps, Epoch).
Seed	42	Semilla para la generación de números aleatorios.
Learning_rate	5^{-5}	Velocidad de aprendizaje

Tabla 40. Ajuste de hiperparámetros

Uno de los cambios que se han aplicado para hacer un ajuste del modelo es modificar las epochs en el contexto del entrenamiento, es decir, se han modificado el número de iteraciones por el cual el modelo ve y aprende de cada muestra en el corpus de entrenamiento. El número de epochs ha variado concretamente del valor por defecto (3) a 10 resultando por tanto en un proceso de entrenamiento mucho más lento y costoso computacionalmente.

Etiqueta/Categoría	Precission	Recall	F1-Score	Support
OFP	0.7518	0.5697	0.6482	2340
OFG	0.3828	0.3791	0.3810	211
NO	0.8838	0.9576	0.9192	9651
NOE	0.7301	0.6068	0.6628	1404
Micro avg	0.8457	0.8457	0.8457	13606
Macro avg	0.6871	0.6283	0.6528	13606
Weighted avg	0.8375	0.8457	0.8378	13606

Tabla 41. Métricas ajuste hiperparámetros

Actual\Predicción	OFP	OFG	NO	NOE
OFP	1333	46	773	188
OFG	33	80	84	14
NO	244	52	9242	113
NOE	163	31	358	852

Tabla 42. Matriz confusión ajuste hiperparámetros

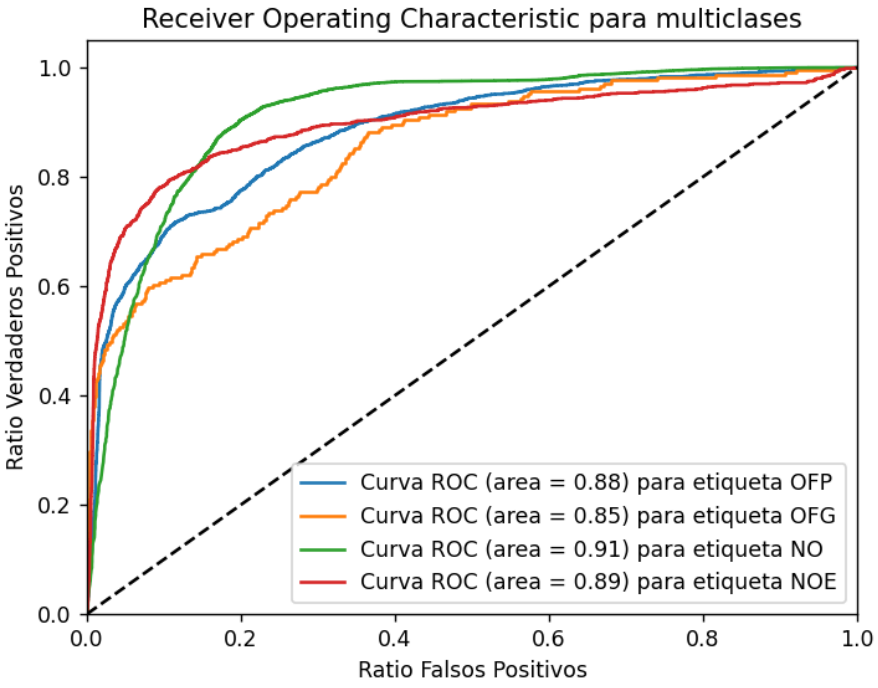


Ilustración 38. AUC-ROC hiperparámetros multiclase

En este ejemplo observamos que a pesar de que ahora el modelo ha aumentado la media armónica f1 score tanto para las clases OFP y OFG, el f1-score así como la precisión y recall de NO y NOE han bajado, otorgando un menor micro avg y macro avg.

Por tanto, ante un posible problema de sobreajuste, se ha probado otro ajuste de hiperparámetros reduciendo el número de epochs a 6 pero disminuyendo a 3e-5 la velocidad de aprendizaje.

Etiqueta/Categoría	Precission	Recall	F1-Score	Support
OFP	0.7846	0.5714	0.6612	2340
OFG	0.5120	0.3033	0.3810	211
NO	0.8835	0.9620	0.9211	9651
NOE	0.7352	0.6645	0.6981	1404
Micro avg	0.8539	0.8539	0.8539	13606
Macro avg	0.7288	0.6253	0.6653	13606
Weighted avg	0.8454	0.8539	0.8450	13606

Tabla 43. Métricas ajuste 2 hiperparámetros

Actual\Predicción	OFP	OFG	NO	NOE
OFP	1337	22	798	183
OFG	18	64	95	34
NO	218	30	9284	119
NOE	131	9	331	933

Tabla 44. Matriz confusión ajuste 2 hiperparámetros

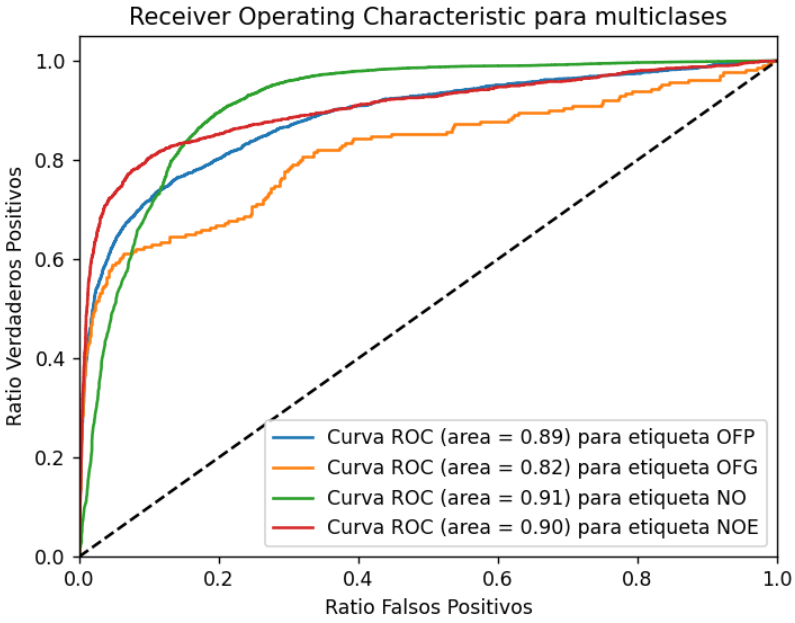


Ilustración 39. AUC-ROC ajuste 2 hiperparámetros multiclase

Tal y como puede apreciarse, las variaciones respecto al sistema BERT de base con 3 epochs son mínimas, de hecho, los resultados lejos de mejorar llegan a empeorar en algunas circunstancias como la clasificación para la etiqueta OFG.

6.2.3.2.2. Modelos BETO

En el apartado anterior se han detallado las mejoras aplicadas sobre los modelos BERT basadas en post procesamiento y ajuste de hiperparámetros, debido a la incapacidad de mejora se han estudiado otros modelos que se basen en la arquitectura transformer capaz de procesar textos en español como los modelos BETO.

Los resultados siguientes son del modelo BETO en su versión base entrenado con 3 epochs:

Etiqueta/Categoría	Precission	Recall	F1-Score	Support
OFP	0.6309	0.3162	0.4213	2340
OFG	0.0000	0.0000	0.0000	211
NO	0.8407	0.9742	0.9026	9651
NOE	0.6696	0.5962	0.6307	1404
Micro avg	0.8069	0.8069	0.8069	13606
Macro avg	0.5353	0.4716	0.4887	13606
Weighted avg	0.7739	0.8069	0.7777	13606

Tabla 45. Métricas BETO

Actual\Predicción	OFP	OFG	NO	NOE
OFP	740	0	1252	348
OFG	38	0	151	22
NO	206	0	9402	43
NOE	189	0	378	837

Tabla 46. Matriz confusión BETO

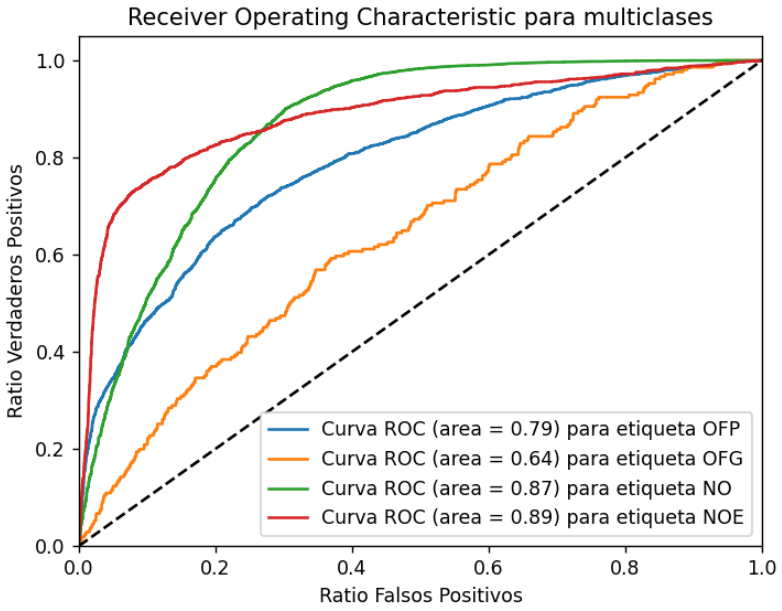


Ilustración 40. AUC-ROC BETO multiclase

Tal y como puede comprobarse este modelo alberga mucho ruido y presenta un macro-avg mediocre debido a que el sistema ha sido incapaz de encontrar representaciones OFG. Si aumentamos el número de epochs de 3 a 5 obtenemos estos resultados:

Etiqueta/Categoría	Precision	Recall	F1-Score	Support
OFP	0.7132	0.4603	0.5595	2340
OFG	0.0000	0.0000	0.0000	211
NO	0.8638	0.9667	0.9124	9651
NOE	0.7012	0.6467	0.6728	1404
Micro avg	0.8316	0.8316	0.8316	13606
Macro avg	0.5696	0.5184	0.5362	13606
Weighted avg	0.8077	0.8316	0.8128	13606

Tabla 47. Métricas ajuste hiperparámetros BETO

Actual\Predicción	OFP	OFG	NO	NOE
OFP	1077	0	974	289
OFG	52	0	131	28
NO	251	0	9330	70
NOE	130	0	366	908

Tabla 48. Matriz confusión ajuste hiperparámetros BETO

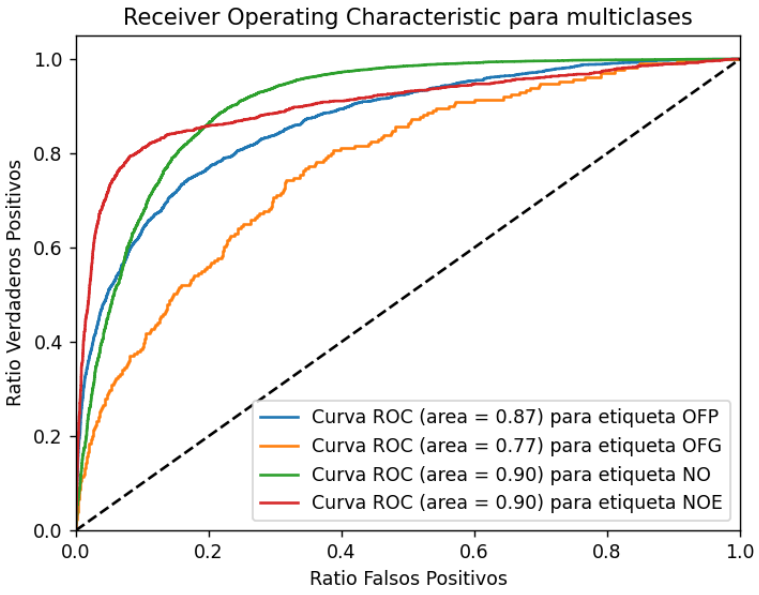


Ilustración 41. AUC-ROC ajuste hiperparámetros BETO multiclases

Si bien el modelo ha mejorado el micro avg y macro avg de todos los campos, aún es incapaz de clasificar en la etiqueta OFG.

Finalmente, si se entrena el modelo BETO en su configuración Large, sus métricas aumentan exponencialmente, siendo capaz de encontrar casos de OFG.

Etiqueta/Categoría	Precision	Recall	F1-Score	Support
OFP	0.7634	0.5765	0.6569	2340
OFG	0.4765	0.3839	0.4252	211
NO	0.8888	0.9553	0.9208	9651
NOE	0.7328	0.6759	0.7032	1404
Micro avg	0.8525	0.8525	0.8525	13606
Macro avg	0.7154	0.6479	0.6765	13606
Weighted avg	0.8447	0.8525	0.8453	13606

Tabla 49. Métricas BETO Large

Actual\Predicción	OFP	OFG	NO	NOE
OFP	1349	24	762	205
OFG	26	81	83	21
NO	265	46	9220	120
NOE	127	19	309	949

Tabla 50. Matriz confusión BETO Large

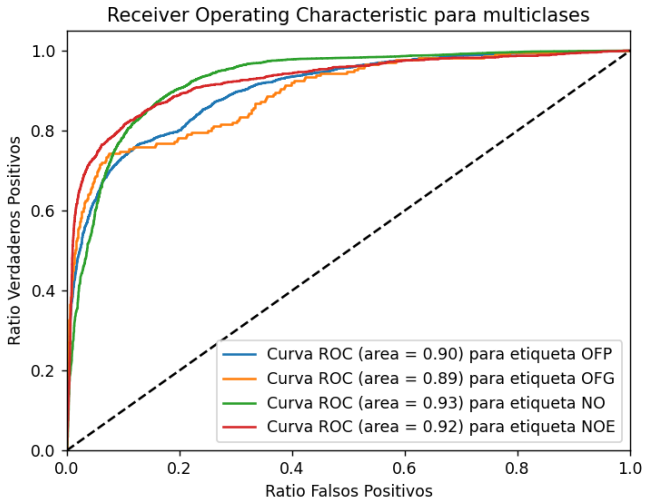


Ilustración 42. AUC-ROC BETO Large

6.2.3.2.3. Modelos RoBERTa

Un modelo RoBERTa Large con 3 epochs y argumentos de entrenamiento de serie ofrece estos resultados:

Etiqueta/Categoría	Precision	Recall	F1-Score	Support
OFP	0.8044	0.5731	0.6693	2340
OFG	0.5746	0.3649	0.4464	211
NO	0.8947	0.9689	0.9303	9651
NOE	0.7539	0.7265	0.7399	1404
Micro avg	0.8665	0.8665	0.8665	13606
Macro avg	0.7569	0.6584	0.6965	13606
Weighted avg	0.8597	0.8665	0.8583	13606

Tabla 51. Métricas RoBERTa Large

Actual\Predicción	OFP	OFG	NO	NOE
OFP	1341	20	754	225
OFG	21	77	99	14
NO	189	17	9351	94
NOE	116	20	248	1020

Tabla 52. Matriz confusión RoBERTa Large

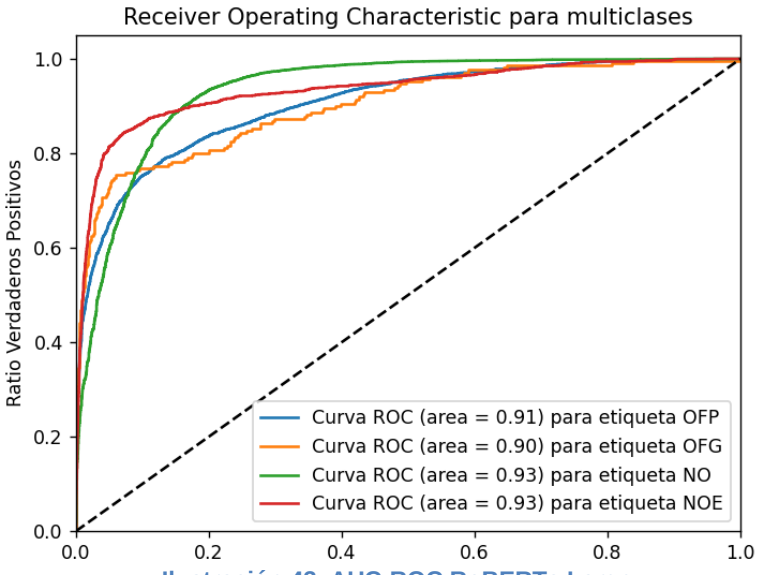


Ilustración 43. AUC-ROC RoBERTa Large

Se puede apreciar como mejoran todas las variables respecto a los modelos BETO Large a excepción del recall de OFP y OFG que disminuyen muy ligeramente. Si aumentamos el número de epochs a 5 obtenemos:

Etiqueta/Categoría	Precision	Recall	F1-Score	Support
OFP	0.7995	0.6321	0.7060	2340
OFG	0.5060	0.3981	0.4456	211
NO	0.8990	0.9632	0.9300	9651
NOE	0.7832	0.6973	0.7378	1404
Micro avg	0.8701	0.8701	0.8701	13606
Macro avg	0.7469	0.6727	0.7048	13606
Weighted avg	0.8639	0.8701	0.8641	13606

Tabla 53. Métricas RoBERTa Large ajuste hiperparámetros

Actual\Predicción	OFP	OFG	NO	NOE
OFP	1479	22	662	177
OFG	33	84	88	6
NO	236	31	9296	88
NOE	102	29	294	979

Tabla 54. Matriz confusión RoBERTa Large ajuste hiperparámetros

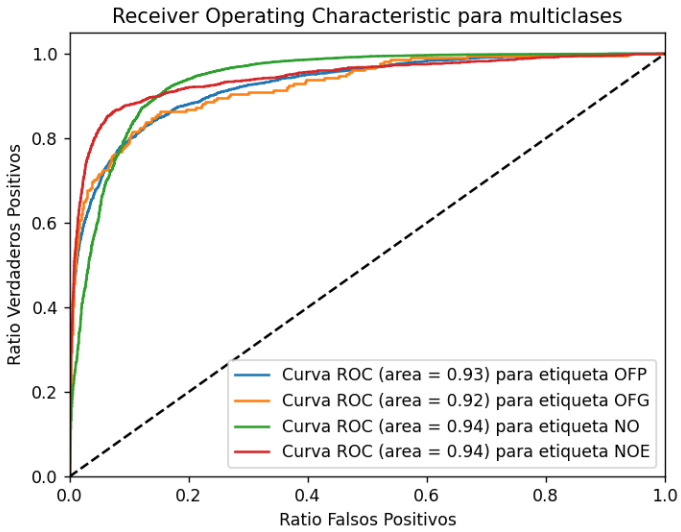


Ilustración 44. AUC-ROC RoBERTa Large ajuste hiperparámetros

6.2.3.2.4. Modelos XML-RoBERTa

Los modelos XML-ROBERTA son los más potentes, en su versión large y configurado con 5 epochs obtenemos los siguientes resultados:

Etiqueta/Categoría	Precision	Recall	F1-Score	Support
OFP	0.8228	0.6628	0.7342	2340
OFG	0.6024	0.4739	0.5305	211
NO	0.9030	0.9687	0.9347	9651
NOE	0.7970	0.6823	0.7352	1404
Micro avg	0.8789	0.8789	0.8789	13606
Macro avg	0.7813	0.6969	0.7337	13606
Weighted avg	0.8736	0.8789	0.8734	13606

Tabla 55. Métricas XML-RoBERTa Large

Actual\Predicción	OFP	OFG	NO	NOE
OFP	1551	20	622	147
OFG	22	100	77	12
NO	192	25	9349	85
NOE	120	21	305	958

Tabla 56. Matriz consufión XML-RoBERTa Large

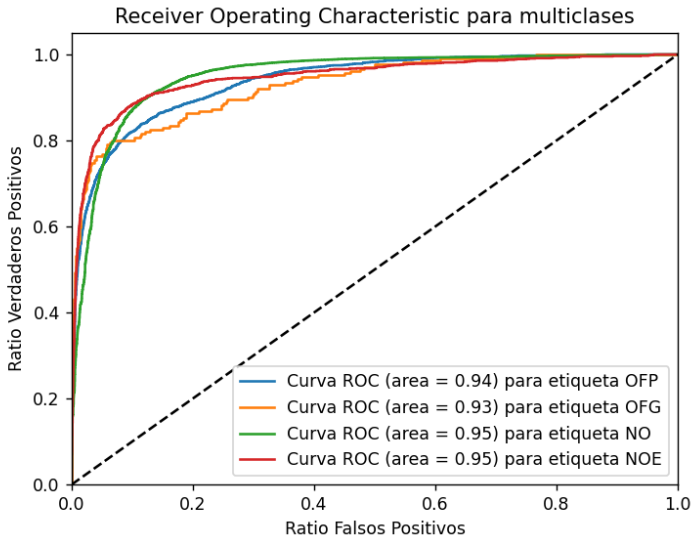


Ilustración 45. AUC-ROC XML-RoBERTa Large

6.2.4. Entorno de entrenamiento

Una vez expuestos los diferentes modelos derivados de BERT, en este apartado se detallan los entornos escogidos para su entrenamiento y su justificación.

Uno de los principales entornos utilizados ha sido el local, que cuenta con una GPU NVIDIA rtx 3070m. Esta potente tarjeta gráfica ofrece un rendimiento muy elevado, permitiendo entrenar modelos relativamente básicos como BERT o BETO base en aproximadamente media hora. Sin embargo, al entrenar modelos con más parámetros, las limitaciones del hardware comienzan a aparecer.

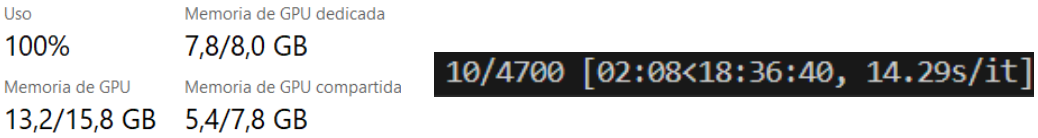


Ilustración 46. Memoria y tiempo requerido en el entrenamiento en local

Tal y como puede apreciarse en la **Ilustración 46**, aun contando con 8GB de VRAM, el entrenamiento de modelos avanzados como XML-RoBERTa Large requiere de un pequeño intercambio con la memoria principal del sistema (swapping), fruto de la gran envergadura del modelo. Por tanto, dado el largo tiempo de entrenamiento requerido para entrenar un modelo de 500 millones de parámetros, se ha optado por entrenar estos modelos simultáneamente en local y a través de herramientas en la nube como Google Collab.

Google Colab[7] es una plataforma gratuita que proporciona acceso a GPUs y TPUs para la ejecución de tareas de ML y DL. Google Collab permite el uso de una GPU Tesla T4 con 16GB de VRAM de forma gratuita, lo que evita la necesidad de utilizar memoria RAM adicional, mejorando así el rendimiento.

[7] https://colab.research.google.com/

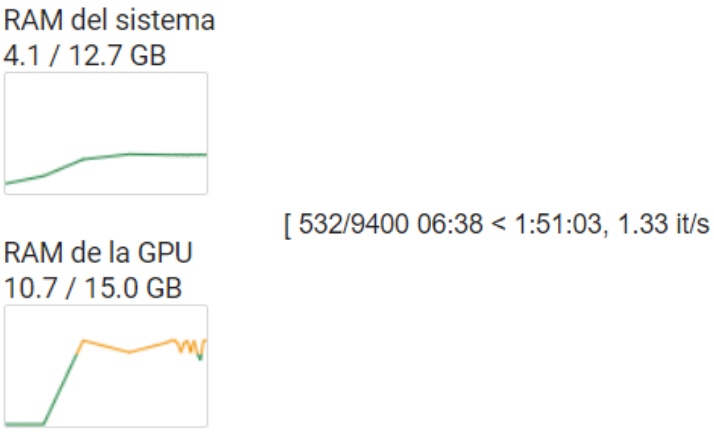


Ilustración 47. Memoria y tiempo requerido en el entrenamiento en la nube

Tal y como se aprecia en la **Ilustración 47**, Google Colab permite reducir drásticamente el tiempo requerido de entrenamiento de los modelos más avanzados (de 18 a 2 horas). Además, gracias al contar con el doble de VRAM en comparación con el hardware local, no es necesario el uso de memoria RAM repercutiendo en un menor tiempo de entrenamiento.

El uso simultáneo del hardware local para entrenar modelos más básicos y del hardware en la nube para entrenar modelos más avanzados permite obtener una gran variedad de modelos clasificatorios de lenguaje ofensivo de manera eficiente y rápida.

Aunque Google Colab proporciona una gran ayuda para acelerar el entrenamiento de estos modelos, su uso se ha visto reducido debido a la escasez de créditos en su plan gratuito. Esto ha provocado que, después de cada entrenamiento, no se pudiera utilizar la plataforma de inmediato.

6.2.5. Competición de los modelos en la campaña de evaluación

Una vez expuestas las diferentes arquitecturas de modelos desarrolladas para el proyecto, así como los resultados obtenidos con ellos, podemos concluir que el modelo desarrollado bajo la arquitectura XML-ROBERTA Large es el mejor en todos los casos, tanto en precisión micro como macro. Este será por tanto el modelo propuesto en las comparativas mostradas a continuación.

Tras haber elegido el modelo de clasificación de lenguaje ofensivo que ofrece los mejores resultados, podemos compararlo con los diferentes modelos presentados en la subtarea 1 y 2 de la tarea “MeOffendEs Offensive Language Detection in Spanish Variants” en la campaña de evaluación IberLEF 2021.

6.2.5.1. Micro F1-Score

Tal y como se ha detallado en el apartado **6.2.2**, el Micro F1-Score agrega las contribuciones de todas las clases para calcular el rendimiento global. Esta métrica pondera las clases según su prevalencia, proporcionando una visión del rendimiento del modelo en el conjunto de datos como un todo.

Equipo	Precision	Recall	F1-Score
NLP-CIC	0.8815	0.8815	0.8815
Equipo TFG	0.8789	0.8789	0.8789
UMUTeam	0.8782	0.8782	0.8782
GDUFS_DM	0.8732	0.8732	0.8732
Marta Isabel	0.8416	0.8416	0.8416
Baseline-svm	0.8285	0.8285	0.8285

Tabla 57. Comparativa Micro F1-Score competición

En nuestro estudio, el modelo propuesto alcanzó la segunda posición siendo superado únicamente por el modelo propuesto por el equipo NLP-CIC.

6.2.5.2. Macro F1-Score

Tal y como se ha detallado en el apartado **6.2.2**, el Macro F1-Score mide la precisión y exhaustividad en igual medida para cada clase, calculando un promedio simple de los F1-Scores de cada clase. Esta métrica es especialmente valiosa para medir el rendimiento con clases desbalanceadas, ya que estas contribuyen igualmente al score final.

Equipo	Precision	Recall	F1-Score
Equipo TFG	0.7813	0.6969	0.7342
NLP-CIC	0.7679	0.7093	0.7324
UMUTeam	0.7861	0.6919	0.7101
GDUFS_DM	0.7565	0.7002	0.7239
Marta Isabel	0.5781	0.5451	0.5595
Baseline-svm	0.6278	0.4831	0.5236

Tabla 58. Comparativa Macro F1-Score competición

En contraste, en términos de Macro F1-Score, nuestro modelo obtiene el mejor resultado, logrando una ventaja a la hora de manejar un equilibrio entre las clases, incluidas las minoritarias, ofreciendo un tratamiento más equitativo en la clasificación.

En términos generales, nuestro modelo obtiene excelentes resultados, rivalizando fuertemente con el primer lugar de esta tarea. La elección entre nuestro modelo o el de NLP-CIC debería basarse en la prioridad de la tarea a conseguir. Si el objetivo es maximizar la eficiencia en las clases predominantes, el modelo de la competencia es más adecuado. Sin embargo, si el enfoque está en promover la equidad entre clases de frecuencia variada, incluidas las menos representadas como OFG, nuestro modelo es más adecuado.

6.3. Desarrollo del backend

En este apartado se detalla la implementación de los servicios del backend, enfocándose en la arquitectura de servicios distribuidos utilizada para el desarrollo del sistema.

Una arquitectura de servicios distribuidos se refiere a una arquitectura en la cual diferentes partes de una aplicación se implementan y ejecutan como servicios separados, que pueden interactuar con la aplicación en su conjunto. Estos servicios no necesariamente se comunican entre sí directamente, sino que pueden ser independientes y se acceden desde el frontend.

A lo largo de la carrera, se ha enseñado a desarrollar servicios con arquitectura monolítica esbozando algunos detalles sobre las arquitecturas de microservicios. Sin embargo, una arquitectura de servicios distribuidos ofrece numerosas ventajas al trabajar en proyectos de este tipo que combinan algoritmos de aprendizaje profundo con desarrollo de aplicaciones web.

Categoría	Arquitectura de servicios distribuidos	Arquitectura monolítica	Arquitectura de microservicios
Escalabilidad	Alta, los servicios de Spring y Flask pueden escalarse de forma independiente	Baja, todo el sistema debe escalarse junto	Alta, los servicios de Spring y Flask pueden escalarse de forma independiente
Desacoplamiento	Alto. Los servicios están desacoplados y puedend esplegarse por separado	Bajo. Alta dependencia entre componentes	Alto. Cada servicio es autónomo y desacoplado
Flexibilidad	Alta. Permite usar Flask para servicios ligeros como las funcionalidades de los algoritmos de clasificación y Spring Boot para servicios robustos de interactividad entre usuarios	Baja. Todas las partes del sistema están estrechamente ligadas	Alta. Se pueden usar diferentes tecnologías para varios servicios

Tabla 59. Comparativa de arquitecturas backend

Los datos expuestos en la tabla anterior ejemplifican que las arquitecturas de microservicios y de servicios distribuidos son las que mejor escalan y mejores resultados ofrecen en aplicaciones que incorporan una gran variedad de módulos y tecnologías. La mayor diferencia entre una arquitectura de microservicios y una de servicios distribuidos reside en la comunicación de cada uno de los diferentes módulos.

En una arquitectura de microservicios, los servicios deben implementar distintos protocolos y medidas para comunicarse entre sí, mientras que en una arquitectura de servicios distribuidos estos no necesitan comunicarse directamente entre sí, sino a través de un frontend.

6.3.1. Desarrollo del servicio en Spring Boot

En este apartado se describe la estructura de los archivos y la implementación de las diferentes capas del servicio en Spring Boot, incluyendo la capa de dominio, la capa de persistencia y capa de servicios. Cada una de estas capas cumple un papel específico en la arquitectura de la aplicación, asegurando una separación clara de responsabilidades y facilitando el mantenimiento y la escalabilidad del sistema.

6.3.1.1. Estructura de archivos

La estructuración del proyecto es fundamental para mantener coherencia y organización en el código. En este proyecto, la estructura de archivos se ha organizado siguiendo buenas prácticas de desarrollo en Spring Boot.

La estructura del proyecto se organiza en los siguientes paquetes:

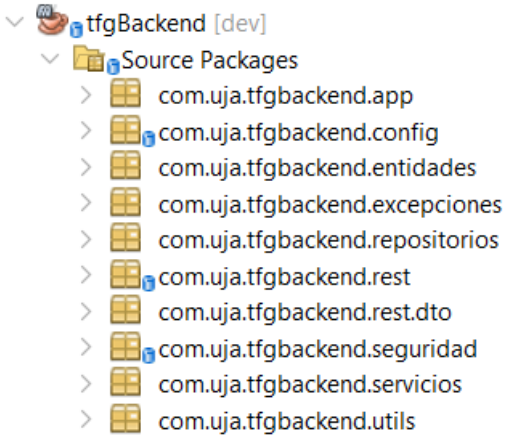


Ilustración 48. Archivos servicio Spring Boot

- **com.uja.tfgbackend.app**: Contiene la clase principal que inicia la aplicación Spring Boot.
- **com.uja.tfgbackend.config**: Incluye las configuraciones necesarias para el funcionamiento de la aplicación, como la configuración de Swagger o la inicialización de la base de datos.
- **com.uja.tfgbackend.entidades**: Define las entidades de dominio que representan los objetos de negocio y sus relaciones.
- **com.uja.tfgbackend.excepciones**: Maneja las excepciones personalizadas que se utilizan en la lógica de negocio.
- **com.uja.tfgbackend.repositorios**: Contiene los repositorios de JPA que interactúan con la base de datos.
- **com.uja.tfgbackend.rest**: Define los controladores REST que manejan las solicitudes HTTP y esponen los endpoints de la API.
- **com.uja.tfgbackend.rest.dto**: Incluye los objetos de transferencia de datos DTOs que se utilizan para transferir datos entre el cliente y el servidor.
- **com.uja.tfgbackend.seguridad**: Configura los aspectos de la seguridad de la aplicación, como la autenticación y la autorización.
- **com.uja.tfgbackend.servicios**: Contiene la lógica de negocio implementada en los servicios.
- **com.uja.tfgbackend.utils**: Proporciona utilidades y componentes auxiliares que se utilizan en diversas partes de la aplicación.

6.3.1.2. Implementación de la capa de dominio

La capa de dominio es el núcleo de la lógica de negocio de la aplicación. Representa los conceptos clave y las reglas de negocio específicas del dominio de la aplicación. En esta capa, se modelan las entidades principales y sus relaciones, encapsulando las reglas de negocio y asegurando la integridad de los datos.

Para la implementación de la capa de dominio, prestaremos especial atención al Driven Domain Design (DDD), una influyente corriente de desarrollo del software que otorga la mayor importancia al correcto modelado de la capa de dominio.

Según el DDD, el dominio debe de modelarse usando los siguientes elementos:

- **Entidades:** Modelan elementos importantes del dominio que mantienen una identidad única y permanente que los diferencian del resto.
- **Objetos-valor:** Representan una información, pero no tienen una identidad diferenciada que sea necesario mantener.
- **Servicios:** Agrupan operaciones que no pueden ser asignadas a una entidad u objeto-valor concreto, posiblemente porque implican interacciones con múltiples objetos.
- **Repositorios:** Son servicios especializados que gestionan la persistencia de los objetos del dominio, permitiendo su almacenamiento, búsqueda y borrado.
- **Factorías:** Sirve para construir objetos cuya creación e iniciación pueda ser compleja o porque existan varios métodos para tal fin.
- **Agregados:** Conjunto de objetos relacionados donde existe una fuerte dependencia. Un objeto hace de raíz y el resto están ligados a este.

Uno de los elementos principales introducidos en esta capa ha sido la inclusión de validación de beans. La validación de beans permite evitar mucho código de comprobación iterativo y de escaso interés. La validación de beans se centra en garantizar que los argumentos de operaciones o de los atributos tengan valores correctos. La validación de beans puede ser utilizada en Spring Boot incorporando la siguiente dependencia al archivo de configuración pom.xml.

```
1. <dependency>
2.             <groupId>org.springframework.boot</groupId>
3.             <artifactId>spring-boot-starter-validation</artifactId>
4. </dependency>
```

Código 18. Instalación Bean Validation en Spring Boot

Podemos encontrar validación de beans en todas las entidades. Para aclarar su significado se expondrá la entidad Usuario como ejemplo.

```
 1. @Entity
 2. @Table(name = "usuarios")
 3. public class Usuario {
 4.
 5.     @NotBlank
 6.     @Size(min = 3, max = 20)
 7.     @Id
 8.     private String nombreUsuario;
 9.
10.     @Size(min = 3, max = 20)
11.     private String nombre;
12.
13.     @Size(min = 3, max = 20)
14.     private String apellidos;
15.
16.     @Pattern(regexp = "[0-9]{8}[A-Z]")
17.     private String DNI;
18.
19.     @PastOrPresent
20.     private LocalDate fechaNacimiento;
21.
22.     @Pattern(regexp = "[0-9]+")
23.     private String telefono;
24.
25.     @NotBlank
26.     @Email
27.     private String email;
28.
29.     @NotBlank
30.     @Size(min = 6)
31.     private String claveAcceso;
32.
33.     @DecimalMin(value = "0.0", inclusive = true)
34.     @DecimalMax(value = "5.0", inclusive = true)
35.     private double valoracion;
36.
37.     @NotNull
38.     private int numValoracionesRecibidas;
39.
40.     @NotBlank
41.     private String ubicacion;
42.  …
43.}
```

Código 19. Ejemplo de Bean Validation

En este ejemplo, se utilizan algunas anotaciones de validación como:

- **@NotBlank:** Valida que un campo no sea null y que, además, no esté vacío.
- **@Size:** Valida que la longitud de una cadena esté dentro de los límites especificados.
- **@Pattern:** Valida que una cadena coincida con una expresión regular especificada.
- **@Email:** Valida que una cadena tenga el formato de una dirección de correo válida.
- **@DecimalMin:** Valida que un valor numérico sea mayor o igual a un valor mínimo especificado.
- **@DecimalMax:** Valida que un valor numérico sea menor o igual a un valor mínimo especificado.
- **@NotNull:** Valida que un campo no sea null.

6.3.1.3. Implementación de la capa de persistencia

La capa de persistencia es responsable de gestionar la interacción directa con la base de datos. En Spring Boot, esta capa se implementa principalmente utilizando Spring Data JPA e Hibernate. Hibernate es un framework de mapeo objeto-relacional (ORM) que facilita la persistencia de objetos Java en una base de datos relacional, mientras que Spring Data JPA proporciona una capa adicional de abstracción para simplificar el uso de JPA e Hibernate.

Los objetivos de la capa de persistencia son los siguientes:

- **Abstracción de la base de datos** proveyendo una interfaz simple y coherente para acceder y manipular datos.
- Asegurar que las operaciones sobre la base de datos se realizan **dentro de transacciones**, garantizando la consistencia y la integridad de los datos.
- **Mapear las entidades del dominio a las tablas de la base de datos**, facilitando la persistencia y recuperación de objetos Java.
- Permitir la ejecución de **consultas eficientes** y optimizadas a la base de datos a través del uso de consultas JPQL.

Hibernate juega un papel crucial en la capa de persistencia proporcionando las siguientes funcionalidades:

- El mapeo de entidades se produce a través de **anotaciones JPA** (@Entity @Table, @Id, entre otros).
- **Hibernate gestiona las sesiones y transacciones**, asegurando que las operaciones de la base de datos se realicen de manera consistente y eficiente.

Hibernate mejora diversos tipos de relaciones entre entidades, las cuales son fundamentales para modelar la lógica de negocio en una base de datos relacional. Estas relaciones pueden ser:

- **One to One:** Se define cuando una entidad está relacionada con una sola instancia de otra entidad.
- **One to Many:** Se define cuando una entidad está relacionada con múltiples sola instancia de otra entidad.
- **Many to One:** Se define cuando múltiples entidades están relacionadas con una sola instancia de otra entidad.
- **Many to Many:** Se define cuando múltiples entidades están relacionadas con múltiples instancias de otra entidad.

Estas relaciones pueden apreciarse con más claridad en el apartado **4.3.1.2**.

Otro aspecto fundamental en la capa de persistencia con Hibernate son las estrategias de carga que determinan cuándo y cómo se cargan las entidades relacionadas desde la base de datos.

Existen 2 estrategias de carga:

- **Carga perezosa:** Las entidades relacionadas se cargan sólo cuando se accede a ellas explícitamente. Esta estrategia es utilizada por defecto en las relaciones 1 a muchos o muchos a muchos.
- **Carga en cascada:** Las entidades relacionadas se cargan inmediatamente junto a la entidad principal en el momento que se consulta. Esta estrategia asegura que todas las entidades relacionadas estén disponibles de inmediato, pudiendo causar una sobrecarga si las relaciones son demasiado grandes. Esta estrategia es utilizada por defecto en las relaciones 1 a 1 o muchos a 1.

Para ejemplificar todo lo expuesto hasta ahora, se mostrará la entidad Usuario:

```
@Entity
@Table(name = "usuarios")
public class Usuario {
    @OneToMany(mappedBy = "creador", fetch = FetchType.EAGER)
    @MapKey(name = "idModelo")
    private Map<Integer, Modelo> modelosCreados;

    @OneToMany(mappedBy = "solicitante")
    @MapKey(name = "idPeticion")
    private Map<Integer, Peticion> peticionesEnviadas;
  …
```

Código 20. Ejemplo de persistencia con Hibernate

Tal y como se puede comprobar, primero definimos la anotación JPA @Entity y posteriormente declaramos dos mapas que representan los modelos creados y las peticiones enviadas por cada usuario.

Además de las relaciones unidireccionales, encontramos también relaciones bidireccionales. Estas permiten que dos entidades se refieran mutuamente, facilitando la navegación desde una entidad hacia la otra en ambas direcciones. En este ejemplo podemos observar la bidireccionalidad entre las entidades Modelo, Usuario y Peticion. Implementar bidireccionalidad implica utilizar la anotación JoinColumn en una de ellas. Esta anotación se utiliza para definir la columna en la tabla de la base de datos que se usará para unir una entidad con otra en una relación de muchos a 1 o 1 a 1. Esta columna actúa como una clave foránea que referencia la clave primaria de la entidad relacionada.

Podemos encontrar esta anotación JoinColumn en entidades como Modelo:

```
@Entity
@Table(name = "modelos")
public class Modelo {
    @ManyToOne
    @JoinColumn(name = "creador_id")
    private Usuario creador;
  …
```

Código 21. Ejemplo de relación bidireccional

Con el **Código 21** ya habríamos establecido la bidireccionalidad entre Usuario y Modelo.

Una vez definidas las entidades y las relaciones entre estas con las anotaciones de JPA, es fundamental la creación de los repositorios. Los repositorios son interfaces que proporcionan métodos predefinidos para las operaciones CRUD básicas.

Para este proyecto, se ha creado un repositorio para cada una de las entidades que componen el sistema, siendo su estructura muy parecida salvo las consultas específicas JPQL requeridas para cada entidad.

En el siguiente ejemplo podemos ver el repositorio de la entidad Usuario:

```
 1. @Repository
 2. public class RepositorioUsuario {
 3. 
 4.     @PersistenceContext
 5.     EntityManager em;
 6. 
 7.     @Cacheable("usuarios")
 8.     @Transactional(propagation = Propagation.SUPPORTS, readOnly = true)
 9.     public Optional<Usuario> buscar(String nombreUsuario) {
10.         return Optional.ofNullable(em.find(Usuario.class, nombreUsuario));
11.     }
12. 
13.     @Transactional
14.     public void guardar(Usuario usuario) {
15.         em.persist(usuario);
16.     }
17. 
18.     @Transactional
19.     public void actualizar(Usuario usuario) {
20.         em.merge(usuario);
21.     }
22. 
23.     @Transactional
24.     public List<Object[]> findTopModelCreators(int limite) {
25.         String jpql = "SELECT u.nombreUsuario, COUNT(m) FROM Usuario u JOIN u.modelosCreados
m GROUP BY u.nombreUsuario ORDER BY COUNT(m) DESC";
26.         return em.createQuery(jpql, Object[].class)
27.                 .setMaxResults(limite)
28.                 .getResultList();
29.     }
30. 
31.     @Transactional
32.     public List<String> findAllLocations() {
33.         String jpql = "SELECT u.ubicacion FROM Usuario u WHERE u.ubicacion IS NOT NULL";
34.         return em.createQuery(jpql, String.class).getResultList();
35.     }
36. }
```

Código 22. Repositorio de la entidad Usuario

En el **Código 22** podemos apreciar la anotación JPA @Repository, indispensable para que Spring lo reconozca. Las consultas guardar y actualizar permiten guardar o actualizar datos en la base de datos. Para las consultas más complejas se ha utilizado JPQL, que ofrece una sintaxis orientada a objetos que mejora la claridad y la portabilidad de las consultas a la base de datos.

6.3.1.4. Implementación de la capa de servicios

La capa de servicios es responsable de contener la lógica de negocio de la aplicación. Esta capa actúa como un intermediario entre la capa de persistencia y la capa de presentación. La capa de servicios es crucial para separar la lógica de negocio del frontend.

La capa de servicios de este proyecto incorpora una API REST que ha sido detallada previamente en el apartado **4.3.1.5** además de otras funcionalidades imprescindibles para asegurar la seguridad.

En una aplicación Spring Boot, la seguridad es una capa crítica que garantiza la protección de los datos. La configuración de seguridad se maneja a través de la clase ServicioSeguridadTfgbackend. Esta clase define las reglas de seguridad, los métodos de autenticación y la configuración de la política de sesiones.

Una de las características más importantes de esta clase es la implementación de JWT como método de autenticación. JWT es un estándar abierto que define una forma compacta y autónoma para transmitir información de manera segura entre un cliente y un servidor como un objeto JSON. Esta información puede ser verificada y confiable porque está firmada digitalmente.

Un JWT consta de 3 partes:

- **Header:** Incluye el tipo de token (JWT) y el tipo de firma, en este proyecto se ha usado la firma HMAC SHA256.
- **Payload:** Contiene las afirmaciones, estas son declaraciones sobre una entidad y datos adicionales.
- **Signature:** La firma se crea utilizando el encabezado codificado, el payload codificado, un secreto o clave privada, y el algoritmo especificado en el encabezado.

Un JWT funciona de la siguiente forma:

1. Cuando un usuario se autentica, el servidor valida las credenciales y, si son correctas, genera un JWT que incluye información sobre el usuario y otras afirmaciones.
2. El cliente almacena el JWT, en este caso se ha escogido el almacenamiento local.
3. Para acceder a recursos protegidos, el cliente envía el JWT en el encabezado de autorización de las solicitudes HTTP *Authorization: Bearer "token".*
4. El servidor recibe la solicitud y extrae el JWT del encabezado de autorización. El servidor verifica la firma del JWT para asegurarse que no ha sido manipulado y valida las afirmaciones.
5. Si el JWT es válido, el servidor permite el acceso a los recursos solicitados.

En la siguiente tabla se detallan las ventajas de JWT frente a métodos más simples como Basic Auth.

Característica	JWT	Basic Auth
Seguridad	Firma digital para asegurar integridad	Depende de HTTPS para la seguridad
Autonomía	Sí, contiene toda la información necesaria	No, requiere transmisión continua de credenciales
Escalabilidad	Alta, adecuado para servicios distribuidos o microservicios	Baja, necesita manejar sesiones
Flexibilidad	Alta, control preciso de expiración de tokens	Baja
Verificación	Descentralizada, cualquier servicio con la clave puede verificar el token	Centralizada, requiere verificación continua de credenciales en el servidor
Rendimiento	Alto, gracias a evitar consultas de sesión en el servidor	Medio, debido a la verificación continua de credenciales

Tabla 60. Comparativa métodos de seguridad

Para implementar JWT ha sido fundamental incorporar un decodificador de JWT utilizando una clave secreta codificada en Base64 para verificar y decodificar los tokens.

```
1. @Bean
2.     public JwtDecoder jwtDecoder() {
3.         String base64Key = Encoders.BASE64.encode(SECRET_KEY_BASE64.getEncoded());
4.         byte[] keyBytes = Decoders.BASE64.decode(base64Key);
5.         SecretKey originalKey = new SecretKeySpec(keyBytes, 0, keyBytes.length, "HmacSHA256");
6.         return NimbusJwtDecoder.withSecretKey(originalKey).build();
7.     }
```

Código 23. Decodificador JWT

Para la implementación de este mecanismo de seguridad también ha sido necesario contar con métodos auxiliares que se pueden encontrar en el paquete utils. Estos métodos son los siguientes:

- **generateToken:** Genera un JWT basado en el nombre de usuario proporcionado y se establece la fecha de emisión y expiración. Permite crear tokens de forma segura.

```
 1. public String generateToken(String username) {
 2.         return Jwts.builder()
 3.                 .setSubject(username)
 4.                 .claim("roles", "ROLE_USER")
 5.                 .setIssuedAt(new Date())
 6.                 .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 60 *
10))
 7.                 .signWith(SECRET_KEY_BASE64)
 8.                 .compact();
 9.     }
```

Código 24. Generación de tokens

- **validateToken:** Valida el JWT asegurándose de que el nombre de usuario en el token coincida con el proporcionado y que el token no haya expirado.

```
1. public Boolean validateToken(String token, String username) {
2.         try {
3.             final String usernameFromToken = extractUsername(token);
4.             return usernameFromToken.equals(username) && !isTokenExpired(token);
5.         } catch (JwtException | IllegalArgumentException e) {
6.             return false;
7.         }
8.     }
```

Código 25. Validación de tokens

- **extractUsername:** Extrae el nombre de usuario del token JWT.

```
1. public String extractUsername(String token) {
2.         return extractClaim(token, Claims::getSubject);
3.     }
```

Código 26. Extracción de nombre a través de tokens

- **extractExpiration:** Extrae la fecha de expiración del token JWT.

```
1. private Date extractExpiration(String token) {
2.         return extractClaim(token, Claims::getExpiration);
3. }
```

Código 27. Extracción de la fecha de

- **extractClain:** Permite extraer cualquier claim del token.

```
1. private <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
2.         final Claims claims = extractAllClaims(token);
3.         return claimsResolver.apply(claims);
4. }
```

Código 28. Extracción de claim del token

- **extractAllClaims**: Permite extraer todos los claims del token.

```
1. private Claims extractAllClaims(String token) {
2.         return Jwts.parser()
3.                 .setSigningKey(SECRET_KEY_BASE64)
4.                 .parseClaimsJws(token)
5.                 .getBody();
6. }
```

Código 29. Extracción de todos los claims del token

- **isTokenExpired:** Verifica si el token JWT ha expirado comparando la fecha de expiración con la fecha actual.

```
1. private Boolean isTokenExpired(String token) {
2.         return extractExpiration(token).before(new Date());
3. }
```

Código 30. Comprobación de expiración del token

Otra de las grandes características implementadas en la capa de servicio para fomentar la seguridad es la configuración de CORS. CORS permite o restringe las solicitudes realizadas desde un dominio diferente al dominio desde el cual se sirve el recurso. Es crucial para el desarrollo de aplicaciones web modernas con un frontend y backend alojados en diferentes orígenes.

CORS es un mecanismo que utiliza cabeceras HTTP adicionales para permitir que un usuario autorizado obtenga permiso para acceder a recursos seleccionados desde un servidor en un origen distinto al del sitio desde el cual se realizó la solicitud inicial. Es una medida de seguridad que ayuda a prevenir ciertos ataques como el Cross-Site Request Forgery.

En este proyecto, CORS se ha configurado mediante la creación de una clase de configuración específica llamada corsConfiguration.

Este corsConfiguration implementa muchos métodos, sin embargo, los más importantes son:

- **allowedOrigins:** Especifica los orígenes permitidos para hacer solicitudes a la API. En este caso se permiten solicitudes desde el puerto 5173 del localhost.
- **allowedMethods:** Especifica los métodos HTTP permitidos.

Una vez definidos los métodos y orígenes permitidos es esencial configurar el método filterChain del ServicioSeguridad. Este método permite especificar qué peticiones HTTP deben estar protegidas y cuáles deben estar abiertas, en función de diferentes criterios como el tipo de método HTTP, la URL solicitada, los roles de usuario, entre otros.

```
1. public class ServicioSeguridadTfgbackend {
2.
3.     @Bean
4.     public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
5.         http
6.                 .cors()
7.                 .and()
8.                 .csrf().disable()
9.                 .authorizeHttpRequests(
10.                         authz -> authz
11.                                 .requestMatchers("/swagger-ui.html", "/swagger-ui/**",
"/v3/api-docs/**").permitAll()
12.                                 .requestMatchers(HttpMethod.POST,
"/tfgbackend/usuarios").permitAll()
13.                                 .requestMatchers(HttpMethod.POST,
"/tfgbackend/usuarios/login").permitAll()
14.                                 .requestMatchers(HttpMethod.GET,
"/tfgbackend/usuarios/top").permitAll()
15.                                 .requestMatchers(HttpMethod.GET,
"/tfgbackend/usuarios/ubicaciones").permitAll()
16.                                 .requestMatchers(HttpMethod.GET,
"/tfgbackend/usuarios/{nombreUsuario}").authenticated()
17.                                 .requestMatchers(HttpMethod.GET,
"/tfgbackend/usuarios/{nombreUsuario}/token").permitAll()
18.                                 .requestMatchers(HttpMethod.GET,
"/tfgbackend/usuarios/{nombreUsuario}/valoracion").permitAll()
19.                                 .requestMatchers(HttpMethod.POST,
"/tfgbackend/modelos/corpus").permitAll()
20.                                 .requestMatchers(HttpMethod.POST,
"/tfgbackend/modelos/**").authenticated()
21.                                 .requestMatchers(HttpMethod.POST,
"/tfgbackend/modelos/{idModelo}/peticionesAcceso").authenticated()
22.                                 .requestMatchers(HttpMethod.PUT,
"/tfgbackend/modelos/{idModelo}/peticionesAcceso/{idPeticion}").authenticated()
23.                                 .requestMatchers(HttpMethod.GET,
"/tfgbackend/modelos").permitAll()
24.                                 .requestMatchers(HttpMethod.GET,
"/tfgbackend/modelos/**").permitAll()
25.                                 .anyRequest().authenticated()
26.                 )
27. …
```

Código 31. Cadena de seguridad del servicio Spring Boot

En este ejemplo podemos comprobar que algunos endpoints del controlador REST como */tfgbackend/usuarios* no requieren de autenticación mientras otras como */tfgbackend/modelos/{idModelo}/peticionesAcceso* sí.

Otro de los servicios fundamentales para la ejecución de este proyecto es el servicio de Almacenamiento. Este servicio es una parte esencial para la gestión y actualización del corpus de entrenamiento para entrenar los modelos de clasificación de lenguaje ofensivo. Este servicio es capaz de interactuar con los servicios en la nube de Azure Blob Storage para subir, descargar y actualizar archivos de manera eficiente y segura.

Azure Blob Storage es un servicio de almacenamiento de objetos escalable y seguro en la nube, proporcionado por Microsoft Azure. Es ideal para almacenar cantidades masivas de datos no estructurados, como archivos de texto.

El servicioAlmacenamiento gestiona la interacción con Azure Blob Storage y presenta los siguientes métodos principales:

- **uploadFile:** Permite subir un archivo local a un contenedor específico.

```
1. public void uploadFile(String localFilePath, String containerName, String blobName)
throws Exception {
2.          BlobContainerClient containerClient =
blobServiceClient.getBlobContainerClient(containerName);
3.          BlobClient blobClient = containerClient.getBlobClient(blobName);
4.          blobClient.uploadFromFile(localFilePath, true);
5.      }
```

Código 32. Subida de archivos a un contenedor

- **downloadFile:** Permite descargar un archivo blob de un contenedor específico en Azure Blob Storage.

```
1. public byte[] downloadFile(String containerName, String blobName) {
2.          BlobContainerClient containerClient =
blobServiceClient.getBlobContainerClient(containerName);
3.          BlobClient blobClient = containerClient.getBlobClient(blobName);
4.          return blobClient.downloadContent().toBytes();
5.      }
```

Código 33. Descarga de archivos de un contenedor

- **appendToFile:** Añade contenido a un archivo existente en Azure Blob Storage.

```
 1. public void appendToFile(String containerName, String blobName, String comment,
String label) throws Exception {
 2.         BlobContainerClient containerClient =
blobServiceClient.getBlobContainerClient(containerName);
 3.         BlobClient blobClient = containerClient.getBlobClient(blobName);
 4.
 5.         if (!blobClient.exists()) {
 6.             throw new Exception("Blob not found: " + blobName);
 7.         }
 8.
 9.         // Descargar el contenido existente del Blob
10.         ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
11.         blobClient.download(outputStream);
12.         String existingContent = new String(outputStream.toByteArray(),
StandardCharsets.UTF_8);
13.
14.         // Elimina cualquier línea nueva final del contenido existente para evitar
líneas en blanco adicionales
15.         existingContent = existingContent.trim();
16.
17.         // Agregar una nueva línea con campos vacíos para las columnas no utilizadas
18.         String newContent = existingContent + String.format("\n\t%s\t\t\t\t%s",
comment, label);
19.
20.         // Convertir string a InputStream
21.         byte[] newContentBytes = newContent.getBytes(StandardCharsets.UTF_8);
22.         InputStream dataStream = new ByteArrayInputStream(newContentBytes);
23.
24.         // Subir el nuevo contenido del Blob
25.         blobClient.upload(dataStream, newContentBytes.length, true);
26.         dataStream.close();
27.     }
28.
```

Código 34. Añadir contenido a un archivo de Azure Blob Storage

6.3.1.5. Elementos adicionales de seguridad

Para garantizar la seguridad de las contraseñas de los usuarios, se ha implementado el algoritmo de cifrado Argon2. Este algoritmo es ampliamente reconocido por su seguridad y eficiencia en la protección de contraseñas.

Argon2 es un algoritmo de hashing de contraseñas que fue el ganador del Password Hashing Competition (PHC) en julio de 2015. Es conocido por su resistencia contra ataques de fuerza bruta y su capacidad de ajuste en términos de tiempo y memoria, lo que lo hace especialmente adecuado para proteger contraseñas en aplicaciones modernas.

La implementación de Argon2 es muy sencilla, para conseguirlo se ha creado una clase en el paquete utils llamada CustomPasswordEncoder.

```
 1. @Component
 2. public class CustomPasswordEncoder implements
org.springframework.security.crypto.password.PasswordEncoder {
 3.
 4.     private Argon2 argon2 = Argon2Factory.create();
 5.
 6.     @Override
 7.     public String encode(CharSequence rawPassword) {
 8.         char[] charArray = rawPassword.toString().toCharArray();
 9.         return argon2.hash(2, 65536, 1, charArray);
10.     }
11.
12.     @Override
13.     public boolean matches(CharSequence rawPassword, String encodedPassword) {
14.         char[] charArray = rawPassword.toString().toCharArray();
15.         return argon2.verify(encodedPassword, charArray);
16.     }
17. }
```

Código 35. Implementación del cifrado Argon2

Una vez incorporada la lógica del método de cifrado elegido, podemos incorporar esta funcionalidad añadiendo una simple línea al constructor de la entidad Usuario.

```
1. public Usuario(String _nombre_usuario, String _nombre, String _apellidos, String _DNI,
LocalDate _fechanacimiento, String _telefono, String _email, String _claveAcceso, String
_ubicacion) {
2.         …
3.         this.claveAcceso = passwordEncoder.encode(_claveAcceso);
4.         …
5.     }
```

Código 36. Uso del cifrado Argon2

6.3.1.6. Testing

Una vez elaborado el backend en su totalidad, es de vital importancia comprobar la funcionalidad de este. En este apartado se describen las pruebas implementadas sobre este backend a través de JUnit.

Para realizar las pruebas del backend se ha utilizado la metodología Test-Driven Development (TDD), la cual se baja en el ciclo de desarrollar pruebas unitarias antes de escribir el código funcional. Este enfoque introducido por Kent Beck se ha convertido en una técnica fundamental para asegurar la calidad y confiabilidad del software.

La metodología TDD se basa en un ciclo iterativo de tres pasos conocidos como Red-Green-Refactor. Este ciclo se basa en escribir una prueba que falle, posteriormente escribir el código mínimo necesario para que la prueba pase y finalmente mejorar el código asegurándose que las pruebas continúen pasando.

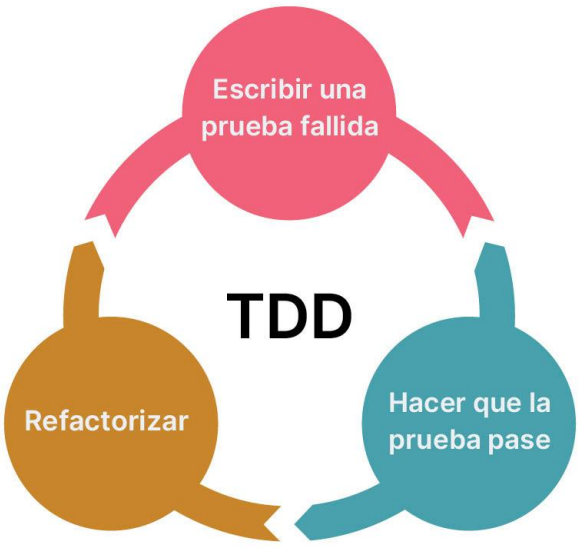


Ilustración 49. Metodología TDD

Esta metodología ofrece numerosos beneficios:

- **Mejora la calidad del código** obligando a los desarrolladores a pensar en los requisitos y en los posibles errores antes de escribir el código, lo que reduce la cantidad de bugs y mejora la calidad general del software.

- TDD **fomenta el desarrollo incremental e iterativo**, perfecto para trabajar con metodologías ágiles como Scrum, utilizada para este proyecto.
- **Los errores se detectan inmediatamente después de escribir código**, lo que reduce significativamente el tiempo dedicado a la depuración.

Junit es una herramienta principal en la práctica de TDD, ya que permite ejecutar las pruebas automatizadas de manera eficiente. Para incorporar JUnit a un backend en Spring Boot únicamente es necesario añadir una dependencia al archivo pom.xml.

```
1. <dependency>
2.             <groupId>org.junit.jupiter</groupId>
3.             <artifactId>junit-jupiter</artifactId>
4.             <scope>test</scope>
5. </dependency>
```

Código 37. Incorporar JUnit a Spring Boot

Los Test que se han integrado con Junit son de 2 tipos, tests unitarios y de integración. Los tests unitarios tienen como objetico verificar el correcto funcionamiento de las validaciones de beans implementadas a las entidades durante el desarrollo de la capa de dominio. Los tests de integración por otro lado tienen como objetivo verificar la interacción entre los diferentes componentes del sistema. Esto hace que los test de integración sean mucho más complejos y lentos de ejecutar frente a los tests unitarios.

Para este proyecto se han desarrollado 26 test en total, de los cuales 10 son unitarios y 16 de integración. Esos 16 tests de integración se han dividido en 9 tests para el controlador de la API REST y 7 tests para el servicio.

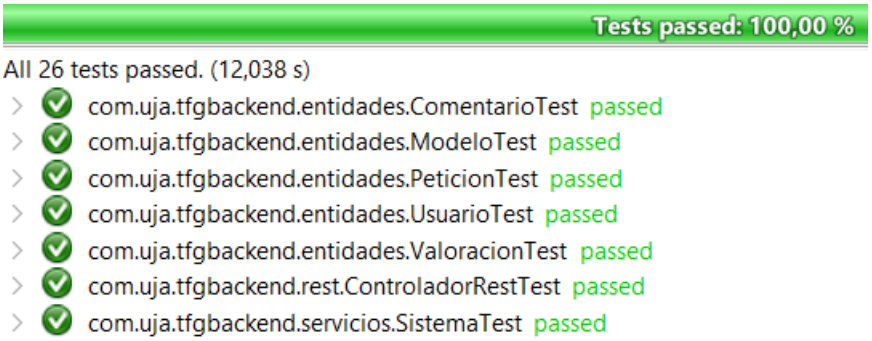


Ilustración 50. Resultados tests unitarios y de integración

6.3.2. Desarrollo del servicio en Flask

En este apartado se detalla la implementación de un servidor web que utiliza Flask para gestionar el uso de los modelos de DL encargados de la clasificación de lenguaje ofensivo. Este servicio es crucial para permitir la carga dinámica de modelos, predicción de etiquetas, y actualización de datos, entre otros.

A diferencia del desarrollo del servicio en Spring Boot, no se ha diferenciado el código en una arquitectura de capas independientes debido a la flexibilidad que nos ofrece Python y a la escasez de endpoints expuestos en la API.

Una de las ventajas más notorias que nos ofrece Flask es la facilidad para la creación de APIs. Para exponer un servicio a una API RESTful solo es necesario añadir una línea de código como la siguiente:

```
1. @app.route('/prediccion', methods=['POST'])
```

Código 38. Exposición de un servicio con Flask

En este ejemplo, estamos definiendo una ruta con el endpoint */prediccion* que utiliza una petición POST para invocar al método asociado.

Los endpoints expuestos por la API son:

- **/modelos** (POST)**:** Permite cargar modelos de clasificación.
- **/prediccion** (POST)**:** Permite realizar predicciones sobre un texto dado utilizando un modelo previamente cargado.
- **/estado-proceso** (GET)**:** Proporciona el estado actual del proceso de carga de un modelo.
- **/matriz-confusion** (GET): Calcula y devuelve la matriz de confusión para las predicciones realizadas por un modelo.
- **/corpus** (GET): Genera un enlace de descarga para el corpus de entrenamiento desde Azure Blob Storage.

Uno de los aspectos críticos que debe manejar el servicio Flask es la gestión eficiente de la memoria de vídeo de la GPU, especialmente al trabajar con múltiples modelos de DL. Para evitar exceder la capacidad máxima de la VRAM y mantener el sistema funcionando de manera eficiente se ha implementado un balanceador de carga basado en la política First in First Out (FIFO).

La política FIFO es una estrategia de gestión de recursos donde los elementos más antiguos se eliminan primero cuando se necesita liberar espacio para nuevos elementos. En el contexto de este trabajo, el modelo más antiguo se descarga primero si la memoria VRAM alcanza un determinado límite, en este caso el 80% de la capacidad total de la GPU.

```
VRAM_LIMIT = torch.cuda.get_device_properties(0).total_memory * 0.8

def check_memory_and_cleanup():
    print("USANDO: "+ str(torch.cuda.memory_allocated() / (1024 ** 3)) + "GB de VRAM")
    if torch.cuda.memory_allocated() > VRAM_LIMIT:
        # Aplicar política FIFO para eliminar el modelo más antiguo
        if models:
            old_model = models.pop(0)
            del old_model['model']
            torch.cuda.empty_cache()
```

Código 39. Planificador de tareas del servicio Flask

Este fragmento del **Código 39** verifica constantemente el uso de la memoria VRAM y, si se supera el límite establecido, elimina el modelo más antiguo de la memoria para liberar espacio.

6.4. Desarrollo del frontend

En este apartado se detalla la implementación de la capa de presentación de la aplicación utilizando React, TailwindCSS y Typescript. Estas tecnologías han sido previamente descritas en el apartado 3 de este proyecto.

6.4.1. Estructura de archivos

Uno de los aspectos más importantes para tener un frontend escalable es tener una estructura bien organizada y segmentada, que facilite la navegación y el mantenimiento del proyecto. Para proyectos como este donde la capa de presentación implementa una gran cantidad de módulos es importante definir una gran variedad de carpetas. Cada una tiene un propósito claro, lo que mejora la comprensión del código.

En este proyecto se ha seguido la siguiente estructura de archivos:

- **interfaces/:** Contiene definiciones de tipos e interfaces para TypeScript.

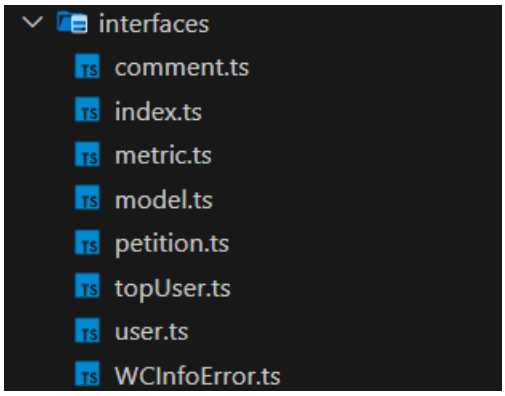


Ilustración 51. Carpeta interfaces del frontend

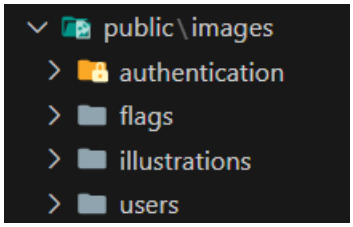


Ilustración 52. Carpeta public del frontend

- **public/:** Contiene archivos públicos que no pasan por el proceso de construcción del Webpack, como iconos de perfiles ficticios, entre otros.

- **src/:** Contiene el código fuente principal de la aplicación. Cuenta con varios subdirectorios:
    - **api/:** Contiene las rutas base de los 2 servicios para interactuar con sus respectivas API, encontramos un apiBase.ts y un flaskBase.ts.

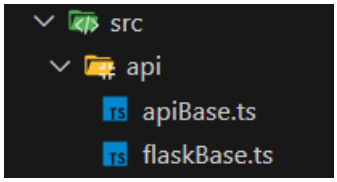


Ilustración 53. Carpeta src del frontend

    - **components/:** Contiene los componentes específicos de la aplicación por categorías. Por ejemplo, el directorio models alberga los componentes que muestran información sobre los modelos, mientras que el directorio reusable contiene componentes reutilizables para evitar

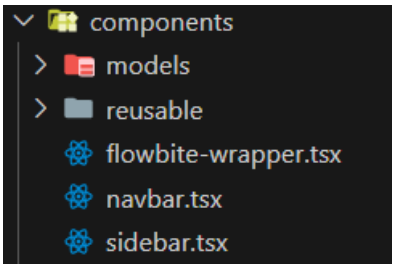


Ilustración 54. Carpeta components del frontend

la duplicación de código en múltiples partes de la aplicación.

- **context/:** Almacena contextos y providers para gestionar parte del estado global.

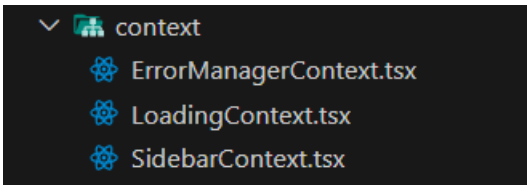


Ilustración 55. Carpeta context del frontend

- **helpers/:** Contiene funciones de ayuda.
- **layouts/:** Define los diseños generales de la aplicación, concretamente los del navbar y sidebar.

- **pages/:** Posiblemente uno de los directorios más importantes, contiene las páginas de la aplicación. Estas páginas se estructuran según su funcionalidad, dando lugar a las subcarpetas authentication, models, pages y tests.
- **router/:** Gestiona las rutas de la aplicación, pudiendo ser tanto privadas como

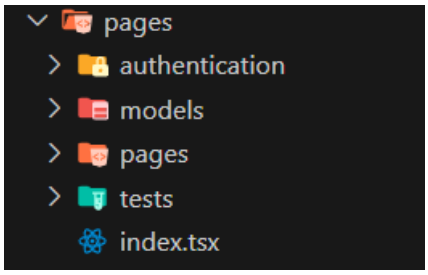


Ilustración 56. Carpeta pages del frontend

públicas.

- **store/:** Junto a pages, es con amplia diferencia el directorio más importante. Almacena los diferentes slices que gestionan el estado del frontend a través del patrón Redux y la arquitectura Flux. Debido a la magnitud de esta carpeta, diferenciamos cada uno de sus elementos:

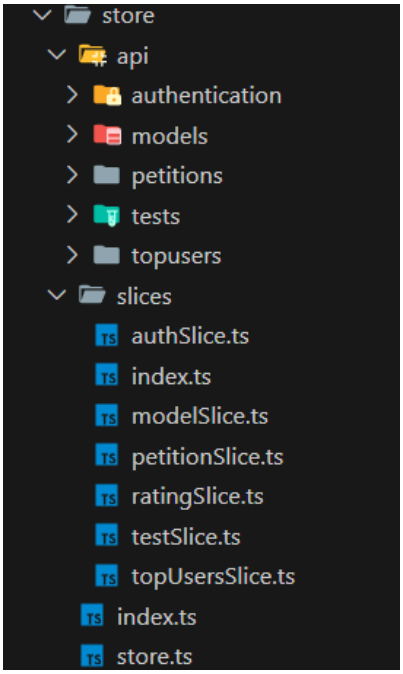


Ilustración 57. Carpeta store del frontend

  - **api/:** Almacena la lógica de interacción con las APIs RESTful definidas en los dos servicios del backend. Esta lógica se estructura en varios thunks que representan operaciones a endpoints específicos. Estos thunks se organizan en varios subdirectorios según el tipo de página con

la que interactúan, dando lugar a subdirectorios como, por ejemplo, authentication, models, petitions o tests.

  - **slices/**: Define los slices de Redux que dividen el estado en partes manejables.

- **utils/:** Almacena utilidades y funciones generales.

Podemos observar la gran cantidad de archivos creados y modularizados en diferentes directorios para mantener los principios de React: la construcción de componentes pequeños, escalables y reutilizables. En la siguiente ilustración se mostrará la jerarquía general de directorios del frontend:

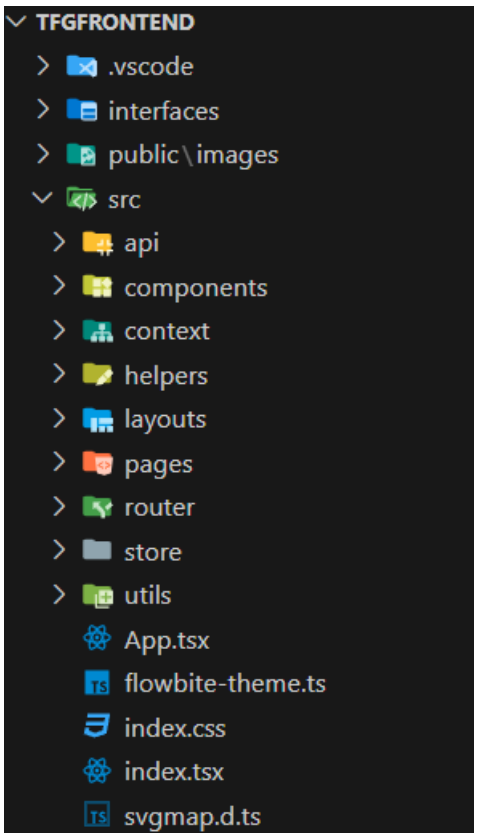


Ilustración 58. Jerarquía de archivos del frontend

6.4.2. Implementación

La implementación de la capa de presentación de este proyecto ha sido una tarea que ha requerido una gran cantidad de esfuerzo, en este apartado se detallan algunas de las principales funcionalidades y patrones de diseño utilizados para el desarrollo de la interfaz de usuario.

6.4.2.1. Enrutamiento

El enrutamiento en aplicaciones web permite navegar entre diferentes vistas o componentes dentro de la aplicación, sin necesidad de recargar la página completa. Este tipo de aplicaciones, conocidas como Single Page Application (SPA), utilizan el enrutamiento para facilitar la transición suave entre diferentes estados de la aplicación, proporcionando una experiencia de usuario más fluida y rápida.

Para conseguir el funcionamiento deseado, la aplicación utiliza react-router-dom para gestionar el enrutamiento. Este paquete proporciona los componentes necesarios para definir rutas y manejar la navegación en una aplicación React.

```
1. const App: FC = () => {
2.   return (
3.     <Provider store={store}>
4.       <BrowserRouter>
5.         <LoadingProvider>
6.           <Routes>
7.             <Route element={<FlowbiteWrapper />}>
8.               <Route element={<PublicRoute />}>
9.                 <Route path="/" element={<DashboardPage />} index />
10.                <Route path="/authentication/sign-in" element={<SignInPage />} />
11.                <Route path="/authentication/sign-up" element={<SignUpPage />} />
12.                <Route path="/tests" element={<TestsPage />} />
13.                <Route path="*" element={<NotFoundPage />} />
14.              </Route>
15.
16.              <Route element={<PrivateRoute />}>
17.                <Route path="/my-models" element={<ModelsPage />} />
18.                <Route path="/my-models/:id" element={<PredictPage />} />
19.                <Route path="/all-models" element={<ModelsPage />} />
20.                <Route path="/all-models/:id" element={<PredictPage />} />
21.                <Route path="/pages/maintenance" element={<MaintenancePage />} />
22.              </Route>
23.
24.            </Route>
25.          </Routes>
26.        </LoadingProvider>
27.      </BrowserRouter>
28.    </Provider>
29.  );
30. };
31. export default App;
```

Código 40. Enrutamiento frontend

Tomando de referencia el **Código 40**, conviene centrarse en la separación de rutas privadas y públicas. Esto se logra gracias a los componentes PrivateRoute y PublicRoute.

- **Rutas públicas:** Estas rutas son accesibles para todos los usuarios, independientemente de si están autenticados o no. Ejemplos de estas rutas son las páginas de inicio de sesión (/authentication/sign-in) y de registro (/authentication/sign-up).
- **Rutas privadas:** Estas rutas están protegidas y solo son accesibles para los usuarios autenticados. Para gestionar esto, el componente PrivateRoute valida el token JWT guardado en localStorage. Si el token es válido, el usuario puede acceder a la página solicitada; de lo contrario, será redirigido al menú de inicio de sesión.

La aplicación está diseñada para permitir el acceso a las funcionalidades básicas al mayor número de personas posible, mientras que ciertas funciones avanzadas están reservadas para los usuarios registrados. Este enfoque asegura que los recursos sensibles y las acciones críticas estén adecuadamente protegidos, mejorando tanto la seguridad como la experiencia del usuario.

6.4.2.2. Patrones de diseño

En el desarrollo de la aplicación, se han implementado varios patrones de diseño para mejorar la estructura del código. A continuación, se detallan algunos de los implementados:

6.4.2.2.1. Patrón Observador

El patrón observador es un patrón de diseño que define una relación de dependencia 1 a muchos entre objetos, de manera que cuando un objeto cambia su estado, todos sus dependientes son notificados y actualizados automáticamente. Este patrón es especialmente útil en aplicaciones donde varios componentes necesitan reaccionar a cambios en el estado de la aplicación.

En React, el patrón observador se puede implementar fácilmente utilizando el hook useEffect. Este hook permite ejecutar efectos secundarios en componentes funcionales, como la suscripción a un servicio, la actualización de un estado global o la realización de limpiezas después de que un componente se desmonte.

A continuación, se muestra un ejemplo de cómo se utiliza el hook useEffect para implementar el patrón observador:

```
1.  useEffect(() => {
2.        if (models.length > 0) {
3.           dispatch(fetchPetitions({ models, userName }));
4.        }
5.    }, [models]);
```

Código 41. Patrón observador en React

Este ejemplo pertenece a uno de los dos useEffect que encontramos en ModelsPage, la página encargada de mostrar el listado de modelos. Aquí, useEffect se utiliza para suscribirse a cambios de los modelos en la aplicación.

El mencionado useEffect se ejecuta cuando el componente se monta y cada vez que cambia el estado de models, algo totalmente lógico ya que cuando se añade un nuevo modelo, es necesario volver a recoger las peticiones que han recibido los modelos.

6.4.2.2.1. Patrón Redux

Para gestionar el estado global de la aplicación, se ha utilizado el patrón Redux, explicado en profundidad en el apartado **4.3.2.1**. A continuación, se detalla la implementación del patrón Redux en el contexto de la gestión de modelos de clasificación.

En el componente ModelsPage, la función principal es mostrar un listado de modelos. Aunque esta información podría almacenarse dentro del propio componente usando hooks como useState, surgen varias complicaciones cuando otros componentes requieren también de esta información.

En escenarios simples, si los componentes que requieren esta información son hijos del componente que la gestiona, se podría pasar la información a través de props. Sin embargo, esto conlleva el uso de prop drilling, una técnica que puede volverse engorrosa y difícil de manejar a medida que la aplicación crece. El mayor inconveniente aparece cuando componentes no relacionados jerárquicamente necesitan acceder a los mismos datos, lo que resultaría en múltiples llamadas redundantes a la API REST, afectando negativamente al rendimiento de la aplicación.

En el componente ModelsPage, se utiliza el hook useSelector para acceder al estado de los modelos almacenados en el store de Redux y useDispatch para despachar acciones que actualizan el estado.

```
 1.  useEffect(() => {
 2.        setIsLoading(true);
 3.        Promise.all([dispatch(fetchModels({ userName: currentPath.includes("all-models") ?
undefined : userName, date: undefined, modelName: undefined }))])
 4.           .then(() => {
 5.              setIsLoading(false);
 6.           })
 7.           .catch((error) => {
 8.              console.error(error);
 9.              setIsLoading(false);
10.           });
11.     }, []);
```

Código 42. Patrón Redux en React

Para realizar las llamadas asíncronas a la API y actualizar el estado, se utiliza el thunk fetchModels:

```
interface Props {
    modelName: string | undefined,
    userName: string | undefined,
    date: string | undefined
}

export const fetchModels = createAsyncThunk(
    "models/fetchModels",
    async (args: Props, { dispatch }) => {
        try {
            // Dispatch la acción de inicio de búsqueda
            dispatch(fetchModelStart());
            const { userName, modelName, date } = args

            const params = modelName !== undefined
                ? { nombre: modelName }
                : userName !== undefined
                    ? { nombreUsuario: userName }
                    : date !== undefined
                        ? { fecha: date }
                        : {};

            const { data } = await apiBase.get<Model[]>(`/modelos`, {
                params
            });

            // Dispatch la acción de éxito con los resultados
            dispatch(fetchModelSuccess(data));

            for (const model of data) {
                const nombreUsuario = model.nombreUsuarioCreador;
                await dispatch(fetchUserRating({ nombreUsuario }));
            }

            return Promise.resolve();

        } catch (error) {
            const err = error as AxiosError<InfoError>;
            return Promise.reject(err);
        }
    }
);
```

Código 43. Implementación de thunk de Redux

Una vez recibidos los datos por parte del endpoint de la API REST, se despacha fetchModelSuccess para actualizar el slice con los modelos obtenidos.

7. Conclusiones y trabajos futuros

Durante este proyecto, se han desarrollado una amplia variedad de tareas, desde el desarrollo de una aplicación web completa hasta el modelado de avanzados sistemas clasificatorios capaces de reconocer ofensividad en textos. Para conseguirlo, ha sido necesaria una extensa base de documentación que permitiera desarrollar una avanzada arquitectura de servicios distribuidos en el backend y una moderna capa de presentación para elaborar interfaces intuitivas, usables y funcionales. Además, el modelado de los sistemas clasificatorios ha marcado un punto crucial en el desarrollo del proyecto, siendo necesaria una larga etapa de investigación para descubrir las arquitecturas más avanzadas de la actualidad para la construcción de sistemas basados en técnicas de aprendizaje profundo.

Desde un punto de vista técnico, este proyecto me ha permitido adquirir una vasta cantidad de conocimientos sobre diversas tecnologías, desde el desarrollo web hasta complejas arquitecturas de aprendizaje profundo.

Desde un punto de vista organizativo, la gran cantidad de tiempo enfocada a la planificación y análisis del proyecto me ha permitido afianzar numerosas técnicas útiles para trabajar en grandes equipos de desarrollo en un futuro.

En términos de resultados, el proyecto ha logrado cumplir cada uno de los objetivos propuestos al inicio. Se ha incorporado con detalle cada una de las capas que componen al sistema, incluyendo funcionalidades que no estaban previstas desde un principio. Abordando las necesidades requeridas por parte de los modelos clasificatorios, este sistema ha sido capaz de rivalizar con el mejor competidor de la tarea de evaluación elegida. En cuanto a la construcción del backend, se han desarrollado servicios avanzados que proporcionan una amplia gama de funcionalidades rigurosamente testadas. Como se mencionó al principio del proyecto, este TFG no solo se enfoca a la elaboración de sistemas clasificatorios, sino que va mucho más allá. La elaboración de avanzados modelos de lenguaje carece de sentido sin una aplicación práctica, algo que se ha enfatizado en este TFG al permitir que cualquier usuario pruebe sus características y fomente la mejora constante de los modelos.

El punto más fuerte de este proyecto ha sido, sin duda, su impacto personal. Este TFG me ha permitido explorar muchas áreas que en inicialmente no pensé jamás que serían posibles de desarrollar, fruto no solo de mi propio esfuerzo, sino también del apoyo de muchas más personas, incluyendo a mis tutoras de TFG que me han acompañado durante esta etapa.

Uno de los puntos sobre los cuales pueden trabajarse a futuro, es el uso de un servidor en la nube para albergar los servicios y no depender de su ejecución en local, algo que en este proyecto se ha evitado por las repercusiones económicas que tendría.

En resumen, este TFG constituye el mayor trabajo que he realizado a lo largo de la carrera, un proyecto al cual se le han dedicado 300 horas de largo esfuerzo y dedicación. Sin embargo, lejos de ser un fin en sí mismo, este TFG representa un comienzo. Vivimos en una época donde la IA y especialmente el PLN están en su máximo apogeo, y aunque no podemos prever con certeza las arquitecturas y avances que nos deparará el futuro a corto plazo, lo que es incuestionable es que tarde o temprano llegarán. Este trabajo, por tanto, representa un punto de partida para la creación de sistemas de detección más precisos en el futuro, modelos capaces de rivalizar más estrechamente con la comprensión humana y lo más importante, modelos que contribuyan significativamente a reducir la propagación del discurso de odio.

8. Apéndice I. Manual de instalación

En este apartado se detallan los pasos necesarios para ejecutar todas las funcionalidades del proyecto en local. La ejecución del proyecto es independiente del sistema operativo elegido. Los requisitos previos son instalar Docker Desktop desde su web oficial, instalar el gestor de paquetes Yarn y tener el puerto 3306 o 3307 disponibles para su uso por parte de la base de datos MySQL.

Para instalar el gestor Yarn es necesario seguir los siguientes pasos:

- **Windows:**

  1. Ir a la web oficial de Yarn y descargar el instalador MSI para Windows.
  2. Ejecutar el archivo MSI descargado y seguir las instrucciones del instalador.

- **Linux:**

a. Configurar el repositorio con el comando:

```
1.  curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add -
2.  echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee
    /etc/apt/sources.list.d/yarn.list
```

b. Actualizar e instalar yarn con la siguiente instrucción:

```
1.  sudo apt update && sudo apt install yarn
```

En ambos casos podemos verificar que Yarn se ha instalado correctamente a través del comando:

```
1. yarn --version
```

Una vez cumplidos estos requisitos previos los pasos a seguir son los siguientes:

1. La aplicación Docker Desktop debe estar ejecutándose.
2. En un directorio a elección propia, debe situarse el archivo *docker-compose.yml* que forma parte de los recursos del proyecto.
3. Se debe abrir una consola de comandos, navegando al directorio donde se encuentra el archivo *docker-compose.yml* y ejecutar las instrucciones:

```
1.  docker-compose pull
2.  docker-compose up
```

Estos comandos desplegarán las 2 imágenes de Docker del backend, instalando todas sus dependencias y archivos auxiliares. Estas imágenes tienen un peso de 23,8GB para el servicio Flask y de 170,5MB para el servicio Spring Boot.

4. En otra terminal, se debe navegar al directorio donde se encuentre la carpeta *tfgfrontend* y una vez situados dentro de ella, ejecutar la instrucción:

```
1. yarn preview
```

O si esto fallase:

```
1. yarn dev
```

Una vez desplegado el frontend de la aplicación, se podrá ver la plataforma usando la URL: http://localhost:4173/ para preview (aconsejable) o http://localhost:5173/ para dev.

9. Apéndice II. Manual de usuario

En este apartado se van a mostrar las diferentes pantallas que contiene nuestra aplicación, fruto del esfuerzo de las etapas previas. El diseño de las interfaces se ha basado en principios de usabilidad para garantizar una experiencia de usuario intuitiva y eficiente. Algunos de estos principios incluyen el uso de una paleta de colores limitada para mantener una apariencia limpia y profesional, la disposición coherente de elementos para facilitar la navegación y la presentación clara de la información relevante.

Uno de los detalles importantes a mencionar antes de adentrarnos en el diseño final de estas interfaces, es que existen algunos usuarios registrados por defecto. Si se quieren utilizar estos perfiles se deben usar los siguientes datos:

- Para el usuario _**danielcc**:
  - **Username:** _danielcc
  - **Password:** usuario23
- Para el usuario **Fernan21**:
  - **Username:** Fernan21
  - **Password:** fernando23

Todas y cada una de las interfaces implementan diseños responsivos, siendo visibles desde dispositivos de diferente tamaño. Además, todas las interfaces se componen de un esqueleto compuesto por una barra de navegación y una barra lateral, que son indispensables para conseguir una navegación fluida en toda la aplicación.

Cada una de las interfaces presenta su versión propia para dispositivos móviles, permitiendo que el uso de la aplicación sea óptimo en cualquier dispositivo. Para adaptar las interfaces a estas pantallas de menor tamaño, se ha utilizado el Samsung Galaxy S20 Ultra como dispositivo de referencia.

A continuación, se mostrarán las interfaces de escritorio y dispositivos móviles de cada una de las partes de la aplicación:

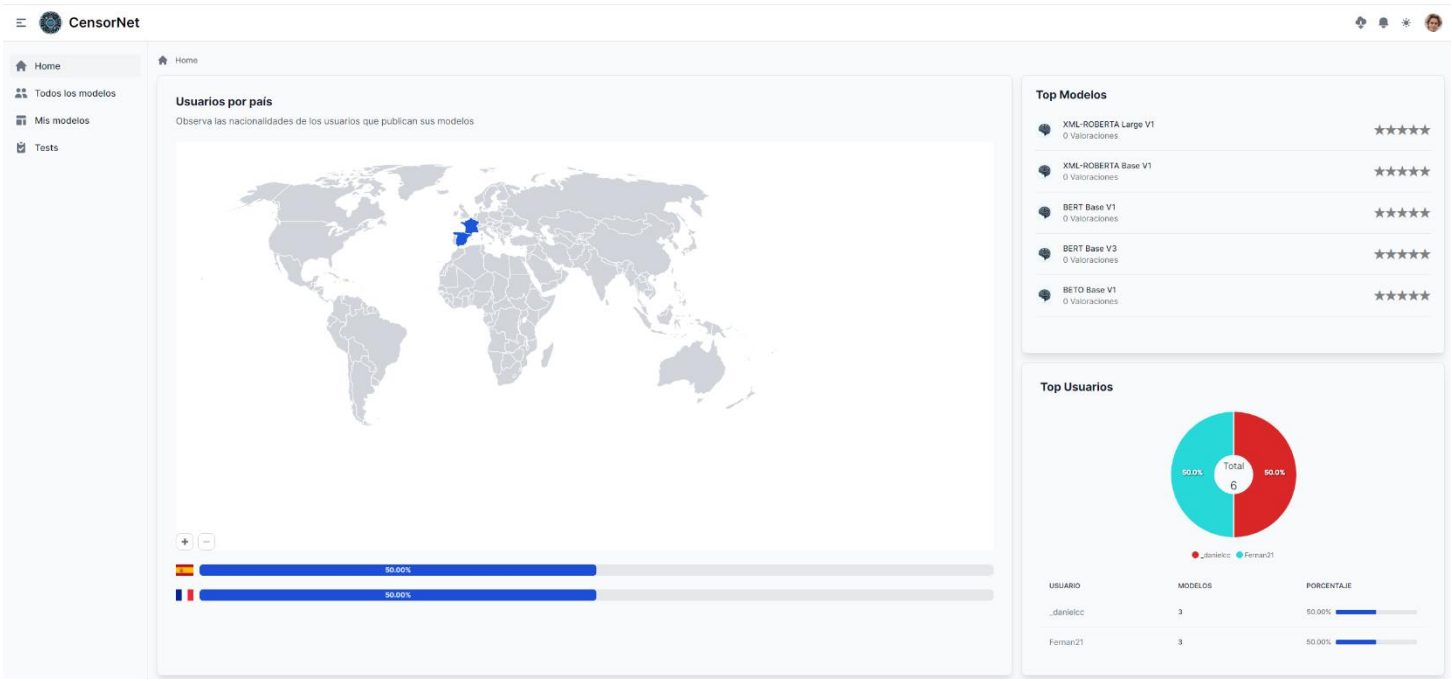


Ilustración 60. Dashboard de escritorio con mapa de usuarios, top modelos más valorados y top usuarios con más modelos

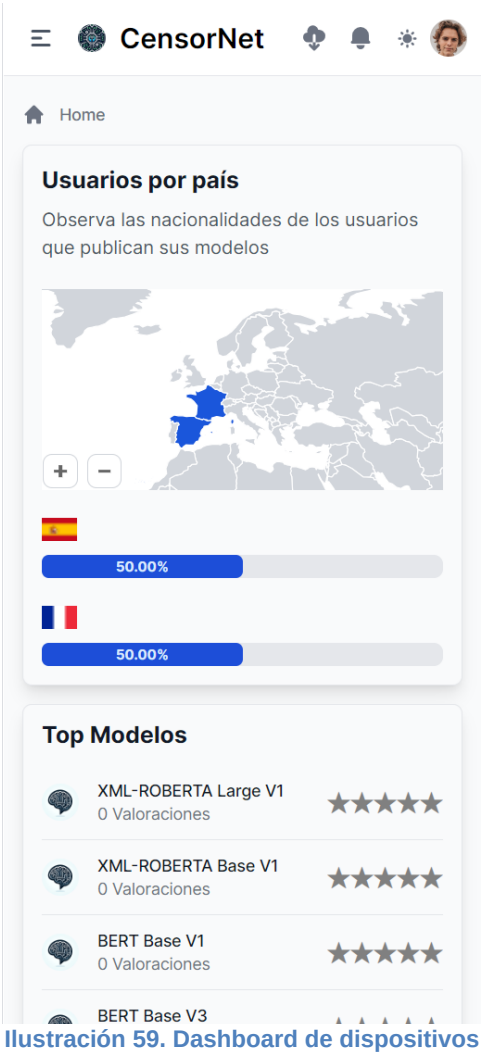


Ilustración 59. Dashboard de dispositivos móviles

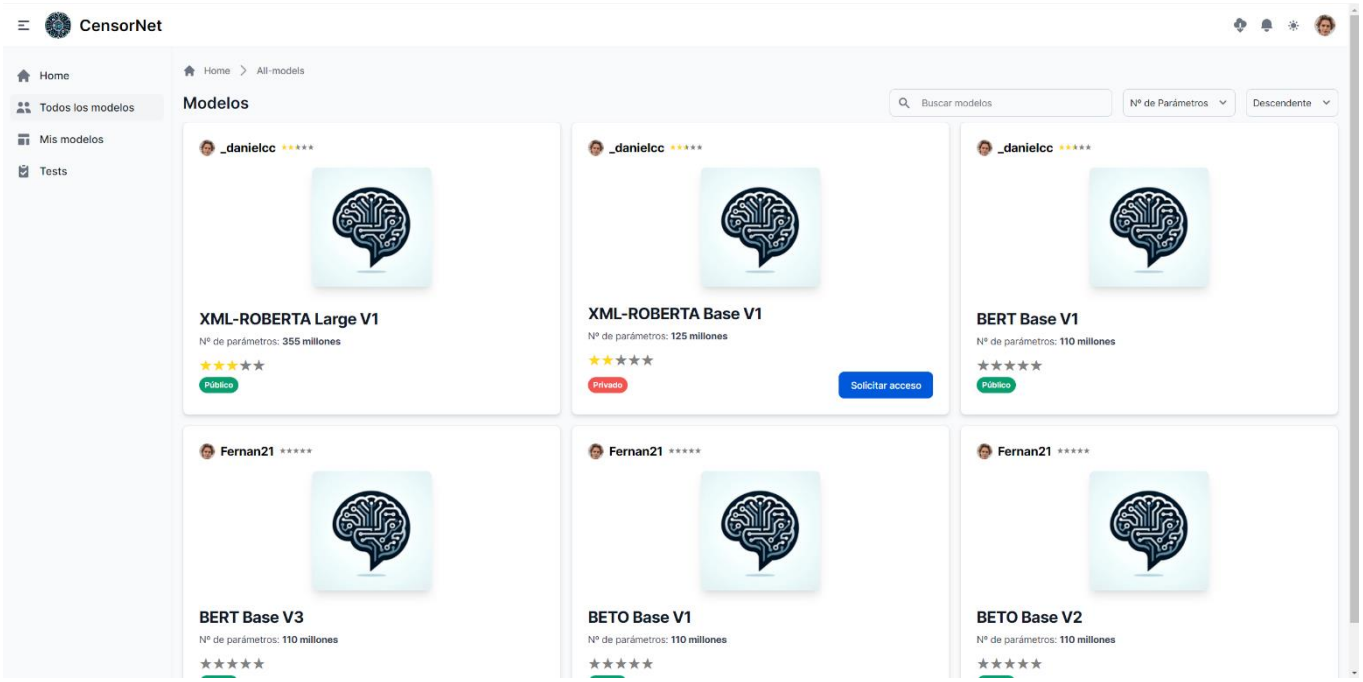


Ilustración 62. Listado de modelos de escritorio con información variada y filtros de búsqueda, puntuación y nº de parámetros

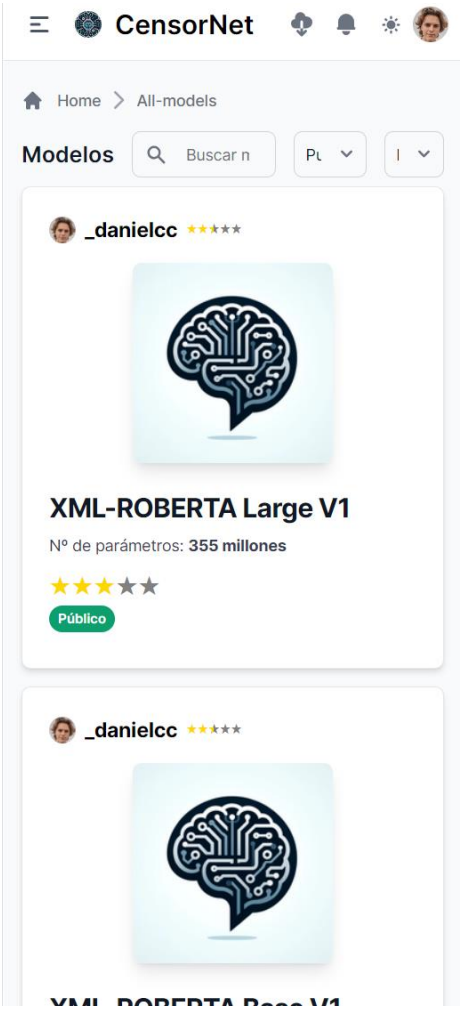


Ilustración 61. Listado de modelos de dispositivos móviles

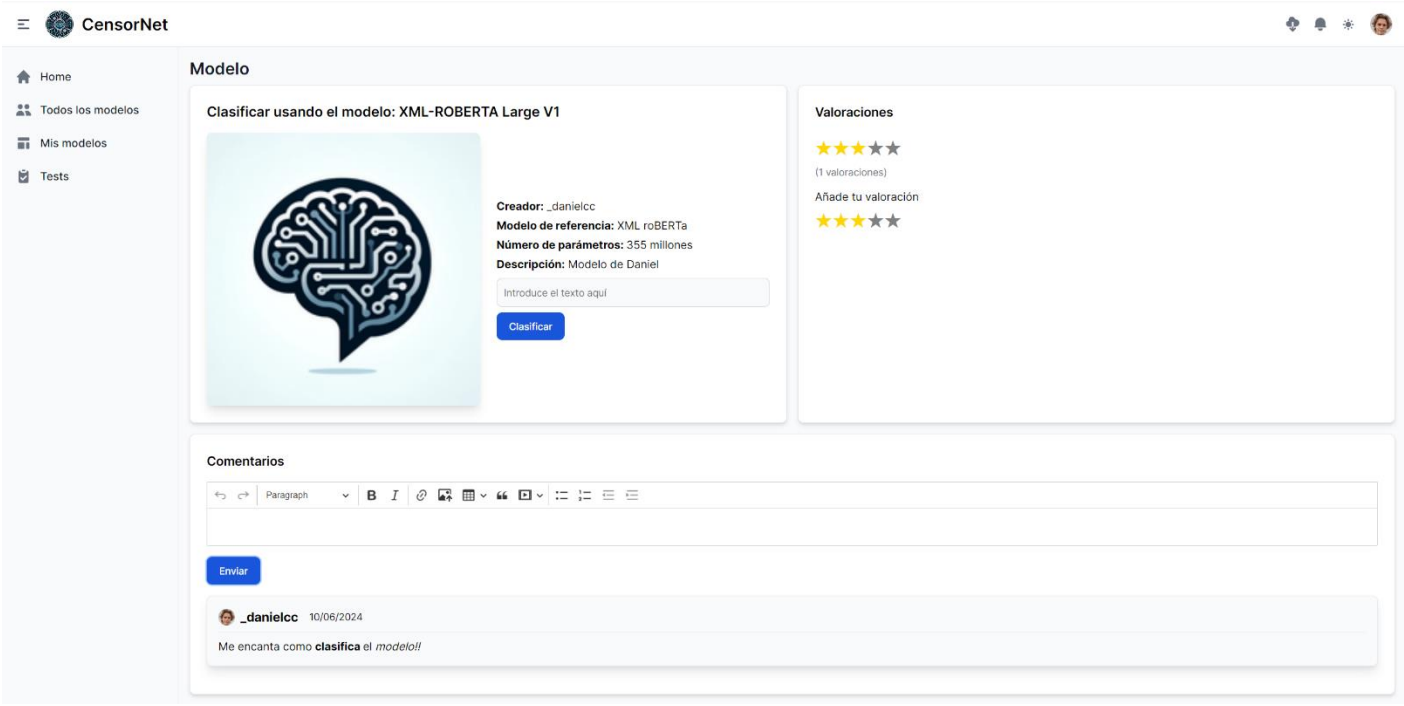


Ilustración 64. Página de modelo de escritorio con zona de clasificación, valoraciones y tablón de comentarios con editor WYSIWYG

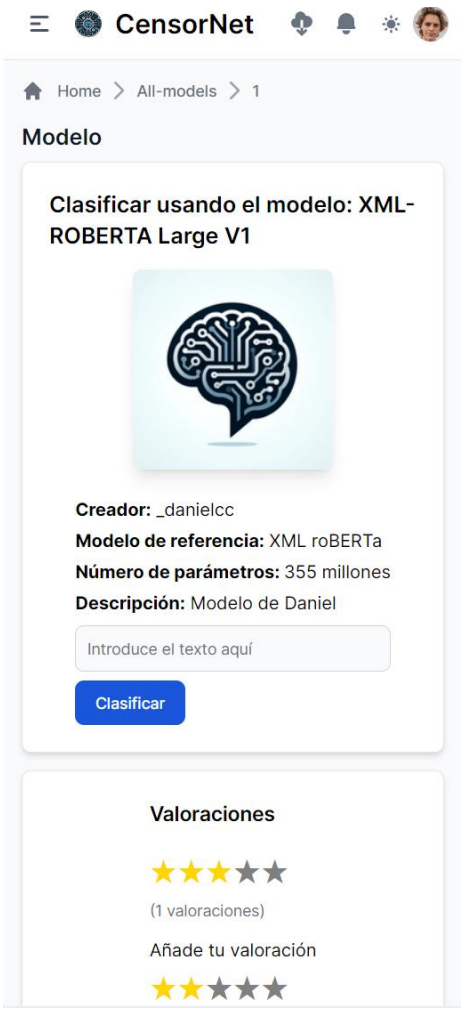


Ilustración 63. Página de modelo de dispositivos móviles

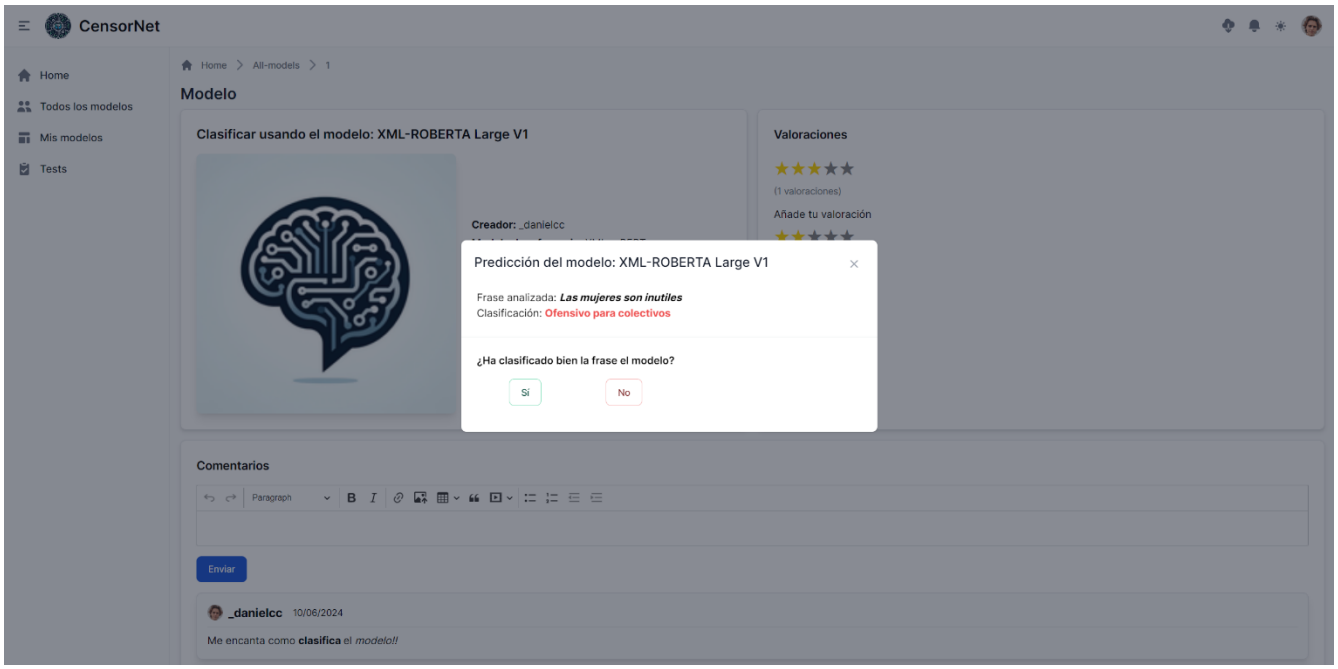


Ilustración 66. Modal de escritorio con resultados de clasificación textual

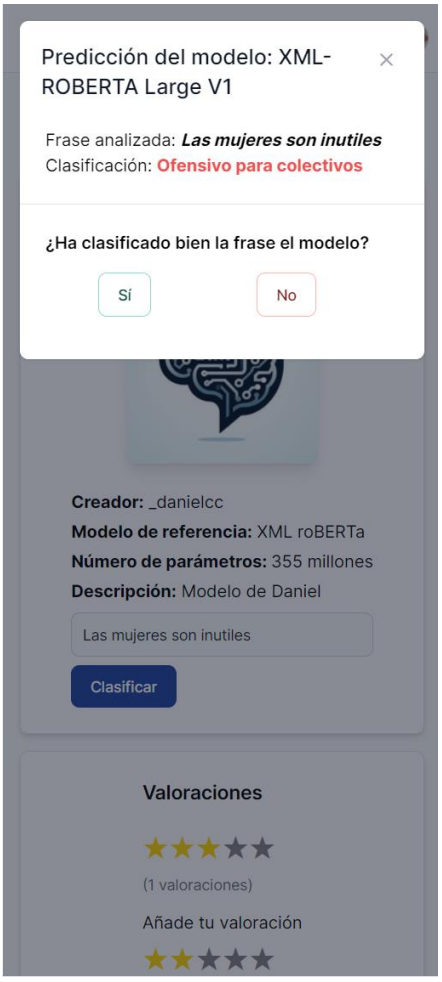


Ilustración 65. Modal de dispositivos móviles

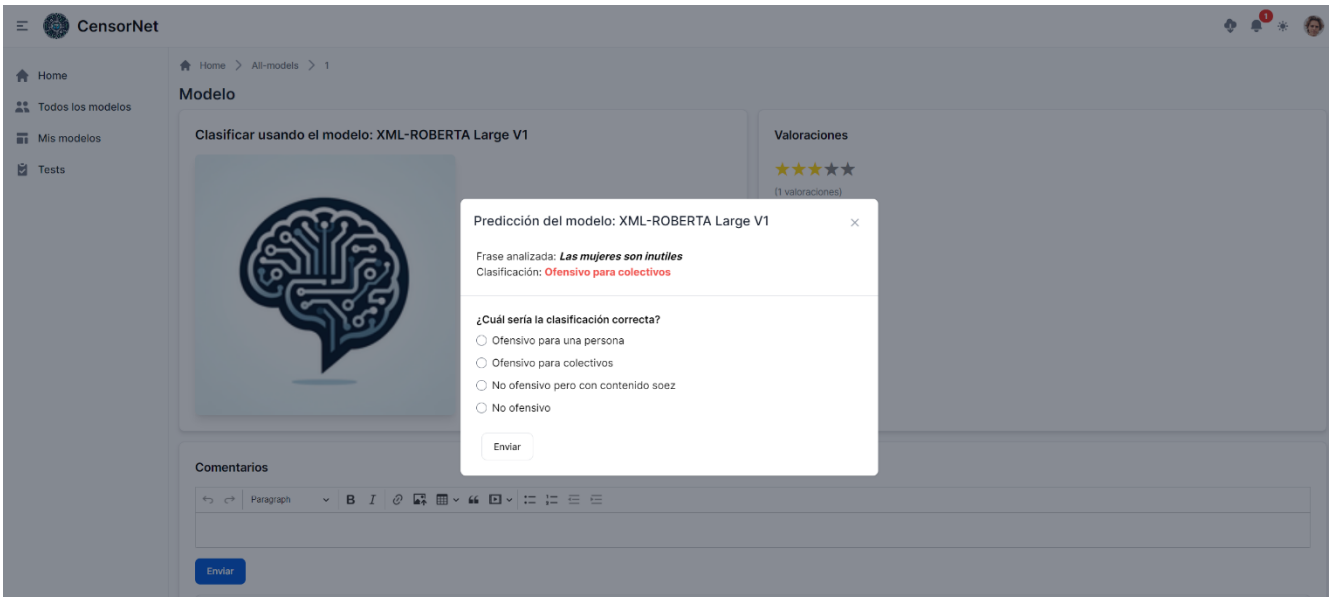


Ilustración 67. Modal de escritorio preguntando la etiqueta correcta

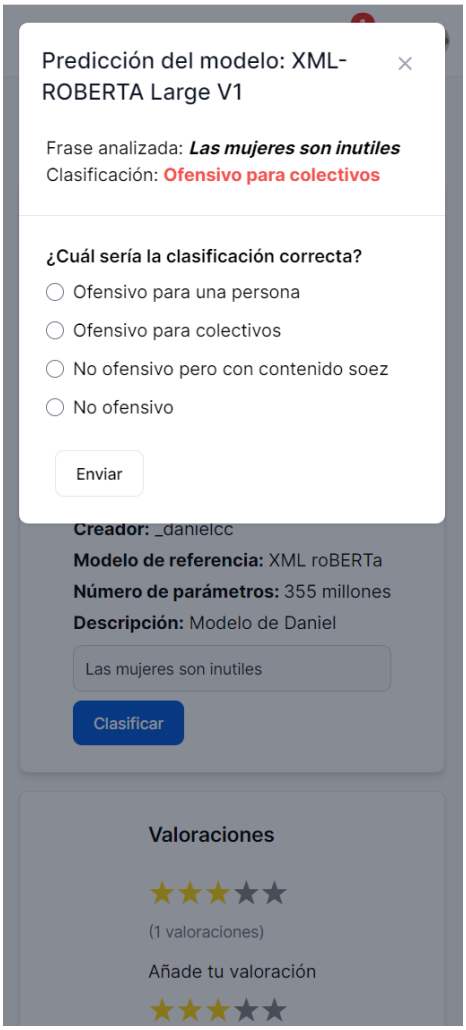


Ilustración 68. Modal de dispositivos móviles

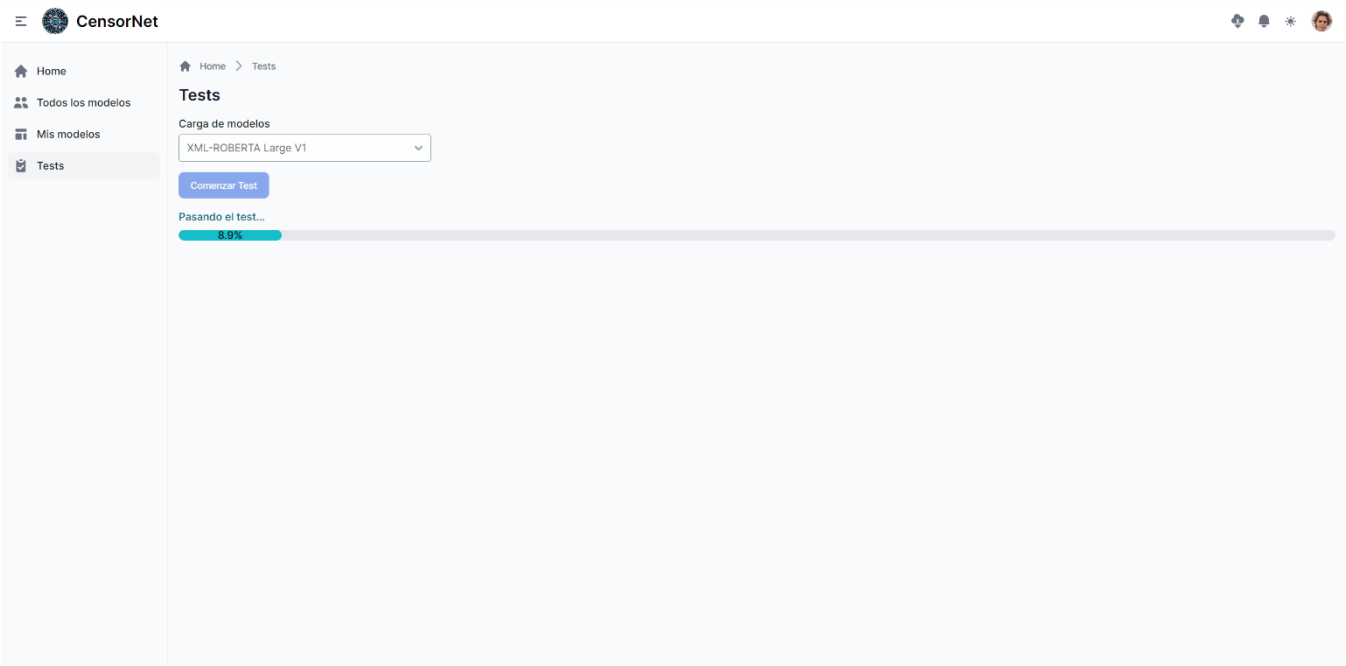


Ilustración 70. Página de obtención de métricas de los modelos en escritorio con selector de modelo y barra de carga

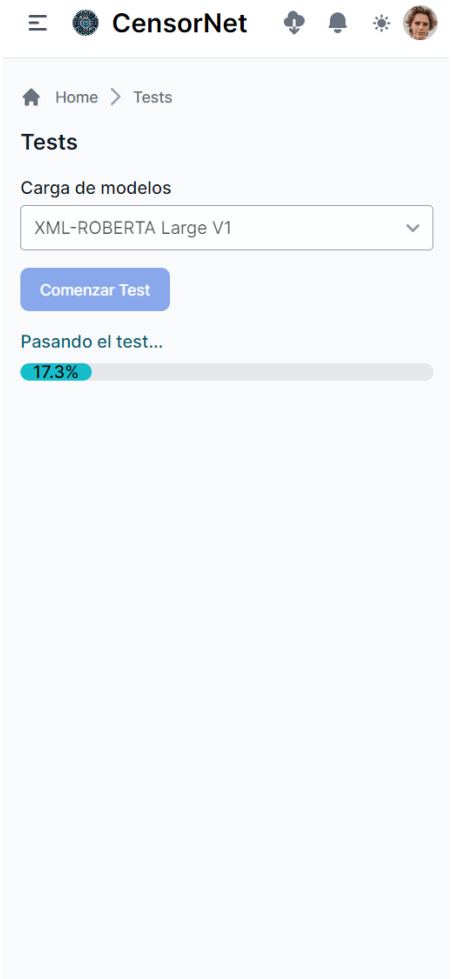
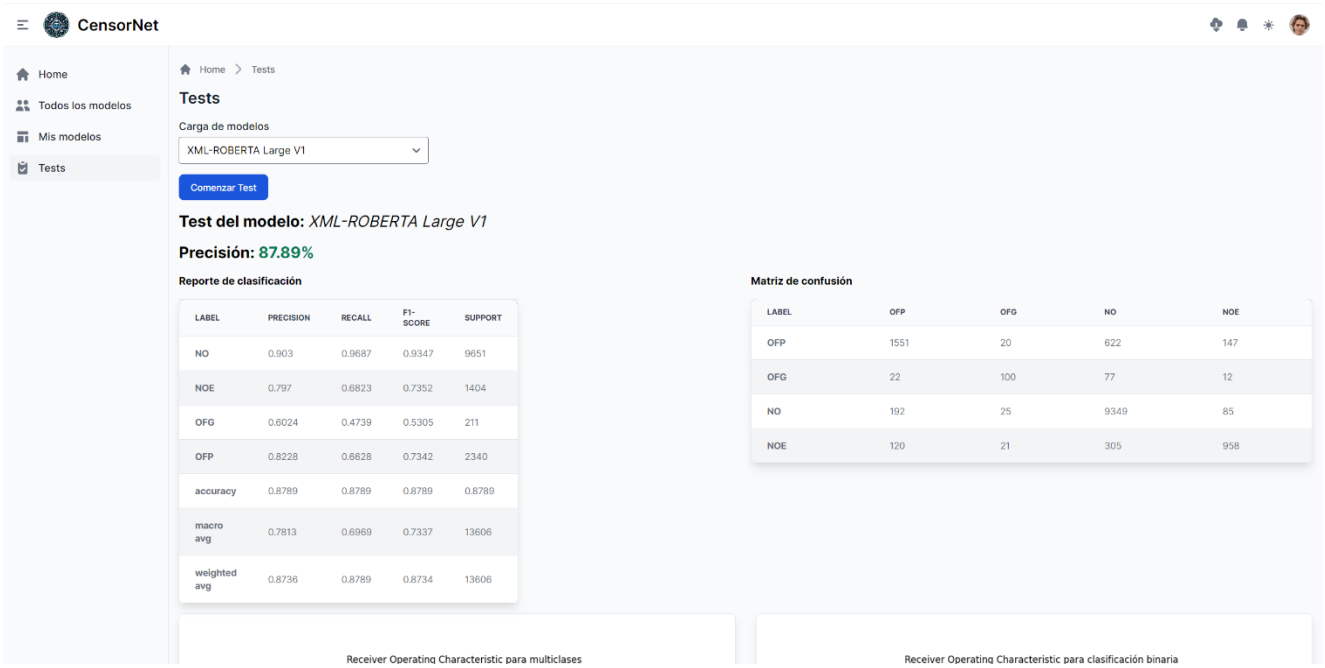


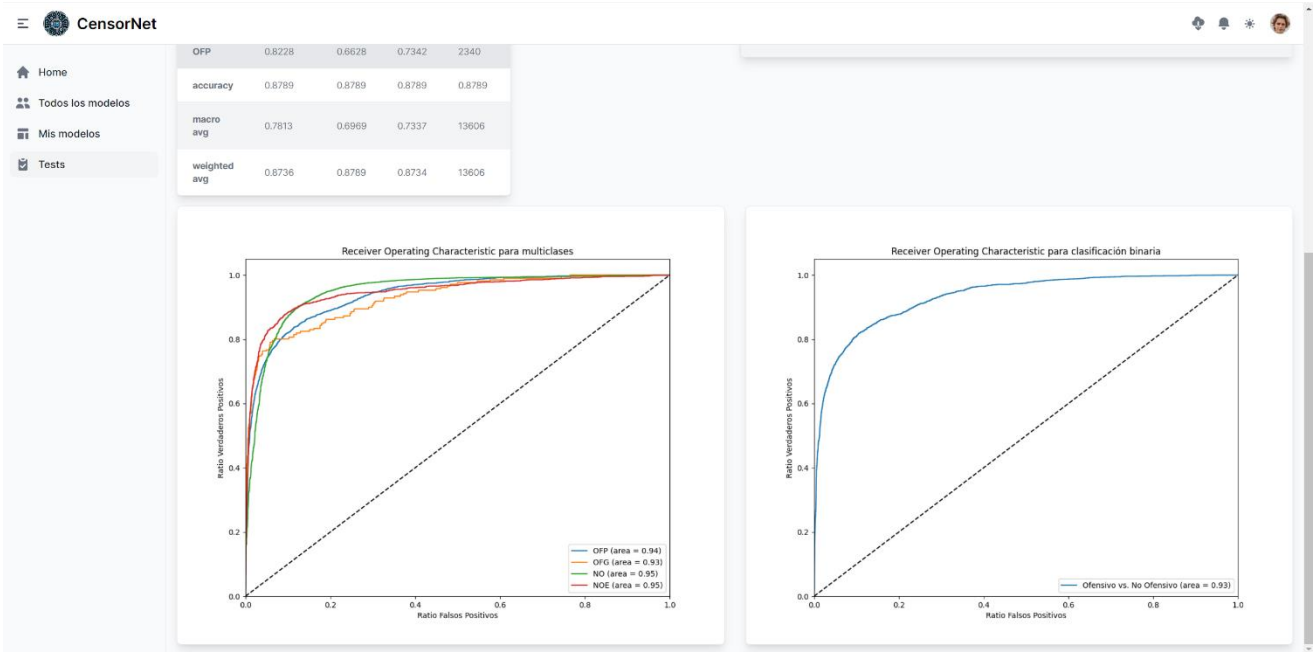
Ilustración 69. Página de obtención de métricas de los modelos en dispositivos móviles



LABEL	PRECISION	RECALL	F1-SCORE	SUPPORT
NO	0.903	0.9687	0.9347	9651
NOE	0.797	0.6823	0.7352	1404
OFG	0.6024	0.4739	0.5305	211
OFP	0.8228	0.6628	0.7342	2340
accuracy	0.8789	0.8789	0.8789	0.8789
macro avg	0.7813	0.6969	0.7337	13606
weighted avg	0.8736	0.8789	0.8734	13606

LABEL	OFP	OFG	NO	NOE
OFP	1551	20	622	147
OFG	22	100	77	12
NO	192	25	9349	85
NOE	120	21	305	958

Ilustración 72. Resultados de la obtención de métricas de reporte de clasificación, matriz de confusión y gráficas AUC-ROC en escritorio



OFP	0.8228	0.6628	0.7342	2340
accuracy	0.8789	0.8789	0.8789	0.8789
macro avg	0.7813	0.6969	0.7337	13606
weighted avg	0.8736	0.8789	0.8734	13606

Ilustración 71. Página de obtención de métricas parte 2

Ilustración 74. Página de obtención de métricas en dispositivos móviles

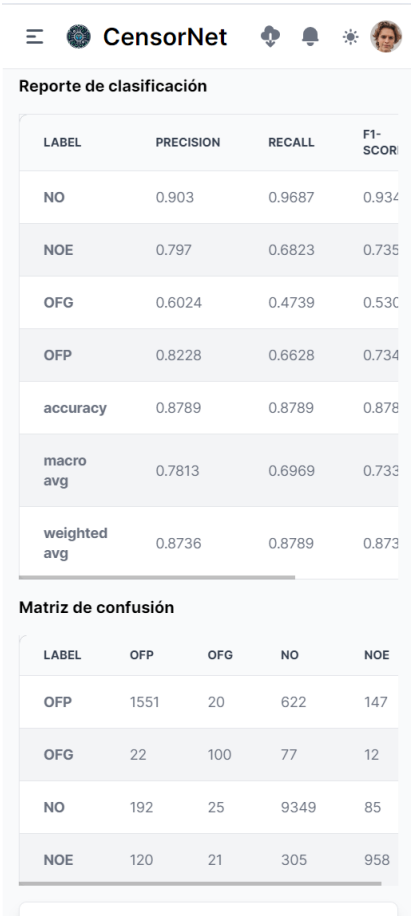


Ilustración 73. Página de obtención de métricas en dispositivos móviles parte 2

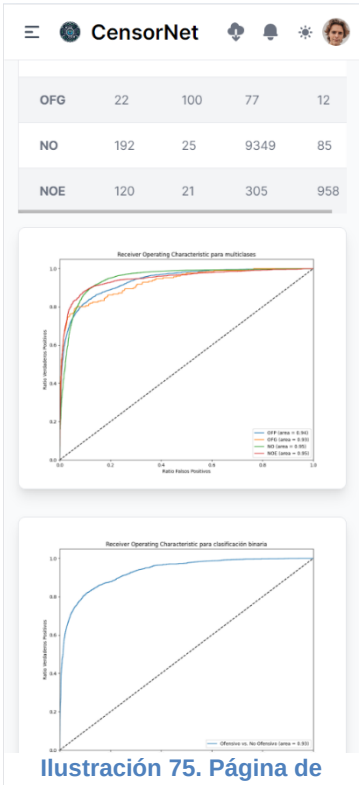


Ilustración 75. Página de obtención de métricas en dispositivos móviles parte 3

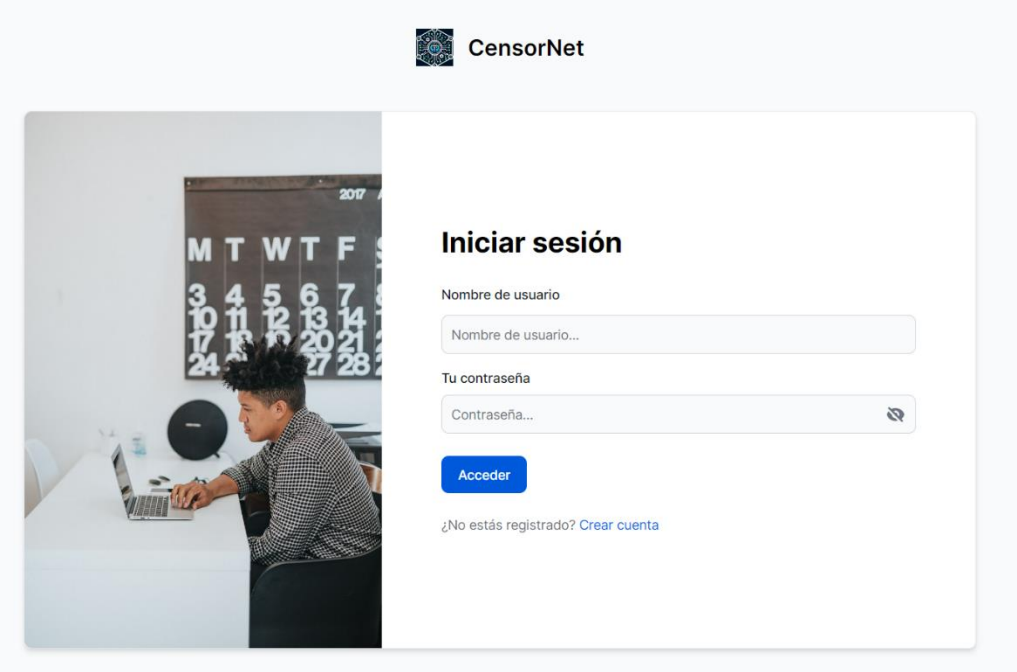


Ilustración 77. Página de login en escritorio

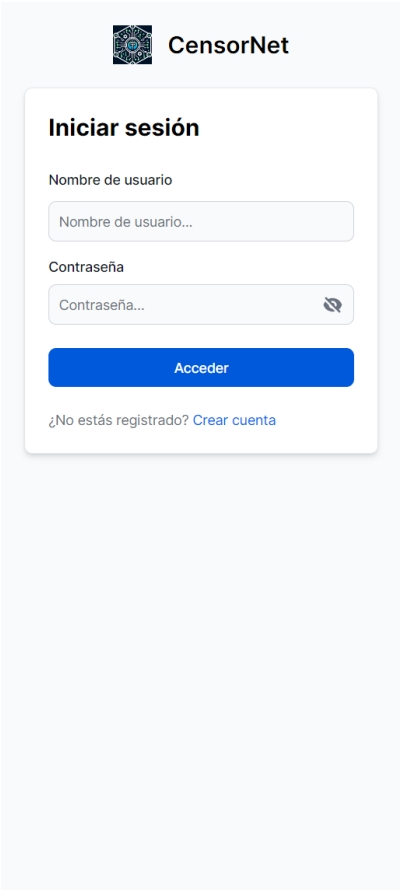


Ilustración 76. Página de login en dispositivos móviles

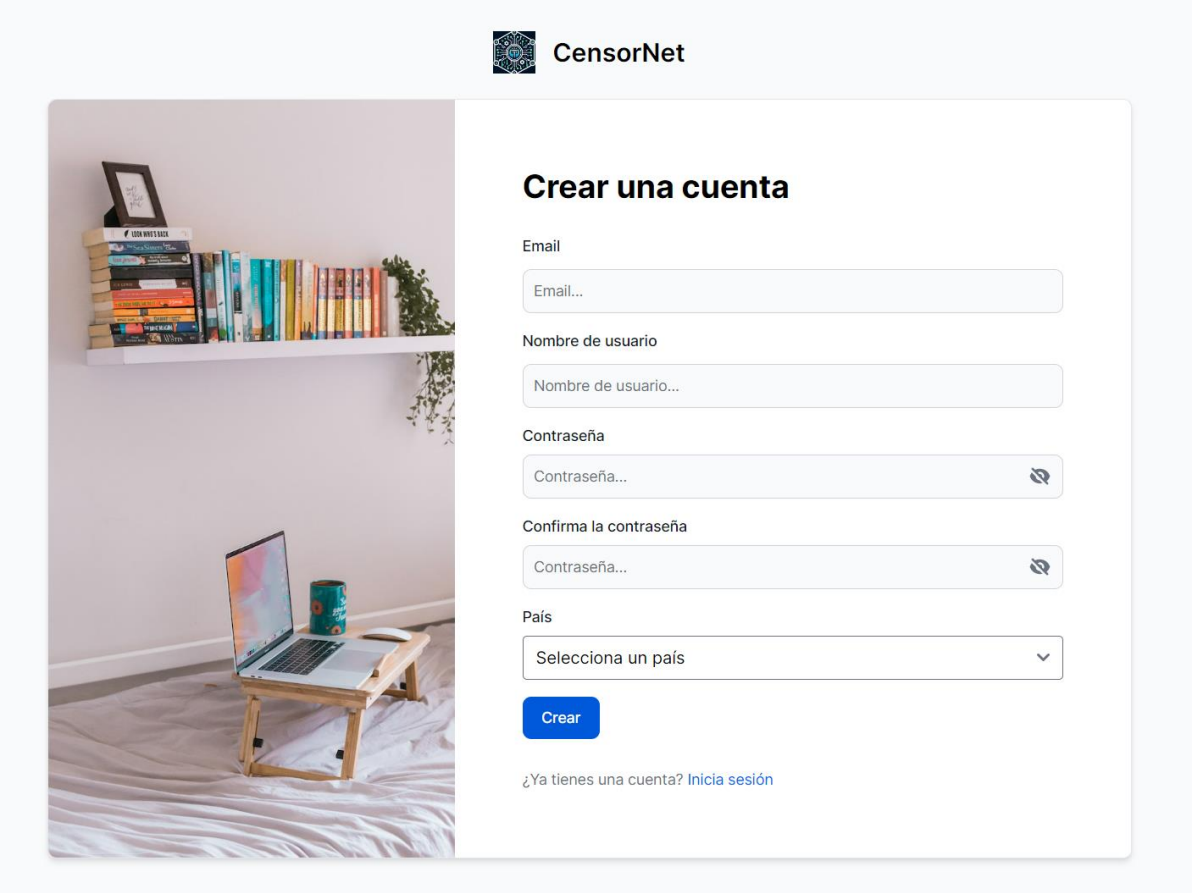


Ilustración 79. Página de registro en escritorio

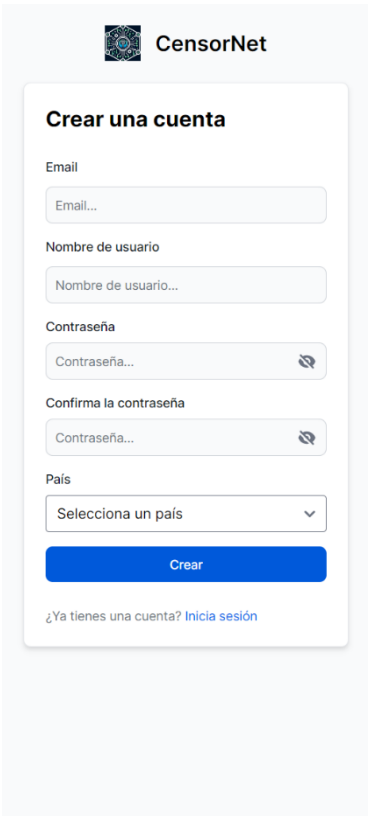


Ilustración 78. Página de registro en dispositivos móviles

Cada una de las interfaces anteriormente expuestas cuenta además con una versión para el modo oscuro, para mostrar esta apariencia se expondrá una única interfaz como ejemplo pues el resto son triviales.

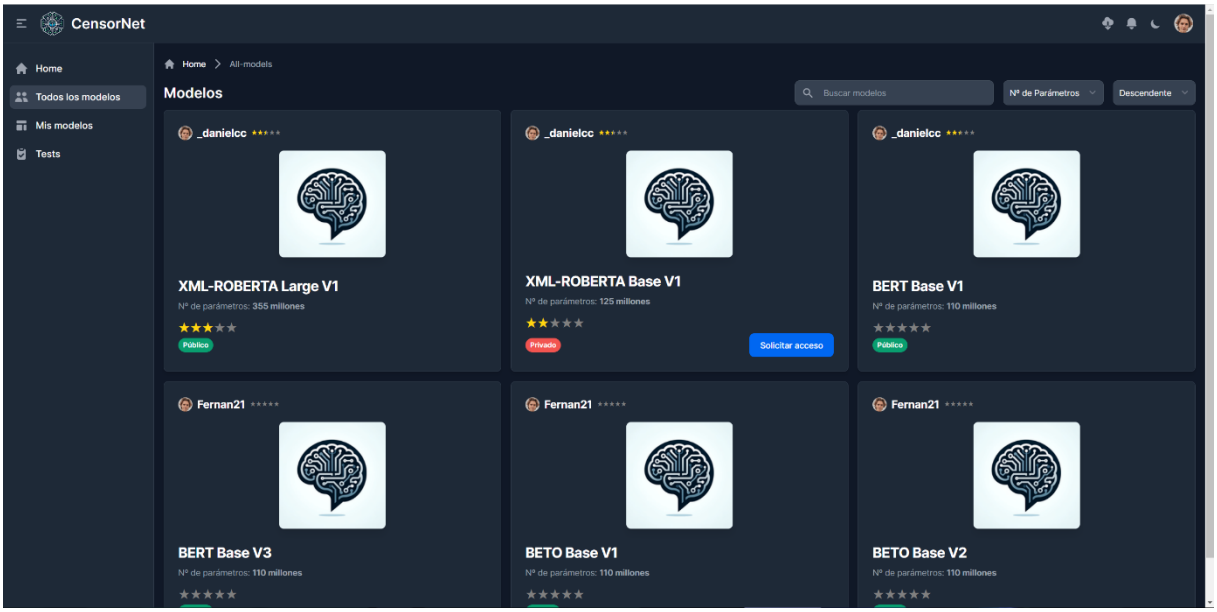


Ilustración 81. Página de listado de modelos con modo oscuro en escritorio

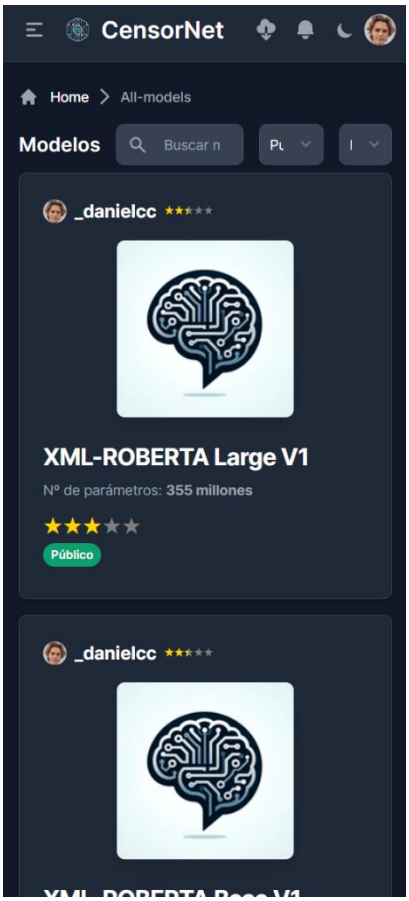


Ilustración 80. Página de listado de modelos con modo oscuro en dispositivos móviles

Una vez descrito el diseño y la información que muestran las diferentes partes de la aplicación, se detalla el funcionamiento básico de los elementos más importantes situados en la barra de navegación de la aplicación. Esta barra se compone principalmente de 6 iconos:

- **Líneas horizontales:** Permiten agrandar o encapsular la barra lateral.
- **Logo de la aplicación:** Permite navegar al menú principal de la aplicación independientemente de la página en la que se encuentre el usuario.
- **Icono de la nube:** Permite descargar el corpus de entrenamiento con los datos que han proporcionado los usuarios en vivo. Este corpus puede ser usado para reentrenar los modelos gracias al feedback de los usuarios.
- **Icono de la campana:** Indica las notificaciones del usuario. Estas notificaciones indican al creador de un modelo si existen usuarios que han solicitado acceder a sus modelos privados. El usuario creador puede permitir o denegar su acceso desde esta sección.
- **Icono del sol/luna:** Indica el modo de visualización de la aplicación. El sol indica que la aplicación se visualiza en modo diurno y la luna en modo nocturno.
- **Icono de perfil:** Muestra la foto de perfil del usuario además de información básica de interés.

Ilustración 82. Iconos del navbar de la aplicación

Aunque el funcionamiento de la mayoría de iconos es trivial, a continuación, se detalla la información mostrada por el icono de las notificaciones y el icono de perfil:

- **Icono de notificaciones:** En la **Ilustración 83** podemos observar cómo el usuario Fernan21 ha enviado una petición de acceso a un modelo privado del usuario que está viendo la notificación (_danielcc). El usuario _danielcc tiene la opción de aceptar o rechazar la solicitud; en ambos casos, la solicitud desaparece hasta nuevo aviso por parte del usuario solicitante. El número situado en la parte arriba a la derecha de la campana indica el número de solicitudes que ha recibido el usuario creador de modelos.

Ilustración 83. Información del icono de notificaciones

- **Icono de perfil:** En la **Ilustración 84** podemos observar la información básica del usuario, como su foto de perfil, su nombre de usuario, correo y unos atajos rápidos para ir al menú principal o cerrar sesión.

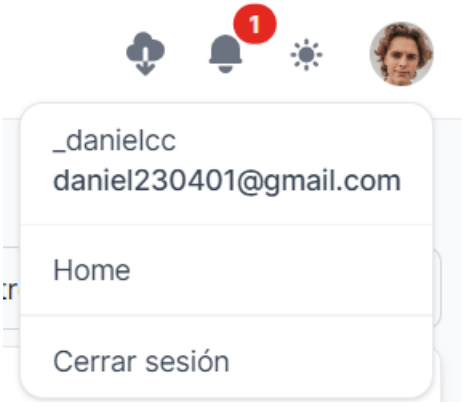


Ilustración 84. Información del icono de perfil

10. Bibliografía

[1] Plaza-del-Arco, F. M., Casavantes, M., Escalante, H. J., Martín-Valdivia, M. T., Montejo-Ráez, A., Montes, M., ... & Villaseñor-Pineda, L. (2021). Overview of MeOffendEs at IberLEF 2021: Offensive language detection in Spanish variants. Procesamiento del Lenguaje Natural, 67, 183-194. (accedido 11 de noviembre de 2023).

[2] Cortes, C. y Vapnik, V. (1995), redes de vectores de soporte, aprendizaje automático, 20, 273-297. (accedido 19 de noviembre de 2023).

[3] Das, B., & Chakraborty, S. (2018). An improved text sentiment classification model using TF-IDF and next word negation. arXiv preprint arXiv:1806.06407. (accedido 23 de noviembre de 2023).

[4] Li, H. (2018). Deep learning for natural language processing: advantages and challenges. National Science Review, 5(1), 24-26. (accedido 4 de diciembre de 2023).

[5] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. (accedido 18 de diciembre de 2023).

[6] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems, 26. (accedido 23 de diciembre de 2023).

[7] Sonbol, R., Rebdawi, G., & Ghneim, N. (2022). The use of nlp-based text representation techniques to support requirement engineering tasks: A systematic mapping review. Ieee Access, 10, 62811-62830. (accedido 19 de diciembre de 2023).

[8] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30. (accedido 1 de diciembre de 2023).

[9] Wang, B., & Klabjan, D. (2017). An attention-based deep net for learning to rank. (accedido 10 de diciembre de 2023).

[10] «BERT en el Natural Language Processing», Codificando Bits, https://www.codificandobits.com/blog/bert-en-el-natural-language-processing/ (accedido 15 de enero de 2024).

[11] «Usuarios de Internet en el Mundo», Marketing4eCommerce, https://marketing4ecommerce.net/usuarios-de-internet-mundo/ (accedido 19 de enero de 2024).

[12] «¿Qué es el discurso de odio?», Ayuntamiento de Barcelona, https://ajuntament.barcelona.cat/bcnvsodi/es/que-es-el-discurso-de-odio/ (accedido 20 de febrero de 2024).

[13] «Metodología CRISP-DM: Ciencia de Datos», Instituto de Ingeniería del Conocimiento, https://www.iic.uam.es/innovacion/metodologia-crisp-dm-ciencia-de-datos/ (accedido 6 de marzo de 2024).

[14] «Diferencias entre desarrollo iterativo e incremental», Sentrio, https://sentrio.io/blog/diferencias-entre-desarrollo-iterativo-e-incremental/ (accedido 9 de marzo de 2024).

[15] «Metodologías ágiles vs tradicionales», Positivo Blog, https://www.positivo.pro/blog/metodologias-agiles-vs-tradicionales/ (accedido 12 de marzo de 2024).

[16] «¿Qué es Kanban?», Asana, https://asana.com/es/resources/what-is-kanban (accedido 14 de marzo de 2024).

[17] «Average Salary of Python Developer in Spain», Salary Explorer, https://www.salaryexplorer.com/average-salary-wage-comparison-spain-python-developer-c203j6273 (accedido 12 de abril de 2024).

[18] «Auditing ChatGPT», Capgemini, https://www.capgemini.com/insights/expert-perspectives/auditing-chatgpt/ (3 de abril de 2024).

[19] «What are the Greenest Programming Languages?», Medium, https://medium.com/codex/what-are-the-greenest-programming-languages-e738774b1957 (accedido 2 de mayo de 2024).

[20] «Java Trends: Top 10 Frameworks of 2020», DevM, https://devm.io/java/java-trends-top-10-frameworks-2020-168867 (accedido 1 de abril de 2024).

[21] «Angular vs React vs Vue», NPM Trends, https://npmtrends.com/angular-vs-react-vs-vue (accedido 2 de marzo de 2024).

[22] «Learn React», React Dev, https://react.dev/learn (accedido 3 de marzo de 2024).

[23] «Introduction to Vue.js», Vue.js, https://vuejs.org/guide/introduction (accedido 2 de marzo de 2024).

[24] «Angular Documentation», Angular, https://v17.angular.io/docs (accedido 4 de marzo de 2024).

[25] «ReactJS: ¿Qué es y cómo funciona?», Deusto Formación, https://www.deustoformacion.com/blog/programacion-tic/reactjs-que-es-funciona. (accedido 3 de marzo de 2024).

[26] «OffendES Dataset», Hugging Face, https://huggingface.co/datasets/fmplaza/offendes (accedido 20 de febrero de 2024).

[27] «RESTful API», RESTful API, https://restfulapi.net/ (accedido 6 de mayo de 2024).

[28] Docker, «Docker Documentation». https://docs.docker.com/ (accedido 23 de mayo de 2024).

[29] «What is Docker», Red Hat, https://www.redhat.com/es/topics/containers/what-is-docker (accedido 23 de mayo de 2024).

[30] «Tailwind UI Documentation», Tailwind UI, https://tailwindui.com/documentation (accedido 10 de marzo de 2024).

[31] «Spring Boot Documentation», Spring, https://spring.io/projects/spring-boot (accedido 6 de abril de 2024).

[32] «Spring Security Documentation», Spring, https://spring.io/projects/spring-security (accedido 8 de abril de 2024).

[33] «Argon2 Documentation», Argon2, https://argon2.online/documentation (accedido 12 de abril de 2024).

[34] «CORS in Spring Boot», Spring, https://spring.io/guides/gs/rest-service-cors/ (accedido 23 de abril de 2024).

[35] «Installing Yarn», Yarn, https://yarnpkg.com/getting-started/install (accedido 1 de marzo de 2024).

[36] «CUDA Documentation», NVIDIA, https://docs.nvidia.com/cuda/ (accedido 19 de enero de 2024).

[37] «Usando CUDA con Python», NVIDIA Developer, https://developer.nvidia.com/how-to-cuda-python (accedido 20 de enero de 2024).

[38] «PyTorch Documentation», PyTorch, https://pytorch.org/docs/stable/index.html (accedido 22 de enero de 2024).

[39] «TensorFlow Documentation», TensorFlow, https://www.tensorflow.org/guide (accedido 22 de enero de 2024).

[40] «Guía de Redux», Redux, https://es.redux.js.org/ (accedido 11 de abril de 2024).

[41] «Documentación de Hooks de React», React, https://es.reactjs.org/docs/hooks-intro.html (accedido 29 de marzo de 2024).

[42] «What is Hate Speech», United Nations, https://www.un.org/es/hate-speech/understanding-hate-speech/what-is-hate-speech (accedido 3 de diciembre de 2023).

[43] «Guía para usar JWT en Spring Boot», Baeldung, https://www.baeldung.com/spring-boot-jwt (accedido 2 de mayo de 2024).

[44] «Cómo funciona Basic Auth», Mozilla Developer Network (MDN), https://developer.mozilla.org/es/docs/Web/HTTP/Authentication (accedido 2 de mayo de 2024).

[45] «Configuración de MySQL en Spring Boot», Spring Framework Guru, https://springframework.guru/configuring-spring-boot-for-mysql/ (accedido 28 de abril de 2024).