UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

IMPLEMENTACIÓN DE MODELOS DE IA Y TASKS EN DASHAI

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

OZIEL SALVADOR AGUILERA FARÍAS

PROFESOR GUÍA:
FELIPE BRAVO MÁRQUEZ

MIEMBROS DE LA COMISIÓN:
MAURICIO CERDA VILLABLANCA
FEDERICO OLMEDO BERÓN

SANTIAGO DE CHILE
2023

Resumen

DashAI es un *framework* en desarrollo implementado en Python, que aspira a facilitar el entrenamiento, la evaluación y la comparación de diversos modelos de aprendizaje automático (ML) para un amplio espectro de tareas, todo a través de una interfaz gráfica de usuario (GUI). Es un proyecto colaborativo con contribuciones de ingenieros y memoristas, cada uno encargado de distintas facetas del *framework*. El objetivo es que *DashAI* sea *open source*, orientado a tareas, interoperable y extensible, con una interfaz gráfica interactiva.

Debido a que *DashAI* esta en desarrollo todavía muestra varias limitaciones notables, principalmente una escasez de tareas y modelos disponibles. Actualmente, solo cuenta con la tarea de clasificación tabular y modelos de ML clásicos como *Random Forest*, KNN y SVM. Esta carencia es preocupante dado que la diversidad de tareas y modelos es crucial por diversas razones: abordar una variedad de problemas que la IA enfrenta en la realidad, atender a la complejidad variable de los problemas con modelos desde los más sencillos hasta los más sofisticados como los *transformers*, permitir la experimentación con distintas técnicas de ML para adaptarse mejor a las necesidades del usuario y proporcionar una plataforma educativa que permita a los usuarios aprender sobre los puntos fuertes y débiles de diferentes modelos y tareas, y cuándo y cómo aplicarlos.

Para fortalecer el *framework* se incorporaron tareas adicionales: clasificación de imágenes, clasificación de texto y traducción. Asimismo, se añadieron los modelos ViT, DistilBert y uno específico para la traducción de inglés a español del *Tatoeba-Challenge*, cada uno correspondiente a una tarea respectiva. Todos estos modelos se integraron mediante la librería *transformers* de *HuggingFace*. Además, se añadieron métricas en *DashAI* y se realizaron modificaciones en las etapas previas al entrenamiento para mejorar sus funcionalidades.

Para concluir, las modificaciones e implementaciones se llevaron a cabo de manera exitosa, permitiendo el entrenamiento de modelos más sofisticados a través de la interfaz de *DashAI* y ofreciendo una mayor variedad de tareas y métricas para evaluar el rendimiento de estos modelos. Sin embargo, es importante destacar que, dado el alcance de este trabajo, existen numerosos aspectos que no se han abordado en profundidad, debido a que son trabajos en desarrollo de otros memoristas.

In case I don't see ya, good afternoon, good evening, and good night!

- Truman Burbank

Agradecimientos

En primer lugar, quiero agradecer a mis padres, quienes se aseguraron de que nunca me faltara nada y me brindaron la oportunidad de estudiar. A mi Papi y Mami, por su dedicación en mi crianza y por permitirme estar con ellos durante estos 23 años de mi vida.

Quisiera expresar mi gratitud a mis profesores del DCC por su entrega y dedicación al enseñar y explicar el contenido de las asignaturas. Un agradecimiento especial a Felipe Bravo por su constante apoyo y tiempo durante la escritura de esta memoria.

También quiero agradecer al equipo de *DashAI*: Rodrigo, Maximiliano, Nacho, Pablo y Cristian; por su amabilidad, disposición para ayudar y su paciencia al corregir errores. Aprecio sinceramente a todo el equipo de Unholster por brindarme el tiempo necesario para concluir este trabajo. Además, quiero agradecer a mis amigos de la *Primera Fika VIP*, quienes estuvieron a mi lado durante toda mi etapa universitaria, con almuerzos y carretes cuando era posible.

Además, quiero agradecer a mis amigos Yorichi, Pyro, Pato y Checho, quienes estuvieron a mi lado durante uno de los momentos más difíciles de mi vida. Una dedicatoria especial a Ana Banana por las conversaciones, películas y salidas que compartimos durante todo este tiempo, que siempre elevaban mi ánimo. En esa misma línea, deseo dedicar este trabajo a mis queridas mascotas Niña, Rudy y Molly, quienes estuvieron a mi lado durante este proceso. A Niña y Rudy, aunque ya no estén físicamente conmigo, les envío mis más tiernas caricias allá donde estén, llevando siempre en mi corazón su inolvidable compañía.

Finalmente, deseo expresar mi gratitud a aquellos que, aunque no estén conmigo, han sido una parte importante de mi paso por la Universidad y de mi vida en general, y han contribuido a ser la persona que soy hoy.

Tabla de Contenido

1. Introducción **1**

1.1. Motivación 1

1.2. Objetivos 5

1.2.1. Objetivo General 5

1.2.2. Objetivos específicos 5

1.2.3. Metodología de desarrollo 6

1.3. Alcances y limitaciones 7

1.4. Estructura del documento 8

2. Estado del Arte **9**

2.1. Soluciones existentes 9

2.1.1. Librerías científicas 9

2.1.2. Herramientas de escritorio 12

2.1.3. Herramientas en la nube 13

3. Situación inicial **15**

3.1. Frontend 15

3.2. Backend 17

3.2.1. Objeto configurable 17

3.2.2. Componentes 18

3.2.3. Sistema de registro de componentes 19

4. Problemas **20**

4.1. Descripción 20

4.1.1. Dataloaders 20

4.1.2. Datasets 20

4.1.3. Tareas 21

4.1.4. Modelos 21

4.1.5. Métricas 21

5. Solución **22**

5.1. Desarrollo de apoyo 22

5.1.1. Dataset 22

5.1.2. Métricas 24

5.2. Desarrollo principal 26

5.2.1. Tareas 26

5.2.2. Modelos 27

5.3. Refactoring 30

5.3.1. Dataloaders 30

5.3.2. Tareas 34

5.3.3. Modelos 35

5.4. Caso de uso 37

6. Evaluación **39**

6.1. Testing de componentes 39

6.1.1. Dataloaders 39

6.1.2. DashAIDataset 39

6.1.3. Tareas 40

6.1.4. Modelos 41

6.1.5. Métricas 41

6.2. Reproducibilidad de modelos del estado del arte 42

7. Conclusiones **49**

7.1. Retrospectiva 49

7.2. Trabajo Futuro 50

Bibliografía **53**

Índice de Ilustraciones

3.1. Vista principal de *DashAI* 15
3.2. Elección de la tarea 16
3.3. Elección del *dataloader* según el formato 16
3.4. Carga de datos 16
3.5. Elección de hiperparámetros para modelo SVM 17

5.1. Diagrama de secuencia de uso de los componentes 38

6.1. Interfaz principal de *DashAI*. 42
6.2. Interfaz de carga de *datasets*. 43
6.3. Selección de tarea para los *datasets*. 43
6.4. Selección de *dataloader* para los *datasets*. 44
6.5. Configuración del *dataloader* para los *datasets*. 44
6.6. Selección de la tarea y nombre del experimento. 45
6.7. Selección del *dataset* para el experimento. 45
6.8. Inclusión del modelo para entrenamiento. 46
6.9. Configuración del modelo. 46
6.10. Ejecución de experimentos. 47
6.11. Pestaña de resultados. 47
6.12. Métricas del experimento 48

Capítulo 1

Introducción

El primer capítulo de este documento se estructura en cuatro secciones fundamentales: motivación, objetivos, alcances y estructura del documento.

En la sección de motivación, se proporcionará el impulso y la inspiración detrás de este trabajo, ofreciendo una visión del contexto actual en el campo de la Inteligencia Artificial y explorando las circunstancias y los retos que llevaron a la concepción del *framework DashAI*.

Posteriormente, en la sección de objetivos, se delineará una lista detallada de las metas y las aspiraciones que se pretenden alcanzar a lo largo de esta memoria. Esta sección servirá como una guía que describa las tareas primordiales que se llevarán a cabo durante el desarrollo del presente.

A continuación, en la sección correspondiente a los alcances y limitaciones, se delineará de manera explícita el ámbito de esta tesis. Dado que *DashAI* es una iniciativa de investigación colaborativa con múltiples contribuyentes que operan simultáneamente, es esencial señalar que ciertos componentes del *framework* serán desarrollados de manera exhaustiva por colaboradores simultáneos.

Finalmente, en la sección de estructura del documento, se presentará una descripción concisa de cada uno de los capítulos que componen esta tesis. Esto permitirá una comprensión clara y estructurada de los temas que se tratarán en cada sección del documento.

1.1. Motivación

En la última década se ha estado viviendo un momento histórico en el que la Inteligencia Artificial ha experimentado un crecimiento exponencial en diversas áreas y se espera que esto siga así, de hecho en 2021, el tamaño del mercado de IA fue valorado en 328.34 mil millones de dólares y se proyecta que crezca a 2025.12 mil millones de dólares para 2030 [17]. Esto es debido a que la IA está llegando y transformando diversas industrias. En el área de *e-commerce*, por ejemplo, la redacción de textos asistida por IA está cambiando la forma en que las marcas fabrican copias de ventas; las herramientas de escritura con

inteligencia artificial pueden crear copias de marketing en segundos y además los minoristas de comercio electrónico están aumentando el uso de chatbots y otros asistentes virtuales, que brindan soporte 24/7 a todos sus clientes en línea [34, 35]. Otra área es el ámbito de diseño de *hardware*, en donde empresas como IBM están desarrollado nuevos dispositivos y arquitecturas para soportar el procesamiento y complejos cálculos que requiere desarrollar inteligencia artificial [24]. Y por último, es importante mencionar a la IA generativa la cual ha llamado la atención de todo el mundo con ChatGPT de OpenIA [20], Bard de Google [8] y muchísimas más; de estas herramientas se espera que en 2032 lleguen a captar un mercado valorado en 151.9 mil millones de dólares [13]. Todos estos datos y proyecciones económicas refuerzan la idea de que estamos en un boom de la IA que se espera que siga en crecimiento para desarrollar un papel cada vez más activo en nuestro día a día.

A pesar de estos avances, es importante poner mucha atención a la accesibilidad que tendrá. Como cualquier tecnología poderosa, existe el riesgo de que solo un puñado de personas o empresas con recursos suficientes puedan beneficiarse de ella, por lo tanto, es esencial implementar medidas que democraticen el acceso a esta para prevenir la formación de una "élite de la IA". Algunos de los motivos por los cuales se debe poner énfasis en este aspecto son los siguientes:

- Dos de los mayores desafíos a los que la humanidad se ve enfrentada son el cambio climático y la prevención de enfermedades, y en ambos campos la IA puede ayudar a encontrar soluciones [11, 25]. Y solo a través de la incorporación de una amplia gama de perspectivas y experiencias se podrán desarrollar soluciones que aborden de manera efectiva estos problemas.
- Para prevenir la acumulación de influencias y garantizar una sociedad equitativa, es fundamental que todos tengan acceso a la IA. En un tiempo en que los datos se han convertido en el motor de la economía moderna [23], asegurar un acceso inclusivo a esta tecnología puede ser un medio efectivo para igualar las condiciones de competencia y proporcionar oportunidades a todos, sin importar su procedencia o sus circunstancias personales.
- Por último, pero no menos importante, facilitar el acceso a la IA puede fomentar la creatividad, ya que es más probable que surjan nuevas ideas y soluciones cuando más personas y organizaciones puedan utilizarla. La historia de la tecnología ha enseñado que la innovación a menudo surge de lugares inesperados, y al democratizar su acceso, se aumentan las posibilidades de avances disruptivos y significativos.

Por lo tanto, aunque la Inteligencia Artificial ofrece muchas ventajas y oportunidades, es fundamental que todos tengan acceso a ella; esto no solo es una cuestión de justicia, sino también una necesidad esencial para desbloquear completamente el potencial de la IA y forjar un futuro beneficioso para todos. Afortunadamente, este tema es de gran importancia para muchos grupos y organizaciones, de hecho existen numerosas herramientas, desde librerías hasta softwares completos, que son gratuitas y de código abierto. Algunos ejemplos incluyen *TensorFlow* [7], *Scikit-learn* [22] y *WEKA* [14]. Sin embargo, estas herramientas no están exentas de dificultades y desafíos, por ejemplo, el uso de librerías como *TensorFlow* y *Scikit-learn* requiere habilidades de programación, mientras que herramientas como *WEKA* pueden

ser limitadas en cuanto a las tareas que pueden resolver. En esa misma línea, en la nube existen herramientas muy fáciles de usar como *VertexAI* [5] o *AutoML* [2], pero estas no son de código abierto y requieren un pago para su uso.

De esta necesidad surge *DashAI*, un *framework* desarrollado en Python que permite el entrenamiento, evaluación y comparación de varios modelos de *machine learning* para distintas tareas, utilizando los datos suministrados por el usuario. Además, estará equipado con módulos para el preprocesamiento de datos, optimización de hiperparámetros e incluso permitirá la utilización de métodos para explicar los modelos (lo que es conocido como *explainable artificial intelligence*). El uso de esta herramienta se realizará mediante una interfaz gráfica de usuario, donde se podrán cargar datos en diferentes formatos, seleccionar la tarea a realizar, el modelo a utilizar y sus hiperparámetros, entre otras funciones, para que de este modo esté al alcance de usuarios sin conocimientos de programación. Las características clave de *DashAI* serán las siguientes:

1. Código abierto: *DashAI* no tendrá restricciones de uso porque será de código abierto. Por el contrario, se espera que quienes quieran complementar el *framework* agreguen e integren nuevas técnicas (modelos) y posibles problemas a resolver (tareas). De esta manera, se espera que la comunidad mantenga y mejore el *software* y como consecuencia, sea posible que se mantenga al día con el estado del arte y nuevas innovaciones en el campo [27].
2. Orientado a tareas: *DashAI* está diseñado con una orientación basada en tareas específicas, es decir, integra modelos de *machine learning* (ML) para abordar problemas particulares, también referidos como tareas. El usuario tiene la libertad de seleccionar la tarea que desea resolver, cargar un conjunto de datos apropiado y elegir un modelo para entrenar. Como producto final del proceso de entrenamiento, se obtiene un modelo capaz de realizar predicciones en donde el alcance y naturaleza de estos resultados es determinado por el tipo de problema seleccionado. De esta forma, los usuarios tienen la posibilidad de interactuar con datos nuevos, explorando y experimentando con una vasta selección de modelos y librerías de ML aplicables a una misma tarea y conjunto de datos.
3. Interoperabilidad: dado que existen diversas librerías de programación altamente utilizadas para el diseño de modelos de ML (por ejemplo, *Pytorch*, *TensorFlow* o *HuggingFace* [31]), que operan de manera independiente unas de otras, es imprescindible que *DashAI* pueda integrar estos entornos bajo una interfaz unificada. De esta manera, para el usuario se elimina la necesidad de tener conocimientos específicos en cada una de ellas. Esta es una característica crucial de la herramienta, ya que la adopción de la gran cantidad de tecnologías disponibles representa una barrera significativa para muchas organizaciones y *DashAI* aspira a mitigar este obstáculo facilitando la utilización de estas tecnologías de manera más accesible y efectiva.
4. Extensibilidad: es fundamental que una herramienta como *DashAI* sea fácilmente extensible para incorporar rápidamente modelos más avanzados debido a la variedad de campos de aplicación de ML y el acelerado avance científico y tecnológico en todas sus subáreas. Esta es la razón principal de que *DashAI* sea un *software* de código abierto y alojado en un repositorio público de Github (en esta se proporcionan instrucciones detalladas sobre cómo crear extensiones externas en ese repositorio). Con esta estructura,

cualquier experto en el campo puede desarrollar nuevos modelos e incluso ampliar la plataforma para resolver nuevos problemas. De esta manera, se facilita la disponibilidad de estas innovaciones para usuarios que pueden no tener la misma profundidad de conocimientos en el campo.

5. Interfaz gráfica interactiva: *DashAI* tendrá una interfaz gráfica para interactuar con los datos que se ingresan y apoyar la experimentación con modelos de ML con el objetivo de reducir las barreras de entrada. Cada uno de los pasos de este proceso será apoyado por el *software*, incluida la carga de datos, la selección de características, la selección de modelos y la configuración de parámetros, y la comparación de modelos a través de visualizaciones de métricas de desempeño. El usuario contará con elementos gráficos que ayudarán a sus prácticas de trabajo a elegir el modelo más adecuado en cada etapa del proceso de ejecución y experimentación.

Como *DashAI* está en desarrollo, aún tiene muchas deficiencias, y una de las más importantes es que cuenta con pocas tareas y modelos para utilizar, de una manera más concreta, solamente tiene disponible la tarea de clasificación tabular y modelos de ML como *random forest* [9], KNN [16] y SVM [12]. Esto es un problema, ya que tener diversas tareas y modelos es muy importante:

1. Diversidad de problemas: el procesamiento del lenguaje natural y la clasificación de imágenes son solo algunos de los muchos problemas que enfrenta la IA en el mundo real. Por lo tanto, proporcionar una variedad de tareas es fundamental para hacer que el *framework* sea útil y valioso para una amplia gama de usuarios.

2. Complejidad variable de los problemas: no todos los problemas requieren los modelos más complejos y potentes. Algunos pueden ser abordados de manera efectiva con modelos de ML clásicos, que pueden ser más fáciles de entender e interpretar. Al ofrecer una gama de modelos desde los clásicos de ML hasta *transformers* [30] es posible adaptarse mejor a las necesidades específicas del usuario y al nivel de complejidad del problema.

3. Cobertura de técnicas: el campo del ML está en constante evolución, y una variedad de modelos pueden ser más apropiados para una variedad de problemas o datos. Al incorporar una variedad de tipos de modelos, los usuarios pueden experimentar con diferentes enfoques y encontrar el que mejor se adapte a sus necesidades, además que según el teorema "*no free lunch*" no existe a priori un modelo mejor que otro [32].

4. Educación y aprendizaje: el proporcionar una variedad de modelos y tareas permitirá que los usuarios aprendan sus puntos fuertes y débiles, así como cuándo y cómo aplicarlos, lo que lo convierte en una herramienta educativa en cierto punto.

Por lo tanto, para el proyecto *DashAI* es muy importante que se agreguen nuevas tareas y modelos, lo cual es el foco central de esta memoria. Con el objetivo de lograrlo, se hará uso de diversas librerías que ya implementan modelos de distintos tipos. Los modelos y tareas específicos que se agregarán serán descritos en la sección 1.2. Asimismo, en esta tesis se desarrollarán componentes para garantizar una adecuada ejecución de las etapas previas y posteriores al entrenamiento de un modelo. El proceso previo al entrenamiento implica la

correcta carga y preparación de los datos, que si bien existe en el estado actual, requiere de ajustes y mejoras. La etapa posterior al entrenamiento implica la incorporación de un sistema básico de métricas para los modelos, aspecto esencial en la comparación de modelos y la selección del más apropiado para una tarea específica. Más detalles sobre el orden en el que se abordarán cada uno de estos aspectos se especificarán en la subsección 1.2.3.

Con respecto a los restantes requerimientos asociados al *software*, se cuenta con un equipo de tesistas/memoristas que trabajan en otros aspectos cruciales de *DashAI*, como el desarrollo del *frontend*, la interfaz de usuario, la arquitectura global de *DashAI*, la implementación de la API y otros elementos del *backend* (se ahondará un poco más en el capítulo 3). Es importante señalar que la naturaleza y alcance específico de este trabajo, en relación con las labores de los demás tesistas, serán delineados en la sección 1.3. En dicha sección, se establecerá claramente hasta dónde se extenderán los avances a realizar en el marco de la presente memoria. Además, cabe mencionar que *DashAI* es un proyecto de largo plazo, por lo tanto, su desarrollo y mejoras continuarán más allá de la finalización del trabajo expuesto en este documento.

1.2. Objetivos

1.2.1. Objetivo General

El principal objetivo de esta memoria es dotar a *DashAI* con tareas y modelos de distintas librerias, de esta manera se asegura la interoperabilidad y se pone a prueba la extensibilidad.

1.2.2. Objetivos específicos

Implementar las siguientes tareas:

- Clasificación de imágenes.
- Clasificación de texto.
- Traducción.

Implementar los siguientes modelos:

- ViT [33].
- Distilbert [26].
- *Transformer* para traducción inglés a español del *Tatoeba-Challenge* [29].

Implementar las siguiente métricas:

- *Accuracy* para tareas de clasificación.
- *Precision* para tareas de clasificación.
- *Recall* para tareas de clasificación.
- *F1* para tareas de clasificación.
- *BLEU* para tareas de traducción.
- *TER* para tareas de traducción.

Y además:

- Corregir errores de los *dataloaders* (componentes encargados de la carga de datos).
- Encontrar alguna estructura de datos que tenga lo necesario para utilizarse de manera simple para las tareas y modelos.
- Corroborar que funciona de manera correcta la tarea de clasificación tabular.
- Corroborar que funcionan correctamente los modelos disponibles para la tarea de clasificación tabular.

1.2.3. Metodología de desarrollo

Para cumplir con los objetivos establecidos anteriormente, se propone una metodología de desarrollo compuesta por los siguientes pasos:

1. Iniciar con la mejora de la implementación de los *dataloaders*. Estos componentes son esenciales, ya que son responsables de leer los diversos formatos de datos y transformarlos en una estructura común. Aunque ya se han implementado *dataloaders* para distintos formatos, es necesario realizar mejoras y pruebas adicionales para garantizar su funcionamiento adecuado.
2. En relación con lo anterior, es esencial idear una modificación de la estructura actual de manejo de datos. Este cambio debe tener en cuenta que los datos serán leídos por diferentes tareas y modelos.
3. El siguiente paso implica la creación de componentes asociados a las tareas. Se deben diseñar y desarrollar los métodos correspondientes que estos componentes emplearán.
4. Posteriormente, se debe agregar una lógica base para los modelos. Esto implica implementar métodos generales para los modelos y, al mismo tiempo, agregar modelos específicos.
5. Una vez realizadas estas mejoras, es importante incorporar una lógica para obtener métricas de los modelos. Este paso es crucial para evaluar el rendimiento y la precisión de los modelos implementados.

6. Finalmente, cada una de las mejoras y adiciones antes mencionadas debe estar acompañada de pruebas correspondientes. Esta etapa garantiza que cada componente funcione según lo esperado y en conjunto con el sistema general.

En cuando a la metodología de trabajo con el equipo, tal como se expuso en la sección 1.1, *DashAI* es un proyecto colaborativo en el que interactúan diversas partes interesadas. Para mantener la coherencia y la comunicación fluida, se organizaban reuniones diarias de 20 minutos cada mañana. En estos encuentros, cada participante compartía un informe de las actividades realizadas el día anterior y los planes para el día en curso. Adicionalmente, existían ingenieros designados para revisar y aprobar las *pull requests*, garantizando así una incorporación cuidadosa y efectiva de nuevas funcionalidades a la rama principal del repositorio de *DashAI*.

1.3. Alcances y limitaciones

Dado que *DashAI* es un proyecto de colaboración con múltiples participantes y esta tesis tiene objetivos particulares, se establecerán ciertos alcances específicos. Cabe resaltar que los alcances y limitaciones delineados en esta sección se determinan de acuerdo con lo expuesto en la subsección 1.2.1.

En primer lugar, y de mayor importancia, las tareas seleccionadas se determinaron con el propósito de explorar diversas modalidades de carga de datos al *software*, y en el caso de la traducción, se optó por esta tarea para experimentar con una problemática distinta a la clasificación. La decisión de no incorporar tareas adicionales se basó en la percepción de que el número actual era adecuado según las discusiones mantenidas con el equipo y el profesor guía. Un razonamiento similar se aplicó a los modelos. Se decidió emplear modelos de tipo *transformers* para evaluar si *DashAI* era compatible con el *fine-tuning* de estos, además la implementación de un modelo para cada tarea se consideró una cantidad adecuada también en línea con lo determinado por el equipo y el profesor guía. Los posibles hiperparámetros que se pueden cambiar de un modelo se limitaron según a los más usados en cada uno, ya que a nivel de interfaz todavía no se decidía el conjunto completo de las configuraciones posibles para los modelos. Finalmente, en *DashAI* no es posible utilizar modelos pre-entrenados para predecir, ya que deben pasar por una instancia de entrenamiento (*fine-tuning* en el caso de los pre-entrenados).

En lo que respecta a los *dataloaders*, la única necesidad imperante era implementar el *imagedataloader* para permitir la carga de imágenes en *DashAI* y corregir ciertos errores en las implementaciones existentes. No se contemplaron ni se implementaron funcionalidades adicionales, ya que existe una propuesta de tesis destinada a mejorar la manipulación de datos, incluyendo transformaciones de columnas, visualización de datos, permitir al usuario modificar tipos, transformar columnas a formato *one-hot encoding*, entre otras funcionalidades.

En relación a las métricas, las seleccionadas fueron el resultado de deliberaciones con el equipo de proyecto y el asesor académico. Se optó inicialmente por no convertirlas en "objeto

configurable" (se define en la subsección 3.2.1) debido a que aún no se habían diseñado las interfaces de usuario para incorporar dichas configuraciones.

Algunas otras limitantes más específicas del desarrollo serán expuestas en el capítulo 5, ya que es necesario tener más contexto.

1.4. Estructura del documento

Este documento se divide en siete capítulos. En la sección anterior, se discutió la relevancia de la IA en el contexto actual, la necesidad de una herramienta como *DashAI* además de los objetivos y alcances de esta memoria. En el capítulo 2, se explorarán soluciones existentes que son similares al *framework*. En el capítulo 3, se presenta una inspección de la situación inicial de *DashAI* a nivel de *frontend* y *backend*. Posteriormente, en el capítulo 4, se delinea en detalle los problemas específicos que esta memoria se propone abordar. En el capítulo 5, se discuten los componentes creados y/o modificados para solucionar los problemas mencionados. A continuación, en el capítulo 6, se abordará la evaluación de estos componentes, teniendo en cuenta el objetivo de utilizar algunos modelos del estado del arte en *DashAI*. Finalmente, en el capítulo 7, se presentarán las conclusiones del trabajo, incluyendo tanto las posibles modificaciones que se podrían haber realizado, como los futuros desarrollos a emprender.

Capítulo 2

Estado del Arte

En el siguiente capítulo se indagará más a fondo en algunas soluciones existentes que tienen características similares a *DashAI*.

2.1. Soluciones existentes

Tal como se comentó en la subsección 1.1 en el último tiempo han aparecido varias herramientas que comparten algunas de las características de *DashAI*. Estas pueden divirse en 3 grandes grupos: librerías científicas, herramientas de escritorio y herramientas en la nube.

2.1.1. Librerías científicas

Son librerías de código abierto que se pueden acceder principalmente a través del lenguaje de programación Python. Estas herramientas están dirigidas a usuarios programadores y su principal ventaja es que se pueden utilizar para una variedad de propósitos mediante el desarrollo de *software* personalizado. Las más populares son:

Scikit-Learn

También conocida como *sklearn*, es una librería para el lenguaje de programación Python que ha alcanzado un alto grado de reconocimiento en la comunidad que trabaja con modelos de ML. Aporta una amplia gama de algoritmos de aprendizaje automático supervisado y no supervisado, incluyendo clasificación, regresión, clustering, y reducción de dimensionalidad [22]. Es notable por su diseño limpio y uniforme, que permite a los usuarios aplicar y combinar algoritmos de aprendizaje automático de manera rápida y eficiente. Algunas otras características son:

1. Todos los objetos comparten una interfaz consistente y sencilla. Por lo que es relativamente fácil de utilizar una vez aprendida.
2. Los hiperparámetros de los estimadores pueden ser ajustados proporcionando un control completo sobre estos.
3. Acepta datos en formato *NumPy* [15] o en *Dataframes* de *Pandas* [18].

Tensorflow

Tensorflow es una librería de código abierto desarrollada por el equipo de *Google Brain* diseñada para permitir cálculos numéricos eficientes, con una arquitectura fuertemente orientada hacia las tareas de ML y en particular, hacia el desarrollo y entrenamiento de redes neuronales profundas (*deep learning*) [7]. Algunas características de esta son:

1. Puede ejecutarse en una variedad de plataformas, desde máquinas locales hasta plataformas en la nube. Además, soporta una amplia variedad de lenguajes de programación, siendo Python el más destacado.
2. Permite la computación distribuida y optimizada, lo que facilita la gestión de aplicaciones de aprendizaje automático a gran escala y de alto rendimiento. Esto incluye el soporte para diferentes tipos de *hardware*, desde CPUs y GPUs hasta TPUs.
3. Además de la librería base, el ecosistema de *Tensorflow* incluye muchas otras librerías y extensiones, como *TensorBoard* para la visualización de modelos y métricas, y *TensorFlow Extended* para la implementación de *pipelines* de ML en producción.
4. Gracias a su API de alto nivel, *Keras*, permite un diseño y entrenamiento sencillo y rápido de modelos de *deep learning*, sin sacrificar la capacidad de personalización.

PyTorch

Es una librería desarrollada por *Facebook*, que ha ganado gran popularidad debido a su flexibilidad y eficiencia [21]. Es bastante parecida a *Tensorflow* y cuenta con las siguientes características:

1. Proporciona una interfaz fácil de usar que permite a los usuarios desarrollar modelos de ML de una manera más intuitiva gracias a su estilo de programación imperativo.
2. Utiliza la diferenciación automática y un paradigma de ejecución dinámica para realizar cálculos de gradientes de manera eficiente. Esto es particularmente útil al desarrollar modelos de *deep learning*, ya que simplifica el proceso de entrenamiento y permite la implementación de arquitecturas de redes más complejas.
3. Dado que los modelos se ejecutan inmediatamente en lugar de ser definidos y luego ejecutados (como en *Tensorflow*), la depuración en *PyTorch* puede ser más sencilla y transparente.

4. Al igual que *Tensorflow*, *PyTorch* puede aprovechar las GPUs para acelerar los cálculos numéricos, y también puede ser utilizado en un modo distribuido para entrenar modelos en grandes conjuntos de datos.
5. Se integra perfectamente con el ecosistema de Python, lo que facilita el uso de librerías de análisis de datos como *NumPy* y *Pandas*.

Hugging Face Transformers

Transformers es el nombre de la librería desarrollada por la empresa *Hugging Face* [31] la cual permite utilizar modelos que utilizan la arquitectura de redes neuronales con mecanismo de atención llamada *transformers* [30]. Alguna de sus características son:

1. Proporciona implementaciones de última generación de modelos para diferentes modalidades, como texto, imágenes y audio. Estos modelos pueden ser aplicados en tareas como clasificación de texto, extracción de información, respuesta a preguntas, resumen, traducción, generación de texto en más de 100 idiomas; clasificación de imágenes, detección de objetos y segmentación; reconocimiento automático del habla, clasificación de audio, entre otros.
2. Es compatible tanto con *PyTorch* como con *Tensorflow*, lo que permite a los usuarios elegir la librería que mejor se adapte a sus necesidades.
3. Ofrece acceso a miles de modelos preentrenados en múltiples lenguajes, lo que facilita enormemente la puesta en marcha de proyectos.
4. Cuenta con una comunidad muy activa que constantemente contribuye con nuevos modelos y mejoras.
5. A pesar de la complejidad de los modelos que implementa, es muy fácil de usar y permite a los usuarios centrarse en sus tareas sin tener que preocuparse por los detalles de implementación de los modelos.

Deficiencias

Tal como se vió anteriormente, las librerías anteriores son extremadamente potentes y versátiles. Sin embargo, esto también puede representar desafíos significativos para los nuevos usuarios o aquellos sin una sólida formación en programación y ciencia de datos, ya que el uso de estas requiere una comprensión en profundidad de la programación, a menudo en lenguajes como Python, que pueden ser intimidantes para los recién llegados. Además, cada librería tiene su propio estilo y conjunto de convenciones que los usuarios deben aprender.

Por otro lado, el campo del ML está avanzando a un ritmo vertiginoso, esto significa que los usuarios deben mantenerse al día con una avalancha constante de nuevos modelos, técnicas y librerías. Aunque esto es emocionante desde un punto de vista de la investigación, puede ser abrumador para los practicantes, especialmente aquellos en industrias o roles donde la ciencia de datos es solo una parte de sus responsabilidades.

Por último, ninguna de estas librerías ofrece una interfaz gráfica de usuario (GUI) integrada. Una GUI puede hacer que una herramienta sea más accesible para los no programadores, ya que permite a los usuarios interactuar con la herramienta utilizando elementos visuales como menús y botones en lugar de escribir código. La falta de una GUI en estas significa que los usuarios deben interactuar con ellas a través de la programación, lo que aumenta aún más la barrera de entrada.

Por lo tanto, a pesar del poder y la flexibilidad de estas librerías, la barrera de entrada puede ser alta. Esto puede impedir que personas con habilidades valiosas, pero sin formación en programación, aprovechen plenamente las ventajas que le ofrecen los modelos de ML.

2.1.2. Herramientas de escritorio

Son herramientas de escritorio de código abierto para realizar tareas tradicionales de ML sobre datos tabulares como clasificación, regresión y clustering. Poseen interfaz gráfica de usuario que permite diseñar experimentos de manera interactiva. De estas se pueden mencionar las siguientes:

WEKA

Es un popular *software* para el aprendizaje automático de código abierto y escrito en Java. Desde su creación en la Universidad de Waikato en Nueva Zelanda, *WEKA* ha acumulado una base de usuarios sólida en la comunidad (cuenta con alrededor de 10 millones de descargas a la fecha) de ciencia de datos debido a su facilidad de uso y extensiva gama de algoritmos de ML incorporado [14]. Algunas de sus características son:

1. Cuenta con una interfaz gráfica de usuario que proporciona un acceso fácil e intuitivo a sus funcionalidades. Esto hace que *WEKA* sea especialmente atractivo para los principiantes o aquellos usuarios que prefieren una interacción visual en lugar de programar directamente. También permite a los usuarios seleccionar y configurar modelos, ejecutar experimentos y visualizar los resultados, todo a través de la interfaz gráfica, de hecho esta herramienta fue una de las inspiraciones principales para el proyecto *DashAI*.
2. Proporciona un conjunto de herramientas para el preprocesamiento de datos, como la limpieza de datos, la selección de características, la normalización, la discretización y otras transformaciones que pueden mejorar la calidad de los datos.

RapidMiner

RapidMiner (o *YALE* como era conocido en su primera versión) es una potente plataforma de ciencia de datos y aprendizaje automático que ofrece funcionalidades de extremo a extremo para todo el ciclo de vida del modelado predictivo. Desarrollada originalmente en la Universidad de Dortmund en Alemania, *RapidMiner* ha ganado reconocimiento en la

comunidad de ciencia de datos debido a su enfoque centrado en el usuario y su interfaz gráfica interactiva [19]. Tiene las siguientes características:

1. Al igual que *WEKA* cuenta con una interfaz gráfica de usuario, en ella, los usuarios pueden diseñar y ejecutar flujos de trabajo de análisis de datos de manera intuitiva arrastrando y soltando operadores, configurando sus parámetros y conectándolos entre sí. Este enfoque de "caja de herramientas" permite a los usuarios, tanto principiantes como expertos, experimentar y aprender rápidamente sobre diferentes técnicas de análisis de datos.
2. Cuenta con una amplia gama de operadores para tareas de preprocesamiento de datos, modelos de ML, validación de modelos y visualización de resultados. Su catálogo de operadores es extenso y cubre un amplio espectro de técnicas, desde las más tradicionales hasta algunas más avanzadas.

Deficiencias

Aunque *RapidMiner* y *WEKA* son herramientas potentes y fáciles de usar, presentan ciertas limitaciones. En el caso de *RapidMiner*, a pesar de su amplia gama de operadores, puede no estar al día con las técnicas y métodos más recientes que se manejan hoy en día, además es un producto comercial con una versión gratuita limitada, y las funcionalidades adicionales requieren suscripciones de pago. En el caso de *WEKA*, su dependencia de Java y la limitada capacidad para integrarse con otras librerías y lenguajes de programación populares en el campo de la ciencia de datos, como Python, pueden ser restrictivas y además, su soporte para los métodos más recientes y avanzados puede no ser tan actualizado o extenso. Ambas herramientas tienen limitaciones para interactuar de manera nativa con librerías de *deep learning* basadas en Python y carecen de una forma natural de ejecutar tareas complejas como la traducción automática, el procesamiento de audio y video, y la generación de imágenes, entre otras.

2.1.3. Herramientas en la nube

Son herramientas de *software* ofrecidas como servicios por grandes compañías multinacionales como *Google*, *Amazon* o *Microsoft* que permiten realizar todo el ciclo que siguen los modelos de ML en la nube. Las herramientas más populares de este tipo son:

Amazon SageMaker

Es un servicio de *Amazon* enfocado en aprendizaje automático completamente administrado que permite a los desarrolladores y científicos de datos crear, entrenar y desplegar modelos rápidamente [1].

IBM Watson Studio

IBM Watson Studio es una plataforma que permite a los científicos de datos, desarrolladores y analistas crear, ejecutar y administrar modelos de IA y optimizar decisiones [4].

Google Vertex AI

La plataforma de aprendizaje automático *Google Vertex AI* permite a los usuarios crear, implementar y escalar modelos de ML más rápido. Además, ofrece herramientas para poner en producción estos modelos [5].

HuggingFace AutoTrain

HuggingFace AutoTrain es una herramienta que entrena modelos de última generación sin necesidad de código para tareas de NLP, visión por computador y audio principalmente. Permite encontrar el mejor modelo según los datos y tarea que entrega el usuario [3].

Deficiencias

A pesar de que herramientas en la nube ofrecen una amplia gama de funcionalidades, estas plataformas tienen limitaciones notables. En su mayor parte, estas herramientas requieren una suscripción pagada, lo que puede limitar el acceso a aquellos usuarios que no tienen la posibilidad de costearlas. Además, debido a que su código es privativo, estos *softwares* pueden ser difíciles de extender a nuevas tareas o personalizar para satisfacer necesidades específicas lo que limita su flexibilidad en comparación con las soluciones de código abierto. Otra restricción reside en que dichas plataformas están concebidas para operar en la nube, lo cual puede representar una barrera en el contexto de organizaciones con información delicada y que debido a estrictas normativas de seguridad y privacidad se exige que los datos se procesen localmente y no sean transmitidos a servidores remotos.

Capítulo 3

Situación inicial

En el siguiente capítulo se mostrará el estado en el cual estaba *DashAI* a nivel de *frontend* y *backend* en el momento que se inició esta memoria.

3.1. Frontend

Si bien la explicación detallada del *frontend* excede el alcance de esta memoria, es esencial mencionar su funcionalidad clave. Permite la creación, configuración y ejecución de experimentos con modelos de *machine learning* sin necesidad de programación directa. Facilita la visualización de métricas y resultados para comparar diferentes modelos aplicados a la misma tarea. Su comunicación con el *backend* se realiza a través de una API. A continuación, se presentan algunas vistas de la interfaz del *software*:

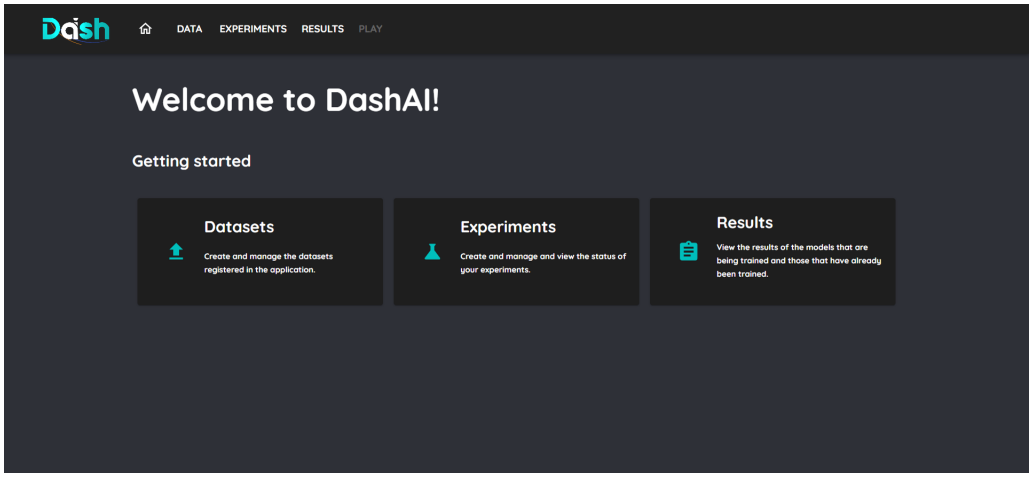

Figura 3.1: Vista principal de *DashAI*

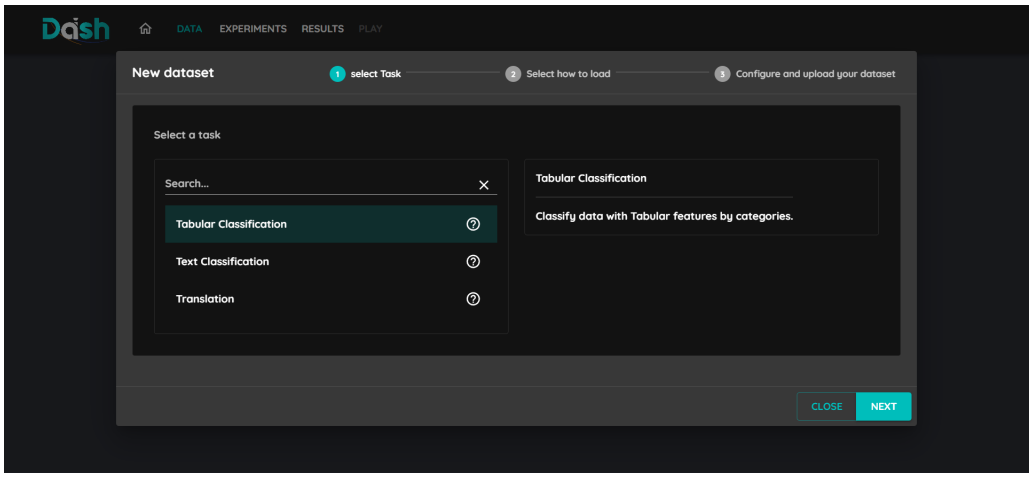

Figura 3.2: Elección de la tarea

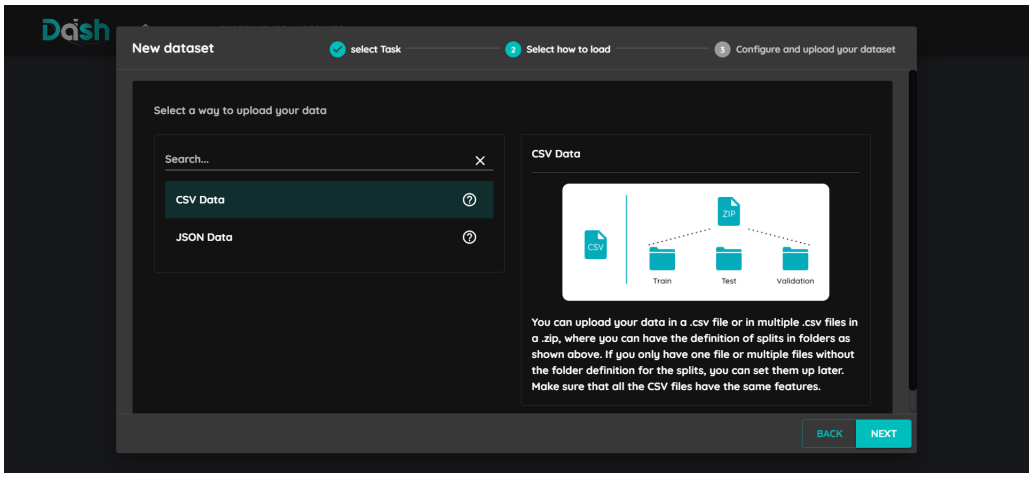

Figura 3.3: Elección del *dataloader* según el formato

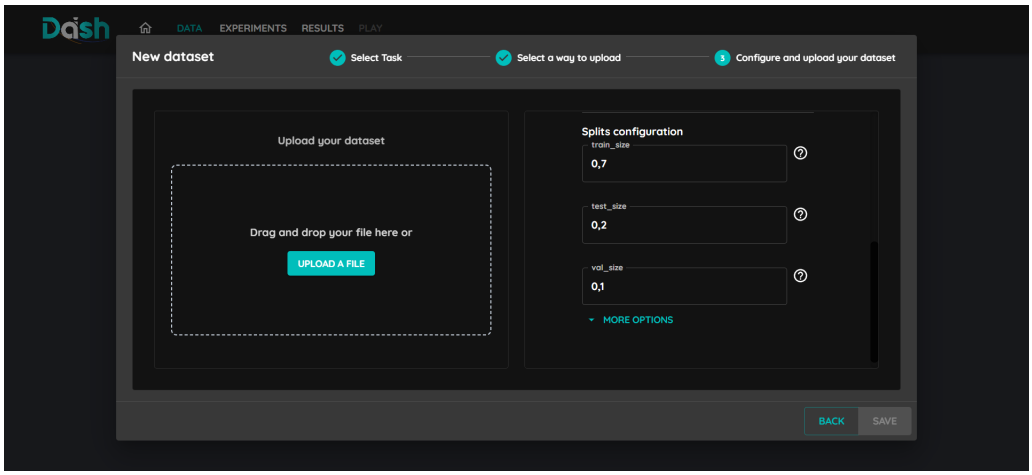

Figura 3.4: Carga de datos

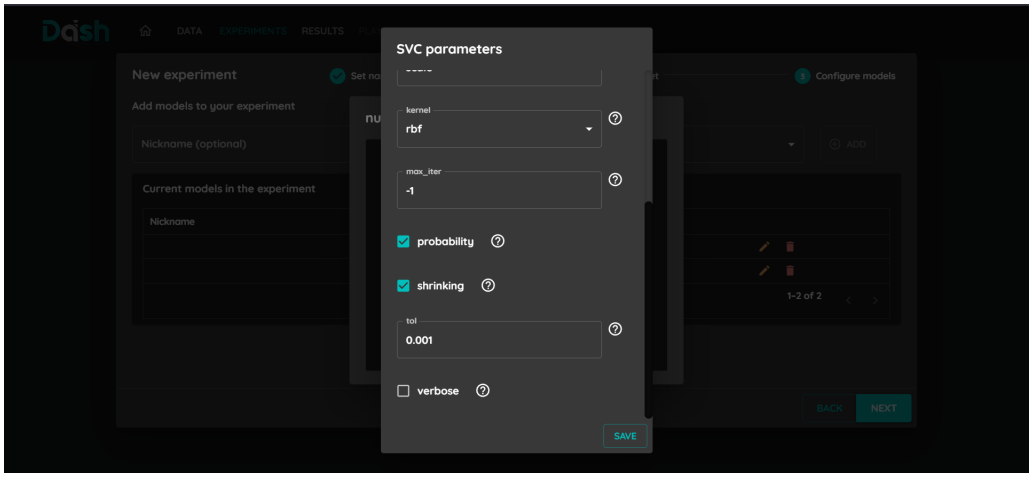

Figura 3.5: Elección de hiperparámetros para modelo SVM

3.2. Backend

Está escrito en el lenguaje de programación Python y su principal función podría resumirse en unificar diversas librerías de *machine learning* con distintos componentes. La conexión entre el *backend* y la GUI se realiza a través de una API con distintos *endpoints*. Estos *endpoints* permiten una serie de acciones fundamentales: la carga de datos al sistema, la visualización de las tareas y modelos disponibles que se pueden utilizar, la ejecución de los algoritmos de ML internamente y la provisión de las métricas de evaluación que resultan de dicho proceso. Por otro lado, cuenta con una base de datos que permite administrar conjuntos de datos etiquetados (ya sean cargados en el momento o previamente) necesarios para el entrenamiento de los modelos. Asimismo, en la base de datos se guardan resultados finales y parciales, lo que permite reanudar sesiones iniciadas anteriormente. Esta característica es especialmente útil para comparar distintas ejecuciones, mostrando información detallada sobre el mejor modelo, incluyendo sus parámetros, resultados, la fecha en la que fue entrenado y datos relevantes sobre su proceso.

En el párrafo anterior se proporcionó una visión general de los tipos de *endpoints* que ofrece la API de DashAI y de lo que se almacena en su base de datos, no se ahondará más en este aspecto debido a que es tópico principal de otra tesis la cual está en desarrollo. A continuación, en las siguientes subsecciones se explorarán en detalle las partes más relevantes del *backend* que conciernen a esta memoria y sus alcances.

3.2.1. Objeto configurable

Dentro de la arquitectura del *backend*, se identifican ciertos componentes como "objetos configurables". Esta designación emula la flexibilidad proporcionada por las librerías científicas, donde los modelos de ML presentan alta configurabilidad, permitiendo una optimización exhaustiva de los hiperparámetros para encontrar la solución más adecuada al problema en cuestión. Para preservar esta capacidad de configuración, se introduce la abstracción de "objetos configurables", evitando una sobrecarga de complejidad en el frontend.

Los "objetos configurables" proporcionan una estructura que especifica un conjunto de parámetros susceptibles de ser configurados por el usuario. Además, detalla el tipo de parámetro esperado e impone las restricciones correspondientes. Este esquema se materializa a través de un *JSON Schema*, el cual es interpretado por la interfaz de *DashAI*, permitiendo su visualización y configuración por parte del usuario. De esta manera, el *frontend* está habilitado para solicitar o permitir al usuario la entrada de valores para uno o más parámetros de un componente.

Adicionalmente, presentan una propiedad de recursividad. En otras palabras, un "objeto configurable" puede requerir o incluir otro "objeto configurable" dentro de sus parámetros. Un ejemplo ilustrativo de esta situación se observa en un modelo de clasificación de texto, el cual requiere de un tokenizador para procesar las entradas de texto. En este caso, el tokenizador se convierte en un parámetro del modelo y, al ser un "objeto configurable", requiere que el usuario introduzca sus propios parámetros, los cuales son independientes del modelo que lo invoca.

3.2.2. Componentes

A continuación se enumerarán cada uno de los componentes del *backend* de *DashAI*, los cuales se representan como clases de Python.

Dataset

Los *dataset* son guardados como objetos Dataset de la librería *datasets* de *Huggingface*, que a su vez consisten en tablas de *Apache Arrow*. Estas estructuras son altamente flexibles y eficientes, permitiendo compatibilidad con las distintas tareas y modelos. Este componente no es extensible, es solamente la estructura de datos que maneja los datos durante el flujo.

Dataloader

Componente extensible, corresponden a objetos responsables de recibir como entrada datos en distintos formatos y transformarlos en la representación interna de *DashAI* (el componente anterior). Los *dataloaders* dependen de la tarea que se esté intentando resolver y por ende son compatibles con una o más de ellas. Internamente utilizan el método load_data de la librería *datasets*.

Task

Componente extensible, su responsabilidad es definir la estructura del *input* y *output* de la tarea por resolver. Esto se traduce en definir el tipo de datos que recibirán los métodos *fit* y *predict* de los modelos disponibles para la tarea. Solo existe la tarea de clasificación tabular.

Model

Componente extensible, es un objeto *wrapper* de algún modelo/metamodelo que define cómo entrenar y predecir según la tarea. Solo existen los modelos *SVM*, *KNN* y *Random Forest* desde la librería *Scikit-learn*.

3.2.3. Sistema de registro de componentes

También *DashAI* cuenta con lo que se denomina un registro de componentes, la idea de este sistema es que exista un registro para cada uno de los componentes extensibles de *DashAI* (*tasks*, *models* y *dataloaders*) que se encuentren disponibles. Este sistema de registros se ejecuta al principio de la ejecución del *software* y se va actualizando a medida que el usuario instala "plugins". Para ello, se creó un paquete de Python llamado "registries" donde se encuentran los registros de cada componente extensible y un archivo "registration" donde se inicializan los registros y también es el encargado de actualizarlos. Junto a esto, los componentes que requieren compatibilidad con una *task* específica, es decir, los *models* y *dataloaders* son evaluados mediante introspección al momento de añadirse a su registro correspondiente. Este proceso se encarga de comprobar la compatibilidad de ciertos componentes con diferentes *tasks*, si se detecta que un componente es compatible con una *task* determinada, entonces se incorpora al registro de *tasks*. Es importante notar que este registro no solo enumera las *tasks* disponibles, sino también los componentes que pueden interactuar eficazmente con cada una de ellas.

Capítulo 4

Problemas

4.1. Descripción

Considerando que ya se tienen conocimientos sobre la estructura y componentes de *DashAI*, es momento de pasar a enumerar los diversos problemas que tiene en las áreas que conciernen para esta memoria.

4.1.1. Dataloaders

- El método *load_data* puede recibir tanto el *path* del archivo que debe cargar como una URL. Existen validadores para los parámetros dentro de los *Dataloaders*, pero estos tienen errores y tratan de validar cosas que no se deberían. Por ejemplo, tratan de validar que la URL sea un *string* aún cuando puede que este parámetro no se ingrese, por lo que es una fuente de errores.
- Se necesita incorporar correctamente otro *dataloader* si se quiere trabajar con imágenes.
- Algunos de los métodos esperan que exista una columna de la clase. Es necesario generalizar estos métodos, ya que se espera que no todas las tareas sean de clasificación y/o pueden haber más columnas de salida para el modelo.
- No hay *tests* para corroborar que se configuran correctamente los *dataloaders*.

4.1.2. Datasets

Una vez cargados, los datos son representados en el formato *Dataset* o *DatasetDict* (un diccionario de *Datasets*) de la librería *datasets* de *HuggingFace*. Sin embargo, este formato presenta una limitación, no permite definir cuál es la información que corresponde a las columnas de entrada y salida, una distinción necesaria para el entrenamiento de un modelo. Es importante considerar que, al menos en las etapas iniciales, el *framework* se centrará exclusivamente en el aprendizaje supervisado, por lo que siempre existirá, al menos una

columna de salida. En resumen, es imprescindible que los *datasets* contengan información adicional para facilitar la correcta utilización de los modelos y tareas en etapas posteriores.

4.1.3. Tareas

- Solo existe la tarea de clasificación tabular.
- No tiene métodos que corroboren tipos de *inputs* y *outputs* u otras funcionalidades que se necesiten para las tareas.
- No hay *tests* para corroborar que se inicializan y configuran correctamente las tareas.
- Falta revisar como funciona la extensibilidad de *DashAI* al momento de agregar nuevas tareas.

4.1.4. Modelos

- Solamente existen los modelos *KNN*, *Random Forest* y *SVM*; específicamente las variantes para clasificación tabular.
- Contiene los métodos *fit*, *predict*, *save* y *load*, pero es necesario testear que cumplan sus funciones correctamente.
- Falta revisar como funciona la extensibilidad de *DashAI* al momento de agregar nuevos modelos.

4.1.5. Métricas

El componente responsable de las métricas no está implementado, lo que exige una planificación para su desarrollo. La incorporación de métricas es un factor crítico en cualquier modelo de ML, ya que proporcionan una evaluación objetiva del rendimiento del modelo y también permite comparar diferentes modelos entre sí y elegir el que mejor se adapte a las necesidades del problema.

Capítulo 5

Solución

En las siguientes secciones se presentará un análisis exhaustivo de las soluciones implementadas para abordar los desafíos anteriormente mencionados en el capítulo 4 separados a nivel de componente. Se proporcionará una exposición detallada de todas las clases y métodos añadidos y modificados, incluyendo una justificación de las decisiones tomadas en el proceso de diseño e implementación. Es importante reconocer que los problemas detallados en el Capítulo 4 se conectan de diferentes maneras con el objetivo general señalado en la subsección 1.2.1. Algunos de estos problemas tienen una relación directa con este objetivo principal (corresponderá a la sección 5.2), otros, en cambio, representan desarrollos de apoyo que, aunque no están directamente ligados al objetivo, contribuyen de manera significativa a su cumplimiento (se verán en la sección 5.1). Finalmente, existen problemas que requerían un *refactoring* del código preexistente que se verá en la sección 5.3. Por lo tanto, las soluciones a estos problemas se presentarán y discutirán en tres categorías correspondientes a estas distintas conexiones con el objetivo general.

5.1. Desarrollo de apoyo

5.1.1. Dataset

Como se comentó anteriormente, un *DatasetDict* es un diccionario de *Datasets*. Algo importante es que los *Datasets* tienen inferencia de tipos para las columnas, las cuales pueden ser las siguientes:

- *Value*: toda la columna contiene valores de un mismo tipo simple. En donde los tipos más importantes soportados son *string*, *float*, *int* y *boolean*.
- *ClassLabel*: la columna es de tipo categórica. Posee métodos especiales para mapear los *strings* a índices para que tengan representación númerica.
- *Sequence*: columna que dentro tiene otros subcampos y tipos como un diccionario. Suele ser común en tareas como NER.

- *Array2D-Array5D*: columna para arreglos multidimensionales.
- *Audio*: columna que tiene la ruta hacia el archivo de audio en formato *string* o puede ser un diccionario con el *path* hacia el archivo y tener el contenido del audio en sí (en bytes).
- *Image*: columna con el *path* hacia la imagen o puede ser un diccionario que además del *path* contenga el contenido de la imagen en sí (en bytes), en formato *array* de *Numpy* o como un objeto de tipo PIL.

Lo importante de esto es que según el *dataloader*, podrían inferirse ciertos tipos, por ejemplo, el *ImageDataloader* entrega los siguientes tipos sobre una tarea de clasificación de imágenes: *Image* y *ClassLabel*. Para otros casos puede ser posible hacer cambios de los tipos dentro de los *Datasets*. Los tipos es un concepto muy importante para luego validar que ciertos *datasets* se puedan utilizar en ciertas tareas, es por esta razón y las expuestas en la parte anterior de que se decidió seguir usando esta misma estructura de datos. Pero, estos omiten una información muy importante para *DashAI* y es la información sobre cuales son las columnas de *input* y *output* para un modelo. La solución para esto fue crear una nueva clase la cual extiende a *Dataset* de *HuggingFace* llamada *DashAIDataset*, a la cual se le agregaron los siguientes atributos:

- *inputs_columns*: es una lista que indica el nombre de las columnas que corresponden al *input*.
- *outputs_columns*: es una lista que indica el nombre de las columnas que corresponden al *output*.

Además se adaptaron algunos métodos de los *Dataset* para que devuelvan un *DashAIDataset*:

- *change_columns_type*: esta función cambia el tipo de algunas columnas del *DashAIDataset*. Recibe un diccionario cuyas claves son los nombres de las columnas a modificar y los valores son los nuevos tipos. Devuelve el *DashAIDataset* con los tipos de las columnas modificados.
- *validate_inputs_outputs*: esta función valida las columnas seleccionadas como entrada y salida para el *DashAIDataset*. Recibe tres listas: una contiene los nombres de las columnas del *dataset*, otra que contiene los nombres de las columnas a seleccionar como entradas y la última los nombres de las columnas que corresponderán a las salidas.
- *load_dataset*: esta función carga un *DatasetDict* con *DashAIDatasets* dentro. Recibe un *string* que especifica la ruta donde queda almacenado. Devuelve el *DatasetDict* de *DashAIDatasets*.
- *save_dataset*: Esta función guarda un *DatasetDict* con *DashAIDatasets* dentro. Recibe dos argumentos: un *DatasetDict* a guardar, y un *string* que especifica la ruta donde se almacenará.

De esta forma un *DatasetDict* pueda almacenar *DashAIDataset* en vez de *Dataset* y se mantiene el mismo comportamiento esperado de un *DashAIDataset* que un *Dataset*.

Esta nueva estructura de datos simplificará el trabajo de muchos de los componentes que se verán luego.

5.1.2. Métricas

Para implementar el sistema de métricas era necesario abstraer sus funciones, entradas y salidas. De este modo se tenían las siguiente consideraciones:

- Como en *DashAI* solamente se considerarán modelos del tipo aprendizaje supervisado, siempre se tendrá una columna de etiquetas.
- Considerando el primer punto, las métricas solamente reciben las etiquetas verdaderas y las etiquetas predichas.
- La separación de las métricas se da según el tipo de tarea.

En consideración a las necesidades del proyecto, el sistema de métricas debe poseer extensibilidad, dado el variado y creciente número de métricas que podrían surgir en el futuro, lo que implica la creación de una clase base (al igual que el resto de componentes extensibles) para estas. Además, dada la dependencia de las métricas con la tarea (o un conjunto de tareas), se deberá vincular las métricas a las tareas mediante el sistema de registros (el sistema de registros está explicado en la subsección 3.2.3). En vista de las entradas requeridas para las métricas, es posible que sea necesario realizar transformaciones en ciertos casos, dado que las etiquetas verdaderas serán columnas de un *DashAIDataset* y las predichas provendrán del modelo. Finalmente, es importante tener en cuenta que, aunque las métricas pueden ser configuradas, este aspecto no será tratado como un objeto configurable por el momento, ya que el enfoque principal de esta memoria no reside en este componente y no se dispone de soporte desde el frontend (esto fue mencionado en la subsección 1.3).

Se creó la clase padre llamada *BaseMetric*. Y las clases hijas *classification_metric* y *translation_metric*, en donde la primera está enlazada a las tareas de clasificación y la segunda a la traducción. Ambas cuentan con un método llamado *validate_inputs*, el cual valida que los tamaños sean iguales. Luego, al registro se agregan las métricas. Las métricas están separadas en:

Métricas de clasificación

Como primer gran conjunto se tienen las métricas de clasificación. Para estas, se utilizan funciones proporcionadas por *sklearn*, por lo tanto, dentro de la clase *classification_metric* existe un método llamado *prepare_to_metric* el cual transforma la columna de salida del *DashAIDataset* a un *array* de tipo *Numpy*. También está el método auxiliar *obtain_pred_classes*

que toma una lista de probabilidades predichas por el modelo y la transforma en un *array* de las clases predichas utilizando *numpy.argmax*.

A continuación, se muestran las métricas de este tipo:

Accuracy

Para calcular esta métrica se utiliza *accuracy_score* de la librería *sklearn*, las funciones anteriores permiten transformar las entradas a los tipos necesarios para calcular el *accuracy* correctamente.

Precision

Para calcular esta métrica se utiliza *precision_score* de la librería *sklearn*, las funciones anteriores permiten transformar las entradas a los tipos necesarios para calcular el *precision* correctamente. En caso de ser multiclase se utiliza *macro average* para su cálculo.

Recall

Para calcular esta métrica se utiliza *recall_score* de la librería *sklearn*, las funciones anteriores permiten transformar las entradas a los tipos necesarios para calcular el *recall* correctamente. En caso de ser multiclase se utiliza *macro average* para su cálculo.

F1

Para calcular esta métrica se utiliza *f1_score* de la librería *sklearn*, las funciones anteriores permiten transformar las entradas a los tipos necesarios para calcular el *F1* correctamente. En caso de ser multiclase se utiliza *macro average* para su cálculo.

Métricas de traducción

Para las métricas de traducción se utilizaron las proporcionadas por la librería *evaluate* de *Hugging Face*. En la clase *translation_metric* existe un método llamado *prepare_for_metric*, la cual transforma la columna de *output* a tipo lista.

A continuación, se muestran las métricas de este tipo:

BLEU

Para calcular esta métrica se utiliza *load* con el argumento *bleu* de la librería *evaluate*, las funciones anteriores permiten transformar las entradas a los tipos necesarios para calcular el

BLEU correctamente.

TER

Para calcular esta métrica se utiliza *load* con el argumento *ter* de la librería *evaluate*, las funciones anteriores permiten transformar las entradas a los tipos necesarios para calcular el *TER* correctamente.

Comentarios generales

La incorporación de métricas provenientes de librerías establecidas, *evaluate* de HF o *sklearn* en este caso, es una estrategia efectiva y recomendable. Estas métricas han sido sometidas a rigurosas pruebas y validaciones por la comunidad de aprendizaje automático, lo que garantiza su fiabilidad y precisión. El uso de estas herramientas existentes permite un ahorro significativo de tiempo y esfuerzo, permitiendo centrarse en otros aspectos cruciales del proyecto.

5.2. Desarrollo principal

5.2.1. Tareas

Se implementaron las siguientes tareas:

TextClassificationTask

Corresponde a la tarea de clasificación de texto. Su esquema es el siguiente:

```
inputs_types: [Value string]
outputs_types": [ClassLabel]
inputs_cardinality": 1
outputs_cardinality": 1
```

Lo que se puede entender de la siguiente forma: su tipo de *input* debe ser *Value string*, su tipo de *output* debe ser categórico y se puede tener solo una columna de *input* y una de *output*.

Su método *prepare_for_task* (el cual se explica en la subsección 5.3.2) cambia los tipos del *output* de *Value string* a tipo *ClassLabel*, es decir, categóricos.

ImageClassificationTask

Corresponde a la tarea de clasificación de imágenes. Su esquema es el siguiente:

```
inputs_types: [Image]
outputs_types": [ClassLabel]
inputs_cardinality": 1
outputs_cardinality": 1
```

Lo que se puede entender de la siguiente forma, su tipo de *input* debe ser *Image*, su tipo de *output* debe ser categórico, ya que es una tarea de clasificación. Además, se puede tener solo una columna de *input* y *output*.

También al método *prepare_for_task* se le asignó que cambie los tipos del *output* de *Value string* a *ClassLabel*.

TranslationTask

Corresponde a la tarea de traducción. Su esquema es el siguiente:

```
inputs_types: [Value string, Sequence]
outputs_types": [Value string, Sequence]
inputs_cardinality": 1
outputs_cardinality": 1
```

Lo que se puede entender de la siguiente forma: su entrada debe ser de tipo *Value string* o *Sequence*, su tipo de salida debe ser de tipo *Value string* o *Sequence* y se puede tener solamente una columna de entrada y salida.

Comentarios generales

La estructura de las tareas permite que sea muy simple agregar una nueva, de hecho basta con crear la clase, su esquema y establecer que tipos podrían llegar a cambiar (esto es poco probable además del cambio de *Value string* a *ClassLabel* que se suele hacer en tareas de clasificación). Lo mismo ocurre con el esquema, el cual es muy flexible e intuitivo sobre como cambiar/interpretar la cardinalidad o los tipos.

5.2.2. Modelos

La inclusión de nuevos modelos en *DashAI* implicó el uso de la librería *HuggingFace*. Este cambio se debió principalmente a la decisión de no entrenar los modelos desde cero, sino

aplicar el método de *fine-tuning* utilizando los datos proporcionados por el usuario. En este contexto, los objetos de tipo *DashAIDataset* se pueden utilizar directamente como entrada para estos modelos.

DistilBert

Para este modelo se utilizó la versión pre-entrenada llamada *distilbert-base-uncased* cargada a través del objeto *DistilBertForSequenceClassification.from_pretrained*. Como es un modelo especializado en hacer clasificación de texto es necesario utilizar un tokenizador, por lo que se usó el que provee este mismo modelo, es decir, *DistilBertTokenizer.from_pretrained*. Sus métodos funcionan de la siguiente forma:

- *get_tokenizer*: tokeniza los textos, de manera que esten listos para usarlos en el modelo.
- *fit*: recibe un *DashAIDataset* y utiliza la función anterior para tokenizar las columnas. Luego utilizando el objeto *Trainer* de la librería *transformers* de HF es posible hacer *fine-tuning* del modelo *DistilBert*. Es importante mencionar que dentro del *Trainer* se entregan los argumentos que ingresa el usuario para entrenarlo.
- *predict*: recibe un *DashAIDataset*, utiliza la función *get_tokenizer* para tokenizar el *input* y luego se le entrega al modelo para que retorne el vector de probabilidades por clase (para modelos de clasificación se acordó este comportamiento según lo comentado en la subsección 5.3.3).
- *save*: recibe un *string* que corresponde al nombre del archivo y se guarda utilizando el método *save_pretrained* de HF.
- *load*: recibe un *string* que corresponde al nombre del archivo que contiene al modelo guardado y se lee con *DistilBertForSequenceClassification.from_pretrained*.

Sus parámetros configurables son los siguientes:

- *epochs*: que indica la cantidad de épocas de entrenamiento.
- *batch_size*: el tamaño de los *batches* para entrenar los modelos.
- *learning_rate*: el valor inicial del *learning rate* para el optimizador.
- *device*: el *hardware* que se utilizará para entrenar el modelo el cual puede ser: *gpu* o *cpu*.
- *weight_decay*: La decadencia de peso que se debe aplicar (si no es cero) a todas las capas excepto a todos los pesos de sesgo y normalización de capas en el optimizador.

ViT

Para este modelo se utilizó la version pre-entrenada llamada *google/vit-base-patch16-224* cargada a través del objeto *ViTForImageClassification.from_pretrained*. Como es un modelo especializado en hacer clasificación de imágenes es necesario utilizar un *feature extractor* para las imágenes, por lo que se usó el que provee este mismo modelo, es decir, *ViTFeatureExtractor.from_pretrained*. Sus métodos funcionan de la siguiente forma:

- *preprocess_images*: recibe y devuelve las imágenes ya procesadas por el *feature extractor* para ser usadas en el modelo.
- *fit*: recibe un *DashAIDataset*, utiliza la función anterior para obtener las *features* de las imágenes. Luego utilizando un *Trainer* es posible hacer *fine-tuning* del modelo, es importante mencionar que dentro de este se entregan los argumentos que ingresa el usuario para entrenarlo.
- *predict*: recibe un *DashAIDataset*, utiliza la función *preprocess_images* para procesar el *input* y luego se le entrega al modelo para que retorne el vector de probabilidades por clase
- *save*: recibe un *string* que corresponde al nombre del archivo y se guarda utilizando el método *save_pretrained* de HF.
- *load*: recibe un *string* que corresponde al nombre del archivo que contiene al modelo guardado y se lee con *ViTForImageClassification.from_pretrained*.

Sus parámetros configurables son los siguientes:

- *epochs*.
- *batch_size*.
- *learning_rate*.
- *device*.
- *weight_decay*.

Tatoeba-Challenge En-Es

Para este modelo se utilizó la version pre-entrenada llamada *Helsinki-NLP/opus-mt-en-es* cargada a través del objeto *AutoModelForSeq2SeqLM.from_pretrained*. Como es un modelo que trabaja con texto necesita tener un tokenizador, por lo que se usó el que provee este mismo modelo, es decir, *AutoTokenizer.from_pretrained*. Sus métodos funcionan de la siguiente forma:

- *fit*: recibe un *DashAIDataset*, dentro de esta misma tokeniza los textos en inglés y español. Luego, utilizando el objeto *Seq2SeqTrainer* es posible hacer *fine-tuning* del modelo, es importante mencionar que dentro de esta se entregan los argumentos que ingresa el usuario para entrenar.
- *predict*: recibe un *DashAIDataset*, dentro de esta tokeniza la entrada y luego se le entrega al modelo para que entregue los textos traducidos.
- *save*: recibe un *string* que corresponde al nombre del archivo y se guarda utilizando el método *save_pretrained* de HF.
- *load*: recibe un *string* que corresponde al nombre del archivo que contiene al modelo guardado y se lee con *AutoModelForSeq2SeqLM.from_pretrained*.

Sus parámetros configurables son los siguientes:

- *epochs*.
- *batch_size*.
- *learning_rate*.
- *device*.
- *weight_decay*.

Comentarios generales

Es crucial destacar que los modelos previos, al ser sometidos a *fine-tuning*, generan diversos *checkpoints* a medida que transcurren las épocas de entrenamiento. No obstante, la versión de *DashAI* tratada en este documento no requiere el soporte para el uso de estados intermedios del entrenamiento, como por ejemplo para la implementación del *early stopping*. Por lo tanto, sólo se preserva el estado del modelo después de que se han ejecutado la cantidad de épocas especificada por el usuario. Lo anterior, es una limitación de la implementación que no se mencionó en la subsección 1.3 debido a requería más contextualización.

5.3. Refactoring

5.3.1. Dataloaders

Los *dataloaders* al ser un componente extensible quedaron con una clase base abstracta llamada *BaseDataLoader* con los siguientes métodos:

- *get_schema*: método para obtener el esquema considerando que es un objeto configurable (recordar sección 3.2.1). Es decir, toma en cuenta los parámetros ingresados por el usuario para configurar la carga de los datos.

- *load_data*: método abstracto para cargar los datos. Cuando se haga el desglose de los tipos de *dataloaders* se comentará con más detalle lo que hace.
- *extract_files*: método para leer los archivos subidos por el usuario y descomprimirlos si es necesario. Retorna una ruta que es donde se guardarán los archivos ya extraídos.
- *split_dataset*: este método tiene como objetivo dividir el conjunto de datos en conjuntos separados para el entrenamiento, validación y pruebas. Es posible establecer diversos parámetros como:
  - *seed*, que permite establecer una semilla para garantizar la reproducibilidad de las divisiones.
  - *shuffle*, que habilita la selección aleatoria de filas durante la división.
  - *stratify*, contribuye a mantener la proporción original de las clases en los conjuntos de entrenamiento, validación y prueba.
  - El nombre de la columna que contiene las clases.

  Este método emplea la función *train_test_split* de la librería *sklearn* para dividir el conjunto de datos y toma en cuenta los argumentos anteriores si estos se seleccionan. Es importante destacar que lo que devuelve es un *DatasetDict* cuyos *Datasets* son de tipo *DashAIDataset* (recordar subsección 5.1.1). Los diversos argumentos mencionados anteriormente derivan de los *JSON SCHEMAS* proporcionados por el *frontend*.
- *to_dashai_dataset*: método para convertir a tipo *DashAIDataset* los *Dataset* que conforman un *DatasetDict*.

Las siguientes clases son las que se extienden desde *BaseDataloader* y dependen del formato en que vienen los datos:

CSVDataloader

Esta clase es la que se utiliza cuando los datos del usuario vienen en formato CSV. Dentro de está el método *load_data* que recibe los siguientes argumentos:

- *dataset_path*: es de tipo *string* y define el *path* en donde se guardará el *dataset*, recordando que se almacenan en tablas de tipo *Arrow*. Esta ruta es la que se guardará en la base de datos de *DashAI*.
- *params*: es un diccionario que contiene los parámetros configurables que definió el usuario, para este caso solo es el tipo de separador que tiene el CSV.
- *file*: puede ser el archivo CSV en sí mismo o un ZIP que dentro contenga varios CSV. Puede ser vacío en caso de que se ingrese una URL, aunque por ahora esto no esté permitido a nivel de *frontend*.
- URL: URL que indica donde está el archivo CSV en caso de que no se quiera subir el archivo directamente. Puede ser vacío en caso de que se suba el archivo.

Luego, en primera instancia se hacen las respectivas validaciones, la cuales son:

- La primera validación comprueba que tanto *file* como URL no sean nulos al mismo tiempo. Esto garantiza que al menos uno de los dos debe ser proporcionado para proceder.
- La segunda validación se asegura de que *file* y URL no sean ambos proporcionados al mismo tiempo. Esto se realiza para evitar conflictos o ambigüedades acerca de qué fuente de datos utilizar.
- La tercera validación verifica que *dataset_path* sea un *string*.
- La cuarta validación asegura que *params* sea un diccionario. Esto es vital porque *params* debería contener pares clave-valor que representan los parámetros de configuración para la carga de datos entregados por el usuario.
- La quinta validación comprueba que el parámetro *separator* esté presente en el diccionario *params*. Este parámetro es necesario para indicar cómo se deben separar los campos en el archivo CSV.
- La sexta validación se asegura de que el *separator* proporcionado sea un *string*.
- La séptima validación verifica que *file* sea un archivo subido por el usuario o un valor nulo. Esto asegura que si se proporciona un archivo, sea del tipo correcto.
- La última validación se asegura de que URL sea un *string* que represente una dirección web o un valor nulo. Esto garantiza que si se proporciona una URL, sea del formato correcto.

Después, se utiliza el método *load_dataset* de la librería *datasets* de *HuggingFace* que permite cargar un *dataset* considerando el formato en que vienen los datos. Según el caso se tienen las siguientes alternativas:

1. Si proviene de una URL basta con:

   ```
   load_dataset("csv", data_files=url, sep=separator)
   ```

2. Si proviene de un archivo primero se extrae con el método *extract_files* el cual lo deja en la ruta *files_path* y luego se lee con:

   ```
   load_dataset("csv", data_files=files_path, sep=separator)
   ```

3. Si proviene de un archivo ZIP primero se extraen los archivos con el método *extract_files* el cual lo deja en la ruta *files_path* y luego se lee con:

   ```
   load_dataset("csv", data_dir=files_path, sep=separator)
   ```

Finalmente, *load_dataset* retorna un *DatasetDict*, es decir, un diccionario en donde la llave es el nombre de la separación y el valor es un *Dataset*.

JSONDataloader

Es análogo al *CSVDataloader*, recibe los mismos parámetros y hace validaciones del mismo tipo. La diferencia fundamental es que utiliza:

```
load_dataset("json", data_dir=files_path, field=field)
```

Es decir, indicar que es de tipo JSON y en que campo está la información de la data. Al igual que en el *dataloader* anterior, tiene 3 variaciones según desde donde se leen los datos.

ImageDataloader

Es análogo a los dos *dataloaders* anteriores, la principal diferencia radica en que en este caso los archivos deben venir comprimido en formato ZIP y se utiliza:

```
load_dataset("imagefolder", data_dir=files_path)
```

Comentarios generales

La característica principal de esta implementación radica en que todos los conjuntos de datos se simplifican a tablas, independientemente del tipo de entrada. Por ejemplo, en la clasificación de imágenes, se dispone de una columna que indica la ruta hacia la imagen y otra que especifica la etiqueta correspondiente a la imagen; en el caso de la traducción, una columna contiene el texto en el idioma original mientras que la otra alberga el texto en el idioma *target*. Esta disposición favorece una selección sencilla de las columnas de entrada y salida para el modelo. Y este enfoque se ve reforzado por la capacidad del *software* para trabajar exclusivamente con modelos de aprendizaje supervisado.

Otro aspecto, es que se decidió seguir utilizando los *dataloaders* de *HuggingFace* debido a su flexibilidad y simplicidad para poder recibir distintos tipos de formatos; además su forma de manejar los datos en tablas *Arrow* tiene algunas ventajas como:

- Permite cargar y procesar datos de manera muy eficiente, ya que mantiene los datos en formato de columnas en lugar de filas. Esto hace que operaciones como la filtración y la agrupación de datos sean mucho más rápidas.
- Almacena los datos en la memoria de manera compacta, lo que permite trabajar con conjuntos de datos grandes que no caben en la memoria RAM. Esto significa que se puede trabajar con conjuntos de datos de cualquier tamaño.

Finalmente, los *dataloaders* de HF contiene la simple y poderosa función *load_dataset* que únicamente requiere un argumento apropiado según el tipo de formato. De hecho, hay otros tipos disponibles como TXT, Parquet, Arrow, SQL y audio. También, esta estructura de clases se mantuvo para garantizar la extensibilidad, ya que permite agregar nuevos *dataloaders* de una manera bastante simple.

5.3.2. Tareas

Al igual que los *dataloaders*, al ser un componente extensible, contiene una clase llamada *BaseTask*. A las tareas se le agregó un nuevo atributo de tipo diccionario llamado *schema*, lo que hace es determinar cuales son los tipos válidos para la tarea (los tipos mencionados en la subsección 5.1.1) y su cardinalidad, es decir, si puede recibir o entregar múltiples *inputs* o *outputs* respectivamente. Considerando esto se hicieron los siguientes métodos:

- *validate_dataset_for_task*: recibe un *DatasetDict* con uno o más *DashAIDataset* dentro y busca que tenga los tipos y cardinalidad adecuados para la tarea. En caso de no cumplir entrega un excepción de tipo o de valor dependiendo en que falló.
- *prepare_for_task*: recibe un *DatasetDict* y hace cambios a los tipos de ciertas columnas de forma automática. Este método es posible debido a que en primera instancia del flujo se escoge la tarea, por lo tanto, con esto ya se pueden hacer algunos cambios para poder validarse correctamente. Esto último se da principalmente porque la inferencia de tipos se hace sobre los tipos más generales posibles, por ejemplo, si se tiene un CSV donde una columna es de tipo *string*, se asignará el tipo *Value string*, pero puede que esta sea una columna categórica, lo que en ese caso el tipo más adecuado sería *ClassLabel*, es por eso que este método es importante y sirve para hacer estos cambios sutiles que son necesarios, este sería uno de los pocos casos donde se debe aplicar este cambio, porque para el resto de casos los tipos ya vienen correctamente asignados.

La única tarea que ya estaba implementada, pero le faltaban métodos y atributos era la siguiente:

TabularClassificationTask

Corresponde a la tarea de clasificación tabular. Se definió el esquema (recordar los tipos de la sección 5.1.1) de esta tarea como el siguiente:

```
inputs_types: [ClassLabel, Value float, Value int, Value bool]
outputs_types": [ClassLabel]
inputs_cardinality": "n"
outputs_cardinality": 1
```

Lo que se puede entender de la siguiente forma: sus tipos de entrada pueden ser categóricos o de tipo *Value*, su tipo de salida debe ser categórico, ya que es una tarea de clasificación, se puede tener una cantidad variable de entradas y solamente una salida.

Al método *prepare_for_task* se le asignó que cambie los tipos *Value string*, a tipo *ClassLabel*, es decir, categóricos.

5.3.3. Modelos

Al ser un componente extensible se tiene la clase *BaseModel*, en esta se define que todos los modelos debían tener los siguientes métodos: *fit*, *predict*, *save*, *load* y *get_schema*. El último método es debido a que los modelos son objetos configurables (la definición de estos está en la subsección 3.2.1). Para mantener información sobre cuales modelos son compatibles con cada tarea se tiene una clase hija de *BaseModel* asociada a las distintas tareas, de esta forma cada modelo de cierto tipo debe tener como clase padre a la que enlaza la tarea a los modelos. Por ejemplo, los modelos asociados a la tarea de clasificación tabular tienen una clase llamada *TabularClassificationModel* y luego los modelos en sí mismos son clases hijas de esta. Por último, es importante mencionar que los modelos de por sí tienen la responsabilidad de formatear los datos (si corresponde) desde un *DashAIDataset* al tipo que le parezca más conveniente, esto es debido a que el *framework* no tiene la capacidad de hacer transformaciones elaboradas a los datos aunque se espera que se pueda agregar un módulo de este tipo en próximas iteraciones.

A continuación. se enumerarán algunos de los modelos que ya estaban en *DashAI* y como quedaron finalmente.

KNN

Para su implementación se utiliza el objeto *KNeighborsClassifier* de la librería *sklearn*. Los métodos son los siguientes:

- *format_data*: recibe un *DashAIDataset* y lo transforma a un *DataFrame* de *Pandas*, retorna una tupla *(x,y)* en donde x corresponde a la entrada e y corresponde a la salida.
- *fit*: recibe un *DashAIDataset*, utiliza la función anterior para dejarlo como la tupla *(x,y)*, luego se llama al método *fit* de la clase *KNeighborsClassifier* con la tupla anterior.
- *predict*: recibe un *DashAIDataset*, utiliza la función anterior para dejarlo como la tupla *(x,y)*, luego se llama al método *predict_proba* (en la subsección 5.3.3 se explicará esta decisión) de la clase *KNeighborsClassifier* con la tupla anterior.
- *save*: recibe un *string* que corresponde al nombre del archivo y se guarda con *dump* de la librería *joblib*.
- *load*: recibe un *string* que corresponde al nombre del archivo que contiene al modelo guardado y se lee con *load* de la librería *joblib*.

Sus parámetros configurables son los siguientes:

- *n_neighbors*: define el número de vecinos a considerar.
- *weights*: determina si todos los vecinos pesan igual (*uniform*) o los más cercanos pesan más (*distance*).

- *algorithm*: selecciona el algoritmo para calcular los vecinos más cercanos: *auto* (selecciona automáticamente), *ball_tree*, *kd_tree* (ambos aceleran las búsquedas) o *brute* (fuerza bruta).

Random Forest Classifier

Para su implementación se utiliza el objeto *RandomForestClassifier* de la librería *sklearn*. Los métodos son los siguientes:

- *format_data*: recibe un *DashAIDataset* y lo transforma a un *DataFrame* de *Pandas*, retorna una tupla *(x,y)* en donde x corresponde a la entrada e y corresponde a la salida.
- *fit*: recibe un *DashAIDataset*, utiliza la función anterior para dejarlo como la tupla *(x,y)*, luego se llama el método *fit* de la clase *RandomForestClassifier* con la tupla anterior.
- *predict*: recibe un *DashAIDataset*, utiliza la función anterior para dejarlo como la tupla *(x,y)*, luego se llama al método *predict_proba* de la clase *RandomForestClassifier* con la tupla anterior.
- *save*: recibe un *string* que corresponde al nombre del archivo y se guarda con *dump* de la librería *joblib*.
- *load*: recibe un *string* que corresponde al nombre del archivo que contiene al modelo guardado y se lee con *load* de la librería *joblib*.

Sus parámetros configurables son los siguientes:

- *n_estimators*: el número de árboles en el bosque.
- *max_depth*: la profundidad máxima de los árboles.
- *min_samples_split*: el número mínimo de muestras requerido para dividir un nodo.
- *min_samples_leaf*: el número mínimo de muestras requerido para ser un nodo hoja.
- *max_leaf_nodes*: número máximo de nodos hoja.
- *random_state*: semilla utilizada por el generador de números aleatorios.

SVM

Para su implementación se utiliza el objeto *SVC* de la librería *sklearn*. A diferencia de los anteriores, se debe inicializar con el argumento *probability* como *True* para utilizar el método *predict_proba* como en los anteriores. Los métodos son los siguientes:

- *format_data*: recibe un *DashAIDataset* y lo transforma a un *DataFrame* de *Pandas*, retorna una tupla *(x,y)* en donde x corresponde a la entrada e y corresponde a la salida.
- *fit*: recibe un *DashAIDataset*, utiliza la función anterior para dejarlo como la tupla *(x,y)*, luego se llama el método *fit* de la clase *SVC* con la tupla anterior.
- *predict*: recibe un *DashAIDataset*, utiliza la función anterior para dejarlo como la tupla *(x,y)*, luego se llama al método *predict_proba* de la clase *SVC* con la tupla anterior.
- *save*: recibe un *string* que corresponde al nombre del archivo y se guarda con *dump* de la librería *joblib*.
- *load*: recibe un *string* que corresponde al nombre del archivo que contiene al modelo guardado y se lee con *load* de la librería *joblib*.

Sus parámetros configurables son los siguientes:

- C: parámetro de regularización, también conocido como parámetro de penalización de errores.
- *coef0*: término independiente en la función kernel.
- *degree*: grado de la función polinomial del kernel ("poly").
- *gamma*: coeficiente para los "rbf", "poly" y "sigmoid" kernels.
- *kernel*: especifica el tipo de kernel a utilizar.
- *max_iter*: máximo número de iteraciones para el solucionador.
- *shrinking*: uso de la heurística de encogimiento.
- *tol*: tolerancia para el criterio de detención.

Comentarios generales

Para el caso de modelos de *sklearn* no era posible entregarles directamente *DashAIDatasets*, por lo tanto, en todos los casos fue necesario transformarlas previamente a *Dataframes* de *Pandas* para su utilización, ya que era la forma más simple y directa de utilizarlas. El método *predict_proba* se utiliza, ya que como algunas métricas de clasificación utilizan los *labels* predichos y otras las probabilidades de pertenecer a cada clase, se decidió por dejar el vector de las probabilidades, ya que era un resultado más general.

5.4. Caso de uso

Ya explicados los componentes anteriores la figura 5.1 contiene un ejemplo de como es la secuencia que siguen los componentes anteriores:

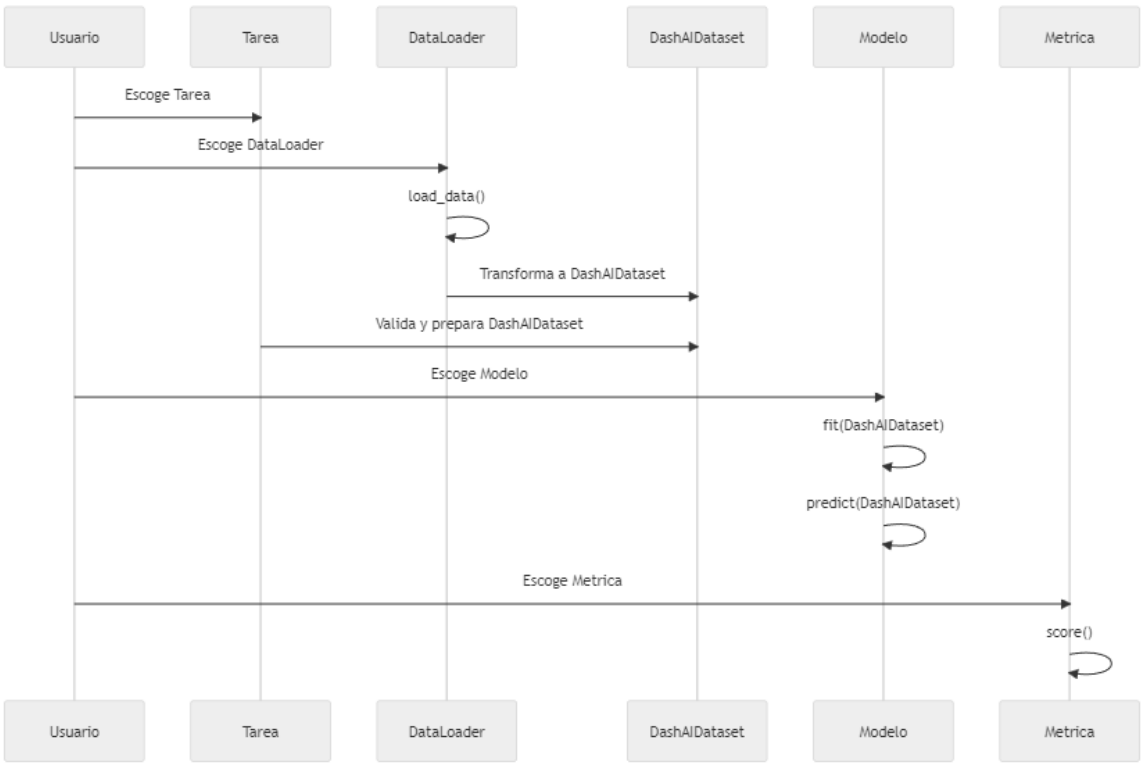

Figura 5.1: Diagrama de secuencia de uso de los componentes

Capítulo 6

Evaluación

6.1. Testing de componentes

Para probar los diversos componentes y considerando el flujo que estos van siguiendo, se hizo *testing* de forma incremental con el siguiente orden:

6.1.1. Dataloaders

Para el caso de los distintos *dataloaders* se subieron archivos de prueba en formato CSV, JSON y ZIP. Se simulaba la carga de datos tal como lo recibiría la API con la librería *starlette* la cual es la que maneja la estructura de datos en la que quedan los archivos luego de utilizar *FastAPI*.

Con lo anterior, y simulando los parámetros con un diccionario, ya era posible crear el *Dataloader* según el formato y cargar los datos con el método *load_data*. De estas se hicieron las siguientes pruebas para los tres tipos de *dataloaders*, es decir, CSV, JSON e Image:

- Que la creación efectivamente entregue un *DatasetDict* como resultado.
- Que si no se entregan los parámetros obligatorios se lance un excepción de que falta cierto parámetro.
- Que lance una excepción en caso de no encontrar el archivo en la ruta que se le entregó.
- Que lance una excepción en caso de entregarle un archivo de la cual no se puede generar un *dataset* correctamente, ya que no calza la cantidad de columnas, por ejemplo.

6.1.2. DashAIDataset

Ya teniendo creados satisfactoriamente los *DatasetDict* desde los datos cargados, era momento de testear la correcta creación de los *DashAIDataset*. Para ello se tenían las siguientes

pruebas:

- Primero se crea manualmente una lista de las columnas que serían *input* y *output*, y luego se transforma un *Dataset* a uno de tipo *DashAIDataset*. De este último se comprueba que la información en los atributos *inputs_columns* y *outputs_columns* se guarde correctamente.
- Como al momento de crearse los *DashAIDataset* se verifican las columnas que serán de entrada y salida se testeó que efectivamente se lance un error cuando se entregan más elementos como *input* y *output* que la cantidad de columnas que tiene el *dataset*.
- En la misma línea que lo anterior, se testeó que se lance un error cuando se entrega un nombre de *I/O* que no corresponda con alguna columna.
- Acorde con los últimos dos tests, se lanza un error cuando se dejan columnas sin clasificar como *I/O*.
- Se testeó que se lance un error cuando se trate de cambiar el tipo, con el método *change_columns_type*, de una columna que no existe.
- Se testeó que se lancé un error cuando se trate de cambiar el tipo, con el método *change_columns_type*, de una columna a un tipo que no se puede.
- Se testeó que al utilizarse el método *change_columns_type* se mantengan los atributos de *DashAIDataset*.
- Se testeó que al separarse en conjuntos de entrenamiento, validación y testing; se sigan teniendo los atributos de los *DashAIDatasets*, además de que en el proceso no falten o se dupliquen filas.
- También se testeó que al guardarse y posteriormente leerse un *DashAIDataset* se mantengan sus mismos atributos.

6.1.3. Tareas

En esta fase, con los *dataloaders* y *DashAIDataset* ya comprobados, se procedió a leer los datos del archivo de prueba mediante un *dataloader* para luego transformarlos a un *DashAIDataset*. Este procedimiento permitió realizar las siguientes pruebas:

- Se verificó el funcionamiento del validador de tareas sobre un *DashAIDataset*, asegurándose tanto de que acepte los *DashAIDataset* correctos, como de que emita un error en caso de recibir un *DashAIDataset* con tipos incorrectos.
- Se probó que el método *prepare_for_task* haga los cambios efectivos de los tipos, y que esto permita validarlos correctamente si corresponde.

Estos tests se hicieron para todas las tareas.

6.1.4. Modelos

Para evaluar los modelos, se emplearon pequeños conjuntos de datos de prueba. Los conjuntos de prueba utilizados fueron los siguientes:

- *Iris Dataset* para modelos de clasificación tabular, es decir, *SVM*, *Random forest* y *KNN*. El *dataset* contiene información acerca de tres especies de Iris (Iris setosa, Iris virginica e Iris versicolor). Las características que se registran son la longitud y el ancho del sépalo y el pétalo.
- *IMDB sentiment Dataset* para modelos de clasificación de texto, es decir, *DistilBert*. Este conjunto de datos está compuesto por reseñas de películas de la base de datos de *Internet Movie Database* (IMDB). Cada reseña está asociada a una etiqueta de sentimiento, es decir, positivo o negativo.
- *Beans Dataset* para modelos de clasificación de imágenes, es decir, *ViT*. Este es un conjunto de datos de imágenes que contiene hojas de plantas de frijoles y se utiliza para identificar enfermedades comunes en estas plantas.
- *Translation EngSpa Dataset* para modelos de traducción, es decir, *opus-mt-en-es*. Este *dataset* contiene pares de oraciones en inglés y español.

Luego, se hicieron los siguientes tests para los modelos:

- Para verificar el entrenamiento de los modelos se implementó un enfoque específico según la librería. Para los modelos de *sklearn*, se empleó el método *check_is_fitted*. En el caso de *Hugging Face*, se recurrió a un atributo de la clase que imita esta funcionalidad después de aplicar la función *fit* del modelo.
- Se validó que el método *predict* de los modelos retorne resultados con una longitud que coincida con la del conjunto a predecir.
- Se aseguró que los modelos que no han pasado por la función *fit* no sean capaces de predecir.
- Se verificó la capacidad de los modelos para ser guardados y cargados correctamente validando la posibilidad de almacenarlos después de entrenarlos y cargarlos posteriormente para realizar predicciones.
- Se testeó que los modelos tengan su esquema correspondiente para permitir su entrenamiento con hiperparámetros distintos a los predefinidos.

6.1.5. Métricas

Al igual que para los modelos se hicieron *datasets* de prueba para entrenar modelos de clasificación y traducción para luego poder aplicar las métricas correspondientes. Se hicieron los siguientes tests:

- Se probó que cada una de las métricas sean utilizables y que entregan valores acordes a lo que se espera.
- Se probó que las métricas entregue un error si es que las etiquetas reales y predichas no calzan en longitud.

6.2. Reproducibilidad de modelos del estado del arte

Habiendo probado y preparado todos los componentes, la estrategia más efectiva para evaluarlos en el contexto de *DashAI* es simular la interacción típica de un usuario siguiendo un flujo normal de trabajo. Cabe mencionar que existen múltiples enfoques de prueba relacionados con la interacción del usuario, sin embargo, estos van más allá del alcance de esta memoria.

En lo que sigue, ilustraremos cómo llevar a cabo el *fine-tuning* de un modelo del estado del arte, como *DistilBert*, en DashAI con los datos del *IMDB sentiment Dataset* y obtener sus métricas correspondientes:

En la figura 6.1, se muestra la interfaz principal de *DashAI* que presenta tres pestañas disponibles para el usuario.

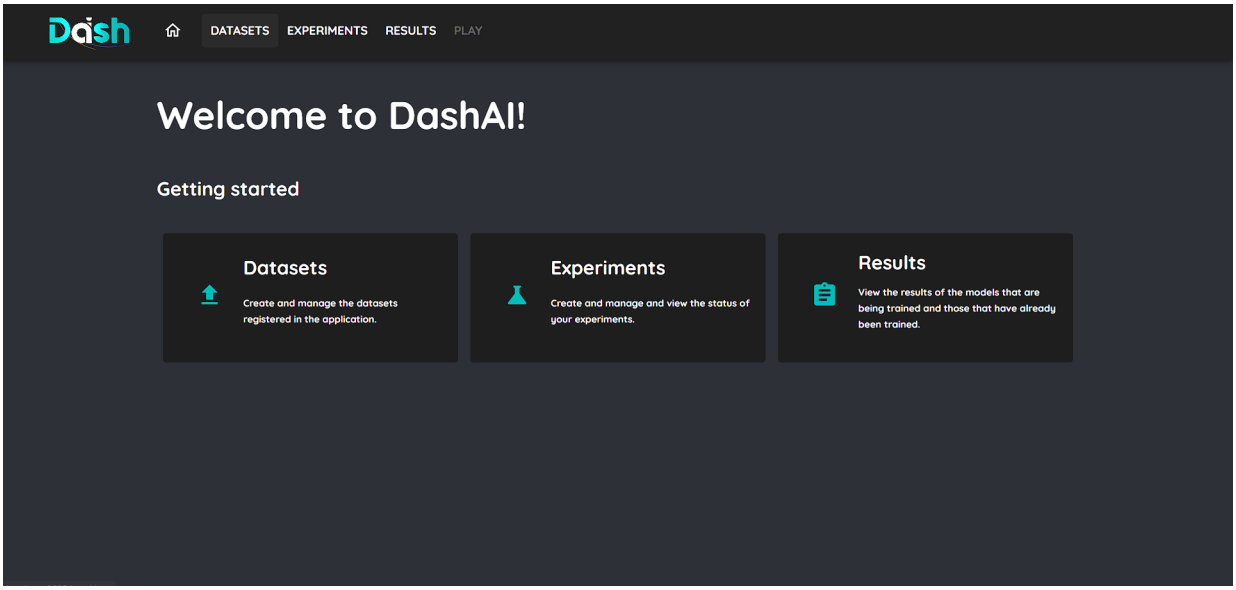

Figura 6.1: Interfaz principal de *DashAI*.

Al seleccionar la primera opción para cargar un *dataset*, se presenta la vista mostrada en la figura 6.2.

Como se puede apreciar en la figura 6.3, es imprescindible elegir primero la tarea correspondiente. Este ejemplo utiliza un modelo *DistilBert* y, por tanto, la tarea seleccionada es la clasificación de texto.

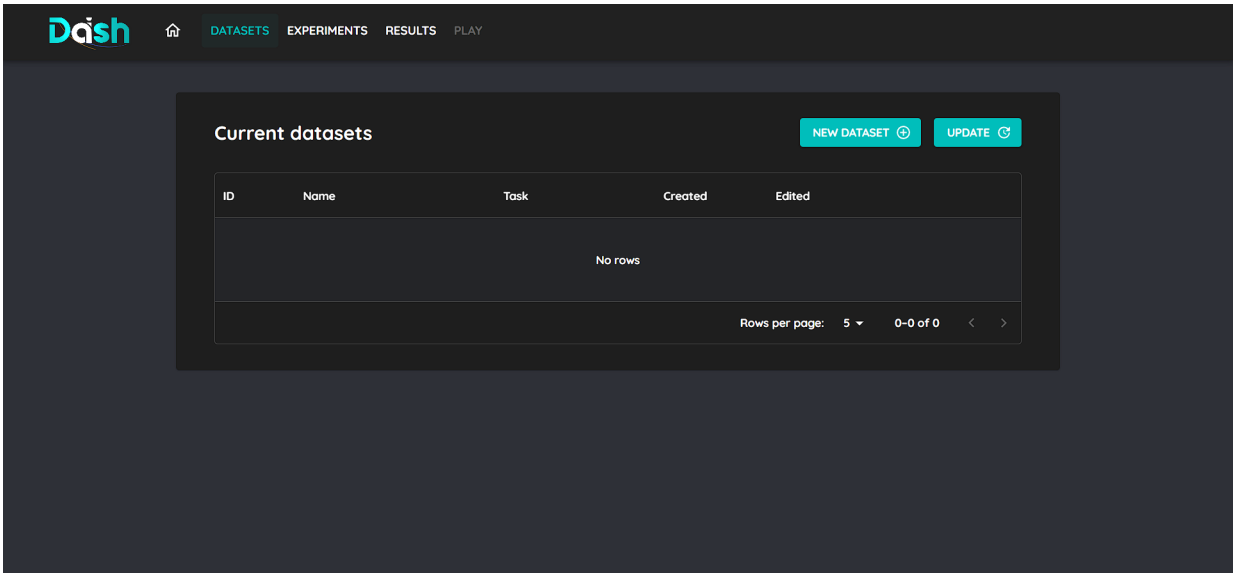

Figura 6.2: Interfaz de carga de *datasets*.

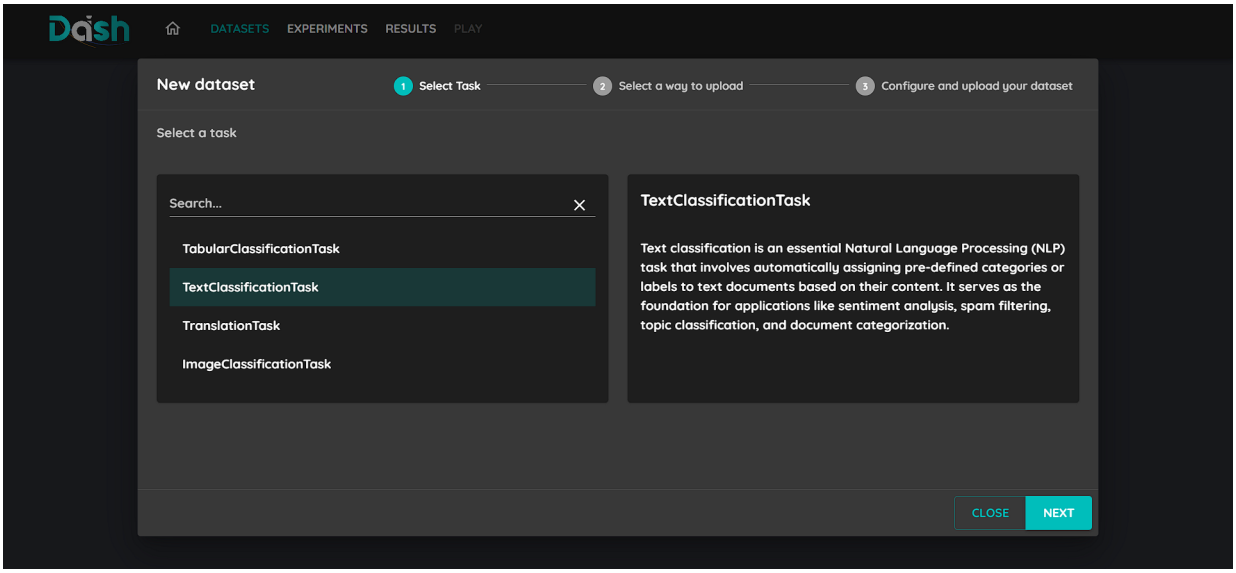

Figura 6.3: Selección de tarea para los *datasets*.

Posteriormente, como se muestra en la figura 6.4, se deben seleccionar los *dataloaders* que sean compatibles con la tarea elegida. Para este caso, se utiliza el *JSONDataloader*.

Tras seleccionar el *dataloader*, se procede a configurarlo. En la figura 6.5 se puede apreciar este proceso. En este ejemplo, se asigna un nombre, se determina el campo que contiene los datos y se especifica la columna de salida. Más abajo (no visible en la imagen), hay configuraciones adicionales relacionadas con la separación de los conjuntos (en la figura 3.4 se puede ver esta parte).

Una vez cargado y configurado el *dataset*, es necesario pasar a la pestaña *Experiments* para crear el experimento. Como se ilustra en la figura 6.6, el primer paso consiste en seleccionar la tarea y el nombre del experimento.

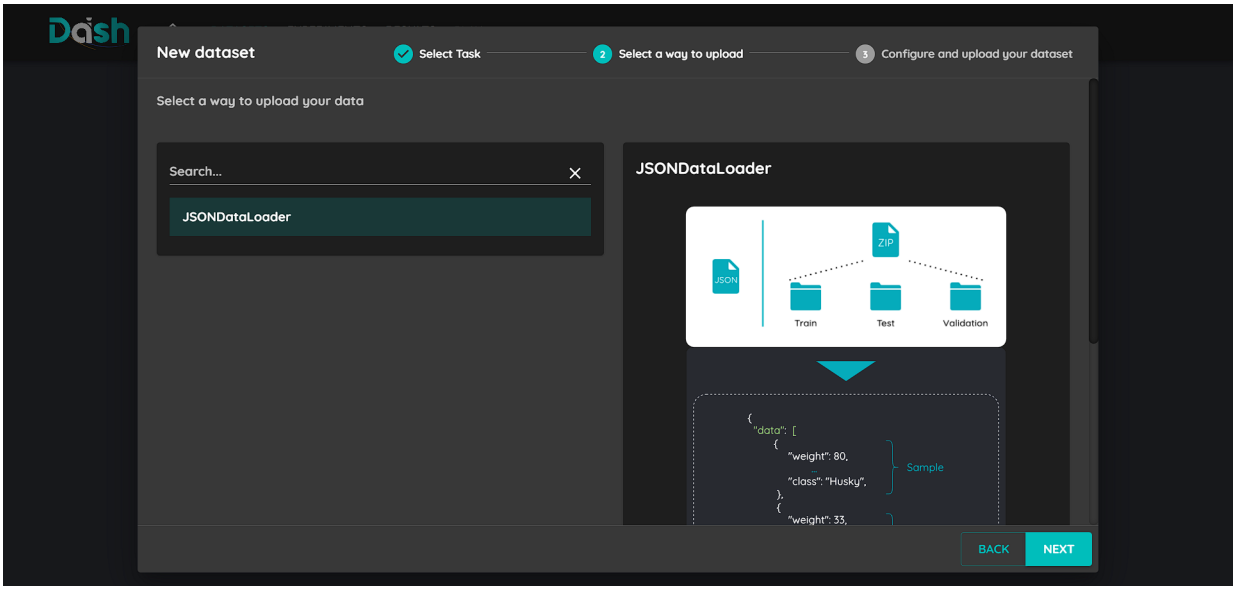

Figura 6.4: Selección de *dataloader* para los *datasets*.

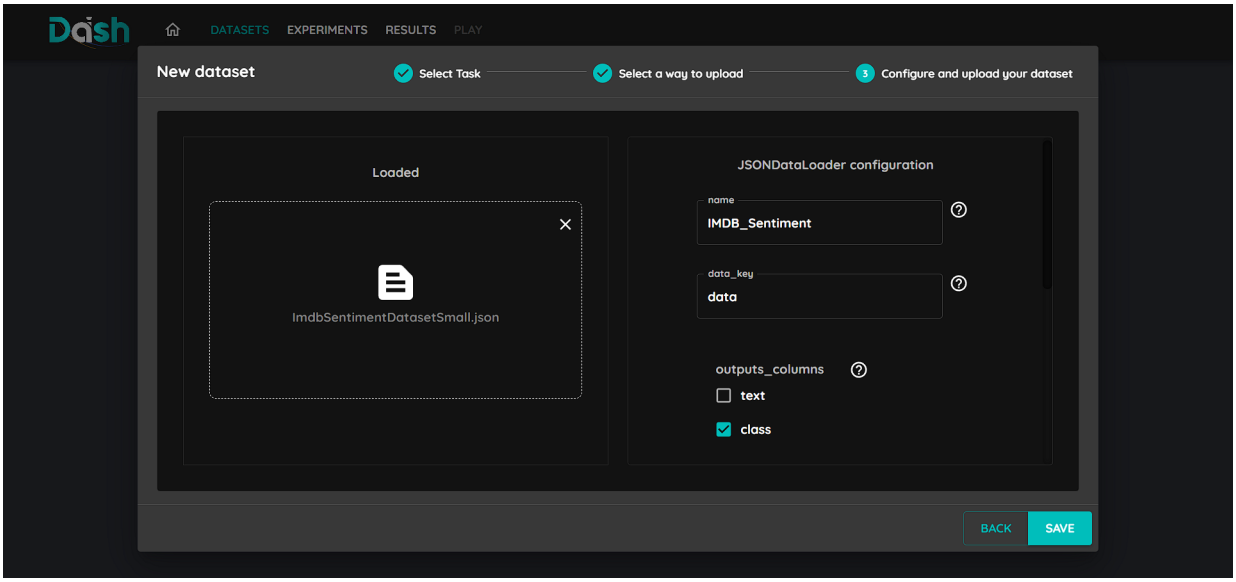

Figura 6.5: Configuración del *dataloader* para los *datasets*.

En la siguiente interfaz (figura 6.7), se muestran los *datasets* disponibles en función de la tarea elegida. Es posible verificar que el *dataset* cargado previamente está disponible.

Bajo la opción *Select a model to add*, se pueden elegir los modelos que sean compatibles con la tarea seleccionada. En este caso, se selecciona el modelo *DistilBert*. Tras añadirlo, el modelo aparece listo para ser configurado, como se muestra en la figura 6.8.

Si se presiona el ícono del engranaje en la figura 6.8, se accede a las distintas opciones de configuración del modelo para su entrenamiento, como se muestra en la figura 6.9.

Después de configurar el modelo, se pueden seleccionar todos los experimentos para ser ejecutados, tal y como se muestra en la figura 6.10.

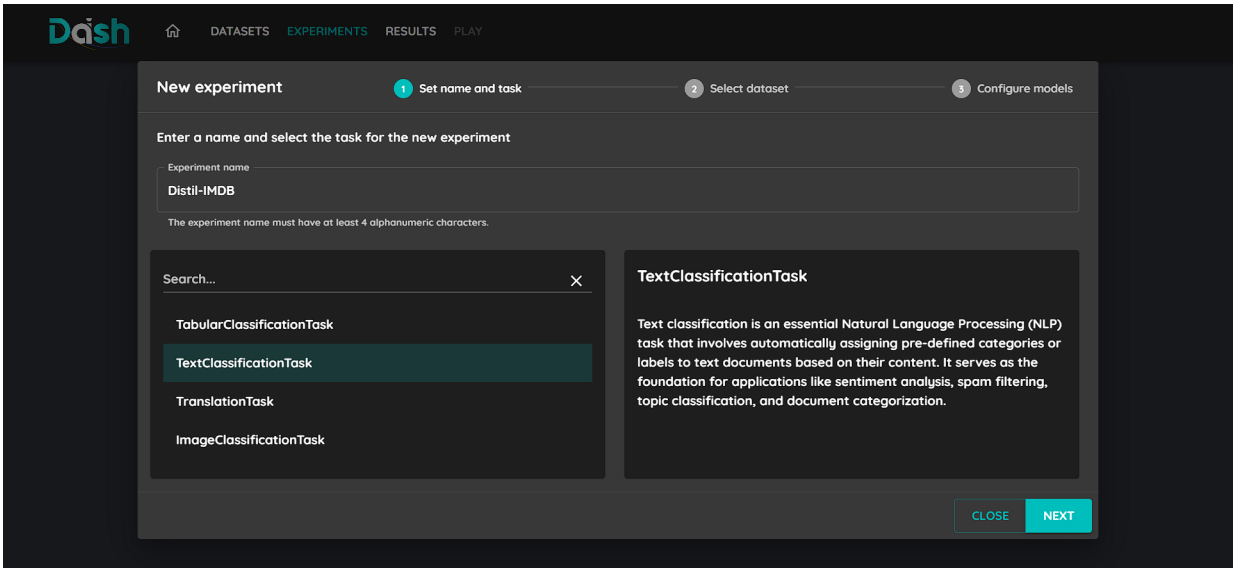

Figura 6.6: Selección de la tarea y nombre del experimento.

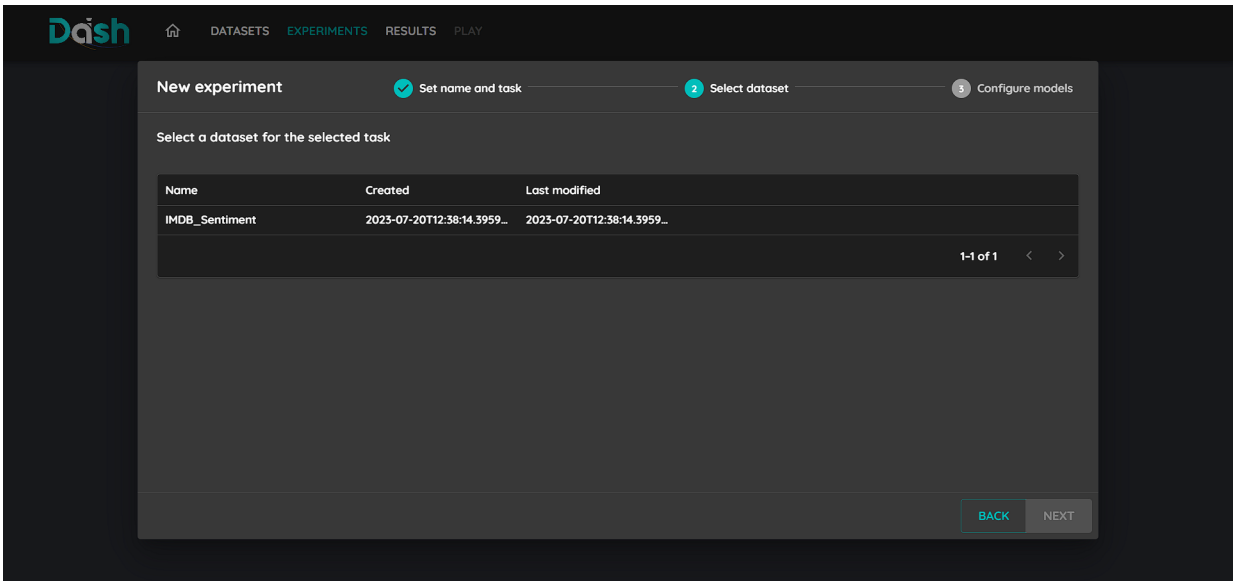

Figura 6.7: Selección del *dataset* para el experimento.

Una vez terminados los experimentos, se debe ir a la pestaña *Results* para revisar los resultados. Esta sección se puede apreciar en la figura 6.11.

Finalmente, tras seleccionar el experimento del cual se desean ver los resultados, se presentan algunas métricas referentes al conjunto de prueba, como se muestra en la figura 6.12. Se pueden apreciar las métricas agregadas para las tareas de clasificación. El rendimiento subóptimo del modelo puede atribuirse principalmente a la pequeñez del conjunto de prueba.

A lo largo de esta tesis, se ha demostrado que, gracias a las mejoras y extensiones implementadas por el autor, ahora *DashAI* es capaz de realizar el *fine-tuning* de modelos más avanzados. Las contribuciones del autor han permitido que los usuarios puedan seleccionar una mayor cantidad de tareas, cargar y configurar *datasets* a través de *dataloaders*, de manera

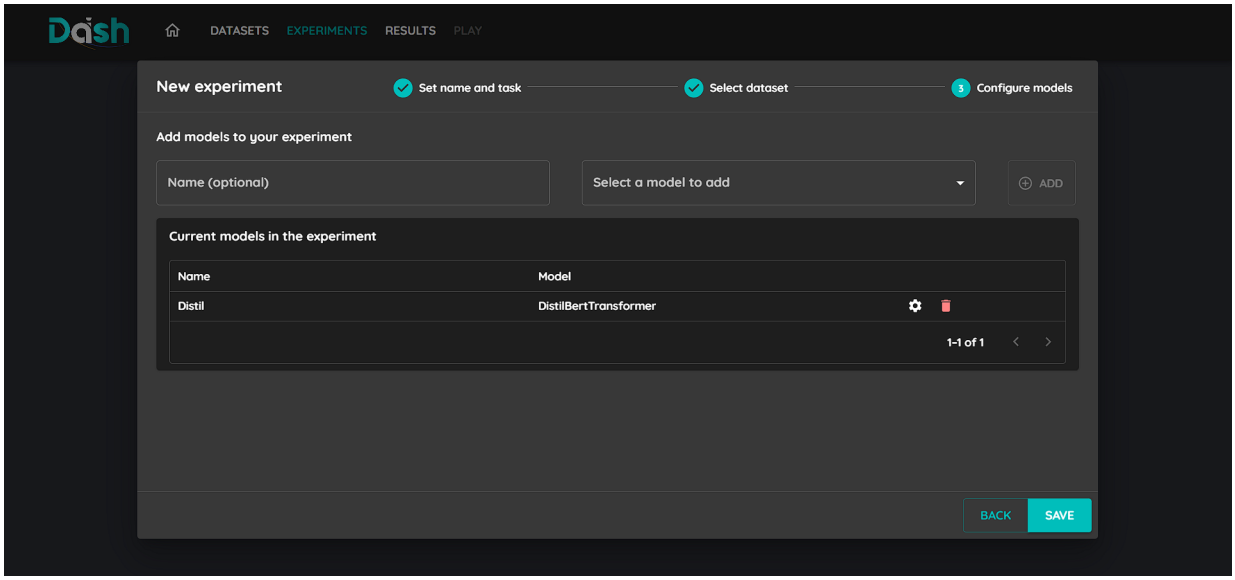

Figura 6.8: Inclusión del modelo para entrenamiento.

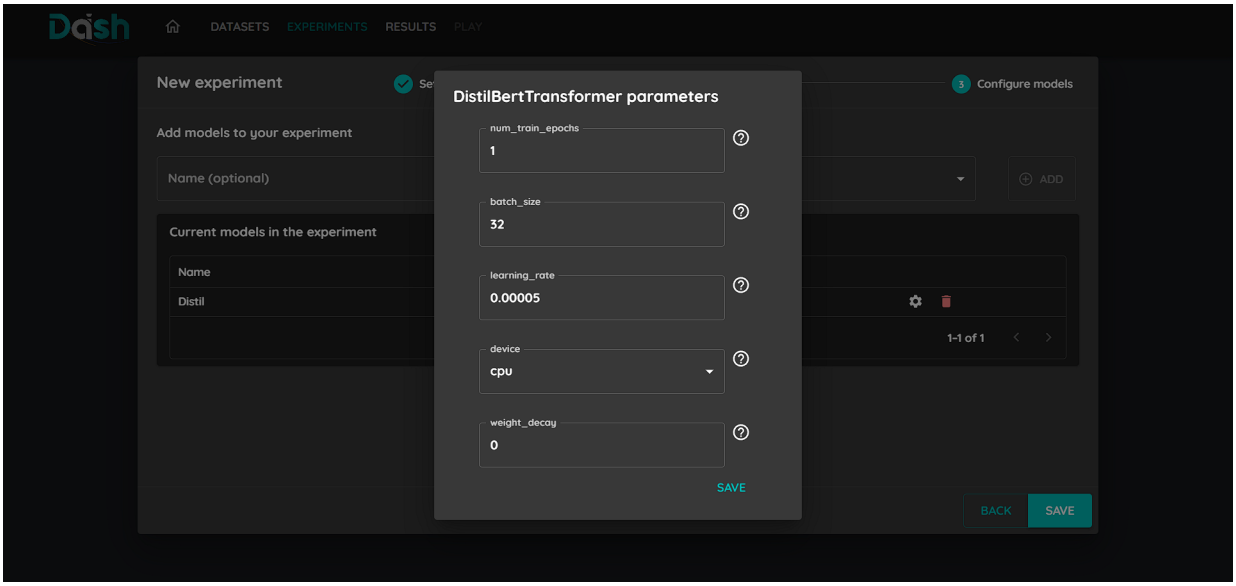

Figura 6.9: Configuración del modelo.

sencilla y directa. Asimismo, se ha logrado habilitar la selección y configuración de una mayor variedad de modelos. Adicionalmente, los usuarios ahora pueden visualizar una amplia gama de métricas para evaluar de forma exhaustiva el rendimiento de sus modelos. En resumen, con esta evaluación se pudo notar que el trabajo del memorista ha enriquecido *DashAI* con una serie de funcionalidades y características que la convierten en una herramienta aún más completa y accesible para la experimentación en ML.

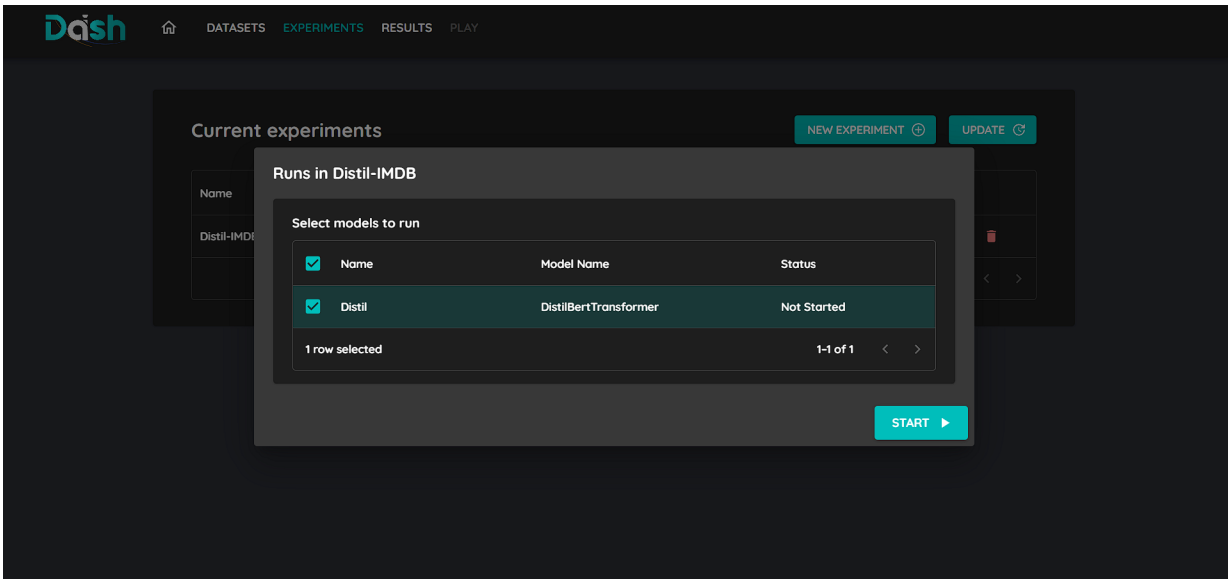

Figura 6.10: Ejecución de experimentos.

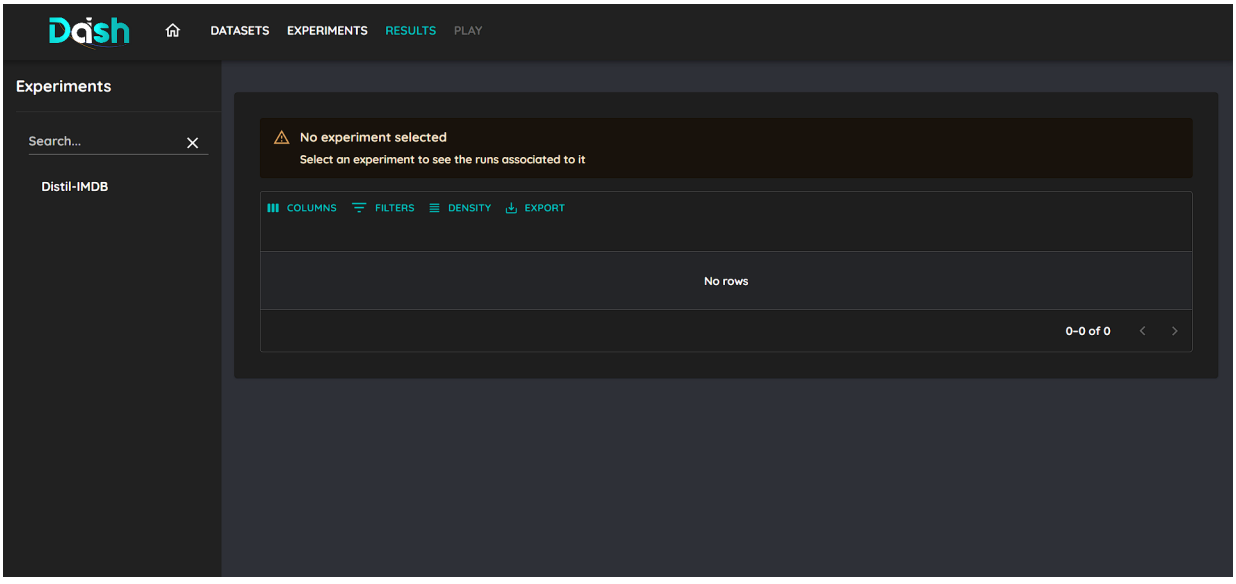

Figura 6.11: Pestaña de resultados.

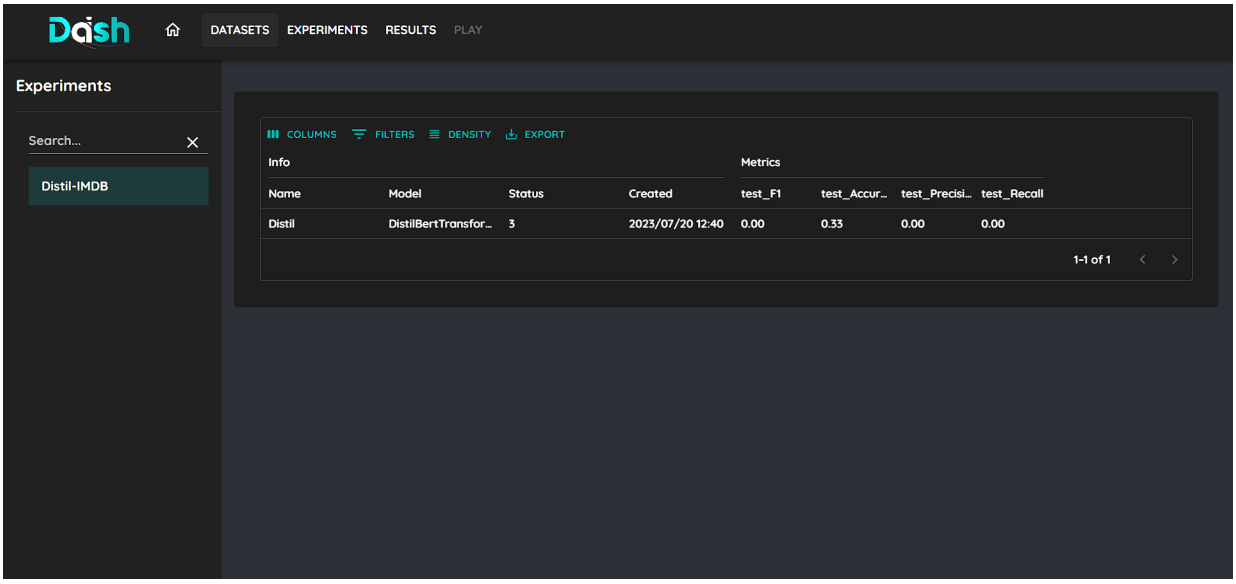

Figura 6.12: Métricas del experimento

Capítulo 7

Conclusiones

En este último capítulo se realizará una revisión retrospectiva de los logros, dificultades y lecciones aprendidas a lo largo del desarrollo de esta memoria en la sección 7.1. Posteriormente, en la sección 7.2 se presentarán diversas mejoras que pueden construirse sobre la base de este trabajo, aportando aún más al desarrollo de *DashAI*. Juntas, estas dos secciones proporcionarán una visión global y un camino a seguir basados en los hallazgos y experiencias de esta memoria.

7.1. Retrospectiva

La experiencia de trabajar en el proyecto *DashAI* ha demostrado ser tanto desafiante como enriquecedora para el estudiante. Entre los desafíos más notables se encontraba la necesidad de familiarizarse y trabajar con diversas librerías de *software* desconocidas anteriormente, integrarse a un interesante y prometedor nuevo proyecto y ajustarse en la dinámica de grupo que tenía este. Superar este desafío no sólo proporcionó un entendimiento más profundo de herramientas técnicas, sino que también mejoró su capacidad para adaptarse y aprender en un entorno tecnológico en constante evolución. Un aspecto crucial a la cual el memorista debía llevarle el ritmo es a la comunicación efectiva y es que con numerosos colaboradores trabajando en tareas diferentes pero interrelacionadas, y con la guía de profesores y expertos, la necesidad de comunicar ideas, progresos y dificultades de manera clara se volvió primordial. El estudiante aprendió rápidamente que la comunicación efectiva puede marcar la diferencia entre el progreso fluido y los malentendidos que pueden desacelerar el desarrollo del proyecto. En resumen, la participación en el proyecto *DashAI* ha sido un viaje de aprendizaje continuo, proporcionando al estudiante una experiencia invaluable en el campo del aprendizaje automático y el desarrollo de *software* en un entorno colaborativo y dinámico.

Por otro lado, en relación a los objetivos planteados en la sección 1.2, que fueron delimitados por los alcances y limitaciones especificados en la sección 1.3, se puede afirmar que todos se han cumplido con éxito. Es decir, *DashAI* ahora cuenta con más tareas y modelos que el usuario puede emplear a través de la GUI, además de ofrecer la capacidad de comparar y visualizar las métricas de desempeño correspondientes a los modelos. Este desarrollo repre-

senta un paso fundamental para expandir la utilidad de *DashAI*, incrementando su relevancia para una variedad más amplia de usuarios que buscan soluciones de aprendizaje automático. En resumen, el trabajo llevado a cabo en esta memoria actúa como un catalizador para el crecimiento y éxito de *DashAI*.

7.2. Trabajo Futuro

A pesar del cumplimiento de los objetivos hay muchas líneas de trabajo que pueden y seguirán avanzando, algunas de ellas fueron comentadas de forma superficial en la sección 1.3. Los asociados directamente a esta memoria es la de agregar más tareas y modelos, o poder agregar más formas de configurar los modelos a medida de que se va desarrollando el *frontend*. Lo mismo con las métricas, tanto incorporar más métricas como poder agregar más configuraciones a las existentes. Por otro lado, hay muchas *features* que se quieren incorporar a *DashAI* que son etapas intermedias a lo visto en este trabajo y que serán desarrollados por otros memoristas, estas son:

- Carga de datos que incluya la visualización del *dataset*, capacidad de cambiar tipos, *data augmentation*, *onehot-encoding*, eliminar columnas, *feature selection*, etc.
- Optimización de hiperparámetros a través del método de la grilla u otros más avanzados proporcionados por librerías.
- Explicabilidad de los modelos para proporcionar más transparencia respecto a las decisiones que toman estos.

Esto demuestra que queda mucho trabajo futuro para convertir a *DashAI* en una herramienta aún más integral y versátil para el aprendizaje automático, proporcionando a los usuarios una plataforma única que pueda adaptarse a una diversidad de necesidades y aplicaciones. En definitiva, el futuro de *DashAI* se presenta prometedor, con un potencial inmenso para el crecimiento y la innovación.

Bibliografía

[1] Amazon sagemaker. `https://aws.amazon.com/sagemaker/`. Accessed: 2023-06-18.

[2] Automl. `https://cloud.google.com/automl/`. Accessed: 2023-06-18.

[3] Huggingface autotrain. `https://huggingface.co/autotrain`. Accessed: 2023-06-18.

[4] Ibm watson studio. `https://www.ibm.com/cloud/watson-studio`. Accessed: 2023-06-18.

[5] Vertex ai. `https://cloud.google.com/vertex-ai`. Accessed: 2023-06-18.

[6] Artificial intelligence market size, share & trends analysis report by solution, by technology (deep learning, machine learning, natural language processing, machine vision), by end use, by region, and segment forecasts, 2022 - 2030. *Grand View Research*, 1:1–125, 04 2022.

[7] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning, 2016.

[8] Bard. Bard: A large language model from google ai. `https://bard.google.com/`, 2023. Accessed: 2023-06-25.

[9] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[10] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. *CoRR*, abs/2005.12872, 2020.

[11] Josh Cowls, Antonios Tsamados, Mariarosaria Taddeo, and Luciano Floridi. The ai gambit: leveraging artificial intelligence to combat climate change—opportunities, challenges, and recommendations. *AI & Society*, 38(1):283–307, 2023.

[12] Theodoros Evgeniou and Massimiliano Pontil. Support vector machines: Theory and applications. volume 2049, pages 249–257, 09 2001.

[13] GlobeNewswire. Generative ai market observes strong growth potential with projected market size of usd 151.9 bn by 2032. apr 2023.

[14] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. 11(1):10–18, nov 2009.

[15] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, sep 2020.

[16] IEEE. A brief review of nearest neighbor algorithm for learning and classification. *IEEE Conference Publication*, 2019.

[17] Fortune Business Insights. Artificial intelligence (ai) market size, share covid-19 impact analysis, by component (hardware, software, and services), by technology (computer vision, machine learning, natural language processing, and others), by deployment (cloud, on-premises), by industry (healthcare, retail, it telecom, bfsi, automotive, advertising media, manufacturing, and others), and regional forecast, 2021-2029. 2021.

[18] Wes McKinney. pandas: a foundational python library for data analysis and statistics. 2011.

[19] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler. Yale: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 935–940. ACM, 2006.

[20] OpenAI. Chatgpt. `https://openai.com/chatgpt`, 2023. Accessed: 2023-06-25.

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[22] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Andreas Müller, Joel Nothman, Gilles Louppe, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python, 2018.

[23] Bambang Hadi Prabowo. The digital economy and the importance of technology in economic growth. *Tamansiswa Accounting Journal International*, 2022.

[24] IBM Research. Ai hardware.

[25] Alejandro Rodríguez-González, Massimiliano Zanin, and Ernestina Menasalvas-Ruiz. Public health and epidemiology informatics: Can artificial intelligence help future global challenges? an overview of antimicrobial resistance and impact of climate change in disease epidemiology. *Yearbook of medical informatics*, 28(1):224–231, 2019.

[26] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

[27] Sören Sonnenburg, Mikio Braun, Cheng Soon Ong, Samy Bengio, Léon Bottou, Geoffrey Holmes, Yann Lecun, Klaus-Robert Müller, Fernando Pereira, Carl Rasmussen, Gunnar Rätsch, Bernhard Schölkopf, Alexander Smola, Pascal Vincent, Jason Weston, and Robert Williamson. The need for open source software in machine learning. *Journal of Machine Learning Research*, 8:2443–2466, 10 2007.

[28] Rachel Thomas. Google's automl: Cutting through the hype, 2018.

[29] Jörg Tiedemann. The Tatoeba Translation Challenge – Realistic data sets for low resource and multilingual MT. In *Proceedings of the Fifth Conference on Machine Translation*, pages 1174–1182, Online, November 2020. Association for Computational Linguistics.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, L ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[31] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

[32] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[33] Bichen Wu, Chenfeng Xu, Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Zhicheng Yan, Masayoshi Tomizuka, Joseph Gonzalez, Kurt Keutzer, and Peter Vajda. Visual transformers: Token-based image representation and processing for computer vision, 2020.

[34] Ying Xiong. The impact of artificial intelligence and digital economy consumer online shopping behavior on market changes. *Discrete Dynamics in Nature and Society*, 2022.

[35] YEC. Four ways artificial intelligence is transforming e-commerce. feb 2022.