Universidad de la República

Facultad de Ingeniería

Aplicación de MLOps en la Detección de Irregularidades en Redes Eléctricas

Informe de Proyecto de Grado presentado por

Lerder Alexander Martins Masner
Maximiliano Andrés Barragán Pavoni

como requisito de graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Lorena Etcheverry
Pablo Massaferro

Montevideo, 7 de abril de 2026

Agradecimientos

Queremos expresar nuestro más sincero agradecimiento a todas las personas que estuvieron cerca de nosotros durante este proceso y nos alentaron a seguir adelante.

A Cecilia, esposa de Alexander, por su incondicional apoyo y compañía en todo momento. Su aliento constante fue fundamental para que él lograra esta meta.

A Abril, Chichila, Graciela, Majo y Jorge, que acompañaron a Maximiliano durante toda la carrera, y siempre dieron las palabras y fuerzas necesarias para salir adelante y lograr los objetivos. Fueron pilares fundamentales para llegar hasta aquí, y se les va a estar eternamente agradecido.

Va el agradecimiento y reconocimiento también para todos los compañeros con los que tuvimos el gusto de compartir estos años, y especialmente a los compañeros que se convirtieron en amigos durante el proceso. Con ellos hemos superado muchos obstáculos, hemos tenido inmensas jornadas de estudio, y hemos aprendido muchísimas cosas juntos. Sin ellos esto tampoco sería posible.

Nuestra gratitud también va dirigida a Lorena Etcheverry y Pablo Massaferro, del Instituto de Computación y del Instituto de Ingeniería Eléctrica de la Universidad de la República. Gracias por su tutoría, consejos y empatía durante esta etapa. Su participación fue esencial en el recorrido de nuestra Tesis de Grado.

Finalmente, dedicamos este logro a los seres queridos que ya no están presentes, como Carmen, quienes seguramente estarían muy orgullosos de este momento. Parte de esto se debe a su apoyo y compañía mientras estuvieron a nuestro lado.

Resumen

La Industria 4.0, caracterizada por su evolución basada en la fabricación de tecnologías digitales, integra el Aprendizaje Automático (AA) o Machine Learning (ML) como un pilar central para procesos industriales más inteligentes. En este contexto, *Machine Learning Operations* (MLOps) surge como un conjunto de prácticas, técnicas y herramientas para la implementación efectiva en ML.

Este proyecto aborda la aplicación de técnicas de MLOps al software Detector Automático de Irregularidades en Consumos Eléctricos (DAICE) y Detector Automático de Irregularidades en Consumos Eléctricos con técnicas de aprendizaje profundo (DeepDAICE). Estos sistemas fueron desarrollados en colaboración entre la Facultad de ingeniería de la Universidad de la República (UdelaR) y la Subgerencia de Recuperación de Energía de UTE. Ambos se enfocan en la detección de irregularidades en las redes eléctricas, adquiriendo importancia a medida que los datos se vuelven cada vez más masivos, utilizando técnicas de ML para clasificar patrones de consumo e información de clientes.

El software utilizado actualmente está fuertemente basado en entornos iterativos de ejecución, como por ejemplo, Jupyter Notebooks en Python. En esta tesis se propone llevar estos procesos a entornos de producción modularizados y automatizados. De esta manera, se busca registrar los diferentes modelos y artefactos generados, así como los resultados que se desprenden del proceso, y *operacionalizar* la tarea de detección de estas irregularidades.

Un desafío clave para esta tesis fue el estudio y utilización de un conjunto de herramientas de código abierto que permitan tener una solución de extremo a extremo. Se puso especial énfasis en el uso de herramientas como Airflow, MLflow y TFX. Airflow permitió la creación del flujo de datos a partir de la definición de Grafos Acíclicos Dirigidos (DAG - *Directed Acyclic Graph*), el paso de parámetros y ejecución programada; MLflow se utilizó para el registro de experimentos, modelos, métricas y artefactos; y se experimentó con TFX, una solución integral que además incluye ejemplos de su uso detallados en la documentación, para el tratamiento de los datos.

A través de ejemplos prácticos y casos de estudio, en este trabajo se muestran las soluciones propuestas para abordar los desafíos en la detección de irregularidades en las redes eléctricas. Se subraya la importancia de la calidad de los datos, el versionado de modelos, la evaluación del rendimiento y el despliegue. La tesis concluye destacando la relevancia de esta primera experiencia como un paso en dirección a alcanzar mayores niveles en la madurez en la aplicación de

MLOps, alineándose con las tendencias de la Industria y abriendo camino para futuros trabajos de aplicación de este tipo de técnicas en mayor profundidad y de forma generalizada en la empresa.

Palabras clave: DAICE, DeepDAICE, UTE, Aprendizaje Automático, MLOps, Machine Learning, Pipeline, DAG, Airflow, TaskFlow, MLflow, Industria 4.0, Automatización, Sistemas de Apredizaje Automático, Arquitectura, Detección de irregularidades, Redes eléctricas, Redes neuronales, TFX

Índice general

Capítulo 1

Introducción

La industria 4.0, caracterizada por su evolución hacia la fabricación de tecnologías digitales, integra el aprendizaje automático (ML por sus siglas en inglés) como un pilar central para procesos industriales más inteligentes. En este ámbito, Machine Learning Operations (MLOps) emerge como un conjunto crítico de prácticas, técnicas y herramientas para la implementación efectiva de ML en la práctica industrial. A pesar de la prevalencia de DevOps (Desarrollo y Operaciones) en sistemas de software puros, su aplicación en sistemas de la Industria 4.0 ha recibido menos atención. Esto destaca la necesidad de explorar cómo MLOps puede adaptarse y potenciar los sistemas en la industria actual, aprovechando su capacidad para manejar complejas operaciones de datos y modelos de aprendizaje automático [1] [2].

MLOps, derivado de los principios especializados de DevOps para aplicaciones de ML, busca la integración de desarrollo de software con operaciones de sistemas de TI. Este enfoque se alinea con los objetivos de eficiencia y agilidad en la Industria 4.0, donde la rápida innovación y adaptación son cruciales. Las actividades asociadas con MLOps incluyen, pero no se limitan a, la recolección, análisis y preparación de datos, así como la construcción, entrenamiento, evaluación, selección, empaquetado, despliegue y monitoreo de los modelos [1].

El ciclo de vida de MLOps en la Industria 4.0 se caracteriza por fases como Ingeniería de Datos, Ingeniería de Modelos y Operaciones, cada una con actividades específicas y apoyadas por tareas adicionales. Estas fases se ven influenciadas por la arquitectura general del entorno de desarrollo en el que se implementan. Los desafíos identificados en la aplicación de MLOps en la Industria 4.0 se categorizan según las actividades de MLOps, abarcando desde la ingeniería de datos hasta el soporte operativo y el monitoreo de los resultados [1].

Aunque muchos desafíos de MLOps en la Industria 4.0 son similares a los encontrados en el contexto tradicional de ingeniería de software, existen desafíos adicionales específicos de ciertos escenarios en su aplicación. La complejidad y la diversidad del entorno de la industria actual, particularmente en lo que respecta a los sistemas ciberfísicos y a la heterogeneidad de las plataformas de hardware

y software, presentan retos únicos. La planificación futura incluye estudios industriales más detallados sobre MLOps y la aplicación de estos conocimientos a plataformas y marcos de trabajo diseñados para implementar MLOps en la Industria 4.0 [1].

La operacionalización de modelos de ML en producción presenta desafíos significativos, tal como se destaca en un estudio sistemático de la literatura [3]. Este trabajo indica que, a pesar de que muchos estudios se enfocan en adaptar enfoques y técnicas de ingeniería de software (SE) para ML, pocos han resumido el estado y los desafíos de operacionalizar modelos de ML. La operacionalización implica todos los pasos después del entrenamiento y evaluación del modelo, incluyendo el encapsulado en un formato adecuado para el despliegue, la publicación en un registro o almacenamiento de los modelos, la integración en un sistema de software más amplio, el servicio y el monitoreo.

En la industria, la adopción de ML aún está en sus primeras etapas y está creciendo rápidamente. La mencionada revisión muestra que la operacionalización de ML es un área que presenta desafíos reales para los profesionales, consistiendo en llevar un modelo de ML entrenado y evaluado a un estado de servicio en el entorno de producción previsto. Abordar estos desafíos requiere adoptar buenas prácticas y utilizar herramientas adecuadas. Estudios actuales, centrados en investigar en profundidad la operacionalización de ML, han puesto más énfasis en las herramientas y la infraestructura. Los estudios referentes a las técnicas y herramientas actuales para operacionalizar modelos de ML revelan varias oportunidades para investigar en la mejora de las herramientas e infraestructuras, como enfoques automáticos para cambiar modelos para cumplir con los objetivos de nivel de servicio esperado, herramientas para monitorear los pipelines de datos y modelos, y soluciones eficientes para implementar modelos de ML en dispositivos de borde [3].

La implementación exitosa de productos de ML en entornos de producción es un desafío considerable, a menudo complicado por la dificultad de automatizar y operacionalizar estos sistemas. El paradigma de MLOps ha emergido como una solución a estos retos. Las investigaciones que combinan revisiones de literatura, análisis de herramientas y entrevistas con expertos han proporcionado un panorama detallado de los principios, componentes y roles esenciales en MLOps, junto con la arquitectura y flujos de trabajo relacionados. Estos estudios también han ofrecido una definición completa de MLOps, destacando los desafíos pendientes en el área y proporcionando orientación a investigadores y profesionales de ML para automatizar y gestionar eficazmente sus productos de ML con tecnologías específicas [4].

En la actualidad, se desarrollan más productos de ML debido a la creciente disponibilidad de datos y la necesidad constante de innovación sobre dichos datos. Sin embargo, solo una pequeña proporción de estos conceptos avanzan hacia su implementación y producción. La investigación académica se ha centrado en gran medida en la construcción y benchmarking de modelos de ML, pero no tanto en la operación de sistemas de ML complejos en escenarios reales. MLOps aborda estos desafíos, facilitando la gestión de flujos de trabajo de ML que, hasta ahora, han sido manejados en gran medida de manera manual por

los científicos de datos. Un estudio de métodos mixtos, incluyendo el análisis de literatura existente, herramientas y entrevistas con expertos en el campo, ha revelado los aspectos clave de MLOps: principios, componentes, roles y arquitectura. Este conocimiento fomenta una comprensión común del concepto de MLOps y sus conceptos asociados, proporcionando una base para investigadores y profesionales para desarrollar con éxito productos de ML en el futuro [4].

En lo que respecta a las redes eléctricas, son vitales para el funcionamiento de nuestra sociedad, su monitoreo nos enfrenta a retos significativos debido a robos y manipulaciones en los sistemas de medición y sus cercanías. Estas acciones no autorizadas resultan en lo que se conoce como pérdidas no técnicas (NTL, por sus siglas en inglés), que generan impactos económicos y desafíos técnicos considerables al intentar subsanarlos. Según informes del Banco Interamericano de Desarrollo (BID), en regiones como América Latina y el Caribe, las pérdidas totales de energía eléctrica alcanzan aproximadamente el 17 % del total generado, traduciéndose en un coste económico anual de unos 11 mil millones de dólares. En este contexto, la detección y regularización de estas pérdidas no técnicas es de suma importancia para acercarse a una mayor eficiencia [5].

1.1. Pérdidas No Técnicas en el Consumo de Energía y Herramientas de Detección

Las NTL incluyen problemas como el robo de electricidad, medidores defectuosos y errores en la facturación. Estas pérdidas representan un desafío significativo para las empresas eléctricas, ya que pueden alcanzar hasta el 40 % de la electricidad distribuida en algunos países, causando un impacto económico considerable en el orden de millones de dólares. Además de las pérdidas financieras, las NTL también representan riesgos de seguridad y afectan la estabilidad y confiabilidad de la red eléctrica.

En Uruguay, el problema de las NTL es abordado mediante estudios analíticos, inspecciones periódicas y la evaluación de resultados para mejorar los niveles de detección continuamente [6]. Para dar un ejemplo, después de una intervención por una irregularidad, se puede observar una baja en el consumo de energía, seguida de una recuperación tras la inspección y la regularización del suministro, como se ilustra en la Figura 1.1.

En respuesta a estos desafíos, ha habido un creciente interés en la investigación en ingeniería eléctrica y ciencias de la computación por emplear inteligencia artificial para predecir NTL. Este enfoque ofrece una visión integral del impacto de las NTL en las economías, como la pérdida de ingresos y la reducción de la estabilidad y confiabilidad de las redes eléctricas, como se mencionó anteriormente. Se han revisado diversos métodos de detección de NTL que emplean inteligencia artificial, desde sistemas expertos hasta modelos de aprendizaje automático como técnicas de soporte vectorial y redes neuronales [7].

Para abordar este problema en Uruguay, se desarrollaron dos herramientas en colaboración entre la Administración de Usinas y Transmisiones Eléctricas

Figura 1.1: Ejemplo de variación en el consumo de energía tras la detección de una irregularidad con DAICE. El círculo negro representa la inspección, y el rombo rojo representa el aumento en el consumo luego de la inspección. Fuente: Monitoreo de consumo en UTE.

(UTE) y la Universidad de la República (UdelaR): el *Detector Automático de Irregularidades en Consumos Eléctricos* (DAICE), y su evolución, *DeepDAICE*, que incorpora técnicas de aprendizaje profundo (Deep Learning). Estas herramientas están diseñadas para optimizar la selección de inspecciones realizadas a nivel nacional, mejorando la precisión en la identificación de irregularidades y optimizando el uso de recursos, como los inspectores y vehículos. Gracias a DAICE, es posible identificar hurtos y otros tipos de irregularidades, recuperando la energía una vez regularizado el suministro (Figura 1.1).

En el contexto global, la detección de hurtos o estafas eléctricas y pérdidas no técnicas ha sido abordada con distintos enfoques, desde balances energéticos topológicos en la red hasta modelos de aprendizaje automático. Un estudio representativo es el Sistema de Detección de Fraudes (FDS), que ofrece una perspectiva técnica de la detección de NTL, enfocándose en algoritmos, requerimientos de datos y métricas, sin centrarse en las causas socioeconómicas. Este trabajo subraya la importancia de una buena gestión de modelos para un des-

empeño efectivo y proporciona un marco valioso para proyectos como DAICE, que buscan integrar enfoques técnicos avanzados y una gestión eficaz de los modelos de aprendizaje automático para mejorar la detección de irregularidades en el consumo eléctrico [8].

Actualmente, en el Departamento de Recuperación de Energía de UTE, se han utilizado principalmente soluciones enfocadas en la creación y optimización de modelos de Machine Learning, incluyendo la búsqueda de hiperparámetros y la mejora de la arquitectura de los modelos. Sin embargo, este enfoque, llevado a cabo a menudo mediante Jupyter Notebooks o scripts personalizados, carece del soporte de herramientas de orquestación, registro de experimentos o un enfoque estandarizado. Esto plantea dificultades adicionales, ya que los sistemas que se apoyan en aprendizaje automático presentan una compleja interdependencia entre datos y modelos, lo que hace que sean difíciles de mantener a mediano y largo plazo.

Aquí es donde MLOps se convierte en el protagonista, proporcionando un marco más amplio y estructurado para gestionar estos sistemas, mejorando el proceso de detección de pérdidas no técnicas.

1.2. Motivación para la Aplicación de MLOps en el Desarrollo de DAICE y DeepDAICE

Si bien DAICE y DeepDAICE han logrado avances significativos en la detección de irregularidades en redes eléctricas, enfrentan retos principalmente en la gestión y comparativa de modelos, así como en la creación y gestión de pipelines. Estos desafíos surgen no tanto del volumen de datos, sino de la necesidad de una gestión eficiente de modelos de aprendizaje automático y una infraestructura robusta para el manejo de experimentos.

MLOps ofrece una solución estructurada para estos desafíos, enfocándose en mejorar la gestión de modelos y la eficiencia de los procesos. La implementación de MLOps facilita la creación de pipelines más eficientes y la comparación efectiva de modelos, lo que es crucial para el éxito de proyectos como DAICE y DeepDAICE.

El enfoque principal de la adopción de MLOps en este contexto es optimizar la búsqueda de hiperparámetros y el entrenamiento de modelos. Estos procesos, críticos para la precisión y eficacia de los modelos de aprendizaje profundo, requieren infraestructuras que soporten la computación intensiva y el manejo avanzado de experimentos. Con MLOps, se busca implementar tecnologías que permitan una gestión más efectiva de estos aspectos, asegurando así un enfoque más estratégico y orientado a resultados en el tratamiento de los datos y el entrenamiento de modelos de DAICE y DeepDAICE.

En este contexto, la aplicación de MLOps se vuelve fundamental para la gestión, monitoreo y optimización de todo el ciclo de vida de los modelos de aprendizaje automático. Además, teniendo en cuenta que los datos son uno de los insumos más importantes, gestionarlos de manera eficiente es crucial. No

solo se trata de construir modelos, sino de asegurarse de que los resultados obtenidos sean precisos. Esta disciplina ofrece soluciones para la trazabilidad, el despliegue, la automatización y el control de los modelos en producción, lo que puede permitir que los resultados obtenidos sean confiables y se mantengan en el tiempo.

1.3. Aplicación de MLOps para asegurar la calidad de los datos

Ahora bien, si MLOps es el guardián de la eficiencia en el aprendizaje automático, ¿qué papel juega en la calidad de los datos? En el contexto de DAICE, la implementación de cambios en los procedimientos almacenados o consultas relacionados con la extracción de datos desde las bases de datos es una tarea crítica que requiere meticulosidad y precisión. No podemos subestimar la importancia de la calidad de los datos, especialmente cuando se trata de aplicaciones tan sensibles como la detección de irregularidades en redes eléctricas. Aquí, MLOps no solo asegura que los modelos funcionen como deben, sino que también facilita que los datos en los que se basan los algoritmos sean de la más alta calidad, debido a que se permite integrar procesos de verificación de calidad tan rigurosos como se desee.

Para garantizar la integridad de los datos obtenidos y asegurar la efectividad del sistema, es necesario llevar a cabo verificaciones exhaustivas. Entre estas verificaciones, se incluyen métricas de correlación, análisis exploratorio de datos e información estadística. Sin embargo, realizar estas tareas manualmente puede ser una labor complicada y propensa a errores.

En este sentido, la aplicación de MLOps en DAICE proporciona soluciones para mejorar la gestión y el control de estos procesos. Mediante la automatización de pruebas, es posible realizar verificaciones sistemáticas cada vez que se realicen cambios en los procedimientos de extracción de datos. Esto garantiza que los datos sean confiables y estén libres de errores conocidos o contemplados, lo que a su vez mejora la precisión de los modelos de detección de irregularidades.

La eficiencia y la precisión son fundamentales en los sistemas de aprendizaje automático. Con la aplicación de MLOps, se puede mejorar en estas áreas, enfocándonos en la monitorización continua del sistema y la calidad de los datos. Esto permite corregir desviaciones o incongruencias de manera más proactiva, mejorando así el rendimiento operativo.

La automatización del despliegue, una de las ventajas de MLOps, permite mayor agilidad en la implementación de cambios, reduciendo el tiempo de espera y la posibilidad de errores humanos, aspectos críticos en un entorno de producción.

1.4. Aplicación de MLOps para Asegurar la Mejora Continua de Modelos

La implementación de un sistema de gestión de modelos con MLOps significa un cambio muy importante en la forma en la que se gestionan los modelos en el Departamento de Recuperación de Energía (DRE) de UTE. Con MLOps, la automatización y la trazabilidad no son solo características deseables, son estándares. Ya que permite registrar y almacenar métricas y resultados en una base de datos centralizada, podemos tener una visión completa de todas las ejecuciones de modelos a lo largo del tiempo.

Esto no solo facilita la recuperación de resultados históricos, sino que también nos permite comparar diferentes versiones de modelos. Por si fuera poco, garantiza que solo los modelos con un rendimiento satisfactorio sean liberados en producción. Es decir, MLOps no solo mejora nuestros modelos actuales, sino que también nos prepara para el futuro, asegurando que el sistema sea siempre mejor versión tras versión.

Otro aspecto importante es la eficiencia y productividad que se logra con MLOps. La automatización y trazabilidad proporcionadas por esta metodología reducen considerablemente el tiempo y el esfuerzo requeridos para llevar a cabo pruebas de rendimiento y comparaciones de modelos. Al eliminar la necesidad de realizar estas tareas manualmente en Jupyter Notebooks, los científicos de datos y analistas pueden dedicar más tiempo a la investigación y el desarrollo de modelos más avanzados.

1.5. Planteo del Problema

La detección de irregularidades en redes eléctricas es un desafío significativo para compañías eléctricas como UTE, ya que estas pueden causar pérdidas económicas considerables. En proyectos previos, como **DAICE** y **DeepDAICE**, se han utilizado algoritmos de aprendizaje automático para identificar irregularidades en el consumo de energía. Sin embargo, estos sistemas carecen de una infraestructura de **MLOps** que permita gestionar el flujo de datos y modelos de forma automática e integrada.

Actualmente, uno de los principales problemas es la ausencia de un sistema que permita la operacionalización completa del proceso de detección de irregularidades. Esto incluye etapas que van desde la gestión de los datos hasta la validación y monitoreo de modelos, lo cual limita la capacidad de la empresa para optimizar el sistema de detección de irregularidades. Además, sin un control adecuado de las métricas y versiones de los modelos, es difícil mantener la confiabilidad del sistema en el tiempo.

1.6. Objetivos Generales y Particulares

1.6.1. Objetivo General

Investigar y aplicar conceptos de MLOps a las herramientas **DAICE** y **DeepDAICE** con el fin de diseñar e implementar un prototipo de sistema que permita la detección automática de irregularidades en redes eléctricas, complementado con funcionalidades de visualización y monitoreo de métricas de desempeño, gestión de configuraciones de los modelos de aprendizaje automático, y el uso de pipelines para el procesamiento de datos y modelos.

1.6.2. Objetivos Particulares

- **Investigación y Aplicación de MLOps**: Estudiar en profundidad los conceptos de MLOps y determinar cuáles de sus fases o pasos pueden aplicarse de manera efectiva al contexto de detección de irregularidades en redes eléctricas.

- **Desarrollo de un Pipeline de Datos**: Diseñar y construir un pipeline para la extracción, validación, y preparación de datos. Este pipeline debe integrar datos de diversas fuentes, tanto relacionales como no relacionales (por ejemplo, bases de datos Oracle, HBase y/o archivos).

- **Gestión y Monitoreo de Modelos**: Crear un sistema que permita la gestión completa del ciclo de vida de los modelos, incluyendo el preprocesamiento de datos, el entrenamiento, la validación y la evaluación de modelos en algoritmos como XGBoost y TensorFlow. Implementar un sistema de versionado que mantenga un registro detallado de las versiones de los datos utilizados en cada etapa del flujo de trabajo.

- **Visualización y Monitoreo de Métricas**: Diseñar e implementar un módulo para visualizar métricas de desempeño de los modelos y de los datos utilizados en el proceso de detección de irregularidades, lo que permitirá monitorear la calidad de los datos de entrada y la precisión de los modelos generados.

1.7. Resultados Esperados

Al finalizar el proyecto, se espera contar con un prototipo funcional que cumpla con los siguientes requisitos:

- **Pipeline de Modelos**: Un flujo completo para el entrenamiento, evaluación y clasificación de modelos de aprendizaje automático en entornos de producción, garantizando la trazabilidad de cada etapa del proceso.

- **Pipeline de Datos**: Un sistema que facilite la extracción, validación y preparación de datos, asegurando que los datos sean consistentes y cumplan con los requisitos estadísticos necesarios para su uso en modelos de aprendizaje automático.
- **Entrenamiento y Validación de Modelos**: La posibilidad de entrenar modelos de aprendizaje automático utilizando diversas fuentes de datos, incluyendo bases de datos relacionales y archivos.
- **Visualización de Métricas de Desempeño**: Un módulo que permita al usuario monitorear las métricas de rendimiento de los modelos, evaluando su precisión, recall, F1 y otras métricas relevantes.
- **Versionado y Registro de Modelos**: Un sistema de registro de modelos que mantenga un historial detallado de los modelos generados, sus hiperparámetros, métricas de desempeño y versiones de los datos utilizados.
- **Funcionalidades Adicionales Opcionales**:
  - Sistema de autenticación de usuarios y gestión de permisos.
  - Módulo para la visualización de resultados de inspecciones de campo, en caso de que se incluya como una funcionalidad adicional en la implementación.

Estos resultados permitirán una gestión más robusta, escalable y trazable del sistema de detección de irregularidades en redes eléctricas, optimizando la operativa de UTE en la identificación de anomalías en el consumo eléctrico.

1.8. Resumen Extendido del Proyecto

Para llevar a cabo el proyecto de detección automática de irregularidades en redes eléctricas y aplicar conceptos de MLOps a los sistemas DAICE y DeepDAICE, se siguió la siguiente metodología:

Inicialmente, dado el primer objetivo de estudiar MLOps y entender de qué manera sus técnicas podían ser aplicables para ayudar en el contexto actual, se comenzó por realizar una búsqueda bibliográfica, no solo para conocer que tendencias se siguen en la industria, sino para entender realmente qué es MLOps. De esta búsqueda bibliográfica, surgieron diversas herramientas relacionadas, que también se estudiaron, en mayor o menor profundidad.

Habiendo explorado la bibliografía sobre MLOps y explorado algunas herramientas, se decidió consultar con un referente en el área, para validar el proceso y lo estudiado hasta el momento.

Luego de la elección de las herramientas, se procedió a la configuración del entorno de trabajo para comenzar a explorar las elegidas en primera instancia. Se instaló Airflow en Windows utilizando *Windows Subsystem for Linux* (WSL), y

se configuraron los requisitos y dependencias necesarios para su correcto funcionamiento. También se instaló y se configuró MLflow en el entorno para realizar el registro de modelos y el tracking de parámetros y demás artefactos.

A continuación, se diseñaron distintos *Directed Acyclic Graph* (DAG), a modo de Prueba de Concepto, para realizar las primeras validaciones de Airflow y su integración con MLflow. Como parte de estas pruebas de concepto, se implementaron DAGs que buscaban hacer un sencillo preprocesamiento de datos, el entrenamiento de algún modelo simple, entre otros, definiendo cada una de estas etapas como un *step* de cada DAG. Además, se probó la integración con MLflow haciendo el registro de modelos con el Model Registry de la herramienta, y también el registro de distintos metadatos como hiperparámetros, métricas de rendimiento, versiones de los datos utilizados, entre otros, por medio de la API de Tracking de MLflow.

Además, como parte del desarrollo de las pruebas de concepto de este proyecto, y como parte de la validación de las herramientas elegidas, se realizó una prueba de implementación de *TensorFlow Extended* (TFX). Durante esta validación, se siguieron algunos de los ejemplos típicos expuestos en la documentación oficial de TFX [1]. Estos ejemplos típicos se ejecutaron primero en la plataforma *Google Colab*, y luego se llevó uno de estos ejemplos al ambiente de trabajo local utilizando Airflow.

Una vez configurado el entorno, realizadas las pruebas de concepto, y validadas las herramientas a utilizar, se trabajó en la definición de una arquitectura que permita trabajar, de manera modular, distintas etapas generalmente encontradas en problemas de Machine Learning.

Tras la selección y validación de las herramientas, y la definición de la arquitectura propuesta, se procedió a refactorizar y modularizar la herramienta DAICE. Estos ajustes fueron un paso clave para permitir y facilitar la posterior incorporación de técnicas de MLOps y utilizar dichas herramientas.

Luego, se trabajó en la implementación de distintos DAGs que, apoyados en la arquitectura definida, permitan realizar la validación y verificación de datos. Además, se aplicaron procesos de validación y verificación de datos dentro del flujo de trabajo, asegurando la calidad de los datos utilizados en el entrenamiento y clasificación de los modelos.

Esta metodología resultó en un sistema útil para la detección de irregularidades en el consumo eléctrico. Los conceptos de MLOps aplicados, junto con la modularización de DAICE, permitieron diseñar y ejecutar un flujo de trabajo efectivo, con capacidades de visualización, monitoreo y reutilización de modelos registrados. El resultado es un sistema capaz de gestionar integralmente el ciclo de vida de los datos y modelos, alineado con las mejores prácticas de la industria, y que permite operacionalizar el trabajo realizado actualmente por el Departamento de Recuperación de Energía de UTE.

[1] El tutorial que usaba el conjunto de datos de taxis de Chicago y el conjunto de datos de pingüinos, que previamente estaban alojados en Google Cloud, ahora están disponibles en la plataforma TensorFlow.

1.9. Descripción del Documento

A continuación, se describen los siguientes capítulos del proyecto:

En el **Capítulo 1** se ofrece un contexto general sobre las pérdidas no técnicas de energía (NTL), las herramientas de detección disponibles, y la motivación para aplicar MLOps en la mejora de la calidad de datos y la actualización continua de modelos. También se plantean el problema a resolver, los objetivos generales y específicos, y los resultados esperados. Para finalizar el capítulo, se presenta también un resumen extendido del proyecto.

El **Capítulo 2** está dedicado al estado del arte, abarcando tendencias actuales en la operacionalización del aprendizaje automático (AA), las definiciones de MLOps según distintas fuentes, y una revisión de niveles de madurez según Google Cloud. También se incluye una entrevista con un referente en herramientas de aprendizaje automático.

En el **Capítulo 3**, se exploran diversas herramientas existentes en el ámbito de MLOps, incluidas Airflow, MLflow, TFX, DVC, Kubeflow, entre otras. Además, se presenta un análisis de algunas herramientas complementarias como Apache Hadoop, HBase, Spark y el uso de WSL, así como la justificación de las herramientas seleccionadas.

El **Capítulo 4** define el problema en profundidad y detalla los requerimientos funcionales y no funcionales de la solución, abordando los desafíos específicos de trazabilidad de modelos y verificación de datos en producción.

En el **Capítulo 5**, se presenta la arquitectura de la solución, con diagramas y descripciones técnicas de cada componente.

El **Capítulo 6** expone la aplicación de la solución en el caso de DAICE, explicando la migración a Airflow y MLflow, el uso de TFX en DeepDAICE, y la evaluación de modelos mediante métricas como precisión y recall. También se detallan los procesos de integración con la Industria 4.0, la estandarización de pipelines, y la creación de DAGs para AA.

Finalmente, en el **Capítulo 7** se presentan las conclusiones del proyecto, los resultados alcanzados, reflexiones finales, y propuestas de trabajo a futuro, enfocadas en la mejora continua de la solución planteada y su escalabilidad en la detección de irregularidades en redes eléctricas.

Capítulo 2

Fundamentos Teóricos y Avances Recientes en MLOps para la Detección de Irregularidades en Redes Eléctricas

2.1. Tendencias actuales relacionadas con la Operacionalización del Aprendizaje Automático

Como se comentó en el capítulo de introducción, las nuevas aplicaciones y productos a desarrollar en la industria se están volviendo cada vez más centrados en el uso de Aprendizaje Automático (AA). En general, los modelos de AA siguen siendo una pequeña parte dentro del ecosistema de tecnologías involucradas en un sistema de software basado en AA. Se sabe que los sistemas AA, es decir, un sistema de software que incorpora Aprendizaje Automático como una de sus partes, agregan varios aspectos nuevos que antes no se conocían en el panorama del desarrollo de software. Además, los flujos de trabajo manuales de AA son una fuente importante de deuda técnica [9]. A diferencia del software estándar, los sistemas de AA tienen un entrelazamiento complejo con los datos cuando lo comparamos con el código estándar. Esta relación compleja hace que estos sistemas sean bastante más difíciles de mantener a mediano y largo plazo. Diferentes expertos, como desarrolladores de aplicaciones, científicos de datos, ingenieros de dominio, entre otros, típicamente tienen que trabajar en conjunto en este tipo de sistemas. También, a diferencia de los sistemas de código

estándar, los sistemas que integran AA involucran diferentes ciclos de desarrollo temporal y diferentes herramientas, y la gestión de datos se está convirtiendo en el nuevo enfoque en los sistemas basados en AA. Como reacción a estos desafíos, surge *Machine Learning Operations* (MLOps). Esta disciplina (MLOps) puede ayudar a evitar y/o reducir esta deuda técnica generada, ya que promueve la automatización de todos los pasos involucrados en la construcción de un sistema AA, desde el desarrollo hasta la puesta en producción. Formalmente, MLOps es una disciplina que está formada por una combinación de AA, DevOps e ingeniería de datos, para implementar sistemas de AA de manera confiable y eficiente [9].

En los enfoques de resolución de problemas basados en datos, la calidad de los datos y su preprocesamiento son fundamentales para obtener resultados que generalicen bien [10]. Es bien sabido que los datos son parte central de cualquier sistema AA y generalmente se pueden dividir en tres categorías: *datos estructurados*, *datos semiestructurados* y *datos no estructurados* [9]. El procesamiento de grandes volúmenes de datos con AA y en especial con Redes Neuronales Profundas (DNN) es un proceso complejo que requiere muchos recursos. Dado que el entrenamiento de DNN requiere una cantidad significativa de datos, la evaluación automatizada de la calidad de los datos es un paso crítico en un flujo de trabajo de MLOps. Se sabe que si la calidad de los datos se ve comprometida en algún nivel específico, esto hace que el sistema sea más propenso a fallar [9]. Es un desafío definir métricas para la evaluación de calidad automatizada porque métricas como integridad o consistencia, no siempre se pueden definir universalmente. Por otro lado, los datos estructurados son más fáciles de procesar, ya que se pueden definir claramente los tipos de datos, las calificaciones de calidad (establecidas por expertos en el dominio) y las pruebas de evaluación de calidad [9].

Además, hay varias herramientas disponibles para crear un flujo de trabajo de MLOps automatizado (ver figura 2.1). Estas herramientas se pueden utilizar para lograr mejores resultados desde el desarrollo de un modelo hasta la implementación y el mantenimiento. En la mayoría de los casos, la creación de un flujo de trabajo de MLOps requiere varias herramientas que colaboran para cumplir con partes individuales [9]. Como MLOps es un fenómeno relativamente nuevo, no hay suficiente trabajo relacionado que pueda guiar la implementación, lo cual hace que la información acerca de instrucciones específicas para iniciar tales proyectos o informes relacionados con la experiencia con dicho enfoque sea más escasa.

Los proyectos de aprendizaje automático se han vuelto cada vez más relevantes para varios casos de uso del mundo real. El éxito de los modelos complejos de AA depende de muchos factores, y las técnicas de MLOps surgen como requisito para la gestión de desarrollo de proyectos estructurados y centrados en el aprendizaje automático, debido a la multitud de herramientas disponibles para las diferentes fases operativas, las responsabilidades y los requisitos, que se vuelven cada vez menos claros [9]. En el paradigma de *Machine Learning Operations* (MLOps), es importante poner el foco en la interconectividad de herramientas específicas, tener en cuenta el rendimiento del modelo de AA, los

datos de entrada y las métricas de calidad [9].

Figura 2.1: Conjunto de herramientas frecuentemente utilizadas en MLOps para la automatización y optimización del ciclo de vida de sistemas de Machine Learning.

Además, el panorama de las herramientas de MLOps ha evolucionado enormemente en los últimos años. Ha habido un surgimiento de soluciones de software de alta calidad en términos de opciones comerciales y de código abierto. Las plataformas y herramientas comerciales disponibles en el panorama de MLOps hacen que el desarrollo de sistemas de AA sea más manejable. Un ejemplo de ello es el marco AWS MLOps [9]. El marco se basa en dos componentes principales, es decir, primero, el orquestador y segundo, la instancia de AWS CodePipeline. Es un marco extensible que puede inicializar un pipeline pre-configurado a través de una simple llamada a la API. Los usuarios reciben una notificación por correo electrónico sobre el estado del flujo de Aprendizaje Automático (AA). Hay ciertas desventajas de usar plataformas comerciales para MLOps. El proceso de desarrollo requiere múltiples iteraciones, y es posible que termine gastando mucho dinero en una solución que no sirva [9]. Muchas ejecuciones de entrenamiento pueden no producir resultados sustanciales. Para tener un flujo de trabajo flexible y evolutivo, es fundamental tener un flujo de trabajo 100 % transparente, y con las soluciones comerciales, esto no se puede asegurar por completo. En general, las herramientas de código abierto son más modulares y, a menudo, ofrecen mayor calidad que sus contrapartes [9].

La creación de un flujo de trabajo de aprendizaje automático automatizado para la detección de irregularidades es de gran interés para muchas organizaciones, lo cual determina que un flujo de trabajo aplicando MLOps para la detección de irregularidades sea todavía de mayor interés [9].

2.2. Definiciones referentes a MLOps

El uso de MLOps permite atacar algunos problemas existentes en la industria del AA. Uno de los problemas que pretende atacar es la capacidad de reproducir la creación de modelos conocidos, es decir, seguir su trazabilidad, tomar versiones instantáneas y poder manejar los recursos necesarios para volver a obtener el modelo nuevamente. Este tipo de reproducibilidad permite mayor colaboración y capacidad de compartir el *pipeline* entre los equipos de desarrollo. También el uso de MLOps permite tener una mayor auditoría sobre el proceso y capacidad de explicar el proceso de elaboración, permitiendo mantener la integridad de los recursos. Los modelos son más portables para diferentes plataformas y cuentan con validación funcional, cubriendo con estándares de calidad definidos previamente y que son controlados a lo largo del pipeline. El uso de MLOps ayuda al despliegue de los modelos en producción, asegurando confiabilidad desde su elaboración hasta su utilización en producción [11].

Una gran cantidad de proyectos de ML fallan, quedando solo en pruebas de concepto que nunca llegan a producción. Esto en gran parte se puede explicar dado que en la mayoría de los proyectos, la comunidad de ML se ha enfocado mucho más en la construcción de modelos, que en la preparación de estos productos para salir a producción. Al día de hoy, en muchas aplicaciones industriales, los workflows de ML son manejados en gran medida de manera manual. MLOps se define en en primera instancia como una guía para mostrar como las cosas deberían hacerse[4], relacionado con las buenas prácticas según profesionales del sector. Se identifican diez **principios**, los cuales son:

1. **Integración Continua (Continuous Integration, CI):** La Integración Continua es un principio fundamental dentro de las prácticas de MLOps, que consiste en la automatización de la integración de código a un repositorio compartido. Cada integración se verifica mediante la ejecución automática de pruebas, lo que permite identificar y corregir errores con rapidez, mejorar la calidad del software y reducir el tiempo de validación y lanzamiento de nuevas funcionalidades. En el contexto de MLOps, la CI se extiende para incluir no solo el código de la aplicación, sino también el código de los modelos de aprendizaje automático, los scripts de preprocesamiento de datos y las pruebas asociadas. Esto asegura que cualquier cambio en el sistema, ya sea en el código de la aplicación o en los modelos de ML, se integre de manera fluida y coherente, manteniendo la estabilidad y la confiabilidad del sistema en producción.

2. **Automatización de CI/CD**: La automatización de Integración Continua y Despliegue/Entrega Continua (CI/CD) abarca las etapas de construcción, prueba, entrega y despliegue. También proporciona retroalimentación rápida a los desarrolladores sobre el éxito o fracaso de ciertos pasos, aumentando así la productividad general. CI/CD pone en práctica los principios de DevOps y, por lo tanto, se puede considerar como una táctica de DevOps.

3. **Orquestación del *Flujo de trabajo* (Workflow)**: La orquestación de flujos de trabajo coordina las tareas de un pipeline de flujo de trabajo de AA mediante grafos DAGs. Los DAG definen el orden de ejecución de las tareas al considerar relaciones y dependencias entre ellas.

4. **Reproducibilidad**: Se refiere a la capacidad de repetir un experimento de Aprendizaje Automático y obtener exactamente los mismos resultados.

5. **Versionado**: Asegura el versionado de los datos, modelos y código para que no solo haya Reproducibilidad, sino también trazabilidad.

6. **Colaboración**: Asegura la posibilidad de trabajar de manera colaborativa con datos, modelos y código. La meta es fomentar la colaboración y comunicación entre los diversos participantes, de manera de evitar el aislamiento de información.

7. **Entrenamiento y evaluación continua de ML**: Implica la práctica de reentrenar periódicamente un modelo de ML utilizando nuevos datos. Este proceso continuo se apoya en una componente de monitoreo, un loop de retroalimentación y un pipeline automatizado de ML. El entrenamiento continuo siempre incluye una evaluación para medir los cambios en la calidad del modelo.

8. **Tracking/Logging de metadata**: Se rastrea y registran metadatos para cada tarea del Workflow que se encuentre orquestada. Es necesario realizar el seguimiento y registro de estos metadatos para cada iteración, registrando, por ejemplo, fecha y hora, parámetros utilizados, métricas de rendimiento resultantes, etc. También se registra la "línea de tiempo" del modelo, es decir, los datos y código utilizados, para garantizar la trazabilidad.

9. **Monitorización continua**: Implica la evaluación de manera periódica de: datos, modelos, código, rendimiento del modelo, entre otros, para detectar errores potenciales o cambios que puedan influir en la calidad del producto. La idea es garantizar que ante algún problema, esto se detecte de manera rápida y poder actuar

10. **Loop de retroalimentación (Feedback Loop)**: Refiere a la necesidad de tener múltiples iteraciones entre distintos pasos para retroalimentar unos a otros. Por ejemplo, un feedback loop desde un componente de monitoreo (observando, por ejemplo, el rendimiento de un modelo) hacia el scheduler para realizar un reentrenamiento del mismo. La idea es que los conocimientos obtenidos en distintos pasos, se incorporen al resto para mejorar continuamente.

Además de los principios, se introduce un concepto más que es el de las *Componentes Técnicas*, que relacionan todos estos principios previamente mencionados, agrupándolos según las características de cada principio.

1. **Componente de CI/CD**: Relacionado con los principios 1, 6 y 9. Este componente gestiona las etapas de build, pruebas, entrega y despliegue, brindando un feedback rápido al desarrollador sobre el estado de cada paso. Algunos ejemplos de herramientas relacionadas con esta componente son *Jenkins* [12] o *GitHub actions*.

2. **Repositorio de código fuente**: Relacionado con los principios 4 y 5. El código es versionado en un repositorio. Permite que varios desarrolladores puedan colaborar fácilmente. Ejemplos de herramientas relacionadas son *GitHub* [13] o *GitLab* [14].

3. **Componente de Orquestación del Workflow**: Relacionado con los principios 2, 3 y 6. Esta componente proporciona la orquestación de tareas del Workflow por medio de DAG. Con estos grafos se representa el orden de ejecución y el uso de distintos artefactos de cada paso particular del Workflow. Ejemplos de herramientas relacionadas son *Apache Airflow* [15] o *Kubeflow* [16].

4. **Sistema de *Feature Store***: Relacionado con los principios 3 y 4. Este sistema asegura un almacenamiento centralizado de distintas features que son utilizadas en común. Típicamente, cuenta con dos bases de datos: Una con una latencia normal que sirve para experimentación de manera offline, y otra con baja latencia, que se usa para predicciones productivas. Algunas herramientas relacionadas son *Google Feast* o *Amazon AWS Feature Store* [17].

5. **Infraestructura de Model Training**: Relacionado con el principio 7. Recursos computacionales, ya sean distribuidos o no distribuidos, locales (no escalable) o en la nube (escalable). Herramientas como *Kubernetes* [18] están relacionadas.

6. **Model Registry**: Relacionado con los principios 3 y 4. El Model Registry almacena de manera centralizada los modelos entrenados junto con su metadata. Tiene dos funcionalidades principales, justamente: Almacenar los artefactos de los modelos, y almacenar la metadata de los modelos. Ejemplos de herramientas avanzadas son *MLflow* [19] y *AWS SageMaker Model Registry* [20].

7. **Almacenamiento de Metadatos**: Relacionado con los principios 4 y 7. Permite realizar el tracking de distintos tipos de metadata, por ejemplo, para cada paso de un Workflow de ML. También se puede registrar dentro del Model Registry la metadata de un entrenamiento, por ejemplo, fecha y tiempo de ejecución, junto con los parámetros utilizados para entrenar. Como herramientas relacionadas se destaca a *MLflow* que provee un almacenamiento de metadata en combinación con el Model Registry.

8. **Componente de Servido de modelo**: Relacionado con el principio 1. Puede ser configurado con distintas finalidades. Los casos de uso más

comunes son realizar inferencia en tiempo real (típicamente servido por medio de un web service RESTFul [21]) o en batch (utilizando un Workflow de MapReduce [22], por ejemplo).

9. **Componente de monitoreo**: Relacionado con los principios 8 y 9. Se encarga del monitoreo continuo de métricas de los modelos, como por ejemplo, la precisión de una predicción, entre otras.

En la figura 2.2 se puede ver un diagrama relacionando cada *Principio* dentro de cada *Componente Técnica*

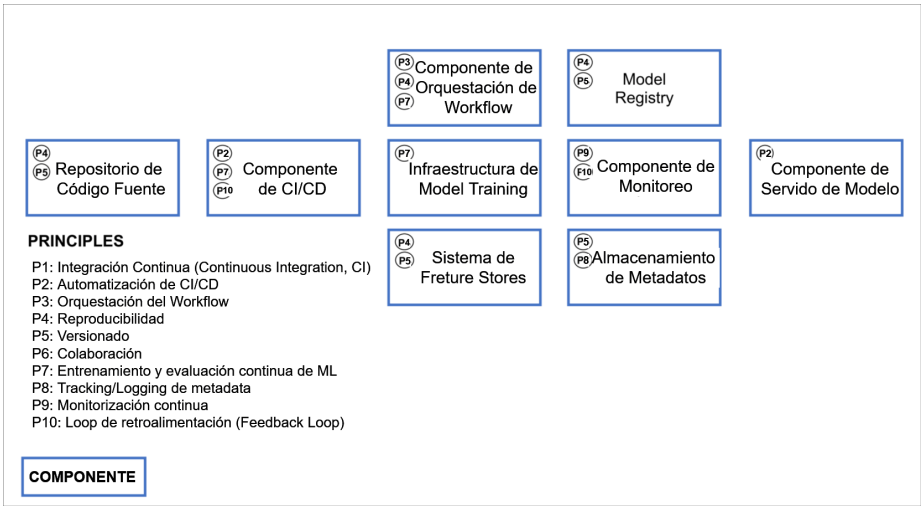


Figura 2.2: Implementación de los Principios dentro de las Componentes Técnicas - Imagen tomada de [4]

2.3. MLOps según Google Cloud

En lo que sigue de esta sección, se ha tomado como referencia el artículo *"MLOPS: Continuous delivery and automation pipelines in machine learning, cloud architecture center, Google Cloud"* [23].

La implementación exitosa de la CI, la CD en producción y el CT de los modelos para sistemas de aprendizaje automático puede beneficiarse del enfoque de MLOps. Esta metodología se vuelve aún más relevante cuando el principal desafío no radica simplemente en desarrollar un modelo de AA que tenga métricas de desempeño favorables, sino en crear un sistema de aprendizaje automático completamente integrado y mantenerlo operativo de manera continua y confiable en un entorno de producción.

Antes de seguir, definamos brevemente de que hablamos cuando nos referimos a CI, CD y CT:

- CI: Implica integrar cada cambio realizado a la rama principal del proyecto de manera rápida y automatizada. Con cada cambio se corren validaciones automáticas (por ejemplo: chequeo de vulnerabilidades en dependencias, estilo de código, análisis estático, etc.), y se integran los cambios, con la finalidad de evitar errores lo más temprano posible. Se garantiza que el código que se agregue cuente con los tests necesarios.

- CD: Va un paso más allá de la CI. Implica la provisión de la infraestructura necesaria para desplegar el código que haya pasado exitosamente por CI y ponerlo disponible en producción sin necesidad de intervención humana.

- CT: Refiere a la práctica de realizar pruebas de manera continua a lo largo del ciclo de vida del desarrollo de software. Incluye varios tipos de pruebas: pruebas unitarias, pruebas de integración, pruebas de aceptación del usuario, entre otras.

Los sistemas de AA integrados constan de diversas etapas y componentes, donde una pequeña fracción de código está dedicada a los modelos predictivos. La mayor parte del sistema se enfoca en otros aspectos críticos, tales como la obtención de datos, verificación de datos, ingeniería de características, análisis de modelos, manejo de metadatos, infraestructura de servicio, subsistema de monitoreo y la depuración del código.

La metodología MLOps proporciona un enfoque sistemático y automatizado para gestionar todas estas fases, lo que permite mejorar la eficiencia, la trazabilidad y la fiabilidad del sistema de AA a lo largo de su ciclo de vida.

Las prácticas e ideas de Desarrollo y Operaciones (DevOps) son populares para el desarrollo y la operacionalización de sistemas de software profesional e industrial. Entre los beneficios de aplicar dichas ideas se puede destacar la reducción de los ciclos de desarrollo, mayor velocidad de implementación y la disponibilización de productos actualizados y confiables. El paralelismo existente entre MLOps y DevOps se debe a que ambos requieren la CI, realización de pruebas unitarias y CD de nuevas versiones.

Sin embargo, la MLOps también requiere que además del código fuente, se necesite probar y validar los datos y sus esquemas asociados. La liberación de nuevas versiones debe tener en cuenta que el flujo de AA se mantenga correcto, como también el servicio de inferencia o predicción.

2.3.1. Flujo de trabajo en la ciencia de datos

En el área del AA, se siguen una serie de pasos destinados a elaborar modelos predictivos que ayuden en la toma de decisiones. A continuación se describen las etapas de dicho flujo:

1. Lo primero es la **extracción de los datos**, lo que implica obtenerlo de las diversas fuentes, como pueden ser bases de datos relacionales o no relacionales, archivos de texto, entre otras.

2. Al tener los datos debemos realizar un **Análisis Exploratorio de los Datos (EDA)**, donde se busca comprender el esquema de datos que se está manejando y ver si es necesario aplicar alguna ingeniería de datos que permita potenciar las características.

3. La etapa de **preprocesamiento**, donde los datos son preparados con el formato correcto. De esta forma pueden ser utilizados por las librerías de AA. Entre las actividades pertenecientes a la etapa, está la limpieza de los datos, la transformación de los datos y la división en conjuntos para entrenamiento, validación y *testing*.

4. Luego se realiza el **entrenamiento** de los modelos de AA. En esta etapa se prueban diferentes algoritmos, con potencialmente diferentes características y formatos de datos de entrada. También se incluye la búsqueda de hiperparámetros con miras de obtener el modelo entrenado de mejor rendimiento.

5. El modelo obtenido en la fase anterior es sometido a **pruebas de evaluación** utilizando el subconjunto de *testing* para ver si sus métricas de rendimiento son apropiadas.

6. Luego el modelo es **validado** y utilizado en producción, solamente si supera cierto umbral de rendimiento esperado, conocido como línea base de rendimiento.

7. En este paso se brinda el **servicio del modelo** para que sus predicciones puedan ser utilizadas en producción.

8. Por último, se realiza el **monitoreo** con el fin de controlar el rendimiento a nivel de producción.

El nivel de automatización de los pasos comentados anteriormente define la madurez del proceso de AA, lo que refleja la velocidad de entrenar nuevos modelos dados nuevos datos, o crear nuevos modelos para nuevos datos. En las próximas secciones se describen 3 niveles de automatización [1], empezando por el menor nivel de automatización, hasta el más automático.

2.3.2. MLOps nivel 0: Proceso manual

Los equipos de AA tienen mano de obra capacitada para elaborar modelos en el estado del arte, pero el proceso que se lleva a cabo para elaborar y reentrenar dichos modelos es manual. Esto implica que el nivel de madurez del proceso de AA es el más básico (nivel 0 en esta escala).

[1] Estos niveles de automatización fueron presentados en el manual de Google Cloud [23] y permiten tener una referencia acerca del grado de automatización que se tiene.

Características

En el nivel 0 de MLOps, se observa un enfoque manual y falta de automatización en los procesos de implementación de sistemas de AA en producción. Cada paso del flujo de trabajo, desde la obtención de datos hasta el despliegue del modelo, se realiza manualmente, lo que, generalmente, puede causar demoras y aumentar los errores humanos. La ausencia de CI/CD implica que los cambios en código, modelos o funcionalidades no se integran automáticamente, resultando en procesos manuales de prueba. Esta falta de automatización genera una desconexión entre los equipos de ciencia de datos y de ingeniería, lo que puede causar problemas en la producción.

En este nivel, las actualizaciones de modelos son ocasionales y la falta de automatización dificulta la gestión de versiones y aumenta el riesgo de errores. Además, la falta de monitoreo del rendimiento de los modelos en producción impide la detección temprana de problemas o cambios en el comportamiento del modelo, afectando la calidad y confiabilidad del sistema de AA. En resumen, el nivel 0 de MLOps presenta limitaciones en eficiencia, agilidad en el desarrollo, coordinación entre equipos y monitoreo del rendimiento.

Desafíos

El nivel 0 de MLOps es común en empresas que están dando sus primeros pasos en el uso del AA en su estrategia de negocio. En este nivel, el proceso es principalmente manual, basado en análisis científicos y experimentación iterativa. Aunque puede ser útil cuando se modifican o entrenan modelos de forma poco frecuente, este enfoque presenta desafíos cuando se busca implementar modelos en producción de manera confiable.

La falta de automatización y la baja frecuencia de actualización de modelos pueden provocar que los modelos no se adapten adecuadamente a los cambios en el entorno o a los datos que describen el problema en cuestión. Esto puede llevar a una degradación del rendimiento del modelo y a la falta de precisión en las predicciones a medida que cambian las circunstancias.

Para abordar estos desafíos y mejorar el funcionamiento de los modelos en producción, se sugiere llevar a cabo una supervisión activa de la calidad del modelo en producción. Implementar un monitoreo constante del rendimiento del modelo permitirá detectar rápidamente cualquier degradación en el desempeño y tomar acciones correctivas de manera oportuna.

Otra recomendación es reentrenar los modelos con la mayor frecuencia posible utilizando los datos más recientes. De esta manera, los modelos pueden capturar patrones emergentes y adaptarse a los cambios en los datos y el entorno, manteniendo su relevancia y precisión.

Asimismo, es importante llevar a cabo una experimentación continua para implementar mejoras en los modelos. Incorporar nuevas técnicas, arquitecturas e hiperparámetros mediante la experimentación permitirá aprovechar las últimas tendencias y avances en tecnología para mejorar el rendimiento del modelo.

Para enfrentar estos desafíos y avanzar hacia un enfoque más robusto y au-

tomatizado, se recomienda adoptar prácticas de MLOps, como la CI, la CD y el CT de modelos de AA. Mediante la implementación de flujos de trabajo o pipelines de AA, se puede lograr una mayor eficiencia en el desarrollo, entrenamiento y despliegue de modelos, así como una mayor confiabilidad y precisión en el funcionamiento de los mismos.

En las próximas secciones, se analizarán en mayor detalle las funciones y beneficios de las prácticas de MLOps en el contexto de CI, CD y CT, para lograr una gestión más efectiva de los modelos de AA en producción.

2.3.3. MLOps nivel 1: Automatización del flujo de AA

El nivel 1 tiene como objetivo realizar un entrenamiento continuo del modelo mediante la automatización del pipeline de AA, esto permite lograr el despliegue continuo del servicio de predicción del modelo. Para automatizar el proceso de uso de datos actualizados con la finalidad de volver a entrenar los modelos que están siendo usando a nivel operativo, se deben ingresar los datos de forma automatizada, seguir los pasos de validación de modelos en el pipeline, los activadores del flujo de trabajo y brindar la administración de metadatos.

Características

En el nivel 1 de MLOps, se centra en lograr una mayor automatización y eficiencia en el desarrollo, entrenamiento y despliegue de modelos de AA. Este nivel, adoptado por empresas que buscan mejorar la gestión de sus modelos, se caracteriza por flujos automatizados que agilizan la experimentación y preparan el flujo de trabajo para una transición rápida hacia la producción.

La automatización se extiende al entrenamiento de modelos en producción, asegurando que se adapten automáticamente a datos actualizados, mejorando su capacidad para enfrentar cambios en el entorno. La simetría entre experimentación y producción es clave, utilizando la misma implementación del pipeline de AA en ambos entornos para una gestión coherente.

La modularización del código es esencial en este nivel, con componentes y flujos de trabajo de AA que son reutilizables y compatibles. Se enfatiza la modularización del código fuente para aislar componentes y garantizar la reproducibilidad en distintos entornos. Además, destaca el despliegue continuo de modelos, entregando servicios de predicción de manera constante mediante procesos automáticos para asegurar la disponibilidad de los modelos más recientes en producción.

Componentes Adicionales para el Entrenamiento Continuo en AA: Validación de Datos, Modelos y Feature Store

En este apartado se abordan algunos componentes que deben añadirse a la arquitectura para permitir el entrenamiento continuo de Aprendizaje Automático (AA).

La validación de datos y modelos desempeña un papel fundamental al implementar el flujo de AA en producción. Para garantizar el comportamiento esperado, es esencial contar con pasos automatizados de validación tanto para los datos como para los modelos.

La validación de datos se lleva a cabo antes del entrenamiento de los modelos, evaluando si es necesario reentrenar el modelo o interrumpir el flujo de AA. Este proceso se activa automáticamente al detectar variaciones en el esquema de datos o valores inesperados, indicando cambios significativos en las propiedades estadísticas de los datos y la necesidad de reentrenamiento.

La validación de modelos ocurre después del entrenamiento, evaluando el rendimiento del modelo antes de su implementación en producción. Se comparan métricas de evaluación del modelo recién entrenado con el modelo en producción y se verifica su compatibilidad con la infraestructura y coherencia con la aplicación de servicio de predicción. Además, se realiza una validación en línea utilizando implementaciones de versiones tipo *Canary* (se prueba la nueva versión para un pequeño porcentaje del tráfico) o *pruebas A/B* antes de utilizar las predicciones en línea.

El *almacenamiento de características* (o Feature Store) también es esencial, siendo un componente adicional para la automatización del flujo en el Nivel 1 de MLOps. El Feature Store es una base de datos centralizada que estandariza la definición, almacenamiento y acceso a las características utilizadas en el entrenamiento e inferencia. Almacenando características y metadatos relacionados, ayuda a minimizar las variaciones entre el entrenamiento y la inferencia, garantizando coherencia en el uso de características durante ambas fases del modelo.

Administración de metadatos

El registro de información en cada ejecución del flujo de trabajo de AA proporciona trazabilidad, reproducibilidad y comparación de resultados. Estos metadatos incluyen versiones del flujo de trabajo, detalles de inicio y finalización, duración de pasos, responsables, parámetros, y ubicación de artefactos como datos preparados, anomalías detectadas y estadísticas calculadas. Este registro facilita la depuración, la reanudación desde un paso específico en caso de interrupción y la reversión a versiones anteriores del modelo.

El seguimiento del modelo previamente entrenado es esencial, permitiendo revertir a versiones anteriores o evaluar métricas de versiones anteriores con nuevos datos. Las métricas de evaluación del modelo para conjuntos de entrenamiento y prueba también se registran, facilitando la comparación del rendimiento del modelo actual con el anterior.

El pipeline de AA puede ser activado para reentrenar modelos bajo ciertas circunstancias, como cambios en datos, deterioro del rendimiento o desviación de concepto (concept drift). La gestión de estos pipelines puede presentar desafíos, y en casos simples, una evaluación manual puede ser suficiente. Sin embargo, para entornos más complejos con múltiples grafos de AA, se requiere un sistema de integración continua y despliegue continuo para automatizar compilación, pruebas y despliegue, manteniendo la eficiencia y eficacia del proceso de AA.

2.3.4. MLOps nivel 2: CI/CD del pipeline de AA

Para obtener una actualización rápida y confiable de los pipelines en producción, es necesario un sistema de CI/CD consistente y automatizado. Este sistema automatizado de CI/CD permite que se pueda explorar con rapidez nuevas estrategias acerca de la ingeniería de atributos, la arquitectura de modelos y la exploración de hiperparámetros. Lo cual permite aplicar estas estrategias, compilar, evaluar y luego, implementar de forma automática los nuevos componentes del pipeline en el entorno de destino.

Flujo de trabajo del pipeline de AA con integración de CI/CD.

Esta configuración de MLOps incluye los siguientes componentes:

- Control del código fuente.
- Servicios de compilación y prueba.
- Servicios de implementación.
- Registro de modelos.
- Almacenamiento de características.
- Almacenamiento de metadatos de AA.
- Orquestador de pipelines de AA.

Características

El pipeline consta de las siguientes etapas:

- Desarrollo y experimentación: las pruebas son hechas de manera iterativa a los algoritmos de AA y los nuevos modelos, donde, los pasos del experimento están orquestados. El resultado de esta etapa es el código fuente de los pasos del pipeline de AA, que luego se envía a un repositorio de código.
- Integración continua del pipeline: implica la compilación del código fuente y ejecución de varias pruebas de forma automatizada. Los resultados de esta etapa son componentes del pipeline (paquetes, ejecutables y artefactos) que se desplegarán en una etapa posterior.
- CD del pipeline: se despliegan los artefactos producidos durante la etapa de la CI en el entorno objetivo. El resultado de esta etapa es el despliegue del pipeline actualizado con el nuevo modelo.
- Activación automática: el pipeline se ejecuta de forma automática en producción según una programación planificada o en respuesta a un activador. El resultado de esta etapa es un modelo entrenado que se envía al registro de modelos.

- Despliegue continuo de modelos: implica el despliegue del modelo entrenado como un servicio de predicción. El resultado de esta etapa es un servicio de predicción actualizado.
- Monitoreo: recopila estadísticas sobre el rendimiento del modelo en función de los datos en tiempo real. El resultado de esta etapa es un activador para ejecutar nuevamente el pipeline o un ciclo experimental.

El paso de análisis de datos sigue siendo un proceso manual antes de que el pipeline comience una iteración nueva del experimento. El paso de análisis del modelo también es un proceso manual.

Integración continua

En esta configuración, el pipeline y sus componentes se compilan, evalúan y empaquetan cuando se confirma un código nuevo o se envía al repositorio. Además de compilar paquetes, imágenes de contenedores y ejecutables, el proceso de CI puede incluir las siguientes pruebas:

- Realizar pruebas unitarias de la lógica de ingeniería de características.
- Realizar pruebas unitarias de los diferentes métodos implementados en el modelo.
- Evaluar si hay una convergencia en el entrenamiento de modelos (es decir, la función de pérdida del modelo se reduce con las iteraciones y sobreajusta algunos registros de la muestra).
- Evaluar si el entrenamiento de modelos no produce valores erróneos debido a la división entre cero o a la manipulación de valores pequeños o grandes [23].
- Evaluar que cada componente del pipeline produzca los artefactos esperados.
- Evaluar la integración entre componentes del pipeline.

Despliegue continuo

En este nivel de implementación de MLOps, se centra en la entrega continua de nuevas implementaciones del pipeline [2] en el entorno de producción, proporcionando servicios de predicción con modelos recién entrenados. Para garantizar una entrega continua rápida y confiable, se debe considerar lo siguiente:

1. **Compatibilidad del Modelo con la Infraestructura**: Verificar que el modelo sea compatible con la infraestructura de destino, asegurándose de que los paquetes necesarios estén instalados y que los recursos requeridos estén disponibles.

[2] El pipeline puede evolucionar y volverse más complejo con el tiempo.

2. **Pruebas de la API de Servicio**: Llamar a la API de servicio con datos de entrada esperados para evaluar el servicio de predicción y confirmar que la respuesta sea la esperada. Por lo general, esta prueba captura problemas que pueden ocurrir cuando se actualiza la versión del modelo y el nuevo espera una entrada diferente.

3. **Evaluación del Rendimiento del Servicio**: Realizar **pruebas de carga** para evaluar métricas como consultas por segundo (QPS) y la latencia del modelo, garantizando un rendimiento adecuado.

4. **Validación de Datos**: Validar los datos para reentrenamiento o predicción por lotes, asegurando la calidad y relevancia de los datos.

5. **Cumplimiento de Objetivos de Rendimiento**: Verificar que los modelos cumplan con los objetivos de rendimiento predictivo antes de la implementación.

6. **Implementación Automatizada en Entornos de Prueba y Preproducción**: Realizar implementaciones automatizadas en entornos de prueba y preproducción, utilizando sistemas de CI/CD para probar y establecer implementaciones de pipeline nuevas de manera automática.

7. **Implementación Manual en Entorno de Producción**: Realizar implementaciones manuales en el entorno de producción después de varias ejecuciones satisfactorias del pipeline en entornos previos.

En resumen, la implementación de AA en un entorno de producción va más allá de implementar su modelo como una API para la predicción. En realidad, significa implementar un pipeline de AA que pueda automatizar el reentrenamiento y la implementación de modelos nuevos. La configuración de un sistema de CI/CD permite probar y establecer implementaciones de pipelines de forma automática, abordando cambios rápidos en los datos y el entorno empresarial. Estas prácticas pueden implementarse gradualmente para mejorar la automatización en el desarrollo y producción del sistema de AA.

Para adentrarnos aún más en estas consideraciones prácticas y entender cómo se están abordando en el mundo real, consultamos con un experto en herramientas de Machine Learning. Su experiencia nos ofreció una visión invaluable sobre los desafíos y oportunidades que se presentan en la industria actual.

2.4. Perspectivas de la Industria: Entrevista con un referente de un E-commerce de la Región Sobre Soluciones de Machine Learning

Para complementar nuestra comprensión del estado del arte en MLOps, se realizó una entrevista con un experto en herramientas de Machine Learning de

Mercado Libre. Esta sección tiene como objetivo proporcionar una breve visión práctica y aplicada del campo, desde la perspectiva de alguien profundamente involucrado en la industria.

En el marco del proyecto, se identificó la necesidad de tener una visión más completa y actualizada del estado del arte en cuanto a las prácticas de MLOps en la industria. Esto se debe a que la academia y la industria a veces operan en paralelo, pero con diferentes conjuntos de desafíos y soluciones. Fue por esto que surgió la idea de ponernos en contacto con un experto en una herramienta de machine learning de una importante empresa de comercio electrónico de la región.

Tras coordinar una reunión, pudimos conversar sobre la estructura y las operaciones de su departamento de ciencia de datos. Nos llamó particularmente la atención la similitud entre las metodologías que estábamos investigando en el ámbito académico y las que ya estaban siendo aplicadas en un entorno de producción en esta empresa. Esto fortaleció la premisa de que las soluciones y enfoques que estábamos estudiando tenían un valor práctico real.

Una de las principales revelaciones fue la escala de su operación, ya que cuentan con muchos equipos que trabajan la ciencia de datos con distintos fines, contando varios de ellos con muchos integrantes de distintos perfiles, lo cual resalta la complejidad y el alcance del trabajo que se está realizando en entornos industriales. Además, esta empresa no solo aplica técnicas de MLOps, sino que también está en el proceso de desarrollar su propia herramienta de MLOps para automatizar todo el ciclo de vida del aprendizaje automático.

Este experto destacó que, incluso en la actualidad, los módulos pertenecientes al pipeline de MLOps suelen desarrollarse de forma manual y personalizada. Este comentario refleja el nivel de madurez que aún está alcanzando el campo de MLOps, lo que subraya la importancia de adaptar y personalizar soluciones según las necesidades específicas de cada proyecto

Se hizo especial énfasis en la importancia de desarrollar de forma modular y asegurarse de que estos módulos sean idempotentes, lo cual facilita enormemente la gestión y el mantenimiento de los sistemas. Esto es crítico, especialmente cuando se trabaja con miles de características para la detección de fraudes económicos, una tarea que requiere mucho cuidado.

Este acercamiento entre el mundo académico y el industrial no solo enriqueció nuestra tesis, sino que también destacó la relevancia y aplicabilidad de nuestra investigación en problemas reales, aportando un valor significativo tanto a la academia como a la industria.

Capítulo 3

Herramientas existentes

En los últimos años, han surgido numerosas herramientas diseñadas para automatizar el flujo de trabajo del aprendizaje automático. La selección del *stack* de herramientas para MLOps depende en gran medida del contexto específico del aprendizaje automático al que se desea aplicar [24]. En este capítulo, se revisan tanto los trabajos relacionados a nivel académico como las herramientas ofrecidas por la industria.

Posteriormente, se realiza un análisis de diversas herramientas que pueden integrarse en el flujo de MLOps, teniendo en cuenta las etapas descritas en el capítulo anterior. A partir de los conceptos definidos hasta este punto, se identifican un conjunto de herramientas de MLOps, reconociendo que algunos requerimientos pueden ser satisfechos por varias de ellas simultáneamente. Asimismo, es posible identificar combinaciones de herramientas que cubran un espectro más amplio de los requerimientos del sistema.

3.1. Exploración de herramientas para MLOps

En esta sección, se comparan y describen diferentes herramientas de código abierto para MLOps que vienen emergiendo en la actualidad. Estas herramientas se comparan con principal enfoque en los requisitos y descripciones mencionados acerca de los niveles de MLOps, además de brindar una clasificación de qué tan bien cumplen con esos requisitos. A continuación se brinda una breve descripción de cada una de las herramientas existentes con el fin de identificar los elementos clave en función de los cuales se puede decidir usar la herramienta en particular o descartarla. En las siguientes subsecciones, se discuten las características de cada una de las herramientas, con principal énfasis en la creación de un flujo de trabajo MLOps.

3.1.1. Interacción y Aplicación de Airflow, MLflow y TFX en el Pipeline de Aprendizaje Automático

En el panorama de las herramientas de aprendizaje automático, tres plataformas sobresalen por sus destacadas características y su versatilidad en las distintas etapas de un pipeline de machine learning: Airflow, MLflow y TFX.

3.1.2. Airflow

Airflow [15] es una plataforma de gestión de flujos de trabajo que ofrece funcionalidades para gestionar y programar tareas representadas como DAG. Airflow no se limita a Kubernetes como la herramienta Kubeflow [25]; sino que está diseñado para integrarse en el ecosistema de Python [26]. No es intuitivo por diseño y tiene una curva de aprendizaje empinada para nuevos usuarios. Se pueden definir tareas en Python y orquestar un flujo de trabajo simple. También es difícil integrar Airflow para proyectos AA que ya están en desarrollo [27].

3.1.3. MLflow

MLflow [19] es una plataforma de aprendizaje automático que se puede utilizar para manejar el ciclo de vida de AA, permitiendo la automatización parcial de los proyectos. Esta plataforma cuenta con cuatro herramientas o componentes principales, donde cada una de ellas se enfoca en una fase específica del ciclo de vida de los proyectos de AA [28][29], y una que nos facilita la creación de pipelines.

A continuación se hace una breve descripción de las herramientas:

- ***MLflow Tracking***[30]: permite rastrear y registrar los experimentos. Primero, almacena los parámetros, las métricas, los archivos de salida al ejecutar el código, fecha y hora de inicio y fin del experimento, etc. Luego es posible visualizar los resultados de todos los experimentos en un servicio de publicación local en formato web.
- ***MLflow Projects***[31]: es útil para empaquetar código de Ciencia de Datos (DS) de forma reutilizable y reproducible.
- ***MLflow Models***[32]: se usa para empaquetar modelos de aprendizaje automático. Existen varios métodos para guardar y cargar modelos de MLflow.
- ***MLflow Model Registry***[33]: es una herramienta diseñada para gestionar de manera eficiente el ciclo de vida de los modelos de machine learning, desde su desarrollo hasta su implementación en producción. Su función principal es ofrecer un sistema centralizado para registrar, versionar y organizar los modelos, proporcionando una cronología detallada de cada modelo a medida que evoluciona a lo largo del tiempo. Además, permite agregar descripciones y anotaciones a cada versión del modelo, facilitando el rastreo y la trazabilidad de los cambios realizados. Esta herramienta

resuelve una dificultad clave en MLOps: la gestión de múltiples versiones de modelos y su despliegue ordenado, asegurando que los equipos puedan colaborar y mantener control sobre el estado y la calidad de los modelos a lo largo de todo su ciclo de vida. Asimismo, permite recuperar fácilmente versiones anteriores de los modelos para analizar su desempeño pasado y realizar comparaciones, garantizando que se pueda revertir a modelos previamente efectivos si es necesario.

- ***MLflow Recipes***[34]: provee templates de proyectos con una estructura predefinida que permiten acelerar la implementación de soluciones de problemas típicos de regresión o clasificación binaria.

3.1.4. TensorFlow

TensorFlow (TF)[35] permite a los desarrolladores crear grafos de flujo de datos, estructuras que describen cómo se mueven los datos a través del grafo o una serie de nodos de procesamiento. Cada nodo en el grafo representa una operación matemática, y cada arista entre nodos es una matriz de datos multidimensional o tensor. Además, el mayor beneficio que ofrece *TensorFlow* para el desarrollo de AA es la abstracción. En lugar de ocuparse de los detalles medulares de la implementación de algoritmos, o de descubrir formas adecuadas de conectar la salida de una función a la entrada de otra, el desarrollador puede concentrarse en la lógica general de la aplicación [36].

3.1.5. TensorFlow Extended

TensorFlow Extended (TFX)[37] es una plataforma integral para implementar el *pipeline* de aprendizaje automático. El *pipeline* se forma como una secuencia de componentes que implementan el pipeline de AA. Para beneficiarse de TFX, es imprescindible usar las bibliotecas del ecosistema de *TensorFlow* (TF) [36] para construir los distintos componentes del *pipeline*, el cual se puede orquestar mediante Apache Beam [38], Apache Airflow o Kubeflow [25].

El costo de integración utilizando TFX tiende a ser elevado debido a varios factores clave: la necesidad de una infraestructura robusta para manejar grandes volúmenes de datos, la complejidad de la implementación de pipelines personalizados, y la inversión en recursos de desarrollo y capacitación del equipo para aprovechar al máximo las funcionalidades avanzadas de la plataforma. Por estas razones, TFX es una opción altamente eficiente para proyectos de gran escala con necesidades de actualización continua de modelos, pero puede no ser la más recomendable para proyectos pequeños o aquellos donde los modelos se actualizan con poca frecuencia, ya que los costos asociados pueden superar los beneficios obtenidos.

Los componentes que brinda TFX para lograr la construcción del *pipeline* son los siguientes:

- **ExampleGen**: es el punto de partida para que ingresen los datos al *pipeline*. Dada una ruta local o también una ruta de almacenamiento en la

nube, recopila archivos en el formato de registros de TensorFlow (TFRecords) de acuerdo con las configuraciones de la entrada. Se puede usar un conjunto de datos pequeño, y leer los datos directamente desde archivo csv. El componente **CsvExampleGen**, especializado en la lectura de archivos csv, apuntará a un archivo csv local.

- **StatisticsGen**: este componente analiza los datos de entrada y genera estadísticas descriptivas que pueden utilizarse para comprender mejor la distribución y características de los datos. Las estadísticas generadas son fundamentales para otros componentes como **SchemaGen** y **ExampleValidator**.

- **TensorFlow Data Validation (TFDV)**: es una biblioteca que ayuda a explorar y validar datos en un pipeline de machine learning. TFDV permite generar estadísticas sobre los datos, detectar anomalías y comparar los datos de entrenamiento con los datos de evaluación para asegurar que los datos utilizados sean consistentes y apropiados para el modelo.

- **TensorFlow Transform (TFT)**: Este componente es esencial para realizar preprocesamientos complejos a los datos. TFT permite aplicar transformaciones a los datos antes de que lleguen al modelo de machine learning, asegurando que las mismas transformaciones se apliquen tanto en el entrenamiento como en la inferencia del modelo.

- **TensorFlow Metadata (TFMD)**: es utilizado por varios componentes del pipeline de TFX para almacenar información estructurada sobre los datos. TFMD ayuda a gestionar y almacenar esquemas y estadísticas sobre los datos, facilitando la validación y el seguimiento de versiones de los datos.

- **SchemaGen**: utilizando las estadísticas generadas por **StatisticsGen**, este componente genera un esquema que define los tipos y restricciones de los datos. El esquema es utilizado en pasos posteriores para validar los datos y detectar posibles problemas.

- **ExampleValidator**: usando el esquema generado por **SchemaGen**, este componente valida los datos de entrada para detectar y reportar anomalías, asegurando que los datos sean consistentes con lo que el modelo espera.

- **Transform**: Este componente no siempre es necesario, pero resulta muy útil cuando se debe realizar un preprocesamiento con un alto costo computacional. Para utilizarlo, es necesario crear una función pura de TensorFlow llamada `preprocessing_fn`. TFX aplicará esta transformación a todos los puntos de datos proporcionados por el componente **ExampleGen**.

  Una función pura es aquella que siempre produce el mismo resultado dado el mismo conjunto de entradas y que no causa efectos secundarios (es decir, no modifica variables externas ni interactúa con el entorno externo).

En este caso, la función `preprocessing_fn` debe usar exclusivamente operaciones de TensorFlow. Esto permite que el paso de preprocesamiento se serialice en un grafo de *TensorFlow* para aprovecharlo en el modelo final. Además, esta función se puede ejecutar en Apache Beam, lo que habilita la paralelización para manejar grandes conjuntos de datos. La función `preprocessing_fn` recibe un objeto similar a un diccionario y debe devolver un diccionario propiamente dicho, que luego se serializará para alimentar el modelo.

- **Trainer**: en este punto se hace el entrenamiento del modelo. Al igual que en el componente **Transform**, se apunta a un archivo de módulo con la función "run_fn" definida en él. La función "run_fn" recibe una serie de parámetros de entrada, que se referenciarán al construir el *pipeline*.

- **TensorFlow Model Analysis (TFMA)**: esta herramienta permite evaluar y analizar modelos de machine learning en producción. TFMA genera métricas detalladas y permite comparar el rendimiento de distintos modelos en diferentes subconjuntos de datos, asegurando que el modelo funcione correctamente y de forma consistente.

- **Pusher**: es el último componente del *pipeline* de TFX. Su propósito es, que dado el modelo resultante de **Trainer**, subirlo a un repositorio de almacenamiento en la nube. Estando en la nube se puede usar como servicio (ya sea por una instancia de TF Serving[1] que escuche al repositorio o por una aplicación web/móvil que use TF Lite[2] [39], [40]) [41]. Sin embargo, no es obligatorio que los modelos estén en la nube. Los modelos pueden almacenarse y servirse localmente (en servidores on-premises) dependiendo de las necesidades y recursos de la organización.

Antes de subir el modelo a la nube, hay una serie de validaciones que se pueden hacer, por ejemplo, evaluar las métricas en el conjunto de prueba o incluso verificar si realmente se ejecuta en un contenedor de TF Serving [41].

3.1.6. DVC

Data Version Control (DVC) [42] es un sistema de control de versiones para proyectos de AA. La herramienta es capaz de rastrear modelos, conjuntos de datos (incluidos conjuntos de datos grandes de 10 a 100 GB frente al límite de 2 GB en GitHub[13]). Es independiente de la nube y el almacenamiento, es decir, los conjuntos de datos se pueden almacenar, acceder y versionar localmente o en alguna plataforma en la nube. Es compatible con Git [43] y puede realizar un seguimiento de los experimentos. Funciona creando el archivo '*.dvc*' en la

[1] TensorFlow Serving es un sistema optimizado para la implementación de modelos en entornos de servidor, permitiendo servir modelos de machine learning como APIs para predicciones en tiempo real.

[2] TensorFlow Lite es una versión optimizada de TensorFlow diseñada para ejecutar modelos de machine learning en dispositivos móviles y embebidos, permitiendo realizar inferencias localmente con baja latencia y sin necesidad de conexión a internet.

raíz del proyecto, donde se almacena la meta-información, como la ubicación de almacenamiento y el formato de los datos. Para los datos no estructurados, los cambios se rastrean como nuevas versiones por sí mismos, lo que requiere una alta capacidad de almacenamiento. Es una herramienta muy liviana que ofrece excelentes funciones con el mínimo esfuerzo necesario para integrarla en cualquier flujo de trabajo de GitFlow[3] [44].

3.1.7. Kubeflow

Kubeflow [25] es una plataforma de aprendizaje automático que se ejecuta sobre Kubernetes y proporciona funcionalidad de punta a punta para ejecutar proyectos de AA. La empresa Google lo creó para el uso interno con el fin de ejecutar tareas de TensorFlow en Kubernetes, y luego lo lanzó como código abierto en 2018. Kubeflow se ha convertido en una de las herramientas esenciales para proyectos de AA en la actualidad, ya que permite la abstracción de la complejidad subyacente de Kubernetes y proporciona una plataforma que permite el despliegue de proyectos de AA. La herramienta permite una ejecución sencilla y una operacionalización más rápida de proyectos de AA. Permite atacar los puntos más débiles en los proyectos de AA [16].

3.1.8. H2O

La plataforma *H2O*[45] es parte de un conjunto de herramientas de análisis de datos, de las cuales no todas son de código abierto. Permite trabajar con algoritmos de aprendizaje automático como *Extreme Gradient Boosting* (XGB) [46] [4] y aprendizaje profundo, entre otros. Se presenta con una interfaz de usuario de tipo web. Aunque se considera una plataforma de análisis, también es vista como una biblioteca debido a las diversas API que ofrece. La herramienta se usa principalmente para ejecutar modelos predefinidos con capacidades de automatización del AA, aunque no proporciona total flexibilidad para integrar modelos personalizados. Está basado en Java y requiere Java 7 o posterior. Además, carece de capacidades completas de gobernanza de modelos, como el acceso centralizado a los modelos en producción, versionado de modelos, creación de documentación, monitoreo de modelos y sus resultados, así como el desarrollo de sistemas de AA en línea con las políticas técnicas establecidas.

H2O funciona sobre la infraestructura de *big data* existente, como servidores Hadoop, Spark o Kubernetes. Puede ingerir datos directamente desde HDFS, Spark, S3, Azure Data Lake, o cualquier otra fuente de datos.

[3] GitFlow es una estrategia de branching para Git que organiza el desarrollo en ramas principales, de desarrollo, de características, de liberación y de correcciones de errores, facilitando un flujo de trabajo estructurado para la gestión de versiones. Más detalles disponibles en la documentación oficial de GitFlow.

[4] XGBoost (XGB) es particularmente importante porque se ha utilizado en UTE para direccionar inspecciones.

3.1.9. Pachyderm

Pachyderm[47] se ejecuta sobre Kubernetes y Docker, facilitando la configuración para la integración continua con imágenes de contenedores. Además de su sistema de control de versiones, que funciona como un sistema de archivos inmutable similar a git, *Pachyderm* también se destaca por su capacidad para gestionar pipelines de datos y realizar un seguimiento detallado del linaje de los datos[5]. Presenta una curva de aprendizaje pronunciada debido a la dependencia del uso de Kubernetes en su versión gratuita. En comparación con DVC, ofrece muchas opciones para la ingeniería de datos, como la creación de un linaje de datos mediante el seguimiento de las fuentes de datos. El panel web no está disponible en su versión gratuita, aunque sí se incluye en la versión comunitaria [48].

3.1.10. Great Expectations

Great Expectations[49] es una herramienta para validar y analizar datos. Se puede conectar y ampliar fácilmente con muchos entornos de trabajo. Además, ofrece soporte de integración para muchas fuentes de datos, incluyendo a MSSQL [50], SQLAlchemy [51] y BigQuery [52]; sin embargo, SQL Oracle [53] no aparece en la lista. El objetivo principal de esta herramienta es evaluar la calidad de los datos a través de pruebas automatizadas, pero se limita solo a datos tabulares.

3.2. Herramientas Complementarias en el Proyecto

En esta sección, describimos brevemente algunas de las herramientas adicionales que forman parte de la infraestructura tecnológica de UTE. La empresa cuenta con una infraestructura que incluye servidores equipados con tecnologías como HBase [54], Hadoop [55] y Apache Spark [56]. Estos servidores se utilizan principalmente para la extracción y el procesamiento de grandes volúmenes de datos.

Esta infraestructura está diseñada para soportar la manipulación de datos a gran escala, lo cual es crucial en proyectos que requieren una ingesta continua y el procesamiento distribuido de grandes conjuntos de datos. En el contexto de este proyecto, aunque estas herramientas no forman parte directamente del pipeline de MLOps, se incluyeron en esta sección porque también fueron parte del entorno utilizado.

[5] Pachyderm no solo se utiliza para el control de versiones; su principal fortaleza radica en la creación y gestión de pipelines de datos, permitiendo automatizar procesos de transformación y análisis de datos mediante la integración de contenedores. Esto lo convierte en una herramienta poderosa para el machine learning y la ingeniería de datos, donde el control del linaje de datos y la reproducibilidad son esenciales.

3.2.1. Uso de WSL en el Desarrollo para MLOps

El *Windows Subsystem for Linux (WSL)* [57] ha sido seleccionado como una herramienta importante en el desarrollo del software de MLOps en este proyecto. La decisión de utilizar WSL sobre otras soluciones como contenedores se basa en la facilidad de programar y ejecutar directamente en Windows, especialmente en el ámbito del desarrollo y operaciones de modelos de *machine learning*.

WSL permite ejecutar un entorno de Linux genuino directamente en Windows, facilitando una integración más fluida de las herramientas de Linux en flujos de trabajo predominantemente basados en Windows. Esta capacidad es particularmente útil en MLOps, donde herramientas y bibliotecas de código abierto en Linux, como TensorFlow y PyTorch [58], desempeñan roles vitales. La integración de WSL elimina la necesidad de gestionar máquinas virtuales o configuraciones de Docker [59] más complejas, proporcionando un acceso más directo a estas herramientas desde el entorno de desarrollo principal [6] [60].

Además, WSL ofrece mejoras significativas en términos de rendimiento comparado con soluciones basadas en virtualización como Docker, especialmente cuando se trata de operaciones de E/S que son críticas en el entrenamiento y despliegue de modelos de *machine learning*. WSL2, con su kernel Linux real, ofrece un sistema de archivos mucho más rápido y una compatibilidad mejorada con las operaciones de Linux, lo que es esencial para ejecutar simulaciones y modelos de ML que requieren alta interacción con el sistema de archivos [61].

Al operar dentro del ecosistema de Windows, WSL consume menos recursos que las máquinas virtuales completas y permite una gestión más efectiva del hardware subyacente. Esto se traduce en una asignación de recursos más eficiente durante las fases intensivas de computación de los proyectos de MLOps, como el entrenamiento de modelos y la validación de algoritmos [62].

3.2.2. Apache Hadoop

Apache Hadoop [55] es una colección de herramientas de software de código abierto que se apoya en el uso de una red de muchas computadoras para resolver problemas que involucran cantidades masivas de datos y computación. Proporciona una estructura de referencia para el almacenamiento distribuido y el procesamiento de grandes volúmenes de datos utilizando el modelo de programación MapReduce. *Hadoop* se diseñó originalmente para agrupaciones de computadoras con hardware básico [63]. Desde entonces, también ha encontrado uso en conjuntos de hardware de gama alta [64] [65]. Todos los módulos en *Hadoop* están diseñados con la suposición fundamental de que las fallas de hardware son ocurrencias comunes y deben ser manejadas automáticamente por el sistema de *Hadoop*.

[6] El entorno de desarrollo incluye Python como lenguaje principal en sistemas operativos Windows y Linux, utilizando librerías populares como Pandas para manipulación de datos y TensorFlow para el desarrollo de modelos de machine learning.

3.2.3. Apache HBase

Apache HBase [54] permite el acceso aleatorio para la lectura/escritura en tiempo real de *Big Data*. Con el objetivo de albergar tablas muy grandes (miles de millones de filas y millones de columnas) sobre grupos de hardware básico. Apache HBase es una base de datos no relacional, versionada, distribuida y de código abierto, basada en Bigtable de Google [66]: un sistema de almacenamiento distribuido para datos estructurados. Así como Bigtable aprovecha el almacenamiento de datos distribuidos proporcionado por el sistema de archivos de Google, Apache HBase proporciona capacidades similares a las de Bigtable además de Hadoop y HDFS [67] (por sus siglas en inglés *Hadoop Distributed File System* [55]).

Las tablas en HBase pueden servir como entrada y salida para los trabajos de MapReduce que se ejecutan en Hadoop, y se puede acceder a ellas a través de la API de Java. A diferencia de las bases de datos relacionales y tradicionales, *HBase* no admite hacer consultas SQL, en cambio, el equivalente está escrito en Java, siendo similar al uso de MapReduce.

En el contexto de este proyecto, *HBase* se podría utilizar para acceder a las tablas que contienen la información de entrada para los algoritmos de Machine Learning (ML) relacionados con DeepDAICE.

3.2.4. Apache Spark

Apache Spark [56] es un motor de computación unificado y un conjunto de bibliotecas para el procesamiento de datos en paralelo en granjas de computadoras. Hasta el momento, *Spark* es el motor de código abierto más activamente desarrollado para esta tarea, lo que lo convierte en una herramienta estándar para cualquier desarrollador o científico de datos interesado en **big data**. *Spark* admite múltiples lenguajes de programación ampliamente utilizados (Python, Java, Scala y R [26] [68] [69] [70]), incluye bibliotecas para diversas tareas que van desde SQL hasta aprendizaje automático, y puede ser ejecutado en sistemas básicos, es decir, desde computadoras portátiles hasta en una granja de miles de servidores. Esto lo convierte en un sistema de fácil despliegue y permite escalar hacia el procesamiento de *big data* [71].

3.2.5. Etapas del MLOps que cubren las herramientas

Con las herramientas mencionadas anteriormente se buscan cubrir en mayor o menor medida las siguientes etapas del proceso de AA:

Etapa de preprocesamiento de datos:

1. **Integración de fuentes de datos (PD-1)**: Es fundamental integrar diferentes fuentes de datos, como bases de datos o archivos.

2. **Preprocesamiento y transformaciones de datos (PD-2)**: Los datos deben ser limpiados y transformados antes de ser utilizados.

3. **Medición de la calidad de los datos (PD-3)**: Validación automática para asegurar la calidad de los datos utilizados.
4. **Gestión de datos (PD-4)**: Administración centralizada de los datos para identificar errores o problemas.
5. **Versionado de datos (PD-5)**: Asegura la trazabilidad y reproducibilidad al guardar versiones de los datos.
6. **Etiquetado de datos (PD-6)**: Etiquetar datos correctamente para proyectos de aprendizaje supervisado.

Etapa de entrenamiento de modelos:

1. **Selección del tipo de modelo (EM-1)**: Permitir la selección entre distintos tipos de modelos según el problema.
2. **Ajuste de hiperparámetros (EM-2)**: Configuración de los parámetros del modelo para optimizar su rendimiento.
3. **Seguimiento de modelos (EM-3)**: Monitorear y registrar los experimentos de entrenamiento para reproducir resultados.
4. **Compatibilidad con diferentes frameworks (EM-4)**: Integración con múltiples frameworks como TensorFlow o PyTorch.
5. **Versionado de código (EM-5)**: Guardar versiones del código para asegurar la trazabilidad y revertir cambios fácilmente.

Etapa de gestión de modelos:

1. **Empaquetado de modelos (GM-1)**: Preparar los modelos entrenados para su despliegue y reutilización.
2. **Registro de modelos (GM-2)**: Almacenar los modelos y sus versiones junto con metadatos en un registro centralizado.

Etapa de despliegue de modelos:

1. **Soporte para diferentes patrones de despliegue (DM-1)**: Desplegar modelos en diferentes entornos como la nube, dispositivos o en contenedores.

Etapa de operaciones y monitoreo:

1. **Monitoreo del sistema (OM-1)**: Supervisar la infraestructura donde se despliegan los modelos, controlando la carga del sistema y el rendimiento.
2. **Monitoreo de datos de entrada (OM-2)**: Revisar los datos entrantes para detectar errores, desviaciones o cambios en los patrones.

3. **Monitoreo del rendimiento del modelo (OM-3)**: Asegurar que los modelos sigan siendo efectivos con el tiempo y reentrenarlos si es necesario.

Requisitos generales:

1. **Escalabilidad (RG-1)**: Capacidad de aumentar o disminuir recursos según la demanda del proyecto.
2. **Disponibilidad de APIs (RG-2)**: Facilitar la automatización de procesos a través de APIs.
3. **Madurez de la herramienta (RG-3)**: Uso probado y confiabilidad en entornos productivos.
4. **Facilidad de uso (RG-4)**: Herramientas fáciles de manejar para usuarios no técnicos.
5. **Compatibilidad con Windows (RG-5)**: Indica si la herramienta puede ejecutarse en sistemas operativos Windows.
6. **Compatibilidad con Linux (RG-6)**: Indica si la herramienta puede ejecutarse en sistemas operativos Linux.

Comparación de herramientas

Estas herramientas se evalúan en la Tabla 3.1 de acuerdo con los requisitos mencionados en la sección 3.2.5, dando además una clasificación de cuán bien cumplen con dichos requisitos. La clasificación es la siguiente:

- **o**: La funcionalidad está ausente.
- +: La funcionalidad está presente, pero no cumple completamente con el requisito o no es fácil de usar.
- ++: La funcionalidad está presente y es fácil de integrar y usar.

3.2.6. Justificación de herramientas seleccionadas

Basándonos en los criterios expuestos, se seleccionaron las siguientes tres herramientas **clave** para el desarrollo del proyecto:

Airflow

Como se comentó en la sección 3.1, Airflow es una herramienta de orquestación que nos permite definir por medio de DAG distintos steps y la interacción entre ellos en donde se conforma el pipeline.

Se eligió la herramienta debido a su capacidad para orquestar estos pipelines de manera ordenada. Otra de sus ventajas principales es la visualización en tiempo real de las ejecuciones, lo que facilita el monitoreo y la gestión de los procesos. Aunque Airflow no tiene soporte nativo para Windows, el uso de WSL permitió su implementación en los entornos existentes.

TaskFlow API de Airflow

TaskFlow API permite la definición de DAG en Airflow, que fue introducido a partir de la versión 2.0 de la herramienta y que nos permite generar nuestros DAG e intercambiar datos entre distintas *Tasks* apoyándose sobre XCom[7] de una manera mucho más sencilla y de manera menos extensa que con la versión anterior. Por medio de decoradores, TaskFlow encapsula mucha lógica repetitiva en la definición, tanto de DAG como de Tasks, lo que no solo nos simplifica el trabajo, sino que nos deja con un código más legible y mantenible.

Como se mencionó anteriormente, una de las mayores ventajas que nos da TaskFlow si la comparamos con la forma tradicional de generar DAG, es que nos permite generarlo utilizando menos código y de una manera mucho más concisa. Por medio del decorador `@dag` podemos transformar un método de Python en un DAG. A su vez, por medio del decorador `@task`, también podemos transformar un método Python en una Task.

Veamos con un ejemplo concreto, inspirado en la documentación oficial de Airflow y de TaskFlow API, las diferencias al definir DAG y Tasks con y sin TaskFlow API. Los ejemplos completos se encuentran en la Sección .1.

Como podemos observar, al definir el DAG y los decoradores de Tasks de TaskFlow API requerimos de menos código, ya que no tenemos que poner *PythonOperator()* cada vez, partiendo tanto el DAG como las Tasks, de métodos de Python.

Además, con TaskFlow la dependencia entre Tasks es directamente inferida por Airflow dada la definición e interacción entre cada una de ellas, y no debe ser explicitada como si deberíamos hacerlo de la manera convencional. En el ejemplo, Airflow entiende que la Task de **train**, depende de la Task de **pre_process**, que a su vez depende de **read_input**.

Por otro lado, al definir el pipeline con TaskFlow e indicar:

```
var = read_input()
pre_processed_input = pre_process(var)
```

Airflow no solo detecta la relación entre las Tasks **pre_process** y **read_input**, sino que también se encarga de transferir automáticamente los datos entre ellas, en este caso, la variable **var**.

MLflow

Por otra parte, dado que las funcionalidades más potentes de MLflow son su API de Tracking y su Model Registry, decidimos utilizarlo justamente con esos fines: para registrar (Tracking) los parámetros de cada ejecución y/o experimento, y para mantener un registro y versionado (Model Registry) de los

[7] En el contexto de Airflow, XCom (abreviatura de *cross-communication*) es un mecanismo que permite el intercambio de datos entre tareas dentro de un DAG. Cada mensaje de XCom consta de una clave, un valor y metadatos asociados, y puede ser utilizado para compartir información como resultados de tareas, parámetros de configuración y otros datos necesarios para la ejecución del flujo de trabajo.

distintos modelos entrenados y utilizados. Mediante una interfaz de usuario podemos ver distintos parámetros y modelos registrados para distintas ejecuciones de distintos experimentos, como se describirá más a detalle en la sección 6.3. Su capacidad para integrarse en múltiples etapas del flujo de trabajo de Machine Learning y su compatibilidad con Windows y Linux hicieron de MLflow una opción ideal.

TensorFlow Extended

Se decidió utilizar TensorFlow Extended, ya que ofrece soporte para prácticamente todas las etapas del ciclo de vida de MLOps. TFX fue elegido por su enfoque integral y su capacidad para participar en procesos de entrenamiento, validación y despliegue de modelos. Si bien está optimizado para entornos Linux, el uso de WSL facilitó su integración en los sistemas Windows utilizados por UTE.

WSL

Finalmente, la inclusión de **WSL** fue crucial para garantizar la compatibilidad y ejecución de herramientas como Airflow y TFX en máquinas que corren Windows, cumpliendo con los requisitos específicos del entorno técnico de UTE para la ejecución de las herramientas de direccionamiento de inspecciones.

Justificación de la exclusión de WSL en la tabla comparativa:

No se incluyó **WSL** en la tabla comparativa de herramientas debido a que no es una herramienta de MLOps en sí, sino una solución que permite la ejecución de un entorno Linux directamente sobre el kernel de Windows. WSL no proporciona funcionalidades específicas para el manejo de datos, entrenamiento de modelos, gestión o monitoreo, como las demás herramientas evaluadas. En lugar de ello, WSL actúa como una capa de compatibilidad que habilita el uso de herramientas optimizadas para Linux en máquinas que corren Windows, permitiendo que el software de MLOps, como Airflow y TFX, se ejecute de manera nativa sin las limitaciones que tendrían en un sistema Windows tradicional.

Dado que **WSL no afecta directamente los flujos de trabajo de MLOps**, sino que facilita la compatibilidad del sistema operativo, se decidió no incluirlo en la tabla comparativa de herramientas, enfocándonos en soluciones que ofrecen funcionalidades específicas para el desarrollo y despliegue de AA.

Nombre de la Herramienta	**Preprocesamiento de Datos**	**Entrenamiento de Modelos**	**Gestión de Modelos**	**Despliegue de Modelos**	**Operaciones y Monitoreo**	**Requisitos Generales**
Airflow	+, 1, 2	o	o	o	+, 1	+, 1, 2, 3, 6
MLflow	o	+, 3, 4	+ 1, 2	+, 1	o	+, 2, 3, 4, 5, 6
TFX	+, 2, 3, 4, 5	+, 3	++, 1, 2	+, 1	+, 2, 3	+, 1, 3, 6
TensorFlow	+, 2	+, 2, 3	o	+, 1	+, 3	1, 2, 3, 5, 6
H2O	+, 1, 2	+, 1, 2	1	o	o	1, 2, 4, 5, 6
Kubeflow	+, 2, 5	++, 1, 2, 3, 4, 5	+, 1	+, 1	+, 1	+, 1, 3, 6
Great Expectations	+, 3	o	o	o	+, 2	+, 3, 4, 5, 6
DVC	+, 1, 5	+, 3, 5	+, 2	o	o	++, 1, 2, 3, 4, 5, 6
Pachyderm	+, 1, 2, 5	+, 5	o	o	o	+, 1, 3, 6

Cuadro 3.1: Herramientas MLOps con abreviaturas de sus funcionalidades y etapas que cubren. Los números describen cuál de las diferentes subetapas son cubiertas dentro de la etapa.

Capítulo 4

Descripción del Problema y Requerimientos de la Solución

Esta sección detalla el problema a resolver, poniendo énfasis en los desafíos y requerimientos específicos de la solución. Se discute la necesidad de una plataforma centralizada para el registro de modelos, la gestión eficiente del flujo de datos de diversas fuentes y los pasos cruciales en el preprocesamiento, limpieza, transformación, validación y evaluación de datos para formar un conjunto de datos adecuado para el entrenamiento de modelos de aprendizaje automático.

4.1. Descripción del problema y contexto

Las colaboraciones entre la Facultad de Ingeniería y UTE se han centrado en proyectos relacionados con los software DAICE y DeepDAICE. Desde 2018, DAICE ha sido utilizado con éxito para direccionar inspecciones, obteniendo buenos resultados en campo. Sin embargo, hasta 2019 se enfrentaba un cuello de botella en la fase de extracción de datos. Dicha extracción se realizaba de manera programática y no aprovechaba las ventajas de contar con bases de datos alojadas en servidores más potentes, por lo que gran parte del procesamiento se realizaba en computadoras personales.

Este desafío impulsó el desarrollo de una solución que ejecutara la extracción directamente sobre las bases de datos mediante el uso de procedimientos almacenados. Esto no solo incrementó la velocidad de extracción, sino que también facilitó la obtención de los datos. Como resultado, se logró escalar el sistema para abarcar todo Uruguay. Anteriormente, el software trabajaba con bases de datos de aproximadamente 30 mil instancias [1]; la mejora en el proceso de ex-

[1] Las instancias se refieren a los datos recopilados para cada medidor o suministro de energía instalado.

tracción permitió aprender de hasta 400 mil instancias y clasificar todos los suministros de energía del país para el año 2022.

Debido al impacto positivo que tuvo la optimización en el direccionamiento de inspecciones mediante el uso del software de AA, surgió la necesidad de continuar mejorando los modelos, optimizar el flujo de datos, agregar y probar nuevas características, y visualizar los resultados obtenidos en campo, incluyendo las tasas de acierto.

Por otro lado, la herramienta DeepDAICE, que trabaja con datos provenientes de medidores inteligentes con mediciones cada 15 minutos, maneja grandes volúmenes de datos (*Big Data*). El acceso a esta información, fundamental para alimentar los modelos de aprendizaje profundo, es posible gracias a la implementación de tecnología Big Data, basada en un repositorio HBase gestionado por el stack de Cloudera, conocido como MDM (Meter Data Management) en UTE [72].

Dado este enorme volumen de datos, no es posible cargar todos los registros en memoria para su procesamiento en modelos de AA. Por ello, es esencial adaptar el sistema a un entorno de trabajo con mayor escalabilidad y capacidad de cómputo.

El entrenamiento de estos modelos de aprendizaje profundo requiere una cantidad significativa de tiempo (superior a 2 horas para entrenar un solo modelo durante unas 20 épocas). Además, el ajuste de hiperparámetros para mejorar el rendimiento de los modelos añade complejidad al proceso.

4.1.1. Optimización de la Frecuencia de Actualización de Datos para Mejorar la Precisión de los Modelos

En el contexto de DAICE, la implementación de cambios en los procedimientos almacenados o consultas relacionados con la extracción de datos desde las bases de datos es una tarea crítica que requiere gran meticulosidad. Para asegurar la calidad de los datos y el correcto funcionamiento del sistema, se realizan exhaustivas verificaciones basadas en métricas de correlación, análisis exploratorio de datos e información estadística, lo que garantiza la integridad de los datos obtenidos.

Un aspecto crucial del proceso es la frecuencia de actualización de los datos utilizados para generar nuevos modelos de clasificación. En la figura 4.1, se observa la relevancia de ciertas características, destacando los últimos consumos medidos como factores clave en las decisiones del algoritmo. Entre las 20 características más importantes, sobresalen los consumos C24 y C25, que forman parte del tramo final utilizado. Los meses más recientes de consumo muestran una importancia significativa, lo que impulsa la necesidad de actualizar mensualmente tanto los modelos como los datos de entrada para mantener un rendimiento óptimo. En general, se utilizan los últimos 36 consumos (C1 hasta C36).

En cuanto a la frecuencia de actualización, estudios realizados por el Departamento de Recuperación de Energía (DRE) analizaron los puntos de servicio inspeccionados en noviembre de 2022 en la zona norte del país. Se llevaron a

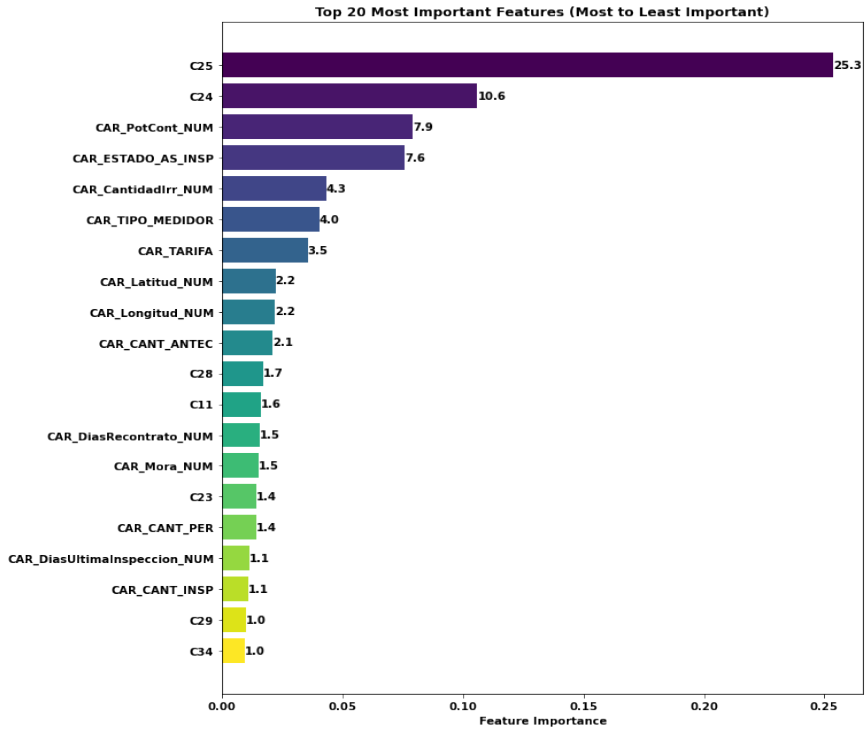


Figura 4.1: Importancia de las características para DAICE, las primeras 2 barras representan los últimos consumos, luego se muestran las demás características utilizadas.

cabo un total de 3069 inspecciones, algunas de las cuales habían sido clasificadas en los meses de enero, mayo, junio, septiembre y octubre. Los resultados fueron los siguientes:

- Para enero, en un total de 1443 inspecciones, el acierto fue de 6.2 %
- En mayo, en un total de 547 inspecciones fue de 9.1 %
- En junio, en un total de 181 inspecciones, el acierto fue de 2.2 %
- En setiembre, en 519 inspecciones, el acierto fue de 10.59 %
- en octubre, en 379 inspecciones, el acierto fue de 11.34 %

Estos resultados resaltan la importancia de actualizar con frecuencia los datos en DAICE, con el objetivo de realizar inspecciones lo más cercanas posible en el tiempo a la clasificación inicial. Esto minimiza la posibilidad de que el

acierto disminuya debido a fenómenos como el data drift y el concept drift[2]. Aunque observamos una anomalía en el mes de junio, estos datos no demuestran de manera concluyente la presencia de data drift, pero sí sugieren indicios de su ocurrencia.

4.1.2. Verificación y Validación del Modelo en Producción: Estrategias para Garantizar su Eficacia

En el proceso de verificación y validación del modelo en producción, se emplean diversas técnicas para garantizar su eficacia. Una de las estrategias clave consiste en graficar la correlación entre un modelo ya utilizado en campo y otro recientemente elaborado. Esta comparación se realiza con el propósito de evaluar la consistencia entre ambas versiones del modelo, siempre y cuando no haya cambios demasiado significativos. La correlación entre ambos modelos se realiza de la siguiente manera:

- Primero se seleccionan dos modelos que sean de interés para comparar.
- Luego, se clasifican los clientes para generar una puntuación o probabilidad de irregularidad según cada uno de los modelos.
- Se ordenan ambos listados de mayor a menor puntuación.
- Finalmente, se genera un gráfico de calor para mostrar las posiciones de los clientes en cada una de las clasificaciones.

En la figura 4.2, se puede apreciar la correlación entre las predicciones de dos modelos elaborados con diferentes versiones de DAICE.

La visualización resultante proporciona una perspectiva clara, tomándose como un indicador acerca de la consistencia entre los modelos y permite interpretar si el nuevo modelo es adecuado para su uso en producción. Además, se utiliza el índice de correlación lineal como una métrica de referencia para cuantificar la relación entre las predicciones de ambos modelos. Si existe una correlación significativa, se puede inferir que el nuevo modelo puede ser utilizado en producción con mayor confianza.

Además de las ventajas mencionadas previamente, el proceso de A/B testing [3] manual también puede proporcionar información crucial para determinar si es el momento adecuado para pasar de una versión beta de un modelo a una versión oficial para su implementación en producción.

[2]El data drift se refiere a cambios en la distribución de los datos con el tiempo, lo que puede afectar negativamente el rendimiento del modelo, mientras que el concept drift ocurre cuando las relaciones entre los datos y las etiquetas (resultados) cambian. Ambos fenómenos pueden degradar la precisión de los modelos de clasificación.

[3]El A/B testing es una técnica utilizada para comparar dos versiones de un modelo o sistema, generalmente la versión actual (A) y una nueva versión propuesta (B). Se implementan ambas versiones en paralelo y se evalúan sus rendimientos en función de métricas específicas, como precisión o tasas de error. Esto permite tomar decisiones basadas en datos sobre cuál versión es más efectiva para los objetivos definidos.

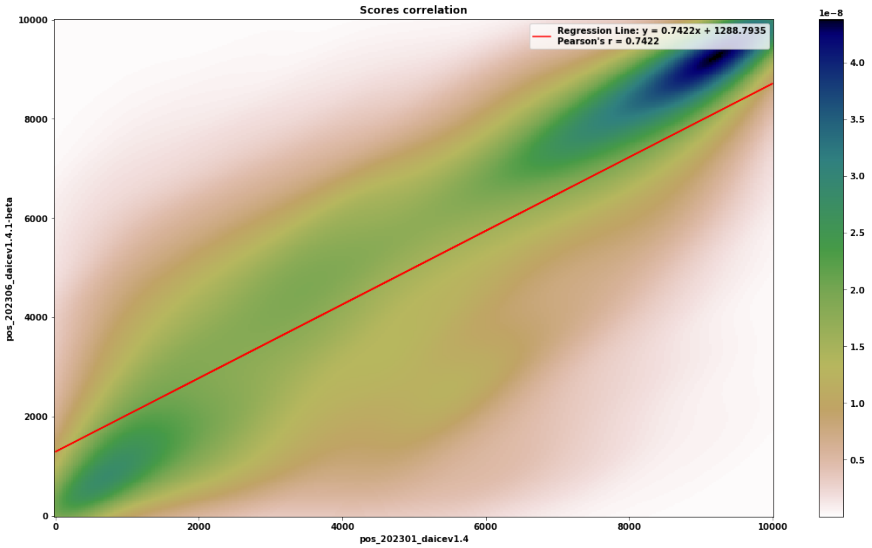


Figura 4.2: Técnica usada para comparar diferentes modelos, donde se compara la correlación de posiciones ordenadas según la probabilidad inferida de ser irregular.

Cuando se desarrollan nuevas versiones de modelos de machine learning, es común que se realicen pruebas en una fase de desarrollo inicial o beta. Durante esta etapa, el equipo de ciencia de datos lleva a cabo una serie de experimentos y ajustes para mejorar el rendimiento y la eficacia del modelo. Sin embargo, antes de lanzar una versión oficial del modelo para su implementación en el entorno de producción, es necesario evaluar su desempeño en condiciones reales y en un escenario más amplio.

Es en este punto donde el A/B testing manual juega un papel fundamental. Al enviar inspecciones a campo utilizando diferentes versiones del modelo, el equipo puede comparar su rendimiento en situaciones reales y verificar si las mejoras implementadas en la versión beta realmente generan una mejora significativa en la detección de irregularidades en redes eléctricas.

Durante el A/B testing manual, se evalúan meticulosamente aspectos clave como la precisión de la detección de irregularidades, la tasa de falsos positivos y negativos, y la eficiencia general del modelo en la identificación de comportamientos irregulares en el consumo eléctrico. Esto proporciona una visión más completa de cómo se desempeñan las diferentes versiones del modelo en el contexto de producción y ayuda a tomar decisiones fundamentadas sobre cuál versión es más adecuada para su uso a nivel de producción.

En el departamento de recuperación de energía, antes de liberar un modelo de machine learning a producción, se llevan a cabo pruebas de rendimiento o evaluaciones para asegurar que el modelo cumpla con los requerimientos de ren-

dimiento. Para ello, se evalúan métricas clave como el PR_AUC [4] y la precisión utilizando conjuntos de datos de validación o prueba que no fueron utilizados durante el entrenamiento del modelo. El PR_AUC es especialmente relevante en problemas de clasificación desbalanceados, como la detección de irregularidades, donde los casos positivos son menos frecuentes que los negativos. La precisión, por otro lado, mide la proporción de verdaderos positivos entre todas las instancias clasificadas como positivas.

Además de las métricas, se generan gráficas de precision-recall para visualizar cómo varía la precisión en función del recall a diferentes umbrales de clasificación. Estas gráficas proporcionan una comprensión más detallada del rendimiento del modelo y su capacidad para recuperar correctamente los casos positivos.

Una parte esencial del proceso implica comparar el rendimiento del modelo con el de otros modelos previamente desarrollados y evaluados experimentalmente. Esta comparación permite identificar cuál modelo muestra un rendimiento superior y cuál es más adecuado para su implementación en campo. Esto garantiza que el modelo sea evaluado en situaciones realistas que representan su funcionamiento en producción.

Hay que destacar que las comparaciones de estas métricas se realizan a menudo a nivel de Jupyter Notebooks, lo que implica que el proceso de evaluación puede ser manual y laborioso. Esto se debe a que cada ejecución del modelo y el cálculo de las métricas deben registrarse y analizarse individualmente en estos entornos interactivos.

Sin embargo, esta forma de trabajar presenta desafíos, especialmente cuando se busca obtener valores de métricas de modelos anteriores, incluso que se pierdan algunos datos o sean difíciles de encontrar. La búsqueda y recuperación de resultados históricos en los notebooks puede ser complicada, y puede requerir la revisión y ejecución de celdas específicas para obtener los valores deseados. Esta falta de automatización y trazabilidad puede llevar a dificultades para obtener una visión completa del rendimiento de los modelos y datos de experimentaciones a lo largo del tiempo.

4.1.3. Desafíos de Rastreabilidad en la Gestión de Modelos de Machine Learning en Producción

La transición de un entorno interactivo y manual a uno de producción amplifica los desafíos asociados con la rastreabilidad y la gestión de modelos. En producción, la frecuencia de las actualizaciones y modificaciones a los modelos de aprendizaje automático se intensifica, lo que hace que la necesidad de una gestión y rastreabilidad eficaz sea aún más imperativa.

[4] El PR_AUC (Área Bajo la Curva de Precision-Recall) es una métrica utilizada en problemas de clasificación, especialmente cuando los datos están desbalanceados. Mide la relación entre la precisión (proporción de verdaderos positivos sobre todas las predicciones positivas) y el recall (proporción de verdaderos positivos sobre todos los casos positivos reales) a diferentes umbrales de clasificación. Un PR_AUC más alto indica un mejor rendimiento del modelo en la recuperación de casos positivos mientras minimiza falsos positivos.

En este escenario de producción, la persistencia de los desafíos de rastreabilidad previamente enfrentados en los Jupyter Notebooks se traduce en dificultades amplificadas para el equipo de ciencia de datos. Ahora, no solo se trata de comparar métricas y evaluar modelos, sino de asegurar que cada versión del modelo y los datos asociados estén adecuadamente documentados y sean fácilmente accesibles. Esta rastreabilidad no solo facilitaría la comparación entre diferentes modelos, sino que también permitiría una comprensión más profunda de cómo las variaciones en el código o en el procesamiento de datos impactan en el rendimiento del modelo en un entorno real y dinámico.

Así, se subraya la importancia de contar con sistemas robustos de gestión de modelos y rastreabilidad, tanto en las etapas de evaluación como en la implementación en producción, para asegurar una operación eficiente, un monitoreo efectivo y una mejora continua de los modelos de Machine Learning.

La falta de acceso a la información detallada sobre la configuración y contexto de los modelos previos puede generar incertidumbre y limitar la capacidad de realizar análisis comparativos sólidos. Esta falta de rastreabilidad puede afectar la toma de decisiones estratégicas y dificultar el desarrollo de mejoras progresivas en los modelos de aprendizaje automático.

Para complicar aún más la situación, los desafíos en la rastreabilidad no se limitan a los modelos de aprendizaje automático en sí, sino que también se extienden a las etapas previas del flujo de trabajo, como la **extracción de datos**.

4.1.4. Desafíos en el Rastreo de Errores en la Extracción de Datos

Al igual que con los modelos de aprendizaje automático, la rastreabilidad en la extracción de datos es crucial para el éxito de cualquier proyecto de ciencia de datos. En el ámbito de producción, existen diversos desafíos derivados de la complejidad en el rastreo y la detección de errores durante la extracción de datos. En ocasiones, estos errores son el resultado de acciones humanas, lo que dificulta aún más su identificación y corrección. La necesidad de generar extracciones de datos de forma mensual agrega una presión adicional a este proceso y aumenta la probabilidad de que ocurran tales errores. Estos pueden pasar desapercibidos durante cierto tiempo, afectando negativamente el rendimiento y la precisión de los modelos de machine learning utilizados en el entorno de producción.

En este sentido, en el transcurso de nuestra investigación, hemos identificado la necesidad de establecer un sistema robusto de rastreo y registro de los pasos involucrados en cada etapa del ciclo de vida de los modelos de machine learning. Además, hemos enfocado nuestros esfuerzos en desarrollar una plataforma que nos permita recuperar y visualizar de manera clara y sencilla la información relevante sobre cada modelo y sus datos de entrada.

Con este enfoque centrado en la trazabilidad y la gestión adecuada de los datos y modelos, no solo buscamos optimizar el proceso de extracción de datos, sino también minimizar la incidencia de errores humanos. Esto nos lleva a una mayor confiabilidad de los modelos de machine learning en producción, lo cual es

crucial para cumplir con una serie de requisitos y tareas críticas que detallaremos a continuación.

4.2. Requerimientos funcionales y no funcionales

En esta sección se describirán los requerimientos, tanto funcionales como no funcionales, definidos para el proyecto. Los requerimientos surgen de las etapas clave del proceso de MLOps (ver sección 3.2.5), que guían el desarrollo y la implementación del sistema. A continuación, se detallan estos requerimientos, haciendo referencia a las fases del preprocesamiento de datos, entrenamiento de modelos, gestión de modelos, despliegue de modelos y monitoreo de modelos.

4.2.1. Requerimientos Funcionales

Verificación de calidad de los datos

Este requerimiento está alineado con la **etapa de preprocesamiento de datos** (ver sección 3.2.5), que incluye la verificación de la calidad y consistencia de los datos antes de que sean utilizados por el modelo.

- **Verificación de los datos (desde la base de datos y archivos h5 y CSV)**
  El sistema verificará la calidad de los datos, como se describe en la etapa de preprocesamiento, para asegurar que sean válidos antes de su uso.

- **Verificación del esquema de los datos**
  Basado en la *medición de la calidad de los datos* (PD-3), el sistema verificará que los datos sigan el esquema definido.

- **Verificación de la consistencia de los datos**
  En concordancia con la gestión de datos (PD-4), se validará que los datos estén dentro de los rangos esperados.

- **Verificación de sesgos entre datos de entrenamiento y de inferencia**
  Alineado con la *medición de calidad* y *seguimiento de modelos* (EM-3), el sistema comparará los datos de entrenamiento y producción.

Pipeline del modelo

Los requerimientos de esta sección están relacionados con la **Etapa de entrenamiento de modelos** (ver sección 3.2.5) y la **Etapa de gestión de modelos** (ver sección 3.2.5), que cubren el ciclo de vida de los modelos, desde su entrenamiento hasta su registro.

- **Entrenamiento del modelo**
  El sistema deberá permitir el entrenamiento, alineado con la selección y ajuste del modelo (EM-1, EM-2).

- **Evaluación del modelo**
  Comparación de métricas obtenidas durante el entrenamiento, como se menciona en la fase de seguimiento de modelos (EM-3).
- **Validación del modelo**
  Comparación con otros modelos utilizando *AB Testing*, siguiendo la gestión de modelos (GM-1, MM-2).
- **Registro del modelo**
  Los modelos deberán ser registrados en un *model registry*, alineado con la etapa de gestión de modelos (GM-2), almacenando hiperparámetros y métricas relevantes.

Conexión a Big Data de medidores inteligentes

Este requerimiento está alineado con la **Etapa de preprocesamiento de datos** (PD-1), que incluye la integración de datos desde fuentes como bases de datos no relacionales y Big Data, así como con la **Etapa de despliegue de modelos** (DM-1) que asegura que los modelos puedan acceder a los datos relevantes.

Rendimiento del modelo

Este requerimiento se alinea con la **Etapa de operaciones y monitoreo** (OM-3), que detalla la necesidad de monitorear continuamente el rendimiento de los modelos en producción.

Servicio de clasificación

La solución deberá permitir el acceso al modelo desde una página web, utilizando un sistema de clasificación basado en los datos de entrada, como se indica en la **Etapa de despliegue de modelos** (DM-1).

4.2.2. Requerimientos No Funcionales

- El proyecto debe desarrollarse utilizando herramientas Open Source.
- Se debe tener en cuenta la Ley N.º 18331 de protección de datos personales.
- El sistema debe desplegarse en las máquinas propietarias de UTE.
- El sistema debe tener la capacidad de ejecutarse en el sistema operativo Windows (10 o superior) y en sistemas operativos Linux (relacionados con las etapas de los requisitos generales RG-5 y RG-6 en la sección 3.2.5).

Capítulo 5

Propuesta de Arquitectura

La arquitectura propuesta se basa principalmente en la combinación de las herramientas de software libre Airflow y MLflow, utilizando los distintos artefactos que se pueden generar en cada una de ellas, aprovechando las capacidades de TaskFlow [73] y los archivos de configuración para diseñar un pipeline de procesamiento de datos eficiente y escalable. Se describirá cómo se integran estas herramientas y se explicará su funcionamiento en conjunto, resaltando las ventajas que ofrecen en términos de orquestación de tareas, registro de modelos, gestión de métricas y monitoreo del flujo de trabajo.

5.1. Diagramas

La **arquitectura de software** representa la decisión de diseño más temprana y está condicionada por criterios de calidad como la mantenibilidad, seguridad y el rendimiento. Ya que una vez establecida es modificable más tarde solo con gran esfuerzo, la decisión acerca de su diseño es uno de los puntos más críticos e importantes en el proceso de desarrollo de software. Dicho esto, el diagrama de arquitectura presentado en la figura 5.1 ilustra la propuesta de arquitectura presentada en este proyecto. Esta arquitectura propuesta consta de distintos pipelines, que se encargan de resolver distintas etapas de un flujo "típico" de Machine Learning. En el se pueden ver 4 componentes principales, que nos permiten realizar la lectura de los datos de alguna fuente, realizar una validación y preparación de datos, realizar un entrenamiento de un modelo a partir de estos datos, así como realizar la clasificación de algunos datos utilizando algún modelo previamente entrenado y registrado.[1].

Se puede apreciar claramente como la arquitectura principal se divide en cuatro pipeline principales, los cuales son **Data Ingestion Pipeline**, **Data Pipeline**, **Model Pipeline** y **Classification Pipeline**.

[1]Los términos en el diagrama se han dejado en inglés, ya que reflejan mejor el contexto técnico manejado en este trabajo, donde los nombres de las herramientas y tecnologías están mayoritariamente en ese idioma.

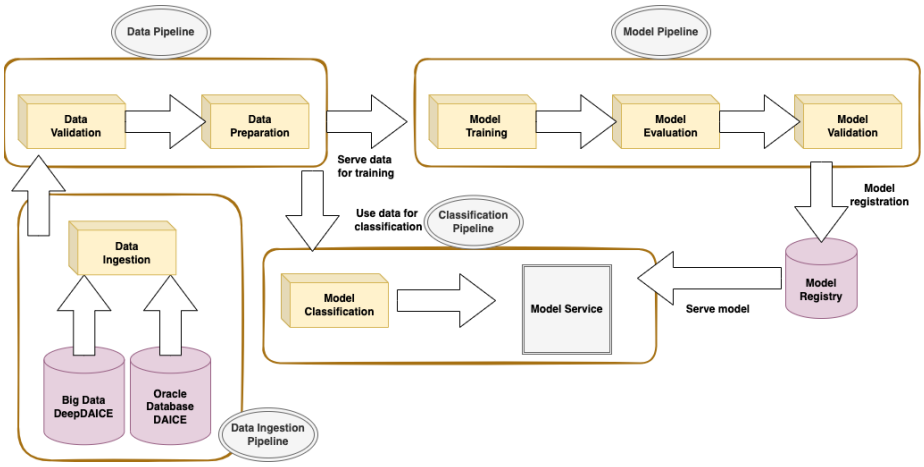


Figura 5.1: Vista general de la arquitectura propuesta con los pipelines correspondientes

En la figura 5.1 se observan, como parte del **Data Ingestion Pipeline**, las fuentes de datos correspondientes a las bases de datos disponibles: la de Big Data para **Deep DAICE** y Oracle Database mayormente usada por **DAICE**.

A continuación, se da una descripción un poco más detallada de la idea de cada pipeline (o módulo) de la arquitectura:

- En el **Data Ingestion Pipeline** se obtienen los datos de las diferentes fuentes (como se mencionaba previamente, podría ser de una base de datos de Big Data, o de cualquier base de datos relacional o no relacional). La idea es que este pipeline permita obtener los datos, y dejarlos disponibles en el formato que sea requerido, para ser utilizados por el pipeline siguiente
- En el **Data Pipeline** se obtienen los datos disponibilizados por el pipeline anterior, buscando verificar su calidad. También se aplican transformaciones sobre los mismos o se formatean adecuadamente. Esto incluye procesos de limpieza y obtención de características relevantes directamente de la fuente.
- El **Model Pipeline** está fundamentalmente relacionado con las fases típicas de los modelos de aprendizaje automático, como son el entrenamiento, la evaluación y la validación.
- El **Classification Pipeline** está fundamentalmente relacionado con realizar la clasificación o inferencia, utilizando datos previamente obtenidos y validados por los pipelines anteriores, y de un modelo también entrenado por el pipeline anterior (y registrado en el Model Registry).

Se hace además una mención especial a una componente importante de la

arquitectura, que no es un pipeline en sí mismo, sino que nos permite registrar los modelos que se entrenan.

- **Model Registry** permite la trazabilidad y el registro de cada modelo que sea entrenado utilizando el sistema. A través de este registro es posible recuperar cada parámetro o artefacto guardado, además de justamente, los modelos. De esta forma, los modelos quedan disponibles para el servicio de clasificación según se desee.

A continuación se presenta también una extensión del diagrama anterior, donde se muestran también, donde participa cada una de las herramientas elegidas 5.2.

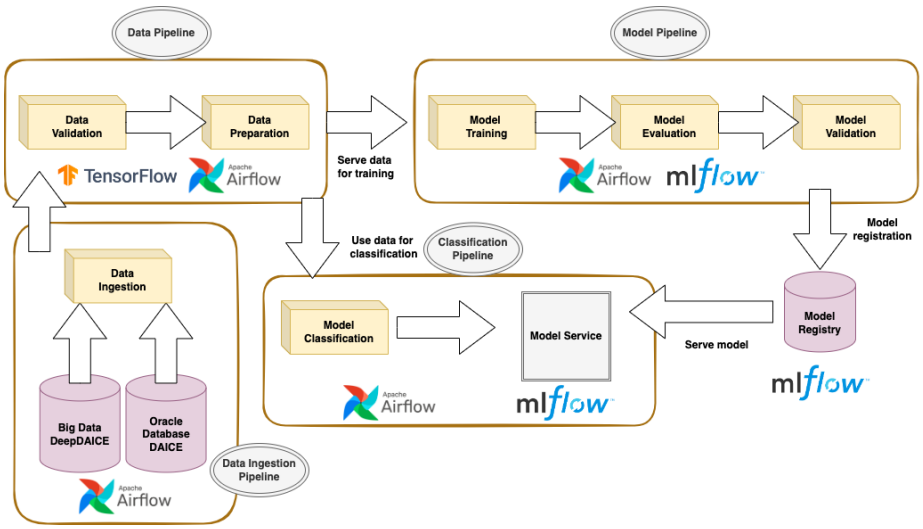


Figura 5.2: Vista general de la arquitectura propuesta con los pipelines correspondientes, incluyendo además las herramientas seleccionadas

5.2. Visión General de la Arquitectura

Como comentamos previamente, la arquitectura propuesta se basa, no solo en contar con distintos módulos que permitan resolver distintas etapas del flujo, sino que también se basa en la combinación de las herramientas de software libre Airflow y MLflow, intentando aprovechar las mejores capacidades de ambas para obtener un sistema que cubra las necesidades. Por ejemplo, utilizando la TaskFlow API de Airflow y la posibilidad de ejecutar los DAG con archivos de configuración para diseñar un pipeline de procesamiento de datos eficiente y escalable, junto con el Tracking API y el Model Registry de MLflow. En la sección 6.1.9 se describirá cómo se integran estas herramientas y se explicará su funcionamiento en conjunto, resaltando las ventajas que ofrecen en términos

de orquestación de tareas, registro de modelos, gestión de métricas y monitoreo del flujo de trabajo.

Al adoptar esta arquitectura, entendemos que se logra una modularización efectiva del proceso de Machine Learning, lo que nos acercaría a un nivel más avanzado de MLOps, ya que el que se tiene actualmente es el nivel 0, es decir, sin la aplicación de estos conceptos. La modularización permite una mayor reutilización de componentes y una mejor organización del flujo de trabajo, lo que facilita el mantenimiento, la escalabilidad y la colaboración en el desarrollo de soluciones de Machine Learning en general.

Capítulo 6

Aplicación de la solución definida al caso DAICE y DeepDAICE

En este capítulo se aborda la adaptación de nuestra solución MLOps al contexto específico del DAICE, enfocándonos en la detección avanzada de irregularidades en redes eléctricas [74]. Iniciamos con un análisis explicativo de las métricas clave como Precisión-Recall y ROC AUC, empleando MLflow para evaluar de forma general el desempeño de nuestros modelos de aprendizaje automático.

Posteriormente, se discute la evaluación integral de los modelos actuales, considerando no solo su eficacia actual en la detección de irregularidades, sino también explorando vías para la inclusión y automatización de nuevos modelos bajo el paradigma MLOps, lo que representa un paso más para el logro de un mayor nivel de MLOps o automatismo permitiendo mejorar la detección de irregularidades.

La creación de un pipeline de aprendizaje automático robusto y dinámico utilizando Airflow es otro de los temas que se exploran, destacando cómo esta herramienta facilita la orquestación o automatización de procesos.

Se profundiza en el uso de MLflow para lograr la trazabilidad de los experimentos, parámetros, métricas y artefactos, un aspecto crítico para la iteración y mejora continua de los modelos. Además, se examina la transición del sistema DAICE hacia una infraestructura más sofisticada y automatizada, soportada por Airflow y MLflow, delineando los pasos y consideraciones necesarios para esta migración.

Este capítulo también incluye una exploración de flujos de trabajo de aprendizaje automático de punta a punta con TFX, probados a través de la implementación de tutoriales disponibles con datos publicados en la web. Este análisis brinda una aproximación valiosa para la aplicación de la solución en el ámbito de DeepDAICE, también muestra el potencial de TFX en diferentes contextos

de datos.

Finalmente, se evalúa cómo la integración de TFX, Airflow y MLflow cumple con los requerimientos establecidos para el proyecto, marcando un hito en la implementación de soluciones MLOps para la detección de irregularidades en redes eléctricas y estableciendo un marco para futuras iniciativas en el ámbito de la inteligencia artificial aplicada.

6.1. Análisis Integral de Modelos de Detección de Irregularidades

Este segmento profundiza en la evaluación de los modelos desarrollados a lo largo del trabajo haciendo uso de los diferentes pipelines de procesamiento. Detallaremos las métricas clave para evaluar la eficacia de estos modelos, explicando su importancia y el proceso de almacenamiento y registro. Presentaremos ejemplos específicos de métricas esenciales en el contexto de la detección de irregularidades, proporcionando una comprensión integral de su aplicación y comparación.

6.1.1. Automatización de la detección de irregularidades en redes eléctricas utilizando MLOps

En este apartado, se presenta un ejemplo de flujo de trabajo básico de nivel 1 de MLOps, basado en la detección de irregularidades en redes eléctricas mediante el análisis de curvas de consumo. Se combinan algunas de las herramientas mencionadas anteriormente para demostrar cómo se pueden utilizar dichas herramientas de código abierto en los casos de uso de UTE, donde no se requieren nuevos modelos con alta frecuencia y no hay necesidad de una implementación continua. Este ejemplo puede ser valioso para futuras etapas de automatización y mayores niveles de MLOps en lo que tiene que ver con el desarrollo de ML.

Es muy común el uso de herramientas de versionado de código como Git y GitHub. Este caso de uso también aplica el proceso de desarrollo basado en el control de versiones, que ya ha sido adoptado por la empresa. Las técnicas aplicadas en el desarrollo de este proyecto, permiten que incluso se puedan reutilizar antiguos notebooks y códigos de ciencia de datos para diseñar un flujo de trabajo tan simple o complejo como se requiera.

El diseño de un flujo de trabajo automatizado de MLOps sigue un patrón similar para un gran conjunto de dominios. Lo que en general varía de un dominio a otro son las métricas utilizadas en la evaluación de la calidad de los datos, la evaluación del modelo y la evaluación del sistema. Por ejemplo, al trabajar con la detección de irregularidades en redes eléctricas basada en curvas de consumo, al utilizar los algoritmos de DAICE o DeepDAICE, existen diversas métricas relevantes para evaluar el desempeño de los modelos. Entre las métricas más relevantes se encuentran:

- F1: Es una métrica que combina la precisión y la recuperación en un solo valor, proporcionando una medida del equilibrio entre ambas métricas.
- PR_AUC (Área bajo la curva precisión-recall): Similar al PR_AUC, esta métrica evalúa la capacidad del modelo para clasificar correctamente las muestras positivas y negativas.
- ROC_AUC (Área bajo la curva ROC): Esta métrica evalúa la capacidad del modelo para discriminar entre clases positivas y negativas, considerando la tasa de verdaderos positivos y la tasa de falsos positivos.

Además de estas métricas, también se consideran otras métricas clásicas como la exactitud (accuracy), la precisión (precision) y la recuperación (recall). Estas métricas brindan información adicional sobre el rendimiento del modelo en diferentes aspectos.

6.1.2. Evaluación de Modelos con MLflow: Enfoque en Precision-Recall y ROC

Al profundizar en nuestro estudio, MLflow surge como una herramienta importante para recuperar artefactos generados durante la evaluación de modelos, como por ejemplo, archivos csv que contengan valores de precision y recall, además de información relevante que permita también realizar el análisis de la Curva Característica de Operación del Receptor (ROC). En las figuras 6.1 y 6.2, respectivamente, se pueden ver ejemplos de gráficos realizados a partir de estos datos mencionados anteriormente. Estas funcionalidades se manejan a través del módulo *experiment_analysis.py*, el cual se implementó para recuperar artefactos de MLflow y generar gráficas de precisión-recall para cada experimento o ejecución dentro del experimento denominado **DAICE_dag_train_eval**. Además, permite analizar y visualizar la eficacia de los modelos a través de las curvas ROC y precision-recall, que resultan fundamentales para evaluar modelos en casos de datos desbalanceados.

La integración del módulo de análisis a nuestro estudio permite una evaluación más completa del rendimiento de los modelos, permitiendo discriminar entre los diferentes modelos, cuál tiene mayor poder para separar las clases positivas de las negativas. Para saber que modelo tiene mejor rendimiento, típicamente se utiliza la gráfica de la figura 6.1, observando cuál acumula mayor área para seleccionarlo.

Esta dimensión adicional de análisis es crucial, especialmente en contextos donde la relación entre la sensibilidad y especificidad del modelo es un factor determinante para su eficacia práctica.

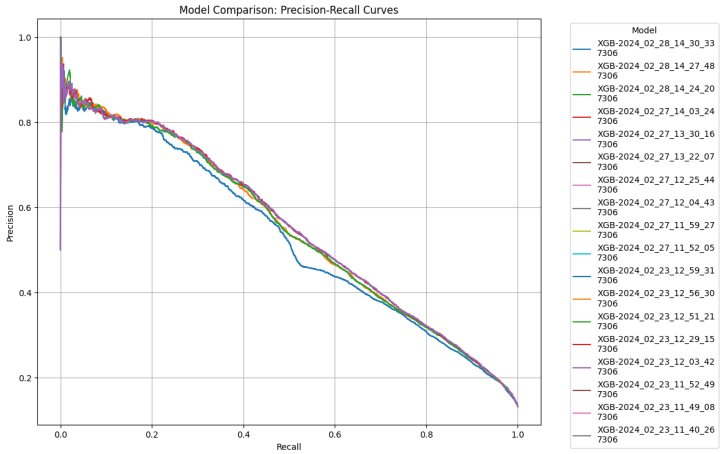


Figura 6.1: Gráficas de Precisión-Recall de Experimentos con MLflow

El proceso para generar estas gráficas se apoya en la lectura de los registros de MLflow, donde los valores predichos por los modelos y las etiquetas verdaderas de los conjuntos de datos de prueba se analizan para calcular la TPR y la FPR en varios umbrales. La capacidad para rastrear y visualizar estas métricas a lo largo de diferentes corridas y configuraciones de modelos subraya la robustez de MLflow como una herramienta integral para el monitoreo y la mejora continua del rendimiento de modelos en proyectos de Machine Learning.

La inclusión de este análisis enriquece nuestro caso de estudio de DAICE, proporcionando una perspectiva más holística del rendimiento de los modelos y destacando la versatilidad de MLflow y Airflow en la orquestación y evaluación de flujos de trabajo complejos en Machine Learning.

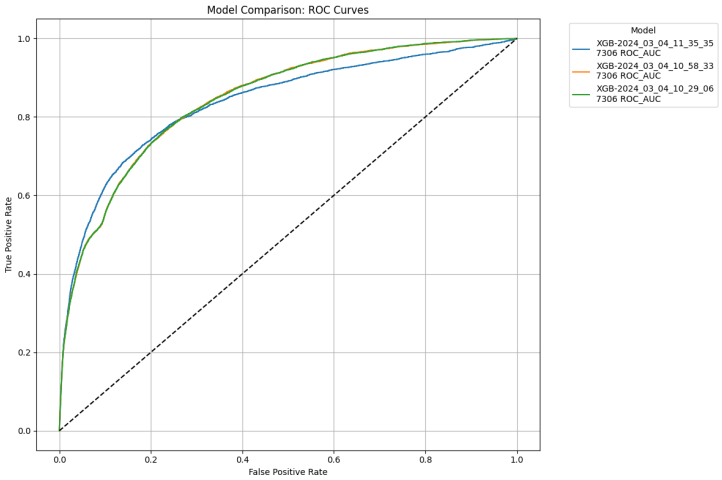


Figura 6.2: Gráficas de ROC_AUC de Experimentos con MLflow

6.1.3. Automatización del Pipeline de Detección de Irregularidades: Un Enfoque Modular

Un punto de partida adecuado para desarrollar el flujo de trabajo es pensar en un pipeline manual para la detección de irregularidades y automatizar cada etapa individualmente. A partir del pipeline manual, se necesitan automatizar cinco etapas: análisis de datos, preparación de datos, entrenamiento del modelo, prueba del modelo y, finalmente, el modelo final que potencialmente puede ser utilizado en producción. A continuación se describe como estás herramientas participan en cada etapa:

6.1.4. Análisis y Preparación de Datos

Para el análisis de datos, se emplean TaskFlow y PythonOperators [75][1] para gestionar el flujo de trabajo y las tareas relacionadas con el análisis de datos. Esto implica definir tareas específicas en Python que realizan el análisis necesario, como cargar los datos, realizar transformaciones, aplicar filtros y cualquier otro procesamiento requerido.

En lo que concierne a la preparación de datos, TaskFlow y PythonOperators también se utilizan para gestionar las tareas de preparación de datos. Esto implica definir tareas en Python que realicen las acciones necesarias para preparar los datos para su posterior procesamiento, como limpiar los datos, seleccionar características, normalizar los datos, entre otras tareas de preparación.

6.1.5. Entrenamiento, Prueba y Producción del Modelo

Luego, para el entrenamiento del modelo, la prueba del modelo y la puesta en producción, se utiliza MLflow en combinación con TaskFlow y PythonOperators. TaskFlow se encarga de orquestar las tareas relacionadas con el entrenamiento del modelo, la prueba del modelo y la puesta en producción, mientras que MLflow se utiliza para realizar el seguimiento, registro de los experimentos y las métricas de rendimiento, así como para gestionar los modelos y sus versiones. Las tareas en Python definidas utilizando PythonOperators pueden utilizar las funcionalidades de MLflow para entrenar los modelos, evaluar su rendimiento y desplegarlos en producción.

La puesta en producción de modelos se limita a guardar los modelos de ML como una función de Python en una ubicación fija. Además, estas herramientas no requieren instalaciones adicionales significativas y la mayoría de ellas ya se utilizan comúnmente para el desarrollo de software. Esto garantiza la reutilización y, para automatizar un antiguo script de entrenamiento del modelo, solo es necesario integrar la API de MLflow, configurar un nuevo repositorio de Git y determinar el pipeline de uso. Esto se puede hacer partiendo de un notebook de experimentación.

6.1.6. Hacia la Automatización en MLOps: Niveles de Madurez y Modularidad

En cuanto al nivel de automatización de un flujo de trabajo, también cabe destacar que un flujo de trabajo de MLOps se puede distinguir según el nivel de madurez del proceso de ciencia de datos implementado y la cantidad de usuarios finales que interactúan con el modelo de ML. Un nivel de madurez más bajo representa un pequeño equipo que trabaja con una ejecución manual del flujo de trabajo de ML (en la mayoría de los casos, un científico de datos individual).

[1] PythonOperator es una clase en Apache Airflow que permite ejecutar funciones de Python dentro de flujos de trabajo definidos en Airflow. Es útil para realizar tareas personalizadas mediante scripts Python y permite una gran flexibilidad en la ejecución de código arbitrario. Más detalles en: `https://airflow.apache.org/docs/apache-airflow/stable/howto/operator/python.html`

El primer paso para adoptar MLOps para este tipo de casos de uso es automatizar el proceso de entrenamiento de un nuevo modelo tan pronto como haya nuevos datos disponibles para producción. Esto implica ejecutar automáticamente el pipeline de ML con pasos como la evaluación automatizada de la calidad de los datos y las pruebas del modelo. El problema a resolver es principalmente cómo automatizar la ejecución de un nuevo pipeline de entrenamiento y cómo entregar el modelo resultante si es que este supera las pruebas de calidad según sus métricas.

Scripts de Inicio y Finalización

Para mejorar la automatización y simplificar la ejecución de los servicios de Airflow y MLflow para levantar el sistema, se realizó la implementación de un conjunto de scripts (*start_services.sh* y *stop_services.sh*) que permiten dar inicio y fin a la ejecución de todos los servicios necesarios para ejecutar. Esto elimina la necesidad de ejecutar cada proceso de forma manual y en diferentes consolas, como se tuvo que hacer en gran parte del proyecto. Con estos scripts no es necesario activar pipenv [76] de manera manual.

6.1.7. Integración con la Industria 4.0

La transición hacia una mayor automatización y la adopción de prácticas de MLOps se alinean estrechamente con los principios fundamentales de la Industria 4.0, que promueve la digitalización y la interconexión de procesos industriales. En este contexto, la implementación de soluciones MLOps, como se describe en este capítulo, no solo representa un avance en el campo específico de la detección de irregularidades en redes eléctricas, sino que también es un reflejo de una tendencia más amplia hacia la automatización inteligente y la optimización de procesos.

La capacidad de MLOps para facilitar el desarrollo, despliegue y monitoreo de modelos de aprendizaje automático de forma automatizada es un componente clave. Esto se debe a que permite la integración de sistemas inteligentes de análisis de datos y toma de decisiones en tiempo real, mejorando así la eficiencia operativa, la calidad del software.

La adopción de estos enfoques en el marco del proyecto DAICE y DeepDAICE, como su potencial aplicación en otros contextos industriales, resalta la importancia de la convergencia entre las tecnologías de la información.

6.1.8. Implementación de Pipelines Modulares en Airflow

Se implementaron algunos pipelines en la herramienta Airflow, intentando respetar la arquitectura definida en el capítulo 5. El motivo por el cual se implementaron los pipelines en Airflow es la posibilidad separar las diferentes etapas del proceso y asegurar un flujo de trabajo más modular. Cada uno de los DAGs implementados, cubre una o más de las funcionalidades provistas por cada pipeline descrito en el capítulo de arquitectura. La implementación de diferentes

flujos en Airflow, entendemos que contribuye a alcanzar el nivel 1 de MLOps. El nivel 1 se refiere a la automatización parcial del proceso de desarrollo de machine learning, y se caracteriza por tener una integración básica de los componentes de MLOps. Cada pipeline se enfoca en una tarea específica, lo que permite un mayor control y flexibilidad en el desarrollo y ejecución de las tareas.

Desglose de Pipelines Específicos

- **data_management_dag**

  Este DAG permite generar un archivo en formato CSV de los datos, leídos directamente de una base de datos Oracle[2], donde se mantienen los registros de las lecturas, haciendo de esta forma una extracción inicial de un conjunto de datos. En este flujo se extraen los datos de la base de datos, se juntan, se aplican las transformaciones necesarias y se realiza la limpieza de los datos antes de ser utilizados en pasos posteriores. Al separar esta etapa en un pipeline individual o modular, se facilita el mantenimiento del DAG y del sistema en general, así como la actualización del mismo, en caso de desear agregar más datos y que los mismos puedan ser utilizados por otros modelos de deep learning o machine learning clásico. La conexión directa a la base de datos relacional en el pipeline de extracción y limpieza de datos también contribuye al nivel 1 de MLOps. Permite el manejo de datos en tiempo real y asegura que los modelos se entrenen y clasifiquen utilizando datos actualizados. Esto es especialmente importante en el contexto de la detección de irregularidades en redes eléctricas, donde la detección temprana de errores es crucial para evitar gastos innecesarios.

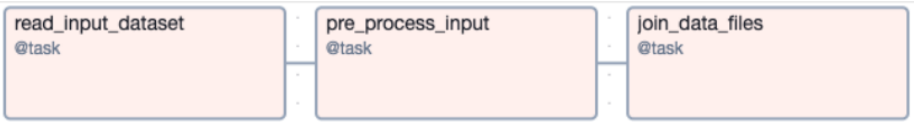


Figura 6.3: Vista del DAG **data_management** en Airflow

- **DAICE_train_eval_dag**

  Este DAG realiza la lectura de los parámetros a partir de la configuración. Luego realiza el preprocesamiento de los datos configurados, genera las *features* correspondientes, entrena y evalúa el modelo. Es decir, este DAG es la representación del pipeline que realiza la evaluación de un modelo previamente creado y registrado en el *Model Registry*. Es el DAG es la implementación que representa a DAICE de manera más completa.

- **DAICE_evaluate_sets_dag**

[2]La conexión a la base de datos implicó la instalación de un manejador de base de datos en WSL, lo cual sirvió como una prueba de concepto para validar la viabilidad.

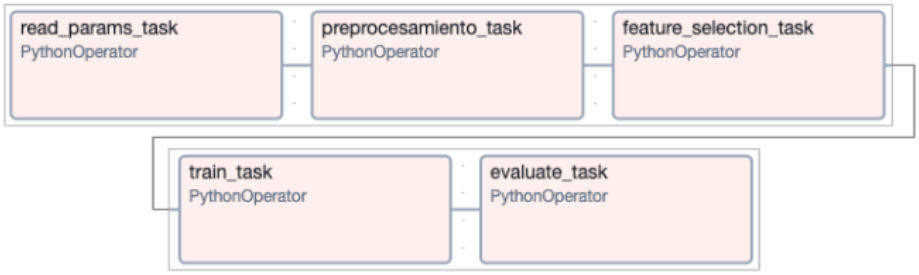


Figura 6.4: Vista del DAG **DAICE_train_eval_dag**

Con este grafo, se realiza la evaluación de los modelos en diferentes conjuntos de datos, en particular una de las formas de segmentación utilizadas en el trabajo fue la de regiones particulares de Uruguay como Norte, Centro, Este, entre otras.

- **DAICE_classification_dag**

  Este DAG permite realizar clasificaciones a partir de distintos modelos previamente entrenados.

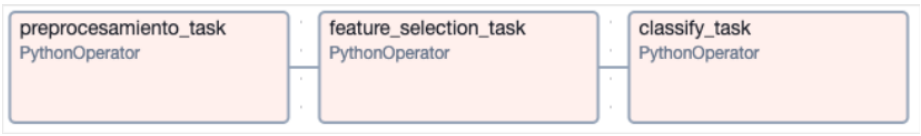


Figura 6.5: Vista del DAG **DAICE_classification_dag**

Al dividir el flujo de trabajo en pipelines separados, como se definió en nuestra propuesta de arquitectura, se establece una estructura clara y definida para cada etapa del proceso. Esto facilita la gestión y el seguimiento de cada paso, así como la identificación de problemas en el proceso. Cada pipeline puede ejecutarse de manera independiente, permitiendo además la reutilización de tareas y la ejecución paralela de múltiples pipelines si es necesario. Esta modularidad no solo optimiza los procesos, sino que también prepara el terreno para la integración de prácticas estandarizadas en el ámbito de MLOps.

6.1.9. Integración entre herramientas

Como se mencionó en la sección 5.2, para cumplir con los requerimientos se definió combinar ambas herramientas para aprovechar las mejores funcionalidades de cada una. La integración de Airflow con MLflow fue posible mediante el uso conjunto de ambas librerías en uno de los nodos del pipeline. En particular, esta combinación se realiza dentro del artefacto **evaluate_task** en el DAG **DAICE_train_eval_dag** (último artefacto o step en la figura 6.4). Los datos

registrados pueden ser visualizados en formato de tabla directamente en la vista web de MLflow (ver figura 6.11).

6.1.10. Estandarización de Pipelines en MLOps Mediante Archivos de Configuración

Como una característica interesante de Airflow, queremos destacar que es posible ejecutar cada DAG de dos maneras distintas (ver figura 6.6): por defecto o con configuraciones específicas en formato JSON. Esta segunda forma nos da más flexibilidad, ya que podemos definir distintos parámetros a utilizar en un archivo JSON en la herramienta de Airflow, y que sean esos los parámetros utilizados durante la ejecución del pipeline. Además, para cada DAG podemos definir un archivo JSON por defecto que será presentado (y podrá ser modificado) antes de realizar la ejecución del DAG.

Figura 6.6: Vista de ejecución del DAG con y sin configuración

Esto nos brinda la practicidad de cambiar los distintos parámetros antes de iniciar la ejecución y sin la necesidad de realizar cambios en el código.

Estos archivos contienen la información necesaria para cada tarea y pipeline, lo que simplifica la configuración y puede ayudar a evitar errores manuales. Al utilizar un sistema de control de versiones, como por ejemplo Git, para gestionar estos archivos de configuración, se puede lograr la trazabilidad y reproducibilidad de los pipelines.

Para facilitar la configuración y ejecución de los pipelines en Airflow, se utiliza un archivo YAML de configuración. Este archivo contiene los detalles de las tareas, sus dependencias y otros parámetros relevantes para cada pipeline. Asimismo, se utiliza un archivo JSON de entrada de pipeline al ejecutar el DAG directamente en Airflow.

Al seguir un flujo de trabajo estructurado y automatizado, se puede lograr una mayor eficiencia y confiabilidad en la detección de irregularidades. Es importante tener en cuenta que este caso de uso es solo un punto de partida y se puede adaptar y personalizar según los requisitos y las herramientas disponibles.

Dada la complejidad incremental del entorno de desarrollo, también existe una dependencia de los datos. Es necesario contar con información sobre el entorno de desarrollo, el conjunto de datos y los hiperparámetros para poder reproducir el mismo modelo. A medida que aumenta el número de experimentos, la cantidad de información que se debe almacenar aumenta de forma dramática.

Toda esta información difícilmente sea posible registrarla manualmente, por lo que es necesario contar con una herramienta que facilite el trabajo del científico de datos.

El flujo de trabajo que fue implementado brinda al equipo de desarrollo la libertad de probar nuevos modelos sin cambiar otras partes, permitiendo trabajar de forma modular y modificando solamente la caja que se quiera actualizar dentro del grafo. A través del componente de registro de MLflow, es posible guardar las diferentes etapas y versiones de desarrollo para un solo modelo. Esta información ayuda en la depuración cuando un modelo en particular falla en el entorno de implementación. A través del flujo de trabajo utilizado, los modelos pueden mejorar continuamente, lo que facilita la complejidad del proceso de desarrollo de aprendizaje automático.

En el flujo de trabajo usado, la vuelta atrás a la versión anterior de los modelos se realiza a través de la API de MLflow que muestra como correr el modelo seleccionado (Ver figura 6.9). Anteriormente, dado que el producto final era solo un modelo y no un pipeline o artefactos de MLflow, el monitoreo del modelo se hacía revisando notebooks o archivos de texto en lugar de plataformas como MLflow y Airflow.

Make Predictions

Predict on a Spark DataFrame:

```
import mlflow
from pyspark.sql.functions import struct, col
logged_model = 'runs:/d4739d1b49574df7ae3d6d83e0cf4cf4/XGB'

# Load model as a Spark UDF. Override result_type if the model does not return double values.
loaded_model = mlflow.pyfunc.spark_udf(spark, model_uri=logged_model, result_type='double')

# Predict on a Spark DataFrame.
df.withColumn('predictions', loaded_model(struct(*map(col, df.columns))))
```

Predict on a Pandas DataFrame:

```
import mlflow
logged_model = 'runs:/d4739d1b49574df7ae3d6d83e0cf4cf4/XGB'

# Load model as a PyFuncModel.
loaded_model = mlflow.pyfunc.load_model(logged_model)

# Predict on a Pandas DataFrame.
import pandas as pd
loaded_model.predict(pd.DataFrame(data))
```

Figura 6.7: Ejecución de un modelo guardado anteriormente. Captura de pantalla de la interfaz de MLflow.

A medida que se avanza con la madurez en el equipo de ML, es posible enfocarse más en la contenerización y la automatización completa de los flujos de

trabajo de aprendizaje automático, además de la automatización de los flujos de trabajo de CI/CD. Esto también se conoce como el nivel final de automatización (MLOps Nivel 2). Con este paso, se pueden implementar múltiples pipelines como una única entidad en lugar de un modelo individual. Estos pipelines se pueden versionar. Con la contenerización, entran en juego los orquestadores de contenedores como Kubernetes. Todos estos elementos son importantes para gestionar el flujo de trabajo en evolución. Para lograr flujos de trabajo de MLOps continuos y escalables, los contenedores, los orquestadores y los administradores de recursos forman una herramienta esencial. Con estas herramientas, se logran flujos de trabajo de ML continuos y escalables.

Es precisamente esta aspiración a alcanzar un nivel más alto de madurez en MLOps lo que nos lleva a considerar la migración del sistema DAICE a tecnologías más robustas como Airflow y MLflow.

6.2. Creación de pipeline de AA usando Airflow

Airflow desempeña un papel significativo en la etapa inicial del pipeline de aprendizaje automático, es decir, en la ingestión de datos. Se muestra como una plataforma que permite programar y monitorizar flujos de trabajo, Airflow puede automatizar y gestionar los procesos de recolección y validación de datos. De este modo, se garantiza que los datos ingresados al pipeline son consistentes, confiables y están listos para las etapas posteriores. Airflow también es capaz de gestionar tareas dependientes en flujos de trabajo complejos o *Directed Acyclic Graph* (DAG), lo que permite un control eficiente del proceso de validación de datos [27].

Para ejemplificar, en la figura 6.8 se muestra un pipeline sencillo de 5 steps, que representa uno de los DAG que forman parte de la solución diseñada para DAICE.

Figura 6.8: Vista de un DAG en Airflow

Apache Airflow se presenta como una de las principales herramientas para orquestar y gestionar pipelines de aprendizaje automático, ya que proporciona una interfaz de usuario clara y completa que permite la visualización del pipeline de aprendizaje automático. A través de esta interfaz, es posible monitorizar la totalidad de los pipelines activos o no (ver figura 6.9), así como cada ejecución de experimentos individual junto con sus respectivas fechas de ejecución. Un aspecto crucial de Airflow es su capacidad para ilustrar el desarrollo incremental del pipeline. Con el paso del tiempo y la adición de más componentes, como se muestra en la figura 6.10, el pipeline se puede hacer tan complejo como se desee, y es posible observar este crecimiento de forma gráfica.

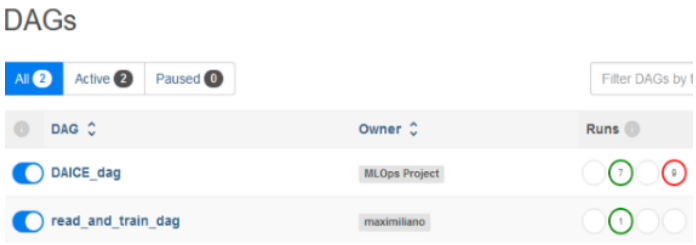


Figura 6.9: Interfaz de Airflow que permite el monitoreo de todos los pipelines activos e inactivos, incluyendo la visualización individual de cada experimento ejecutado junto con sus respectivas fechas de ejecución.

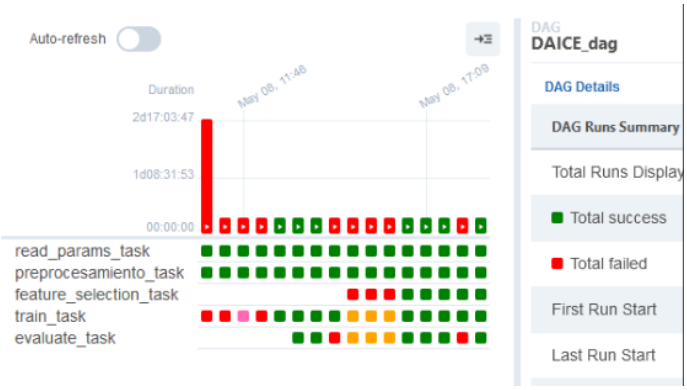


Figura 6.10: Desarrollo incremental del pipeline en Airflow. Con el paso del tiempo y la adición de más componentes, Airflow ilustra visualmente estos cambios, proporcionando un seguimiento efectivo del progreso y la evolución del pipeline.

6.3. Uso de MLflow para trazabilidad de experimentos, parámetros y métricas

MLflow se destaca en su habilidad para tratar con varias etapas de un pipeline de aprendizaje automático, desde la experimentación hasta la implementación del modelo. MLflow permite la trazabilidad de los modelos y la reproducibilidad de los experimentos, lo que facilita la implementación de los modelos en el ámbito de producción. Además, una de las características destacables de MLflow es su capacidad para rastrear métricas y parámetros, lo que permite a los científicos de datos comparar diferentes experimentos para seleccionar el modelo más eficaz. Ver la Figura 6.11 para una demostración de cómo MLflow registra y rastrea los datos.

					Metrics			
Run Name	Created	Duration	Source	Models	FP	TOTAL	accuracy	fvalue
XGB-2023_05_08_15_19_44	3 minutes ago	6.7s	airflow	XGB/11	23950	23950	0.089	0.089
XGB-2023_05_08_15_08_33	14 minutes ago	0.5s	airflow	-	-	-	-	-
XGB-2023_05_08_14_48_01	35 minutes ago	4.2s	airflow	XGB/10	23950	23950	0.089	0.089
XGB-2023_05_08_14_12_06	1 hour ago	4.0s	airflow	XGB/9	23950	23950	0.089	0.089
XGB-2023_05_08_11_27_21	3 hours ago	4.0s	airflow	XGB/8	23950	23950	0.089	0.089
XGB-2023_05_08_10_58_15	4 hours ago	4.0s	evaluate....	XGB/7	23950	23950	0.089	0.089
XGB-2023_05_08_10_56_33	4 hours ago		D:\sourc...	-	23950	23950	0.089	0.089
XGB-2023_05_08_10_53_07	4 hours ago	4.4s	evaluate....	XGB/6	23950	23950	0.089	0.089
XGB-2023_05_08_10_49_24	4 hours ago	5.1s	evaluate....	XGB/5	23950	23950	0.089	0.089

Figura 6.11: MLflow y su capacidad para rastrear métricas y parámetros, lo que permite a los científicos de datos comparar diferentes experimentos para seleccionar el modelo más eficaz.

6.4. Migración del sistema DAICE a Airflow y MLflow

Partiendo de un código en Python de Machine Learning, se fueron desacoplando los diferentes pasos como el preprocesamiento, el entrenamiento y la evaluación del los modelos, por mencionar algunos, para pasarlos a un formato de pipeline. Lo cual implicó hacer una serie de cambios en el código existente, para lograr modularizar el proceso. De esta forma se pudo poner a prueba la plataforma elegida con la finalidad de lograr una aproximación a MLOps Nivel 1.

Se encontró especial dificultad en tener que usar 3 consolas de Windows Subsystem for Linux para ejecutar el Airflow Scheduler, Airflow WebServer y MLflow. Además, para corregir errores de código, se vio que el proceso fue más engorroso.

Sin embargo, posteriormente se pudo hacer todo usando Visual Studio Code[77], allí se pueden abrir todas las consolas en una sola plataforma, que además permite ejecutar el WSL con Ubuntu de Windows, de esta forma fue mucho más sencillo depurar errores en el código, como también el del armado del pipeline.

Para registrar métricas del modelo se agregan librerías de MLflow al módulo de evaluación de modelos, si bien existe la posibilidad de registrar métricas luego de haber entrenado el modelo.

La métrica PR_AUC se tiene en consideración para decidir si el modelo actual es mejor que el último modelo generado, y se hace en el módulo de entrenamiento, si bien sería más adecuado hacerle un nuevo módulo que se encargue del "Model Blessing". Esta métrica es recuperada utilizando funciones de MLflow, ya que al guardar un nuevo modelo, junto a este también se almacenan sus métricas.

Es este enfoque en la robustez y fiabilidad del modelo lo que nos lleva a la siguiente sección, donde aplicamos estas consideraciones a un caso práctico. En particular, veremos cómo estas métricas y prácticas podrían implementarse en el análisis de datos de taxis de Chicago

para asegurar resultados confiables.

6.5. Exploración de flujos de trabajo de aprendizaje automático de punta a punta con TFX: un caso de estudio sobre datos de taxis de Chicago

En este capítulo, se muestra como aplicamos las metodologías y herramientas de MLOps discutidas en los capítulos anteriores a un caso práctico: el análisis de datos de taxis de Chicago [78]. Este caso ilustra cómo las técnicas de MLOps pueden abordar problemas del mundo real en el ámbito del transporte urbano.

El objetivo de este caso de estudio fue mostrar cómo la integración de herramientas como Airflow y TFX puede automatizar el proceso de análisis de datos, desde la ingesta hasta la visualización y evaluación de modelos. En la figura 6.12 se puede ver como se combinan ambas herramientas, donde *AirflowDagRunner* es la definición del grafo para Airflow y la función que crear el pipeline es referente a TFX.

```
DAG = AirflowDagRunner(
    AirflowPipelineConfig(_airflow_config)
  ).run(
    _create_pipeline(
        pipeline_name=_pipeline_name,
        pipeline_root=_pipeline_root,
        data_root=_data_root,
        module_file=_module_file,
        serving_model_dir=_serving_model_dir,
        metadata_path=_metadata_path,
        beam_pipeline_args=_beam_pipeline_args))
```

Figura 6.12: Código para definir el pipeline en Airflow combinado con la plataforma TFX.

Es decir, utilizamos un pipeline de MLOps que integra Airflow para la orquestación de tareas y TFX (ver figura 6.13). Se puede apreciar como el pipeline presenta una gran complejidad y viene con una configuración ya establecida en la documentación de Google, pero puede y debe ser modificado o personalizado para aplicarlo al problema específico que se quiera atacar. Este enfoque asegura que el análisis sea reproducible y escalable, ya que puede ser paralelizado en diferentes servidores.

Los datos de taxis de Chicago ofrecen un rico conjunto de información que incluye tarifas, rutas y tiempos de viaje. Este conjunto de datos presenta desafíos y oportunidades únicas para aplicar técnicas de MLOps, como la automatización del flujo de trabajo y el seguimiento de métricas. Si bien los componentes ya vienen predefinidos en el tutorial [79], a partir de estos es posible definir y/o

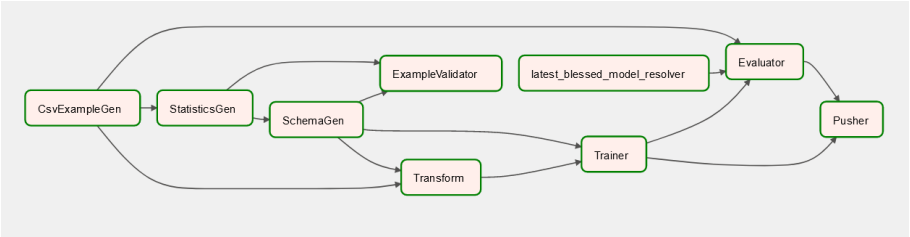


Figura 6.13: Flujo de trabajo de Airflow con TFX para el caso de estudio de datos de taxis de Chicago.

modificar el grafo de procesamiento o las diferentes operaciones que se aplican a nivel de cada nodo.

La definición del pipeline se hace llamando los componentes previamente definidos y usando la función *_create_pipeline*, en la cual se definen los diferentes componentes del pipeline, como se muestra en el código de la figura 6.14.

Inicialmente, lo ejecutamos directamente en Windows, el siguiente paso fue correr Airflow en la plataforma de WSL, ya que Airflow es nativo para Linux. También pudimos implementar y ejecutar con éxito los artefactos de CsvExampleGen, StatisticsGen, SchemaGen, ExampleValidator, Transform, Trainer, Resolver, EvalConfig, Evaluator y Pusher para el ejemplo de los taxis. También se pudo correr TFX en el **cluster de supercomputación**[80] logrando procesar los datos de entrada, lo cual hizo que el proceso fuera mucho más rápido que de forma local en una PC.

La experimentación y análisis de TFX mostraron idoneidad para cubrir múltiples aspectos en el proceso de desarrollo de modelos de machine learning. En primer lugar, TFX ofreció herramientas (como se mencionó en 3.1.5) para el preprocesamiento y la transformación de datos. Se utilizó TensorFlow Data Validation (TFDV) y TensorFlow Transform (TFT) para realizar análisis estadísticos, identificar anomalías y validar el esquema de los datos del conjunto de Chicago Taxi.

Además, TFX permitió abordar la problemática del versionado, ya que genera archivos comprimidos en formato de registros de TFX. Además, permite el manejo de datos mediante la integración de TensorFlow Data Validation y TensorFlow Metadata (TFMD). Estas herramientas permitieron almacenar metadatos relevantes sobre los datos utilizados, incluyendo información sobre el esquema, estadísticas y versiones. Esta funcionalidad facilitó el seguimiento de cambios en los datos y aseguró la consistencia en los flujos de trabajo durante las pruebas realizadas con el conjunto de datos de Chicago Taxi.

La plataforma TFX también ofreció capacidades para el seguimiento y empaquetado del modelo. Se utilizó TensorFlow Model Analysis (TFMA) para evaluar y comparar el rendimiento de los modelos utilizando métricas específicas, en el contexto de los casos de prueba de Chicago Taxi.

Por último, TFX se ocupó del monitoreo tanto de los datos de entrada como

```
def _create_pipeline(
        pipeline_name: str,
        pipeline_root: str,
        data_root: str,
        module_file: str,
        serving_model_dir: str,
        metadata_path: str
        # beam_pipeline_args: List[str] = ''
    ) -> pipeline.Pipeline:
    data_root_runtime = data_types.RuntimeParameter(
        'data_root', ptype=str, default=data_root)
    example_gen = CsvExampleGen(input_base=data_root_runtime)
    statistics_gen = StatisticsGen(...)
    schema_gen = SchemaGen(...)
    example_validator = ExampleValidator(...)
    transform = Transform(...)
    trainer = Trainer(...)
    model_resolver = resolver.Resolver(...).with_id(...)
    eval_config = tfma.EvalConfig(...)
    evaluator = Evaluator(....)
    pusher = Pusher(...)
    return pipeline.Pipeline(
        pipeline_name=pipeline_name,
        pipeline_root=pipeline_root,
        components=[
            example_gen, statistics_gen, schema_gen,
            example_validator, transform,
            trainer, model_resolver, evaluator, pusher
        ],
        enable_cache=True,
        metadata_connection_config=metadata.
          sqlite_metadata_connection_config(
            metadata_path
        ),
    )
```

Figura 6.14: Código para definir el pipeline en TFX usando la función _create_pipeline.

del rendimiento de los modelos durante las pruebas realizadas con el conjunto de datos. Con la ayuda de TensorFlow Data Validation, se detectaron y manejaron anomalías en los datos de entrada, garantizando la calidad de los mismos durante las pruebas. TensorFlow Model Analysis permite evaluar de manera continua el rendimiento de los modelos, detectando posibles desviaciones.

6.6. Caso de uso de TFX para DeepDAICE

La transición de la familiarización inicial con TFX, mediante el ejemplo Penguin Simple, a la implementación avanzada en el caso de los taxis de Chicago, pavimentó el camino para una exploración más profunda en el marco de DeepDAICE. Con un mayor dominio de TFX y Airflow, se abrió la oportunidad de

desafiar estas plataformas y buscar la adaptación del caso de uso de DeepDAICE con este conjunto de herramientas tecnológicas. La elección inicial de un formato de datos con 8640 columnas, representando 90 días de consumos quinceminutales, marcó el inicio de un enfoque estructurado para probar las capacidades y limitaciones en el manejo de grandes conjuntos de datos y la eficacia de TFX y Airflow en un escenario de análisis de series temporales a gran escala.

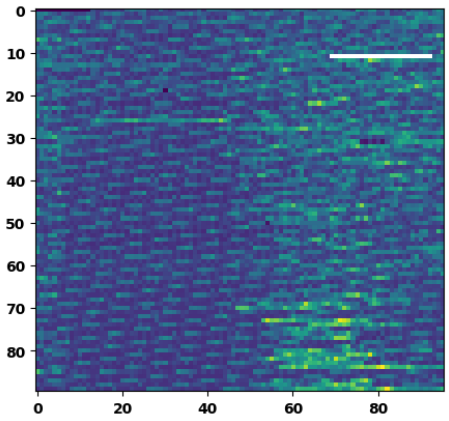


Figura 6.15: Ventana de consumos de 90 días graficada como matriz.

Luego de una investigación más exhaustiva, decidimos cambiar al formato de TensorFlow Records, el cual nos permitió almacenar las características de las curvas de consumo como matrices (en la figura 6.15 se observa la matriz de consumos en una ventana de 90 días), lo cual se ajusta a nuestras necesidades y a lo que ya se venía utilizando en el proyecto de DeepDAICE. Además, encontramos que el uso de TensorFlow Records tenía ventajas significativas de rendimiento en comparación con la separación de la matriz de datos en miles de columnas individuales.

Durante la implementación con TFX, se exploraron varios formatos de datos con el fin de evaluar tanto la herramienta como la eficiencia en el procesamiento. Después de realizar diferentes pruebas, se decidió utilizar TensorFlow Records por su rendimiento superior. En una de las pruebas, ejecutamos el componente CsvExampleGen con un conjunto de 34.474 instancias, lo que tomó aproximadamente 95 minutos en una computadora de escritorio[3].

TensorFlow Records utiliza un enfoque más optimizado para el almacenamiento y acceso a los datos en forma de matrices, lo que permite un procesamiento más eficiente a través de operaciones vectorizadas y los datos que se presentan en un formato comprimido. Lo cual, nos brinda beneficios adicionales en términos de espacio en disco.

[3]El equipo utilizado para las pruebas fue un HP ProDesk 400 G7 Small Form Factor PC, con un procesador Intel(R) Core(TM) i5-10500 CPU @ 3.10 GHz (12 núcleos), 16 GB de RAM, y sistema operativo Windows 11 Pro 64 bits.

El almacenamiento de los datos en formato comprimido y matricial facilita la transferencia de datos entre diferentes etapas del flujo de trabajo. También, es posible aprovechar las capacidades de paralelismo de TensorFlow para procesar grandes volúmenes de datos de forma más eficiente.

Durante el proceso de desarrollo, nos enfrentamos a dificultades para encontrar versiones compatibles de TFX y Airflow que funcionaran correctamente en el entorno de Windows Subsystem for Linux (WSL). Esto resultó en fallas en algunos módulos durante las pruebas realizadas. Además, la instalación de TFX y Airflow lleva un tiempo considerable, lo que añadió complejidad y dificultad al proceso. Además, a pesar de la investigación en la web y en la literatura, no encontramos información clara sobre las versiones específicas de estas librerías que fueran compatibles entre sí, lo que dificultó aún más la tarea de establecer un entorno de trabajo adecuado. Si bien, luego se encontró un repositorio con versiones compatibles para TFX y Airflow [81].

6.6.1. Componentes de TFX usados en DeepDAICE

Se aprovecharon los componentes de TFX para realizar tareas específicas relacionadas con el caso de uso de DeepDAICE. En particular, se utilizaron componentes de TFX para la generación de TensorFlow Records, la generación automática de esquemas y la validación de datos utilizando StatisticsGen.

La generación de TensorFlow Records se llevó a cabo utilizando el componente de TFX llamado TfExampleGen. Este componente permitió convertir los datos de entrada en el formato adecuado para su procesamiento en TensorFlow, lo que facilitó la carga y el procesamiento eficiente de los datos en el pipeline de DeepDAICE.

Además, TFX ofreció herramientas para la generación automática de esquemas mediante el componente SchemaGen, luego de generar el esquema de forma automática, se puede reutilizar el mismo esquema para validar futuras corridas del pipeline. Esta funcionalidad resultó especialmente útil para el caso de uso de DeepDAICE, ya que permitió definir el esquema esperado de los datos y validar automáticamente si los datos de entrada cumplían con dicho esquema. Esto garantizó la integridad y consistencia de los datos utilizados en el pipeline de entrenamiento y clasificación de DeepDAICE.

Por último, se utilizó el componente StatisticsGen de TFX para realizar la validación de datos. Este componente generó estadísticas descriptivas sobre los datos de entrada, permitiendo identificar posibles anomalías, valores atípicos o errores en los datos. La validación de datos utilizando StatisticsGen fue un paso crítico en el proceso de preprocesamiento y limpieza de los datos, asegurando la calidad y confiabilidad de los mismos para su uso en el pipeline de DeepDAICE.

6.7. Requerimientos cumplidos por la solución

Ahora vamos a ver como fueron cumplidos los requerimientos. Primero se va a describir como en una etapa inicial del proyecto se exploró la herramienta

TFX en conjunto con Airflow, en ejemplos sintéticos con conjuntos de datos genéricos y luego como se pudieron aplicar estas herramientas en algunas fases de DeepDAICE. Luego pasamos a hacer un repaso general acerca de todos los requerimientos cubiertos por la solución final.

6.7.1. Cumplimiento de los Requerimientos mediante el Uso de TFX, Airflow y MLflow

En la etapa previa a la migración de DAICE a la plataforma de Airflow con MLflow, se llevaron a cabo tareas que cubrieron los requerimientos relacionados con la verificación de calidad de los datos, el flujo de datos automatizado, el entrenamiento y evaluación del modelo, la comparación y registro de modelos, y la generación de un ranking de clasificación. Estas tareas se realizaron exitosamente utilizando las herramientas TFX, Airflow y MLflow.

Para la verificación automática de los datos para el conjunto de Chicago taxi y de DeepDAICE, se hizo con TFX. Se realizaron análisis de los datos provenientes de la base de datos relacional y de archivos en formato *csv* provenientes de una base de datos no relacional. TFX permitió verificar la validez de los datos mediante la consideración de información estadística y el tipo de datos involucrados.

La verificación del esquema de datos desde la base de datos se llevó a cabo utilizando TFX. Los artefactos de TFX permitieron analizar y validar el esquema de los datos. Esto garantizó que los datos cumplieran con las características esperadas y se consideraran válidos o inválidos con base en la información recopilada.

Asimismo, se verificaron los datos provenientes de archivos, utilizando los componentes de TFX. Estos componentes analizaron la información estadística y el tipo de datos para determinar la validez de los mismos.

La verificación del esquema de los datos ingresados también se realizó utilizando TFX. Se definieron columnas requeridas y tipos de datos correspondientes en el esquema, y se validaron los datos ingresados con base en este esquema.

Luego, a nivel del dataset de los taxis de Chicago, se llevó a cabo una verificación de la consistencia de los datos utilizados en producción, asegurando que se encontraran dentro de los rangos esperados y que no hubiera discrepancias significativas. Esta tarea también se realizó con la ayuda de TFX, que permitió realizar análisis comparativos y estadísticos de los conjuntos de datos utilizados en producción.

La verificación de sesgos entre los datos de entrenamiento y los datos utilizados para la inferencia se realizó mediante la comparación estadística de los conjuntos de datos. TFX y sus componentes, como ExampleValidator y Evaluator, se utilizaron para evaluar las diferencias de valores estadísticos y distribuciones.

En cuanto al flujo de datos automatizado y el entrenamiento del modelo, se utilizó TFX en conjunto con Airflow. TFX proporcionó los componentes necesarios, como ExampleGen, Transform, Trainer y Evaluator, para construir un flujo de datos automatizado que gestionó el preprocesamiento, entrenamiento y evaluación del modelo. Airflow permitió programar y ejecutar estas tareas en

un grafo dirigido acíclico, lo que garantizó un flujo eficiente y ordenado de los datos.

Para la comparación y registro de modelos, y el cálculo del rendimiento del modelo, se utilizó MLflow. Esta herramienta permitió comparar diferentes modelos, realizar experimentos y registrar métricas e hiperparámetros, como también el registro de valores en el archivo de configuración YAML. Se almacenaron los metadatos relacionados con la creación de los modelos, incluyendo los hiperparámetros utilizados y las métricas obtenidas durante los experimentos.

6.7.2. Requerimientos cubiertos en el software final

Para abordar de manera efectiva los desafíos identificados en nuestra investigación, hemos implementado una solución que cumple con una serie de requisitos esenciales. En el desarrollo de nuestra solución de aprendizaje automático, hemos abordado una serie de requisitos y tareas críticas que abarcan desde la gestión de datos hasta la interacción del usuario. Estos elementos son fundamentales para garantizar tanto la calidad como la eficacia de los modelos implementados. A continuación, se detallan los principales componentes y características de la solución, todos los cuales ya han sido implementados:

- **Tratamiento de Datos y Verificación desde Bases de Datos**

  Uno de los pilares de nuestra solución es el tratamiento riguroso de los datos, que es esencial para la eficacia de cualquier modelo de machine learning.

  - Se realizó una verificación exhaustiva de los datos provenientes de bases de datos no relacionales, se atacó el problema inicialmente con el formato H5, pero luego se vio que dicho formato no soportaba un número muy elevado de claves. Por ese motivo se terminó usando archivos individuales *csv*.
  - El uso de artefacto de TFX como fue CsvExampleGen permitieron, a partir de los archivos *csv* originales, generar un comprimido de registros de TFX para lograr así la ingesta correcta de los datos fundamental en el contexto de TFX.
  - La validez de los datos se aseguró mediante análisis estadísticos y validaciones de tipo de datos, dichas validaciones ya vienen dadas por el framework de TFX, lo cual no hizo necesario un abordaje profundo acerca de las técnicas de validación analizando si cumplen con las características esperadas. Esto permite tener en cuenta cambios potenciales en los procedimientos almacenados y consultas SQL, que podrían desencadenar errores.
  - Utilizamos TFX para la preparación y limpieza de los datos, aunque no se integró en la solución final, se pudo validar que TFX aplica al problema que nos enfrentamos. El proceso de preparación utilizado fue el que vino dado por TFX, si bien cada componente o artefacto es personalizable.

- **Orquestación de Flujos de Datos**
  - Para una gestión eficiente, los requerimientos se integraron en un flujo de datos automatizado.
  - Se utilizó un framework de orquestación para ejecutar componentes en secuencia en un grafo dirigido acíclico.
- **Gestión de Modelos**
  - Utilizamos Airflow y MLflow para el seguimiento, versionado y orquestación de los modelos.
- **Implementación del Modelo**
  - Utilización de modelos predefinidos o desarrollados previamente. Con la plataforma MLflow fue posible utilizar los modelos existentes.
  - Se implementan algoritmos de aprendizaje automático usando librerías como XGBoost y TensorFlow.
- **Entrenamiento y Evaluación**
  - El pipeline del modelo incluye fases de entrenamiento, ajuste y optimización.
  - Se lleva a cabo una evaluación del modelo, comparando métricas para determinar su desempeño.
- **Validación del Modelo**
  - Comparación del modelo con otros en producción o modelos de referencia.
  - Utilización de A/B testing para evaluar eficacia y métricas adicionales.
- **Gestión de Modelos**
  - Implementación de un registro de modelos que guarda metadatos, incluidos hiperparámetros y métricas (*f1*, *precisión*, *recall*, *PR_AUC*, *ROC_AUC*, *FPR*, *TPR*).
- **Conexión con Base de Datos**
  - Conexión establecida con una base de datos no relacional que almacena información sobre medidores inteligentes.
  - Los datos recopilados son usados para entrenar y clasificar modelos relacionados con DeepDAICE.
- **Rendimiento y Métricas**
  - Cálculo y visualización de métricas en el subconjunto de datos de prueba.

- Métricas clave incluyen f1, precisión, recall, PR_AUC, ROC_AUC, FPR y TPR.

- **Interacción del Usuario**
  - Los usuarios pueden realizar predicciones usando diferentes modelos.
  - Se genera un ranking de clasificación basado en la probabilidad de irregularidades según el modelo usando el pipeline de clasificación.

Capítulo 7

Conclusiones y Trabajo Futuro

A lo largo de este proyecto, hemos abordado la compleja tarea de adaptar y aplicar soluciones MLOps en el contexto específico del sistema DAICE, centrándonos en la detección avanzada de irregularidades en redes eléctricas. A través de la integración de distintas herramientas como Airflow, MLflow y TFX, hemos demostrado cómo la orquestación de procesos, la trazabilidad de experimentos y la evaluación de modelos pueden automatizarse, y que pueden aportar mucho valor al intentar operacionalizar procesos que involucran Machine Learning.

Hemos enfrentado diversos desafíos, desde la integración de datos hasta la ejecución de pipelines complejos, que nos han llevado a explorar soluciones innovadoras en este ámbito. Entendemos que la flexibilidad y la modularidad de nuestro enfoque han sido fundamentales para abordar los problemas inherentes a la gestión de grandes volúmenes de datos y la implementación de modelos de aprendizaje automático en un entorno de producción.

Adicionalmente, analizamos documentos actuales buscando las tendencias del mercado y entrevistamos a un referente de la empresa de e-commerce más grande de Uruguay. Durante esta entrevista, validamos que nuestra selección de herramientas está alineada con las utilizadas en la actualidad en la industria, y también que nuestra propuesta de arquitectura, donde se busca modularizar y tener distintos pipelines que resuelvan pequeñas porciones del problema para luego funcionar tanto individualmente como en conjunto, también agrega un valor importante para la generalización y reutilización, lo cual entendemos que valida nuestra solución aún más.

7.1. Resultados Alcanzados

Durante el presente proyecto, se pudieron alcanzar los siguientes resultados:

- Análisis y procesamiento automatizado de datos: En el marco del proyecto se utilizó con éxito TFX para realizar ingesta, validación, evaluación, y preparación de los datos.

- Validación y comparación de modelos: Se pudo integrar con éxito la plataforma MLflow, de esta forma se pudo registrar con éxito los diferentes experimentos, diferentes versiones de modelos, métricas y artefactos. Donde cada experimento tenía un modelo asociado con sus métricas.

- Modularización de la solución para DAICE: El equipo logró refactorizar el código fuente de DAICE a una versión modularizada, facilitando así su pasaje a un pipeline de Airflow.

- Integración de Airflow con TFX: Aunque no se integró completamente a la solución final, durante el proyecto se probó conceptualmente, logrando así validar la conjunción de estas tecnologías, como una posible solución de punta a punta para soluciones de Deep Learning, o más complejas.

- Implementación de pipelines: Se desarrollaron distintos pipelines individuales para los procesos de entrenamiento, clasificación y extracción de datos, incluso logrando el manejo de datos en tiempo real. Es decir, obteniendo los datos directamente de la base de datos. Usando la interfaz gráfica de Airflow, es posible visualizar la ejecución de cada DAG junto con el estado de cada paso que lo integra, el tiempo de ejecución de cada uno, logs, entre otros.

- Generación de valor en producción: Algunos de los pipelines desarrollados como parte del proyecto, actualmente son utilizados en producción por parte del Departamento de Recuperación de Energía de UTE, por lo que este proyecto no solo plantea un marco teórico donde se intenta explicar que es MLOps y no solo se plantea una arquitectura que pueda ser utilizada en una gran cantidad de sistemas que utilicen Machine Learning, sino que además, se generó valor a la empresa.

7.2. Trabajo Futuro

- Desarrollo de módulos de integración de datos: Crear módulos flexibles para la integración de datos desde diversas fuentes hacia TFX, mejorando la adaptabilidad como la variedad del proceso de agregación de datos.

- Mejora en la extracción y transformación de datos: Implementar un módulo especializado dentro del pipeline de TFX para la extracción y transformación eficiente de datos, particularmente desde HBase y otras fuentes.

- Exploración de contenedores y orquestadores: Profundizar en el uso de contenedores (como Docker) u orquestadores (como Kubernetes) para mejorar

la escalabilidad y la gestión de los pipelines de MLOps, facilitando la implementación, el despliegue y principalmente el escalado de las soluciones en diferentes entornos.

- Automatización completa y CI/CD: Avanzar hacia una automatización completa de los flujos de trabajo de aprendizaje automático, con la integración de prácticas de CI/CD para agilizar el desarrollo como la implementación de modelos.
- Análisis de casos de uso adicionales: Aplicar las metodologías y herramientas de MLOps a otros casos de uso y dominios, ampliando el impacto y la aplicabilidad de las soluciones desarrolladas.

7.3. Reflexiones Finales

Este trabajo representa un paso hacia la madurez en la implementación de MLOps, alineándose con las tendencias actuales en la Industria 4.0 y abriendo caminos para futuras investigaciones y aplicaciones en el ámbito de la inteligencia artificial aplicada a las redes eléctricas. Entendemos que las soluciones propuestas ofrecen una base sólida para continuar avanzando en la optimización de los procesos de detección de irregularidades, con la adopción de prácticas de MLOps, y que esto puede ser extendido a muchas otras áreas o dominios de aplicación, donde se utilicen o piensen utilizar, soluciones de Machine Learning.

Bibliografía

[1] L. Faubel, K. Schmid y H. Eichelberger, "MLOps Challenges in Industry 4.0," *SN Computer Science*, vol. 4, n.º 6, pág. 828, 2023.

[2] D. Rothman, *Transformers for Natural Language Processing: Build innovative deep neural network architectures for NLP with Python, PyTorch, TensorFlow, BERT, RoBERTa, and more*. Packt Publishing Ltd, 2021.

[3] A. B. Kolltveit y J. Li, "Operationalizing Machine Learning Models: A Systematic Literature Review," en *Proceedings of the 1st Workshop on Software Engineering for Responsible AI*, 2023, págs. 1-8, ISBN: 9781450393195. DOI: 10.1145/3526073.3527584. dirección: https://doi.org/10.1145/3526073.3527584.

[4] D. Kreuzberger, N. Kühl y S. Hirschl, "Machine Learning Operations (MLOps): Overview, Definition, and Architecture," *IEEE Access*, vol. 11, págs. 31 866-31 879, 2023. DOI: 10.1109/ACCESS.2023.3262138.

[5] P. Massaferro, J. M. Di Martino y A. Fernández, "Fraud Detection on Power Grids While Transitioning to Smart Meters by Leveraging Multi-Resolution Consumption Data," *IEEE Transactions on Smart Grid*, vol. 13, n.º 3, págs. 2381-2389, 2022.

[6] P. Massaferro, J. Matías Di Martino y A. Fernández, "NTL Detection: Overview of Classic and DNN-based Approaches on a Labeled Dataset of 311k Customers," en *2021 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, 2021, págs. 1-5. DOI: 10.1109/ISGT49243.2021.9372164.

[7] P. Glauner, J. A. Meira, P. Valtchev, R. State y F. Bettinger, "The challenge of non-technical loss detection using artificial intelligence: A survey," *arXiv preprint arXiv:1606.00626*, 2016.

[8] G. M. Messinis y N. D. Hatziargyriou, "Review of non-technical loss detection methods," *Electric Power Systems Research*, vol. 158, págs. 250-266, 2018, ISSN: 0378-7796. DOI: https://doi.org/10.1016/j.epsr.2018.01.005. dirección: https://www.sciencedirect.com/science/article/pii/S0378779618300051.

[9] P. Ruf, M. Madan, C. Reich y D. Ould-Abdeslam, "Demystifying mlops and presenting a recipe for the selection of open-source tools," *Applied Sciences*, vol. 11, n.º 19, pág. 8861, 2021.

[10] P. Massaferro, J. M. Di Martino y A. Fernández, "Fraud detection in electric power distribution: An approach that maximizes the economic return," *IEEE Transactions on Power Systems*, vol. 35, n.º 1, págs. 703-710, 2019.

[11] Microsoft. "Mlops: Machine learning model management - azure machine learning." (2022), dirección: https://learn.microsoft.com/en-us/azure/machine-learning/concept-model-management-and-deployment.

[12] Jenkins. "Jenkins: An open-source automation server." (jul. de 2024), dirección: https://www.jenkins.io/.

[13] GitHub. "GitHub: Where the world builds software." (jul. de 2024).

[14] G. Inc. "GitLab: The complete DevOps platform." Accessed: 2024-10-04. (2024), dirección: https://about.gitlab.com/.

[15] A. S. Foundation. "Apache Airflow." (2024).

[16] S. Das, K. Kellenberger, G. Fritchey y K. Abhishek. "Kubeflow for data scientists introduction." (abr. de 2021), dirección: https://www.red-gate.com/simple-talk/development/data-science-development/kubeflow-data-scientists/.

[17] A. W. Services. "Amazon AWS: Cloud Computing Services." Accessed: 2024-10-04. (2024), dirección: https://aws.amazon.com/.

[18] T. K. Authors. "Kubernetes: Production-Grade Container Orchestration." Accessed: 2024-10-04. (2024), dirección: https://kubernetes.io/.

[19] Databricks. "MLflow: An open source platform for the machine learning lifecycle." (2024).

[20] A. W. Services. "AWS SageMaker Model Registry." Accessed: 2024-10-04. (2024), dirección: https://docs.aws.amazon.com/sagemaker/latest/dg/model-registry.html.

[21] R. Fielding. "Representational State Transfer (REST)." Accessed: 2024-10-04. (2000), dirección: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

[22] J. Dean y S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." Accessed: 2024-10-04. (2004), dirección: https://research.google.com/archive/mapreduce.html.

[23] Google. "MLOPS: Continuous delivery and automation pipelines in machine learning, cloud architecture center, google cloud." (jul. de 2022), dirección: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning/.

[24] E. Raj, *Engineering Mlops: Rapidly build, test, and manage production-ready machine learning life cycles at scale*. Packt Publishing, 2021.

[25] K. Community. "Kubeflow: The Machine Learning Toolkit for Kubernetes." Accessed: 2024-10-04. (2024), dirección: `https://www.kubeflow.org/`.

[26] P. S. Foundation. "Python: A Programming Language That Lets You Work Quickly and Integrate Systems More Effectively." Accessed: 2024-10-04. (2024), dirección: `https://www.python.org/`.

[27] "Apache Airflow, a Platform Created by the Community to Programmatically Author, Schedule and Monitor Workflows." (2022), dirección: `https://airflow.apache.org/`.

[28] "A platform for the Machine Learning Lifecycle." (2022), dirección: `https://mlflow.org/`.

[29] E. Anello. "An intuitive guide to track your ML experiments with mlflow." (ago. de 2022), dirección: `https://towardsdatascience.com/an-intuitive-guide-to-track-your-ml-experiments-with-mlflow-7ac50e63b09` (visitado 08-2022).

[30] M. Team. "MLflow Tracking - Documentación oficial." Último acceso: octubre 2023. (2023), dirección: `https://mlflow.org/docs/latest/tracking.html`.

[31] M. Team. "MLflow Projects - Documentación oficial." Último acceso: octubre 2023. (2023), dirección: `https://mlflow.org/docs/latest/projects.html`.

[32] M. Team. "MLflow Models - Documentación oficial." Último acceso: octubre 2023. (2023), dirección: `https://mlflow.org/docs/latest/models.html`.

[33] M. Team. "MLflow Model Registry - Documentación oficial." Último acceso: octubre 2023. (2023), dirección: `https://mlflow.org/docs/latest/model-registry.html`.

[34] Databricks. "MLflow Recipes: Standardized Workflow for Developing Machine Learning Models." Accessed: 2024-10-04. (2024), dirección: `https://mlflow.org/docs/latest/recipes.html`.

[35] T. Team. "TensorFlow: An Open Source Machine Learning Framework for Everyone." Accessed: 2024-10-04. (2024), dirección: `https://www.tensorflow.org/`.

[36] S. Yegulalp. "What is tensorflow? The Machine Learning Library explained." (oct. de 2022), dirección: `https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html`.

[37] "Tensorflow extended (TFX): ML Production Pipelines." (2022), dirección: `http://www.tensorflow.org/tfx`.

[38] A. S. Foundation. "Apache Beam: An Advanced Unified Programming Model." Accessed: 2024-10-04. (2024), dirección: https://beam.apache.org/.

[39] T. Team. "TensorFlow Serving: Flexible, high-performance serving system for machine learning models." Accessed: 2024-10-04. (2024), dirección: https://www.tensorflow.org/tfx/guide/serving.

[40] T. Team. "TensorFlow Lite: Lightweight solution for mobile and embedded devices." Accessed: 2024-10-04. (2024), dirección: https://www.tensorflow.org/lite.

[41] P. Esposito, "Building Deep Learning Pipelines with tensorflow extended," *Medium*, jun. de 2022, Blog post. dirección: https://towardsdatascience.com/building-deep-learning-pipelines-with-tensorflow-extended-913869f1f051.

[42] "DVC: Open-Source Version Control System for Machine Learning Projects." (2022), dirección: https://dvc.org/.

[43] G. Community. "Git: Distributed Version Control System." Accessed: 2024-10-04. (2024), dirección: https://git-scm.com/.

[44] V. Driessen. "A successful Git branching model (GitFlow)." Accessed: 2024-10-04. (2010), dirección: https://nvie.com/posts/a-successful-git-branching-model/.

[45] H2O.ai. "H2O Open Source: Fast, Scalable Machine Learning for Everyone." Accessed: 2024-10-04. (2024), dirección: https://h2o.ai/platform/ai-cloud/make/h2o/.

[46] NVIDIA. "XGBoost: Open-Source, Scalable Machine Learning Library." (2024), dirección: https://www.nvidia.com/en-us/glossary/xgboost/ (visitado 14-10-2024).

[47] P. Inc. "Pachyderm: Data Versioning and Pipelines for Machine Learning." Accessed: 2024-10-04. (2024), dirección: https://www.pachyderm.com/.

[48] P. Inc. "Pachyderm Console Now Available to Community Edition Users." Accessed: 2024-10-04. (2024), dirección: https://www.pachyderm.com/blog/pachyderm-console-now-available-to-community-edition-users/.

[49] G. E. Team. "Great Expectations: Data Validation Made Easy." Accessed: 2024-10-04. (2024), dirección: https://greatexpectations.io/.

[50] Microsoft. "Microsoft SQL Server." Accessed: 2024-10-04. (2024), dirección: https://www.microsoft.com/en-us/sql-server.

[51] S. Community. "SQLAlchemy: The Database Toolkit for Python." Accessed: 2024-10-04. (2024), dirección: https://www.sqlalchemy.org/.

[52] G. Cloud. "BigQuery: Fully-Managed Data Warehouse." Accessed: 2024-10-04. (2024), dirección: https://cloud.google.com/bigquery.

[53] O. Corporation. "Oracle Database: Enterprise Data Management." Accessed: 2024-10-04. (2024), dirección: https://www.oracle.com/database/.

[54] "HBase: an open-source non-relational distributed database." (2022), dirección: https://hbase.apache.org/.

[55] "Apache Hadoop." (2022), dirección: https://hadoop.apache.org/.

[56] A. S. Foundation. "Apache Spark: Lightning-Fast Unified Analytics Engine." Accessed: 2024-10-04. (2024), dirección: https://spark.apache.org/.

[57] Microsoft. "Windows Subsystem for Linux (WSL)." Accessed: 2024-10-04. (2024), dirección: https://learn.microsoft.com/en-us/windows/wsl/.

[58] P. Team. "PyTorch: An Open Source Machine Learning Framework." Accessed: 2024-10-04. (2024), dirección: https://pytorch.org/.

[59] D. Inc. "Docker: Enterprise Container Platform." Accessed: 2024-10-04. (2024), dirección: https://www.docker.com/.

[60] "Using Dev Containers in WSL 2 - Visual Studio Code." (2024), dirección: https://code.visualstudio.com/blogs/2020/07/01/containers-wsl.

[61] "Docker WSL2 vs. Hyper-V: Which One Should You Use?" (2024), dirección: https://hatchjs.com/docker-wsl2-vs-hyper-v/.

[62] "Learn WSL." (2024), dirección: https://aka.ms/learnwsl.

[63] P. Judge. "Doug cutting: Big data is not a bubble and Hadoop is here to stay." (oct. de 2012), dirección: https://www.silicon.co.uk/workspace/doug-cutting-big-data-is-not-a-bubble-96694.

[64] A. Woodie. "Why Hadoop on IBM Power." (mayo de 2014), dirección: https://www.datanami.com/2014/05/12/hadoop-ibm-power/.

[65] N. Hemsoth. "Cray launches Hadoop into HPC Airspace." (oct. de 2014), dirección: https://www.hpcwire.com/2014/10/15/cray-launches-hadoop-hpc-airspace/.

[66] G. Cloud. "Bigtable: A Fully-Managed NoSQL Database Service for Large Analytical and Operational Workloads." Accessed: 2024-10-04. (2024), dirección: https://cloud.google.com/bigtable/.

[67] A. S. Foundation. "HDFS: Hadoop Distributed File System." Accessed: 2024-10-04. (2024), dirección: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.

[68] O. Corporation. "Java: The Programming Language." Accessed: 2024-10-04. (2024), dirección: https://www.java.com/.

[69] S. Center. "Scala: A Scalable Programming Language." Accessed: 2024-10-04. (2024), dirección: https://www.scala-lang.org/.

[70] R. Project. "R: The R Project for Statistical Computing." Accessed: 2024-10-04. (2024), dirección: https://www.r-project.org/.

[71] B. Chambers y M. Zaharia. "Spark: The definitive guide." (2022), dirección: https://www.oreilly.com/library/view/spark-the-definitive/9781491912201/ch01.html.

[72] Indra, "Implantación de InGRID MDM en Uruguay," indracompany.com, inf. téc. dirección: https://www.indracompany.com/es/implantacion-ingrid-mdm-uruguay.

[73] Apache Airflow. "TaskFlow API: A better way to author workflows." (2024), dirección: https://airflow.apache.org/docs/apache-airflow/stable/tutorial/taskflow.html (visitado 21-10-2024).

[74] A. Martins y M. Barragán. "Sistema de Aprendizaje Automático para la Detección de Irregularidades en Redes Eléctricas de UTE." Consultado el: 31 de octubre de 2024. (2024), dirección: https://gitlab.fing.edu.uy/proygrado-mlops-ntl/entrega.

[75] A. Airflow. "PythonOperator." Accessed: 2024-10-28. (2024), dirección: https://airflow.apache.org/docs/apache-airflow/stable/howto/operator/python.html.

[76] Python Guide. "Pipenv: Python Development Workflow for Virtual Environments." (2024), dirección: https://docs.python-guide.org/dev/virtualenvs/ (visitado 21-10-2024).

[77] Microsoft. "Visual Studio Code." (2024), dirección: https://code.visualstudio.com/ (visitado 28-10-2024).

[78] P. de Grado - MLOps NTL. "Pruebas de Concepto para TFX y Pipelines." Accedido: 2024-11-04. (2024), dirección: https://gitlab.fing.edu.uy/proygrado-mlops-ntl/entrega-tfx.

[79] TensorFlow Extended (TFX). "Chicago Taxi Dataset." (2024), dirección: https://www.tensorflow.org/tfx/tutorials/tfx/components_keras?hl=es-419 (visitado 21-10-2024).

[80] Cluster.uy. "Cluster de Supercomputación." (2024), dirección: https://www.cluster.uy/ (visitado 28-10-2024).

[81] "TensorFlow Extended (TFX) v1.15.1." (2024), dirección: https://github.com/tensorflow/tfx/tree/v1.15.1.

Glosario

El objetivo de este apartado es permitirle a un lector especializado en el área, aunque no necesariamente en la temática, comprender con mayor facilidad ciertos términos. Se organiza en forma alfabética y en el cuerpo de la obra se lo puede señalar con VERSALITA la primera vez que se menciona.

DAICE Detección Automática de Irregularidades en Consumos Eléctricos (DAICE) es un sistema avanzado diseñado y desarrollado conjuntamente por el Departamento de Procesamiento de Señales de la Facultad de Ingeniería de la Universidad de la República (UdelaR) y la SubGerencia de Recuperación de Energía de UTE. Este sistema tiene como objetivo principal identificar y analizar irregularidades en los consumos eléctricos, proporcionando una herramienta eficaz para el uso de energía. 5, 6, 43, 44, 45, 46

DeepDAICE DeepDAICE es un software avanzado que aprende a partir de datos de medidores inteligentes, con mediciones registradas cada 15 minutos. Desarrollado dentro del proyecto "Detección de anomalías en medidores inteligentes" y entregado a UTE en abril de 2021 como parte del acuerdo UTE-Fing. El manejo del gran volumen de datos generado por los medidores inteligentes requirió un cambio hacia la tecnología big data, utilizando un repositorio HBase controlado por la pila MDM de Cloudera en UTE. DeepDAICE integra registros de eventos en su análisis mediante redes neuronales recurrentes, abordando la naturaleza secuencial de la información de las alarmas. 43, 44

DevOps Una metodología que combina el desarrollo de software y las operaciones de TI para que las aplicaciones y servicios se puedan crear, probar y lanzar de manera más rápida y eficiente. 1

Idempotente En matemáticas e informática, una operación es idempotente si, al aplicarla múltiples veces, el resultado es el mismo que al aplicarla una sola vez. Por ejemplo, en programación, una función es idempotente si, al llamarla repetidamente con los mismos parámetros, produce el mismo resultado y no tiene efectos secundarios adicionales. 28

Industria 4.0 La cuarta revolución industrial que sigue a la Revolución Agrícola, la Primera Revolución Industrial (que introdujo maquinaria), la Segun-

da Revolución Industrial (que trajo la electricidad, el teléfono y los aviones), y la Tercera Revolución Industrial (que fue digital). La Industria 4.0 ha dado lugar a un número ilimitado de conexiones máquina a máquina: bots, robots, dispositivos conectados, autos autónomos, smartphones, y bots que recopilan datos de redes sociales y más. Requiere algoritmos inteligentes que procesen datos y tomen decisiones sin intervención humana a gran escala para manejar esta cantidad de datos sin precedentes en la historia de la humanidad. Las grandes tecnologías necesitaban encontrar un modelo de inteligencia artificial único que pudiera realizar una variedad de tareas que antes requerían varios algoritmos separados, conocidos como modelos fundacionales. Utiliza MLOps (Machine Learning Operations) para automatizar y optimizar el ciclo de vida de los modelos de Machine Learning, mejorando la eficiencia y la capacidad de respuesta en los procesos industriales. 1

Pipeline En el contexto de MLOps, un PIPELINE (o PIPELINES en plural) es una secuencia estructurada de pasos que incluyen la recolección, el procesamiento, el entrenamiento y la validación de modelos de machine learning. Cada paso en el pipeline se ejecuta de manera secuencial o paralela, garantizando un flujo de trabajo eficiente y reproducible para el desarrollo y la implementación de modelos. Los pipelines ayudan a automatizar tareas repetitivas, a mantener la consistencia de los procesos y a facilitar la integración continua y el despliegue continuo (CI/CD) de modelos en producción. 53, 71

Pipenv Pipenv es una herramienta que busca aunar lo mejor de varios mundos en la gestión de paquetes y entornos virtuales en Python. Automatiza la creación de un entorno virtual, la instalación de dependencias y el manejo del archivo Pipfile, que sustituye a requirements.txt para definir los paquetes de los que depende el proyecto. Pipenv es ampliamente utilizado para mejorar la gestión de dependencias en proyectos de Python, asegurando compatibilidad entre entornos y facilitando la colaboración entre desarrolladores. 63

XCom En el contexto de Airflow, XCom (abreviatura de "cross-communication") es un mecanismo que permite el intercambio de datos entre tareas dentro de un DAG. Cada mensaje de XCom consta de una clave, un valor y metadatos asociados, y puede ser utilizado para compartir información como resultados de tareas, parámetros de configuración, y otros datos necesarios para la ejecución del flujo de trabajo. 40

Anexo 1

.1. Ejemplos de DAG con y sin TaskFlow API

A continuación, se muestran los ejemplos de definición de DAG con y sin TaskFlow API, basados en la documentación oficial de Airflow y TaskFlow API.

.1.1. Definición de DAG sin TaskFlow API

```
def read_input():
    # Logic to read input
    pass

def pre_process(data):
    # Logic to pre_process input
    pass

def train(data):
    # Logic to train
    pass

dag = DAG('daice_dag', schedule_interval=None,
max_active_runs=1)

read_input_task = PythonOperator(
    task_id='read_input_task',
    python_callable=read_input, dag=dag)

pre_process_task = PythonOperator(
    task_id='pre_process_task',
    python_callable=read_input, dag=dag)

train_task = PythonOperator(
    task_id='train_task',
    python_callable=read_input, dag=dag)

# Pipeline con operador >>
read_input_task >> pre_process_task >> train_task
```

```
# Pipeline con set_downstream
read_input_task.set_downstream(pre_process_task)
pre_process_task.set_downstream(train_task)
```

.1.2. Definición de DAG con TaskFlow API

```
@task()
def read_input():
    # Logic to read input
    pass

@task()
def pre_process(data):
    # Logic to pre_process input
    pass

@task()
def train(data):
    # Logic to train
    pass

@dag(
    schedule=None,
    start_date=pendulum.datetime(2023, 1, 1, tz="UTC")
)
def daice_dag():
    data = read_input()
    pre_processed_input = pre_process(data)
    train(pre_processed_input)

daice_dag()
```

.2. Plan de proyecto inicial

En este anexo se incluye el plan de proyecto presentado inicialmente para el desarrollo del sistema de detección de irregularidades en redes eléctricas en UTE.

Proyecto de fin de estudios en la carrera Ingeniería en Computación

Desarrollo de un Sistema de Aprendizaje Automático Para la Detección de Fraudes en Redes Eléctricas de UTE

Plan de Proyecto

Autores	Alexander Martins
	Maximiliano Barragán
Tutor	Lorena Etcheverry
Co-tutor	Pablo Massaferro
Fecha	Agosto de 2022

Facultad de Ingeniería – Universidad de la República

1. Resumen

- Estudiantes:
  - Lerder Alexander Martins Masner
    - CI: 4.404.992-9
    - email: alexlrdr@hotmail.com
  - Maximiliano Andrés Barragán Pavoni
    - CI: 4.584.266-1
    - email: maxibp93@gmail.com
- Cliente: Depto. Recuperación de Energía de UTE
- Tutor: Lorena Etcheverry
- Co-Tutor: Pablo Massaferro
- Fecha prevista de finalización
- Total de hs. a realizar previstas por el grupo de proyecto
- Fecha y descripción de los entregables intermedios

2. Descripción del proyecto

La corriente eléctrica es parte fundamental de la sociedad actual, pero su distribución usando redes físicas de conductores expone a este bien al hurto y a la manipulación de los medidores. Según informes del Banco Interamericano de Desarrollo (BID) las pérdidas totales de energía eléctrica en América Latina y el Caribe (ALC) alcanzan el 17% de la energía generada. Las pérdidas no técnicas de energía eléctrica (NTL por sus siglas en inglés) representan un problema significativo para los países en desarrollo, y las compañías eléctricas. Sin tener en cuenta las pérdidas técnicas (dentro de los sistemas de distribución), el perjuicio económico causado por las NTLs para las economías de ALC asciende a 11 mil millones de dólares anuales. En los últimos años el problema ha sido atacado con técnicas de aprendizaje automático con el fin de mitigarlo [1].

Para reducir las pérdidas no técnicas se deben realizar estudios analíticos, ejecutar inspecciones y medir los resultados de las inspecciones para seguir optimizando la habilidad de detección. Además, la energía eléctrica constituye un recurso esencial para el desarrollo de la sociedad, y debido a que las redes se encuentran distribuidas en vastas regiones, son vulnerables al fraude o robo [2].

En este contexto se han desarrollado varios proyectos entre UTE (Administración de Usinas y Transmisiones Eléctricas) y la UdelaR (Universidad de la República). En particular, en el marco del proyecto “Implantación de un sistema de detección automática de irregularidades en el uso de energía eléctrica” financiado por el Fondo Sectorial de Energía de la Agencia Nacional de Investigación e Innovación (ANII-FSE), 2016-2018, se desarrolló el software DAICE (Detector Automático de Irregularidades en Consumos Eléctricos). Luego, en el contexto del proyecto “Detección de anomalías en medidores inteligentes” (2019-2021) como parte de un convenio entre Facultad de Ingeniería y UTE, se desarrolló DeepDAICE: una herramienta de detección de fraudes basada en aprendizaje profundo sobre datos de medidores inteligentes que actualmente el Departamento de Recuperación de Energía de UTE utiliza para el monitoreo de consumos eléctricos, la prevención y detección de pérdidas no técnicas [3,4]. Sin embargo, varios de los procesos asociados al entrenamiento y operacionalización de estas herramientas y modelos no cuentan aún con soluciones automáticas e integradas que faciliten la operativa, permitan la trazabilidad de los procesos, y faciliten la explotación de grandes volúmenes de datos.

Por otro lado, en los últimos años el concepto de MLOps (operacionalización del ***pipeline*** de Aprendizaje Automático) está tomando impulso. Esta corriente engloba un conjunto de buenas prácticas y herramientas que buscan poner en producción y mantener modelos de Aprendizaje Automático en

forma eficiente y confiable [5].

Este proyecto busca aplicar ideas y conceptos de MLOps a las soluciones existentes en UTE para la detección automática de fraudes.

3. Objetivo

Investigar y aplicar conceptos de MLOps a DAICE y DeepDAICE para el diseño e implementación de un prototipo de sistema de detección automática de fraudes que permita, además de la detección del fraude en sí, la visualización y monitoreo de un conjunto de métricas de desempeño, configuración de los distintos modelos de aprendizaje automático utilizados y el uso de ***pipelines*** para el procesamiento de datos (relacionales y no relacionales) y de modelos.

4. Alcance

Se estudiará en profundidad conceptos de MLOps, para comprender y definir qué fases o pasos de esta disciplina se pueden aplicar para la detección de fraudes en redes eléctricas. De esta forma se buscará la operacionalización del flujo tanto de los datos como de los modelos aplicados a la detección de fraudes. Además, se realizará un relevamiento del flujo actual y de las herramientas existentes en la industria relacionadas con la operacionalización o automatización en el área del aprendizaje automático (de aquí en adelante AA).

El prototipo de sistema a desarrollar, deberá permitir a los usuarios gestionar todos los pasos del ***pipeline*** tanto de los datos (validación y preparación en el flujo), como de los modelos. Es decir, el sistema deberá permitir a los usuarios gestionar el pre-procesamiento, visualización y validación/verificación de los datos tanto para entrenamiento como para clasificación. Además, será posible gestionar los distintos modelos a utilizar para la detección de fraudes para algoritmos de XGBoost y Tensorflow. Se podrán visualizar métricas de desempeño de estos modelos o los resultados de los mismos. Además, el sistema deberá mantener registro de versiones de los datos que se usaron para entrenar cada modelo, y que versión de los datos se usaron para clasificar. Las fases de entrenamiento, evaluación y validación de los modelos deberá incluirse en un flujo de AA.

No está dentro del alcance de este proyecto el investigar o desarrollar nuevos modelos de detección de fraude en redes eléctricas.

5. Criterios de éxito

Al finalizar el proyecto se contará con un **prototipo** que cumpla con los requerimientos descritos anteriormente para algoritmos en XGBoost y tensorflow, es decir:

- Contar con un pipeline de modelo (entrenamiento, evaluación y clasificación)
- Contar con un pipeline de datos (extracción, validación y preparación)
- Permitir entrenar modelos de aprendizaje automático desde las diferentes bases de datos y de archivos
- Visualizar si los datos de entrada cuentan con la calidad esperada desde el punto de vista estadístico
- Poder visualizar métricas de rendimiento de los modelos generados
- Se tendrá registro de los rendimientos de los diferentes modelos (*model registry*)
- Poder generar, cargar y utilizar exitosamente modelos de aprendizaje automático

De la wishlist:

- Será posible autenticarse
- Será posible obtener solicitudes de inspecciones
- Permitirá la visualización de los resultados de las inspecciones enviadas a campo

6. Actores

- **Cliente:** Gonzalo Caudullo de UTE
- **Tutora:** Lorena Etcheverry
- **Co-tutor:** Pablo Massaferro
- **Estudiantes:**
    - Alexander Martins
    - Maximiliano Barragán

7. Supuestos

1. El proyecto se basa en la hipótesis que existe un stack tecnológico de código abierto que se pueda integrar y con el que se pueda implementar el sistema.
2. Se espera que sea posible el acceso a la base de datos no relacional de UTE, ya que en este momento el acceso aún está siendo gestionado.

8. Restricciones

- El proyecto debe desarrollarse utilizando herramientas Open Source.
- Se debe tener en cuenta la Ley N° 18331 de protección de datos personales.
- El sistema debe desplegarse en las máquinas propietarias de UTE.

9. Especificación funcional del Proyecto

El sistema deberá permitir a los usuarios ver distintas métricas de desempeño/rendimiento de diferentes modelos de AA que permitan realizar una detección de fraudes en redes eléctricas, que hayan sido generados previamente. Dada la importancia de mantener la trazabilidad de los modelos y medidas de rendimiento, la aplicación deberá ser capaz de almacenar los parámetros de construcción de los diferentes modelos, de tal forma que se permita generar un historial de las diferentes arquitecturas, hiperparámetros, rendimientos obtenidos en cada corrida y en qué marca de tiempo se generó.

En lo que tiene que ver con el flujo de creación de modelos, deberá haber un proceso automatizado que permita la creación, entrenamiento y clasificación referente a los modelos. Los datos utilizados para los modelos de aprendizaje automático serán obtenidos de las bases de datos relacionales, no relacionales y de archivos. También permitirá la producción de nuevos modelos de aprendizaje automático y será posible agregar modelos ya existentes a través de la interfaz web.

La solución deberá permitir analizar que la calidad de los datos de entrada para los modelos de aprendizaje automático, que estén dentro de los valores esperados, evitando así el uso de datos incorrectos que puedan llevar a errores o bajo rendimiento en las predicciones.

Es necesario que la elaboración de la aplicación sea hecha utilizando soluciones de código abierto.

De la wishlist:

El sistema deberá contar con una interfaz web donde el usuario pueda navegar y le permita su autenticación para acceder a funciones administrativas como pudiera ser la creación de solicitudes para la generación de inspecciones que luego puedan ser utilizadas a nivel de campo. También deberá mostrar los resultados de dichas inspecciones a medida que se vayan actualizando en la base de datos.

ESBOZO DE LOS REQUERIMIENTOS DETALLADOS PARA LA APROXIMACIÓN A MLOps NIVEL 1 (Figura 1)

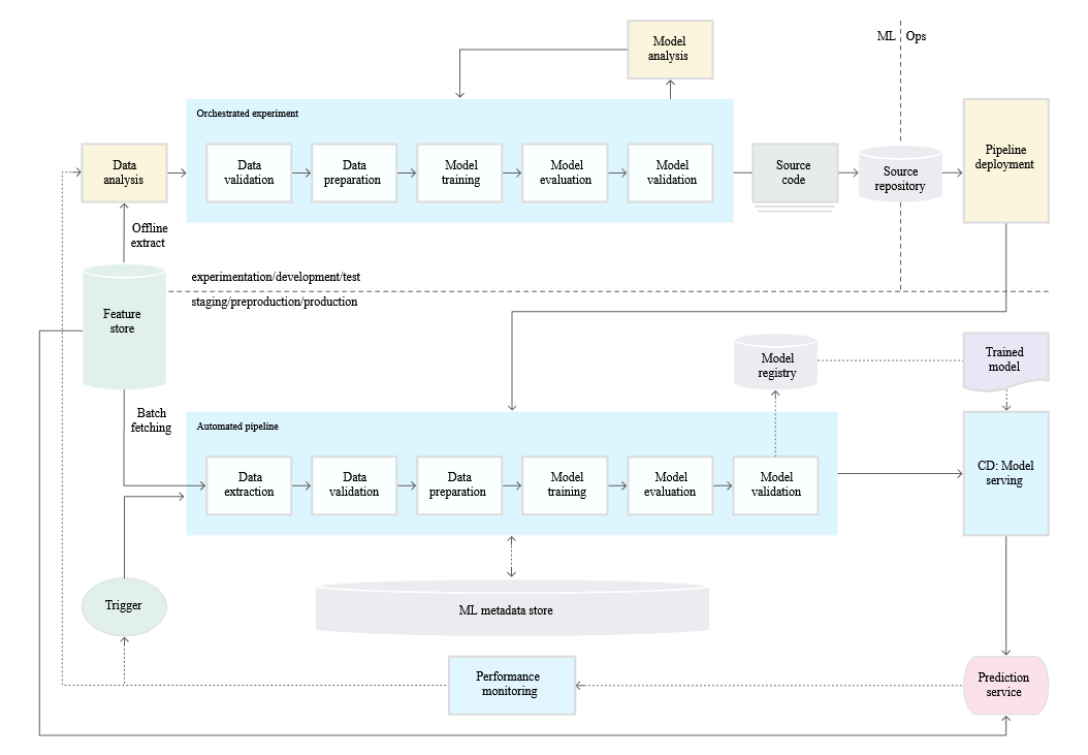


Figura 1: Automatización del flujo de trabajo en AA - Imagen tomada del manual "MLOPS: Continuous delivery and automation pipelines in machine learning, cloud architecture center, Google Cloud , 2022"

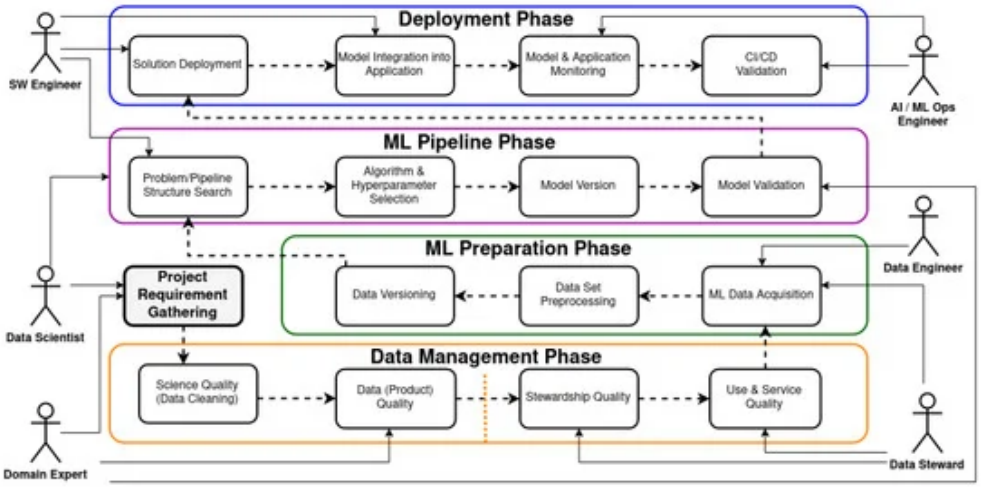


Figura 2: Fases y componentes presentes en un pipeline de AA - Imagen tomada de "Demystifying mlops and presenting a recipe for the selection of open-source tools"

9.1 Verificación de calidad de los datos

Los datos de entrada tienen valores fijos para cada modelo, parte de ellos viene de una base de datos relacional y los otros de una base de datos no relacional, también se podrán consumir datos de archivos csv o h5.

Las características vendrán definidas en las consultas de SQL para XGB y directamente de la base de datos para la información de consumos de medidores inteligentes.

9.1.1 Verificación de los datos desde la base de datos (esquema de datos)

El sistema hará una verificación de los datos provenientes de la base de datos y los dará como válidos o inválidos según información estadística y de tipo de datos. Cantidad de columnas y tipo de datos (puede pasar que hayan cambios en los procedimientos almacenados o en las consultas a nivel de base de datos que pueden provocar errores).

Conjuntos de datos nuevos para modelos ya existentes.

O cambios en la base de datos.

9.1.2 Verificación de los datos desde archivos con datos (h5 y csv)

El sistema hará una verificación de los datos provenientes de la base de datos y los dará como válidos o inválidos según información estadística y de tipo de datos.

9.1.3 Verificación del esquema de los datos

El sistema verificará que los datos ingresados cumplan con el esquema esperado por el modelo, de está forma se validará la integridad de los datos. El esquema de datos se refiere a cuáles son las diferentes columnas de datos que se esperan como también el tipo de datos.

9.1.4 Verificación de la consistencia de los datos

Verifica que los datos utilizados en producción estén dentro de los rangos esperados.

9.1.5 Verificación de sesgos entre datos de entrenamiento y de inferencia

Verifica que los datos utilizados en producción estén alineados con lo que se utilizaron en la fase de entrenamiento. Es decir, que desde un punto de vista estadístico, los datos de entrenamiento y de inferencia son similares, por ejemplo midiendo la diferencia de valores estadísticos de forma porcentual. El conjunto de entrenamiento tiene una estadística similar al conjunto de validación y clasificación. Además que la distribución y representación de cada subconjunto de datos esté correctamente distribuida. Puede ser un visualizador que muestre las estadísticas y que tenga algunos datos de salida.

9.1.6 Flujo de datos de aprendizaje automático

Los puntos referentes a los datos mencionados anteriormente deberán pertenecer a un flujo de datos utilizando alguna librería de orquestación. Lo anterior implica que cada uno de los requerimientos mencionados, de forma individual o agrupada, van a ser considerados componentes, dichos componentes podrán ponerse en secuencia o de forma paralela en un grafo dirigido acíclico, como se muestra en la **figura 2**.

9.2 Pipeline del modelo

Los modelos utilizados ya tendrán una arquitectura predefinida, también se podrán usar modelos ya existentes o entrenados previamente. Además los modelos serán implementaciones de las librerías *XGBoost* y *tensorflow* para los algoritmos de aprendizaje profundo.

9.2.1 Entrenamiento del modelo

El prototipo deberá permitir entrenar el modelo.

9.2.2 Evaluación del modelo

Comparación referente a las métricas del modelo al momento del entrenamiento.

9.2.3 Validación del modelo

Comparación con otros modelos en producción o referentes utilizando la técnica de A/B *testing*. De dichas comparaciones surgirán determinadas métricas que le ayudarán al técnico a decidir si utilizar el modelo en producción.

9.2.4 Registro de modelos

Se contará con un *model registry* donde se almacenarán los metadatos relacionados con la creación de los modelos, sus hiperparámetros y métricas referentes a los experimentos (f1, precision, recall, PR_AUC, ROC_AUC, FPR, TPR).

9.2.5 Flujo de aprendizaje automático

Los puntos anteriores deberán pertenecer a un flujo de datos automatizado utilizando alguna librería. Esto es, que cada uno de los requerimientos mencionados acerca del manejo de modelo, de forma individual o agrupada, llamemosles componentes, podrán ponerse en secuencia o de forma paralela en un grafo dirigido acíclico, como se muestra en la **figura 2**, utilizando algún framework que permita crear dichos grafos.

9.3 Conexión a la Big Data de medidores inteligentes

La aplicación será capaz de conectarse a la base de datos no relacional de donde se obtendrán datos para el entrenamiento y clasificación de DeepDAICE.

9.4 Rendimiento del modelo

Calcular y mostrar métricas obtenidas por el modelo en el subconjunto de datos de prueba (f1, precision, recall, PR_AUC, ROC_AUC, FPR, TPR)

9.4 Servicio de clasificación

El prototipo deberá permitir que el modelo sea accedido en una página web y permitir realizar predicciones en línea generando un ranking de dicha clasificación (mayor valor en la clasificación en los primeros lugares del ranking). Los datos de entrada son fijos para cada modelo.

LOS SIGUIENTES REQUERIMIENTOS PERTENECEN A LA WISHLIST

9.5. Solicitudes de inspecciones

Las solicitudes son pedidos de inspecciones donde se solicita cierta cantidad para ser inspeccionados. Los campos presentes en una solicitud son el id, estrategia DAICE o DeepDAICE y su versión (ejemplo, DAICEv1.2), la descripción y la fecha de la solicitud.

9.5.1 Visualizar solicitudes

El usuario visualiza las solicitudes realizadas, la versión del modelo que se utilizó para enviar, la cantidad de inspecciones incluídas en la solicitud, los detalles o descripción de la solicitud, la estrategía aplicada para realizar dicha solicitud.

9.5.2 Realizar solicitud (iniciado)

El usuario realiza una solicitud nueva de inspecciones, deberá especificar la cantidad de inspecciones, la zona a la que se envían, la descripción y la estrategía.

9.5.3 Eliminar solicitud (baja lógica)

El usuario puede eliminar una solicitud, pero de manera lógica. La información de la solicitud seguirá estando disponible en los repositorios pero no será accesible por los usuarios del sistema.

9.5.4 Editar información de solicitudes

Será posible modificar la información ingresada cuando tenga los permisos correspondientes.

9.5.5 Exportar solicitud creada

El sistema deberá permitir exportar un excel que contenga los clientes a los cuales se necesita inspeccionar.

9.5.6 Visualizar métricas de los modelos e hiperparámetros de modelos

El usuario visualiza los diferentes modelos, sus métricas, sus hiperparámetros, versión del modelo y descripción.

9.5.7 Visualizar resultados de inspecciones enviadas

El sistema mostrará los resultados a medida que estos se vayan actualizando a nivel de base de datos.

9.6 Interfaz de usuario

9.6.1 Registro de usuario

El sistema debe permitir a un usuario registrarse en la aplicación presentando como datos su nombre, apellido, correo electrónico y una contraseña.

9.6.2 Iniciar sesión

El sistema debe permitir a usuarios ya registrados acceder a las funcionalidades de la aplicación iniciando sesión ingresando correo electrónico y contraseña.

9.6.3 Cerrar sesión

El sistema debe permitir a un usuario que ya ingresó a la aplicación a través de la funcionalidad iniciar sesión, salir de esta.

9.6.4 Eliminar usuario (baja lógica)

El administrador del sistema puede eliminar un usuario, pero de manera lógica. La información del usuario seguirá estando disponible en los repositorios pero no será accesible por los usuarios del sistema, y el usuario eliminado lógicamente no podrá iniciar sesión.

9.6.5 Listar usuarios existentes

Se mostrará un listado de usuarios existentes.

9.6.6 Editar información de usuario (iniciado)

Será posible modificar la información ingresada del usuario cuando tenga los permisos correspondientes.

10. Objetivos específicos

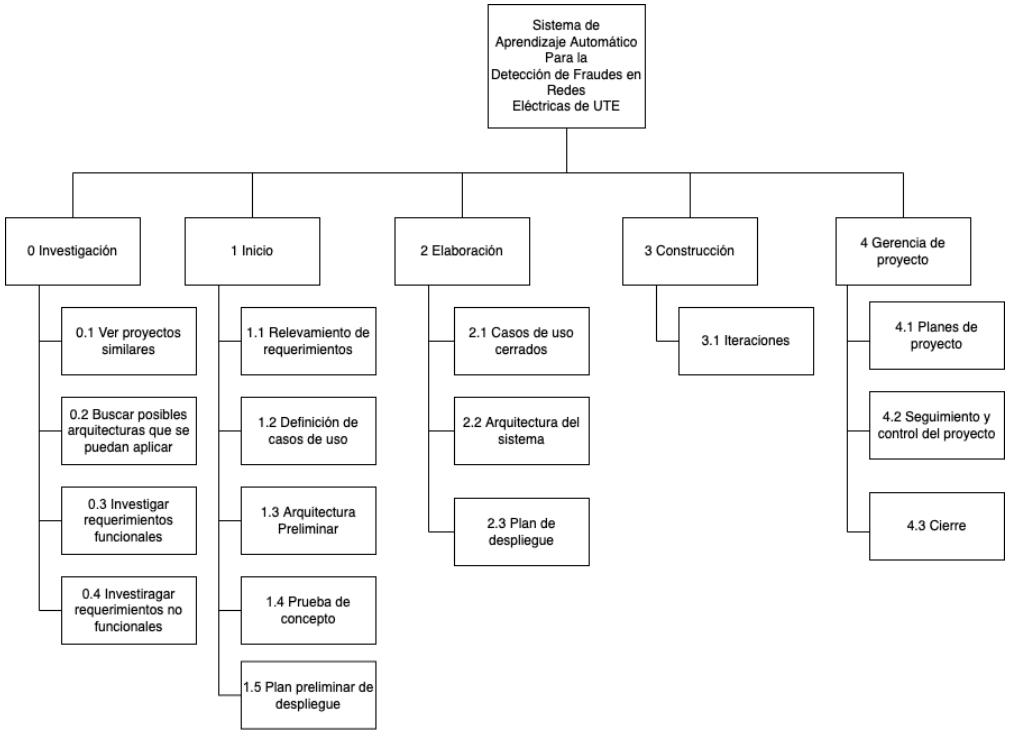
- Estudiar en profundidad el concepto de MLOps

- Realizar un relevamiento de herramientas, en particular de soluciones que abarquen el ciclo de vida de modelos de AA y la orquestación de los diferentes procesos involucrados
- Aplicar lo estudiado al contexto de la detección de fraudes y anomalías en el consumo eléctrico.
- Diseñar e implementar un módulo de gestión de datos:
    - Integración de datos (desde distintas fuentes: bases de datos relacionales, Hbase, archivos H5, etc.)
    - Visualización de datos
    - Consumo de datos
    - Validación de datos
    - Pre-procesamiento de datos
    - Versionado de datos
    - Implementar un pipeline de datos
- Diseñar e implementar un módulo de gestión de modelos:
    - Entrenamiento de modelos
    - Evaluación de modelos
    - Validación de modelos
    - Versionado de modelos
    - Implementar un pipeline de modelos
- Diseñar e implementar un módulo de visualización de métricas

Wishlist:

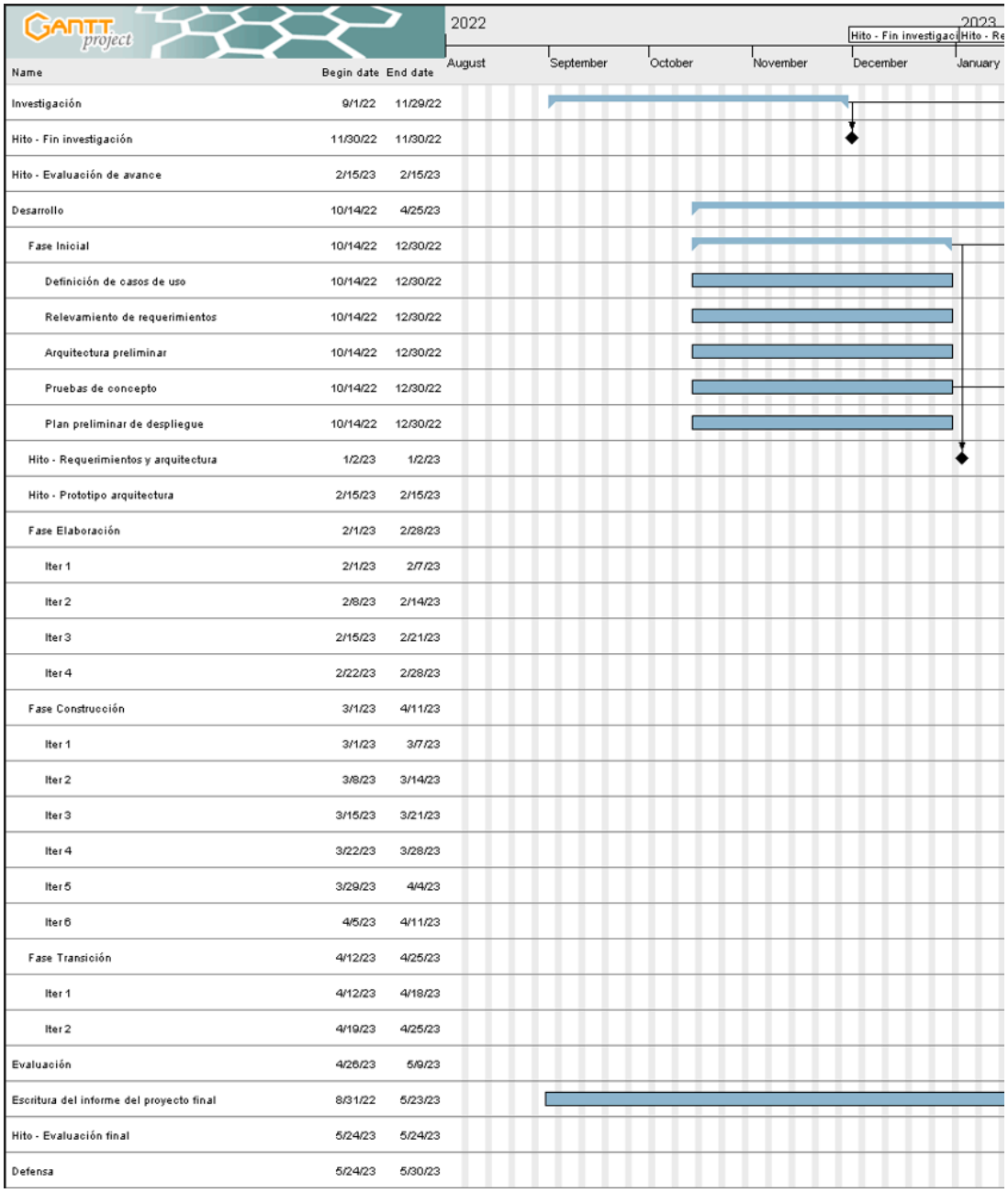
- Diseñar e implementar un módulo de autenticación de usuarios
    - ABM de usuarios
    - Gestión de permisos

11. Tareas


Sistema de Aprendizaje Automático Para la Detección de Fraudes en Redes Eléctricas de UTE
0 Investigación
1 Inicio
2 Elaboración
3 Construcción
4 Gerencia de proyecto
0.1 Ver proyectos similares
0.2 Buscar posibles arquitecturas que se puedan aplicar
0.3 Investigar requerimientos funcionales
0.4 Investiragar requerimientos no funcionales
1.1 Relevamiento de requerimientos
1.2 Definición de casos de uso
1.3 Arquitectura Preliminar
1.4 Prueba de concepto
1.5 Plan preliminar de despliegue
2.1 Casos de uso cerrados
2.2 Arquitectura del sistema
2.3 Plan de despliegue
3.1 Iteraciones
4.1 Planes de proyecto
4.2 Seguimiento y control del proyecto
4.3 Cierre

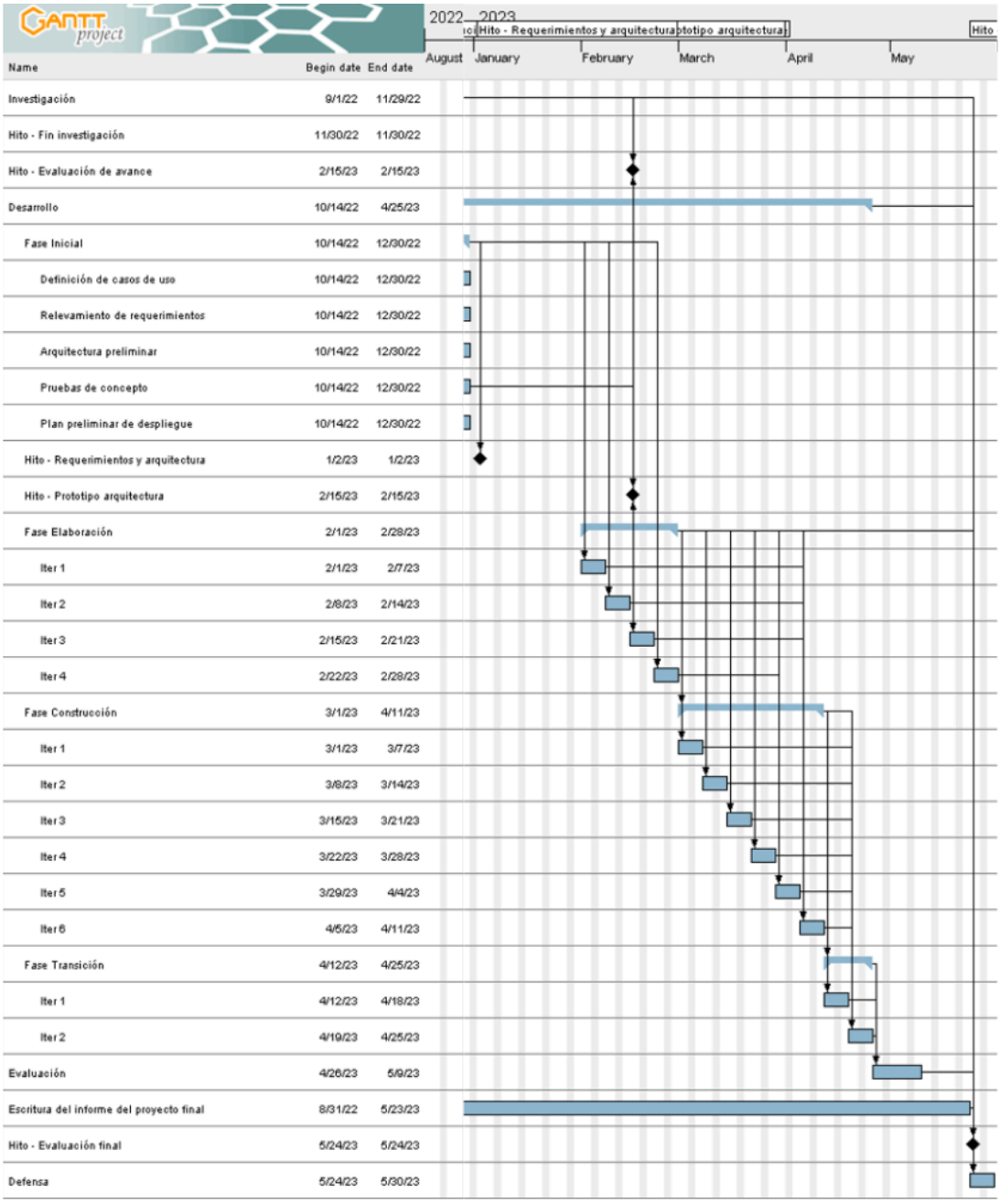
12. Cronograma detallado del Proyecto

Gant parte 1:



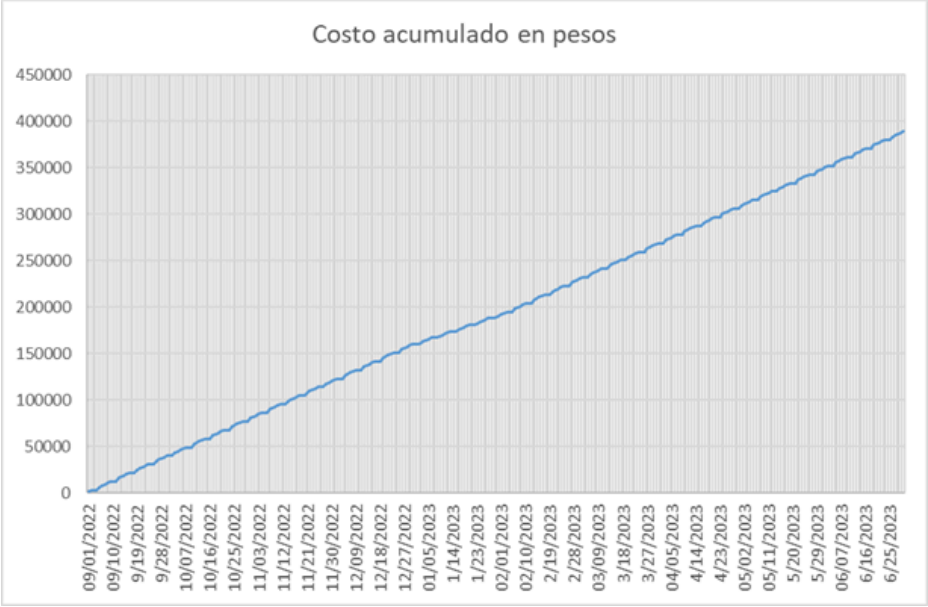
Name	Begin date	End date
Investigación	9/1/22	11/29/22
Hito - Fin investigación	11/30/22	11/30/22
Hito - Evaluación de avance	2/15/23	2/15/23
Desarrollo	10/14/22	4/25/23
Fase Inicial	10/14/22	12/30/22
Definición de casos de uso	10/14/22	12/30/22
Relevamiento de requerimientos	10/14/22	12/30/22
Arquitectura preliminar	10/14/22	12/30/22
Pruebas de concepto	10/14/22	12/30/22
Plan preliminar de despliegue	10/14/22	12/30/22
Hito - Requerimientos y arquitectura	1/2/23	1/2/23
Hito - Prototipo arquitectura	2/15/23	2/15/23
Fase Elaboración	2/1/23	2/28/23
Iter 1	2/1/23	2/7/23
Iter 2	2/8/23	2/14/23
Iter 3	2/15/23	2/21/23
Iter 4	2/22/23	2/28/23
Fase Construcción	3/1/23	4/11/23
Iter 1	3/1/23	3/7/23
Iter 2	3/8/23	3/14/23
Iter 3	3/15/23	3/21/23
Iter 4	3/22/23	3/28/23
Iter 5	3/29/23	4/4/23
Iter 6	4/5/23	4/11/23
Fase Transición	4/12/23	4/25/23
Iter 1	4/12/23	4/18/23
Iter 2	4/19/23	4/25/23
Evaluación	4/26/23	5/9/23
Escritura del informe del proyecto final	8/31/22	5/23/23
Hito - Evaluación final	5/24/23	5/24/23
Defensa	5/24/23	5/30/23

Gant parte 2:


GANTT project
2022
2023
Hito - Requerimientos y arquitectura
ototipo arquitectura
Hito
August
January
February
March
April
May
Name
Begin date
End date
Investigación 9/1/22 11/29/22
Hito - Fin investigación 11/30/22 11/30/22
Hito - Evaluación de avance 2/15/23 2/15/23
Desarrollo 10/14/22 4/25/23
Fase Inicial 10/14/22 12/30/22
Definición de casos de uso 10/14/22 12/30/22
Relevamiento de requerimientos 10/14/22 12/30/22
Arquitectura preliminar 10/14/22 12/30/22
Pruebas de concepto 10/14/22 12/30/22
Plan preliminar de despliegue 10/14/22 12/30/22
Hito - Requerimientos y arquitectura 1/2/23 1/2/23
Hito - Prototipo arquitectura 2/15/23 2/15/23
Fase Elaboración 2/1/23 2/28/23
Iter 1 2/1/23 2/7/23
Iter 2 2/8/23 2/14/23
Iter 3 2/15/23 2/21/23
Iter 4 2/22/23 2/28/23
Fase Construcción 3/1/23 4/11/23
Iter 1 3/1/23 3/7/23
Iter 2 3/8/23 3/14/23
Iter 3 3/15/23 3/21/23
Iter 4 3/22/23 3/28/23
Iter 5 3/29/23 4/4/23
Iter 6 4/5/23 4/11/23
Fase Transición 4/12/23 4/25/23
Iter 1 4/12/23 4/18/23
Iter 2 4/19/23 4/25/23
Evaluación 4/26/23 5/9/23
Escritura del informe del proyecto final 8/31/22 5/23/23
Hito - Evaluación final 5/24/23 5/24/23
Defensa 5/24/23 5/30/23

13. Análisis de Costos

Estimación de presupuesto base del proyecto.



14. Análisis de Riesgos

- Identificar los 5 riesgos más relevantes del proyecto.

    - Obtención del permiso para el acceso a la big data de medidores inteligentes
    - Dificultad para automatizar el flujo de datos de aprendizaje automático
    - Existencia de un stack tecnológico de herramientas de código abierto para implementar las automatizaciones

- El producto no es aceptado por el cliente
- No se encuentra compatibilidad de horario con cliente del proyecto

- Estimación y evaluación inicial de los riesgos identificados.

# Riesgo		**Probabilidad de ocurrencia**	**Impacto**
1	**Obtención del permiso para el acceso a la big data de medidores inteligentes**	Moderado	Alto
2	**Dificultad para automatizar el flujo de datos de aprendizaje automático**	Poco probable	Alto
3	**Existencia de un stack tecnológico de herramientas de código abierto para implementar las automatizaciones**	Poco probable	Medio
4	**El producto no es aceptado por el cliente**	Poco probable	Extremo
5	**No se encuentra compatibilidad de horario con cliente del proyecto**	Moderado	Medio

Matriz de riesgos:

		Probabilidad de ocurrencia		
		Poco probable	Moderado	Muy probable
Nivel de impacto	Ninguno			
	Bajo			
	Medio	R3	R5	
	Alto	R2	R1	
	Extremo	R4		

- Crear un Plan de Respuestas para cada riesgo.

  - **Riesgo 1**: Insistir con el cliente para que el acceso a la big data se logre lo antes posible.
  - **Riesgo 2**: Seguir investigando las soluciones existentes para entender mejor el problema que se está atacando.

- **Riesgo 3**: Buscar herramientas de código abierto que permitan atacar el problema en un prototipo inicial.
- **Riesgo 4**: Interacción entre el cliente y al menos alguno de los integrantes del equipo cada 15 días.
- **Riesgo 5**: Investigar sobre canales de comunicación para mejorar la comunicación con el cliente.

- Estimación y evaluación final de riesgos (luego de realizado el plan de respuestas).

# Riesgo		**Probabilidad de ocurrencia**	**Impacto**
1	**Obtención del permiso para el acceso a la big data de medidores inteligentes**	Poco probable	Alto
2	**Dificultad para automatizar el flujo de datos de aprendizaje automático**	Poco probable	Alto
3	**Existencia de un stack tecnológico de herramientas de código abierto para implementar las automatizaciones**	Poco probable	Medio
4	**El producto no es aceptado por el cliente**	Poco probable	Extremo
5	**No se encuentra compatibilidad de horario con cliente del proyecto**	Poco probable	Medio

Matriz de riesgos:

		Probabilidad de ocurrencia		
		Poco probable	Moderado	Muy probable
Nivel de impacto	Ninguno			
	Bajo			
	Medio	R3, R5		
	Alto	R1, R2		
	Extremo	R4		

Referencias:

[1] Massaferro, Pablo, J. Matías Di Martino, and Alicia Fernández. "NTL detection: Overview of classic and DNN-based approaches on a labeled dataset of 311k customers." 2021 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT). IEEE, 2021.

[2] G. M. Messinis and N. D. Hatziargyriou, “Review of non-technical loss detection methods,” Electric Power Systems Research, vol. 158, pp. 250–266, 2018.

[3] Massaferro, Pablo, J. Matías Di Martino, and Alicia Fernández. "Fraud Detection on Power Grids While Transitioning to Smart Meters by Leveraging Multi-Resolution Consumption Data." IEEE Transactions on Smart Grid 13.3 (2022): 2381-2389.

[4] Massaferro, Pablo, J. Matías Di Martino, and Alicia Fernández. "Fraud detection in electric power distribution: An approach that maximizes the economic return." IEEE Transactions on Power Systems 35.1 (2019): 703-710.

[5] “ML Ops: Machine Learning as an Engineering Discipline”, https://towardsdatascience.com/ml-ops-machine-learning-as-an-engineering-discipline-b86ca4874a3f

[6] Mlops: Continuous delivery and automation pipelines in machine learning, cloud architecture center, google cloud. (2022, 7). Google. Descargado de https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learnin

[7] Ruf, P., Madan, M., Reich, C., y Ould-Abdeslam, D. (2021). Demystifying mlops and presenting a recipe for the selection of open-source tools. Applied Sciences, 11 (19), 8861.