Escuela de Ingeniería
Ingeniería Civil Computación

Mitigación de la Degradación de Modelos de Machine Learning en Ambientes Productivos

Marcelo Guzmán
Profesor(a) guía: Ignacio Bugueño
Comisión evaluadora: Waldo Gálvez – Stefan Escaida

Memoria para optar al título y/o grado de Ingeniero.

Rancagua, Chile
Agosto del 2024

Dedicatoria

A mi amada Katalina, por ser mi *katalizador*.

Agradecimientos

Quiero expresar mi más profundo agradecimiento a todas aquellas personas que han sido fundamentales en mi vida y en la realización de esta memoria.

A mis padres, Marcelo y Celinda, y a mis hermanos, Daniel y Roberto, gracias por el apoyo incondicional y la confianza que han depositado en mí a lo largo de estos años, me han inspirado a seguir adelante y a nunca rendirme.

A Alejandra, le agradezco profundamente por haberme formado con valores y principios, enseñándome lo que realmente significa ser una persona buena y generosa.

A mi pareja, Katalina, te agradezco por tu presencia constante y tu inquebrantable fe en mí, que han sido mi mayor fortaleza. Gracias por estar a mi lado en cada momento, compartiendo tanto las alegrías como los desafíos.

A mis compañeros Benjamín, Ignacio y Nicolás, con quienes inicié este viaje, gracias por una compañía inigualable y por un trabajo en equipo excepcional. Haber llegado juntos a esta ansiada conclusión es un logro que siempre llevaré en mi corazón. Les deseo lo mejor en el futuro.

Finalmente, a mi profesor guía, Ignacio, le agradezco su excelente disposición, apoyo y orientación, elementos esenciales para el desarrollo de este trabajo.

A todos ustedes, mi más sincero agradecimiento.

Índice

Índice de Figuras

1. Resumen

La Inteligencia Artificial (IA) y los modelos de Machine Learning (ML) están revolucionando diversos sectores como las finanzas, salud, agricultura y transporte. Sin embargo, estos modelos son propensos a la degradación con el tiempo debido a cambios en los datos y en el entorno operativo. Esta degradación puede llevar a una disminución en la precisión de los modelos, con consecuencias potencialmente negativas y, en algunos casos, catastróficas.

El objetivo de esta tesis es estudiar y mitigar la degradación de los modelos de ML en ambientes productivos. Se abordarán los conceptos de *model drif* y *data drift*, se analizarán casos reales de degradación y se desarrollará una aplicación web que detecte, monitoree y mitigue estos problemas. La herramienta propuesta permitirá el reentrenamiento de los modelos afectados, asegurando así su rendimiento óptimo. Esta herramienta será puesta a prueba por usuarios para evaluar su desempeño en materias de usabilidad.

La investigación se centrará en identificar mecanismos para la detección de degradación, entrenar modelos en un entorno simulado y diseñar una estrategia de mitigación efectiva. El resultado final será una herramienta que detecta la degradación y permite acciones correctivas inmediatas para mantener la eficacia de los modelos de ML en producción.

Abstract

Artificial Intelligence (AI) and Machine Learning (ML) models are revolutionizing various fields such as finance, healthcare, agriculture, and transportation. However, these models are prone to degradation over time due to changes in data and operational environments. This degradation can lead to a significant decrease in model performance, with potentially negative and sometimes catastrophic consequences.

The aim of this thesis is to study and mitigate the degradation of ML models in production environments. The concepts of *model drift* and *data drift* will be addressed, real cases of degradation will be analyzed, and a web application will be developed to detect, monitor, and mitigate these issues. The proposed tool will enable retraining of affected models, thus ensuring their optimal performance. This tool will be tested by users to evaluate its performance in terms of usability.

The research will focus on identifying mechanisms for degradation detection, training models in a simulated environment, and designing an effective mitigation strategy. The outcome will be a tool that detects degradation and allows immediate corrective actions to maintain the efficacy of Machine Learning models in production.

2. Introducción

2.1 Motivación

La Inteligencia Artificial es un campo de la informática que en la actualidad avanza a una velocidad vertiginosa en materia de desarrollo, y la amplia gama de aplicaciones en las que es ocupada tampoco se queda atrás; específicamente, los modelos de Machine Learning. Estos modelos están a la vanguardia, siendo utilizados en distintos ámbitos del mundo real como las finanzas, la salud, la agricultura, el transporte, etc. Es en esta masificación del uso de la Inteligencia Artificial por medio de modelos de Machine Learning que se hace notorio un problema difícil de evitar durante el ciclo de vida de estos modelos: la degradación.

Los modelos de Machine Learning son dependientes de una serie de factores vitales relacionados a la forma en que se generan y trabajan. Cambios en estos factores pueden causar repercusiones tanto a corto como a largo plazo afectando de manera significativa la calidad de la inferencia de los modelos. Las consecuencias de una inferencia errónea pueden variar dependiendo del área en la que se trabaje, pero siempre serán de carácter negativo, incluso algunos con la posibilidad de ser fatales como por ejemplo que el modelo desarrollado por una empresa enfocado a la conducción autónoma de un vehículo se confunda a la hora de diferenciar a una persona, ocasionando un accidente. Es por estas razones que es necesario entender las causas que conllevan a la degradación, los tipos de degradación que se pueden generar y ser capaces de reaccionar a tiempo por medio de estrategias que garanticen la mitigación de esta problemática.

2.2 Objetivo Principal

El objetivo del proyecto es comprender el problema de la degradación en los modelos de Machine Learning, y a partir de ese entendimiento, diseñar e implementar una herramienta capaz de detectar de manera anticipada dicha degradación. Esta herramienta permitirá mitigar los efectos negativos mediante el reentrenamiento oportuno del modelo. La detección anticipada se basará en el monitoreo continuo de métricas clave, las cuales indicarán posibles signos de degradación, permitiendo así la implementación de acciones correctivas de manera proactiva.

2.3 Objetivos Específicos

- Investigar, entender e identificar los tipos de degradación de modelos de Machine Learning, haciendo énfasis a las métricas necesarias para su detección.
- Entrenar un modelo de Machine Learning para su experimentación y estudio en una simulación de ambiente productivo.

- Implementación de herramientas de monitoreo del modelo, enfocándose en la precisión y las métricas críticas para detectar degradación generando alertas al usuario.
- Desarrollar una aplicación web capaz de reentrenar un modelo mitigando la degradación, además de desplegar todas las herramientas de monitoreo y visualización de métricas.
- Utilizar el servicio de Docker para encapsular la aplicación resultante en un contenedor, garantizando su portabilidad y fácil despliegue en distintos entornos, lo que contribuye a la implementación efectiva de la herramienta de reentrenamiento y monitoreo.
- Evaluar la usabilidad de la aplicación desarrollada mediante un cuestionario dirigido a usuarios, para asegurar que la herramienta es intuitiva y fácil de usar.

2.3 Contribuciones

Los resultados significativos, la documentación y todos los códigos generados serán almacenados en un repositorio de GitHub para su acceso a futuro, su utilización en otros proyectos o para indagar más a fondo en la temática.

2.4 Estructura del Documento

Este trabajo está organizado de la siguiente forma:

- Capítulo 3 Marco Teórico: Se realiza una revisión bibliográfica en el campo de la Inteligencia Artificial, modelos de Machine Learning y su degradación, destacando los conceptos fundamentales para el entendimiento del trabajo.
- Capítulo 4 Marco Metodológico: Se detalla la metodología utilizada para llevar a cabo el trabajo cumpliendo cada uno de los objetivos propuestos.
- Capítulo 5 Resultados: Se presentan los resultados del trabajo abordado, acompañado de un análisis de las métricas obtenidas en el monitoreo y los resultados finales de la aplicación confeccionada. Junto a una evaluación de usabilidad de esta.
- Capítulo 6 Discusiones: Se profundizan y analizan los resultados obtenidos tanto en materia de los experimentos realizados como en el desarrollo y evaluación de la aplicación final.
- Capítulo 7 Conclusiones: se muestran las conclusiones del trabajo, resumiendo los puntos clave y las implicancias encontradas.

3. Marco teórico y revisión de literatura

3.1 Inteligencia Computacional

La Inteligencia Computacional es el estudio de los mecanismos adaptativos que permiten o facilitan el comportamiento inteligente en entornos complejos y cambiantes. Se han logrado enormes éxitos a través de la modelización de la inteligencia biológica y natural. Estos algoritmos inteligentes incluyen la computación evolutiva, la inteligencia de enjambre, el sistema inmune artificial y los sistemas difusos [1]. Junto con la lógica, el razonamiento deductivo, los sistemas expertos, el razonamiento basado en casos y el sistema simbólico de aprendizaje automático, estos algoritmos inteligentes forman parte del campo de la inteligencia artificial [1].

En otras palabras, todos los mecanismo y algoritmos que aborda la inteligencia computacional, inspirados en procesos biológicos, son técnicas o enfoques dentro del campo de la Inteligencia Artificial (Figura 1).

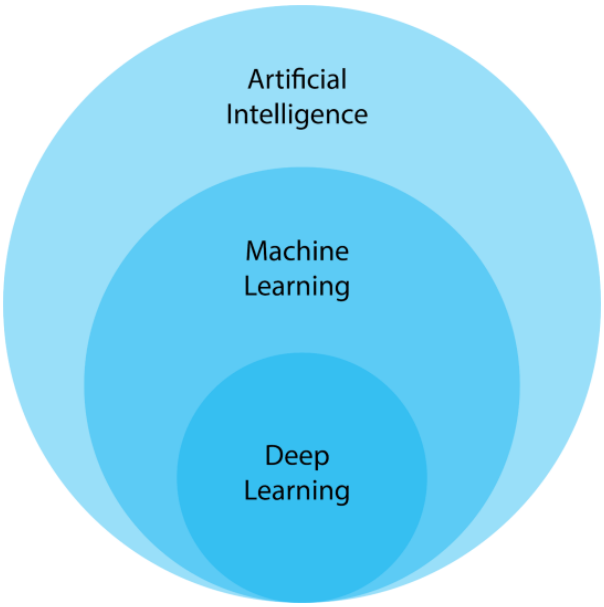


Fig. 1 Diagrama de Venn que muestra la relación entre Inteligencia Artificial, Machine Learning y Deep Learning. Fuente: Extraído de [2].

3.2 Machine Learning

"Machine Learning es la ciencia (y arte) de programar computadoras para que ellas puedan aprender a partir de los datos" [3]. Una definición más general del concepto dice que Machine Learning o Aprendizaje Automático es un campo de la Inteligencia Artificial centrada en la utilización de datos y la creación de algoritmos para el desarrollo de modelos que pueden aprender de patrones en los datos y llevar a cabo tareas específicas sin la necesidad de ser

programados explícitamente. Esto permite a un modelo de Machine Learning imitar el comportamiento de un humano al momento de aprender, mejorando gradualmente la precisión en las tareas que se quieren realizar [4]. Comprender las bases del aprendizaje automático es el primer paso para indagar en la degradación de sus modelos.

Existen distintos tipos aprendizaje automático, dependiendo del tipo y la cantidad de supervisión que reciben al momento de ser entrenados:

- Aprendizaje Supervisado: Este enfoque implica entrenar modelos utilizando datos etiquetados (Figura 2), donde se conoce la relación entre las características de entrada y la variable objetivo. El modelo aprende a realizar predicciones basadas en esta relación. Ejemplos donde se ocupa este tipo de aprendizaje son las tareas de clasificación o la predicción de valores numéricos y regresiones.

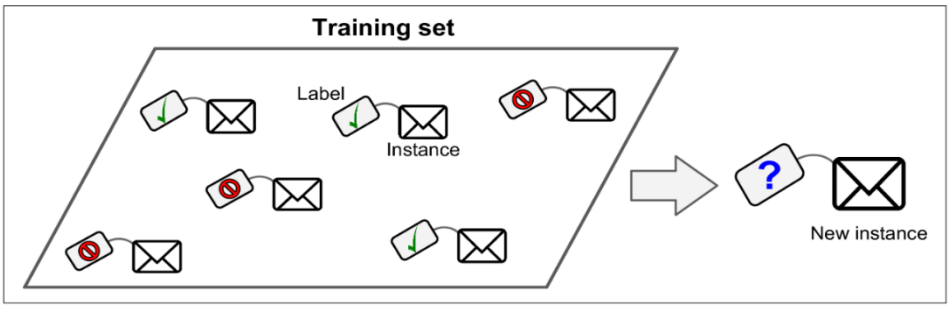


Fig. 2 Set de entrenamiento en aprendizaje supervisado. Fuente: Extraído de [3].

- Aprendizaje No Supervisado: Aquí, los modelos se entrenan utilizando datos no etiquetados (Figura 3) y aprenden a encontrar patrones o estructuras contenidas en los datos. No hay una variable objetivo-específica para predecir. Ejemplo de este tipo de aprendizaje son los algoritmos que realizan *clustering*.

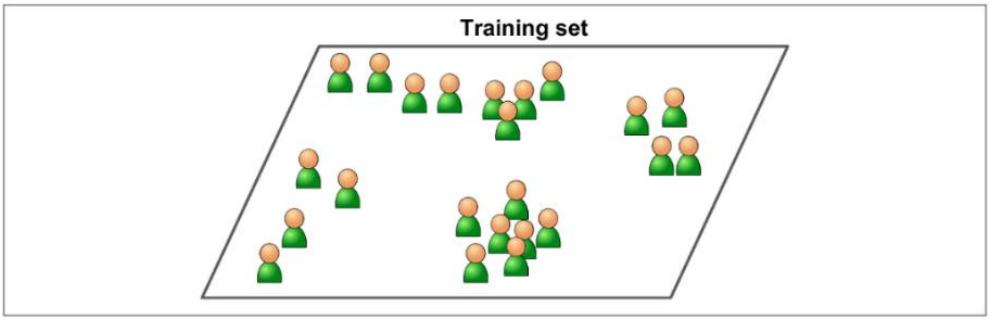


Fig. 3 Set de entrenamiento en aprendizaje no supervisado. Fuente: Extraído de [3].

- Aprendizaje Reforzado: En este tipo de aprendizaje, los modelos aprenden a tomar decisiones secuenciales a través de la interacción con un entorno. El modelo recibe retroalimentación en forma de recompensas o castigos según las acciones que realiza (Figura 4).

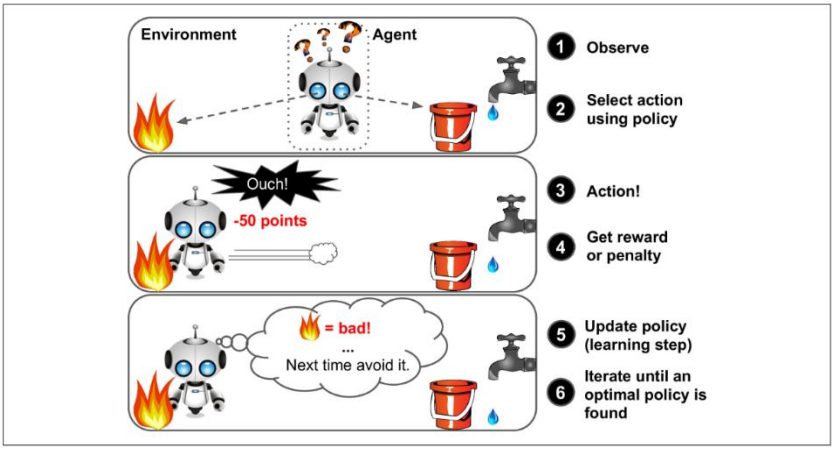


Fig. 4 Aprendizaje reforzado. Fuente: Extraído de [3].

Dentro de estos tipos de entrenamiento se encuentran también diversos tipos de algoritmos, siendo los más comunes los algoritmos de regresión lineal, los árboles de decisión y las redes neuronales [3]. Cada uno de estos algoritmos deben de pasar por un proceso de entrenamiento basado en datos y la selección de métricas de evaluación que concluye con la validación del modelo. Se debe tener en consideración que este proceso está vinculado a una selección oportuna de los parámetros y del modelo a ocupar, esto a razón de evitar el Overfitting o sobreajuste del modelo lo cual interviene en la generalización cuando se trabaja con nuevos datos.

Utilizar Machine Learning a la hora de afrontar problemas que no son simples de trabajar resulta beneficioso ya que en vez de definir un extenso set de reglas que se adapten a una solución el algoritmo aprenderá por sí solo estas reglas mediante el entrenamiento y la identificación de patrones en los datos; además, se añade un factor de adaptabilidad y escalabilidad por la opción de trabajar con nuevos y numerosos datos respectivamente. Los resultados obtenidos en estos procesos pueden ser analizados para comprender en mayor profundidad los problemas, por lo tanto, según Géron [3], los modelos no son los únicos que aprenden en un proceso de Machine Learning, los humanos también lo hacen.

3.3 *MLOps*

MLOps u operaciones de Machine Learning es un concepto y disciplina diseñada para velar por la eficiente gestión del ciclo de vida de un modelo de Machine Learning durante su despliegue y trabajo en un ambiente productivo [5]. El concepto de *MLOps* se encuentra en constante evolución debido a las diversas necesidades y desafíos que se presentan a la hora reducir los riesgos asociados al trabajo con modelos en la industria. Entender este concepto sirve para sentar las bases de escalabilidad del uso de los modelos de Machine Learning en la industria.

El proceso de creación de un modelo de Machine Learning es solo una parte de lo que se necesita para desplegar y mantener en funcionamiento un modelo. Existe una serie de factores que se deben de tener en consideración para que un modelo se implemente y trabaje de la forma que se espera que lo haga [6]. Todos estos factores forman parte del ciclo de vida de un modelo en un ambiente productivo como se ilustra en la Figura 5.

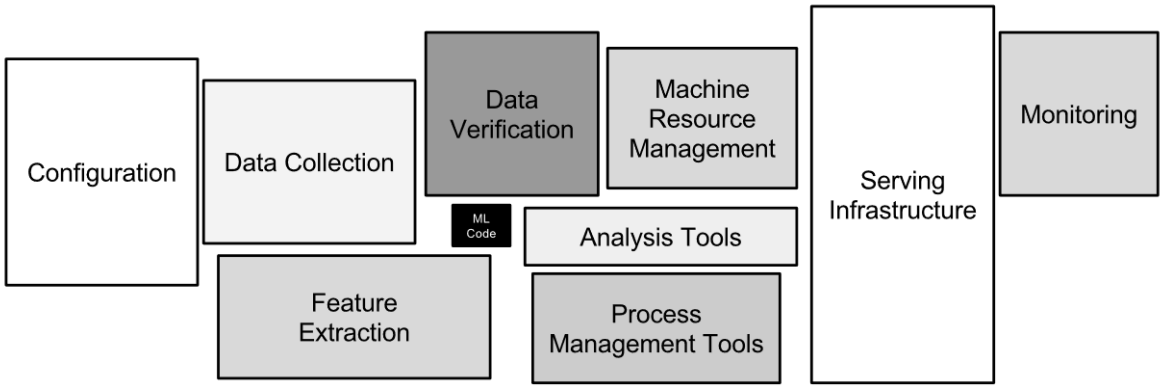


Fig. 5 El modelo, un pequeño componente en un sistema de Machine Learning. Fuente: Extraído de [6].

Un ambiente productivo corresponde al entorno en el cual un modelo de Machine Learning es ejecutado a fin de realizar una inferencia en tiempo real. Este ambiente debe de estar configurado de forma adecuada para soportar la ejecución eficiente del modelo. Dentro de esta configuración podemos encontrar la infraestructura, el almacenamiento de los datos y los distintos servicios que trabajan en la comunicación de las partes que componen al proceso.

3.4 *Model Degradation*

Sabemos que MLOps aborda la gestión del ciclo de vida de los modelos de Machine Learning (Figura 6) en producción. La selección del concepto de "ciclo de vida" no es una coincidencia y tiene una serie de implicancias, desde el nacimiento o desarrollo del modelo, la madurez en forma del exitoso entrenamiento y su implementación, pasando al envejecimiento o la degradación del modelo, continuando con la muerte y renovación del modelo.

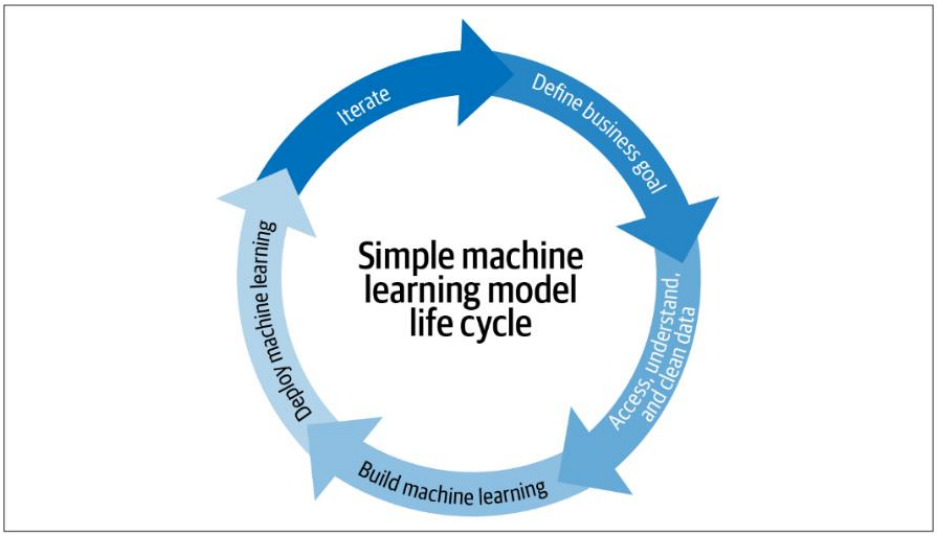


Fig. 6 Ciclo de vida simplificado de un modelo de Machine Learning. Fuente: Extraído de [5].

La fase de degradación de los modelos de Machine Learning corresponde a la fase en que se empieza a observar una decadencia en la calidad de la predicción. Esto puede deberse a una serie de factores, incluidos cambios en los datos de entrada, desviaciones en las condiciones del mundo real en comparación con el entorno de entrenamiento y la obsolescencia del modelo a medida que los requisitos cambian. Estos factores derivan en distintos tipos de degradación o *drift*; los más emblemáticos y los que se trabajarán en esta memoria son los siguientes:

- Data Drift**:** El *data drift* es un cambio en las propiedades estadísticas y características de los datos de entrada. Ocurre cuando un modelo de aprendizaje automático está en producción, ya que los datos que encuentra difieren de los datos con los que el modelo fue entrenado inicialmente o de los datos de producción anteriores. Este cambio en la distribución de los datos de entrada puede provocar un deterioro en el rendimiento del modelo. La razón es que, al crear un modelo de aprendizaje automático, se espera que funcione bien con datos similares a los utilizados para entrenarlo. Sin embargo, puede tener dificultades para realizar predicciones o tomar decisiones precisas si los datos siguen cambiando y el modelo no puede generalizar más allá de lo que ha visto en el entrenamiento [7].

- Model Drift**:** El *model drift* se centra en cómo el modelo mismo ya no puede hacer predicciones precisas debido a la obsolescencia de su lógica interna, es decir, cambios en la relación entre las variables de entrada y la salida del modelo. Se refiere a la degradación de la calidad de un modelo de Machine Learning con el tiempo [8]. Esta degradación es evidenciada directamente por medio del cálculo de métricas directas de calidad al momento de realizar una inferencia.

Para tener la posibilidad de reaccionar a tiempo ante estos tipos de degradación se debe tener el conocimiento y entendimiento de cómo y dónde ocurren. Lo anterior, sumado a un constante monitoreo de métricas claves, nos permitirá reaccionar y mitigar los efectos de la degradación.

3.5 Métricas de Evaluación

En la fase de monitoreo de un modelo de Machine Learning es fundamental entender las mediciones que serán utilizadas para evaluar el rendimiento y la precisión. Estas métricas nos permitirían discernir el tipo de degradación que está ocurriendo y generar alertas a tiempo para evitar un mal desempeño del modelo. Las métricas que se usarán en esta memoria se pueden dividir en dos categorías, unas enfocadas a la detección del *model drift* y otras enfocadas a la detección de *data drift*.

Métricas de detección de *model drift:*

- *Mean Squared Error* (MSE): El Error Cuadrático Medio (MSE) es una métrica fundamental en el ámbito del Aprendizaje Automático, especialmente en el campo del análisis de regresión. Sirve como una herramienta crucial para evaluar el rendimiento y la precisión de los modelos predictivos. Como su nombre lo indica, calcula el promedio de los cuadrados de los errores o residuales. En el contexto del aprendizaje automático, cuantifica la diferencia cuadrática media entre los valores reales y los valores predichos por el modelo [9]. En su fórmula (1) n corresponde al número de observaciones, y_i son los valores reales y x_i los predichos por el modelo en la respectiva posición i.

$$MSE = \frac{\sum_{i=1}^{n}(y_i - x_i)^2}{n} \quad (1)$$

- *Mean Absolute Error* (MAE): métrica utilizada para evaluar la precisión de los modelos de regresión. Mide la diferencia absoluta promedio entre los valores predichos y los valores reales. A diferencia de otras métricas, el MAE no eleva los errores al cuadrado, lo que significa que da el mismo peso a todos los errores, independientemente de su dirección. Esta propiedad hace que el MAE sea particularmente útil cuando se desea entender la

magnitud de los errores sin considerar si son sobreestimaciones o subestimaciones [9]. Su obtención se realiza mediante la fórmula 2 donde *n* es el número total de datos, y_i corresponde a la predicción en el valor *i*, y x_i el valor de verdad en la misma posición.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \quad (2)$$

- *Root Mean Square Error* (RMSE): Una de las medidas más comúnmente utilizadas para evaluar la calidad de las predicciones. Muestra cuán lejos están las predicciones de los valores reales medidos utilizando la distancia euclidiana. Para calcular el RMSE, se calcula el residual (diferencia entre la predicción y el valor real) para cada punto de datos, se calcula la norma del residual para cada punto de datos, se calcula la media de los residuales y se toma la raíz cuadrada de esa media. El RMSE se utiliza comúnmente en aplicaciones de aprendizaje supervisado, ya que el RMSE usa y necesita mediciones reales en cada punto de datos predicho [10]. Calculado mediante la fórmula 3 donde *N* corresponde al número de datos no faltantes, x_i el valor real y x_i primo a los valores estimados en la correspondiente posición *i*.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \hat{x}_i)^2}{N}} \quad (3)$$

- Coeficiente de determinación (R^2): Es un valor entre 0 y 1 que mide qué tan bien la línea de regresión se ajusta a los datos. El R^2 puede interpretarse como el porcentaje de la varianza en la variable dependiente que puede ser explicado por el modelo. Cuanto más cerca esté el R^2 de 1 o del 100%, mejor será el modelo para predecir la variable dependiente [11]. Representada en la fórmula 4 en la cual RSS es la suma residual de los cuadrados y TSS la suma total de los cuadrados.

$$R^2 = 1 - \frac{RSS}{TSS} \quad (4)$$

Métricas de detección de *data drift:*

- *Kolmogorov–Smirnov Test*: La prueba de Kolmogorov-Smirnov (prueba KS) puede detectar diferencias en la distribución en ambas direcciones. La prueba KS es particularmente útil para comparar distribuciones no normalmente distribuidas, o para comparar distribuciones con formas diferentes. La prueba KS funciona comparando las

funciones de distribución acumulativa empírica (ECDFs) de las dos muestras. El ECDF es una función que muestra la proporción de muestras menores o iguales a un valor dado. La estadística KS es la diferencia absoluta máxima entre los dos ECDFs [12]. Como se aprecia en la fórmula 5 donde sup_x *es* el supremo del conjunto de distancias entre la diferencia absoluta entre las dos ECDFs a lo largo de todos los valores de x.

$$KS\ test\ statictic = sup_x\ |F_a(x) - F_b(x)| \quad (5)$$

- *Wasserstein Metric*: La distancia de Wasserstein calcula el costo mínimo requerido para transformar una distribución en otra. Este costo se representa por la cantidad de "trabajo" necesaria para mover "tierra" (puntos de datos) de una distribución a otra. En el contexto de la detección de cambios en los datos, la distancia de Wasserstein puede ser utilizada para comparar la distribución de los datos de entrenamiento con la distribución de los nuevos datos que se están utilizando para la predicción. Si la distancia de Wasserstein está por encima de un cierto umbral, indica que hay una diferencia significativa entre las dos distribuciones, sugiriendo que ha ocurrido un cambio en los datos [13]. La expresión de la fórmula 6 dice que la distancia de Wasserstein de orden p entre dos distribuciones de probabilidad μ y v mide el "costo mínimo" para transformar una distribución en la otra, considerando una función de distancia *d(x,y)* en el espacio, donde γ es una medida que empareja las distribuciones μ y v.

$$W_p(\mu, v) = \ inf_{\gamma \in \Gamma(\mu,v)}\ (E_{(x,y)\sim\gamma} d(x,y)^2)^{1/p} \quad (6)$$

- *Jensen–Shannon Divergence* (JSD): La divergencia de Jensen-Shannon es una medida de la distancia entre dos distribuciones de probabilidad. Se basa en la divergencia de *Kullback-Leibler*, pero es simétrica, lo que significa que no depende de cuál distribución se considera la "referencia" y cuál se considera la "comparación" [14]. Siendo P y Q las distribuciones y M a la suma de las distribuciones dividido en 2 la divergencia JSD se calcula mediante la fórmula 7.

$$JSD(P||Q) = \frac{1}{2} D(P||M) - \frac{1}{2} D(Q||M) \quad (7)$$

3.6 Herramientas Por Utilizar y Tecnologías

Se definen las herramientas o bibliotecas que serán utilizadas en el trabajo a desarrollar. Estas herramientas estarán a cargo de la visualización de métricas claves a los usuarios, detección de degradación, despliegue del modelo junto con la creación y despliegue final de la herramienta de reentrenamiento.

- Prometheus: Servicio que se encarga de la recolección de métricas de un sistema o servicio en tiempo real. Estas métricas son almacenadas para su visualización o uso por alguna otra herramienta.
- Grafana: Plataforma de visualización y creación de tableros dinámicos. Los tableros muestran información en tiempo real obtenida de diversas fuentes como Prometheus. Grafana también nos permite generar alertas en base a la definición de umbrales anómalos.
- Evidently AI: Biblioteca especializada en la evaluación de modelos de Machine Learning en ambientes productivos. Mide la calidad y la existencia de *data drift* en los datos utilizados para realizar inferencias generando informes detallados.
- Streamlit: Biblioteca de Python que permite desarrollar aplicaciones web interactivas de forma simple con enfoque a la visualización de datos y el campo de Machine Learning.
- Flask: *Microframework* de Python que se utiliza para desarrollar aplicaciones web. Se llama *microframework* porque es ligero y simple, proporcionando solo las herramientas y funcionalidades básicas necesarias para crear una aplicación web, sin imponer una estructura rígida o incluir bibliotecas adicionales por defecto.
- Scikit Learn: También conocido como *sklearn*, es una biblioteca de Pyhton que proporciona herramientas simples y eficientes para el análisis de datos y el Aprendizaje Automático.
- MLflow: Plataforma que permite el despliegue de modelos de Machine Learning y la gestión de estos en un ambiente productivo.
- Docker: Aplicación que permite la creación y distribución de aplicaciones de forma rápida y sencilla por medio de contenedores que aseguran la ejecución de estas aplicaciones en cualquier entorno.

Para evidenciar la forma en que estas herramientas y tecnologías se relacionan y colaboran entre sí en el proyecto se presenta el diagrama de arquitectura en la Figura 7.

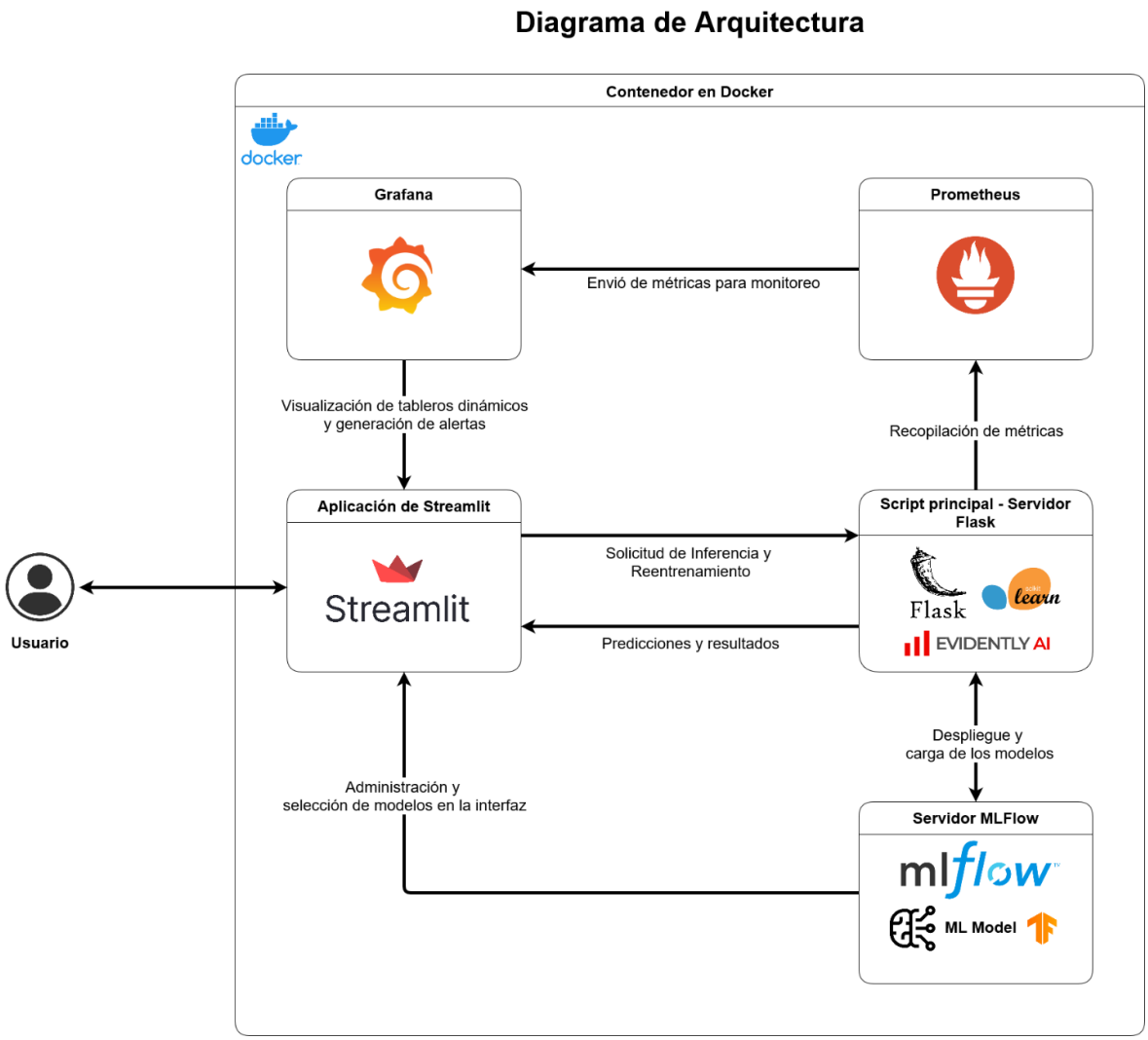


Fig. 7 Diagrama de arquitectura de la aplicación. Fuente: Elaboración propia.

3.7 Antecedentes metodológicos

El problema de la degradación de modelos de Machine Learning ha ganado una considerable tracción a lo largo de la última década con la masificación de esta tecnología abordando un sin fin de aplicaciones. En respuesta a esta problemática, la comunidad de investigación del campo ha desarrollado una variedad de enfoques y técnicas destinadas a mitigar la degradación de los modelos [15]. Estos enfoques van desde métodos de detección, identificación o regularización hasta técnicas de ajuste continuo y estrategias de reentrenamiento de modelos. En el presente estado del arte se destaca la importancia de comprender las causas subyacentes de la degradación del rendimiento de los modelos, así como la necesidad de desarrollar herramientas y metodologías robustas para monitorear, diagnosticar y mitigar este fenómeno de manera efectiva.

Entre los aportes más significativos, se puede destacar el estudio "Temporal quality degradation in AI models" de Vela et al. (2022) [16], centrado en la detección de *data drift*, en el cual se analizaron 4 modelos populares entrenados con 32 conjuntos de datos, correspondientes a diferentes áreas de la industria: salud, finanzas, transporte y clima. En este estudio, se evaluó el MSE (error medio al cuadrado) observado durante la fase de entrenamiento y se comparó con el MSE correspondiente a un valor que dependía de una variable temporal: los días en los que se realizaron mediciones de los datos a futuro.

La Figura 8 ilustra este proceso en el contexto de un modelo de red neuronal entrenado con un conjunto de datos meteorológicos. En el gráfico, la línea negra muestra la diferencia en el MSE a medida que pasan los días después del entrenamiento inicial. La línea amarilla representa el caso con la menor diferencia en el error (es decir, el mejor caso), mientras que la línea roja indica el caso con la mayor diferencia en el error (el peor caso). Como se puede observar, a medida que transcurren más días desde el entrenamiento, la diferencia en el error aumenta significativamente en algunos casos, lo que sugiere una fuerte presencia de *data drift*. Este patrón fue observado en un 91% de los modelos estudiados, indicando una degradación generalizada en la precisión de los modelos con el paso del tiempo.

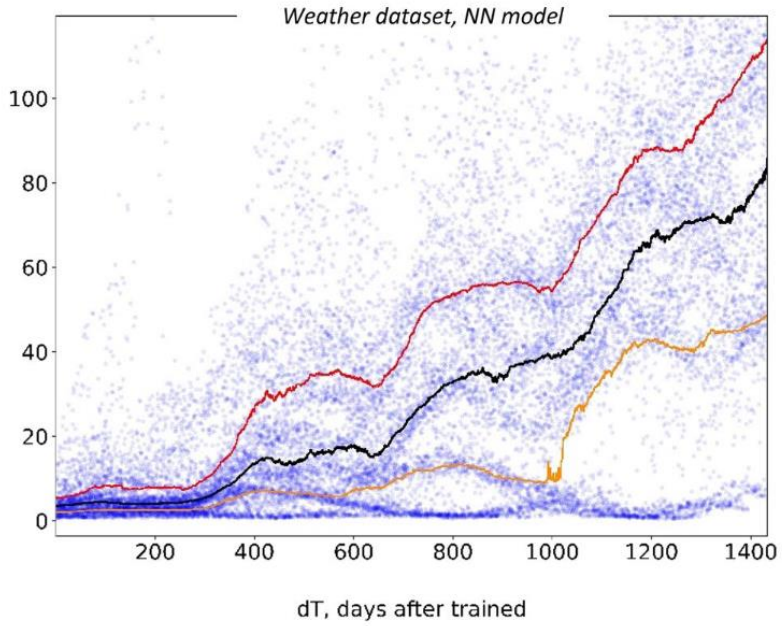


Fig. 8 Gráfico de la diferencia de error en relación con los días en un modelo de red neuronal entrenado con data del climatológica. Fuente: Extraído de [16]

Estos resultados subrayan la importancia de la temporalidad en la generación de *data drift* y la necesidad de monitorear continuamente los modelos para detectar y mitigar este fenómeno, especialmente cuando se introducen nuevos datos que reflejan cambios en las condiciones externas.

Por otro lado, en lo que respecta a estrategias y métodos de mitigación de este problema existen diversos estudios, entre los cuales se encuentra *Matchmaker* [15]. En este artículo se postula un nuevo método capaz de afrontar el *data drift* en sistemas de alta escalabilidad reduciendo también el impacto del *model drift* o *concept drift*. *Matchmaker* funciona de la siguiente forma: "La idea clave de *Matchmaker* es identificar dinámicamente el conjunto de datos de entrenamiento que es más similar a cada muestra de prueba, y usar el modelo de aprendizaje automático entrenado con esos datos para realizar inferencias. La intuición es que, si una muestra de prueba comparte una distribución de datos subyacente similar con un conjunto de datos de entrenamiento, es más probable que el modelo de aprendizaje automático entrenado a partir de ese conjunto sea más preciso" [15].

Para poner a prueba el método *Matchmaker*, se realiza una comparación con otros algoritmos creados para mitigar el data drift, utilizando dos conjuntos de datos experimentales: "Scouts" y "VMCPU". Estos métodos no requieren que todos los datos afectados reales por drift estén anotados, aunque la anotación es crucial para la experimentación, el posterior entrenamiento y evaluación del modelo. En producción, se adaptan utilizando los datos disponibles y las inferencias previas. En el contexto de *Matchmaker*, la rapidez del entrenamiento es gestionada mediante la selección dinámica del conjunto de datos más similar, que ya ha sido preprocesado y está listo para uso inmediato. Si bien el entrenamiento completo de un modelo puede llevar horas o días, la inferencia y la adaptación usando modelos preentrenados son procesos rápidos que pueden ejecutarse en tiempo real o casi real. En los métodos, el uso de 7 conjuntos de datos representa una ventana de tiempo específica que puede variar según el contexto, equilibrando la frescura y la cantidad de datos para entrenar un modelo robusto. Una descripción de los métodos ocupados y el funcionamiento que poseen es la siguiente:

- Win: Utiliza un modelo predictivo entrenado en los últimos 7 conjuntos de datos para predecir los datos entrantes.
- One: Utiliza un modelo predictivo entrenado solo en el conjunto de datos más reciente para predecir los datos entrantes.
- Accuracy Updated Ensemble [17]: Utiliza un conjunto de 7 modelos que se vuelven a entrenar después de cada conjunto de datos recibido. Los pesos de votación para cada modelo se determinan por su precisión en el conjunto de datos más reciente.
- DriftSurf [18]: Utiliza un enfoque adaptativo basado en la detección de cambios. El método DriftSurf detecta cambios en la distribución de datos y responde entrenando nuevos modelos en datos recientes. Este algoritmo mantiene modelos históricos y nuevos, alternando entre ellos durante la detección del cambio y seleccionando el modelo que mejor se desempeñe una vez que el cambio se estabiliza.

En la Figura 9, se puede observar cómo la latencia de inferencia (tiempo de respuesta del modelo) se relaciona con la precisión promedio de varios métodos, mostrando la efectividad de estos para mitigar el *data drift*.

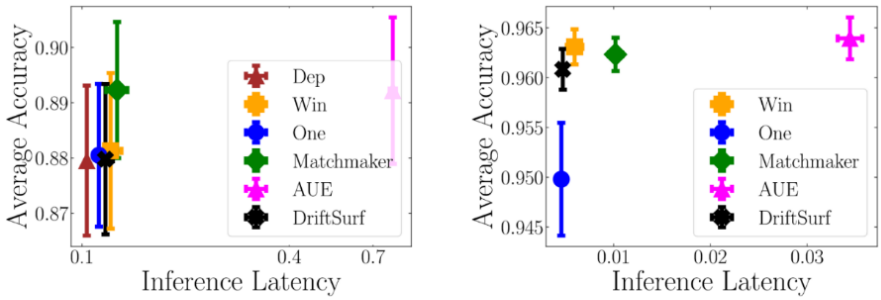


Fig. 9 Precisión vs latencia de los métodos de prevención de data drift en dos modelos de estudio (a) y (b). Fuente: Extraído de [15]

Nuevamente, en la anterior investigación se le hace énfasis a la importancia del monitoreo de los modelos y los datos de entrenamiento con relación a los datos nuevos. Estas soluciones tentativas al problema del *data drift* pueden clasificarse ampliamente en tres categorías [15]:

- Métodos basados en ventanas: Estos métodos utilizan una ventana deslizante sobre datos antiguos para entrenar nuevos modelos. Esta ventana se desplaza a medida que llegan nuevos datos, permitiendo la adaptación del modelo a cambios graduales en la distribución de los datos.
- Métodos de detección de cambio: Estos métodos emplean pruebas estadísticas para detectar el *data drift* y reentrenan los modelos solo cuando detectan que ha ocurrido el *drift*. Esto permite una respuesta más específica y puntual a los cambios en la distribución de los datos.
- Métodos basados en conjuntos: Estos métodos entrenan un conjunto de modelos con datos de entrenamiento antiguos y toman un promedio ponderado de sus predicciones. Esto permite una mayor robustez y estabilidad al combinar múltiples modelos entrenados en diferentes momentos en el tiempo.

Una investigación reciente, titulada "Retrain AI Systems Responsibly! Use Sustainable Concept Drift Adaptation Techniques" de Poenaru-Olaru et al. (2023) [20], explora métodos para mitigar el fenómeno del *model drift* y propone tres estrategias principales para combatir la degradación de modelos de Machine Learning. Estas estrategias se dividen en tres categorías: adaptación del conjunto de datos, adaptación del modelo y adaptación de la frecuencia.

- Adaptación del Conjunto de Datos: Esta categoría abarca técnicas como la ampliación periódica del conjunto de entrenamiento, el olvido gradual y el olvido abrupto. La ampliación periódica consiste en añadir de manera continua nuevos datos al conjunto

de entrenamiento. El olvido gradual, por su parte, pondera más los datos recientes mientras disminuye la influencia de los datos antiguos. El olvido abrupto, en contraste, elimina de manera repentina los datos más antiguos, manteniendo el tamaño del conjunto de datos relativamente constante.

- Adaptación del Modelo: Esta categoría se divide en dos enfoques principales: reinicio del modelo y reentrenamiento del modelo. El reinicio del modelo implica descartar el modelo actual y construir uno nuevo desde cero, mientras que el reentrenamiento consiste en actualizar el modelo existente con nuevos datos, preservando la estructura y conocimiento previos.
- Adaptación de la Frecuencia: Esta técnica puede ser ciega o informada. La adaptación ciega ajusta el modelo a intervalos regulares sin tener en cuenta el estado actual del *model drift*. En contraste, la adaptación informada detecta el *model drift* y realiza ajustes al modelo solo cuando es necesario, lo que evita modificaciones innecesarias y contribuye a un uso más eficiente de los recursos. La forma en que se adapta el modelo puede varias según la técnica y el enfoque específico utilizado.

Esta investigación destaca la importancia de elegir la técnica adecuada para mantener la eficacia de los sistemas de Inteligencia Artificial a medida que evolucionan los datos y los conceptos subyacentes.

Otro punto por tener en consideración en el estado del arte, referente a la mitigación de los efectos de la degradación de modelos de Machine Learning, son los estudios de casos con problemas reales ocasionados por la degradación. Un ejemplo de esto es un estudio que evalúa la degradación en modelos orientados a realizar inferencias para el área de la salud [19], donde modelos que predicen los cuadros médicos, tiempo de recuperación y hasta la fecha de mortalidad de los pacientes son cada vez más comunes. Siendo la fecha aproximada de mortalidad un tema tan delicado, la falta de atención al problema de la degradación se hace evidente en el estudio. Por medio del análisis de un modelo con datos reales, se descubre la existencia de degradación, proporcionando evidencia empírica de cómo el rendimiento de los modelos predictivos basados en Machine Learning puede deteriorarse con el tiempo después del entrenamiento, a pesar de un rendimiento inicial alto.

4. Marco metodológico

4.1 Modelos de estudio

Para entender cómo se aborda el monitoreo de la degradación de los modelos de Machine Learning primero se debe escoger el tipo de problema que se quiere abordar, las bases de datos a utilizar y los modelos con los que se trabajará.

En primer lugar, el problema al cual irá orientada la creación de los modelos es el de *ForeCasting* o predicción de eventos a futuro basándose en datos históricos secuenciales y su respectivo análisis. Por esta secuencialidad que deben tener los datos es que se escogieron dos tipos de set de datos, uno orientado a la predicción de una variable (la temperatura) y otro a la predicción de valores financieros en la bolsa.

- Set de datos de temperatura: Corresponde a las diversas mediciones climáticas obtenidas en la ciudad de Basilia en Suiza. En él se presentan mediciones de temperatura, nubosidad, agricultura, precipitaciones, radiación, viento y geopotencial. Las mediciones son realizadas de forma diaria en intervalos de un registro por hora a través de una serie de sensores desde el año 1940 a la actualidad, 7 de mayo del año 2024. El set de datos está compuesto por 7 columnas y 739.396 filas donde la primera columna describe la marca de tiempo donde fueron realizadas las 7 mediciones antes mencionadas (Figura 14). Extraído de WeatherBlue [21].
- Set de datos financiero: La base datos financiera está compuesta por los precios de apertura, valor máximo diario, valor mínimo diario, precio de cierre, volumen e interés de más de 500 acciones y empresas de la bolsa del estadounidense, de las cuales se escogieron los valores de Stock de Amazon, Apple, Google y Nvidia. Estos conjuntos de datos van desde el año 2000 aproximadamente hasta el año 2017. Los registros son ingresados de forma diaria con su respectiva marca de tiempo. Presentan una variación en el inicio de los datos existentes debido a la entrada al mercado de cada una de las empresas, pero las dimensiones corresponden a 7 columnas y aproximadamente 5000 filas (Figura 15). Extraído desde Kaggle [22].

timestamp	Temperature	Basel Precipitation Total	Basel Wind Gust	Basel Cloud Cover Total	Basel Sunshine Duration	Basel Mean Sea Level Pressure [MSL]	Basel Evapotranspiration
2013-01-01 00:00:00	0.950245	0.0	21.599998	29.100000	0.0	1012.7	0.00000
2013-01-01 01:00:00	0.860245	0.0	21.599998	29.400002	0.0	1012.6	0.00000
2013-01-01 02:00:00	1.200245	0.0	20.880001	28.800001	0.0	1012.5	0.00144
2013-01-01 03:00:00	1.770245	0.0	21.960000	28.200000	0.0	1012.1	0.00144
2013-01-01 04:00:00	2.310246	0.0	23.039999	29.400002	0.0	1011.4	0.00288

Fig. 10 Extracto del set de datos Meteorológico de la ciudad de Basilia. Fuente: Elaboración Propia.

	Date	Open	High	Low	Close	Volume	OpenInt
0	1997-05-16	1.97	1.98	1.71	1.73	14700000	0
1	1997-05-19	1.76	1.77	1.62	1.71	6106800	0
2	1997-05-20	1.73	1.75	1.64	1.64	5467200	0
3	1997-05-21	1.64	1.65	1.38	1.43	18853200	0
4	1997-05-22	1.44	1.45	1.31	1.40	11776800	0

Fig. 11 Extracto del set de datos financiero de Amazon. Fuente: Elaboración propia.

En segundo lugar, una vez definidas las bases de datos a utilizar se procede a la selección y confección de los modelos de estudio. Los modelos seleccionados corresponden a redes neuronales del tipo Redes Neuronales Recurrentes (RNN), *Long Short-term Memory* (LSTM) y Transformer. Estas redes neuronales se caracterizan por obtener buenos resultados en las predicciones de datos secuenciales o temporales debido a que poseen la capacidad de recordar información en los datos anteriores. Se escogen en este orden respondiendo a un problema en específico que se genera en cada una de estas redes.

- RNN: Modelo con la capacidad de recordar información de datos anteriores. Sin embargo, su capacidad para recordar información a largo plazo es limitada debido a un fenómeno conocido como "desvanecimiento del gradiente", que dificulta que la red aprenda patrones en secuencias largas. La arquitectura del modelo creado se puede apreciar en la Figura 16.

Model: "sequential"

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 50, 50)	2,600
dropout (Dropout)	(None, 50, 50)	0
simple_rnn_1 (SimpleRNN)	(None, 50, 50)	5,050
simple_rnn_2 (SimpleRNN)	(None, 50, 50)	5,050
simple_rnn_3 (SimpleRNN)	(None, 50)	5,050
dense (Dense)	(None, 1)	51

Total params: 35,604 (139.08 KB)
Trainable params: 17,801 (69.54 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 17,803 (69.55 KB)

Fig. 12 Arquitectura del modelo de red neuronal recursiva, obtenido mediante el comando Summary(). Fuente: Elaboración propia.

- LSTM: Es una variante de la RNN cuyo objetivo es responder al problema del "desvanecimiento del gradiente" haciéndola apta para trabajar con secuencias que generen dependencias a largo plazo gracias a la introducción de celdas de memoria y puertas que controlan el flujo de la información. Estos modelos son los más utilizados en la actualidad para responder a problemas de predicción de eventos a futuro. La arquitectura del modelo implementado es la siguiente:

Model: "sequential_1"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 50, 50)	10,400
lstm_1 (LSTM)	(None, 50)	20,200
dense_1 (Dense)	(None, 25)	1,275
dense_2 (Dense)	(None, 1)	26

Total params: 95,705 (373.85 KB)
Trainable params: 31,901 (124.61 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 63,804 (249.24 KB)

Fig. 13 Arquitectura del modelo LSTM, obtenido mediante el comando Summary(). Fuente: Elaboración propia.

- Transformer: Último modelo seleccionado que en la actualidad está ganando popularidad debido a un nuevo componente clave llamado "mecanismo de atención". Este mecanismo permite al modelo enfocarse en partes específicas de los datos secuenciales, sin necesidad de procesarlos en un orden estricto. En lugar de analizar los datos secuencialmente, el modelo evalúa todos los elementos de la secuencia a la vez, asignando diferentes niveles de importancia a cada uno, según su relevancia para la tarea en cuestión. De este modo, el modelo puede capturar dependencias tanto a corto como a largo plazo de manera más efectiva. La atención permite al modelo decidir cuáles partes de la secuencia son cruciales y cuáles pueden ser ignoradas, mejorando así la precisión y eficiencia del procesamiento. Además, al trabajar de forma no secuencial, estos modelos pueden ser entrenados en paralelo, lo que aumenta su eficiencia y escalabilidad. La estructura del modelo se describe en el Anexo 1.

Finalmente, estos 3 modelos son puestos a prueba, siendo entrenados con una fracción del 80% de los datos de los sets financieros y de temperatura, respectivamente, mientras que el 20% restante se utilizó para la predicción y evaluación. Esta proporción de 80:20 se eligió para asegurar un equilibrio entre tener suficientes datos para entrenar el modelo de manera efectiva y contar con un conjunto representativo para evaluar su desempeño en datos no vistos, lo que

permite detectar posibles problemas de sobreajuste. Cabe mencionar que los modelos son univariables, es decir, que se utilizó solo una variable para realizar predicciones. En el caso del entrenamiento financiero, se utilizan los valores de apertura de los Stocks, y en el caso del clima, se emplea la variable Temperatura.

4.2 Prueba N°1: Entrenamiento incremental

Para evaluar la eficacia del método de adaptación del modelo mediante reentrenamiento en la mitigación del *model drift*, se propone un experimento utilizando la base de datos de acciones de Amazon. En este experimento, se entrenan tres modelos diferentes con un 50% de los datos disponibles y se realizan predicciones sobre el siguiente 10% de los datos. Luego, se incrementa el porcentaje de datos de entrenamiento en pasos del 10% (60%, 70%, 80%, y 90%), reentrenando y posteriormente evaluando el modelo con el 10% siguiente de datos en cada etapa.

El objetivo es calcular el error cuadrático medio (MSE) para cada configuración, repitiendo el procedimiento 32 veces para obtener una medida representativa del desempeño del modelo y reducir el impacto de resultados atípicos. La importancia de este enfoque radica en observar cómo el error del modelo varía con el tamaño creciente del conjunto de datos de entrenamiento. El método de reentrenamiento es crucial en la adaptación del modelo para enfrentar el *model drift*, ya que permite que el modelo se ajuste continuamente a los nuevos datos sin necesidad de reiniciar el entrenamiento desde cero. La relación entre el tamaño del conjunto de datos de entrenamiento y el MSE resultante proporciona una visión clara de cómo el reentrenamiento impacta en la precisión del modelo, ofreciendo una evaluación integral de su capacidad para mantener la exactitud a medida que se incrementa la cantidad de datos. Este análisis es esencial para entender cómo el reentrenamiento puede ser una técnica efectiva para mitigar la degradación del modelo y mejorar su rendimiento frente a cambios en los datos.

El pseudocódigo del experimento realizado describiendo los pasos que se utilizaron para calcular las predicciones respetando el incremento del 10% en el entrenamiento del modelo es el siguiente. Cabe recalcar que la función “creación del modelo” varía según el modelo que se crea, especialmente en el caso del Transformer y sus ventanas de atención.

```
1. Procedimiento Entrenamiento y Predicción de los Modelos
 2. 1. Inicializar Parámetros:
 3.     largo de la secuencia de entrada, seq_length = 60
 4.     tamaño inicial porcentual del conjunto de entrenamiento, train_size_initial = 0.5
 5.     incremento porcentual del conjunto de entrenamiento train_size_increment = 0.1
 6. 2. Descargar y Preparar Datos:
 7.     leer datos de 'amzn.us.txt'
 8.     convertir 'Date' a formato datetime y establecer como índice
 9.     normalizar datos de 'Open' entre 0 y 1 con MinMaxScaler
```

```
10. 3. Definir Funciones Auxiliares:
11.     Función train_test_split(data, train_size)
12.         dividir datos en conjuntos de entrenamiento y prueba según train_size
13.     Fin-Función
15.     Función create_sequences(data, seq_length)
16.         crear secuencias X e y a partir de los datos con longitud seq_length
17.     Fin-Función
19.     Función create_model()
20.         definir y compilar los modelos
21.     Fin-Función
22. 4. Ejecutar Proceso 32 veces:
23.     Para iteration = 1 hasta 20 se hace
24.         Inicializar listas mse_train y mse_test
25.         Para i = 0 hasta 4 se hace
26.             Calcular train_size = train_size_initial + i * train_size_increment
27.             Si train_size > 0.9, salir del bucle
29.             Calcular índices y fechas de inicio y fin para los conjuntos de train y test
31.             Crear el modelo llamando a create_model()
33.             Dividir los datos en train_data y test_data usando train_test_split
34.             Crear secuencias X_train y y_train con create_sequences
35.             Si X_train está vacío, continuar con la siguiente iteración
37.             Reajustar dimensiones de X_train
38.             Entrenar el modelo con X_train y y_train
40.             Predecir valores en el conjunto de entrenamiento y calcular mse_train
41.             Añadir mse_train a la lista mse_train
43.             Crear secuencias X_test y y_test con create_sequences usando test_data
44.             Si X_test está vacío, continuar con la siguiente iteración
46.             Reajustar dimensiones de X_test
47.             Predecir valores en el conjunto de prueba y calcular mse_test
48.             Añadir mse_test a la lista mse_test
50.             Guardar predicciones escaladas en archivo 'y_pred_rescaled_part_{i + 1}.csv'
51.         Fin-Para
53.         Guardar resultados de mse_train y mse_test en 'train_mse.csv' y 'test_mse.csv'
55.     Fin-Para
56. Fin Procedimiento Entrenamiento y Predicción de los Modelos
```

4.3 Prueba N°2: Simulación Ambiente Productivo

Esta prueba tiene como objetivo evaluar el desempeño de los modelos en un entorno productivo simulado. Se busca analizar cómo el algoritmo maneja las predicciones cuando los datos se alejan temporalmente del conjunto de entrenamiento. El objetivo principal es observar la degradación del modelo en respuesta a los cambios en las tendencias de los datos. Aprovechando la naturaleza de la prueba, que busca simular un entorno productivo, se

introducen algunas de las herramientas que serán empleadas en el desarrollo de la aplicación final.

Para llevar a cabo esta simulación, se despliega uno de los modelos en un *endpoint* utilizando MLflow. La aplicación creada con Python y Flask permite cargar datos y seleccionar fechas para las predicciones. El modelo por utilizar es el LSTM en base a su amplia utilización en problemas de regresión y *forecast* como se mencionó anteriormente, así como en su capacidad para ofrecer una buena relación entre precisión y tiempos de ejecución.

El procedimiento consiste en entrenar el modelo LSTM con datos financieros de Amazon desde el 1 de enero de 2000 hasta el 31 de diciembre de 2003. Una vez entrenado el modelo, se realiza la predicción de manera constante en intervalos de 3 meses, avanzando hasta el final del conjunto de datos. Mientras se generan las predicciones, se calculan métricas en tiempo real, como el MAE, RMSE y R^2, para evaluar el desempeño y detectar posibles degradaciones. El pseudo código simplificado de este experimento es el siguiente:

```
Procedimiento de Predicción y Actualización de Métricas

1. Inicializar Aplicación y Parámetros:
    Inicializar aplicación Flask
    Definir métricas de Prometheus

2. Definir Funciones Auxiliares:
    Función get_mlflow_metrics(run_id)
        Obtener métricas de MLflow
    Fin-Función

    Función generate_evidently_metrics(reference_data, current_data)
        Generar métricas de data drift usando Evidently
    Fin-Función

    Función load_model(run_id)
        Cargar modelo desde MLflow
    Fin-Función

    Función load_and_preprocess_data(file_path, start_date, end_date)
        Leer y preprocesar datos desde archivo
        Normalizar datos con MinMaxScaler
    Fin-Función

    Función make_predictions(model, test_data, scaler)
        Hacer predicciones con el modelo y desescalar
    Fin-Función

    Función log_metrics_to_mlflow(rmse, mae, r2, data_drift_score, data_drift_percentage, data_drift,
run_id)
        Registrar métricas en MLflow
    Fin-Función

    Función update_metrics(run_id, reference_data, current_data)
        Actualizar métricas de Prometheus
        Generar métricas de data drift
```

```
36.     Fin-Función
38.     Función calculate_and_update_metrics(model, test_data, y_true, scaler, run_id, reference_data,
current_data)
39.         Hacer predicciones y calcular métricas (RMSE, MAE, R2)
40.         Actualizar métricas de data drift
41.         Registrar métricas en MLflow
42.         Guardar métricas en CSV
43.     Fin-Función
45. 3. Definir Endpoint de Flask:
46.     Endpoint '/predict':
47.         Obtener datos del request
48.         Cargar y preprocesar datos
49.         Cargar modelo
50.         Calcular y actualizar métricas
51.         Retornar respuesta
52.     Fin-Endpoint
54. 4. Inicializar Servidores:
55.     Inicializar servidor HTTP de Prometheus
56.     Inicializar servidor Flask
58. 5. Bucle Principal:
59.     Mientras True se hace
60.         Calcular y actualizar métricas cada 10 segundos
61.         Esperar 10 segundos
62.     Fin-Mientras
64. Fin Procedimiento de Predicción y Actualización de Métricas
```

4.4 Monitoreo con Prometheus y Grafana

Del experimento anterior obtenemos métricas claves para detectar una degradación en el desempeño de los modelos en una simulación de ambiente productivo donde los modelos desplegados son ejecutados de forma constante generando un gran número de predicciones. En esta simulación se logra obtener las métricas a lo largo del tiempo, pero tan solo para un tiempo definido. En la práctica no existe un límite en el tiempo de uso de los modelos y estas métricas deben ser obtenidas en todo momento si se quiere detectar la degradación a tiempo.

Para responder a esta problemática se utilizan las herramientas Prometheus y Grafana, por medio de la librería *prometheus_client* de Python se logra disponer de las métricas críticas en un *endpoint* vinculado a un puerto de salida, Prometheus se encarga de tomar estos datos direccionándolos a su interfaz para que Grafana pueda acceder a estos y registrar por medio de tableros dinámicos todos los cambios que sufren estas métricas críticas a largo y corto plazo de forma continua y en tiempo real. Grafana también nos permite generar alertas, lo cual es clave en la fase de monitoreo para detectar la degradación. Estas dos herramientas son instaladas mediante Docker con *docker-compose* ejecutando ambos contenedores al mismo tiempo junto

con los respectivos nodos de extracción de datos de Prometheus. Capturas de las ventanas de las distintas aplicaciones se pueden encontrar en los anexos 2, 3 y 4.

4.5 Implementación de una aplicación para detectar, monitorear y combatir el *drift*

Para abordar el problema de degradación, se seleccionan las siguientes metodologías: Como se ha discutido previamente, una estrategia eficaz es el monitoreo continuo junto con la implementación de técnicas para mitigar el *data drift*, utilizando métodos estadísticos para detectar cambios en las distribuciones (Método de Detección de Cambio). Además, para combatir el *model drift*, se adoptará un enfoque adaptativo mediante el reentrenamiento del modelo (Adaptación del Modelo). Con este fin, se desarrollará una aplicación que facilitará la solicitud de predicciones, el monitoreo de *drift* y el reentrenamiento de modelos de machine learning. El desarrollo de la aplicación iniciará con la creación de un diseño preliminar que definirá las páginas y funciones de esta.

- *Somehow AI Manage*
    - Página principal:
        - Página de solicitud de predicción:
        - Permite subir un archivo de set de datos para hacer una predicción con el modelo.
        - Permite especificar el modelo a utilizar (Run ID en MLflow)
        - Permite escoger el *target* o columna objetivo a predecir del set de datos.
        - Permite limitar las fechas a usar del set de datos, opcional, ignorar si no se dan.
        - Botón de solicitar predicción.
        - Botón de informe de *data drift* usando Evidently AI.
        - Solicitar la base de datos de entrenamiento del modelo para hacer el informe con los datos de predicción.
        - Mostrar los resultados de la predicción y el informe de *data drift* si así se solicita.
        - Incluye un menú desplegable en la parte superior por debajo del logo de la página, aquí nos deja ingresar a todas las páginas disponibles:
            - Página Principal
            - Monitoreo
            - Reentrenamiento del modelo
    - Monitoreo:
        - Conexión con un tablero ya creado de Grafana, muestra los distintos elementos de este *dashboard*.

        - Nos permite visualizar alertas generadas en Grafana. Las alertas que se generen en Grafana deben ser desplegadas en cualquier página como notificaciones interactivas para informar a los usuarios.
    - Reentrenamiento del modelo
        - Seleccionar el modelo a reentrenar de MLflow.
        - Subir la nueva base de datos para realizar el entrenamiento.
        - Desplegar visualización de cómo se realiza el entrenamiento con tiempo estimado, etc.
        - Opción de limitar fechas del set de datos, opcional.

Una vez listo el diseño se confecciona la página principal. Para esto se utiliza la librería de Streamlit para crear el *frontend* de la aplicación. Streamlit permite confeccionar una página de manera simple y rápida, especial para la visualización y el despliegue en el campo de la ciencia de datos y la inteligencia artificial. La página principal muestra las casillas con la información necesaria para solicitar una predicción de uno de los modelos desplegados en MLFlow, al momento de solicitar esta predicción se despliega un *spinner* que señala que el modelo está trabajando, una vez terminado se muestra un gráfico interactivo de los valores predichos junto a los valores reales correspondientes a la fecha señalada ya que los modelos utilizados corresponden a modelos de *forecast* financiero. Adicionalmente se muestra un *dataframe* interactivo de los valores presentes en el gráfico y las fechas correspondientes, al lado de este cuadro también se muestra otro con métricas de evaluación de la predicción, entre las métricas que se muestran tenemos el MAE y RMSE. Estos cuadros de *dataframe* pueden ser descargados para su posterior análisis gracias a las funcionalidades de Streamlit.

También al solicitar una predicción se puede generar un informe de *data drift* detallado de cada una de las columnas del set de datos utilizado en la predicción en relación con el set de datos de entrenamiento del modelo, este informe se genera en formato HTML y es obtenido mediante la librería Evidently AI. Esta librería se especializa en detectar cambios en las distribuciones entre dos conjuntos de datos: el conjunto de referencia (en este caso, el de entrenamiento) y el conjunto de comparación (el utilizado para realizar las predicciones). Evidently AI realiza estos cálculos empleando varios métodos, como la prueba de Kolmogorov-Smirnov (utilizada por defecto), la distancia de Wasserstein, y la divergencia de Jensen-Shannon. Importante es señalar que el conjunto de datos de entrenamiento también debe ser entregado por el usuario a la plataforma.

La página de monitoreo muestra 6 tableros creados en la aplicación Grafana, estos tableros corresponden a: Detección de d*ata drift*, Porcentaje de *data drift*, MAE, RMSE, r^2 y KS test. Cada una de las métricas son generadas y enviadas a Prometheus al momento de solicitar una predicción, posteriormente Grafana recibe estos datos recolectados por Prometheus y los muestra en tableros. Mostrar estos gráficos en la página de monitoreo se hace por medio de

Embed HTML, opción que debe ser habilitada en la instalación de Grafana por motivos de seguridad. En la sección subsiguiente a los tableros se despliegan las alertas activas en Grafana, estas alertas son definidas de forma manual junto a umbrales anómalos para cada una de las métricas monitoreadas (Tabla 1).

Métrica	Umbral medio	Umbral crítico
data drift	0 (No hay *data drift*)	1 (Presencia de *data drift*)
% de data drift	30% de las columnas	60% de las columnas
MAE	Sobre 0.2	Sobre 0.5
RMSE	Sobre 0.7	Sobre 1
r^2	Bajo 1	Bajo 0.8
KS p value	Bajo 0.8	Bajo 0.5

Tabla 1 Umbrales definidos para generar las alertas de Grafana. Fuente: Elaboración propia.

Para notificar al usuario que existen alertas activas o que es necesario reentrenar el modelo con el cual se está trabajando se utiliza la barra lateral de Streamlit, aquí se despliega el número de alertas activas, si es que no hay ninguna alerta activa o si alguna de las alertas activas es crítica mostrando una alerta sugiriendo el reentrenamiento. También se incluye un botón del tipo *popover* que despliega un menú para subir el archivo de un modelo tipo *keras* a las dependencias de MLFlow, este modelo se incorpora a la lista de modelos operativos listo para ser utilizado en nuevas predicciones o para su reentrenamiento.

Por último, la página de Reentrenamiento utiliza casillas para el ingreso de datos similar a como funciona la página de solicitud de predicción, al momento de solicitar el reentrenamiento este se guarda de manera automática en MLFlow y se añade a la interfaz de la página para solicitar predicciones o un nuevo reentrenamiento.

Con lo que respecta a la forma en que se suben y despliegan los modelos dentro de la aplicación se tiene que se ha automatizado la forma en que se carga y genera la data utilizada en la solicitud de predicciones con relación a la forma que posee la primera capa del modelo en cuestión, por lo tanto, no importa la forma del *input_shape* de la primera capa del modelo, la aplicación siempre será capaz de generar la predicción. Por otro lado, en cuanto al formato del modelo, este debe ser en formato *keras* recalcando que el *framewotk* seleccionado para trabajar en el proyecto fue Tensorflow. Para poder cargar otro tipo de modelo se debe cambiar las líneas de código del archivo *flask_server.py* en la función *load_model_func*.

4.6 Creación de un contenedor de la aplicación

El paso final del proyecto amerita la creación de un contenedor utilizando la plataforma de Docker respondiendo a la necesidad de desplegar la aplicación de forma rápida, efectiva y

eficiente a todos los usuarios que quieran hacer uso de ella o que deseen interiorizar aún más en el trabajo realizado. Para esto se utiliza Docker-compose en la creación de un levantamiento múltiple de las imágenes de Docker de Grafana y Prometheus junto la creación de una imagen nueva para nuestra aplicación y la llamada del comando necesario para la ejecución del *endpoint* donde se administran los modelos de la mano de MLflow. El archivo *docker-compose.yml* creado se describe en el anexo 5 y en él se aprecia la creación de los volúmenes necesarios para arrancar la aplicación, los respectivos puertos que usan las herramientas y la creación de una red que les permita a estas aplicaciones comunicarse siendo visibles entre ellas. Se añade un archivo *requirements.txt* con todas las librerías necesarias para generar el ambiente propicio para la aplicación dentro del contenedor.

Junto a la confección de esta arquitectura se preparan los directorios respectivos donde se copian los volúmenes utilizados durante la fase de monitoreo a las carpetas del directorio del proyecto, este paso garantiza que las configuraciones, tableros creados y acceso a las APIs de las herramientas utilizadas sea inmediato y consistente para cualquier usuario. Adicionalmente se agregan 3 modelos como demo, configurados para que MLflow los cargue de forma automática junto con el despliegue de la aplicación, estos modelos corresponden a una red RNN, LSTM y Transformer entrenados con los datos de los valores del Stock de Amazon.

Finalmente se confecciona el archivo *Dockerfile* que crea el contenedor y se ejecuta el comando *docker-compose build* seguido de *docker-compose up* para desplegar la aplicación. De manera local esta aplicación se despliega en la dirección http://localhost:8501. Mayor detalle se puede encontrar en el repositorio adjunto a la memoria (Anexo 6) el cual brinda acceso a la instalación vía Docker, las bases de datos utilizadas, modelos de demostración y algunos de los *notebooks* en los que se trabajó conforme avanzaba el proyecto.

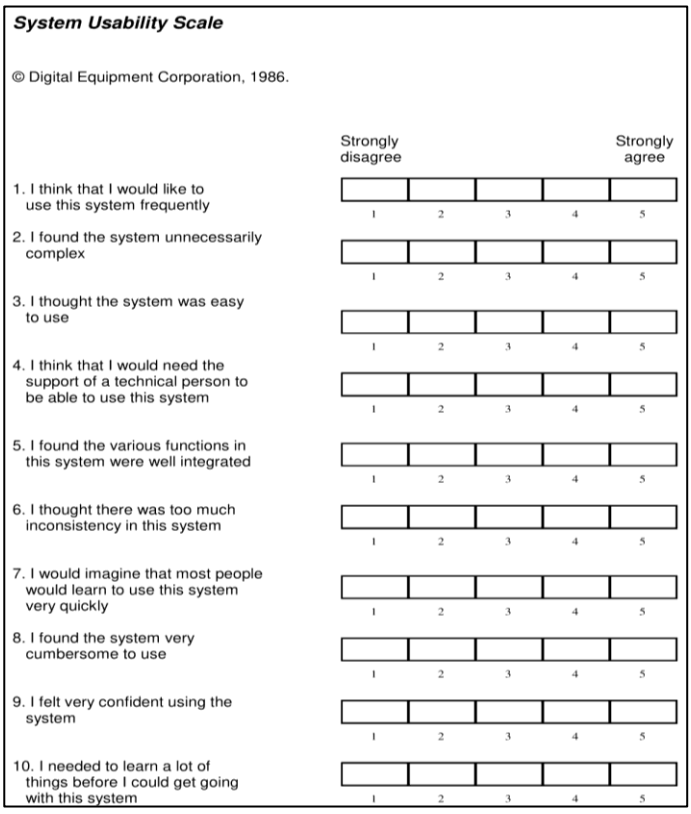
```
#Dockerfile
FROM python:3.10.12-slim

WORKDIR /usr/src/app

COPY app/ .

RUN pip install -r requirements.txt

ENV MLFLOW_TRACKING_URI http://127.0.0.1:5000

EXPOSE 8501 8000 5001 5000

CMD ["streamlit", "run", "run.py"]
```

Para comprobar la robustez de la aplicación se realizaron numerosas pruebas, tanto en cómo se suben los modelos, la solicitud de predicción, el correcto monitoreo y los reentrenamientos. También se realiza un despliegue tanto en Windows utilizando WSL y en un dispositivo con Ubuntu como sistema operativo nativo para asegurar su correcto funcionamiento.

4.7 Encuesta de usabilidad de la aplicación

Para evaluar el desempeño de la aplicación creada en materias de usabilidad se aplica una evaluación que mide el nivel de satisfacción de los usuarios al interactuar con la aplicación, la escala SUS. “La escala de usabilidad del sistema (SUS, por sus siglas en inglés) es una escala simple de diez elementos que ofrece una visión global de las evaluaciones subjetivas de la usabilidad” [23].

Se les solicita a los usuarios que utilicen todas las funciones que la aplicación tiene para ofrecer, en base eso señalan en 10 afirmaciones el grado de desacuerdo o de acuerdo en una escala de 5 opciones, como se puede apreciar en la Figura 14. En total son 100 puntos los que se pueden obtener con las preguntas 1, 3, 5, 7 y 9 contribuyendo el numeró de la escala menos 1 y las preguntas 2, 4, 6, 8 y 10 otorgan 5 puntos menos la posición de la selección en la escala, los puntajes obtenidos son multiplicados por 2.5 obteniendo el resultado final.



System Usability Scale

© Digital Equipment Corporation, 1986.

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

Fig. 14 Cuestionario de la escala SUS. Fuente: extraído de [22].

En total 6 personas son solicitadas para que experimenten con la página y respondan siguiendo la escala SUS. De estas 6 personas 4 corresponden a estudiantes de la carrera de ingeniería civil en computación, entre ellos estudiantes ya egresados de la misma con un grado general de entendimiento sobre las cualidades de la aplicación y los temas que en ella se abordan, el resto de encuestados no presentan mayores conocimientos al respecto obteniendo otra mirada no tan técnica sobre la usabilidad de la aplicación.

5. Resultados o secciones temáticas

5.1 Resultados preliminares

A continuación, se presenta uno de los resultados obtenidos en la prueba preliminar, en la cual cada modelo debía predecir el 20% final de cada uno de los dos conjuntos de datos. La Figura 18 muestra los resultados de los modelos RNN, LSTM y Transformer, entrenados con el 80% de los datos correspondientes al valor de apertura del Stock de Nvidia (línea azul). La línea verde representa el valor real del 20% restante, mientras que las líneas roja y amarilla corresponden a las predicciones realizadas por los respectivos modelos desde el valor 50 hasta el final, lo que permite una mejor comparación entre los valores reales y las predicciones.

Fig. 15 Puesta a prueba de los 3 modelos de estudio seleccionados bajo las mismas condiciones.

5.2 Resultados Prueba N°1

En los resultados de la Prueba 1, se evaluó la eficacia del reentrenamiento incremental en la mitigación del *model drift* para los modelos RNN, LSTM y Transformer, utilizando diferentes porcentajes de datos de entrenamiento. Los modelos RNN y LSTM mostraron un desempeño más estable y consistente, con reducciones en el MSE en algunos casos donde se incrementaba el conjunto de datos. En contraste, el modelo Transformer presentó una mayor variabilidad en el error, especialmente con mayores volúmenes de datos de entrenamiento. Esto

demuestra que el reentrenamiento no contribuye significativamente en la degradación, aunque puede variar dependiendo de factores externos que acompañan a la temporalidad de los datos.

Prueba 1 RNN: La red neuronal recursiva presenta los siguientes resultados, donde las gráficas a) a e) muestran las predicciones de los modelos entrenados con porcentajes crecientes de los datos (50%, 60%, 70%, 80%, 90%) y su comparación con los valores reales en una iteración. La línea verde representa los valores reales del precio de apertura, mientras que la línea amarilla muestra las predicciones generadas por el modelo. Se observa que, aunque el ajuste inicial es razonable, las discrepancias aumentan a medida que se entrena con más datos, especialmente en los casos de 60% y 90%. La gráfica f) resume el error cuadrático medio (MSE) promedio, indicando un incremento del error conforme se entrenan los modelos con más datos, lo que sugiere una posible degradación en la precisión del modelo.

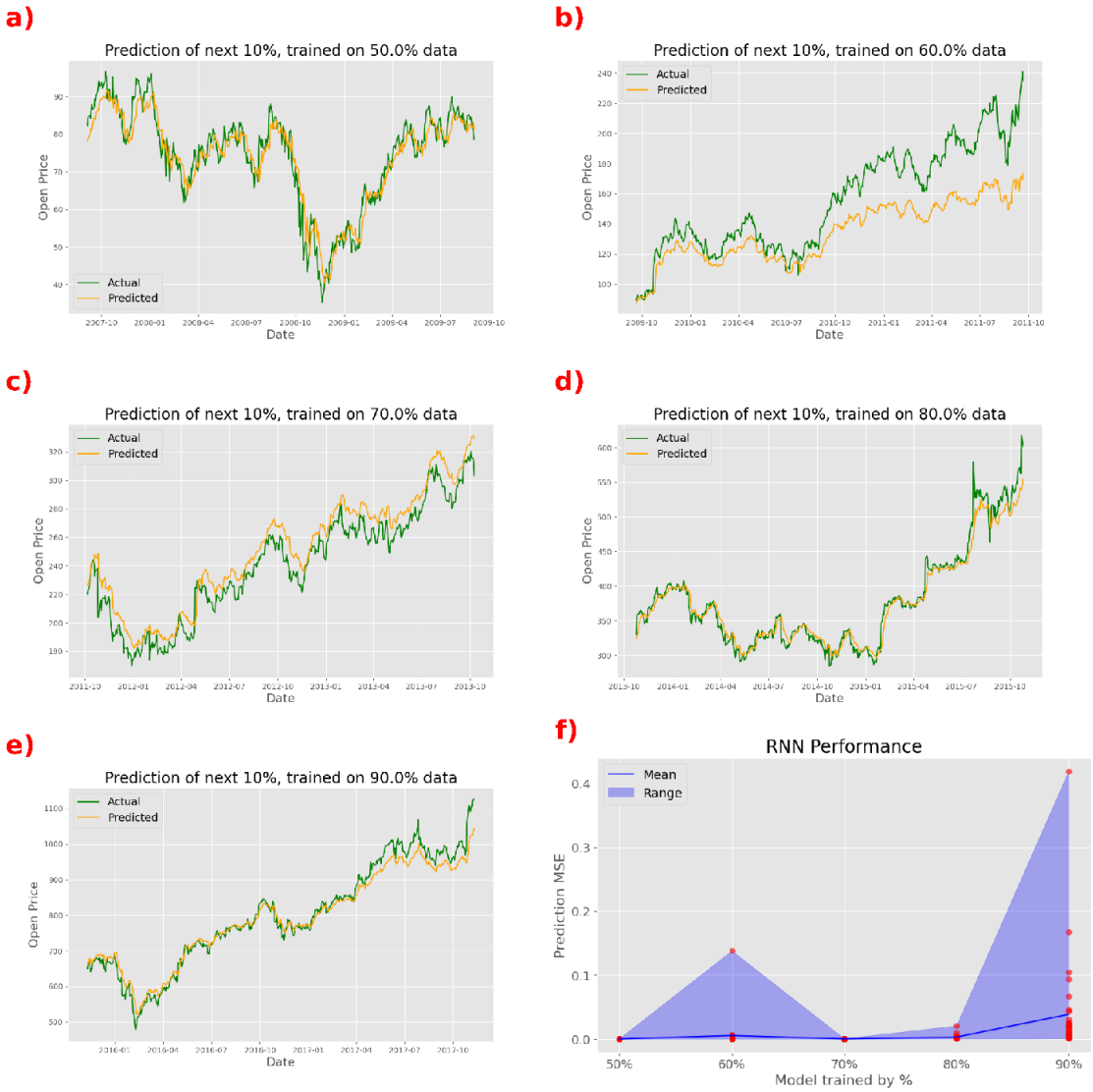


Fig. 16 a), b), c), d) y e) corresponden a gráficos de las predicciones junto a los valores reales de una de las 32 iteraciones durante el experimento 1 con el modelo RNN. El gráfico f) muestra el desempeño del modelo en cuanto MSE obtenido en el total de las predicciones.

Prueba 1 LSTM: En la Figura 19, se observan los resultados de las predicciones realizadas por el modelo LSTM entrenado con diferentes porcentajes del conjunto de datos de acciones de Nvidia. La gráfica f) ilustra el rendimiento general del modelo LSTM, mostrando el MSE promedio y el rango de variación del error a medida que aumenta el porcentaje de datos utilizados para el entrenamiento. Se observa que el LSTM mantiene un buen rendimiento en la mayoría de las configuraciones, aunque el error tiende a aumentar ligeramente al incrementar la cantidad de datos de entrenamiento, especialmente al llegar al 90%. Esto podría ser indicativo de un cierto grado de degradación del modelo a medida que se enfrenta a un mayor volumen de datos.

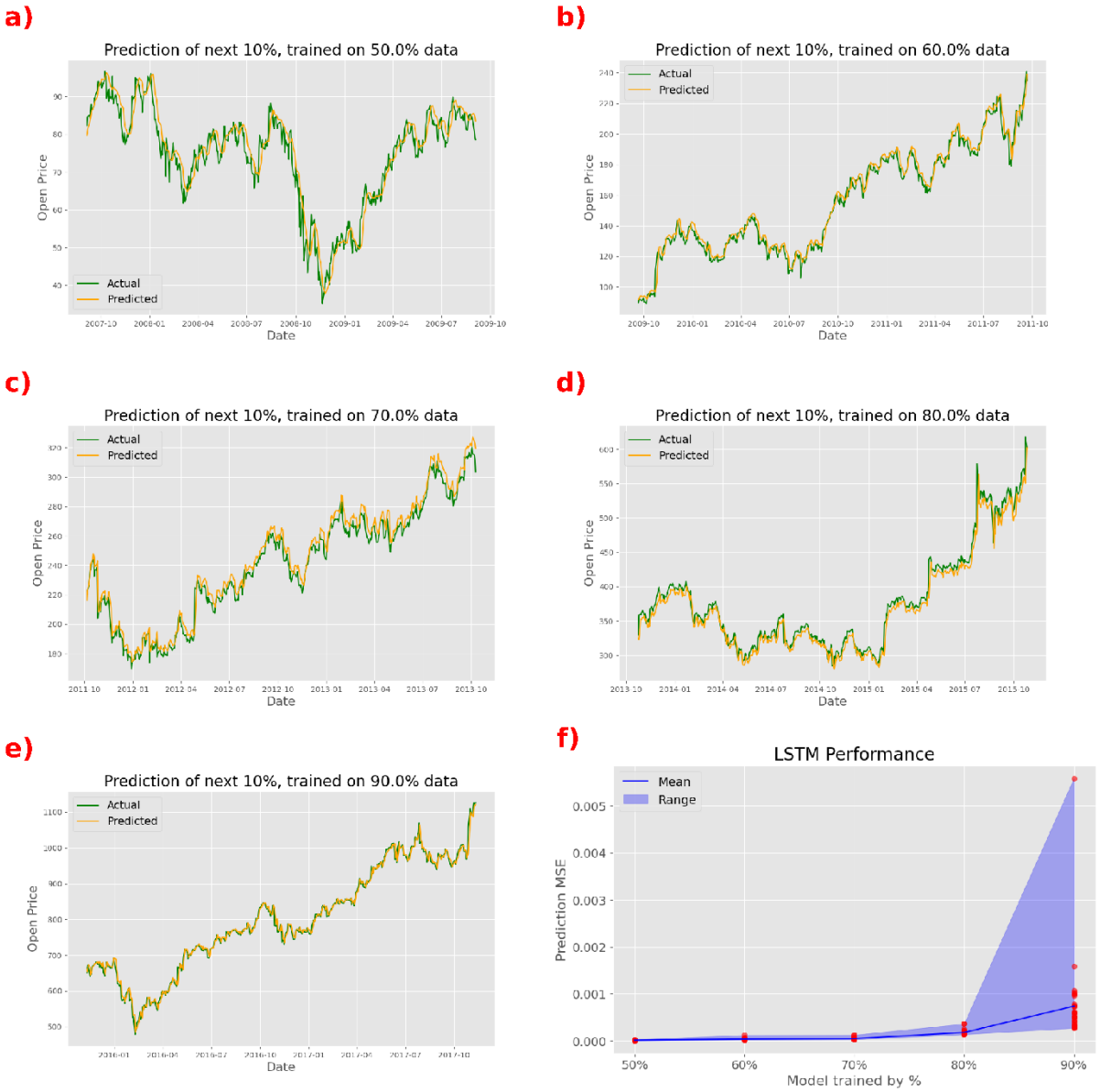


Fig. 17 a), b), c), d) y e) corresponden a gráficos de las predicciones junto a los valores reales de una de las 32 iteraciones durante el experimento 1 con el modelo LSTM. El gráfico f) muestra el desempeño del modelo en cuanto al MSE obtenido en el total de las predicciones.

Prueba 1 Transformer: En la Figura 20 se muestran los resultados de las predicciones realizadas por el modelo Transformer entrenado con diferentes porcentajes del conjunto de datos de acciones de Nvidia. La gráfica f) muestra el rendimiento global del modelo Transformer, mostrando tanto el MSE promedio como el rango de variación del error a medida que se incrementa el porcentaje de datos de entrenamiento. Los resultados muestran que, aunque el Transformer mantiene un rendimiento aceptable con porcentajes menores de entrenamiento, el error cuadrático medio aumenta significativamente cuando se entrena con el 90% de los datos al igual que en los modelos anteriores.

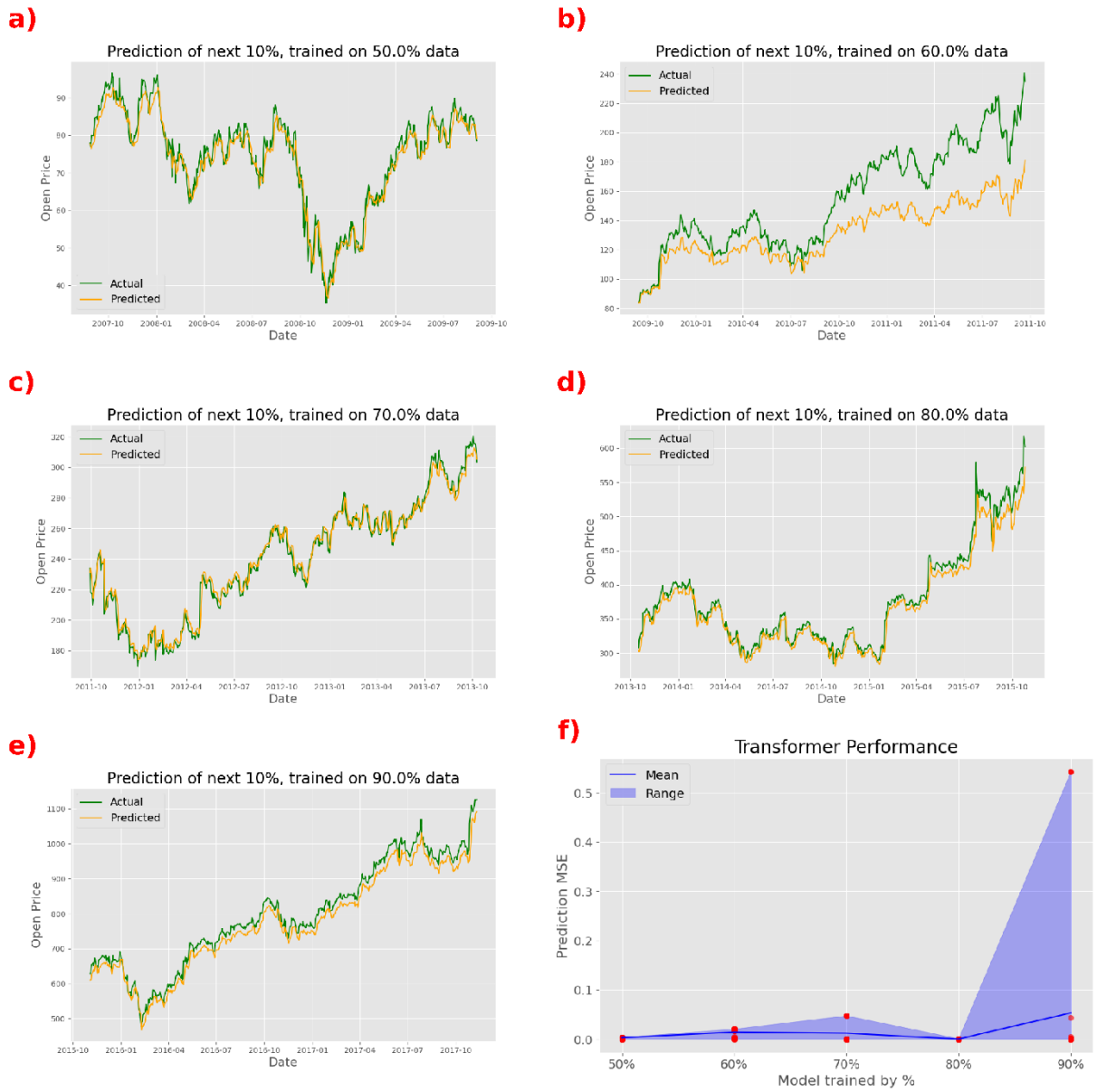


Fig. 18a), b), c), d) y e) corresponden a gráficos de las predicciones junto a los valores reales de una de las 32 iteraciones durante el experimento 1 con el modelo Transformer. El gráfico f) muestra el desempeño del modelo en cuanto al MSE obtenido en el total de las predicciones.

5.3 Resultados Prueba N°2

El siguiente gráfico muestra el MAE calculado en cada predicción durante la prueba N°2. En total son 70 solicitudes de predicción realizadas en la simulación de ambiente productivo. Se aprecia un aumento del error conforme aumenta la variable temporal asociada a los registros. Recalcando la importancia de los factores externos en el comportamiento de los datos. Cuya influencia afecta de manera directa en la calidad predictiva de los modelos.

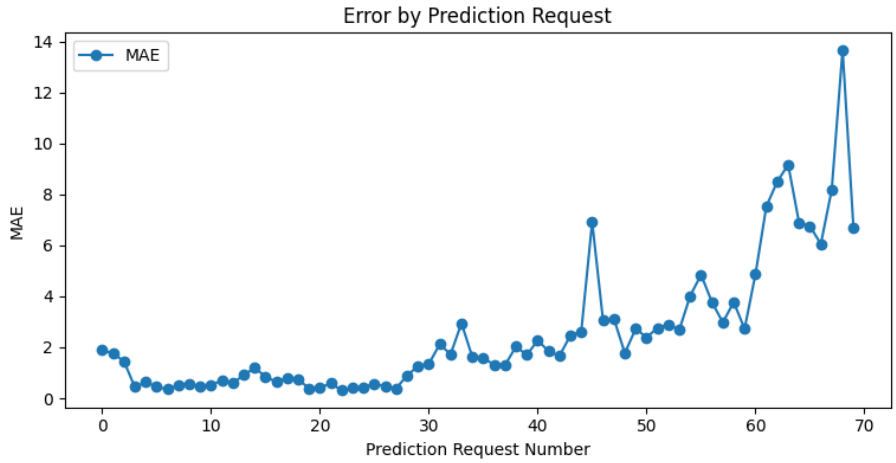


Fig. 19 Gráfico del MAE obtenido en cada una de las 70 predicciones realizadas al modelo LSTM en un ambiente productivo simulado.

Si la Figura 19 se compara a una gráfica de los valores del Stock de Amazon (Figura 20), se evidencia que las variaciones coinciden con el punto de inflexión donde aumenta el error en el experimento dos. El error tiende a aumentar al mismo tiempo que las acciones de Amazon aumentan, demostrando una relación proporcional donde el modelo pierde la capacidad de generalizar los drásticos cambios en el comportamiento de los valores del Stock.

Fig. 20 Valores de apertura del Stock de Amazon desde 1998 al 2017.

5.4 Ventanas de la aplicación.

A continuación, se presentan las distintas vistas de la aplicación web desarrollada. La figura 23 corresponde a la página principal para realizar las solicitudes de predicción, la figura 24 muestra la página de monitoreo desplegando los 6 tableros de Grafana y la figura 25 la página de reentrenamiento. Las capturas de las demás opciones de la aplicación se pueden encontrar en la sección de anexos de este trabajo (Anexos 7, 8, 9 y 10).

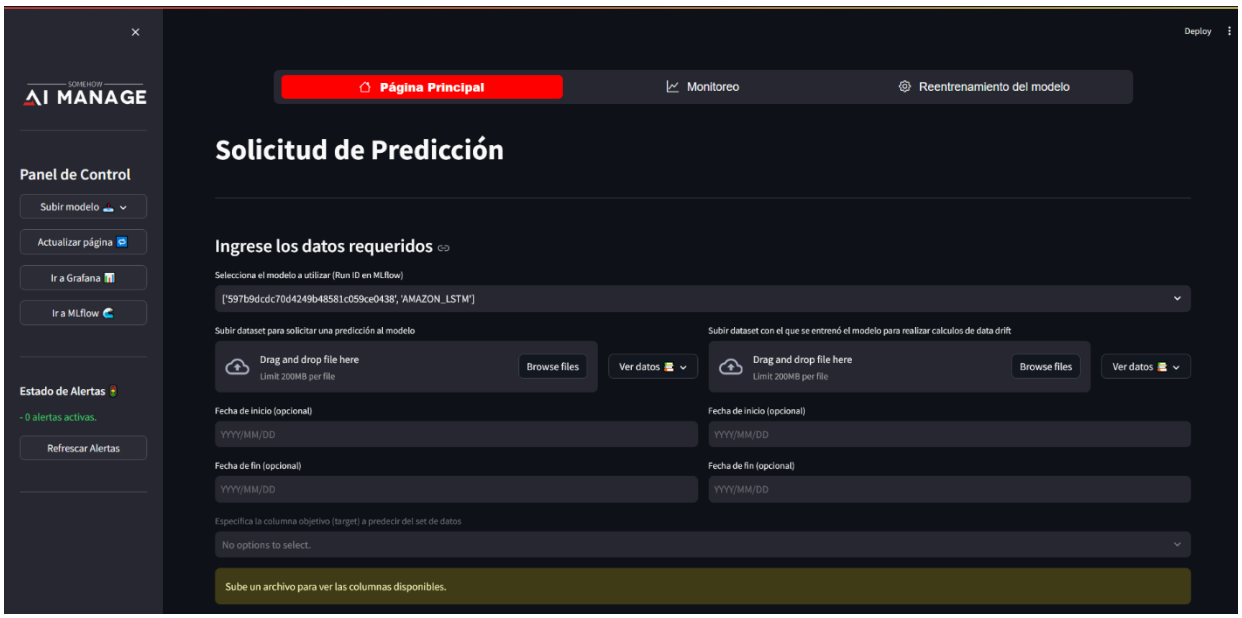


Fig. 22 Página de Solicitud de Predicción de la aplicación.

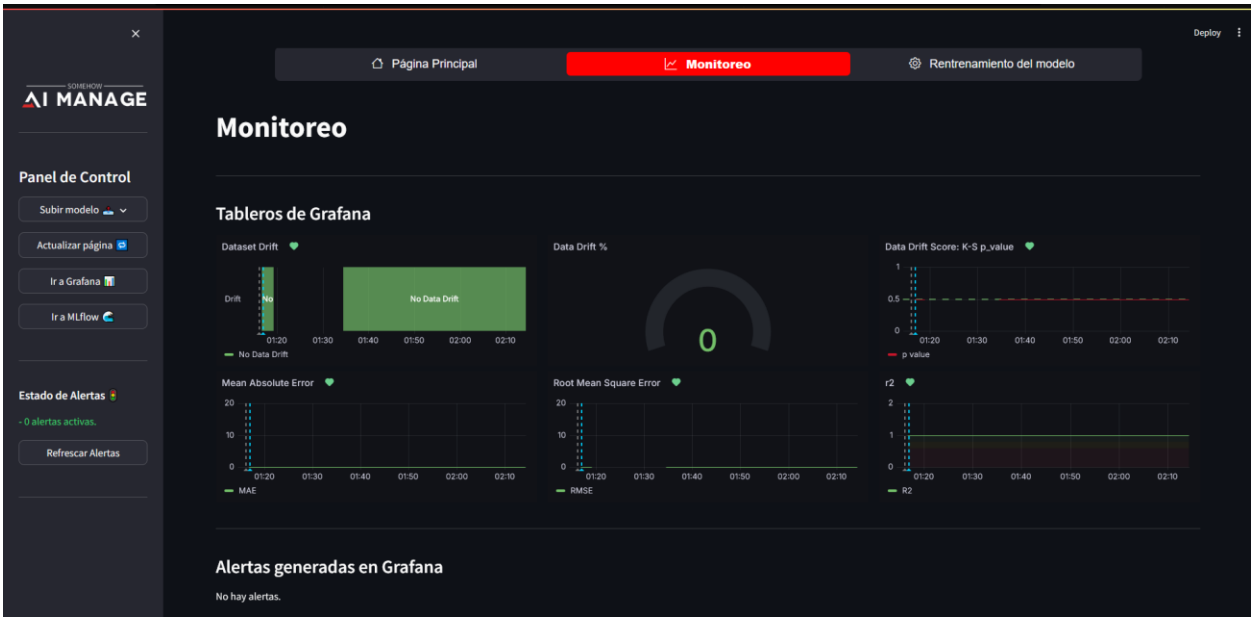


Fig. 21 Página de Monitore de la aplicación.

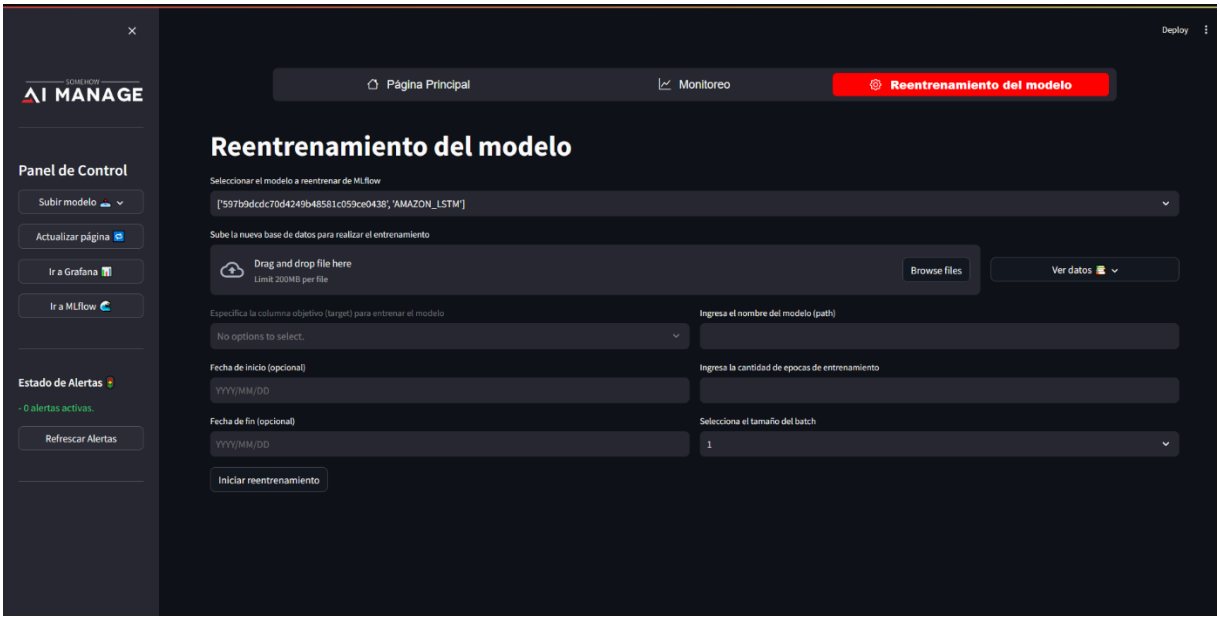


Fig. 23 Página de Reentrenamiento de la aplicación.

5.5 Resultados de la evaluación SUS

Se presentan las siguientes tablas con los puntos otorgados en la evaluación SUS por cada uno de los 6 encuestados incluyendo la nota final de cada participante y otra con la respectiva sumatoria de los puntajes además del promedio de las notas finales. Cabe recalcar que las preguntas pares tienen una connotación negativa sobre la usabilidad de las aplicaciones evaluadas y las preguntas no pares una connotación positiva, en base a esto se remarcan los puntajes más altos obtenidos en cada una de las dos clases de afirmaciones (A) para su posterior análisis en la sección de discusiones.

Participante	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	Nota Final
P. °1	5	1	4	4	4	1	4	1	3	4	72.5
P. °2	3	2	3	4	5	1	4	2	4	2	70
P. °3	3	2	3	4	4	2	5	1	5	2	72.5
P. °4	2	2	4	4	5	1	4	3	3	4	60
P. °5	4	1	5	4	4	2	2	1	3	4	65
P. °6	5	2	3	4	5	2	3	2	4	2	70

Tabla 2 Puntaje por afirmación y nota final por cada participante.

La evaluación de la usabilidad de la aplicación mediante la encuesta SUS arrojó un puntaje de 68.3. Este resultado sitúa la aplicación ligeramente por encima del promedio, ya que un puntaje de 68 es considerado el promedio basado en estudios previos [24]. Según la escala de interpretación de la encuesta SUS, que va desde la calificación “A” a la “F”, un puntaje de 68.3 corresponde aproximadamente a una calificación de “C”, indicando que la usabilidad de la aplicación es aceptable, aunque hay margen para mejoras. Dado que los puntajes superiores a 80.3 (calificación “A”) se asocian con una mayor probabilidad de que los usuarios recomienden el producto, la meta futura sería continuar optimizando la usabilidad para alcanzar esta categoría superior.

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	Promedio
Suma Total	22	10	22	24	27	9	22	10	22	18	68.3

Tabla 3 Suma total por afirmación y nota final promedio. La nota roja corresponde a la afirmación negativa con mayor puntaje y la verde a la afirmativa de mayor puntaje.

Se recalcan los dos ítems con mayor puntaje: uno negativo y uno positivo. La afirmación negativa con mayor puntaje fue "Creo que necesitaría el apoyo de una persona técnica para poder usar este sistema" con 24 de 30 puntos (en rojo). La afirmación positiva con mejor puntaje fue "Encontré que las diversas funciones de este sistema estaban bien integradas", con 27 de 30 puntos (en verde).

6. Discusión

La presente investigación ha abordado el problema de la degradación de modelos de Machine Learning en ambientes productivos, con un enfoque particular en los conceptos de *data drift* y *model drift*. A través de la implementación de un sistema de monitoreo y una aplicación web para la detección y mitigación de la degradación, se han obtenido resultados significativos que se abordarán en mayor profundidad a continuación.

Los primeros resultados obtenidos de la puesta en marcha de los tres modelos (RNN, LSTM y Transformer) muestran un comportamiento coherente con las características esperadas de cada uno de ellos. Se observa que la RNN presenta un margen de error elevado en gran parte de los entrenamientos. Esto se debe a la dificultad que posee este tipo de red neuronal para trabajar con series temporales extensas. En cuanto a la LSTM, esta red mitiga los problemas detectados en el modelo anterior gracias a sus celdas de memoria. Por otro lado, el Transformer presenta mejores resultados en los primeros entrenamientos, pero al igual que los demás modelos de estudio presenta errores elevados en la última iteración.

Los resultados de la prueba N°1 también son coherentes con los esperado ya que la razón principal de realizar este tipo de experimento es detectar la degradación derivada del *ovefitting* o sobre ajuste del modelo al incrementar de forma constantes los datos con los que se entrena, dificultando la capacidad de generalizar bien con relación a los nuevos datos que se proporcionan.

Importante es señalar que una primera iteración de este experimento consideraba incrementar la data de entrenamiento y luego evaluar la predicción con una fracción fija de los datos, esta fracción correspondía a los últimos 3 meses del conjunto de datos. Este primer enfoque no arrojaba los resultados esperados ya que ocurría una disminución del error a medida que se aumentaban los datos de entrenamiento. La razón de este comportamiento se asocia a la variable temporal puesto que cada vez se entrenaba con valores que se acercaban a un comportamiento similar al de los datos de predicción, es decir, la tendencia del entrenamiento se hacía cada vez más parecida a los datos de prueba, este experimento luego derivó en la prueba N°1.

En cuanto a la segunda prueba se aprecia un claro aumento del error en la predicción del modelo alrededor de la solicitud número 30, demostrando la importancia que toman los cambios externos que interactúan con el valor de las acciones de la bolsa, ya que alrededor del año 2008 se nota un incremento significativo del valor del Stock de Amazon debido a la masificación de las compras en línea. Este auge genera un patrón que el modelo no está preparado para identificar generando un incremento muy significativo del error.

La aplicación confeccionada funciona dentro de los parámetros que se habían establecido, es capaz de solicitar las predicciones, reentrenar y subir los modelos manteniendo

la simulación de ambiente productivo. Un detalle por considerar es que Streamlit al ser una aplicación que funciona mediante la re-ejecución de un código en concreto hay ciertas funcionalidades que se ven limitadas como lo es el incorporar una metodo que se ejecute cada cierto tiempo para actualizar las alertas en vez de tener que hacerlo de forma manual. Por otro lado, en cuanto a la comunicación con la API de Grafana para obtener las alertas se tiene que Grafana es una aplicación que presenta un alto grado de seguridad, por lo tanto, se vuelve obligatorio generar un Token para acceder a su API, por lo que considerar un método simple para que el usuario pueda obtener y proveer este token a la aplicación debe ser un factor importante por considerar al momento de generar el contenedor final.

El despliegue de la aplicación mediante una imagen en Docker presentó una serie de desafíos en relación con la forma en que los distintos contenedores utilizados se comunican entre ellos a nivel de Docker. En especial la herramienta MLflow, a la cual no se podía acceder de forma directa para administrar los modelos que se subían en la aplicación, para solucionar esto se debe de acceder con el nombre que se le asigna al respectivo contenedor en el código, por ejemplo, la dirección de acceso pasaba de ser *127.0.0.1:5000* a *mlflow:5000*. También se debe mencionar una problemática recurrente con MLflow, este no era capaz de inicializarse con modelos de demostración a menos que se predefinieran archivos claves en la estructura y dependencia de sus directorios. Utilizar Docker junto a GitHub es una excelente forma de compartir y asegurarse de que la aplicación funcione de manera correcta en cualquier dispositivo, facilitando también a futuros usuarios editar y modificar el código fuente de la aplicación.

La parte final del trabajo se centró en la usabilidad de la aplicación. De los resultados se puede inferir que la aplicación requiere conocimientos técnicos en inteligencia artificial y Machine Learning, lo cual es esperado ya que está dirigida a profesionales o personas con un mayor entendimiento en el área.

La metodología abordada en la confección de este trabajo jugó un papel fundamental en el correcto y oportuno desarrollo de las tareas definidas en una amplia carta Gantt, cumpliendo con cada uno de los objetivos de este proyecto. La aplicación desarrollada siendo el objetivo final, una herramienta capaz de detectar y dar al usuario la opción de mitigar la degradación en los modelos de Machine Learning. Si bien por razones de tiempo y escalabilidad de este proyecto la aplicación solo es trabajada en una simulación de un ambiente productivo los resultados obtenidos demuestran la importancia del monitoreo continuo y del control que se debe tener sobre los algoritmos desplegados en la industria.

7. Conclusión

En este trabajo se ha abordado el problema de la degradación de modelos de Machine Learning en ambientes productivos, un desafío significativo en el campo de la inteligencia artificial y el Machine Learning. La importancia de este proyecto radica en la creciente utilización y dependencia de los modelos de Machine Learning en diversas industrias, como las finanzas, la salud y el transporte, donde una inferencia precisa y confiable es crucial para la toma de decisiones, disminuyendo así riesgos y costos.

Se establecieron como objetivos principales la identificación de los tipos de degradación que pueden afectar a los modelos de Machine Learning, específicamente el *data drift* y *model drift*, así como la implementación de herramientas para su detección y mitigación. Estos objetivos se lograron a través de una exhaustiva revisión teórica, el desarrollo de un sistema de monitoreo con Prometheus y Grafana, y la creación de una aplicación web capaz de detectar y mitigar la degradación mediante reentrenamiento.

Entre las fortalezas de esta investigación se destaca la implementación práctica de una solución de monitoreo y reentrenamiento en un entorno simulado, lo cual demostró la efectividad de estas estrategias para mantener la precisión de los modelos a lo largo del tiempo. Además, la encapsulación de la aplicación en un contenedor Docker asegura su portabilidad y facilita su despliegue en diferentes entornos. Sin embargo, un aspecto de las fortalezas también puede ser considerado una limitación, el hecho de que los experimentos se realizaron en un entorno controlado y simulado, lo que podría no reflejar completamente la complejidad de los entornos productivos reales.

En cuanto a los resultados obtenidos, en contraste con otros estudios se observó que las estrategias de monitoreo y reentrenamiento son ampliamente reconocidas como efectivas para mitigar la degradación de modelos. La aplicación confeccionada aporta una solución práctica que puede ser adaptada y utilizada en diversos ámbitos reales. Los resultados son significativos para la comunidad de Machine Learning, ya que proporcionan una metodología concreta para abordar la degradación de modelos, un problema común pero crítico. La documentación y los códigos generados se han puesto a disposición en un repositorio de GitHub, facilitando su uso y adaptación en futuros proyectos. Este trabajo sienta las bases para futuras investigaciones en la mitigación de la degradación de modelos y destaca la importancia de un enfoque proactivo en el mantenimiento de sistemas de Machine Learning en producción.

8. Referencias

[1] G. Prasath, “Computational Intelligence”, Medium. Accedido: 6 de mayo de 2024. [En línea]. Disponible en: https://medium.com/@cs.gokulprasath98/computational-intelligence-d14e7e50c955

[2] M. Siwach y S. Mann, “A Compendium of Various Applications of Machine Learning”, *Int. J. Res. Eng. Technol.*, vol. 9, pp. 1141–1144, jul. 2022.

[3] A. Géron, “Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow”, 2019.

[4] “¿Qué es el machine learning? - Explicación sobre el machine learning empresarial - AWS”. Accedido: 5 de agosto de 2024. [En línea]. Disponible en: https://aws.amazon.com/es/what-is/machine-learning/

[5] M. Treveil *et al.*, *Introducing MLOps*. “O’Reilly Media, Inc.,” 2020.

[6] D. Sculley *et al.*, “Hidden Technical Debt in Machine Learning Systems”, 2015.

[7] “What is data drift in ML, and how to detect and handle it”. Accedido: 3 de mayo de 2024. [En línea]. Disponible en: https://www.evidentlyai.com/ml-in-production/data-drift

[8] “What is concept drift in ML, and how to detect and address it”. Accedido: 5 de agosto de 2024. [En línea]. Disponible en: https://www.evidentlyai.com/ml-in-production/concept-drift

[9] M. W. Ahmed, “Understanding Mean Absolute Error (MAE) in Regression: A Practical Guide”, Medium. Accedido: 5 de agosto de 2024. [En línea]. Disponible en: https://medium.com/@m.waqar.ahmed/understanding-mean-absolute-error-mae-in-regression-a-practical-guide-26e80ebb97df

[10] “Root Mean Square Error (RMSE)”, C3 AI. Accedido: 5 de agosto de 2024. [En línea]. Disponible en: https://c3.ai/glossary/data-science/root-mean-square-error-rmse/

[11] E. D, “Looking at R-Squared”, Medium. Accedido: 5 de agosto de 2024. [En línea]. Disponible en: https://medium.com/@erika.dauria/looking-at-r-squared-721252709098

[12] “Understanding Data Drift with the Kolmogorov-Smirnov Test”. Accedido: 6 de mayo de 2024. [En línea]. Disponible en: https://www.linkedin.com/pulse/understanding-data-drift-kolmogorov-smirnov-test-andrew-f-ij2ge

[13] “Wasserstein Distance to Detect Data Drift in ML Models”. Accedido: 6 de mayo de 2024. [En línea]. Disponible en: https://www.linkedin.com/pulse/wasserstein-distance-detect-data-drift-ml-models-amit-tiwari-eb2ke

[14] “The Jensen-Shannon Divergence: A Measure of Distance Between Probability Distributions”, Medium. Accedido: 5 de agosto de 2024. [En línea]. Disponible en: https://itsudit.medium.com/the-jensen-shannon-divergence-a-measure-of-distance-between-probability-distributions-23b2b1146550

[15] A. Mallick, K. Hsieh, B. Arzani, y G. Joshi, “Matchmaker: Data Drift Mitigation in Machine Learning for Large-Scale Systems”.

[16] D. Vela, A. Sharp, R. Zhang, T. Nguyen, A. Hoang, y O. S. Pianykh, “Temporal quality degradation in AI models”, *Sci. Rep.*, vol. 12, nº 1, Art. nº 1, jul. 2022, doi: 10.1038/s41598-022-15245-z.

[17] D. Brzezinski y J. Stefanowski, “Reacting to Different Types of Concept Drift: The Accuracy Updated Ensemble Algorithm”, *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, nº 1, pp. 81–94, ene. 2014, doi: 10.1109/TNNLS.2013.2251352.

[18] A. Tahmasbi, E. Jothimurugesan, S. Tirthapura, y P. B. Gibbons, “DriftSurf: A Risk-competitive Learning Algorithm under Concept Drift”, 2 de agosto de 2020, *arXiv*: arXiv:2003.06508. doi: 10.48550/arXiv.2003.06508.

[19] Z. Young y R. Steele, “Empirical evaluation of performance degradation of machine learning-based predictive models – A case study in healthcare information systems”, *Int. J. Inf. Manag. Data Insights*, vol. 2, nº 1, p. 100070, abr. 2022, doi: 10.1016/j.jjimei.2022.100070.

[20] L. Poenaru-Olaru, J. Sallou, L. Cruz, J. S. Rellermeyer, y A. Van Deursen, “Retrain AI Systems Responsibly! Use Sustainable Concept Drift Adaptation Techniques”, en *2023 IEEE/ACM 7th International Workshop on Green And Sustainable Software (GREENS)*,

Melbourne, Australia: IEEE, may 2023, pp. 17–18. doi: 10.1109/GREENS59328.2023.00009.

[21] “Historical Weather Data Download”, meteoblue. Accedido: 3 de junio de 2024. [En línea]. Disponible en: https://www.meteoblue.com/en/weather/archive/export

[22] “Huge Stock Market Dataset”. Accedido: 3 de junio de 2024. [En línea]. Disponible en: https://www.kaggle.com/datasets/borismarjanovic/price-volume-data-for-all-us-stocks-etfs

[23] J. Brooke, “SUS -- a quick and dirty usability scale”, 1996, pp. 189–194.

[24] J. Sauro, “Measuring Usability with the System Usability Scale (SUS) – MeasuringU”. Accedido: 7 de agosto de 2024. [En línea]. Disponible en: https://measuringu.com/sus/

9. Anexos

Anexo 1. Arquitectura del modelo Transformer, obtenida mediante el comando Summary().

Layer (type)	Output Shape	Param #	Connected to
input_layer_2 (InputLayer)	(None, 5, 1)	0	-
layer_normalization (LayerNormalizatio…	(None, 5, 1)	2	input_layer_2[0]…
multi_head_attenti… (MultiHeadAttentio…	(None, 5, 1)	3,585	layer_normalizat… layer_normalizat…
dropout_2 (Dropout)	(None, 5, 1)	0	multi_head_atten…
add (Add)	(None, 5, 1)	0	dropout_2[0][0], input_layer_2[0]…
layer_normalizatio… (LayerNormalizatio…	(None, 5, 1)	2	add[0][0]
conv1d (Conv1D)	(None, 5, 2)	4	layer_normalizat…
dropout_3 (Dropout)	(None, 5, 2)	0	conv1d[0][0]
conv1d_1 (Conv1D)	(None, 5, 1)	3	dropout_3[0][0]
add_1 (Add)	(None, 5, 1)	0	conv1d_1[0][0], add[0][0]
layer_normalizatio… (LayerNormalizatio…	(None, 5, 1)	2	add_1[0][0]
multi_head_attenti… (MultiHeadAttentio…	(None, 5, 1)	3,585	layer_normalizat… layer_normalizat…
dropout_5 (Dropout)	(None, 5, 1)	0	multi_head_atten…
add_2 (Add)	(None, 5, 1)	0	dropout_5[0][0], add_1[0][0]
layer_normalizatio… (LayerNormalizatio…	(None, 5, 1)	2	add_2[0][0]

conv1d_2 (Conv1D)	(None, 5, 2)	4	layer_normalizat…
dropout_6 (Dropout)	(None, 5, 2)	0	conv1d_2[0][0]
conv1d_3 (Conv1D)	(None, 5, 1)	3	dropout_6[0][0]
add_3 (Add)	(None, 5, 1)	0	conv1d_3[0][0], add_2[0][0]
layer_normalizatio… (LayerNormalizatio…	(None, 5, 1)	2	add_3[0][0]
multi_head_attenti… (MultiHeadAttentio…	(None, 5, 1)	3,585	layer_normalizat… layer_normalizat…
dropout_8 (Dropout)	(None, 5, 1)	0	multi_head_atten…
add_4 (Add)	(None, 5, 1)	0	dropout_8[0][0], add_3[0][0]
layer_normalizatio… (LayerNormalizatio…	(None, 5, 1)	2	add_4[0][0]
conv1d_4 (Conv1D)	(None, 5, 2)	4	layer_normalizat…
dropout_9 (Dropout)	(None, 5, 2)	0	conv1d_4[0][0]
conv1d_5 (Conv1D)	(None, 5, 1)	3	dropout_9[0][0]
add_5 (Add)	(None, 5, 1)	0	conv1d_5[0][0], add_4[0][0]
layer_normalizatio… (LayerNormalizatio…	(None, 5, 1)	2	add_5[0][0]
multi_head_attenti… (MultiHeadAttentio…	(None, 5, 1)	3,585	layer_normalizat… layer_normalizat…
dropout_11 (Dropout)	(None, 5, 1)	0	multi_head_atten…
add_6 (Add)	(None, 5, 1)	0	dropout_11[0][0], add_5[0][0]
layer_normalizatio… (LayerNormalizatio…	(None, 5, 1)	2	add_6[0][0]

conv1d_6 (Conv1D)	(None, 5, 2)	4	layer_normalizat…
dropout_12 (Dropout)	(None, 5, 2)	0	conv1d_6[0][0]
conv1d_7 (Conv1D)	(None, 5, 1)	3	dropout_12[0][0]
add_7 (Add)	(None, 5, 1)	0	conv1d_7[0][0], add_6[0][0]
global_average_poo… (GlobalAveragePool…	(None, 5)	0	add_7[0][0]
dense_3 (Dense)	(None, 256)	1,536	global_average_p…
dropout_13 (Dropout)	(None, 256)	0	dense_3[0][0]
dense_4 (Dense)	(None, 1)	257	dropout_13[0][0]

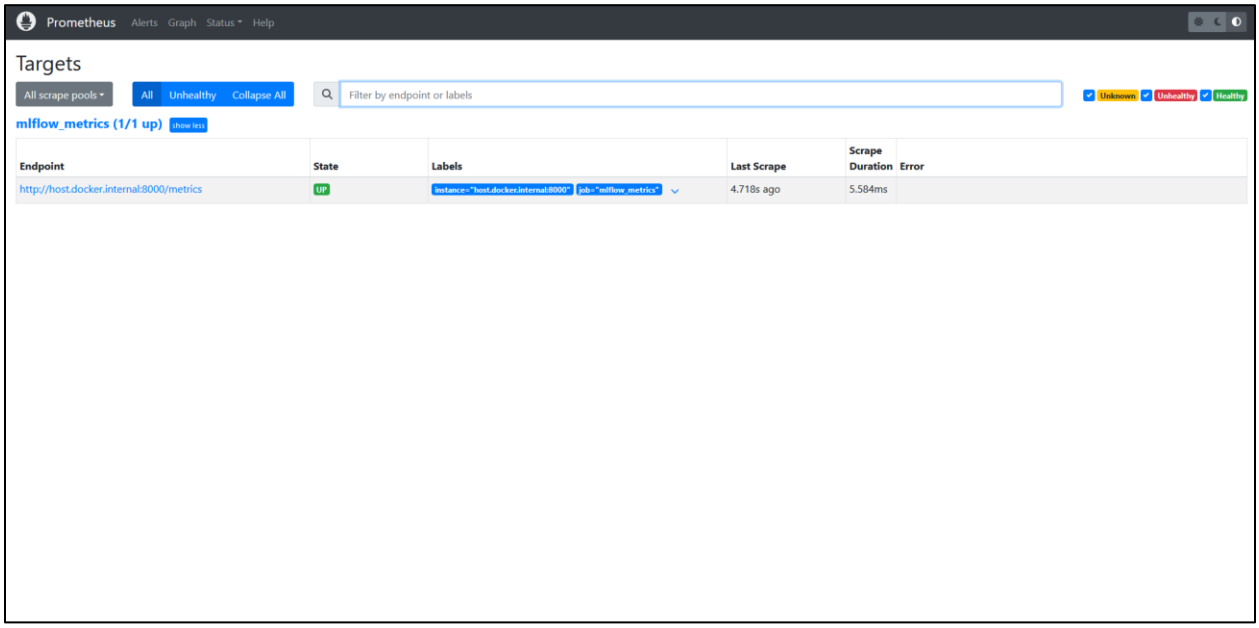
Total params: 48,533 (189.59 KB)

Trainable params: 16,177 (63.19 KB)

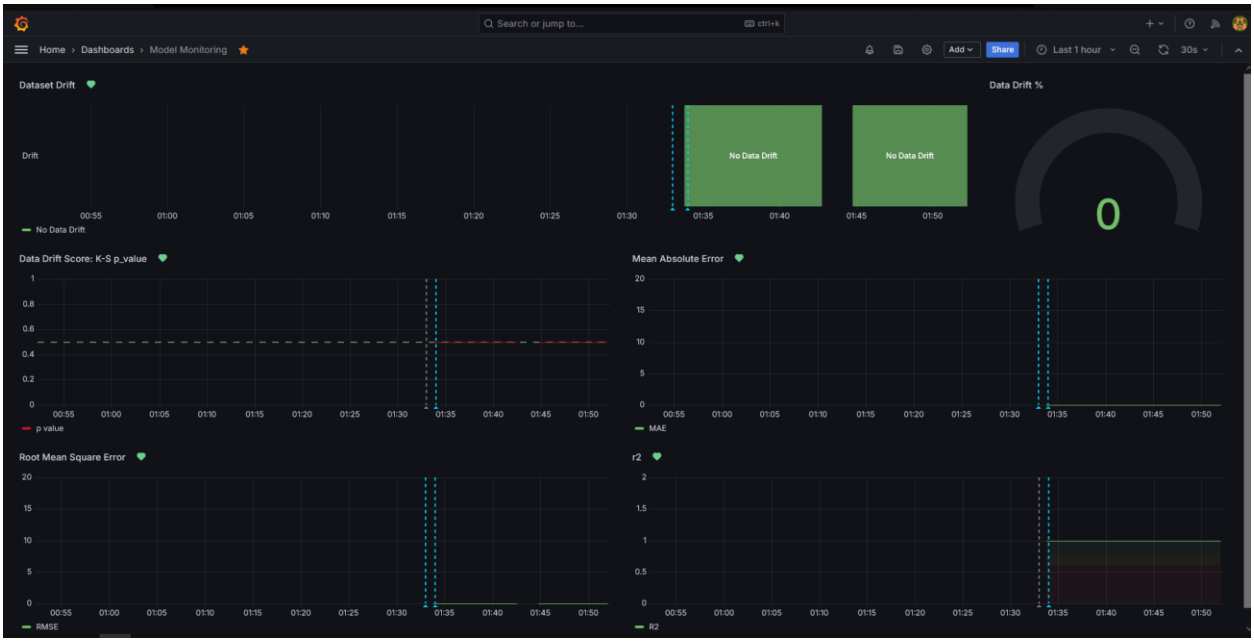
Non-trainable params: 0 (0.00 B)

Optimizer params: 32,356 (126.39 KB)

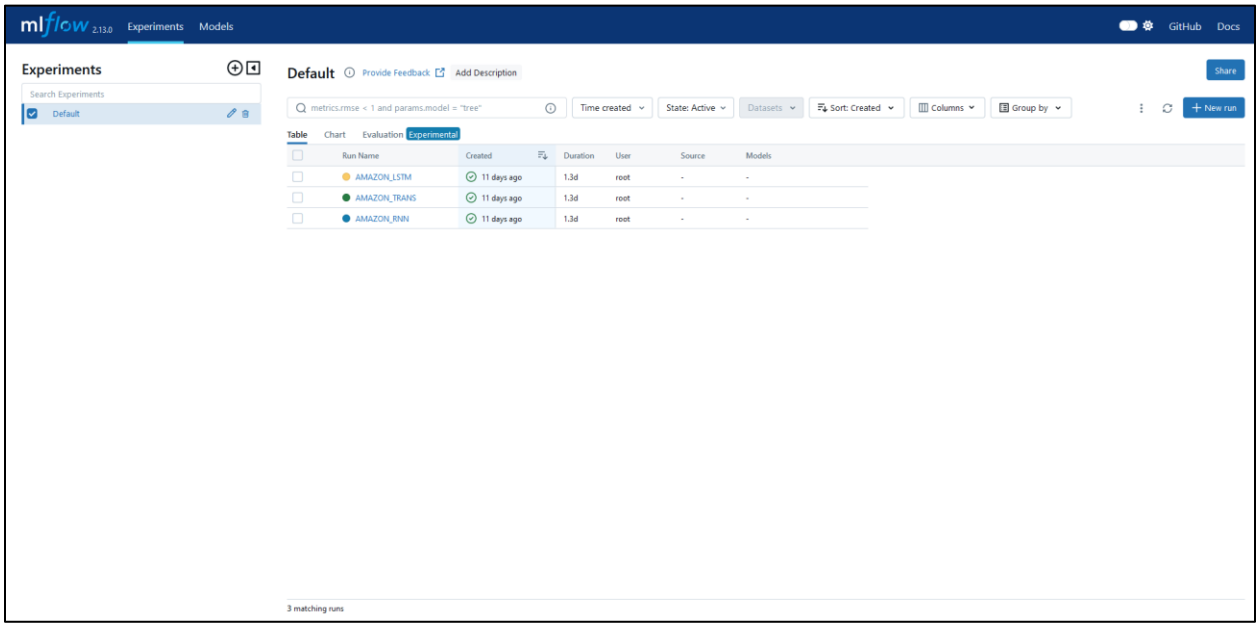
Anexo 2. Captura de la herramienta Prometheus en funcionamiento.



Anexo 3. Captura de la herramienta Grafana en funcionamiento. En el tablero creado "Model Monitoring"



Anexo 4. Captura del servicio MLflow con los 3 modelos de demostración en funcionamiento.

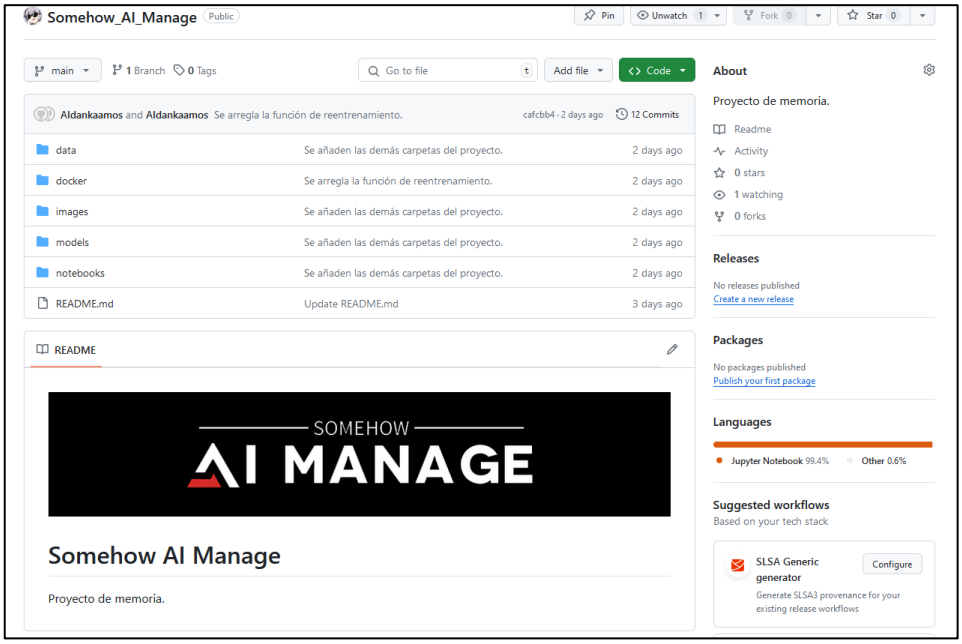


Anexo 5. Archivo "*docker-compose.yml*" de la aplicación.

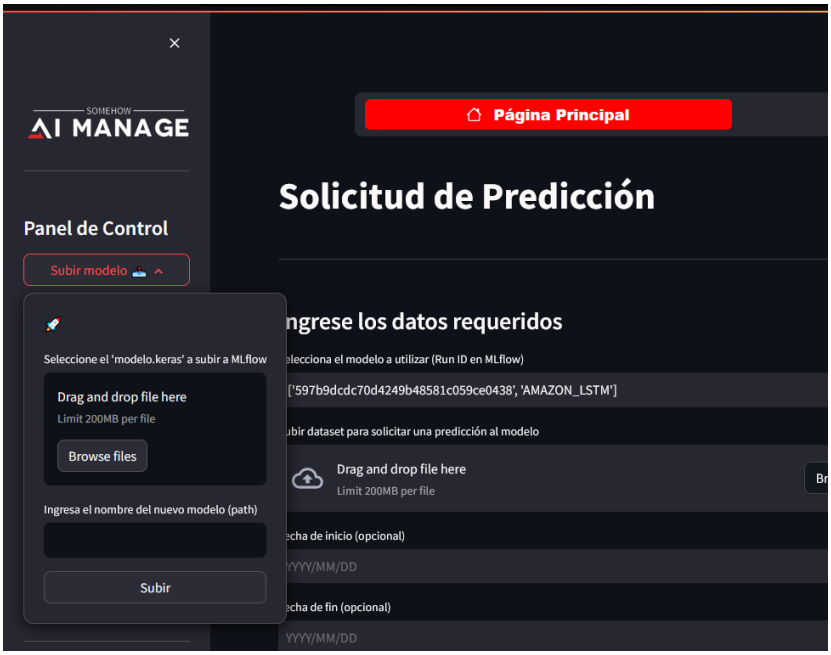
```yaml
volumes:
  prometheus-data:
    driver: local
  grafana-data:
    driver: local
  mlruns:
    driver: local
services:
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    restart: unless-stopped
    volumes:
      - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
      - ./data/prometheus-data:/prometheus
    networks:
      - prometheus-network
    ports:
      - "9090:9090"
    user: "0"
  grafana:
    image: grafana/grafana:latest
    container_name: grafana
    restart: unless-stopped
    volumes:
      - ./data/grafana-data:/var/lib/grafana
    networks:
      - prometheus-network
    ports:
      - "3000:3000"
    environment:
      - "GF_SECURITY_ALLOW_EMBEDDING=true"
      - "GF_SECURITY_COOKIE_SECURE=true"
      - "GF_SECURITY_COOKIE_SAMESITE=none"
    user: "0"
  app:
    build: .
    image: somehowaimanage:latest
    container_name: app
    ports:
      - "8501:8501"
      - "8000:8000"
      - "5001:5001"
    depends_on:
```

```
      - grafana
      - prometheus
    volumes:
      - ./app:/usr/src/app
    networks:
      - prometheus-network
    user: "0"

  mlflow:
    image: somehowaimanage:latest
    container_name: mlflow
    command: ["mlflow", "ui", "--host", "0.0.0.0", "--port", "5000"]
    networks:
      - prometheus-network
    ports:
      - "5000:5000"
    volumes:
      - ./app/mlruns:/mlruns
    user: "0"
networks:
  prometheus-network:
    driver: bridge
```

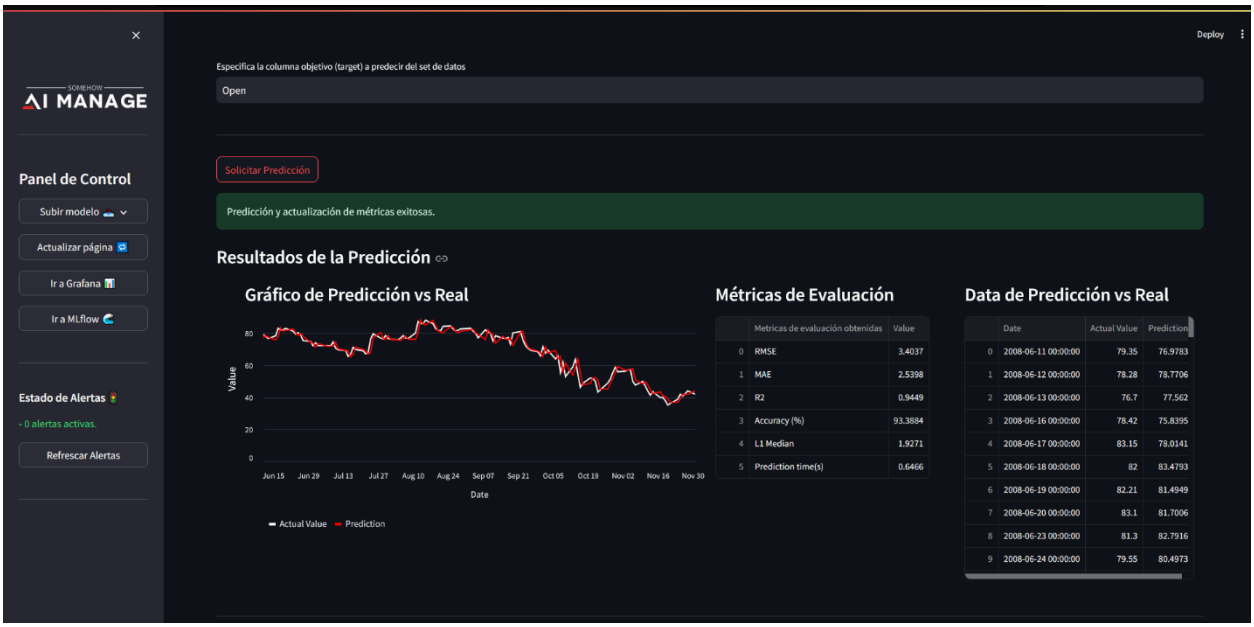
Anexo 6. Captura y link del repositorio final de la aplicación.
Link: https://github.com/Aldankaamos/Somehow_AI_Manage



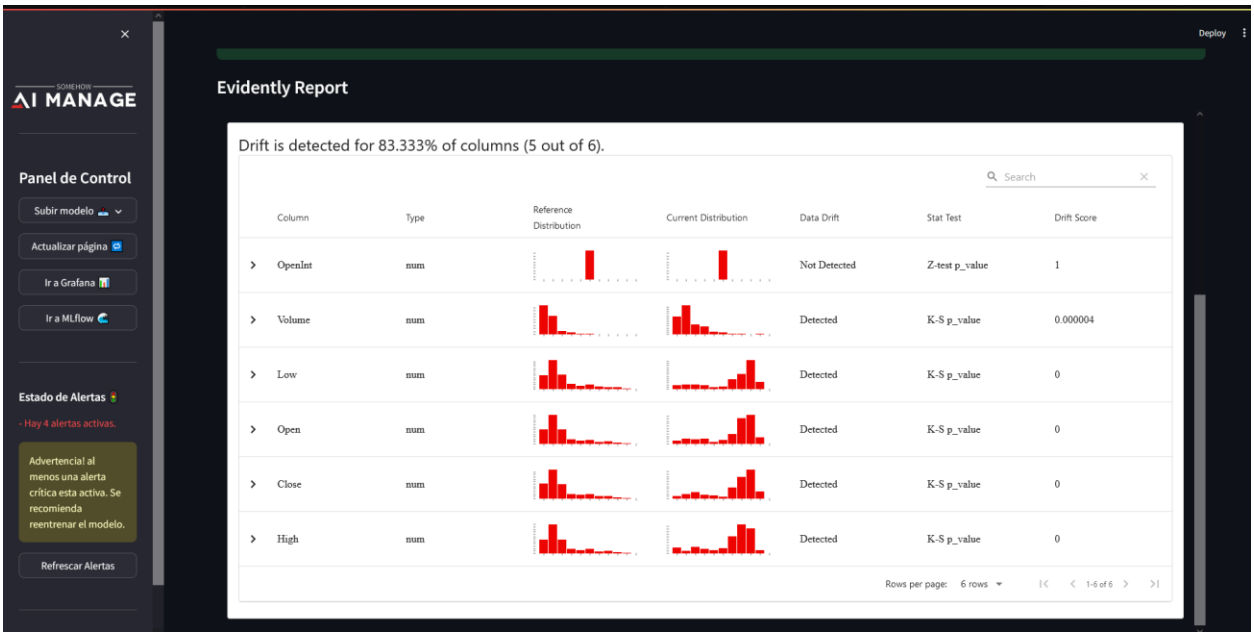
Anexo 7. Captura de la función “Subir modelo” de la aplicación.



Anexo 8. Captura de los resultados de una solicitud de predicción.



Anexo 9. Captura de una solicitud de informe de “data drift” utilizando Evidently AI.



Anexo 10. Captura del despliegue de las alertas de Grafana.