Universidad de Cantabria

Facultad de Ciencias

DESARROLLO DE UNA APLICACIÓN PARA LA CREACIÓN DE MODELOS PREDICTIVOS USANDO LLM Y AUTOML
(Development of an application for creating predictive models using LLM and AutoML)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Néstor Miguel Morante

Director: Camilo Palazuelos Calderón

Co-Director: Rafael Duque Medina

Junio– 2025

Agradecimientos

Lo primero que quiero destacar es el compromiso mostrado por los profesores que han aceptado dirigir este TFG, Camilo Palazuelos y Rafael Duque, su implicación, dedicación y compromiso han hecho posible un trabajo que supera mis expectativas iniciales con creces.

Quiero agradecer además a Iván Rivero y Diego García, contratados de investigación de la Universidad de Cantabria, por compartir conmigo sus conocimientos para enriquecer este TFG.

No puedo dejar de hacer referencia a mis amigos, los que han estado conmigo toda la vida y los que he conocido en estos 5 años, todos ellos han estado a mi lado en los buenos y en los malos momentos.

Y por supuesto no puede faltar mi familia, imprescindible para conseguir llegar a la meta, sin ellos, no sería la persona en la que me he convertido.

Resumen

El crecimiento en la complejidad y disponibilidad de datos ha incrementado la demanda de soluciones de aprendizaje automático (ML) accesibles y eficientes. El Aprendizaje Automático Automatizado (AutoML) busca simplificar la creación de modelos mediante la automatización de etapas clave como el preprocesamiento de datos, la selección de algoritmos y la optimización de hiperparámetros. No obstante, las herramientas AutoML tradicionales suelen presentar dificultades para adaptarse a usuarios sin conocimientos técnicos, recursos o interpretar adecuadamente sus intenciones.

Este Trabajo de Fin de Grado presenta un sistema software que integra inteligencia artificial generativa en flujos de trabajo AutoML, permitiendo a los usuarios diseñar modelos predictivos a través de interfaces conversacionales en lenguaje natural. Se proponen dos versiones del sistema: una implementación en la nube, basada en Google Colab y el modelo LLaMA, y otra de ejecución local mediante la plataforma Ollama y el modelo Gemma3. Ambas versiones utilizan modelos de lenguaje para guiar al usuario en la configuración de los datos, la generación automatizada de archivos YAML para Ludwig y la interpretación de resultados.

El sistema se complementa con una interfaz gráfica desarrollada en Streamlit y desplegada mediante Hugging Face Spaces, que facilita la carga de datos, el entrenamiento del modelo y la visualización de métricas de evaluación. La evaluación está basada en diversos conjuntos de datos públicos, demuestra que la herramienta permite generar modelos con rendimiento competitivo, al tiempo que mejora significativamente la accesibilidad y la experiencia del usuario.

Mediante la combinación de AutoML con la asistencia de modelos de lenguaje, el sistema contribuye a adaptar el uso del aprendizaje automático, acercándolo a usuarios sin formación técnica de forma intuitiva.

PALABRAS CLAVE

Aprendizaje automático automatizado, Modelo de lenguaje grande, Interacción persona-ordenador, Asistente virtual conversacional

Abstract

The increasing complexity and availability of data have intensified the demand for accessible and efficient machine learning (ML) solutions. Automated Machine Learning (AutoML) seeks to simplify model development by automating key stages such as data preprocessing, algorithm selection, and hyperparameter tuning. However, traditional AutoML tools often face challenges in adapting to users without technical expertise or resources, and in accurately interpreting their intentions.

This final degree project presents a software system that integrates generative artificial intelligence into AutoML workflows, enabling users to design predictive models through natural language conversational interfaces. Two versions of the system are proposed: a cloud-based implementation using Google Colab and the LLaMA model, and a local version using the Ollama platform and the Gemma3 model.

Both leverage language models to guide users through data configuration, automated generation of YAML files for Ludwig, and result interpretation.

The system is complemented by a graphical interface developed with Streamlit and deployed via Hugging Face Spaces, which facilitates data upload, model training, and evaluation metric visualization. Evaluation based on multiple public datasets demonstrates that the tool produces competitively performing models while significantly enhancing user accessibility and experience.

By combining AutoML with language model assistance, the system helps adapt machine learning usage to non-technical users in an intuitive and user-friendly manner.

KEYWORDS

Automated Machine Learning, Large Language Model, Human-Computer Interaction, Conversation Virtual Assistant

Índice General

Índice de Imágenes

Índice de Tablas

CAPÍTULO 1

1. Introducción

Las metodologías innovadoras y el crecimiento exponencial en la cantidad de datos existentes han provocado avances significativos en el campo del aprendizaje automático (ML, Machine Learning en inglés) en los últimos años y seguirán desarrollándose en el futuro. ML es una rama en evolución de los algoritmos computacionales diseñados para emular la inteligencia humana aprendiendo del entorno [1].

Gracias a estos avances han podido surgir numerosas aplicaciones en campos influyentes en nuestra sociedad como pueden ser: la salud, el marketing, la ingeniería, etc. Incluso con el creciente dominio del uso de las tecnologías de ML, crear modelos predictivos íntegros sigue siendo un reto para personas con pocos recursos o pocos conocimientos sobre programación y ciencia de datos.

A medida que los datos son considerados como un recurso esencial en múltiples sectores, la habilidad para extraer valor de ellos se ha vuelto una competencia muy demandada. Sin embargo, la construcción de modelos de aprendizaje automático sigue estando limitada, en muchos casos, a perfiles altamente especializados, lo que dificulta una adopción más amplia y equitativa de estas tecnologías.

Por otra parte, la complejidad técnica encontrada al desarrollar modelos predictivos representa un obstáculo importante para usuarios no expertos que intentan utilizar ML en sus tareas. Debido a esta situación, actualmente se están presentando numerosos estudios con enfoques que simplifican el proceso, reducen la barrera de entrada, amplían el rango y velocidad de aprendizaje y permiten que cada vez más gente pueda disfrutar de los beneficios del conocimiento que ofrecen los datos [2]

1.1 Motivación

Los Modelos de Lenguaje Grande (LLM, Large Language Model en inglés) y el Aprendizaje Automático Automatizado (AutoML o ML automatizado) son dos avances tecnológicos que han surgido para resolver estos desafíos.

Los LLMs son modelos de redes neuronales de gran escala entrenados con enormes cantidades de texto para realizar tareas de procesamiento de lenguaje natural (PLN, por sus siglas) [6]. Están diseñados para comprender, generar y razonar sobre lenguaje humano sin requerir instrucciones específicas por tarea. Su capacidad emergente proviene de la escala masiva de datos y parámetros usados en su entrenamiento, es como tener un asistente interactivo hecho a medida para tus necesidades. Ejemplos conocidos de LLMs, muy utilizados en la actualidad son GPT-4 [3] , Gemini [4], LLAMA [5].

AutoML por otra parte, es el subcampo del aprendizaje automático que busca automatizar, en cierta medida, todas las etapas del diseño de un sistema de aprendizaje automático. En el contexto del aprendizaje supervisado, AutoML se ocupa de la extracción de características, el preprocesamiento, el diseño de modelos y el posprocesamiento [7] Aunque este proceso reduce tiempo y conocimientos sobre modelos, su utilidad a veces se ve limitada por la creación de pipelines ineficientes y problemas de usabilidad. ¿Qué pasaría si usáramos LLM y AutoML para la creación de modelos? ¿Se solucionarían las limitaciones mencionadas?

Un nuevo enfoque para el desarrollo de ML es traído por la convivencia entre AutoML y LLMs, propuesto en estudios recientes que me parecieron muy interesante para desarrollar mi Trabajo de Fin de Grado (TFG, por sus siglas) [2]

La IA generativa sirve como guía, intérprete y asistente interactivo durante el proceso, llegando a solucionar dudas a los usuarios sin abundante conocimiento, combinando la generación automática de modelos y la interacción en lenguaje natural, facilitando al AutoML la comprensión del proceso para realizar los pasos técnicos del desarrollo del modelo.

Esta combinación permite una transición hacia marcos de ML accesibles y adaptativos. Estos sistemas modifican dinámicamente los flujos de trabajo, clarificando los resultados y asistiendo a los LLMs en la interpretación de la intención del usuario, facilitando el procesamiento de los pipelines y actuando como asistentes interactivos. Facilitan además el trabajo del usuario, convirtiéndolo en intuitivo y eficiente para los usuarios sin formación en aprendizaje automático que no conozcan los términos.

Con un énfasis en ayudar a los usuarios no expertos en el proceso de modelado, este TFG investiga la creación guiada de modelos predictivos mediante inteligencia artificial (IA, por sus siglas) generativa y AutoML.

1.2 Objetivos

El objetivo principal de este TFG es diseñar y desarrollar una herramienta que permita a usuarios sin conocimientos técnicos en programación o ciencia de datos interactuar con sistemas AutoML y construir modelos predictivos mediante una interfaz conversacional basada en LLMs. La herramienta busca facilitar el uso de estas herramientas de forma cómoda, comprensible y guiada.

Para alcanzar este objetivo general, se definen los siguientes subobjetivos:

- **Diseñar e implementar una interfaz conversacional** apoyada por un asistente virtual basado en LLMs, que permita a los usuarios sin recursos y sin conocimientos interactuar de manera natural con el sistema, pudiendo crear y

entrenar un modelo a partir de un conjunto de datos, y recibir orientación a lo largo del proceso.

- **Integrar técnicas de AutoML** que automaticen la selección de algoritmos e hiperparámetros, el preprocesamiento de datos y la optimización de modelos según los requerimientos del usuario.

- **Traducir objetivos del usuario en tareas de aprendizaje automático** (clasificación, regresión) mediante procesamiento del lenguaje natural, ayudando a formular correctamente el problema y a verificar que el sistema realiza técnicamente lo que el usuario desea, convirtiendo así lenguaje natural en lenguaje técnico y al revés.

- **Evaluar el rendimiento de la herramienta desarrollada** mediante la comparación con conjuntos de datos públicos, analizando la eficacia de la interacción y la calidad de los modelos generados.

Este trabajo tiene como objetivo abstraer la complejidad técnica del modelado predictivo y proporcionar una experiencia guiada y explicable para los usuarios, apoyando estas tareas con una interfaz conversacional respaldada por AutoML y potenciada por LLMs. El objetivo final es aumentar la competencia y la confianza del usuario al interactuar con herramientas impulsadas por IA, además de reducir las barreras de entrada

1.3 Estructura del resto del documento

El trabajo consta de 5 capítulos, además del capítulo actual, el resto son los siguientes:

- **Capítulo 2 - Materiales y métodos:** Se explican las herramientas y tecnologías utilizadas para el desarrollo del chatbot y la interfaz gráfica, así como la metodología de trabajo seguida. También se incluye un cronograma de desarrollo representado mediante un diagrama de Gantt.

- **Capítulo 3 – Desarrollo de las herramientas**: Se describe el diseño e implementación de las herramientas programadas. Se explica el flujo de interacción para construir modelos ML, la integración del asistente virtual basado en IA generativa y la interfaz de usuario utilizada para simplificar el entrenamiento de modelos.

- **Capítulo 4 – Evaluación**: Se describen los conjuntos de datos utilizados, tanto de clasificación como de regresión, presentando además los resultados obtenidos. Posteriormente, validamos la herramienta comparando las métricas con las de otros autores.

- **Capítulo 5 – Conclusiones**: Reflexión sobre el grado de consecución de los objetivos respecto a los objetivos iniciales presentados en el capítulo actual. Además, se identifican posibles mejoras y líneas de trabajo futuras, especialmente en lo relativo a la aplicabilidad del modelo y a una interacción más rica y personalizada con el usuario.

Esta estructura permite abordar de forma progresiva tanto los fundamentos conceptuales como la implementación práctica de la propuesta, facilitando su comprensión y asegurando una visión global del trabajo.

CAPÍTULO 2

2. Materiales y métodos

Este proyecto integra diversas plataformas y herramientas seleccionadas y combinadas adecuadamente para capacitar a individuos con escasa formación técnica en la creación, entrenamiento y optimización de modelos predictivos. Desde el procesamiento de datos inicial hasta la evolución de los resultados, cada tecnología contribuye una etapa específica del flujo de trabajo, creando un entorno cohesivo que respalda el objetivo del TFG, afianzar el aprendizaje automático con la combinación de AutoML y la IA generativa.

2.1 Herramientas y tecnologías

2.1.1 Ludwig

Ludwig es una herramienta de Aprendizaje Automático Automatizado (AutoML) de código abierto desarrollada por Uber, diseñada para facilitar el entrenamiento de modelos [14]

Ludwig se basa en un enfoque declarativo, lo que significa que el usuario solo necesita describir el problema y los datos mediante un archivo de configuración, sin necesidad de escribir código específico para definir arquitecturas o funciones de entrenamiento. Este archivo de configuración está escrito en YAML, un formato de texto sencillo y legible por humanos que se utiliza para estructurar datos jerárquicos. YAML permite especificar qué tipo de datos se utilizarán como entrada y salida, qué arquitectura de red se empleará, cómo se entrenará el modelo y qué métricas se evaluarán [15].

Estas son sus secciones principales y sus propósitos generales:

- **input_features:** Define una lista de características que se utilizarán como entrada del modelo. Cada feature debe indicar al menos un nombre (name) y un tipo (type), como number, category, text, etc. Ludwig utiliza esta información para decidir qué arquitectura de red aplicará a cada tipo de dato.

```
input_features:
  - name: Clump_Thickness
    type: number
  - name: Cell_Size_Uniformity
    type: number
  - name: Cell_Shape_Uniformity
    type: number
# ...resto de atributos que queremos usar como
entrada...
```

- **output_features:** Indica las variables que el modelo debe predecir, etiquetas o valores objetivo. Al igual que las de entrada, cada una debe tener su nombre y tipo, como category para clasificación o number para regresión.

```
output_features:
  - name: Class
     type: category
```

- **preprocessing (opcional):** Permite configurar estrategias de preprocesamiento antes del entrenamiento, como el tratamiento de valores nulos (missing_value_strategy) o el separador de columnas en archivos CSV (Comma-Separated Values en inglés).

```
preprocessing:
  missing_value_strategy: drop_row
  separator: comma
```

- **combiner (opcional):** Define cómo se integran las salidas de las distintas entradas antes de llegar a la parte final del modelo.

```
combiner:
  type: tabnet
```

- **trainer (opcional):** Establece los parámetros del proceso de entrenamiento:

```
trainer:
  batch_size: auto
  learning_rate: 0.001
  learning_rate_scaling: sqrt
  learning_rate_scheduler:
    decay: exponential
    decay_rate: 0.8
    decay_steps: 20000
  optimizer:
    type: adam
  validation_field: class
  validation_metric: accuracy
```

- **hyperopt (opcional):** Define una estrategia de optimización automática de hiperparámetros.

Una de las ventajas clave de Ludwig es su enfoque basado en tipos de datos. Esto le permite adaptarse automáticamente a tareas como clasificación, regresión, análisis o

generación de texto, simplemente reconociendo el tipo de variable definida. Además, Ludwig proporciona herramientas integradas para la visualización de resultados, comparación de modelos y análisis de errores, lo que facilita la experimentación rápida y comprensible incluso para usuarios no expertos [15].

Lo único que necesitamos para el correcto funcionamiento del Ludwig, es el conjunto de datos y un archivo de configuración YAML, que en caso de nuestro TFG fue creado con la ayuda de la IA generativa. La función principal de Ludwig en nuestro proyecto será el entrenamiento del modelo para calcular una serie de métricas que podamos comparar con las calculadas con otros autores y así poder validar nuestras herramientas.

2.1.2 LLAMA

LLaMA (Large Language Model Meta AI en inglés) es una familia de modelos de lenguaje de IA, destaca la capacidad de compresión e interacción con el lenguaje natural [8] . Está basado en la arquitectura transformer, entrenado con gran cantidad de datos y capaz de realizar tareas como generación de texto coherente y contextual, resumen de textos extensas, asistencia en tareas, etc [9]

Específicamente, la versión empleada en este proyecto es LLaMA-2-7B-Chat, es un modelo conversacional optimizado con siete mil millones de parámetros. Para proyectos que buscan integrar comprensión del lenguaje natural en tiempo real es capaz de realizar tareas fundamentales sin tener que sufrir los costes de recursos asociados con modelos más grandes como GPT-4 u otros modelos de LLaMA, ofreciendo un equilibrio adecuado entre rendimiento, eficiencia y disponibilidad [9]

En este TFG, LLaMA-2-7B-Chat es fundamental como asistente de IA generativa. LLaMA interpreta entradas en lenguaje natural, respondiendo dudas del usuario sobre términos técnicos y ofreciendo recomendaciones contextuales para la configuración del YAML (Yet Another Markup Language en inglés), en función de las necesidades del usuario y optimización del modelo, funciona como un nexo entre el usuario y el pipeline de AutoML, utilizando un prompt que configuramos en función de la necesidad del modelo. Facilita que los usuarios no técnicos interactúen con el flujo de trabajo de aprendizaje automático mediante interacción en lenguaje natural. Su inclusión optimiza significativamente la usabilidad del sistema. Además de funcionar como una interfaz conversacional, LLaMA actúa como un asistente interactivo que fomenta la explicabilidad y la mejora continua del modelo.

2.1.3 Gemma3 + OLLAMA

Gemma3 es un modelo que integra múltiples tipos de datos (texto, imágenes, etc.), lo que la convierte en una solución multimodal dentro de la familia Gemma, que incluye modelos abiertos y ligeros, con una escala que va desde 1 hasta 27 mil millones de parámetros [10]. Esta versión introduce capacidades de comprensión visual, una

cobertura más amplia de idiomas y un contexto más largo, de al menos 128K tokens [6].

Dado su tamaño y capacidades avanzadas, Gemma3 requiere recursos computacionales significativos para su ejecución eficiente, es decir ordenadores potentes con una o varias unidades de procesamiento gráfico (GPU, Graphics Processing Unit en inglés), gran cantidad de memoria RAM (Random Access Memory en inglés) y suficiente capacidad de almacenamiento para guardar el modelo. Estos requisitos aseguran que las respuestas generadas sean rápidas y que el modelo pueda manejar interacciones complejas sin interrupciones ni cuellos de botella.

Para facilitar la ejecución de modelos como Gemma 3 en entornos locales, he utilizado Ollama, una plataforma que permite a los usuarios descargar y ejecutar modelos de lenguaje grandes directamente en computadoras personales, sin depender de servicios en la nube [12]. Ollama proporciona una interfaz de línea de comandos y una interfaz de programación de aplicaciones (API, por sus siglas en inglés), facilitando la interacción con diversos modelos [13]

En resumen, la combinación de Gemma 3 y Ollama proporciona una solución poderosa y flexible para ejecutar modelos de lenguaje avanzados localmente, aprovechando al máximo los recursos disponibles y garantizando el control sobre los datos, así que, en caso de tener los recursos necesarios para utilizar este modelo, sería muy positivo sustituir LLAMA por Gemma3 y que este realice las funcionalidades que antes hacía LLAMA y fuese nuestro nueva IA generativa, maximizando la velocidad y eficacia de la ejecución.

2.1.4 Google Colab

Los usuarios pueden programar y ejecutar código Python en una interfaz de cuaderno Jupyter mediante Google Colaboratory (Colab), un entorno de desarrollo integrado en la nube [16] Es especialmente útil para usuarios que no tienen hardware especializado o instalación local, proporciona acceso a recursos computacionales de alto rendimiento, tales como GPUs y TPUs (Tensor Processing Unit en inglés). Colab ha ganado popularidad en el uso del aprendizaje automático debido a su comodidad y facilidad de uso para usuarios nuevos en el campo y sin alta capacidad de recursos y acceso gratuito a una infraestructura robusta, mucho más barato que mejorar el local [17].

Las exigencias computacionales para implementar LLaMA-2-7B-Chat hicieron imprescindible el uso de Colab en este proyecto, ya que mi ordenador no disponía de los recursos necesarios para una ejecución fluida y rápida. El modelo completo de LLAMA fue almacenado en Google Drive, se vinculó con Google Colab y se implementó en una instancia de GPU T4 mediante scripts de Python. La respuesta en tiempo real se logró gracias a esta configuración, que también mejoró considerablemente la capacidad de respuesta del sistema.

Así mismo, el entorno de ejecución del backend del proceso de AutoML es Colab. Implementa la IA generativa necesaria, procesa el archivo YAML ofreciendo una interfaz cómoda al usuario, además de establecer comunicación con el LLM para ofrecer aclaraciones y observaciones. Colab es un elemento fundamental de la implementación por su capacidad de ejecución independientemente de los recursos del usuario y por permitir la edición y ejecución compartida del mismo cuaderno entre varios usuarios en tiempo real.

2.1.5 Streamlit + Hugging face

Streamlit es un framework de código abierto en Python diseñado para facilitar el desarrollo de aplicaciones web interactivas orientadas al análisis de datos y ML [18] La interfaz intuitiva y fácil de usar de Streamlit te permite convertir scripts de datos en aplicaciones web compatibles de forma rápida y sencilla. Ofrece una amplia variedad de widgets (como sliders, selectores y botones) que permiten a los usuarios interactuar directamente con los datos o parámetros del modelo. Con solo unas pocas líneas de código, puedes crear paneles de datos interactivos y visualizaciones de datos que fomentan la interacción cómoda del usuario [19].

Hugging Face es una plataforma de código abierto que actúa como repositorio central para modelos de aprendizaje automático, facilitando su desarrollo, compartición y despliegue, proporciona acceso a una amplia variedad de modelos preentrenados y promueve la reutilización colaborativa de estos a través de herramientas accesibles como *Hugging Face Spaces*, que permite alojar aplicaciones web interactivas [20] [21].

En este proyecto se ha utilizado Streamlit dentro de un Space de Hugging Face para facilitar el despliegue y acceso a la aplicación desarrollada. Esta combinación ofrece ventajas como el alojamiento gratuito, facilidad de acceso a la aplicación y recursos gratuitos.

La interfaz gráfica de usuario (GUI, Graphical User Interface en inglés) empleada por los usuarios para interactuar con el sistema se elabora en este proyecto mediante Streamlit. Esta herramienta permite a los usuarios cargar el conjunto de datos y el YAML devuelto por la ejecución de Google Colab y visualizar los resultados mediante la aplicación.

2.1.6 Python

Python es el lenguaje de programación sobre el que se desarrollan todas las herramientas utilizadas en este proyecto [22]. Aunque muchas de ellas, como Ludwig o Streamlit, permiten evitar el uso de programación por parte del usuario mediante interfaces visuales, su ejecución depende de un entorno Python previamente programado por parte del desarrollador.

En este TFG, Python ha sido esencial para ejecutar los distintos componentes del sistema. Por ejemplo, Ludwig se lanza desde scripts de Python que procesan archivos

YAML y los datasets; la interfaz gráfica fue desarrollada con Streamlit, una librería de Python; y toda la ejecución en Google Colab se realiza en notebooks Python. Además, la comunicación entre el archivo de configuración, el modelo LLaMA y el flujo AutoML también se organiza mediante scripts escritos en este lenguaje.

2.2 Metodología de trabajo

La metodología seguida para el desarrollo del TFG se basa en una estructuración por bloques de trabajo (WP, *Work Packages* en inglés), que permite abordar el proyecto de manera ordenada y controlada. Esta división favorece la planificación temporal, la gestión eficiente y el seguimiento del progreso en cada fase del proyecto.

1.4.1 Organización temporal

El trabajo se desarrolló entre el 1 de febrero y el 27 de junio de 2025, siendo un total de 21 semanas.

La planificación se ha representado mediante un diagrama de Gantt mostrado en la Ilustración 1, que facilita el seguimiento cronológico del proyecto. Este diagrama resulta especialmente útil para mantener una visión global del progreso.

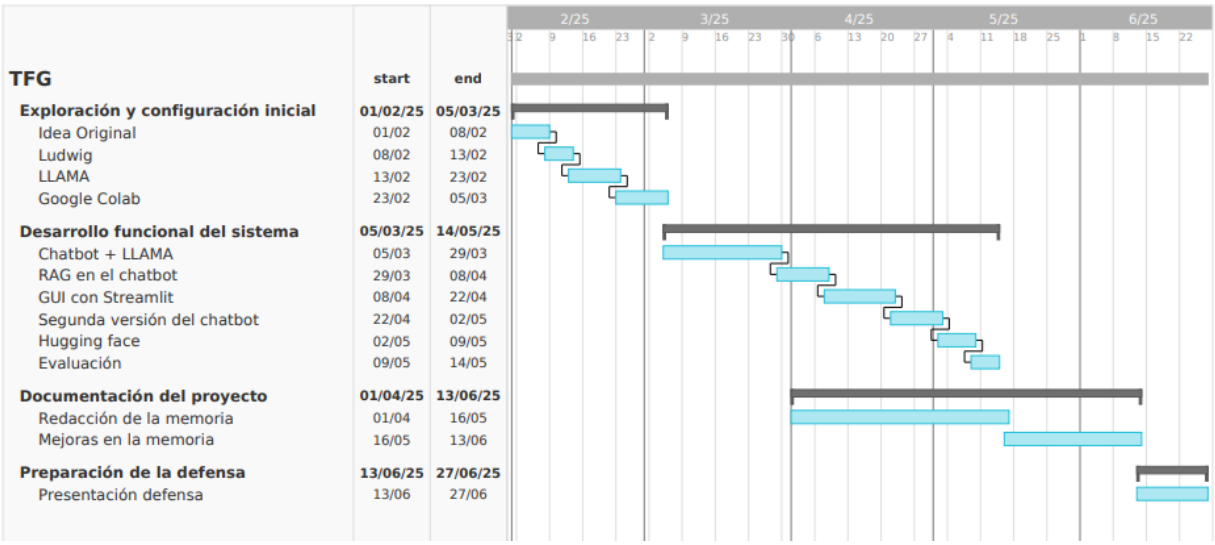

Ilustración 1. Diagrama Gaant del TFG.

1.4.2 Metodología de bloques de trabajo

El desarrollo del proyecto se organizó en cuatro bloques de trabajo detallados a continuación en las siguientes tablas:

WP1 – La Tabla 1 muestra la exploración y configuración inicial

ID	Descripción Tarea	Fecha Inicio	Fecha Fin	Duración
WP1	Asentar la base conceptual y técnica del proyecto, así como familiarizarse con las herramientas que utilizaremos en nuestro proyecto	01/02/2025	05/03/2025	32 días
T1.1	Idea Original	01/02/2025	08/02/2025	7 días
T1.2	Familiarización con Ludwig	08/02/2025	13/02/2025	5 días
T1.3	Ejecución inicial de llama en mi local	13/02/2025	23/02/2025	10 días
T1.4	Resolución de problemas, ejecución de LLAMA en Google Colab por la falta de recursos.	23/02/2025	05/03/2025	10 días

Tabla 1. WP1

WP2 – La Tabla 2 expone el desarrollo funcional del sistema

ID	Descripción Tarea	Fecha Inicio	Fecha Fin	Duración
WP2	Implementar y desarrollar las funcionalidades clave de nuestro TFG.	05/03/2025		97 días
T2.1	Integración chatbot + LLAMA en Google colab	05/03/2025	29/03/2025	24 días
T2.2	Implementación del sistema RAG en el chatbot	29/03/2025	08/04/2025	10 días
T2.3	Desarrollo de interfaz gráfica (GUI) con Streamlit para evaluar los modelos con Ludwig	08/04/2025	22/04/2025	14 días
T2.4	Integración de una segunda versión del chatbot	22/04/2025	02/05/2025	10 días
T2.5	Integración de la interfaz gráfica en hugging fase.	02/05/2025	09/05/2025	7 días
T2.6	Evaluación de los conjuntos de datos y validación de las herramientas	09/05/2025	14/05/2025	5 días

Tabla 2. WP2

WP3 – La Tabla 3 exhibe la documentación del proyecto

ID	Descripción Tarea	Fecha Inicio	Fecha Fin	Duración
WP3	Redactar y revisar la memoria del TFG	01/04/2025	13/06/2025	73 días
T3.1	Redacción de la memoria	01/04/2025	16/05/2025	45 días
T3.2	Mejoras en la memoria	16/05/2025	13/06/2025	28 días

Tabla 3. WP3

WP4 – La Tabla 4 relacionada con la preparación de la defensa

ID	Descripción Tarea	Fecha Inicio	Fecha Fin	Duración
WP4	Elaboración y ensayo de la presentación oral del trabajo	13/06/2025	27/06/2025	14 días
T4.1	Práctica de la presentación del TFG.	13/06/2025	27/06/2025	14 días

Tabla 4. WP4

CAPÍTULO 3

3. Desarrollo de la herramienta

En este capítulo se describe el desarrollo de la herramienta diseñada para facilitar la creación de modelos de AutoML mediante el uso de asistentes virtuales basados en LLM, así como la explicación de las distintas versiones implementadas, sus componentes técnicos y las funcionalidades integradas. Finalmente, se explica la interfaz gráfica que permite a usuarios sin conocimientos avanzados entrenar modelos con facilidad. A continuación, muestro una imagen para poder comprender la estructura de nuestro sistema, para entender mejor la arquitectura y modularidad del TFG.

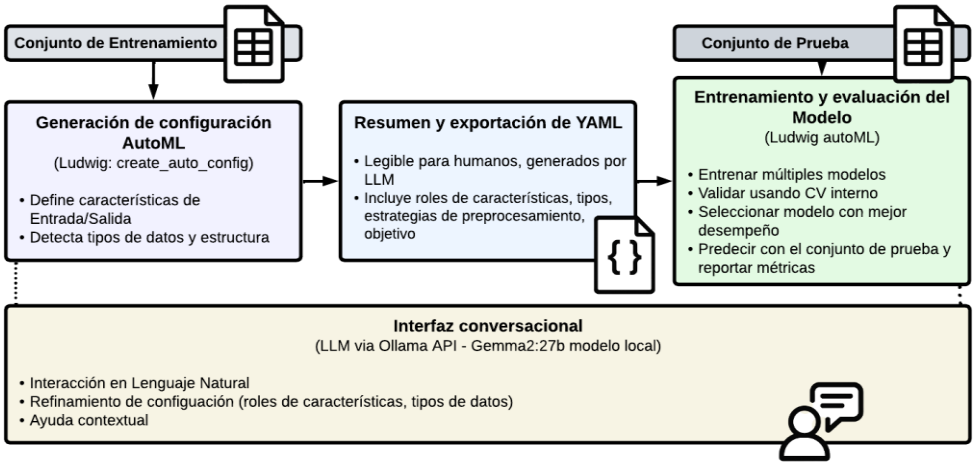

Ilustración 2. Arquitectura del sistema implementado.

La Ilustración 2 presenta la arquitectura general del sistema desarrollado. Esta se basa en un flujo que parte de un conjunto de datos de entrenamiento y, mediante una serie de pasos automáticos y asistidos, permite generar configuraciones en formato YAML para entrenar modelos utilizando Ludwig.

El proceso comienza con la generación automática de la configuración (AutoML config generation) utilizando Ludwig (create_auto_config), que define las características de entrada y salida, y detecta tipos de datos. Luego, esta configuración es resumida y exportada en formato YAML legible para humanos, donde se incluyen roles de características, tipos, estrategias de preprocesamiento y el objetivo del modelo. A continuación, el sistema entrena y evalúa múltiples modelos, seleccionando el de mejor desempeño y reportando métricas sobre el conjunto de prueba.

Todo este flujo está asistido mediante una interfaz conversacional basada en LLMs (utilizando Ollama API y el modelo local Gemma-27b), que permite al usuario interactuar en lenguaje natural.

Esta arquitectura modular permite ofrecer una solución flexible, que se adapta tanto a usuarios con recursos limitados como a aquellos que requieren mayor eficiencia.

3.1 Flujo de interacción para construir modelos ML

En esta sección se presentan dos versiones del flujo conversacional desarrollado para asistir al usuario en la configuración de modelos AutoML, guiándolo paso a paso hasta la generación del archivo de configuración .yaml.

3.1.1 Versión 1: Chatbot en Google Colab con modelo LLaMA y RAG local

Esta versión está diseñada para ejecutarse en Google Colab y utiliza el modelo LLaMA 2-7B-chat-hf, cargado de forma local desde el Google Drive. El entorno Google Colab permite una integración sencilla entre almacenamiento en la nube y ejecución rápida sin necesidad de tener un ordenador potente. La Ilustración 3 muestra el punto de vista del desarrollador, la herramienta se ejecuta dentro de un entorno de Google Colab donde podemos observar las celdas de código que instancian el modelo.

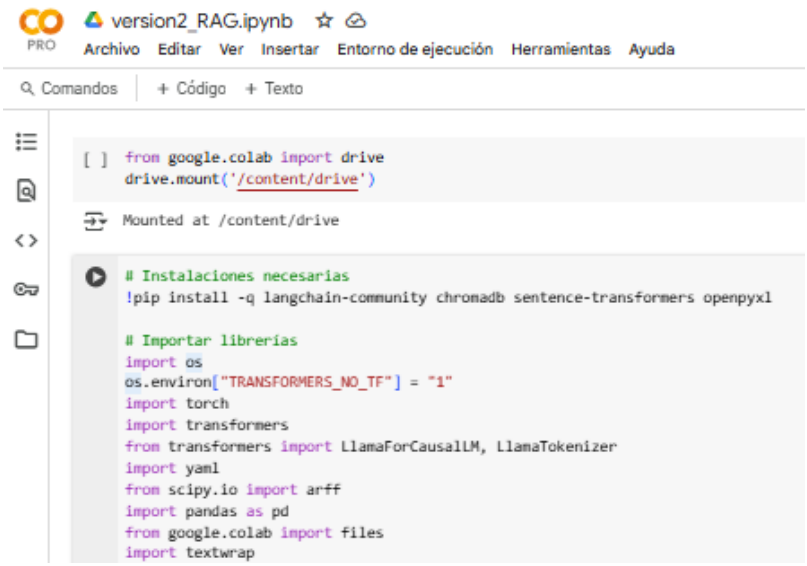

Ilustración 3. Vista del desarrollador.

Además, se incorpora un mecanismo de RAG, que permite al usuario realizar consultas en lenguaje natural durante el proceso. Este enfoque mejora las respuestas de la IA generativa proporcionando respuestas contextualizadas basadas en un .csv con preguntas y respuestas previamente construido.

Al iniciarse, en la Ilustración 4 podemos ver que el chatbot da la bienvenida al usuario y le explica cómo activar el modo de ayuda utilizando el prefijo ayuda: seguido de la pregunta. A continuación, solicita al usuario que cargue el conjunto de datos con el que desea trabajar:

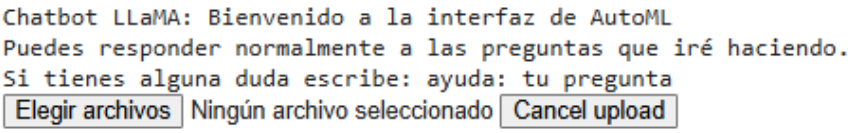

Ilustración 4. Versión 1 del sistema: Mensaje de bienvenida.

Una vez que el usuario ha cargado el archivo desde su sistema local mediante el explorador de archivos de Google Colab, la herramienta muestra las columnas presentes en el conjunto de datos.

A partir de esta información, observamos en la Ilustración 5 e Ilustración 6 que se inicia el proceso de configuración con las primeras preguntas relacionadas con el preprocesamiento:

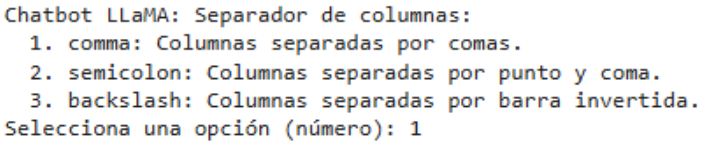

Ilustración 5. Versión 1 del sistema: Elección del separador de columnas.

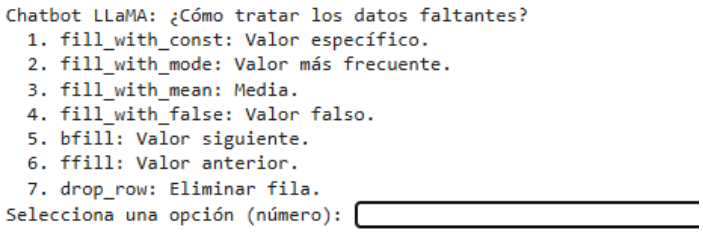

Ilustración 6. Versión 1 del sistema: Elección tratamiento de datos faltantes.

En caso de que el usuario no sepa cómo contestar alguna de las preguntas, como por ejemplo el tratamiento de los datos faltantes, puede activar el asistente virtual escribiendo el prefijo ayuda: seguido de su consulta. Esto permite a la IA generativa proporcionar una explicación contextualizada utilizando información relevante previamente cargada en el .csv mediante RAG, que podemos apreciar en la Ilustración 7.

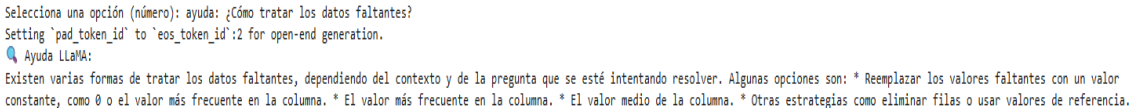

Ilustración 7. Versión 1 del sistema: Función de Ayuda.

Esta respuesta generada por el modelo LLaMA facilita la toma de decisiones al usuario, especialmente si no cuenta con conocimientos técnicos previos. Tras resolver las preguntas generales de preprocesamiento, el asistente continúa con la configuración específica de cada columna.

Para cada una, se solicita al usuario que indique el rol que desempeñará en el modelo (entrada, salida o ninguna). En el caso de las columnas seleccionadas como entrada o salida, se pregunta además por el tipo de dato en la Ilustración 8 e Ilustración 9.

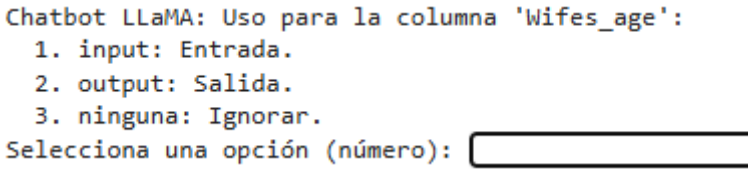

Ilustración 8. Versión 1 del sistema: Elección del rol de cada columna.

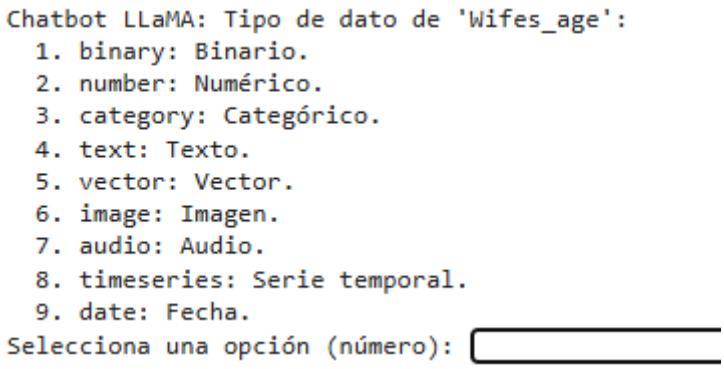

Ilustración 9. Versión 1 del sistema: Elección del tipo de cada columna.

Tras completar la configuración de todas las columnas, la IA generativa elabora un resumen automático en lenguaje natural mostrado en la Ilustración 10, totalmente comprensible para el usuario. Este resume el objetivo del modelo y describe brevemente el archivo de configuración creado.

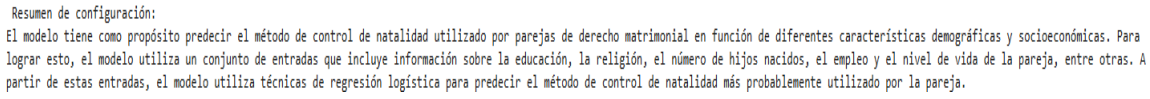
Resumen de configuración:
El modelo tiene como propósito predecir el método de control de natalidad utilizado por parejas de derecho matrimonial en función de diferentes características demográficas y socioeconómicas. Para lograr esto, el modelo utiliza un conjunto de entradas que incluye información sobre la educación, la religión, el número de hijos nacidos, el empleo y el nivel de vida de la pareja, entre otras. A partir de estas entradas, el modelo utiliza técnicas de regresión logística para predecir el método de control de natalidad más probablemente utilizado por la pareja.

Ilustración 10. Versión 1 del sistema: Resumen de configuración.

Finalmente, la Ilustración 11 muestra que el archivo de configuración .yaml se genera automáticamente a partir de las respuestas proporcionadas por el usuario y se descarga al instante.

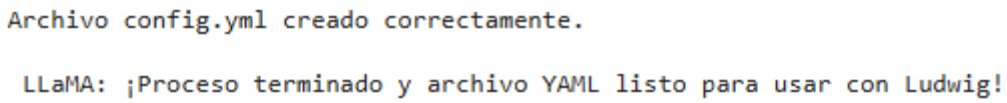

Ilustración 11. Versión 1 del sistema: Mensaje fin de Ejecución y descarga del archivo .yaml.

Esta primera versión está especialmente pensada para usuarios con recursos limitados y conjuntos de datos pequeños, sin demasiados atributos. Sin embargo, cuando el número de columnas aumenta considerablemente, el proceso se vuelve poco eficiente, ya que requiere que el usuario defina manualmente el rol y tipo de cada columna. Debido a esto desarrollé una segunda versión que será explicada a continuación.

3.1.2 Versión 2: Versión local con API Ollama y AutoML de Ludwig

Esta versión está diseñada para ejecutarse de forma local desde la terminal y se comunica directamente con un modelo como Gemma3, desplegado a través de Ollama. Su uso resulta más accesible para el usuario, ya que combina simplicidad y potencia. Además, incorpora la función create_auto_config de Ludwig, que permite generar automáticamente el archivo de configuración YAML a partir del conjunto de datos, agilizando así el proceso de configuración del modelo.

Dado que mi ordenador personal no contaba con los recursos necesarios para ejecutar esta versión desde la terminal, utilicé un equipo de la universidad para llevar a cabo la ejecución. El equipo ofrecía una capacidad computacional más avanzada mostradas en la Ilustración 12 e Ilustración 13, lo que permitía ejecutar esta versión sin problemas de rendimiento.

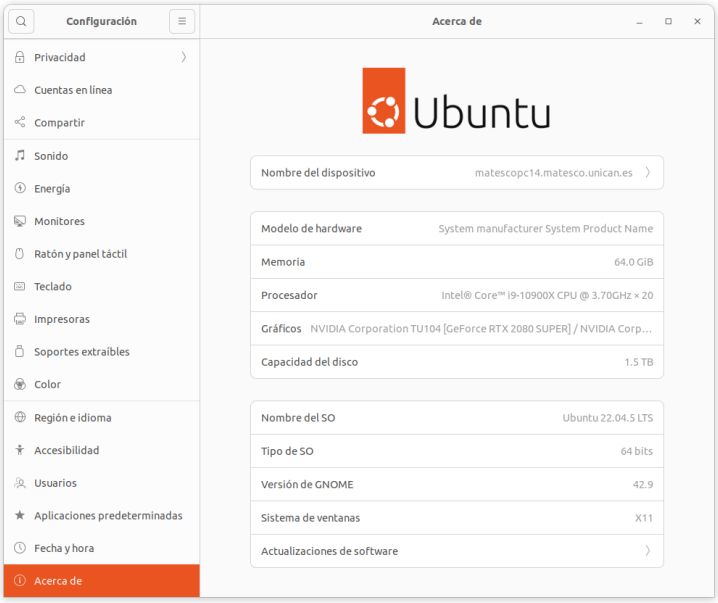

Ilustración 12. Versión 2 del sistema: Características del ordenador

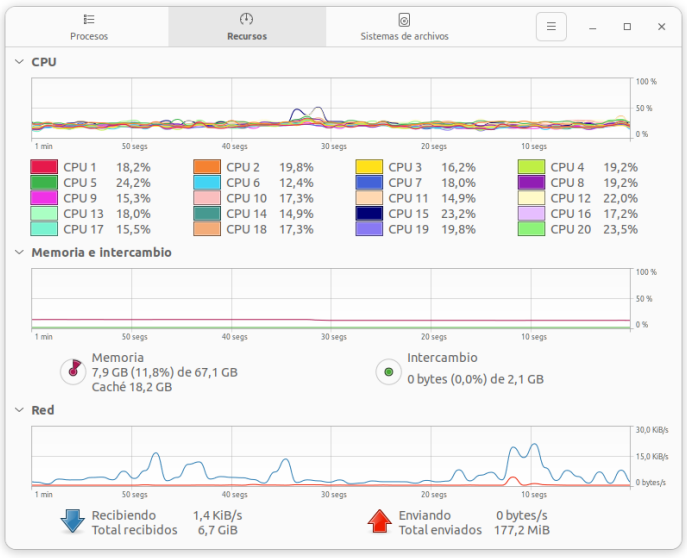

Ilustración 13. Versión 2 del sistema: Valores al ejecutar el sistema.

A diferencia de la versión anterior, esta implementación se ejecuta directamente desde la terminal. Para iniciar el proceso, es necesario proporcionar como argumento la ruta del archivo de datos que desea utilizar el usuario.

Una vez iniciado el script, el sistema establece la conexión con el modelo Gemma3 a través de Ollama y procede a cargar el archivo de datos. A continuación, muestra automáticamente las columnas.

Posteriormente, el asistente solicita al usuario que indique cuál es la columna objetivo, es decir, la variable de salida que el modelo intentará predecir. Esta información es esencial para la generación automática de la configuración mediante Ludwig, mostrada en la Ilustración 14.

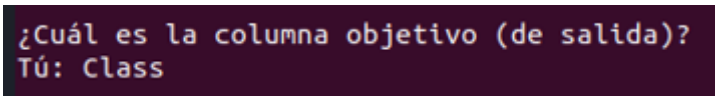

Ilustración 14. Versión 2 del sistema: Columna Objetivo.

Una vez identificada la columna objetivo, la Ilustración 15 nos permite ver como Ludwig AutoML analiza automáticamente todas las columnas del conjunto de datos. Durante este proceso, evalúa cada campo y asigna un tipo de dato apropiado en función de cada columna.

Ilustración 15. Versión 2 del sistema: Análisis realizado por Ludwig.

Tras completar el análisis de las columnas, en la Ilustración 16 e Ilustración 17 vemos como el sistema solicita al usuario algunas configuraciones adicionales relacionadas con el preprocesamiento de los datos:

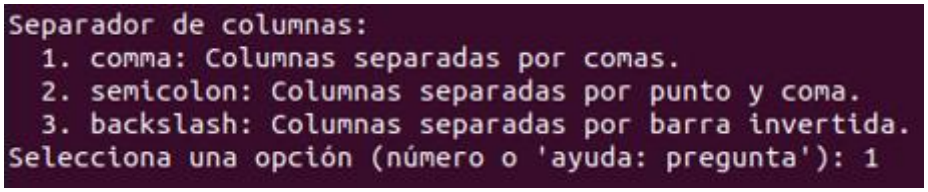

Ilustración 16. Versión 2 del sistema: Elección del separador de las columnas.

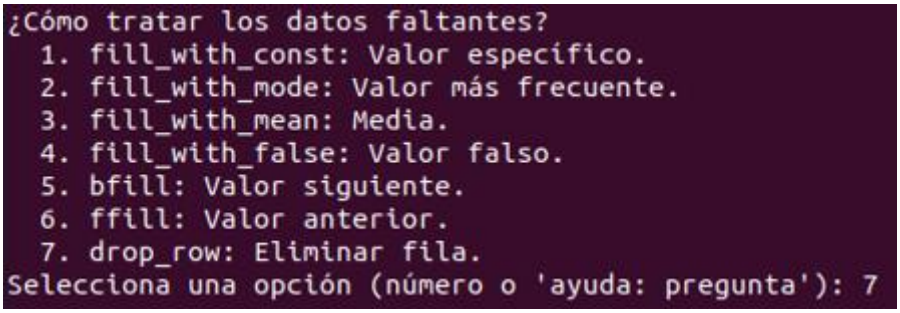

Ilustración 17. Versión 2 del sistema: Elección tratamiento de datos faltantes.

Una vez recopilada toda la información necesaria, el sistema genera automáticamente un archivo de configuración en formato .yaml, el cual es mucho más detallado y completado que en la versión anterior. Tras crear el archivo auto_config.yaml de forma temporal, la Ilustración 18 figura que el sistema ofrece la posibilidad de revisarlo y modificarlo manualmente, permitiendo ajustes en los tipos de columnas o en su rol dentro del modelo (entrada, salida o ignorada).

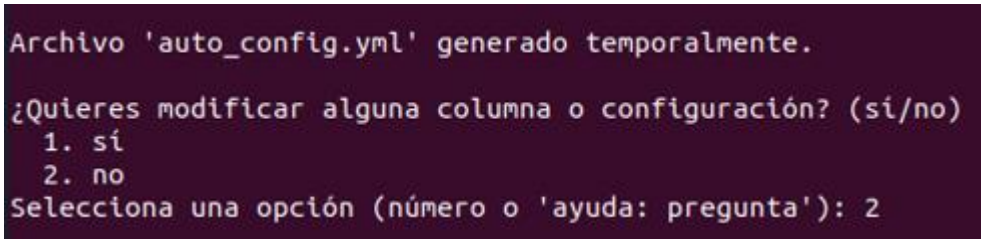

Ilustración 18. Versión 2 del sistema: Modificación YAML creado.

En caso de que el usuario decida modificar alguna columna, el sistema despliega un listado enumerado con todas las columnas detectadas, como se observa en la Ilustración 19. Desde allí, se puede seleccionar una columna específica para editarla, como se muestra en la Ilustración 20 e Ilustración 21:

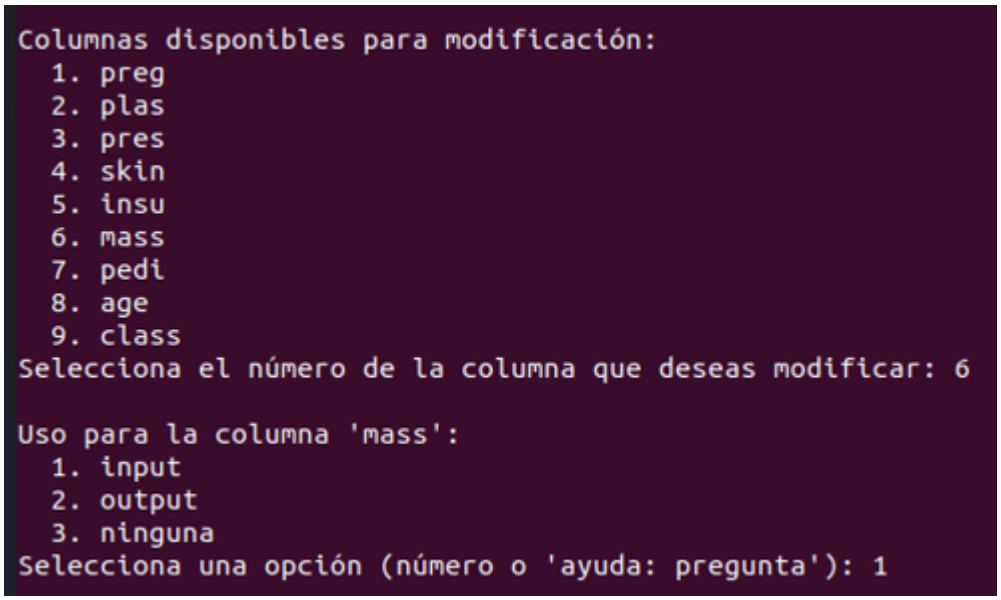

Ilustración 19. Versión 2 del sistema: Modificación rol de las columnas asignado por Ludwig.

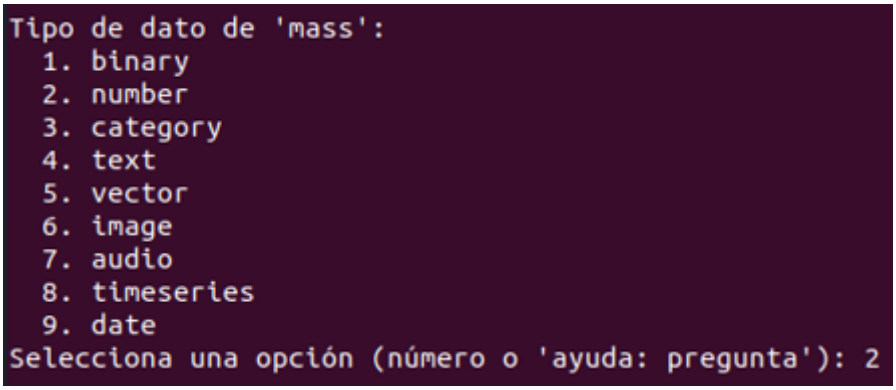

Ilustración 20. Versión 2 del sistema: Modificación del tipo de las columnas asignado por Ludwig.

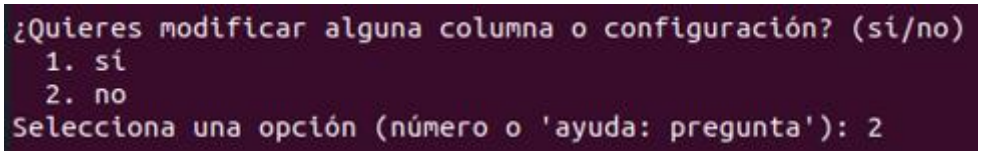

Ilustración 21. Versión 2 del sistema: Continuar o terminar con la modificación de las columnas.

Cuando el usuario indica que no desea realizar más modificaciones, el sistema guarda el archivo auto_config.yaml con todos los cambios finales aplicados. Además, se genera un resumen explicativo mediante el modelo Gemma3, el cual describe en lenguaje natural el propósito del modelo, las variables utilizadas y las decisiones de preprocesamiento elegidas.

Este resumen mostrado en la Ilustración 22 facilita la interpretación del modelo generado.

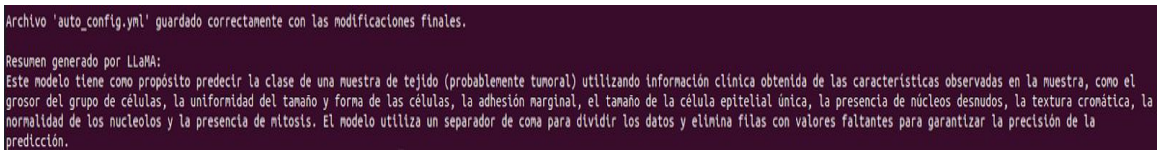

Ilustración 22. Versión 2 del sistema: Fin de ejecución y resumen de la configuración.

Esta segunda versión está pensada para usuarios que necesitan mayor eficiencia y flexibilidad. Al ejecutarse desde la terminal y apoyarse en herramientas como Ludwig AutoML y el modelo Gemma3 mediante Ollama, permite automatizar gran parte del proceso de configuración, reduciendo significativamente el esfuerzo manual requerido.

Otra ventaja de esta versión es que ofrece la posibilidad de ajustar manualmente la configuración generada, lo que la hace adecuada tanto para principiantes como para usuarios con mayor experiencia y más conocimientos. Esta combinación de automatización y personalización resuelve las limitaciones de la primera versión, especialmente en escenarios con conjuntos de datos de mayor tamaño o complejidad.

El código completo para el desarrollo de estas dos versiones puede ser encontrado en github [41].

3.2 Integración del asistente virtual

Con el objetivo de facilitar la generación y configuración de modelos automáticos en AutoML, se ha desarrollado un asistente virtual interactivo que guía al usuario en la interfaz conversacional del chatbot. Este asistente incorpora capacidades avanzadas de PLN que nos ayudan con la implementación del proyecto utilizando modelos de LLMs, LLaMA y Gemma, integrados a través de dos maneras distintas. La solución implementada combina componentes clave como RAG, generación de *prompts* dinámicos, uso de APIs de modelo local y pipelines.

A continuación, se describen en detalle los componentes técnicos utilizados.

3.2.1 RAG (Retrieval-Augmented Generation)

RAG es una arquitectura híbrida en el campo del PLN que combina modelos generativos con mecanismos de recuperación de información para mejorar la calidad y la eficacia de las respuestas generadas. A diferencia de los modelos tradicionales, que generan respuestas únicamente en función del conocimiento de que ya disponían a raíz de su entrenamiento, RAG incorpora una etapa de recuperación de información que permite acceder a documentos relevantes en tiempo real. De este modo, se mitiga una de las principales limitaciones de los modelos generativos: respuestas desactualizadas o imprecisas cuando no disponen de suficiente contexto interno. Este paradigma fue introducido por Lewis et al. (2020), [23] quienes propusieron una arquitectura en la que un módulo de recuperación extrae fragmentos de texto relevantes desde una base de datos no estructurada, que posteriormente son utilizados como contexto por un modelo generativo para producir una respuesta fundamentada y específica.

En la práctica, RAG se estructura en dos fases principales. Primero, se recuperan los documentos más relevantes según la similitud semántica con la pregunta del usuario. Esto se realiza mediante embeddings, que representan fragmentos textuales como vectores numéricos en un espacio implícito, de forma que aquellos con significados similares se encuentran próximos entre sí. Segundo, un modelo generativo utiliza tanto la pregunta original propuesta por el usuario como los documentos recuperados con la ayuda de embeddings para formar una respuesta coherente, actualizada y contextualizada. Estas competencias de recuperar información específica y generar lenguaje natural convierten a RAG en una herramienta idónea para tareas que requieren acceso a conocimientos actualizados, específicos o contextualizados.

En el presente trabajo, se ha incorporado una implementación de RAG con el objetivo de asistir al usuario durante la configuración interactiva de modelos AutoML, proporcionando explicaciones adaptadas al contexto. Para ello, se utiliza una base de datos vectorial construida a partir de un archivo CSV que contiene preguntas y respuestas relacionados con el tema y suelen ser preguntas populares entre los usuarios. Esta base se genera segmentando el texto en minifragmentos y procesándolos mediante un modelo de embeddings ("all-MiniLM-L6-v2") que transforma cada segmento en un vector semántico. Posteriormente, estos vectores se

almacenan en una estructura de datos optimizada para búsquedas por similitud, usando Chroma como vector store.

A continuación, se muestra el código que permite cargar un archivo .csv con preguntas y respuestas, dividirlo en fragmentos pequeños y convertirlos en vectores semánticos mediante un modelo de embeddings. Estos vectores se almacenan en una base de datos vectorial con Chroma para permitir búsquedas por similitud.

```
loader = CSVLoader(file_path=csv_path)
docs = loader.load()
text_splitter = CharacterTextSplitter(chunk_size=500,
chunk_overlap=50)
documents = text_splitter.split_documents(docs)
embedding = HuggingFaceEmbeddings(model_name="sentence-
transformers/all-MiniLM-L6-v2")
vectordb = Chroma.from_documents(documents=documents,
embedding=embedding, persist_directory="./db")
```

Cuando el usuario introduce una consulta iniciada por el prefijo “ayuda:”, se activa el modo de asistencia. Este mecanismo localiza, mediante búsqueda semántica, el fragmento de texto más relevante respecto a la pregunta del usuario en el .csv con la contextualización anteriormente creado. Una vez recuperado el contexto, se construye un nuevo prompt que integra tanto la pregunta del usuario como el contenido recuperado del .csv, el cual es enviado al modelo LLaMA para generar una respuesta más completa. Esta estrategia permite ofrecer respuestas más precisas y pertinentes para el tema abordado, ya que el modelo se apoya solo en información relevante y específica, minimizando el riesgo de generar respuestas irrelevantes o erróneas.

El siguiente fragmento define la función ayuda_mode. Esta función busca el fragmento más relevante en la base de conocimiento, construye un prompt con el contexto recuperado y genera una respuesta usando el modelo de lenguaje.

```
def ayuda_mode(question):
    retrived_docs =
    vectordb.similarity_search(question,k=1)
    contexto = retrived_docs[0].page_content if
    retrived_docs else ""
    prompt = f"""Usa el siguiente contexto para
    explicar al usuario de forma clara y sencilla. Si
    el contexto no es útil, responde de forma general:
    Contexto: {contexto}
    Pregunta: {question}
    Respuesta:""" raw = generate_response(prompt)
    print("🔍 Ayuda LLaMA:\n" +
    wrap_text(raw.replace(prompt, "").strip()))
    print(wrap_text(contexto))
```

La utilidad de este enfoque tiene múltiples ventajas: en primer lugar, dota al sistema de una capacidad de actualización dinámica del conocimiento sin necesidad de reentrenar el modelo generativo, pudiendo añadir más preguntas y respuestas al .csv de manera sencilla y eficiente. En segundo lugar, permite adaptar el comportamiento del asistente a distintos temas o contextos simplemente cambiando el conjunto de documentos usados. Finalmente, mejora la transparencia del sistema, ya que el usuario puede ver el contenido usado como contexto, lo que ayuda a generar confianza y a entender de dónde viene la respuesta. En conjunto, la integración de RAG aporta robustez, adaptabilidad y precisión al sistema interactivo desarrollado para la configuración asistida de modelos AutoML.

En conclusión, la base de conocimiento utilizada para el mecanismo de RAG fue elaborada de forma propia, recopilando preguntas frecuentes y relevantes sobre la configuración de modelos AutoML. Esta base se almacenó en un archivo .csv y se empleó para contextualizar las respuestas del asistente durante la interacción.

3.2.2 Prompt

Un prompt es una instrucción o entrada textual que se proporciona a un modelo de lenguaje para guiar su comportamiento y obtener una respuesta deseada. En el contexto de los LLMs, como los utilizados en este proyecto, los prompts son fundamentales para definir el contexto y la tarea que el modelo debe realizar [24].

El prompt es una cadena de texto que el usuario proporciona al modelo de lenguaje para obtener una salida útil. En el proceso de prompting, la cadena de texto del usuario se pasa al modelo de lenguaje, que genera tokens de manera iterativa condicionada al prompt. De este modo, el prompt crea un contexto que guía a los LLMs para generar salidas útiles que logren el objetivo del usuario [24].

Esta técnica, conocida como prompt engineering, ha surgido como una habilidad esencial para maximizar el potencial de los LLMs sin necesidad de modificar sus parámetros internos ni ampliar el entrenamiento de los modelos como tal. Permite adaptar modelos preentrenados a tareas específicas mediante la elaboración meticulosa de prompts que orienten el comportamiento deseado.

En el código desarrollado para este proyecto, los prompts se utilizan para interactuar con el modelo LLaMA o Gemma3, según la versión elegida y obtener respuestas que asistan al usuario en la configuración de modelos AutoML. Por ejemplo, la función generate_response(prompt) toma un prompt como entrada y genera una respuesta utilizando los modelos. Esta función se emplea en diversos contextos, como en la función ayuda_mode(question), donde se construye un prompt que incluye tanto la pregunta del usuario como el contexto recuperado de la base de datos vectorial, permitiendo al modelo generar una respuesta relevante.

A continuación, se muestra la función generate_response desarrollada, responsable de enviar el prompt al modelo de lenguaje (LLaMA o Gemma3, según la versión utilizada)

y devolver la respuesta generada. Esta función configura los parámetros del modelo, realiza la inferencia y decodifica el resultado en lenguaje natural.

```
def generate_response(prompt):
     inputs = tokenizer(prompt, return_tensors="pt",
     truncation=True, max_length=256).to("cuda")
     outputs = model.generate(
          **inputs,
          max_length=256,
          do_sample=True,
          temperature=0.7,
          top_p=0.9,
          top_k=50,
          num_return_sequences=1,
          eos_token_id=tokenizer.eos_token_id
      )
     response = tokenizer.decode(outputs[0],
     skip_special_tokens=True)
     return response
```

Además, el sistema emplea la función generar_resumen_llama_breve(config, table_config) para generar automáticamente un resumen del propósito del modelo configurado por las selecciones del usuario. Se construye un prompt en español que incluye las características de entrada, la salida esperada, el tipo de separador de columnas y el tratamiento de valores faltantes. El prompt se formula cuidadosamente para generar un único párrafo descriptivo, sin repetir la pregunta ni salirse del tema. Esto permite obtener un resumen claro, personalizado y coherente con los parámetros definidos por el usuario. El diseño de este prompt contribuye a una mejor comprensión del modelo configurado y apoya la toma de decisiones del usuario, explicándole el objetivo final de estas. Transformando así el lenguaje técnico en lenguaje natural comprensible para el usuario.

En el siguiente fragmento se define la función generar_resumen_llama_breve, encargada de generar automáticamente un resumen descriptivo del modelo AutoML configurado por el usuario. Para ello, construye un prompt en español a partir de los parámetros seleccionados (entradas, salida, separador y tratamiento de valores faltantes) y lo envía al modelo para obtener un texto claro y coherente.

```
def generar_resumen_llama_breve(config,
table_config):
    entrada = ", ".join([f['name'] for f in c
     onfig['input_features']])
    salida = ", ".join([f['name'] for f in
    config['output_features']])
    separador = table_config['separator']
    faltantes = table_config['missing_data']
```

```
    prompt = (
        f"¿Cuál es el propósito de este modelo?\n"
        f"- Entradas: {entrada}\n"
        f"- Salida: {salida}\n"
        f"- Separador: {separador}\n"
        f"- Tratamiento de valores faltantes:
{faltantes}\n\n"
        f"Respuesta (solo un párrafo en español, sin
repetir la pregunta):"
    )

    inputs = tokenizer(prompt,
return_tensors="pt").to("cuda")
    outputs = model.generate(
        **inputs,
        max_new_tokens=150,
        do_sample=True,
        temperature=0.7,
        top_p=0.9,
        top_k=50,
        eos_token_id=tokenizer.eos_token_id
    )
    response = tokenizer.decode(outputs[0],
skip_special_tokens=True)
    return response.replace(prompt, "").strip()
```

La implementación de prompts en este sistema permite una interacción más natural y eficaz con el modelo de lenguaje, facilitando la obtención de respuestas precisas y contextualizadas. Además, al diseñar cuidadosamente los prompts, se puede controlar el comportamiento del modelo y adaptar sus respuestas a las necesidades específicas del usuario, lo que es especialmente útil en aplicaciones como la configuración asistida de modelos AutoML.

3.2.3 Pipeline

Un pipeline en el contexto del aprendizaje automático se refiere a una secuencia estructurada de pasos que transforman datos brutos en modelos predictivos listos para su despliegue. Esta estructura modular permite automatizar y estandarizar el flujo de trabajo, facilitando la colaboración entre equipos y mejorando la reproducibilidad de los resultados. Es decir, un pipeline de aprendizaje automático es una serie de pasos interconectados de procesamiento de datos y modelado diseñados para automatizar, estandarizar y agilizar el proceso de construcción, entrenamiento, evaluación y despliegue de modelos de aprendizaje automático [25].

En el código desarrollado para este proyecto, el concepto de pipeline se implementa mediante una serie de funciones y estructuras que guían al usuario desde la carga de

datos hasta la generación de un archivo de configuración YAML para el entrenamiento automático de modelos con Ludwig en la otra herramienta que explico en un futuro.

Este flujo incluye la carga y preprocesamiento de datos, la configuración interactiva de características de entrada y salida de las columnas, es decir, su rol en nuestro modelo, indicar el tipo de columna, la generación de resúmenes mediante modelos de lenguaje y la creación del archivo de configuración final.

La implementación de este pipeline ofrece múltiples ventajas a nuestro sistema:

- **Modularidad**: Cada paso del proceso está desarrollado en funciones específicas, lo que facilita su mantenimiento, reutilización, división del sistema y comprensión del código a nivel técnico.

- **Automatización**: La secuencia de pasos se ejecuta de manera automatizada, reduciendo la intervención manual del usuario y minimizando errores en caso de falta de conocimiento sobre el tema.

- **Estabilidad del proceso:** Una descripción precisa de todas las etapas y parámetros asegura que el procedimiento se pueda ejecutar múltiples veces obteniendo resultados consistentes.

- **Crecimiento eficiente:** La naturaleza modular del pipeline permite ampliarlo para procesar datos más extensos o afrontar nuevas tareas como hicimos en la transformación de la primer a la segunda versión, sin rediseñar la estructura general y complicar el desarrollo.

En resumen, la implementación de un pipeline en este proyecto no solo optimiza el flujo de trabajo en el desarrollo de modelos de aprendizaje automático, sino que también establece una base sólida para futuras ampliaciones y adaptaciones del sistema.

3.2.4 API

Una API es un conjunto de definiciones y protocolos que permiten que diferentes componentes de software se comuniquen entre sí. Una API actúa como un contrato entre distintas aplicaciones, facilitando la integración de funcionalidades sin que los desarrolladores necesiten comprender los detalles internos de cada sistema [26]

En el contexto de este proyecto, las APIs desempeñan un papel fundamental al permitir la interacción entre el sistema desarrollado y los LLM, como LLaMA y Gemma3. Por ejemplo, en la segunda versión del código, se utiliza una API para enviar solicitudes al puerto donde el modelo Gemma3 es alojado localmente a través de Ollama. Esto se logra mediante una llamada HTTP POST que envía un prompt al modelo y recibe una respuesta generada, lo que permite integrar el procesamiento de lenguaje natural en la aplicación sin necesidad de implementar el modelo desde cero.

A continuación, se presenta el fragmento de código utilizado para enviar un prompt al modelo Gemma3 mediante una API local expuesta por Ollama. Esta función realiza una solicitud POST al endpoint del modelo, recibe la respuesta generada en tiempo real y la concatena línea a línea para devolverla como una cadena de texto.

```
def generate_response(prompt):
    respuesta = requests.post(
        "http://localhost:11434/api/generate",
        json={"model": "gemma3", "prompt": prompt},
        stream=True
    )
    respuesta_completa = ""
    for linea in respuesta.iter_lines():
        if linea:
            parte = json.loads(linea)
            respuesta_completa +=
parte.get("response", "")
return respuesta_completa.strip()

print("\nModelo gemma3 conectado con Ollama.")
```

En conclusión, las APIs son fundamentales para la integración del sistema desarrollado en este proyecto, ya que permiten la integración eficiente de los modelos de lenguaje avanzados.

3.3 Interfaz de usuario

En esta última fase del flujo AutoML, se utiliza una interfaz gráfica desarrollada con Streamlit. Esta interfaz fue completamente desarrollada por mí utilizando Streamlit, adaptando su funcionalidad a los distintos tipos de conjuntos de datos (como archivos .arff o particiones .train/.test), haciendo más cómodo el proceso de carga, entrenamiento y evaluación del modelo, y proporcionando visualizaciones de métricas y predicciones para facilitar su interpretación. Esta herramienta permite al usuario cargar un conjunto de datos y un archivo de configuración .yaml de manera cómoda para entrenar un modelo utilizando Ludwig. Está pensada para facilitar el entrenamiento incluso a usuarios con conocimientos técnicos limitados, ya que abstrae completamente la programación interna.

Es importante destacar que esta herramienta ha sido diseñada para funcionar con los archivos .yaml generados en la segunda versión, ya que dicha versión produce configuraciones más completas y eficientes, a diferencia de la primera versión, donde el archivo YAML incluye únicamente las características de entrada, salida y el preprocesamiento básico, el .yaml de la segunda versión incorpora también:

- Parámetros del combinador (combiner) que definen la arquitectura del modelo (por ejemplo, uso de tabnet).

- Configuración del optimizador y del scheduler para el entrenamiento.
- Estrategia de validación y métrica de evaluación.
- Bloque de búsqueda automática de hiperparámetros (hyperopt) con espacio de búsqueda y tipo de algoritmo definido.

Este .yaml más completo permite aprovechar al máximo las capacidades de Ludwig AutoML, incluyendo la optimización automática de arquitecturas y tasas de aprendizaje, lo que no sería posible con el .yaml más básico generado en la primera versión.

La Ilustración 23 exhibe lo que ve el usuario al acceder a la interfaz, se presenta una pantalla de bienvenida donde se solicita al usuario que seleccione el formato del conjunto de datos que desea utilizar para el entrenamiento. La aplicación permite trabajar con archivos en formato .arff o con conjuntos particionados previamente en .train y .test, en formato Excel o CSV.

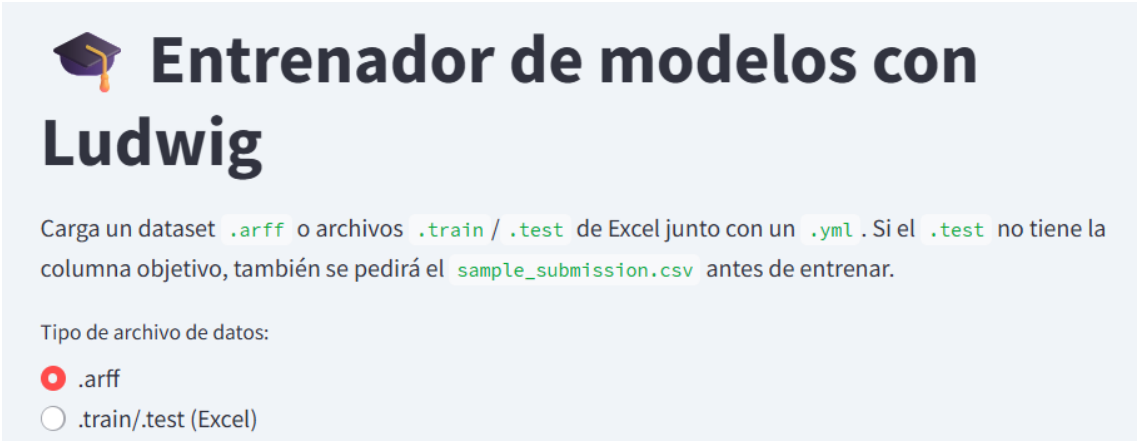

Ilustración 23. Mensaje de bienvenida y elección del tipo de datos.

Si se selecciona el formato .arff, la Ilustración 24 presenta la solicitud de la interfaz al usuario para subir el archivo de configuración .yaml y el conjunto de datos en formato .arff:

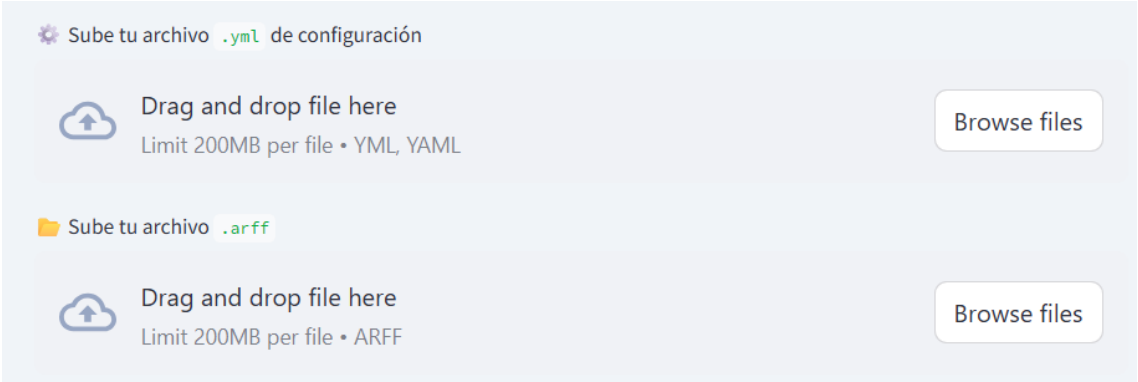

Ilustración 24. Subida archivos tipo .arff.

Una vez cargados correctamente el archivo .arff y el .yaml, se habilita la opción para iniciar el entrenamiento del modelo presentado en la Ilustración 25.

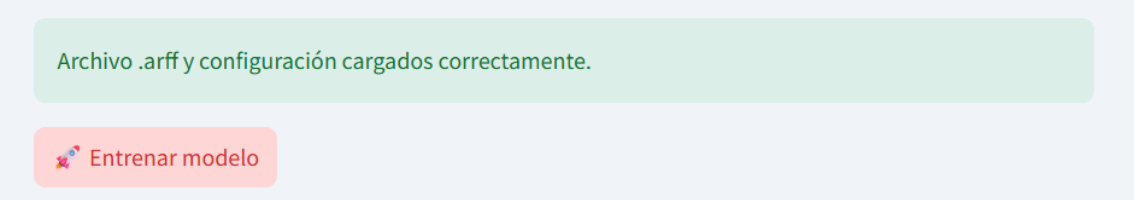

Ilustración 25. Botón de comienzo de entrenamiento.

Cuando se utiliza un archivo en formato .arff, el sistema realiza automáticamente una partición del conjunto de datos, utilizando el 80 % para entrenamiento y el 20 % para prueba. Una vez completado el entrenamiento, la interfaz presenta diversas métricas de evaluación generadas por Ludwig expuestas en la Ilustración 26, una matriz de confusión, que figura en la Ilustración 27, que permite analizar visualmente el rendimiento del modelo en tareas de clasificación, un reporte detallado por clase y predicciones, habiendo la posibilidad de descargar todas las métricas.

Ilustración 26. Métricas de evaluación.

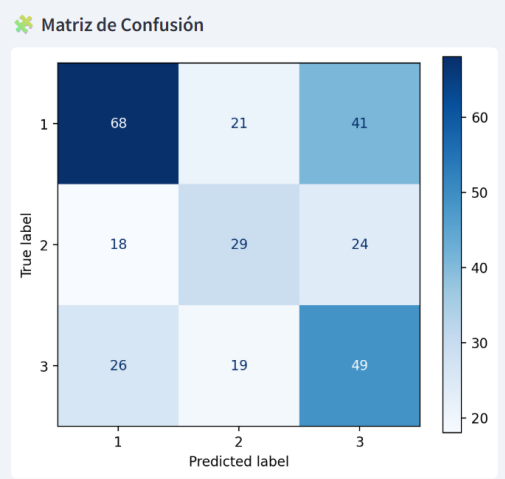

Ilustración 27. Matriz de confusión.

La Ilustración 28 muestra lo que pasa si es del tipo train/test .csv, cargamos el .yaml, el .train, el .test y si este .test no tiene la columna objetivo que es lo que ocurre en los datasets utilizados descargados en kaggle, pide también cargar el sample_submission.csv expuesta en la Ilustración 29:

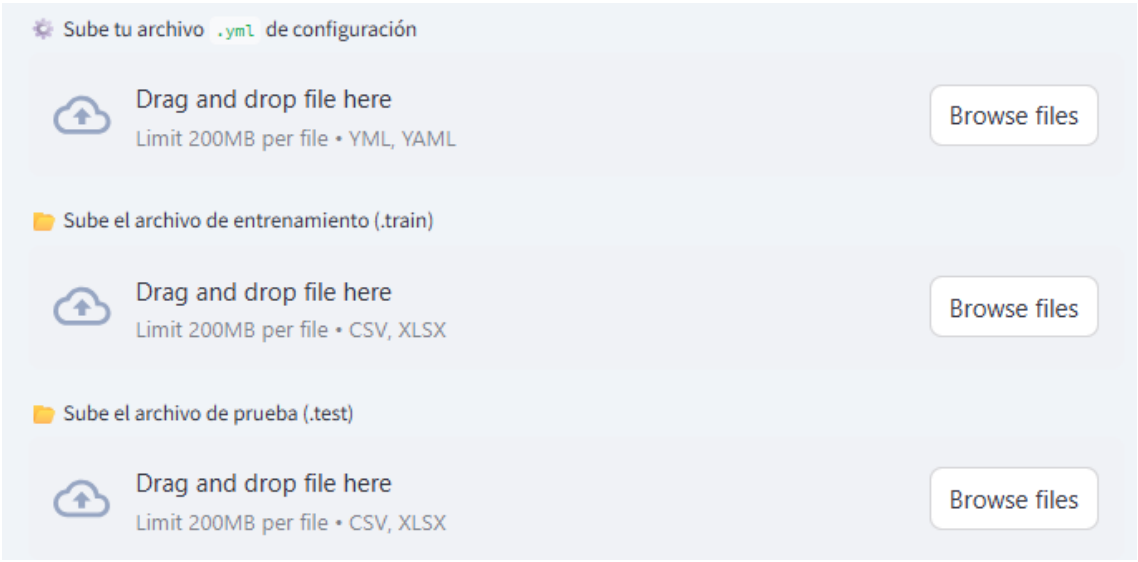

Ilustración 28. Subida Archivos tipo .train/.test.

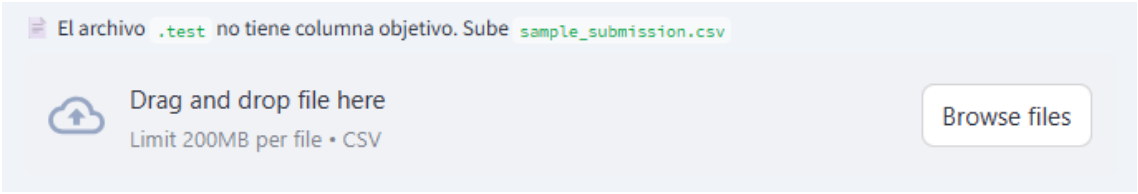

Ilustración 29. Subida sample_submission.

Si se elige el formato .train/.test, la interfaz solicita subir los siguientes archivos: el archivo de configuración .yaml, el conjunto de entrenamiento (.train) y el conjunto de prueba (.test), todos ellos en formato .csv o .xlsx.

En muchos casos, como ocurre con datasets descargados desde plataformas como Kaggle, el archivo .test no incluye la columna objetivo. En esas situaciones, la aplicación también pedirá al usuario subir un archivo sample_submission.csv, necesario para generar correctamente las predicciones en el formato esperado.

Una vez que todos los archivos requeridos han sido cargados correctamente, la interfaz permite iniciar el proceso de entrenamiento del modelo permitiendo activar el botón de la Ilustración 30:

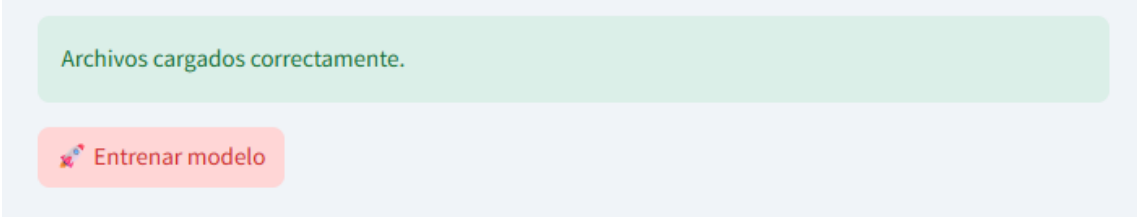

Ilustración 30. Botón de comienzo de entrenamiento.

La Ilustración 31 exhibe el comportamiento de nuestra GUI cuando el conjunto de prueba (.test) no contiene la columna objetivo, el sistema genera automáticamente un archivo submission.csv en el formato requerido por Kaggle. Este archivo puede ser descargado directamente desde la interfaz y subirlo a Kaggle para obtener la puntuación oficial en la plataforma.

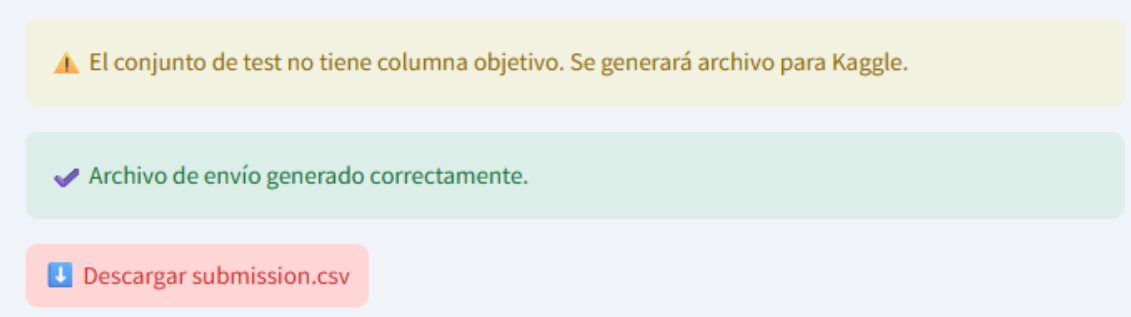

Ilustración 31. Descarga submission.csv.

He subido el código completo para el desarrollo de este GUI a hugging face [40] .

CAPÍTULO 4

4. Evaluación

4.1 Datasets utilizados

Para llevar a cabo la evaluación de los modelos de aprendizaje automático, se seleccionaron diversos conjuntos de datos públicos provenientes de páginas reconocidas como OpenML o Kaggle. Estos datasets se dividen en dos categorías principales según la naturaleza del problema: clasificación y regresión.

La Tabla 5 presenta los conjuntos de datos empleados en tareas de clasificación, en las que el objetivo es predecir etiquetas discretas o categorías, como el diagnóstico de una enfermedad o la pertenencia a un grupo específico.

Nombre	Características	Tamaño Entrenamiento	Tamaño Prueba	Fuente
Pima Indians Diabetes	8	515	253	OpenML_Diabetes [27]
Breast-w	9	469	230	OpenML_Breastw [28]
Contraceptive Method Choice	9	987	486	OpenML_cmc [29]
Hypotheroid	29	2528	1245	OpenML_Hypothyroid [30]

Tabla 5: Datasets de Clasificación

Pima Indians Diabetes

El conjunto de datos *Pima Indians Diabetes* fue compilado originalmente por el NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases, por sus siglas en inglés) y puesto a disposición pública a través del repositorio de la Unidad de Cuidados Intensivos (UCI, por sus siglas). Posteriormente fue incorporado a la plataforma OpenML (taskId: 37) para la evaluación de algoritmos de clasificación. Contiene mediciones diagnósticas médicas de pacientes femeninas de ascendencia Pima con el objetivo principal de predecir la aparición de diabetes. Entre las variables clave se encuentran la concentración de glucosa en plasma, la presión arterial, el índice de masa corporal (IMC, por sus siglas) y la edad. Dada la frecuencia global de la diabetes y su impacto en poblaciones de riesgo, este conjunto de datos se considera altamente relevante en tareas de predicción médica.

Breast-w

Este conjunto de datos fue compilado inicialmente por el Dr. William H. Wolberg del Hospital de la Universidad de Wisconsin y donado a la UCI en 1992. Posteriormente fue

incorporado a OpenML (taskId: 15) como recurso para la evaluación de algoritmos de clasificación. Contiene características derivadas de imágenes digitalizadas de aspirados con aguja fina (FNA) de masas mamarias, con el objetivo principal de distinguir entre tumores benignos y malignos. Las variables incluyen medidas como el radio, la textura, la concavidad y la simetría de los núcleos celulares. Dada la alta incidencia del cáncer de mama y la importancia de su detección temprana, este conjunto de datos es ampliamente utilizado en aplicaciones clínicas de IA.

Contraceptive Method Choice

El conjunto *Contraceptive Method Choice* fue compilado a partir de una muestra del Estudio Nacional de Prevalencia de Anticonceptivos de Indonesia de 1987. Fue donado por Jiee-Sien Lim y está disponible en OpenML con el taskId 23. Contiene información sobre mujeres casadas que no estaban embarazadas, con el objetivo principal de predecir el método anticonceptivo actualmente utilizado en función de características demográficas y socioeconómicas. Entre las variables clave se incluyen la edad de la esposa, nivel educativo, número de hijos, situación laboral, ocupación del esposo, nivel de vida, religión y exposición a medios de comunicación. Este conjunto es especialmente útil en tareas de predicción en el ámbito de la salud pública y la planificación familiar.

Hypothyroid

El conjunto de datos *Hypothyroid* fue compilado a partir de casos clínicos proporcionados por el Garavan Institute y distribuido por el repositorio de UCI. Posteriormente fue incorporado a OpenML (taskId: 57). Contiene información médica detallada de pacientes sometidos a pruebas para diagnosticar disfunciones tiroideas, en particular el hipotiroidismo. Incluye variables tanto categóricas como numéricas, como edad, sexo, síntomas clínicos, resultados de laboratorio y tratamientos previos. Su objetivo principal es identificar correctamente la presencia o ausencia de hipotiroidismo. Debido a que estas enfermedades suelen pasar desapercibidas, este conjunto de datos resulta especialmente útil para el desarrollo de modelos de clasificación médica basados en IA.

La Tabla 6 incluye datasets orientados a regresión, donde la variable objetivo es continua y se busca predecir valores numéricos, como niveles futuros de glucosa en sangre o cifras de mortalidad por COVID-19.

Nombre	Características	Tamaño Entrenamiento	Tamaño Prueba	Fuente
Liver Disorders	5	232	113	OpenML_LiverDisorder [31]
BrisT1D Blood Glucose	506	177024	3644	Kaggle_Brist1d [32]
Covid19 Deaths	18	129156	43052	Kaggle_Covid19Deaths [33]
Covid19 Cases	92	2700	893	Kaggle_Covid19 [34]

Tabla 6: Datasets de Regresión

Liver Disorders

El conjunto de datos *Liver Disorders* fue compilado por la compañía BUPA Medical Research Ltd. y está disponible en OpenML (taskId: 8). Fue diseñado para estudiar la relación entre parámetros bioquímicos en sangre y enfermedades hepáticas relacionadas con el consumo excesivo de alcohol. Cada instancia representa los resultados de pruebas realizadas a varones adultos con distintos niveles de consumo de alcohol. Las variables incluyen enzimas hepáticas, gamma-glutamil transpeptidasa (GGT, Gamma-Glutamyl Transferase por sus siglas en inglés) y niveles de bilirrubina, entre otras. El objetivo de este conjunto es predecir la presencia o ausencia de trastornos hepáticos. Dado su uso frecuente en evaluación biomédica, este dataset es ampliamente utilizado en estudios de regresión supervisada en contextos clínicos.

BrisT1D Blood Glucose

Este conjunto de datos, proveniente de la competición "BrisT1D – Blood Glucose Prediction" en Kaggle, contiene registros de pacientes con diabetes tipo 1 recopilados mediante sensores como monitores continuos de glucosa (CGM), bombas de insulina y dispositivos de actividad. Cada muestra incluye una ventana temporal de seis horas con variables como niveles de glucosa, insulina, carbohidratos y ejercicio, con el objetivo de predecir el nivel de glucosa una hora en el futuro. Este conjunto es clave para desarrollar modelos de predicción en sistemas de control de la diabetes y páncreas artificiales.

Covid19 Deaths

Este conjunto fue creado en la competición "Covid19-Death-Predictions" de Kaggle con el objetivo de modelar y predecir la evolución de muertes por COVID-19 a nivel global. Incluye información diaria por país y región sobre casos confirmados, muertes

acumuladas y variables epidemiológicas. El objetivo es generar predicciones precisas sobre la mortalidad futura asociada al virus. Este conjunto ha sido ampliamente utilizado para validar modelos predictivos en contextos de salud pública, especialmente durante la pandemia.

Covid19 Cases

El conjunto de datos *Covid19 Cases* fue desarrollado como parte de una tarea académica en el curso "MIL 2021 Spring - HW1". Está disponible en Kaggle y tiene como objetivo predecir una etiqueta de clasificación binaria a partir de múltiples características numéricas anónimas. Aunque está enfocado originalmente a clasificación, las variables numéricas que contiene también permiten su uso en tareas de regresión para predecir casos diarios. Este dataset permite evaluar la capacidad general de los algoritmos sin depender del conocimiento contextual del dominio.

4.2 Resultados

Para evaluar el rendimiento de los modelos en tareas de clasificación y regresión se utilizaron métricas muy utilizadas y reconocidas en la comunidad de aprendizaje automático:

- **Accuracy**: La exactitud (accuracy) mide la proporción de predicciones correctas realizadas por el modelo sobre el total de casos evaluados. Es una métrica ampliamente utilizada para evaluar el rendimiento general de los clasificadores [35].

- **ROC-AUC (Receiver Operating Characteristic-Area Under the Curve):** El área bajo la curva ROC (AUC) evalúa la capacidad del modelo para distinguir entre clases. Un valor de AUC cercano a 1 indica un alto rendimiento en la clasificación [35].

- **Loss:** La función de pérdida cuantifica la diferencia entre las predicciones del modelo y los valores reales durante el entrenamiento. Es fundamental para guiar el proceso de aprendizaje del modelo [36].

- **Precision**: La precisión indica la proporción de verdaderos positivos entre todas las instancias que el modelo ha clasificado como positivas. Es útil para evaluar la exactitud de las predicciones positivas del modelo [37].

- **Recall**: El recall, o sensibilidad, mide la proporción de verdaderos positivos que el modelo identifica correctamente entre todas las instancias que realmente son positivas. Es crucial cuando es importante capturar todos los casos positivos [37].

- **F1-Score**: El F1-score es la media armónica entre la precisión y el recall. Proporciona una única métrica que equilibra ambos aspectos, especialmente útil en situaciones con clases desbalanceadas [38].

Los resultados de clasificación obtenidos por nuestra herramienta se resumen en la Tabla 7, donde se muestran las métricas comentadas anteriormente. Estas métricas ofrecen una visión completa del funcionamiento de los modelos entrenados, permitiendo evaluar tanto la capacidad predictiva como la usabilidad de los conjuntos de datos.

Dataset	Accuracy	ROC-AUC	Loss	Precision	Recall	F1-score
Diabetes	0.712	0.714	0.753	0.688	0.685	0.686
Breast-w	0.974	0.956	0.482	0.962	0.956	0.959
Hypotheroid	0.740	0.732	0.113	0.862	0.740	0.788
Contraceptive	0.484	0.656	1.534	0.486	0.484	0.483

Tabla 7: Resultados de Datasets Clasificación

Como puede observarse, los resultados son muy positivos en los conjuntos Breast-w e Hypothyroid, donde se alcanzan valores cercanos a la perfección en métricas clave como Accuracy, F1-score y Precision. Estos resultados indican que el modelo ha logrado aprender patrones relevantes usando las características de entrada, con una alta capacidad de predicción.

En cambio, el dataset Contraceptive presenta un rendimiento inferior que los demás, la baja puntuación en Accuracy (0.484), así como en el resto de las métricas, sugieren que se trata de un conjunto de datos particularmente complejo, puede explicarse por diversas razones, como la presencia de ruido en las variables o una débil correlación entre los atributos de entrada y la variable objetivo.

Con el fin de contextualizar estos resultados y saber si son realmente correctos, se ha realizado una comparación directa con los valores obtenidos por un experto en IA utilizando Ludwig directamente, es decir, empleando configuraciones avanzadas y personalizadas de esta herramienta. Esta comparación, que se muestra en la Tabla 8, permite identificar el nivel de competitividad de nuestra herramienta.

Dataset	Métrica	Nuestro Modelo	Experto en IA usando Ludwig
Diabetes	ROC-AUC	0.714	0.826
Breast-w	ROC-AUC	0.956	0.967
Hypotheroid	Accuracy	0.740	0.798
Contraceptive	Accuracy	0.484	0.495

Tabla 8: Comparación Datasets Clasificación

Los resultados muestran que el experto logra valores ligeramente superiores a los obtenidos con nuestra herramienta en todos los conjuntos de datos. No obstante, las

diferencias observadas son en general pequeñas y no comprometen la validez del enfoque desarrollado. Por ejemplo, en el dataset Breast-w, el valor de ROC-AUC de nuestro modelo es de 0.956, mientras que el del experto es de 0.967, lo cual representa una diferencia insignificante. Esta mínima diferencia en el rendimiento evidencia la robustez del sistema desarrollado, especialmente considerando que no se ha recurrido a técnicas de ajuste avanzadas.

De forma destacable, en el caso del dataset Contraceptive, ambos enfoques muestran resultados bajos, lo que indica que las dificultades no provienen de la herramienta empleada, sino del propio conjunto de datos. Esta coincidencia en el bajo rendimiento, incluso por parte de un experto en IA utilizando Ludwig, indica que el dataset presenta características problemáticas que limitan el ML en general, y no un fallo específico de nuestra herramienta.

En conclusión, estos resultados reflejan un buen desempeño del sistema desarrollado para tareas de clasificación, acercándose al rendimiento de modelos avanzados desarrollados con Ludwig por especialistas en IA. Esta comparación no solo valida la calidad del proyecto, sino que también resalta el potencial de mejora futura mediante la incorporación de técnicas como la optimización de hiperparámetros para acercarnos aún más a los resultados obtenidos por el experto.

Además, en esta sección se presentan los resultados obtenidos por las herramientas desarrolladas en tareas de regresión, utilizando datasets disponibles en la plataforma Kaggle. Estos datasets no vienen acompañados de métricas de evaluación estándar, en su lugar, Kaggle ofrece un sistema de evaluación mediante submissions (archivos .csv) que se comparan con las predicciones esperadas a través de métricas como RMSE (Root Mean Square Error, por sus siglas), el error cuadrático medio mide la raíz cuadrada de la media de los errores al cuadrado entre las predicciones del modelo y los valores reales. Es sensible a errores grandes y proporciona una medida de la precisión del modelo en tareas de regresión, calculadas automáticamente al subir las predicciones al entorno de competición correspondiente al dataset [39].

Para esta evaluación, se utilizó nuestra herramienta de entrenamiento de modelos para generar los archivos submission.csv con las predicciones sobre los conjuntos de datos elegidos. Posteriormente, dichos archivos fueron subidos manualmente a Kaggle, lo que permitió obtener la métrica de RMSE para cada uno de los modelos entrenados. Esta metodología asegura una evaluación justa y alineada con los resultados de la comunidad para estos conjuntos, dejando todo el proceso de evaluación a Kaggle.

Los resultados obtenidos con los datasets de regresión, así como una comparación con los resultados del experto en IA usando Ludwig, se resume en la Tabla 9.

Dataset	Métrica	Nuestro Modelo	Experto en IA usando Ludwig
Liver Disorders	RMSE	5.223	3.912
BrisT1D	RMSE	4.395	3.30
Covid19 Deaths	RMSE	2319	1743
Covid19 Cases	RMSE	2.028	1.521

Tabla 9: Comparación Resultados Regresión

Tal como se muestra en todos los casos, los modelos desarrollados presentan valores de RMSE superiores a los obtenidos por el experto. Esta diferencia puede atribuirse como he comentado con anterioridad al hecho de que el experto en IA aplica configuraciones específicas, ajustes finos y selección manual de parámetros que escapan al alcance de nuestra herramienta automatizada. Aun así, los valores obtenidos son razonables, especialmente considerando la dificultad de los datasets, como en el caso de Covid19 Deaths, donde las predicciones se ven afectadas por la alta variabilidad de las características de entrada.

En términos generales, los resultados en regresión sugieren que la herramienta desarrollada es capaz de generar modelos con un nivel aceptable de precisión, aun cuando existen márgenes de mejora evidentes. Esta validación externa, obtenida a través del entorno competitivo de Kaggle, refuerza la utilidad práctica de la solución propuesta.

4.3 Análisis desde la perspectiva de HCAI

Para complementar la evaluación técnica, se consultó a un experto independiente en IA para evaluar diversos aspectos relacionados con el grado de automatización y el nivel de control humano en el sistema propuesto, el cual integra técnicas de AutoML con LLMs. La evaluación se centró en la alineación del sistema con los principios clave de la IA centrada en el ser humano (HCAI, Human-Centered Artificial Intelligence en inglés).

Para guiar la evaluación, se pidió al experto que respondiera a nueve preguntas. La Tabla 10 presenta las calificaciones dadas por el experto para cada pregunta de la encuesta, utilizando una escala del 1 (muy deficiente) al 10 (excelente). Desde la perspectiva del experto, el sistema parece capaz de manejar eficazmente varios tipos de datos, y el módulo de limpieza de datos está integrado y funciona según lo esperado. Sin embargo, algunos componentes (por ejemplo, la ingeniería de características y la validación) no fueron completamente observables durante la evaluación.

Pregunta (ID + Descripción)	Calificación del Experto
P1 – La aplicación es fácil de utilizar y permite una interacción fluida sin requerir un alto nivel de conocimientos técnicos.	7
P2 – La aplicación ofrece suficiente flexibilidad para configurar o personalizar distintas etapas del pipeline de aprendizaje automático.	7
P3 – La aplicación proporciona información clara e interpretable que facilita la comprensión y supervisión de los modelos generados.	8
P4 – La aplicación es capaz de detectar automáticamente y manejar de forma adecuada distintos tipos de datos (por ejemplo, numéricos, categóricos, texto).	9
P5 – La aplicación realiza automáticamente operaciones básicas de limpieza de datos (como tratamiento de valores ausentes o eliminación de duplicados) sin requerir intervención manual.	8
P6 – La aplicación es capaz de generar o seleccionar automáticamente características relevantes que contribuyan a mejorar el rendimiento del modelo.	6
P7 – La aplicación puede seleccionar y entrenar modelos de aprendizaje automático adecuados de forma automática, en función del conjunto de datos proporcionado.	8
P8 – La aplicación automatiza de manera eficaz el proceso de ajuste y optimización de modelos.	7
P9 – La aplicación incorpora procedimientos automáticos para la evaluación y validación de los modelos entrenados, utilizando técnicas apropiadas.	7

Tabla 10: Resumen de Calificación del Experto (1 = Muy deficiente, 10 = Excelente)

CAPÍTULO 5

5. Conclusiones

5.1 Grado de consecución de los objetivos

A lo largo del desarrollo del TFG, se ha alcanzado en gran medida el objetivo principal: diseñar y desarrollar una herramienta que permita a usuarios sin conocimientos técnicos interactuar con sistemas AutoML mediante una interfaz conversacional basada en LLMs. La solución implementada proporciona un entorno cómodo, accesible y guiado para construir modelos predictivos, lo cual responde a la necesidad de facilitar el uso de la IA para todos.

Respecto a los subobjetivos planteados al inicio del proyecto, el grado de consecución es el siguiente:

- **Diseño e implementación de una interfaz conversacional respaldada por LLMs**: Este objetivo ha sido alcanzado satisfactoriamente. La interfaz desarrollada permite una interacción natural con el sistema, posibilitando que el usuario, sin conocimientos técnicos ni recursos, pueda cargar datos, definir tareas y construir modelos predictivos. El asistente virtual guía el proceso, interpreta correctamente las intenciones expresadas en lenguaje natural por parte del usuario y las traduce en términos técnicos de aprendizaje automático (como clasificación o regresión) que un usuario sin conocimientos sería imposible que los definiese, asegurando que el sistema ejecute las acciones adecuadas.

- **Integrar técnicas de AutoML**: Se ha logrado incorporar un sistema AutoML que automatiza tareas clave como la selección de algoritmos, el ajuste de hiperparámetros y el preprocesamiento de datos. Este sistema permite a los usuarios centrarse en los resultados sin preocuparse por los aspectos técnicos del proceso de modelado.

- **Traducción de objetivos del usuario en tareas de ML mediante lenguaje natural**: Este objetivo también ha sido alcanzado de manera efectiva. La herramienta permite interpretar y explicar correctamente las intenciones del usuario, traduciendo sus objetivos y selecciones expresadas en lenguaje natural. Esto ha sido posible gracias a la integración de modelos LLM.

- **Evaluación del rendimiento de la herramienta**: Se ha realizado una evaluación utilizando conjuntos de datos públicos, analizando tanto la calidad de los modelos como la eficacia. Los resultados indican que la herramienta es válida para generar modelos con rendimiento competitivo y que la interacción conversacional mejora la usabilidad del proceso.

En conclusión, los objetivos establecidos en el capítulo 1 han sido cumplidos de forma satisfactoria. La herramienta desarrollada contribuye a reducir las barreras de entrada

al uso de tecnologías de IA, brindando una experiencia comprensible y guiada para usuarios sin conocimientos técnicos, lo que valida la propuesta de valor de este proyecto.

5.2 Líneas de trabajo futuras

El desarrollo de esta herramienta marca un primer paso importante hacia la adaptabilidad del aprendizaje automático mediante interfaces conversacionales. No obstante, existen múltiples áreas de mejora que pueden explorarse en trabajos futuros para mejorar tanto la funcionalidad como la experiencia del usuario.

Una de las líneas más relevantes es la mejora de la explicabilidad de los modelos generados. Aunque la herramienta proporciona actualmente una interpretación básica de los resultados, sería deseable integrar técnicas más avanzadas que permitan explicar en mayor profundidad el comportamiento del modelo, que podrían ayudar a los usuarios a comprender de manera más clara por qué el modelo toma determinadas decisiones, incluso sin tener conocimientos técnicos. Esto no solo aumentaría la confianza del usuario en el sistema, sino que también podría utilizar la herramienta para aprender.

Además, se considera fundamental mejorar la base de conocimiento utilizada por el sistema, actualmente construida de forma manual a partir de preguntas y respuestas frecuentes relacionadas con la configuración de modelos AutoML. Esta base, almacenada en un archivo .csv, fue elaborada específicamente para este proyecto como una prueba de concepto inicial y comprobar el correcto funcionamiento de esta. En trabajos futuros, se propone ampliar su contenido, mejorando la calidad, con el objetivo de proporcionar respuestas más completas, adaptadas y actualizadas. Esta mejora permitiría al asistente ofrecer una ayuda más completa.

Otro aspecto clave es el fortalecimiento de la interacción con el asistente virtual. En la versión actual, la comunicación sigue un flujo mayormente guiado, lo que puede resultar limitado en ciertos contextos. Se propone evolucionar hacia un sistema más flexible que permita al usuario modificar pasos anteriores del proceso sin necesidad de reiniciar la ejecución, esto contribuiría a una experiencia más fluida y adaptativa, alineada con las necesidades reales del usuario.

Así mismo, una línea prometedora de trabajo podría consistir en integrar todo el flujo de trabajo dentro de una única herramienta conversacional, eliminando pasos intermedios que todavía requieren cierta intervención por parte del usuario. Es decir, se plantea la automatización completa de la generación del archivo .yaml necesario para el entrenamiento con Ludwig, de forma que el propio asistente, a partir de la conversación con el usuario, construya internamente dicho fichero, además de que el entrenamiento del modelo con Ludwig sea gestionado completamente por la IA generativa, de manera que el usuario no tenga que ejecutar manualmente ningún proceso técnico. La idea es que la IA, una vez definidos los parámetros a través del diálogo con el usuario, se encargue tanto de configurar como de ejecutar el entrenamiento, mostrando únicamente los resultados relevantes al usuario de forma

clara y comprensible. Esta automatización completa es especialmente útil para usuarios sin recursos técnicos, ya que ocultaría la complejidad y facilitaría una experiencia fluida, enfocada en la utilidad del modelo seleccionado. Esta unificación permitiría simplificar aún más el proceso, reduciendo barreras técnicas y favoreciendo la accesibilidad, ya que habría una única herramienta, en vez de las dos que hemos desarrollado en este TFG, quitando además la intervención del usuario de llevar el .yaml desarrollado por la primera herramienta a la segunda.

Finalmente, sería interesante explorar formas de optimizar el rendimiento del sistema, totalmente capaz con la rápida evolución de LLMs. Esto ampliaría aún más el alcance del proyecto, haciéndolo importante en contextos educativos, sociales o profesionales donde los recursos son escasos pero la necesidad de herramientas accesibles de IA es alta.

Estas líneas de mejora no solo permitirían ampliar el potencial de la herramienta, sino también reforzar su objetivo principal, acercar la IA a todas las personas, independientemente de su formación o recursos.

Referencias

[1] Zhou, Z.-H. (2021). *Machine Learning*. Springer Nature. https://doi.org/10.1007/978-981-15-1967-3

[2] Duque, R., Tirnauca, C., Palazuelos, C., Casas, A., López, A., & Pérez, A. (2025). *Bridging AutoML and LLMs: Towards a framework for accessible and adaptive machine learning*. In J. Filipe, M. Smialek, A. Brodsky, & S. Hammoudi (Eds.), *Proceedings of the 27th International Conference on Enterprise Information Systems (ICEIS 2025), Porto, Portugal* (Vol. 1, pp. 959–964). SCITEPRESS. https://doi.org/10.5220/0013448500003929

[3] OpenAI. (2023). *GPT-4 Technical Report* https://doi.org/10.48550/arXiv.2303.08774

[4] Google DeepMind. (2024). *Gemini: A family of multimodal language models*. https://deepmind.google/technologies/gemini. Visto por última vez el 11/05/2025.

[5] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., ... & Jegou, H. (2023). *LLaMA: Open and Efficient Foundation Language Models*. arXiv preprint arXiv:2302.13971. https://arxiv.org/abs/2302.13971

[6] Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., ... & Wen, J. R. (2023). *A Survey of Large Language Models*. arXiv preprint arXiv:2303.18223. https://arxiv.org/abs/2303.18223

[7] Escalante, H. J. (2020). *Automated Machine Learning—A brief review at the end of the early years*. arXiv preprint arXiv:2008.08516. https://arxiv.org/abs/2008.08516

[8] Meta AI. (2024). *LLaMA: Open foundation and instruction models*. Meta AI. https://ai.meta.com/llama. Visto por última vez el 15/05/2025.

[9] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., & Lample, G. (2023). *LLaMA: Open and Efficient Foundation Language Models*. arXiv. https://doi.org/10.48550/arXiv.2302.13971

[10] Google. (2024). *Gemma: Lightweight, open models from Google*. https://ai.google.dev/gemma. Visto por última vez el 11/06/2025.

[11] Dadashi, R., & Hussenot, L. (2025). *Gemma 3 Technical Report*. arXiv. https://arxiv.org/abs/2503.19786

[12] Ollama. (2024). *Run Llama 2, Mistral and other models locally*. https://ollama.com. Visto por última vez el 18/05/2025.

[13] Balarabe, T. (2025, marzo 24). *What is Ollama: Running Large Language Models Locally*. Medium https://medium.com/@tahirbalarabe2/what-is-ollama-running-large-language-models-locally-e917ca40defe. Visto por última vez el 18/05/2025.

[14] Uber Technologies. (2024). *Ludwig: Code-free deep learning toolbox*. https://ludwig.ai. Visto por última vez el 19/05/2025.

[15] Molino, P., Dudin, Y., & Miryala, S. S. (2019). *Ludwig: a type-based declarative deep learning toolbox*. arXiv preprint arXiv:1909.07930. https://arxiv.org/abs/1909.07930

[16] Google. (2024). *Google Colab*. https://colab.research.google.com. Visto por última vez el 11/06/2025.

[17] Sukhdeve, S. R., & Sukhdeve, S. S. (2023). *Google Colaboratory*. In Google Cloud Platform for Data Science (pp. 11–34). Apress. https://link.springer.com/chapter/10.1007/978-1-4842-9688-2_2

[18] Streamlit Inc. (2024). *Streamlit: The fastest way to build and share data apps*. https://streamlit.io. Visto por última vez el 11/06/2025.

[19] Sharma, R. (2022). *Web Application Development with Streamlit: Develop and Deploy Secure and Scalable Web Applications to the Cloud Using a Pure Python Framework*. Packt Publishing. https://books.google.es/books?id=KDfZEAAAQBAJ

[20] Hugging Face. (2024). *Hugging Face: The AI community building the future.* https://huggingface.co. Visto por última vez el 11/06/2025.

[21] Castaño, J., Martínez-Fernández, S., Franch, X., & Bogner, J. (2023). *Analyzing the Evolution and Maintenance of ML Models on Hugging Face*. arXiv preprint arXiv:2311.13380. https://arxiv.org/abs/2311.13380

[22] Python Software Foundation. (2023). *Python (versión 3.10) [Lenguaje de programación]*. https://www.python.org. Visto por última vez el 11/06/2025.

[23] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-T., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. arXiv preprint arXiv:2005.11401. https://arxiv.org/abs/2005.11401

[24] Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., & Neubig, G. (2021). *Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing*. arXiv preprint arXiv:2107.13586. https://arxiv.org/abs/2107.13586

[25] IBM. (s.f.). *What is a machine learning pipeline?* Recuperado de https://www.ibm.com/think/topics/machine-learning-pipeline. Visto por última vez el 11/06/2025.

[26] Iqbal, F. N. (2023). *A brief introduction to application programming interface (API)*. AIMS Academy. https://doi.org/10.5281/zenodo.10198423

[27] Van Belle, V., Van Calster, B., Timmerman, D., Bourne, T., & Valentin, L. (2014). *Diabetes* (Version 1) [Conjunto de datos]. OpenML. https://www.openml.org/d/37

[28] Dua, D., & Graff, C. (1996). *breast-w* (Version 2) [Conjunto de datos]. OpenML. https://www.openml.org/d/15

[29] Tjen-Sien Lim (1987). *cmc* (Versión 1) [Conjunto de datos]. OpenML. https://www.openml.org/d/23

[30] Quinlan, J. R., & Garvan Institute. (1987). *Hypothyroid* (Version 1) [Conjunto de datos]. OpenML. https://www.openml.org/d/57

[31] BUPA Medical Research Ltd. Donor: Richard S. Forsyth. (1990). *liver-disorders* (Versión 1) [Conjunto de datos]. OpenML. https://www.openml.org/d/8

[32] Sam Gordon. (2024). *BrisT1D Blood Glucose Prediction Competition* [Conjunto de datos]. Kaggle. https://www.kaggle.com/competitions/brist1d

[33] Umut Toygar. (2022). *COVID-19 Death Predictions* [Conjunto de datos]. Kaggle. https://www.kaggle.com/competitions/Covid19-Death-Predictions

[34] ntuee . (2021). *ML2021Spring-hw1: COVID-19 Cases Prediction* [Conjunto de datos]. Kaggle. https://www.kaggle.com/competitions/ml2021spring-hw1

[35] Powers, D. M. W. (2011). Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2(1), 37–63. https://www.researchgate.net/publication/220766087

[36] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. https://www.deeplearningbook.org/

[37] Saito, T., & Rehmsmeier, M. (2015). The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLOS ONE*, 10(3), e0118432. https://doi.org/10.1371/journal.pone.0118432

[38] Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4), 427–437. https://doi.org/10.1016/j.ipm.2009.03.002

[39] Chai, T., & Draxler, R. R. (2014). Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature. *Geoscientific Model Development*, 7(3), 1247–1250. https://doi.org/10.5194/gmd-7-1247-2014

[40] HUGGING FACE, https://huggingface.co/spaces/nestor208/TFG. Visto por última vez el 11/06/2025.

[41] GITHUB, https://github.com/nestormiguel208/tfg. Visto por última vez el 11/06/2025.

Lista de Acrónimos

API Application Programming Interface

AUTOML Automated Machine Learning

CSV Comma-Separated Values

GGT Gamma-Glutamyl Transferase

GPU Graphics Processing Unit

GUI Graphical User Interface

HCAI Human-Centered Artificial Intelligence

IA Inteligencia Artificial

IMC Índice de Masa Corporal

LLM Large Language Model

LLaMA Large Language Model Meta AI

ML Machine Learning

NIDKK National Institute of Diabetes and Digestive and Kidney Diseases

PLN Procesamiento de Lenguaje Natural

RAG Retrieval-Augmented Generation

RAM Random Access Memory

RMSE Root Mean Square Error

ROC- AUC Receiver Operating Characteristic-Area Under the Curve

TFG Trabajo Fin de Grado

TPU Tensor Processing Unit

UCI Unidad de Cuidados Intensivos

WP Work Packages

YAML Yet Another Markup Language