Metodología MLOps para la entrega continúa de un modelo de Machine Learning para el reconocimiento y control de las plagas *Stenoma catenifer* y *heilipus lauri* en el cultivo de aguacate Hass

Juan Felipe Rodriguez Torres

Proyecto presentado como requisito para optar al título de:
Magister en Ingeniería de Software

Director:
Ph.D. David Arango Londoño

Pontificia Universidad Javeriana Cali
Facultad de Ingeniería
Departamento de Electrónica y Ciencias de la Computación
Cali, Colombia
30 de julio de 2024

Maestría en Ingeniería
Facultad de Ingeniería

Pontificia Universidad JAVERIANA Cali

Acta de Correcciones al Documento de Trabajo de Grado

Santiago de Cali, Julio 30 de 2024

Autor: Juan Felipe Rodriguez Torres

Título del Trabajo de Grado: Metodología MLOps para la entrega continúa de un modelo de Machine Learning para el reconocimiento y control de las plagas Stenoma catenifer y heilipus lauri en el cultivo de aguacate Hass

Director:

Como indica el artículo 2.13 de las Directrices para Trabajo de Grado de Maestría, he verificado que el estudiante indicado arriba ha implementado todas las correcciones que los Jurados del Proyecto de Trabajo de Grado definieron que se efectuaran, como consta en el Acta de Evaluación correspondiente.

David Arango Londoño

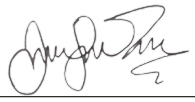
Ph.D. David Arango Londoño

Santiago de Cali, 30 de julio de 2024

Ingeniero:
Juan Carlos Martínez Arias
Director Posgrados de Ingeniería
Facultad de Ingeniería y Ciencias
Pontificia Universidad Javeriana - Cali

Con el fin de cumplir con los requisitos exigidos por la Universidad para llevar a cabo el Trabajo de Grado y posteriormente optar por el título de Magíster en Ingeniería, nos permitimos presentar a su consideración el proyecto de Trabajo de Grado denominado "*Metodología MLOps para la entrega continúa de un modelo de Machine Learning para el reconocimiento y control de las plagas Stenoma catenifer y heilipus lauri en el cultivo de aguacate Hass*", el cual será realizado por el estudiante *Juan Felipe Rodriguez Torres* con código *3070801526* perteneciente al énfasis en Ingeniería de Software, bajo la dirección del profesor *David Arango Londoño*.

El suscrito director del Trabajo de Grado autoriza para que se proceda a hacer la evaluación de este Proyecto ante el Tribunal que para el efecto se designe, toda vez que ha revisado cuidadosamente el documento y avala que ya se encuentra listo para ser presentado oficialmente.

Firma Estudiante 
Nombre Estudiante Juan Felipe Rodriguez Torres
C.C. 1130668332 *de* Cali

Firma Director 
Nombre Director David Arango Londoño
C.C. 1130586950 *de* Cali

Ficha Resumen
Trabajo de Grado de Maestría

Título: Metodología MLOps para la entrega continúa de un modelo de Machine Learning para el reconocimiento y control de las plagas *Stenoma catenifer* y *heilipus lauri* en el cultivo de aguacate Hass.

1. Tipo de proyecto: Aplicado
2. Área de trabajo: Ingeniería de Software
3. Estudiante: Juan Felipe Rodriguez
4. Correo electrónico: jfrodriguezt@javerianacali.edu.co
5. Dirección y teléfono: Carrera 112 48 - 92 Apto 701 Torre 5, 3058176473
6. Director: Ph. D. David Arango
7. Correo electrónico del director: david.arango@javerianacali.edu.co
8. Palabras clave(al menos 5): Big Data, Machine Learning, MLOps, Cultivo de aguacate, Plagas *Stenoma catenifer* y *heilipus lauri*.
9. Fecha de inicio: 15/Septiembre/2023
10. Resumen: La creciente aplicación de avances tecnológicos en diversos ámbitos de la vida ha llevado a la adopción de tecnologías innovadoras en la agricultura para mejorar la productividad y la eficiencia. Dentro de estas tecnologías, la metodología MLOps se destaca por su capacidad para mantener la operación de los modelos de aprendizaje automático y su despliegue, mientras se mejora y reentrenan los modelos, en donde se optimiza la toma de decisiones y el aumento de la precisión de los resultados.

La investigación se centra en el cultivo del aguacate Hass, un componente crucial para la economía y el desarrollo socioeconómico en países como México y Colombia. Sin embargo, este cultivo enfrenta desafíos significativos, como las plagas *Stenoma catenifer* y *heilipus lauri*. Para combatir este problema, se plantea una serie de objetivos que giran en torno a la implementación de la metodología MLOps y un modelo de Machine Learning. La metodología MLOps se presenta como una solución prometedora para combatir este problema, proporcionando un marco de trabajo apropiado para los científicos de datos, lo que les permite integrar, automatizar y monitorear los modelos de Machine Learning.

En concreto, se busca implementar una metodología MLOps que permita la integración, automatización y monitoreo de un modelo de Machine Learning, específicamente diseñado para el reconocimiento y control de dichas plagas en el cultivo del aguacate Hass. Esta metodología fue validada mediante su despliegue en un entorno controlado, lo que permitió monitorear y mejorar continuamente el rendimiento del modelo.

Además, se planeó, desarrolló y entrenó este modelo de Machine Learning utilizando técnicas apropiadas de preprocesamiento y selección de características para garantizar una detección de las plagas en el cultivo de aguacate Hass. Con esto se obtuvo una herramienta digital accesible para los científicos de datos, que facilite la predicción y prevención de la aparición de plagas, lo que constituirá un recurso valioso para ellos.

Finalmente, se espera que los resultados de este proyecto incluyan un informe detallado sobre el diseño, ejecución y evaluación de la metodología MLOps, así como la creación de una metodología MLOps que permitió el monitoreo y reevaluación continua del rendimiento del modelo de Machine Learning a desarrollar. Este enfoque MLOps permitirió un seguimiento más adecuado del cultivo de aguacate Hass, contribuyendo así a la sostenibilidad y productividad del sector agrícola.

Resumen

Este estudio se enfocó en la implementación de una metodología MLOps en la agricultura, específicamente en el cultivo del aguacate Hass, que enfrenta desafíos como las plagas. La metodología MLOps se destaca por mantener la operación y el despliegue de modelos de aprendizaje automático mientras se mejora su rendimiento. El objetivo es desarrollar un modelo de Machine Learning para el reconocimiento y control de plagas, utilizando técnicas de preprocesamiento y selección de características. Se propuso la implementación de una metodología MLOps que permitió la integración, automatización y monitoreo del modelo ML, validándola en un entorno controlado. Se créo una herramienta digital para los científicos de datos, facilitando la predicción y prevención de plagas. El proyecto generó un informe detallado del diseño, ejecución y evaluación de la metodología MLOps, así como la creación de una metodología que permita reevaluar continuamente el rendimiento del modelo de Machine Learning. Este enfoque contribuye a la sostenibilidad y productividad del sector agrícola.

Palabras Clave Big Data, Machine Learning, MLOps, Cultivo de aguacate, Plagas *Stenoma catenifer* y *heilipus lauri*.

Abstract

This study focused on the implementation of the MLOps methodology in agriculture, specifically in Hass avocado cultivation, which faces challenges such as pests. The MLOps methodology stands out for maintaining the operation and deployment of machine learning models while improving their performance. The objective is to develop a machine learning model for pest recognition and control, utilizing preprocessing techniques and feature selection. It was proposed to implement an MLOps methodology that allows for the integration, automation, and monitoring of the ML model, validating it in a controlled environment. The project aims to create a digital tool for data scientists, facilitating pest prediction and prevention.

The project generated a detailed report on the design, execution, and evaluation of the MLOps methodology, as well as the creation of a methodology that enables continuous re-evaluation of the machine learning model's performance. This approach contributes to the sustainability and productivity of the agricultural sector.

Keywords Big Data, Machine Learning, MLOps, Hass avocado cultivation, *Stenoma catenifer* and *heilipus lauri* Pests.

Índice

Índice de figuras

Índice de tablas

1. Introducción

En los últimos años alrededor del mundo se viene implementando la metodología MLOps que permite la implementación y el despliegue eficiente y escalable de modelos de aprendizaje automático (Machine Learning) para los científicos de datos en diferentes entornos productivos como los agrícolas. Este tipo de intervención tecnológica se traduce en la capacidad de desarrollar modelos de predicción y análisis de datos agrícolas, como pronósticos climáticos, análisis de suelos, monitoreo de cultivos y detección temprana de enfermedades o plagas (Organización de las Naciones Unidas para la Alimentación y la Agricultura, 2021).

El uso de los modelos de Machine Learning en la agricultura ofrece varias ventajas significativas, como por ejemplo permite aprovechar los datos recopilados de sensores y dispositivos IoT para tomar decisiones en datos y en tiempo real, asimismo optimizar el riego en el proceso de cosecha, la aplicación de fertilizantes y pesticidas y la planificación de la cosecha, que son aspectos clave para la producción agrícola y necesarios para los científicos de datos.

Con la ayuda de la metodología MLOps, estos modelos propician la automatización de tareas repetitivas y complejas, como el procesamiento de grandes volúmenes de datos, la generación de informes y la gestión de la logística (Arley, O. y Llano, R., 2016; Monsalve, 2021), acciones que ahorran tiempo y recursos, permitiendo que los científicos de datos se enfoquen en actividades estratégicas y en la toma de decisiones de manera eficaz.

En este sentido, el MLOps para el aprendizaje automático, es un enfoque que combina prácticas y herramientas de desarrollo de software con técnicas de aprendizaje automático, brindando a los científicos de datos una serie de beneficios significativos como la automatización de tareas repetitivas, el entrenamiento y despliegue de modelos, además, facilita la colaboración entre los equipos de ciencia de datos y operaciones, promoviendo la comunicación fluida y el intercambio de conocimientos.

El MLOps proporciona a los científicos de datos una infraestructura sólida y procesos eficientes para desarrollar, implementar y mantener modelos de aprendizaje automático, en los distintos escenarios económicos y productivos, donde estas estrategias favorecen a las prácticas de control de versiones y monitoreo continuo, garantizando la trazabilidad y el control de calidad de los modelos.

La adopción de la metodología MLOps en la agricultura obedece a la capacidad para mejorar la calidad y la precisión de los resultados, ya que los modelos de aprendizaje automático pueden analizar patrones complejos en los datos y generar predicciones más precisas sobre el rendimiento de los cultivos, la salud de los suelos y otros aspectos agrícolas.

2. Definición del problema

2.1. Planteamiento del problema

La producción agrícola del aguacate Hass para el caso de México en el año 2019 correspondió a 2.4 millones de toneladas, aportando el 45 % en las exportaciones de este país y aumentando la cantidad de exportaciones en 22 % en el año 2020 (Cruz et al., 2022). Estos aportes se reflejan en el PIB, contribuyendo al avance socioeconómico de las regiones agrícolas.

El aguacate Hass corresponde aproximadamente al 82 % de todos los aguacates, siendo el más consumido a nivel mundial y de acuerdo con los datos de la Organización de las Naciones Unidas para la Alimentación y la Agricultura (FAOSTAT) (2021) el primer país productor de aguacate Hass es México con unas 2.393.849 toneladas al año, seguido de Colombia con unas 876.754 toneladas al año y de República Dominicana con 676.373 toneladas al año.

Los avances en la agroindustria en Colombia contribuyen al ámbito económico y laboral, siendo en la actualidad la producción agrícola del cultivo de aguacate Hass un producto de alta demanda a nivel nacional e internacional. Para el Departamento Administrativo Nacional de Estadística (DANE) (2016), las problemáticas a tener en cuenta en el cultivo del aguacate Hass corresponden a: factores atmosféricos relacionados con la temperatura, las precipitaciones, el viento, la altitud, los factores de las condiciones del terreno y los factores relacionados con la siembra donde se encuentra la fertilización, los abonos y el tratamiento de las plagas y enfermedades.

Dentro de las plagas más preocupantes en el cultivo del aguacate Hass se encuentran la *Stenoma catenifer* y el *heilipus lauri*, insectos y larvas que introducen sus huevos provocando el daño en las semillas de los frutos en crecimiento. Además, el *Stenoma catenifer* impacta en el fruto al perforar el brote terminal y los laterales del aguacate, formando túneles de hasta 25cm, corta los pedúnculos que son el pezón de la hoja y la base de los frutos pequeños, como resultado los frutos verdes y pequeños caen.

Para prevenir y controlar estas plagas, se recomienda implementar prácticas agrícolas adecuadas, como el manejo integrado de plagas, la selección de variedades resistentes y el control de la humedad en el suelo. Además, se deben realizar monitoreos constantes para detectar y tratar a tiempo cualquier plaga o enfermedad que pueda aparecer en el cultivo del aguacate Hass.

Para la detección de este tipo de plaga en la producción agrícola del cultivo del Hass existe el método Manejo Integrado de Plagas (MIP) en el cual se señala el monitoreo de manera manual y de observación constante partiendo de tres elementos, el primero corresponde a la prevención en relación a los cuidados, las restricciones y la limpieza del personal y sus utensilios de trabajo, el segundo al control donde se utilizan evaluaciones y registros manuales, instalación de trampas y el tercero es el manejo de la enfermedad en el cual se genera una protección y cuidado de las plantas dependiendo de los patógenos dañinos (Instituto Colombiano Agropecuario, 2012).

El cultivo del aguacate Hass en Colombia ha tenido una gran demanda a nivel nacional e internacional, generó un crecimiento del 34 % del total de área sembrada de aguacate. Además al ser un producto que presenta una cosecha constante por las condiciones del relieve y climáticas del país viene en un crecimiento de área sembrada de un 65 % anual desde el año 2019 (Proyecto Colombia Mide, 2021).

En este sentido, el Machine Learning permite a los científicos de datos, a través de imágenes, el reconocimiento de patrones de concentración y expansión de las plagas de manera óptima en todo el cultivo generando una reducción económica y mejorando la calidad del producto agrícola.

Dadas estas circunstancias, para los científicos de datos la implementación de la metodología MLOps se presenta como una oportunidad para mejorar la calidad, confiabilidad y eficiencia de los modelos de Machine Learning utilizados en la detección de plagas, evaluación del nivel de daño y reconocimiento de deformaciones y coloraciones específicas en las áreas afectadas. Al someter los modelos a rigurosos procesos de control de calidad, los científicos de datos con esta metodología garantizan la trazabilidad y transparencia a lo largo de todo el ciclo de vida del modelo, brindando así resultados más precisos y confiables.

La utilización de MLOps se ha convertido en una práctica cada vez más extendida en el campo de la ciencia de datos, y se ha demostrado que mejora significativamente la eficiencia y la seguridad en la implementación de modelos de Machine Learning (Géron, 2019). Su aplicación en el contexto de la agricultura y la predicción de plagas y enfermedades puede ser un paso importante para mejorar la productividad y sostenibilidad del cultivo de aguacate Hass y otros cultivos agrícolas.

Es importante destacar la relevancia de utilizar metodologías de MLOps para garantizar el correcto desarrollo, implementación y mantenimiento continuo de un modelo de control y cuidado de plagas permitiendo mejorar los procesos productivos agrícolas en Colombia. El modelo de Machine Learning al ser un programa orientado a la automatización y actualización constante de sus tareas avanza en el mejoramiento y la eficiencia de su procesamiento de información de manera continua a través de la metodología MLOps aportando a los procesos de análisis a los científicos de datos.

2.2. Formulación del problema

En este contexto la investigación busca desarrollar una herramienta digital que ayude a pronosticar y prevenir la presencia o no de las plagas como el *Stenoma catenifer* y el *heilipus lauri* en el cultivo de aguacate Hass, entendiendo que es crítica la detección temprana del brote en un cultivo, se propone la creación de un software accesible para los científicos de datos que les permita abordar esta problemática. Con base en esto, surge la siguiente pregunta de investigación: ¿Cómo el uso de la metodología MLOps en el desarrollo de un modelo de Machine Learning facilita la integración, la actualización y el despliegue continuo del reconocimiento de las plagas *Stenoma catenifer* y *heilipus lauri* en el cultivo de aguacate Hass, contribuyendo a mejorar los modelos agrícolas de forma automática y brindando beneficios económicos y sociales a la comunidad de científicos de datos? Asimismo ¿Cómo mantener el programa de Machine Learning de forma automatizada y con supervisión continua, de modo que no se vea comprometido su rendimiento?

3. Objetivos del proyecto

3.1. Objetivo General

Implementar la metodología MLOps para la entrega continua de un modelo de Machine Learning para el reconocimiento y control de las plagas *Stenoma catenifer* y *heilipus lauri* en el cultivo de aguacate Hass.

3.2. Objetivos específicos

- Implementar técnicas de procesamiento de imágenes para extraer características relevantes y mejorar la capacidad del modelo de Machine Learning en el reconocimiento y detección de las plagas *Stenoma catenifer* y *heilipus lauri* en el cultivo de aguacate Hass a partir de imágenes capturadas en campo.
- Desarrollar y entrenar un modelo de Machine Learning utilizando técnicas apropiadas de preprocesamiento y selección de características, así como algoritmos de aprendizaje supervisado o no supervisado, para lograr una detección de las plagas *Stenoma catenifer* y *heilipus lauri* en el cultivo de aguacate Hass.
- Desarrollar una metodología MLOps que permita la integración, automatización y monitoreo del modelo de Machine Learning diseñado para el reconocimiento y control de las plagas *Stenoma catenifer* y *heilipus lauri* en el cultivo de aguacate Hass.
- Validar el uso de MLOps mediante despliegue en un ambiente controlado, con la capacidad de monitorear y reevaluar continuamente el rendimiento del modelo.

3.3. Resultados esperados

En esta propuesta de investigación para la maestría en desarrollo de ingeniería de software se plantearon los siguientes resultados esperados:

1. Implementación de un modelo de Machine Learning para detectar las plagas *Stenoma catenifer* y *heilipus lauri* en el cultivo de aguacate hass.

2. La implementación de la estrategia MLOps que permitirá la integración, automatización y monitoreo continuo del modelo de Machine Learning.

3. Generación de una vista de despliegue que proporcione los componentes de la metodología MLOps.

4. La demostración del modelo de Machine Learning utilizando la metodología MLOps, con la capacidad de monitorear y reevaluar continuamente el rendimiento del modelo.

4. Alcance

Los alcances que se desarrollaron en esta investigación correspondieron a una implementación de una metodología de MLOps para contribuir a un modelo de Machine Learning permitiendo la integración, la actualización y el despliegue continuo del reconocimiento de plagas *Stenoma catenifer* y *heilipus lauri* en el cultivo de aguacate Hass, partiendo de las siguientes fases:

- **Experimentación / Desarrollo / Pruebas.**
  Feature Store: validación y preparación de los datos.
  Código de fuente.
  Model Registry.
- **Pre-Producción y Producción.**
  El modelo se despliega y se monitoriza. Pipelines
- **Fase de despliegue del modelo**

El desarrollo de una metodología MLOps que permita la implementación de modelos con mayor rapidez con procesos automatizados, contribuyendo a acelerar el tiempo de creación de valor al entregar información de manera ágil.

Fuera del alcance se encuentra la optimización del modelo de manera constante y la reutilización del modelo a medida que los datos evolucionan con el tiempo generando de manera efectiva mejoras al rendimiento y la adaptabilidad del modelo a partir de técnicas de transferencia de aprendizaje, donde se ajustan los pesos y las capas del modelo para adaptarse a los nuevos datos.

5. Justificación del trabajo de grado

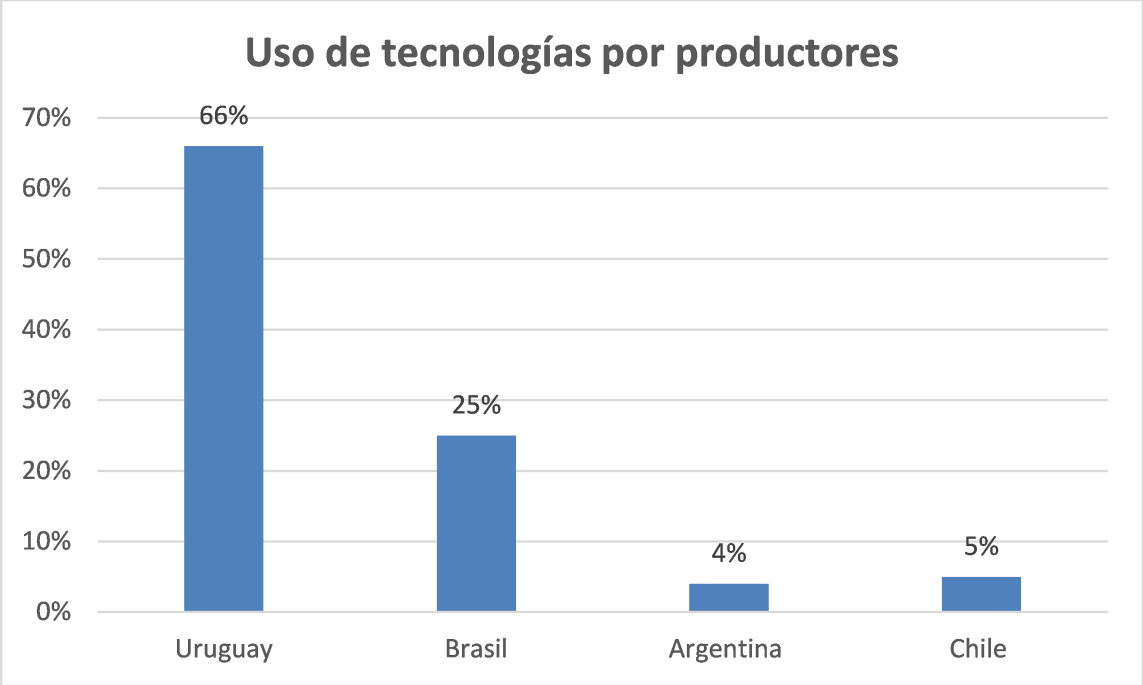
En todos los ámbitos de la vida de las personas como la educación, la economía, la política, la cultura y el medio ambiente pueden existir procesos de transformación. Estas transformaciones sociales a nivel mundial generan acciones de aprendizajes y de mercados globales, donde las tecnologías y los aportes de nuevos o mejorados software los cuales "generan reducciones significativas de costos por la experiencia y utilización de patentes o porque aportan beneficios por la capacidad de vender variedades similares de productos en diversos mercados" (Vela, 2012, p. 27).

En la actualidad, la agricultura es un renglón económico que viene identificando diseños de políticas públicas para la intervención de aplicaciones tecnológicas y generar el aprovechamiento y mejoras de la productividad agraria a nivel mundial. Estas iniciativas resultan de las dificultades económicas y de productividad agraria, las cuales deben de ser transformadas para el desarrollo del campo, como explica la FAO (2021):

> La incorporación de estas tecnologías supone la generación de información (basada en la recopilación y procesamiento de datos) e indicaciones que permiten el monitoreo, el análisis, la planificación y el control inteligente de procesos de producción, transformación, distribución y comercialización de productos agrícolas (Organización de las Naciones Unidas para la Alimentación y la Agricultura, 2021, p.88).

El uso de tecnología y datos se ha vuelto cada vez más importante para mejorar la eficiencia y la productividad en la agricultura. En este contexto, el empleo de la metodología MLOps (Machine Learning Operations) se justifica como una herramienta poderosa para potenciar la agricultura y lograr una gestión más eficiente de los recursos. Estas mejoras en América Latina proporcionan una sostenibilidad en la producción agrícola y contribuye a procesos benéficos para el medio ambiente (ver figura 1).

Figura 1: Acceso y aprovechamiento de tecnología digital en la agricultura de América del Sur



Fuente: Información tomada de FAO 2021

El uso de la metodología MLOps en la agricultura ofrece mejoras debido a su capacidad para optimizar la toma de decisiones, automatizar tareas y mejorar la precisión de los resultados de los modelos de machine learning. Esta tecnología puede marcar una gran diferencia en la eficiencia y la sostenibilidad de la agricultura, ayudando a enfrentar los desafíos actuales y futuros del sector.

De allí que, la metodología MLOps tiene un gran potencial para transformar la agricultura y brindar beneficios significativos a los científicos de datos involucrados en esta industria, debido a que pueden aprovechar varias ventajas, como por ejemplo el MLOps permite una mejor gestión de los modelos de aprendizaje automático, es posible utilizar prácticas de control de versiones para rastrear y gestionar los cambios en los modelos, lo que facilita la colaboración y la reproducibilidad de los resultados en las áreas de producción agrícola. Además, el MLOps garantiza un monitoreo continuo de los modelos en producción, lo que permite identificar y solucionar problemas rápidamente.

Al utilizar técnicas de aprendizaje automático, los científicos de datos pueden analizar

grandes volúmenes de datos agrícolas para identificar patrones y tomar decisiones informadas, ayudando a implementar estos modelos en sistemas integrados, lo que permite la automatización de tareas agrícolas como el riego, la fertilización y la detección de enfermedades. Esto conduce a un uso más eficiente de los recursos, reduciendo costos y minimizando el impacto ambiental.

En este sentido la tecnología constituye una herramienta relevante para el mundo actual, de allí que este tipo de investigaciones permitan abordar métodos aplicables en la ingeniería de software tanto para la generación de conocimientos, como métodos de aplicación en escenarios reales, como la agricultura y, en particular, el cultivo del aguacate Hass. Asimismo, esta clase de investigaciones abre la vía para la articulación entre los profesionales, las universidades y la industria agrícola para fomentar prácticas económicas y tecnológicas, mejorar y fortalecer actividades de investigación en laboratorios de computación y programas de alfabetización digital, con el objetivo de contribuir al desarrollo económico, científico y académico, asegurando que la ingeniería de software se emplee de forma concreta y provechosa en la solución de problemas prácticos en diversas áreas, como la agricultura.

6. Marco teórico de referencia y antecedentes

El marco de referencia está organizado en un primer momento con el análisis del estado del arte para la comprensión del objeto de estudios sobre MLOps para el ámbito agrícola y un segundo momento con los elementos teórico-conceptuales que permiten la argumentación teórica sobre Machine Learning, la metodología MLOps y el cultivo de aguacate Hass con las plagas detalladas para esta investigación.

6.1. Estado del arte

Las investigaciones a nivel internacional se describen a continuación:

Tabla 1: Investigaciones internacionales

Autor	**Título**	**Objetivo**	**Metodología**	**Resultado**
Aimacaña and Columba (2021)	Análisis comparativo de algoritmos de Machine Learning para la detección de plagas en los cultivos representativos de la sierra ecuatoriana	Realizar un análisis comparativo entre distintos algoritmos de Machine y Deep Learning, que permita determinar cuál de ellos ofrece mejores resultados en la detección de plagas en cultivos endémicos de la serranía ecuatoriana.	Metodología CRISPDM la cual sus siglas significan Proceso Estándar Entre Industrias para la Minería de Datos.	El estudio realizado se aplicó el mejor modelo para la construcción de un prototipo el cual fue la red neuronal convolucional InceptionV3; el prototipo realizado, permite al usuario enviar imágenes para ser procesadas por el modelo y este a su vez reciba una predicción e información de los síntomas, además de una sugerencia de un posible tratamiento.

Tabla 1: Investigaciones internacionales

Autor	**Título**	**Objetivo**	**Metodología**	**Resultado**
Castañeda et al. (2021)	Detección de nutrientes del suelo y planta, y pestes en campos de cultivo de banano orgánico con Machine Learning	Diseño una plataforma que sirva de apoyo al agricultor para poder estimar qué macronutrientes tiene en deficiencia su planta de banano, con la finalidad de obtener un mejor producto en la cosecha del banano orgánico, así como una posible interfaz para la detección de plagas en el banano.	Se utilizó un conjunto de fotografías público, el cual fue encontrado en un repositorio y se escogieron aquellas plagas que afectaban al banano. Posteriormente, se realizó un aumento a este conjunto de fotografías mediante transformaciones lineales y las imágenes resultantes fueron pre procesadas en diferentes espacios de color para ser utilizadas como entradas a la red neuronal.	El modelo permitió una alta precisión a través de diferentes métricas. Continuamente se desarrolló un prototipo de plataforma web para que los agricultores en un futuro pudieran acceder al sistema.

Tabla 1: Investigaciones internacionales

Autor	**Título**	**Objetivo**	**Metodología**	**Resultado**
Valenzuela (2022)	Detección y Clasificación de Enfermedades en el Tomate Mediante Deep Learning y Computer Visión	Aplicar el aprendizaje profundo a problemas de visión de computadora tales como detección y clasificación específicamente en las enfermedades del tomate mediante el procesamiento de imágenes digitales.	Redes neuronales preentrenadas utilizando transfer learning para llevar a cabo la detección de hojas de tomate, utilizando inicialmente Faster Mask R-CNN, seguido por una red neuronal convolucional para clasificar la enfermedad. Esta clasificación se basa en el área de la imagen donde se ubica la hoja. Una vez clasificada, el sistema proporciona información sobre los síntomas asociados con la enfermedad, así como recomendaciones en tiempo real sobre la prevención y el tratamiento adecuado.	Con el advenimiento de las redes neuronales, ha permitido mejorar drásticamente la precisión, tanto de la detección de objetos como de la clasificación de imágenes.

Tabla 1: Investigaciones internacionales

Autor	**Título**	**Objetivo**	**Metodología**	**Resultado**
García and Ulloa (2022)	Implementación de modelo Machine Learning aplicado al estudio de enfermedades de café en el centro de investigación Sacha Wiwa, perteneciente a la parroquia Gasaganga, Cantón la Maná, provincia de Cotopaxi	Implementar modelo de clasificación de imágenes empleando técnicas de Machine Learning, para la detección de enfermedades del Cafeto (Coffea arábica) en huertas del Centro de Investigación Sacha Wiwa.	Cualitativa experimental permite la selección, análisis y presentación de los datos documentados de una manera ordenada y siguiendo los objetivos del proyecto.	Utilizando el entorno virtual "Jupyter Lab", se logró desarrollar y entrenar un modelo de Machine Learning que emplea redes neuronales convencionales para clasificar las enfermedades del café mencionadas en el caso de estudio, junto con la clasificación de plantas de café sanas.

Fuente: Elaboración propia

En las investigaciones a nivel internacional se puede observar que el desarrollo del Machine Learning, en particular el uso de redes neuronales, ha demostrado ser de gran importancia en el control de plagas y enfermedades en diversos campos, como la agricultura. Este tipo de método utiliza un modelo de red neuronal convolucional para la detección de plagas y enfermedades como por ejemplo en los cultivos de café, tomate y banano al contribuir al control de plagas.

Uno de los principales beneficios del Machine Learning en el control de plagas es su capacidad para analizar grandes volúmenes de datos y extraer patrones y características relevantes, en las investigaciones se observaron que utilizaban imágenes de plantas enfermas como sanas, lo que le permitió aprender a reconocer los síntomas y distinguir entre diferentes enfermedades. Esto es especialmente útil en el control de plagas, ya que puede fa-

cilitar la detección temprana de enfermedades y ayudar a los agricultores a tomar medidas preventivas o correctivas de manera oportuna.

Para el caso de las investigaciones nacionales se detallan en la siguiente tabla:

Tabla 2: Investigaciones a nivel nacional

Autor	**Título**	**Objetivo**	**Metodología**	**Resultado**
Tovar (2019)	Diseño e implementación de un aplicativo móvil para realizar la detección temprana de la enfermedad de la Sigatoka Negra en los cultivos de plátanos	Diseñar e implementar un aplicativo móvil, para realizar la detección temprana de la enfermedad de la Sigatoka Negra en los cultivos de plátanos.	La metodología empleada para detectar esta enfermedad implicó la creación de una base de datos que contiene imágenes representativas de sus posibles estados, utilizando la escala estándar de Fouré para analizar la progresión de la Sigatoka Negra. Posteriormente, se desarrolló un algoritmo que segmentó cada una de las imágenes para resaltar la enfermedad, extrayendo luego las características pertinentes. Finalmente, mediante técnicas de aprendizaje automático, se generó un modelo de clasificación capaz de identificar el estado de una hoja al capturar una imagen que no esté en la base de datos.	El aplicativo móvil logró una precisión del 80 % en la identificación de la enfermedad de la Sigatoka Negra. Por ende, esta aplicación tiene el potencial de ser una herramienta valiosa para los productores de plátano al prevenir posibles pérdidas económicas.

Tabla 2: Investigaciones a nivel nacional

Autor	Título	Objetivo	Metodología	Resultado
Ávila (2022)	Entrenamiento de redes neuronales para la identificación de plagas en cultivos de café.	Implementar un aplicativo basado en visión artificial para la identificación de Cercospora, Phoma, Roya, minador de hojas y Arañita roja, en el café a partir del análisis de las hojas de la planta.	Esta base será introducida a modelos de redes neuronales: densa, neuronal convolucional y neuronal convolucional con overfitting, usando el lenguaje Python y las bibliotecas: Keras y TensorFlow. Una vez entrenado el modelo, se procede al análisis de las imágenes que contienen la morfología de la planta y que servirán de entrada a una red neuronal para poder comparar los resultados y determinar cuál de los modelos presenta mejores resultados.	El modelo convolucional con overfitting ofrece la mejor capacidad predictiva, aunque sigue siendo un modelo restringido que podría ser sustituido por otros modelos más amplios, como ResNet50, CIFAR-10 o VGG19. Dado el número limitado de plagas disponibles, es claro que el reconocimiento detallado de las plagas no es factible en la actualidad, lo que resalta la necesidad de expandir la base de datos.

Fuente: Elaboración propia

A nivel nacional los diferentes modelos operativos en el contexto de la detección de plaga hace referencia a un modelo convolucional con overfitting que presenta los mejores resultados de predicción, pero se plantea la posibilidad de reemplazarlo por otros modelos

de mayor alcance, que emplean las arquitecturas convolucionales ResNet50, CIFAR-10 o VGG19.

La elección del modelo operativo es crucial en cualquier aplicación de Machine Learning, ya que tiene un impacto directo en la precisión y el rendimiento del sistema. En el caso mencionado, el modelo convolucional con overfitting permitió avanzar en el acierto de la detección de plaga.

6.2. Bases teóricas

6.2.1. Metodología MLOps

Machine Learning Operations (en sus siglas en inglés MLOps) es un conjunto de prácticas que combinan Machine Learning, DevOps e Ingeniería de datos, cuyo objetivo es implementar y mantener sistemas ML en producción de manera confiable y eficiente.

En esa dirección Rivero (2022) explica que, MLOps se refiere a las prácticas y metodologías que permiten gestionar y optimizar el ciclo de vida de los sistemas de aprendizaje automático (Machine Learning) de forma que se puede desarrollar confiablemente. Se tiene en cuenta que, la metodología MLOps combina técnicas y herramientas de desarrollo de software con conocimientos de ciencia de datos y operaciones de infraestructura, para permitir la implementación, el monitoreo y el mantenimiento escalable de los modelos de aprendizaje automático en producción.

En el contexto del desarrollo de modelos de Machine Learning, MLOps aborda los desafíos asociados con la gestión de datos, la experimentación, el entrenamiento, la implementación, el monitoreo y la reevaluación continua del rendimiento del modelo, que lo hacen diferente a DevOps y la Ingenieria de Datos como se describe en la Tabla 3. Esto implica el uso de técnicas como la integración y entrega continua (CI/CD), la orquestación de flujos de trabajo, la automatización de pruebas y la gestión de versiones.

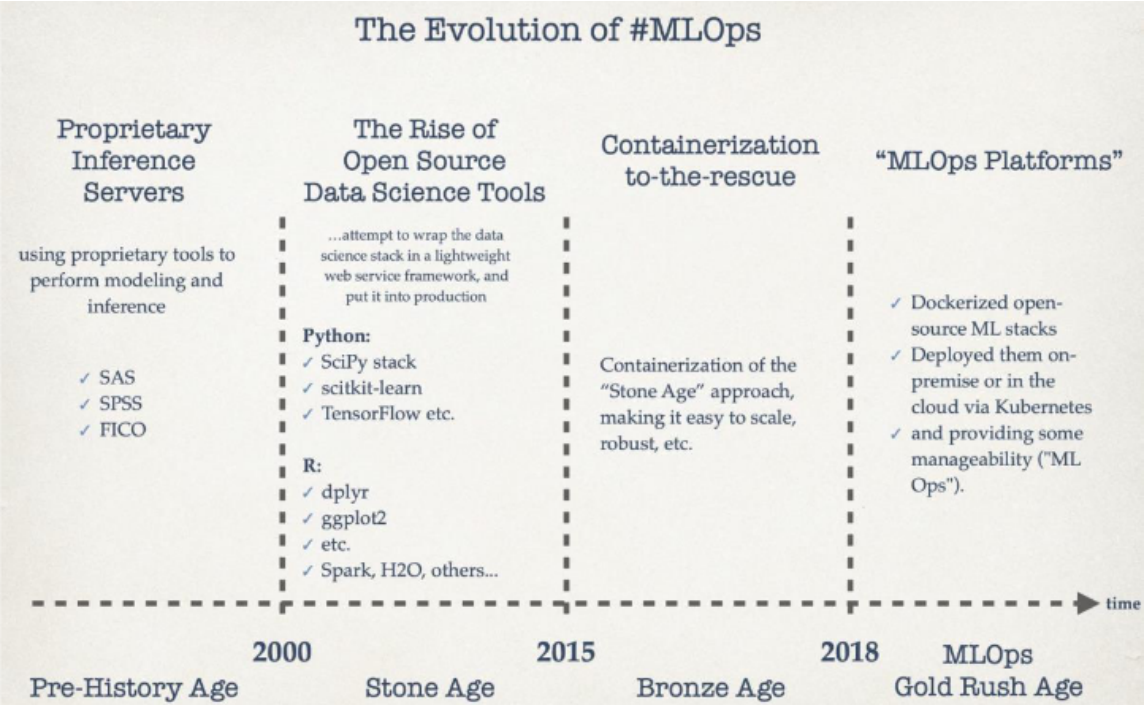
Tabla 3: Comparativo entre DevOps, Ingeniería de Datos y MLOps

Práctica	**DevOps**	**Ingeniería de Datos**	**MLOps**
Control de versiones	Versionamiento de código	Versionamiento de código linaje de datos	Versionamiento de código + datos + modelos (conectados)
Pipeline	n/a	Flujo de datos/ETL	Flujo de ML de entrenamiento, flujo de ML para predicción
Validación de comportamiento	Pruebas unitarias	Pruebas unitarias	Validación de modelo
CI/CD	Despliegue de código en producción	Despliega código en flujo de datos	Despliegue de código a producción + Flujo de datos de ML
Validación de datos	n/a	Validación de negocio y formato	Validación estadística
Monitoreo	Basados en SLOs	Basados en SLOs	SLOs + monitoreo diferencial y estadístico

Fuente: Rivero (2022)

El objetivo fundamental de MLOps es garantizar que los modelos de aprendizaje automático sean confiables, reproducibles y escalables en entornos de producción. Al adoptar prácticas de MLOps, las organizaciones pueden acelerar el tiempo de comercialización de sus modelos, mejorar la colaboración entre equipos multidisciplinarios, minimizar los riesgos asociados con los modelos en producción y facilitar la implementación de mejoras y actualizaciones. Estos avances en la metodología MLOps se vienen desarrollando desde el año 2000 para resolver los distintos problemas para los contextos ML (ver figura 2).

Figura 2: Evolución temporal de MLOps



Fuente: Visengeriyeva et al. (2020)

La metodología MLOps es una disciplina que fusiona las mejores prácticas de desarrollo de software con los desafíos específicos del Machine Learning, con el objetivo de facilitar la implementación y gestión de modelos de aprendizaje automático.

Con la creciente popularidad del software de código abierto y la amplia disponibilidad de datos, un número cada vez mayor de profesionales en el campo del desarrollo de software optaron por utilizar bibliotecas en lenguajes como Python o R para llevar a cabo el entrenamiento de modelos de aprendizaje automático. A medida que avanzaba la tecnología de la contenerización, surgió una solución para abordar el despliegue escalable de modelos mediante el uso de contenedores Docker y Kubernetes. En tiempos recientes, hemos observado una evolución en estas soluciones, que ahora han dado paso a plataformas de implementación de aprendizaje automático que abarcan todo el ciclo de vida del modelo, desde la experimentación inicial, pasando por el entrenamiento, la implementación y hasta el monitoreo continuo del rendimiento del modelo (Visengeriyeva et al., 2020).

6.2.2. Proceso de la metodología MLOps

- **Flujos de ML:** Los procesos de flujo de datos, conocidos también como pipelines de datos o ETL (extraer, transformar y cargar), implican la obtención de datos desde una fuente, su posterior transformación y finalmente su carga en la misma fuente original o en otro sistema. La fase de transformación de los datos abarca todas las modificaciones necesarias para garantizar que los datos estén en el formato requerido por el destino final, que en este caso sería el modelo de aprendizaje automático.

- **Versionado de modelos y de datos:** En aprendizaje automático es crucial, ya que no solo implica mantener un registro de las distintas versiones del código, sino también de las versiones del modelo en sí mismo, así como de los conjuntos de datos utilizados en su entrenamiento, junto con otros detalles como los hiper-parámetros del modelo.

- **Validación de modelos:** Se debe tener en cuenta:

  1. **Precisión del modelo:** de los casos que el modelo clasificó como verdaderos positivos (TP).
  2. **Recall del modelo:** la precisión abierta por categorías.
  3. **Medición de sesgo en los datos:** es importante realizar mediciones que ayuden a entender si el modelo está sesgado.
  4. **Casos críticos:** un modelo puede tener buena precisión, exhaustividad(recall) y no estar sesgado, pero no tomar los casos más importantes en la historia reciente.
  5. **Medición de un modelo de regresión:**

     - Evaluación general del modelo mediante métricas como RMSE, MAE y R2.
     - Desglose de estas métricas por diferentes segmentos para identificar áreas donde el modelo podría necesitar mejoras específicas.
     - Análisis del sesgo presente en los datos, similar al enfoque utilizado en problemas de clasificación.
     - Identificación casos críticos, empleando un enfoque similar al utilizado en problemas de clasificación.

- **Validación de datos:** Se realiza en dos etapas. La primera etapa, más elemental, implica evaluar la calidad de los datos, lo que incluye verificar la cantidad de valores nulos, el cumplimiento del acuerdo de nivel de servicio (SLA) para la actualización de los datos, asegurando que todos los datos sean del tipo esperado, entre otros aspectos. La segunda etapa, más compleja, implica monitorear la distribución de los datos. Cualquier cambio en esta distribución podría deberse a un fallo en alguno de los pipelines o a cambios en los datos mismos. (Rivero, 2022).

Entre las diversas herramientas disponibles para el MLOps, se pueden clasificar en dos categorías distintas: las sencillas y las complejas. Las herramientas sencillas son particularmente útiles para facilitar el despliegue local, es decir, en un mismo ordenador o un servidor local. Este enfoque es especialmente adecuado para equipos pequeños de investigación, donde estos miembros del equipo pueden acceder desde diferentes computadoras de manera local. Estas herramientas sencillas representan el primer paso para llevar la solución a la fase de despliegue, especialmente en entornos de producción.

Por otro lado, las herramientas complejas están diseñadas para gestionar mayores capacidades de cómputo. Estas pueden configurarse con capacidades específicas, e incluso el sistema puede autoajustarse según el tráfico y los requisitos de almacenamiento, incorporando la funcionalidad conocida como "On Demand", donde son capaces de autoescalar según las necesidades del momento.

Después de implementar las herramientas sencillas, se puede aprovechar lo creado para desplegar una aplicación. Esta aplicación, inicialmente desarrollada o creada localmente, puede integrarse con uno de los servicios en la nube más comunes y populares. Estos servicios en la nube, como AWS, Azure, GCP, son seleccionados debido a su amplia cuota de mercado y su uso generalizado en la actualidad. Esto permite llevar la aplicación creada localmente a aprovechar las capacidades y escalabilidad que ofrecen los servicios en la nube para su despliegue.

La implementación de un monitoreo completo plantea ciertas consideraciones, especialmente cuando se considera la adopción de herramientas complejas. La complejidad de estas herramientas conlleva un aumento en los costos, ya que se aplican cargos por el conjunto completo de funcionalidades que ofrecen. En comparación, al optar por el uso de herramientas sencillas locales y utilizar solo una fracción de las herramientas complejas resulta más económico.

Además del aspecto financiero, es crucial tener en cuenta que el uso de herramientas complejas implica una curva de aprendizaje significativa. Se requiere una inversión de tiempo y esfuerzo para adquirir los conocimientos necesarios y comprender cómo manipular eficazmente todo el ciclo del MLOps. Esta curva de aprendizaje elevada puede representar una desventaja, ya que impone una barrera para aquellos que buscan implementar estas herramientas de manera efectiva.

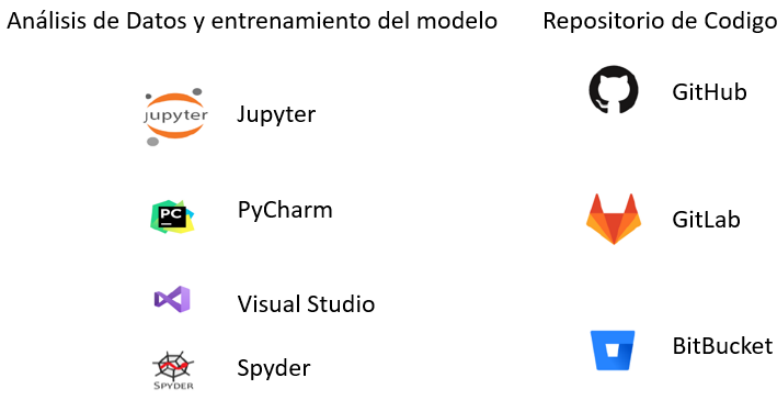
En general, se han ido desarrollando muchas herramientas que se pueden utilizar para implementar la metodología de MLOps tanto de acceso libre (open source) o de pago, para usar localmente o desde internet. Y se deben escoger cada una con base a las necesidades del proyecto.

Figura 3: Herramientas que implementan ciclo completo de MLOps

Fuente: Neptune (2024)

Dentro de las plataformas descritas en la imagen anterior (ver figura 3) se encuentra MetaFlow, esta plataforma en sí misma, es una herramienta gratuita, aunque algunas de sus funcionalidades pueden requerir pagos. Por ejemplo, el despliegue en internet podría estar sujeto a tarifas adicionales. Para completar todo el ciclo de vida del MLOps, es necesario considerar diversas herramientas, muchas de las cuales ofrecen características completas o parciales para gestionar eficazmente este ciclo.

Figura 4: Herramientas de análisis de datos y repositorio de código



Fuente: Elaboración propia

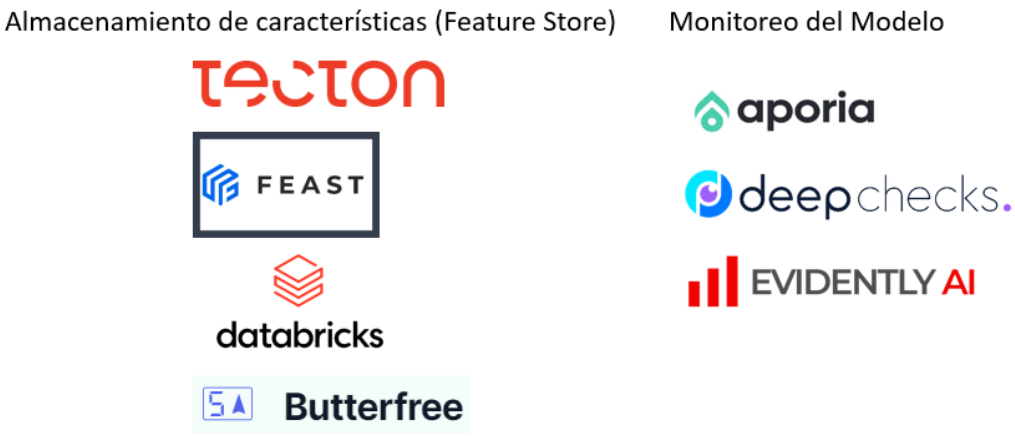
Figura 5: Herramientas de registro de modelo, versionado del modelo y versionado de datos

Fuente: Elaboración propia

Figura 6: Herramientas para Construcción de API y Construcción de aplicación Web

Fuente: Elaboración propia

Figura 7: Herramientas de Feature Store y Monitoreo del Modelo



Fuente: Elaboración propia

Continuando con el tema, centrémonos ahora en las herramientas que simplifican los despliegues de una aplicación o una API de manera automatizada. La necesidad de estas herramientas radica en lo tedioso que implicaría realizar despliegues manualmente. En un enfoque manual, se tendría que especificar la imagen que debe utilizarse desde un contenedor, crear el contenedor, y posteriormente enviarlo a la nube. Luego, sería necesario obtener la URL para su uso. Este proceso, aunque es posible de realizar manualmente, se vuelve más eficiente y menos propenso a errores al utilizar herramientas como GitHub Actions, la cual es una herramienta de automatización integrada con GitHub.

Algunas características clave y conceptos asociados con GitHub son:

- **Repositorios:** Son espacios donde se almacenan los archivos de un proyecto, junto con la información de seguimiento de versiones. Cada proyecto en GitHub se guarda en un repositorio.
- **Control de Versiones:** GitHub utiliza Git, un sistema de control de versiones distribuido. Esto significa que cada desarrollador o miembro de un equipo, que trabaja en un proyecto tiene una copia completa del historial de cambios del proyecto en su máquina local. Git permite rastrear cambios, fusionar contribuciones y revertir a versiones anteriores del código.
- **Automatización:** GitHub Actions es un servicio que permite a los ingenieros de software automatizar diversas tareas en respuesta a eventos específicos dentro del

repositorio. En el contexto del proyecto, las tareas se configuran mediante flujos de trabajo definidos en archivos YAML. Estos flujos de trabajo simplifican la ejecución de pruebas, la construcción de una API o una aplicación, junto con los despliegues. GitHub Actions no solo facilita la automatización de estos procesos, sino que también respalda la integración continua y el despliegue continuo.

Esta integración automatizada con GitHub Actions no solo agiliza el ciclo de desarrollo, sino que también asegura la coherencia y confiabilidad en el proceso de despliegue de una aplicación o una API. Con esta herramienta, se logra una gestión eficiente y automatizada de todo el ciclo de vida del modelo para llevarlo a producción, desde la integración de cambios hasta la entrega y despliegue en un proveedor de servicios en la nube.

6.3. Modelo de Machine Learning

De acuerdo con Salamanca and Castro (2021), el modelo Machine Learning se refiere a la localización automatizada de datos que se encuentran estandarizados, además es una herramienta constante en el proceso de extracción de grandes volúmenes de información para su organización y análisis, definido también como Big Data. En la actualidad el Machine Learning es un campo extenso que abarca diversas estrategias analíticas. Su objetivo principal es desarrollar algoritmos capaces de extraer información valiosa de los datos, ya sea para explicar, clasificar o predecir fenómenos (Pedro et al., 2021).

Estas nuevas tecnologías, a través del internet, permiten la recopilación de bases de datos de un tamaño considerable, pueden ser históricos o datos en tiempo real, siendo posible el análisis de datos debido a las mejoras constantes en la tecnología y el desarrollo de software. Para Salamanca and Castro (2021), este desarrollo ha permitido que las bases de datos se encuentren en distintos formatos, lo que es un desafío para orientar estrategias que permitan resolver distintos problemas de acuerdo a las necesidades de cada formato y de esta manera avanzar en el desarrollo de métodos para el Machine Learning.

Por esta razón con el Machine Learning es posible establecer tres tipos de modelos el geométrico, el probabilístico y el lógico (ver tabla 4) y tres tipos de aprendizajes: el supervisado, el no supervisado y semi-supervisado (ver tabla 5), ya que se busca el aprendizaje continuo, conocer el progreso y la veracidad de la información que se desarrolla, porque:

Su aplicación está determinada en diferentes ámbitos dependiendo de las necesidades del problema, permitiendo a los científicos y especialistas tomar decisiones

referentes a temas tan puntales como si un tipo de material puede ser separado por características de resistencia a la fragmentación o capacidad de absorción de energía pudiendo determinar con esto si es recomendable su uso en una u otra tarea específica (Salamanca and Castro, 2021, p.42).

Tabla 4: Modelos para el desarrollo de Machine Learning

Modelo	**Características**
Geométrico	El espacio de instancias abarca la totalidad de las configuraciones concebibles, ya sea que estén representadas en nuestro conjunto de datos o no. Suele exhibir una estructura geométrica definida. Por ejemplo, cuando todas las características son de naturaleza numérica, cada una puede interpretarse como una coordenada en un sistema cartesiano.
Probabilístico	Numerosos modelos se sustentan en el siguiente concepto fundamental: Consideremos X como las variables que describen, por ejemplo, las características de una instancia conocida; mientras que Y representa las variables objetivo que nos conciernen, como la clase a la que pertenece dicha instancia. La esencia del aprendizaje automático radica en la capacidad de modelar la interrelación entre X e Y.
Lógico	Toma su inspiración de disciplinas como la informática y la ingeniería. Se denomina "lógico"debido a que sus modelos pueden ser traducidos de manera sencilla en reglas comprensibles para los seres humanos, por ejemplo: `if Viagra=1 then Class=Y=spam`

Fuente: Salamanca and Castro (2021, p.43)

Tabla 5: Los tipos de aprendizaje en el ML

Tipos	**Fundamento**
Aprendizaje supervisado	Consta de descubrir el clasificador óptimo $f : X \to Y$ para una problemática específica. El proceso para encontrar este mapeo se conoce como algoritmo de clasificación, el cual genera un modelo para cada instancia de entrada $x \in X$ y su correspondiente clase $y \in Y$. Este modelo representa una aproximación de la distribución de probabilidad conjunta (DPC) de las variables X e Y.
Aprendizaje no supervisado	Se refieren a casos en los que el objetivo no es ajustar relaciones entre pares de datos de entrada y salida, sino en aumentar el entendimiento estructural de los datos disponibles (y de posibles datos futuros derivados del mismo fenómeno). Por ejemplo, esto podría implicar agrupar los datos en función de su similitud (clustering), simplificar sus estructuras manteniendo sus características esenciales (como en procesos de reducción de dimensionalidad), o extrayendo la estructura interna mediante la cual los datos se distribuyen en su espacio original (aprendizaje topológico).
Aprendizaje semi supervisado	El objetivo es la clasificación, pero la entrada contiene datos etiquetados y sin etiquetar. El aprendizaje semi-supervisado, debido a su estructura, se puede dividir principalmente en dos tipos, según el objetivo del análisis que se busque hacer a la base de datos: 1. Aprendizaje transductivo, que separa el conjunto dado en conjunto de entrenamiento (donde la variable objetivo es conocida), y conjunto de prueba donde ésta no es dada. 2. Aprendizaje inductivo trata de obtener una función de predicción usando todo el conjunto $\{(X, Y)\}$, es decir, usando tanto los datos donde la variable objetivo es conocida como aquellos en los que no.

Fuente: Salamanca and Castro (2021, p.44-45)

Autores como Contreras et al. (2022), Salamanca and Castro (2021) y Jara et al. (2016) explican que, en el campo del Machine Learning los métodos de predicción presentan un papel fundamental al permitir que las máquinas realicen predicciones precisas sobre datos no vistos. Existen varios métodos utilizados en el Machine Learning para generar predicciones, entre ellos se destacan los siguientes:

1. Regresión: Es utilizado para predecir un valor numérico continuo basado en variables independientes. Algunos ejemplos comunes son la regresión lineal y la regresión logística, que se utilizan para predecir la relación entre variables (Salamanca and Castro, 2021).

2. Clasificación: Se utiliza para predecir la pertenencia a una clase o categoría específica. Los algoritmos de clasificación, como el árbol de decisiones, las máquinas de vectores de soporte (SVM) y los clasificadores de Naive Bayes, se emplean para clasificar datos en categorías predeterminadas (Salamanca and Castro, 2021).

3. Agrupamiento: Es utilizado para agrupar conjuntos de datos similares en clusters o grupos. Algunos métodos comunes de agrupamiento son el algoritmo k-means y el agrupamiento jerárquico, que ayudan a encontrar patrones y estructuras ocultas en los datos (Contreras et al., 2022).

4. Redes neuronales: Estas técnicas se basan en modelos inspirados en el funcionamiento del cerebro humano. Las redes neuronales profundas, como las redes neuronales convolucionales (CNN) y las redes neuronales recurrentes (RNN), han logrado grandes avances en áreas como la visión por computadora y el procesamiento del lenguaje natural (Salamanca and Castro, 2021).

Estos métodos de predicción en el Machine Learning han demostrado ser eficientes en una amplia gama de aplicaciones, desde la detección de fraudes hasta el diagnóstico médico y la recomendación de productos. La elección del método adecuado depende del tipo de datos, el problema a resolver y los objetivos específicos del proyecto de Machine Learning. Para verificar su eficiencia, estos métodos deben ser validados por una matriz de confusión la cual contiene:

Figura 8: Matriz de confusión para la evaluación de los modelos

1 - Regresión Logística
2 - Yolo (You Only Look Once) V8

Matriz de confusión

Verdadero positivo (VP): El valor real es positivo y la prueba predijo también que era positivo. Es decir, el aguacate tiene plaga y la prueba así lo demuestra.
Verdadero negativo (VN): El valor real es negativo y la prueba predijo también que el resultado era negativo. Es decir, el aguacate no tiene plaga y la prueba así lo demuestra.
Falso negativo (FN): El valor real es positivo, y la prueba predijo que el resultado es negativo. Es decir, el aguacate tiene plaga, pero la prueba dice de manera incorrecta que no lo está. Esto es lo que en estadística se conoce como error tipo II
Falso positivo (FP): El valor real es negativo, y la prueba predijo que el resultado es positivo. Es decir, el aguacate no tiene plaga, pero la prueba nos dice de manera incorrecta que si lo está. Esto es lo que en estadística se conoce como error tipo I

Exactitud (accuracy) = Cantidad de predicciones correctas.
(VP+VN)/(VP+FP+FN+VN)
Precisión = Porcentaje de casos positivos detectados. ***VP/(VP+FP)***
Recall = Proporción de casos positivos correctamente identificados. ***VP/(VP+FN)***

Fuente: Elaboración propia

6.3.1. Cultivo de aguacate Hass

El cultivo del aguacate HASS es cada vez más popular en la agricultura debido a sus características y beneficios. El aguacate HASS es una variedad de aguacate que se distingue por su adaptabilidad a diferentes condiciones climáticas y su alta productividad. Según estudios realizados por el Departamento Administrativo de Nacional de Estadística (Departamento Administrativo Nacional de Estadística (DANE), 2016), esta variedad se ha destacado por su resistencia a enfermedades y plagas comunes en el cultivo del aguacate.

Una de las ventajas de cultivar el aguacate Hass es su capacidad para adaptarse a diferentes climas, según los autores Reyes et al. (2022), esta variedad es apta para regiones con temperaturas que oscilan entre los 10 y 35 grados Celsius. Además, puede tolerar tanto condiciones secas como húmedas, lo que la convierte en una opción viable para diferentes áreas geográficas.

Cabe destacar, que otra de las características del aguacate Hass es su alta productividad, esta variedad puede alcanzar rendimientos superiores a los 20 kilogramos de fruta por árbol al año, esto la convierte en una opción rentable para los agricultores, ya que pueden obtener una mayor producción por unidad de superficie cultivada (Reyes et al., 2022).

Además de su adaptabilidad y productividad, el aguacate Hass presenta resistencia a enfermedades comunes en el cultivo del aguacate, según los autores Agapito et al. (2022),

esta variedad ha mostrado resistencia al hongo Phytophthora cinnamomi, causante de la pudrición de raíz, lo que reduce la necesidad de aplicar productos químicos para su control.

6.3.2. Plagas *Stenoma catenifer* y *heilipus lauri*

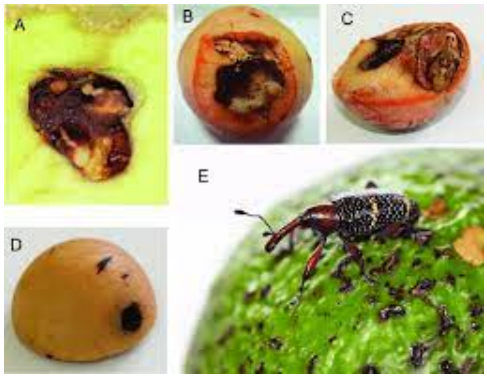
El cultivo del aguacate puede verse amenazado por diversas plagas que pueden afectar tanto las hojas, los frutos como las raíces de la planta, según el Dane (2016) las plagas más comunes son el ácaro del aguacate entre los que se encuentran el *Heilipus lauri* y el *Stenoma catenifer*, quienes provoca un daño en la apariencia de la planta y puede reducir su capacidad de fotosíntesis, para controlar esta plaga, se pueden emplear insecticidas específicos o realizar una poda adecuada de las hojas afectadas.

Para Carabalí et al. (2022), en el cultivo del aguacate Hass la *Stenoma catenifer* (ver figura 9) desarrolla su polilla al alimentarse de los frutos del aguacate, perforando galerías y causando daños considerables en la calidad de la cosecha. Asimismo la plaga del barrenador del hueso del aguacate *Heilipus* spp (ver figura 10), puede causar daños graves al cultivo, las larvas de este insecto se alimentan del interior del hueso del aguacate, lo que afecta el desarrollo adecuado del fruto (Carabalí et al., 2022).

Figura 9: Fotografía del insecto *Stenoma catenifer*

Fuente: Díaz et al. (2017)

Figura 10: Fotografía del insecto *Heilipus lauri*



Fuente: Palacios et al. (2011)

7. Metodología de la investigación

La metodología exploratoria aplicada es un enfoque de investigación que se utiliza para explorar y comprender fenómenos o problemas que son poco conocidos o no han sido investigados previamente de manera exhaustiva (Zafra, 2006). Se emplea cuando existe una falta de información o teorías bien establecidas sobre el tema de estudio, como lo son la identificación y monitoreo de plagas en el cultivo de aguacate Hass a través de metodologías MLOps.

De acuerdo con Zafra (2006), la metodología exploratoria aplicada es flexible y adaptable, permitiendo a los investigadores ajustar su enfoque a medida que se obtiene nueva información. Al principio del proceso, el investigador puede tener una comprensión limitada del fenómeno, pero a medida que avanza la investigación, se van generando nuevas preguntas e ideas que guían el proceso de recolección y análisis de datos.

La metodología exploratoria aplicada fue un enfoque flexible y creativo que permitió a los investigadores en ingeniería de software explorar y comprender fenómenos como el Big Data y las metodologías MLOps que son relativamente nuevos para generar conocimiento y a establecer las bases para investigaciones futuras más rigurosas y proporciona una base sólida para futuras investigaciones y puede contribuir a generar explicaciones y enfoques más avanzados en relación con la producción agrícola.

El estudio se llevó a cabo en cuatro huertos comerciales de aguacate Hass que se encuentran ubicados en los municipios de Sotará y Timbío del departamento del Cauca, donde existe un impacto considerable de las plagas *Stenoma catenifer* y *heilipus lauri* (Zapata et al., 2022). Esta situación de plagas analizada cuenta con una amplia base de datos fotográficas que fueron procesadas para esta investigación.

De allí que se orientó la investigación con las siguientes fases metodológicas que permitieron dar cumplimiento al proyecto:

Fase I: Para el objetivo de implementar técnicas de procesamiento de imágenes para extraer características relevantes y mejorar la capacidad del modelo de Machine Learning en el reconocimiento y detección de las plagas *Stenoma catenifer* y *heilipus lauri* en el cultivo de aguacate Hass a partir de imágenes capturadas en campo:

- Recopilar datos e imágenes que permitan una base de datos para reconocer las características propias de plantas con y sin plagas.
- Procesar los datos e imágenes de la plantación de café donde se desarrolla el proceso

de redimensionar las imágenes, normalizar los valores de los píxeles y elimina los posibles errores de imagen.

- Realizar técnicas de segmentación para aislar las regiones relevantes que contienen la planta de café.

Fase II: El objetivo es desarrollar y entrenar un modelo de Machine Learning utilizando técnicas apropiadas de preprocesamiento y selección de características, así como algoritmos de aprendizaje supervisado o no supervisado, para lograr una detección de las plagas *Stenoma catenifer* y *heilipus lauri* en el cultivo de aguacate Hass.

- Dividir el conjunto de datos preprocesado en un conjunto de entrenamiento y un conjunto de prueba.
- Realizar la implementación de los pipelines necesarios para la automatización de las tareas de encapsulamiento y despliegue.
- Utilizar el conjunto de entrenamiento para entrenar el modelo seleccionado donde el modelo asimilará el reconocimiento de los patrones asociados con y sin las plagas.

Fase III: En el objetivo desarrollar una metodología MLOps que permita la integración, automatización y monitoreo del modelo de Machine Learning diseñado para el reconocimiento y control de las plagas *Stenoma catenifer* y *heilipus lauri* en el cultivo de aguacate Hass:

- Seleccionar el modelo de Machine Learning para la detección de objetos en imágenes.
- Utilizar técnicas como el filtrado, detección de bordes, extracción de texturas y características específicas de las plagas que desees detectar en el modelo.

Fase IV: En el objetivo validar el uso de MLOps mediante despliegue en un ambiente controlado, con la capacidad de monitorear y mejorar continuamente el rendimiento del modelo.

- Evaluar el modelo utilizando el conjunto de prueba y ajustarlos a los parámetros y a la arquitectura del modelo según sea necesario para mejorar su rendimiento.
- Observar el rendimiento del conjunto de prueba para desplegarlo en tiempo real en aplicaciones de detección de plagas.

Todo este proceso se realizó aplicando los principios fundamentales de MLOps, siguiendo las medidas de seguridad informática requeridas para salvaguardar los datos personales, y además, cumpliendo con el conjunto de mejores prácticas en el desarrollo de software en general.

8. Resultados

8.1. Técnicas de procesamiento de imágenes para extraer características relevantes y mejorar la capacidad del modelo de Machine Learning en el reconocimiento y detección de las plagas *Stenoma catenifer* y *heilipus lauri* en el cultivo de aguacate Hass a partir de imágenes capturadas en campo.

8.1.1. Exploración de datos

El análisis exploratorio de datos partió de las 744 fotografías del cultivo de aguacate Hass con las cuales se reveló información valiosa sobre la distribución y características de las imágenes. Al examinar la distribución, se observaron dos parámetros en la información, el primero corresponde a las imágenes que no presentan el impacto de las plagas (50 imágenes) y el segundo las imágenes que evidencian el proceso de la plaga en el cultivo (694 imágenes). Esto sugiere la necesidad de técnicas de pre-procesamiento para normalizar la información antes de cualquier análisis.

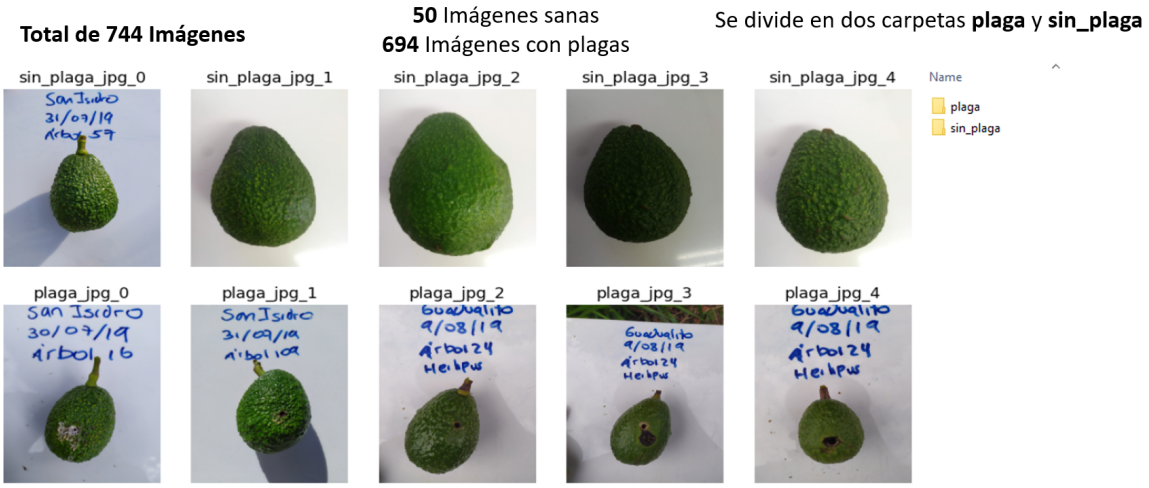
Por esta razón, se establecieron dos carpetas de información donde en una de ellas se consignan las fotografías que presentan las características de las plantas sana y en la otra carpeta se encuentran las fotografías que muestran las tipologías del impacto de las plagas.

En este sentido, se desarrollaron algoritmos de extracción de características, como por ejemplo el histograma de colores, descriptores de texturas y las características basadas en formas, con las cuales se representa la información relevante de las imágenes dentro del cultivo, para que estas características se conviertan en entradas de reconocimiento en el modelo de aprendizaje automático.

El código en Python está diseñado para la visualización de imágenes de cultivos de aguacate Hass[1], clasificadas en dos categorías: “Con Plaga” y “Sin Plaga”. Este programa, destinado a ser ejecutado en un entorno de Jupyter Notebook, su función principal es cargar y mostrar de manera detallada las dos tipologías definidas en las carpetas con imágenes de cada categoría, permitiendo una inspección visual clara de las características distintivas entre los cultivos afectados por plagas y aquellos que se consideran saludables.

[1]El código para la extracción de características de las imágenes se encuentra en GitHub aquí

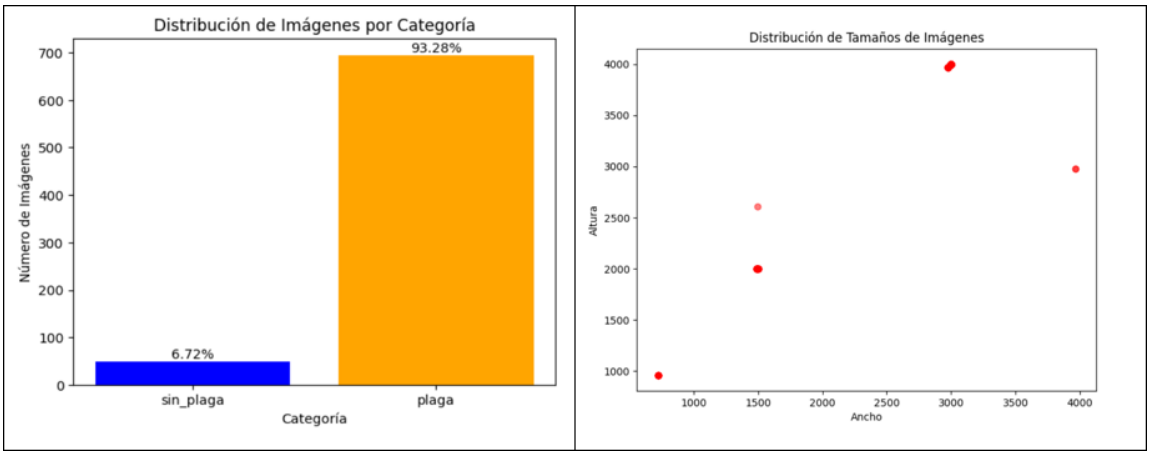
Figura 11: Exploración de datos



Fuente: Elaboración propia

Al realizar un análisis exploratorio de datos para comprender la distribución y características de las imágenes se puede señalar que, las imágenes sin plagas corresponden al 6,72 % y las imágenes con plagas el 93,28 % como se describe a continuación:

Figura 12: Distribución porcentual de las categorías de las imágenes



Fuente: Elaboración propia

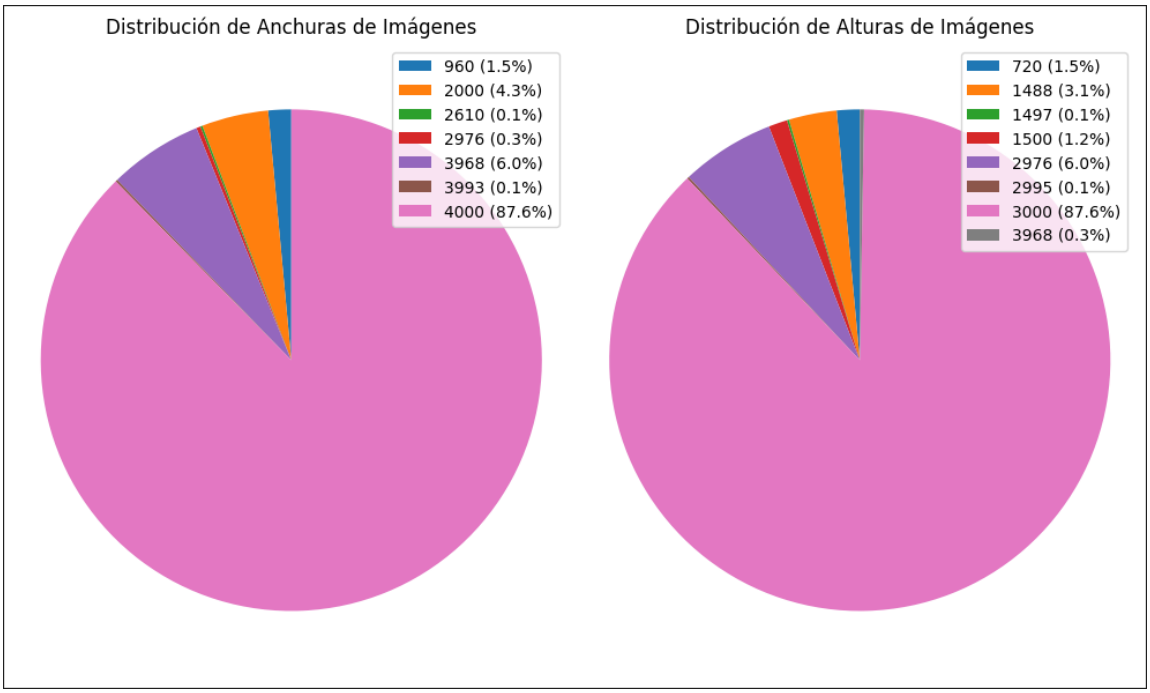
Las imágenes que generan mayor información corresponden a las que tienen plagas. Asimismo, la distribución de tamaño de imagen se observa que, identifican imágenes con un ancho de 4.000 píxeles por 3.000 píxeles de altura o por el contrario 3.000 de ancho por 4.000 de altura, donde se tomaron las fotografías tanto verticales como horizontales manteniendo esa rotación.

Por lo tanto, se generó el código correspondiente para generar tanto la distribución de imágenes por categoría como la distribución por tamaños, y por otro lado, se generó un código para saber cómo está distribuido por cada alto y ancho de las imágenes.

Como se puede ver en la figura 13 que el 87,6 % de las imágenes presentan una resolución de 4.000 píxeles por ancho y por el lado de las alturas más del 87,6 % tienen 3000 píxeles de alto [2].

[2]El código para el tratamiento de imágenes por categoría y alturas se encuentra en GitHub aquí

Figura 13: Distribución alto y ancho de las imágenes



Fuente: Elaboración propia

8.1.2. Remoción de fondo de imagen

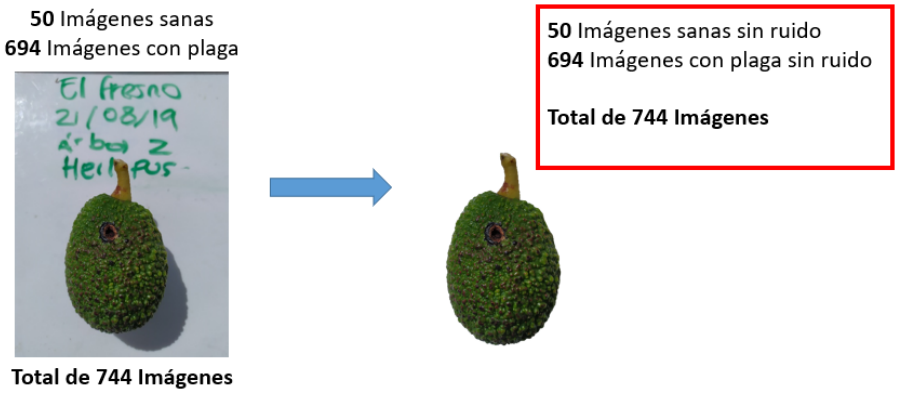
Quitar el fondo de cada una de las imágenes dejando solo el aguacate, enfocándose solamente en las imágenes que muestra signos de plagas, es esencial en el análisis de las imágenes para la detección y monitoreo de enfermedades. Al aislar el aguacate de su entorno, se simplifica la tarea de identificación de posibles plagas y enfermedades, permitiendo una evaluación más precisa y eficiente[3].

[3]El código de eliminación de fondo de las imágenes se encuentra en GitHub aquí

Este enfoque mejora la capacidad de los algoritmos de visión para identificar patrones sutiles asociados con enfermedades específicas, ya que elimina distracciones no relevantes y optimiza la focalización en las características de interés.

Todas las 744 imágenes presentaban un fondo que podría introducir ruido durante el análisis. Por lo tanto, se llevó a cabo el proceso para eliminar las imágenes del fondo, dejándolas libres de ruido. El resultado final consiste en la preservación únicamente de la parte central de la imagen, como se aprecia en la imagen anterior con el código implementado realiza la tarea de eliminar el fondo y retener solo la imagen del aguacate.

Figura 14: Proceso de remoción de fondo de las imágenes



Fuente: Elaboración propia

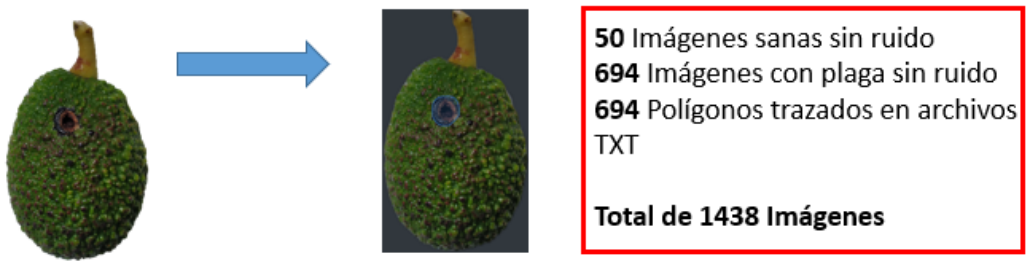
La efectividad de este código asegura que las imágenes estén listas para ser utilizadas en análisis más detallados y precisos relacionados con el aguacate y cualquier aspecto asociado, sin la interferencia de fondos indeseados.

8.1.3. Procesamiento de imágenes

En el procesamiento de la segmentación de imágenes como se viene señalando en los anteriores momentos, permite resaltar la zona en donde se encuentran las características propias de la plaga. El proceso tiene como fin facilitarle al algoritmo de Machine learning interpretar las imágenes. Este enfoque se integra en la fase de pre-procesamiento para resaltar la zona afectada por la plaga, simplificando la tarea del algoritmo y mejorando su capacidad de interpretación.

A cada una de estas 694 imágenes que presentan afectaciones de la plaga se trazó un polígono para saber en dónde estaba ubicada la plaga, este polígono genera unas coordenadas que luego van a ser interpretadas por uno de los algoritmos usados para el proyecto.

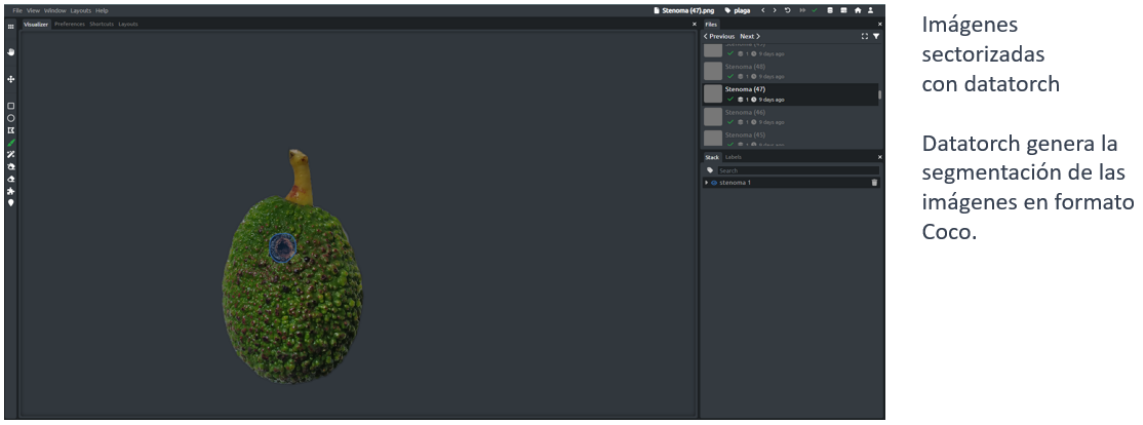
Figura 15: Proceso de segmentación de las imágenes



Fuente: Elaboración propia

Posteriormente se utilizó la aplicación DataTorch para hacer el trazado de los polígonos, en donde se coge una por una de las imágenes y se empiezan a trazar cada una de las áreas en que se expresa la plaga, teniendo en cuenta que se hizo por cada una de las 694 imágenes, se obtuvo un número equivalente de archivos en formato TXT con los polígonos.

Figura 16: Proceso de segmentación de las imágenes con DataTorch



Fuente: Elaboración propia

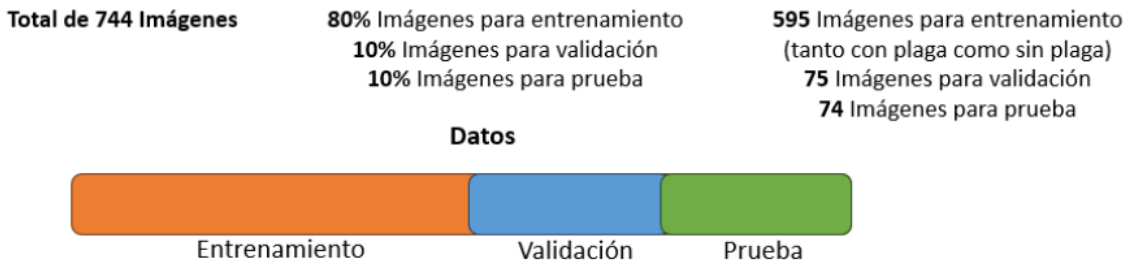
8.1.4. Conjunto de entrenamiento, validación y prueba

El proceso de entrenamiento en Machine Learning implica enseñar a un modelo a reconocer patrones y tomar decisiones basadas en datos. Para lograr esto, es necesario dividir el conjunto de datos normalmente en tres partes: la primera, conocida como conjunto de entrenamiento, debe ser el más grande de todos los conjuntos y se utiliza para alimentar al modelo con ejemplos que le permitan aprender patrones y relaciones entre variables. La segunda parte es el conjunto de validación busca definir cual es la mejor cantidad de estimadores o ajustar los parámetros del modelo para que pueda generalizar y hacer predicciones precisas sobre datos no vistos anteriormente.

La tercera parte es el conjunto de validación, que actúa como una prueba independiente del rendimiento del modelo. Al utilizar datos que el modelo no ha visto durante el entrenamiento, se puede evaluar su capacidad y prevenir problemas como el sobreajuste, donde el modelo memoriza los datos de entrenamiento en lugar de aprender patrones generales.

Dividir el conjunto de datos de las imágenes en conjuntos de entrenamiento, validación y prueba como se describe a continuación:

Figura 17: Proceso de entrenamiento, validación y prueba



Fuente: Elaboración propia

Esta división en conjuntos de entrenamiento, validación y prueba es crucial para garantizar que el modelo pueda adaptarse a nuevos datos, lo que mejora su capacidad predictiva y su utilidad en situaciones del mundo real.

No existe una respuesta clara sobre el tamaño ideal en la división de los conjuntos. Todo a la final depende a la cantidad de información con la que se cuenta y no como una regla establecida. Para esta investigación se tiene un conjunto de datos moderado, es posible manejar un entrenamiento partiendo de una proporción de 80 %, 10 % y 10 %; eso quiere

decir que, del total de las imágenes 595 irán para el entrenamiento, 75 para validación y 74 para prueba, eso incluye la información con plaga como sin plaga, es decir se presenta toda la información conjunta.

8.2. Modelo de Machine Learning utilizando técnicas apropiadas de pre-procesamiento y selección de características, así como algoritmos de aprendizaje supervisado o no supervisado, para lograr una detección de las plagas *Stenoma catenifer* y *heilipus lauri* en el cultivo de aguacate Hass

8.2.1. Selección de algoritmos

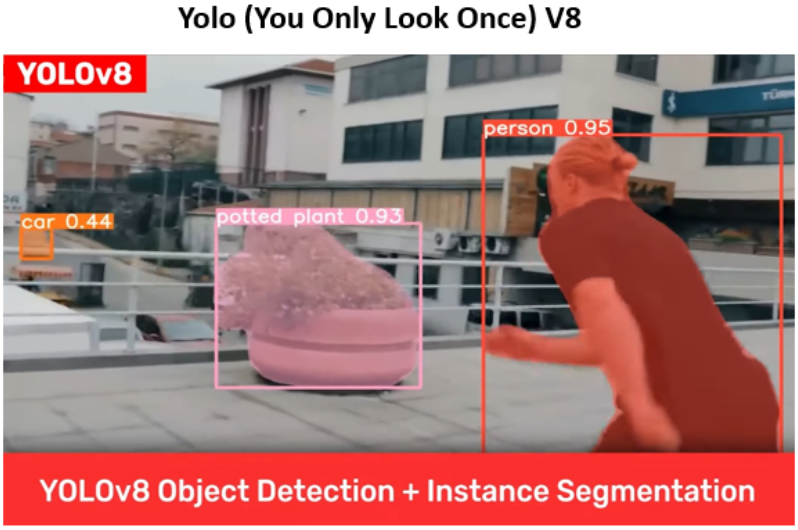
Para el proyecto se seleccionan algoritmos de aprendizaje supervisado o no supervisado adecuados para el problema de detección de plagas. Se seleccionó el algoritmo de la regresión logística, que es un algoritmo de clasificación que ayuda a determinar la probabilidad de que una imagen tenga o no tenga un objeto en este caso una plaga. El proceso corresponde a clasificar de manera binaria, entre cero y uno, donde cero es que no tiene plaga y uno donde sí tiene plaga.

También, se seleccionó el algoritmo de Yolo V8, que sus siglas en ingles significan You Only Look Once. Este algoritmo emplea redes neuronales convolucionales, estamos hablando de Deep learning, para facilitar e interpretar las imágenes. Este enfoque se integra en la fase de pre-procesamiento que se encarga de resaltar la zona afectada por la plaga, simplificando la tarea del algoritmo de Machine Learning y mejorando su capacidad de interpretación.

Estas redes neuronales convolucionales pueden aprender patrones y características complejas, permitiendo la identificación precisa de plagas en imágenes agrícolas. Además, el Deep Learning facilita la interpretación de las imágenes al automatizar la extracción de características, eliminando la necesidad de diseñar manualmente algoritmos específicos.

Para Schmidhuber (2015), la capacidad del Deep Learning para adaptarse a patrones variados en imágenes, aprende automáticamente, agrupan gran cantidad de información, predicen comportamiento, los cuales contribuyen significativamente a mejorar la eficiencia y precisión en la detección de plagas. Su habilidad para generalizar a partir de conjuntos de datos grandes y diversos lo convierte en una herramienta poderosa para interpretar imágenes, proporcionando una base sólida para aplicaciones prácticas en la identificación y monitoreo de plagas, lo que en última instancia beneficia la gestión y la seguridad del proceso del cultivo de aguacate Hass.

Figura 18: Acción del algoritmo Yolo V8



Fuente: Bastian (2023)

En la imagen se puede ver que se clasifica un objeto con un 95 % de probabilidad de que sea una persona, un 93 % de establecer el objeto como "planta" y el otro objeto que se ve corresponde a un 44 % que el objeto sea un "carro". El algoritmo Yolo V8 incluso puede ser usado en video, permitiendo en este caso detectar objetos incluso en tiempo real.

La selección del algoritmo de regresión logística se da porque es uno de los estándares para trabajar clasificación de imágenes en la industria. Por otro lado, el algoritmo Yolo es un algoritmo de Deep learning, para detectar objetos en imágenes esto se hace para determinar con una probabilidad de qué objetos se tienen en una imagen o imágenes e incluso en videos, siendo posible clasificar cada objeto que se encuentra en una imagen o

video.

8.2.2. Pre-procesamiento adicional

El pre-procesamiento adicional correspondió a normalizar el tamaño de las imágenes. Esta acción fue necesaria debido a que permite trabajar con modelos de aprendizaje automático, como la regresión logística y Yolo. Esto se hace con el fin de reducir la complejidad computacional porque al tener imágenes de una resolución muy alta se va a requerir más procesamiento de cómputo para empezar a analizar pixel por pixel; a su vez ayuda a tener una consistencia en la entrada del modelo, porque todos los valores o variables del entrenamiento tendrán las mismas dimensiones y así se evitan problemas relacionados con la variabilidad que se puedan tener en el tamaño de las imágenes.

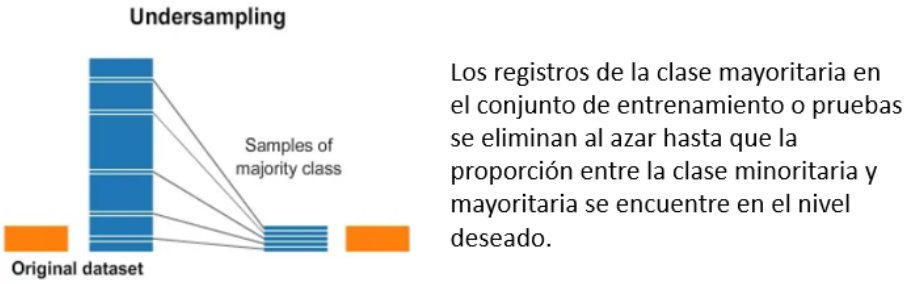
También habrá una normalización del modelo con datos nuevos, es decir que, si es necesario ingresar datos nuevos al modelo, pasarán por esta normalización cada uno de los datos en una resolución determinada.

Para la normalización de las dimensiones se definió un tamaño fijo de 640 píxeles, tanto de ancho como de alto, debido a que Yolo en su versión 8 maneja valores por defecto en esta resolución. Por esta razón, se mantuvo la misma resolución en los dos modelos seleccionados, tanto para el Yolo, que ya tiene por defecto este valor, como en la regresión logística.

8.2.3. Implementación del modelo de regresión logística

Para realizar la implementación del modelo de regresión logística, se realizó un balanceo de la información debido a la disparidad entre los datos con plagas (694), de los datos sanos (50). De allí que, se implementó un balanceo de información hacia abajo, es decir que la clase mayoritaria se tiene que igualar a su clase minoritaria para poder hacer comparaciones y hacer un entrenamiento más proporcional con la información.

Figura 19: Proceso de normalización de imágenes



Fuente: Elaboración propia

Se realizó un submuestreo con validación cruzada (cross-validated subsampling). En este enfoque, se realiza repetidamente un submuestreo de la clase mayoritaria para equilibrar las clases y se evalúa el modelo en cada iteración utilizando la clase minoritaria. Luego, se promedian los resultados obtenidos en todas las iteraciones.

Este enfoque es especialmente útil cuando se trata con conjuntos de datos desequilibrados, donde una clase tiene muchos más datos (imágenes) que la otra. Al realizar varias iteraciones y calcular promedios, se busca obtener una evaluación más robusta del rendimiento del modelo, ya que cada iteración puede tener una muestra diferente de la clase mayoritaria[4].

En el desarrollo de los scripts para implementar el modelo de regresión logística se realizaron los siguientes pasos:

1. Se repite el proceso de entrenamiento y evaluación 10 veces.
2. Se realiza el balanceo de los datos en la clase mayoritaria (con plagas) sacando subconjuntos aleatorios de 50 imágenes del conjunto total (694).
3. Se realiza el balanceo de los datos en la clase minoritaria (sin plagas) sacando subconjuntos aleatorios de 50 imágenes del conjunto total (50), así estos sean repetidos.
4. Se combinan los datos de 'sin plaga' y datos balanceados de 'plaga' para tener un subconjunto de 100 imágenes totales 50 sin plaga y 50 con plaga.

[4]El código desarrollado para el modelo de regresión logística se encuentra aquí.

5. Se dividen los datos en conjuntos de entrenamiento, validación y prueba, es decir, 80 imágenes para entrenamiento, 10 imágenes para validación y 10 imágenes para prueba.

6. Se inicializa el modelo de regresión logística con la dependencia sklearn.

7. Se pasa a entrenar el modelo con los subconjuntos de entrenamiento.

8. Se realizan las predicciones en los subconjuntos de prueba.

9. Se calcula la matriz de confusión y se almacena por cada subconjunto.

10. Se calcula el promedio de las exactitudes (accuracy) obtenidas desde las matrices de confusión.

11. Se calcula el promedio de las precisiones obtenidas desde las matrices de confusión.

12. Se calcula el promedio de las sensibilidades (recall) obtenidas desde las matrices de confusión.

8.2.4. Implementación del modelo Yolo

Para el desarrollo de los scripts que implementan el modelo Yolo V8 se realizaron los siguientes pasos:

1. Se establece que se van a manejar por convención del modelo Yolo, tres particiones (entrenamiento – train, validación – validation y prueba - test) (ver figura 21).

2. Se dividen los datos en las particiones de entrenamiento, validación y prueba en 80 %, 10 %, 10 % de 744 imágenes, es decir, 595 imágenes de entrenamiento, 75 imágenes para prueba y 74 imágenes para validación.

3. Se realiza la transformación del formato Coco al formato Yolo para cada polígono (ver figura 20)[5].

4. Se escribe en un archivo de texto la transformación del formato resultante.

[5]Se realiza la transformación del formato Coco para cada polígono al formato Yolo, porque en el objetivo uno la segmentación de imágenes se realizó con la herramienta DataTorch la cual utiliza el formato Coco para generar las anotaciones correspondientes en las imágenes, pero el algoritmo Yolo solo funciona con el formato Yolo.

5. Se copia el archivo en la partición correspondiente (entrenamiento, validación o prueba).

6. Se importa la dependencia para trabajar con Yolo (ultralytics).

7. Se establece el modelo a Yolo small [6].

8. Se establecen 10 épocas para correr el modelo.

9. Se obtienen los resultados arrojados por el algoritmo (exactitud - accuracy, predicción - prediction y sensibilidad - recall).

Figura 20: Código transformación de formato Coco a formato Yolo

```
# Transforma de formato Coco a formato Yolo
def coco_to_yolo(x, y, w, h, width, height):
    return [((2*x + w)/(2*width)), ((2*y + h)/(2*height)), w/width, h/height]
```

Fuente: Elaboración propia

[6] Yolo en su versión 8 salió en enero del año de 2023 y presenta diferentes modelos para diferentes tareas teniendo en cuenta las necesidades para detectar o segmentar información. Para esta investigación se manejó el modelo correspondiente a la detección de objetos. Para saber más sobre los diferentes tipos de modelos, se puede encontrar información detallada en el enlace tipos de modelos Yolo.

Figura 21: Código convención del modelo Yolo

```
datasets
    - images
        train
            - 1.png
            - 2.png
            ...
            - 595.png
        validation
            - 1.png
            - 2.png
            ...
            - 75.png
        test
            - 1.png
            - 2.png
            ...
            - 74.png
    - labels
        train
            - 1.txt
            - 2.txt
            ...
            - 595.txt
        validation
            - 1.txt
            - 2.txt
            ...
            - 75.txt
        test
            - 1.txt
            - 2.txt
            ...
            - 74.txt
```

Fuente: Elaboración propia

El concepto de “épocas” se refiere a la cantidad de veces que todo el conjunto de datos se ha pasado hacia adelante y hacia atrás a través de la red neuronal durante el proceso de entrenamiento. Cada época representa una iteración completa a través de todo el conjunto de datos. Manejar múltiples épocas durante el entrenamiento puede tener varios beneficios:

Aprendizaje más profundo:

Las redes neuronales, incluidas las arquitecturas complejas como YOLO, pueden aprender patrones más profundos y sutiles a medida que se exponen repetidamente a los datos. Las épocas adicionales permiten que el modelo ajuste sus pesos en consecuencia y mejore su capacidad para generalizar a patrones más complejos.

Mejora de la generalización:

El entrenamiento con más épocas puede ayudar a mejorar la capacidad del modelo para generalizar a datos no vistos. Esto es particularmente útil para evitar el sobreajuste, donde el modelo memoriza los datos de entrenamiento en lugar de aprender patrones, ubicaciones o lugares generales.

Ajuste fino de parámetros:

Las épocas adicionales permiten ajustar finamente los parámetros del modelo. A medida que el modelo ve los datos varias veces, puede hacer ajustes más precisos en los pesos para mejorar el rendimiento en el conjunto de entrenamiento.

Mejora de métricas de rendimiento:

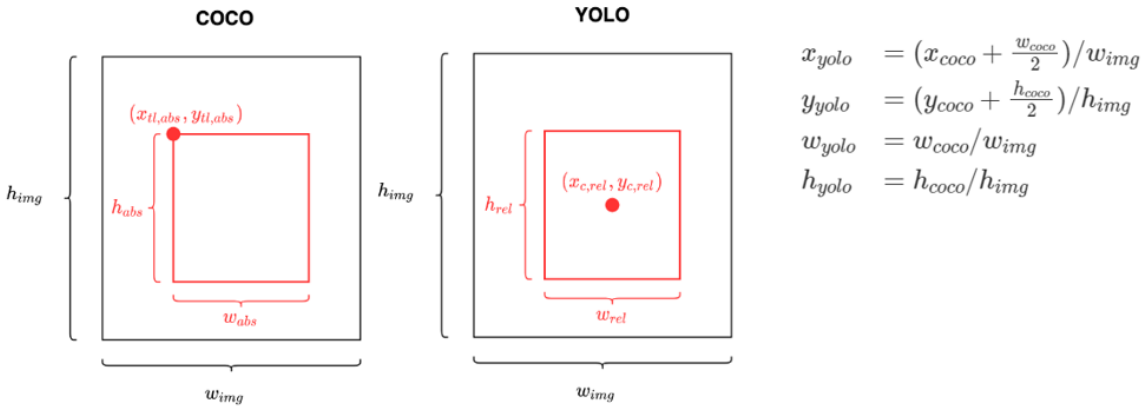
En muchos casos, las métricas de rendimiento (como precisión, recall, F1-score, etc.) pueden mejorar con un entrenamiento más prolongado. Las métricas estabilizan y pueden alcanzar su mejor rendimiento después de varias épocas.

Manejar 10 épocas podría ser mejor que manejar pocas épocas, esto puede depender de la complejidad del problema, del tamaño del conjunto de datos y de la arquitectura específica del modelo. Más épocas generalmente significan más oportunidades para que el modelo aprenda y mejore, siempre y cuando se evite el sobreajuste. Sin embargo, no siempre es necesario o beneficioso utilizar un gran número de épocas, ya que podría aumentar los costos computacionales sin proporcionar mejoras significativas en el rendimiento del modelo.

Dentro del código anterior se especifica el cambio del formato Coco a Yolo partiendo de las categorías establecidas con la plaga y si no tiene plaga no se denota. Como se describe en el código las dimensiones de ancho y alto para las anotaciones, donde dice que para una imagen dentro del polígono se encuentra la plaga denotada en los cuatro lados expresados en cuatro valores (x, y, w, h) que son la coordenada X y Y en donde se ubica el extremo superior del polígono a marcar, como el ancho y alto en donde se encuentra la placa.

Estos parámetros de la coordenada X y Y del polígono en formato Yolo, se posicionan en el centro de la imagen donde está ubicado el objeto, en este caso se mueve a su centro, y esto se hace con una fórmula matemática como se observa en la siguiente imagen:

Figura 22: Proceso de cambio de Coco a Yolo

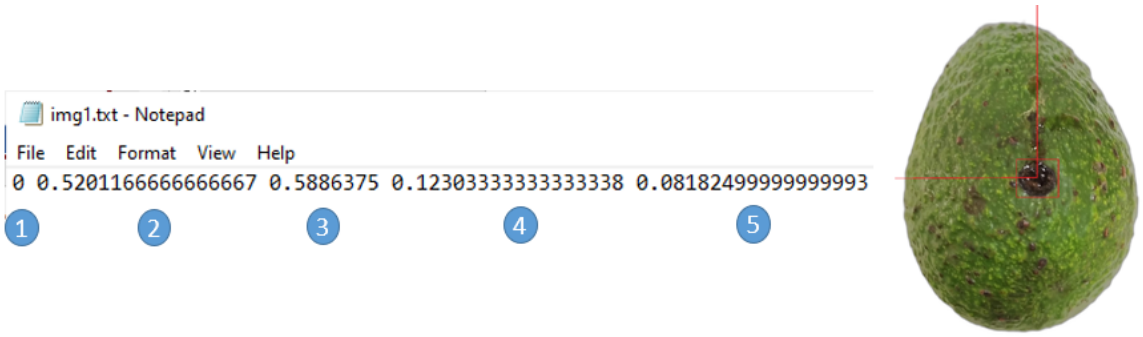


Fuente: Elaboración propia

Después de realizar la transformación, el procedimiento implica identificar y organizar la información (ver figura 23) de la siguiente manera:

- La primera columna especifica la clase a utilizar, en este caso, "plaga" [7].
- La segunda y tercera columna indican las coordenadas del punto medio de un cuadro delimitador, correspondiente a la posición central del objeto a identificar, en este caso, la plaga.
- La cuarta y quinta columna contienen las dimensiones del cuadro delimitador, representando respectivamente el ancho y alto del objeto en cuestión. Para clarificar: la cuarta columna representa el ancho, mientras que la quinta columna refleja el alto. Este proceso se aplica para facilitar la identificación y delimitación de la plaga.

Figura 23: Identificación del polígono cambiado de Coco a Yolo



Fuente: Elaboración propia

Este proceso corresponde a la conversión del formato Coco a Yolo. En otras palabras, se especifica una clase, en este caso, "plaga" designada como cero. Los dos primeros valores indican las coordenadas del punto medio del cuadro que se va a delimitar, mientras que los dos valores restantes representan el alto y el ancho de la imagen en cuestión. Este código pertenece al modelo Yolo V8, donde se lleva a cabo la distribución de las imágenes en las dos particiones correspondientes de entrenamiento, validación y prueba.

[7] La acción matemática es descrita en Rehman (2022) especificando los procedimientos para el cambio de Coco a Yolo.

En resumen, en el proceso de implementación del modelo Yolo[8], se llevan a cabo las siguientes etapas:

- **Preprocesamiento de imágenes:** Se aplican diferentes técnicas de preprocesamiento a las imágenes antes de utilizarlas en el modelo Yolo.
- **Carga del modelo:** Se carga el modelo Yolo V8 en su versión small, especificando las épocas de entrenamiento, dentro del parámetro de la función, en este caso 10. Es importante tener en cuenta que un número mayor de épocas aumenta el costo computacional para obtener un resultado.

8.2.5. Evaluación de los modelos

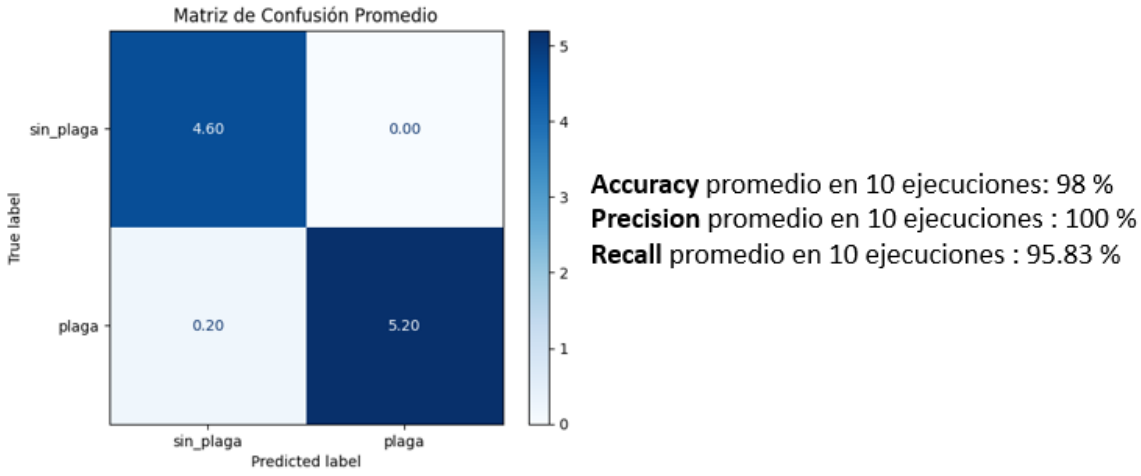
Después de haber entrenado los modelos y pasar a predecir si hay plaga o no hay plaga que se tenga del conjunto de validación, es decir identificar qué tanto aprendió y cuál fue el porcentaje de aprendizaje que se tuvo, para ambos casos se manejó una matriz de confusión donde se tienen cuatro lados: falso positivo, falso negativo, verdadero negativo, verdadero positivo (ver figura 8).

Una interpretación efectiva de la matriz de confusión en un modelo Yolo es esencial para entender cómo se comporta en términos de identificación y ubicación de objetos en las imágenes de entrada. Al analizar los elementos de la matriz, los científicos de datos pueden ajustar el modelo para mejorar su rendimiento y garantizar una detección precisa de objetos en diferentes escenarios y condiciones.

Para la regresión logística al tener 10 iteraciones se calculó el promedio de esas matrices de confusión, es decir que cada vez que se pasaba el modelo en este subconjunto de datos de 100 imágenes se estaba calculando una matriz de confusión, se van guardando cada una de esas matrices y al final se promedian por componente, para tener los resultados de métricas, acurracy, precision y recall [9] en el conjunto de validación como se describe en la figura 24.

[8] El código correspondiente a la implementación del modelo Yolo se encuentra disponible aquí.
[9] El código de evaluación del modelo utilizando regresión logística se encuentra aquí.

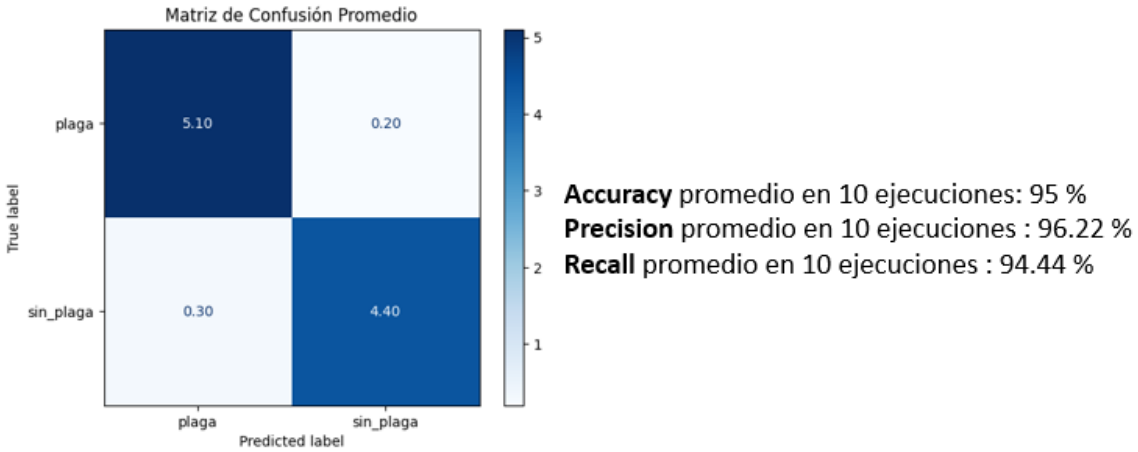
Figura 24: Resultado promedio de la matriz de confusión del modelo de regresión en el conjunto de validación



Fuente: Elaboración propia

El objetivo es determinar la cantidad de datos que estaban etiquetados como afectados por una plaga durante la validación del modelo. En este escenario, se utiliza un subconjunto de 100 imágenes para llevar a cabo la validación, y se espera que dicho subconjunto represente un porcentaje específico del total correspondiente al 10 % para la validación. Además, se realiza una evaluación del modelo con un conjunto de pruebas correspondiente a otro 10 %, y así contrastar los resultados en los dos conjuntos (ver figura 25).

Figura 25: Resultado promedio de la matriz de confusión del modelo de regresión en el conjunto de prueba



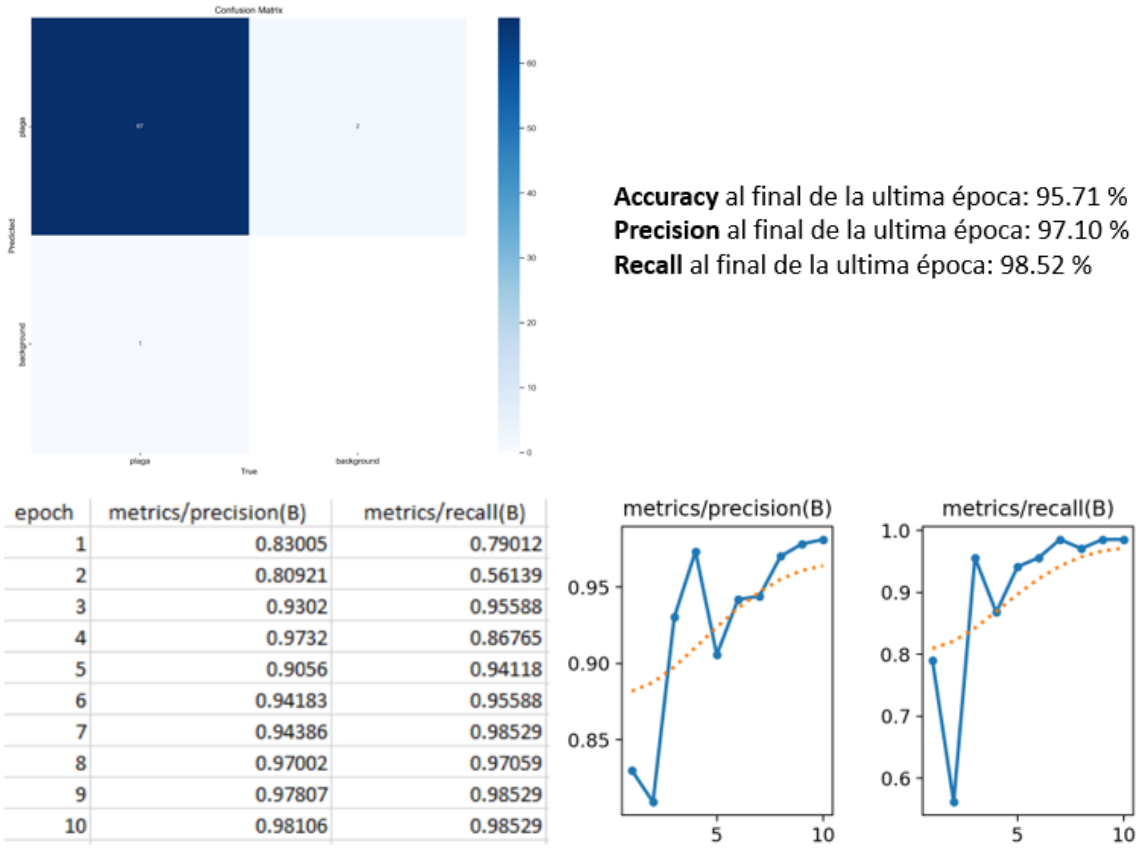
Fuente: Elaboración propia

Al hablar de precisión se especifica la cantidad de casos positivos detectados, para este caso fueron realmente positivos detectados (precisión) en un 100 %, una sensibilidad que corresponde a la proporción de los casos positivos correctamente identificados que fue de un 95.83 %. Respecto a la exactitud, se registró un valor del 98 %, lo que indica el porcentaje de predicciones correctas realizadas. Todos estos resultados fueron obtenidos para el conjunto de validación.

Por otro lado, al evaluar el modelo con el conjunto de prueba, la información varía ligeramente. Es importante destacar que las variaciones observadas en la exactitud, la precisión y la sensibilidad son de 3 % para la exactitud, 3.78 % para la precisión y 1.39 % para la sensibilidad. Estas fluctuaciones, aunque pueden existir cambios sutiles, la diferencia no es significativa.

En Yolo V8, el enfoque se centra únicamente en clasificar las imágenes que efectivamente presentan plagas. Cuando se procesa la información que carece de plagas, el algoritmo simplemente ignora o descarta dicha información, ya que el objetivo principal consiste en identificar el conjunto de prueba que contiene o no una subimagen del polígono o recuadro que pueda representar una plaga. En otras palabras, el propósito es determinar si dentro de la representación visual de un aguacate existe o no la presencia de una plaga.

Figura 26: Resultado del modelo Yolo V8 en el conjunto de validación



epoch	metrics/precision(B)	metrics/recall(B)
1	0.83005	0.79012
2	0.80921	0.56139
3	0.9302	0.95588
4	0.9732	0.86765
5	0.9056	0.94118
6	0.94183	0.95588
7	0.94386	0.98529
8	0.97002	0.97059
9	0.97807	0.98529
10	0.98106	0.98529

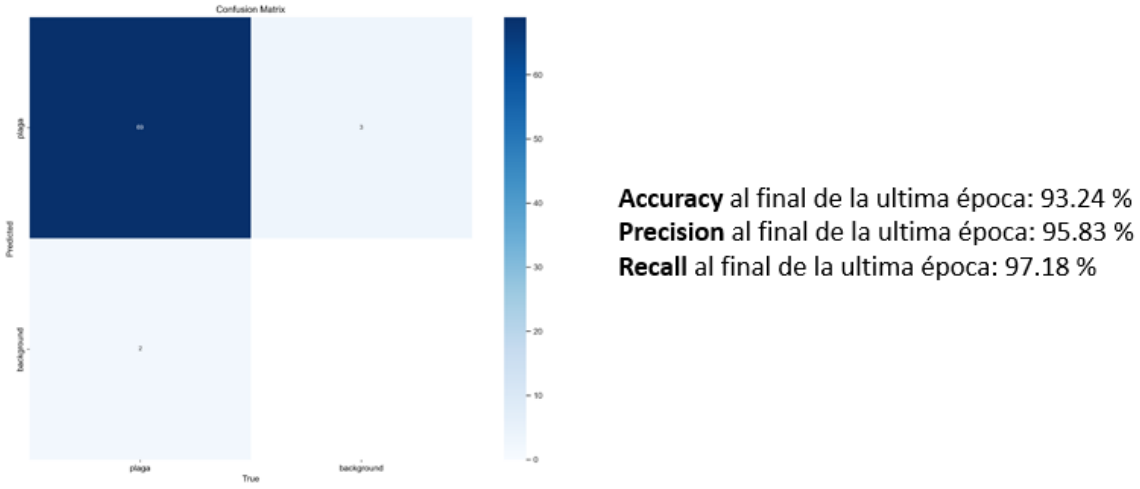
Fuente: Elaboración propia

Después de realizar la evaluación en el conjunto de validación, observamos que la sensibilidad alcanzó un 98.52 %, indicando que el 98.52 % de los casos positivos fueron correctamente identificados respecto al total de imágenes. La precisión, que mide la proporción de casos positivos detectados entre las predicciones positivas, se situó en un sólido 97.10 %.

Por otro lado, la exactitud, que representa la cantidad general de predicciones correctas, se registró en un 95.71 %. Esto implica que, aunque se logró una alta precisión, hubo algunas predicciones incorrectas, especialmente en casos específicos evaluados por el modelo Yolo.

Por otro lado, al evaluar el modelo con el conjunto de prueba (ver figura 27), la información varía ligeramente. Es importante destacar que las variaciones observadas en la exactitud, la precisión y la sensibilidad son de 2.47 % para la exactitud, 1.27 % para la precisión y 1.34 % para la sensibilidad. Estas fluctuaciones, aunque pueden existir cambios sutiles, la diferencia no es significativa.

Figura 27: Resultado del modelo Yolo V8 en el conjunto de prueba



Fuente: Elaboración propia

Es importante destacar que las evaluaciones se llevaron a cabo con imágenes seleccionadas de manera aleatoria que tenían y carecían de plagas, es decir, imágenes consideradas como “sanas”. Además, es relevante señalar que la matriz de confusión se generó de manera automática después de ejecutado el algoritmo, proporcionando una representación visual detallada de los resultados de las evaluaciones.

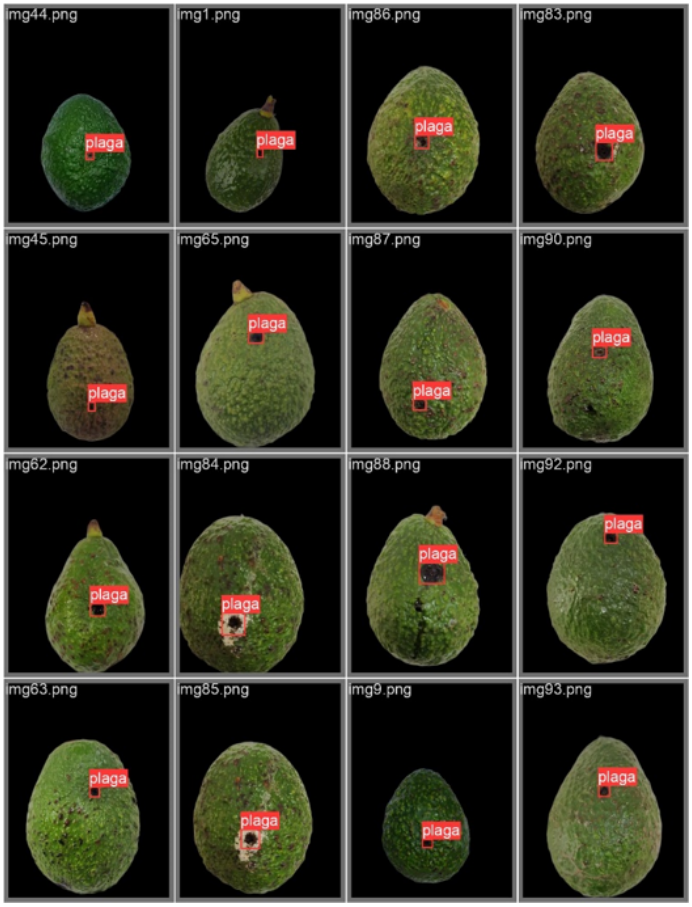
Yolo genera imágenes y proporciona métricas clave como precisión, sensibilidad y curvas de aprendizaje por época. Al analizar las estadísticas por época, se observa una progresión significativa en la precisión a lo largo del entrenamiento. En las primeras épocas, la

precisión fue estándar, alcanzando el 83 % en la primera y baja su rendimiento al 80.9 % en la segunda. Sin embargo, a partir de la tercera época, se observa un aumento marcado llegando al 93.02 % y alcanzando un impresionante 97.32 % en la época 4. Las épocas 8 y 9 y 10 mantienen rendimientos destacados, con la época 10 mostrando una precisión aún mayor que la 9 (ver figura 26).

El análisis detallado revela que, a pesar de obtener altos valores, la sensibilidad se mantuvo en la época 10, este análisis subraya la importancia de equilibrar la precisión y el recall al ajustar el número de épocas. La representación gráfica destaca cómo el modelo evoluciona en cada época, evidenciando mejoras notables a partir de la tercera época. La tabla de la figura 26 complementa visualmente la información proporcionando valores precisos de precisión y recall para cada época. Este análisis proporciona valiosas percepciones para optimizar futuros entrenamientos del modelo Yolo.

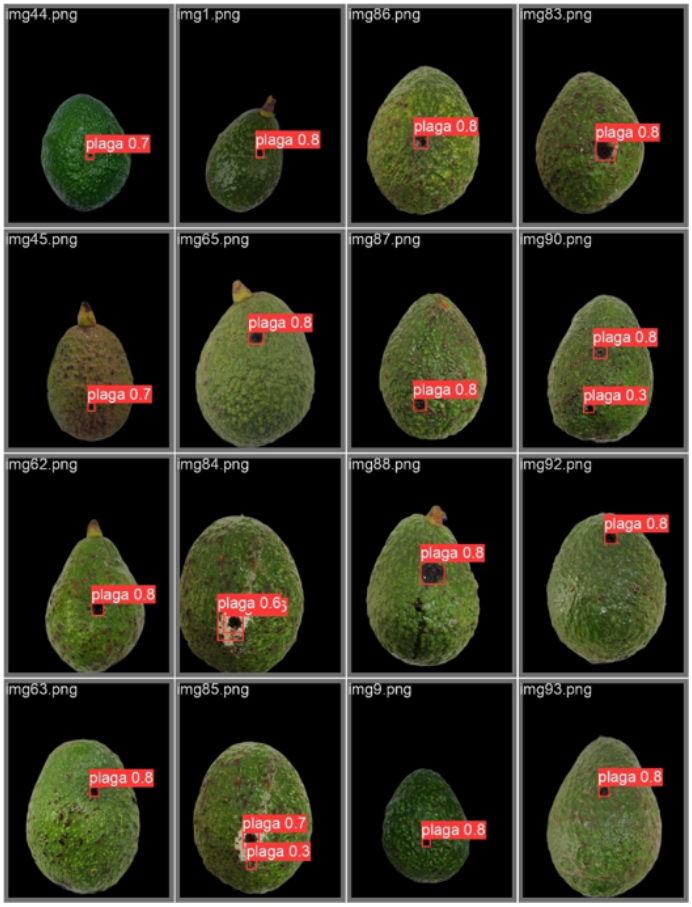
En los resultados proporcionados por el modelo (ver figura 28), se presentan imágenes que fueron clasificadas como casos de plagas. Al analizar estas imágenes, es evidente que el modelo ha identificado correctamente la presencia de plagas, ya que estas son las mismas imágenes que se proporcionaron individualmente al algoritmo para su evaluación. En el conjunto de prueba generado, las representaciones visuales de plagas en las imágenes que se muestran en la figura 29. Estas imágenes son el resultado de la aplicación del algoritmo, demostrando su capacidad para discernir y clasificar eficazmente las instancias de plagas en base a los patrones aprendidos durante el entrenamiento.

Figura 28: Visualización de la evaluación realizada con Yolo V8



Fuente: Elaboración propia

Figura 29: Resultado de la evaluación realizada con Yolo V8



Fuente: Elaboración propia

Estos resultados resaltan los aportes del modelo Yolo en la identificación de plagas, respaldando las predicciones al compararlas con la evaluación visual que se realizó de manera automática. La capacidad del algoritmo para generalizar y reconocer patrones de plagas en nuevas imágenes es un indicativo positivo de su rendimiento y validez en aplicaciones prácticas relacionadas con la detección y clasificación de problemas específicos, como la presencia de plagas en cultivos.

Al evaluar el desempeño del algoritmo Yolo V8 en la validación de imágenes de plagas, se destaca la efectividad de la detección. En el conjunto de prueba, el algoritmo demuestra su capacidad para identificar plagas en imágenes específicas sin conocer previamente la ubicación de los polígonos anotados. La separación del conjunto de datos en entrenamiento y validación es esencial para el entrenamiento, y en específico el conjunto de prueba, con un 10 % de las imágenes, se utiliza para evaluar el rendimiento del modelo.

Las imágenes seleccionadas para la prueba no siguen un orden específico y son elegidas al azar por el algoritmo. Al analizar las imágenes detectadas, el algoritmo asigna porcentajes de confianza para cada región identificada como plaga, facilitando la interpretación de la certeza de las predicciones. Se observa que el algoritmo genera predicciones de plagas con confianza en promedio del 70 %.

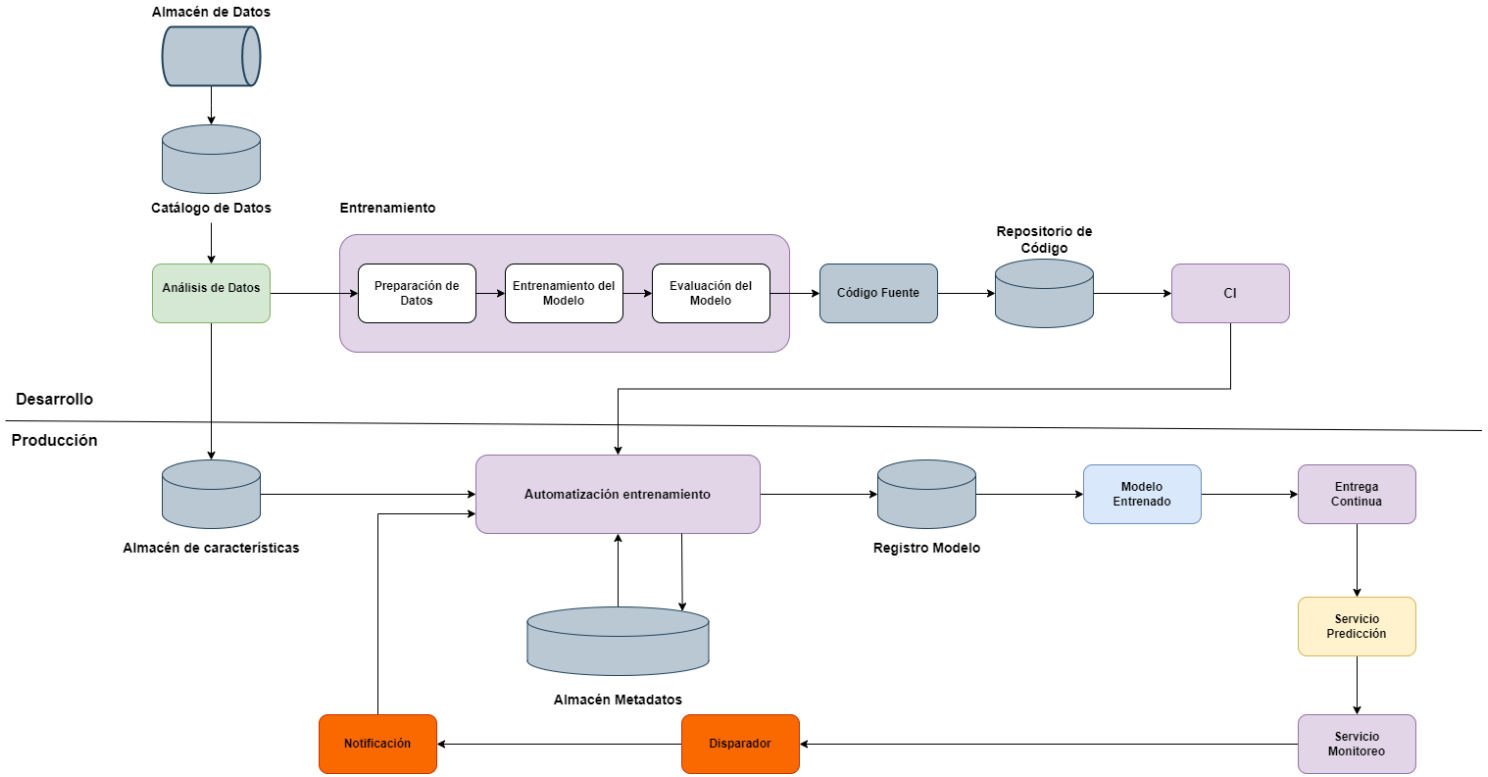
La potencial aplicación práctica del modelo se ilustra al considerar su capacidad para clasificar imágenes en tiempo real, incluso mediante el análisis de videos. Esta capacidad permite no solo la detección de plagas, sino también la visualización gráfica instantánea de las áreas afectadas. La versatilidad del modelo Yolo V8 corresponde al poder aplicarse en situaciones dinámicas y en tiempo real.

8.3. Metodología MLOps permitiendo la integración, automatización y monitoreo del modelo de Machine Learning diseñado para el reconocimiento y control de las plagas *Stenoma catenifer* y *heilipus lauri* en el cultivo de aguacate Hass

8.3.1. Creación del flujo de trabajo con MLOps

Para la implementación del modelo MLOps del proyecto se encuentra detallada en un diagrama de flujo (ver figura 30).

Figura 30: Diagrama de flujo MLOps aplicado al proyecto



Fuente: Elaboración propia

Se observa un proceso que se divide en dos etapas fundamentales: la etapa de desarrollo y la etapa de producción. En la fase de desarrollo, que generalmente se alinea con los objetivos uno y dos, se enfoca en el preprocesamiento de datos, abordando tareas como normalización, escalado de imágenes o redimensionamiento a dimensiones específicas.

Durante esta etapa, la limpieza y preparación adecuada del almacén de datos es realizada en el preprocesamiento durante la fase de desarrollo, donde se normalizan, escalan o redimensionan las imágenes. Además, se inicia la definición del entrenamiento y evaluación de las imágenes.

Con la implementación del MLOps, se introduce el versionado del código, permitiendo el control de cambios y la trazabilidad de la información. Un aspecto destacado es la iniciación del versionado de datos, donde se almacenan características específicas. En el contexto de un modelo de regresión logística, se determinan parámetros críticos como la cantidad de ciclos para el balanceo de datos, el tamaño de la muestra y el porcentaje asignado para las pruebas, cumpliendo con el previamente establecido 20 %. En la fase de producción se identifican los procesos de guardar la información esencial, como los modelos resultantes y sus respectivos desempeños, precisión y otros indicadores, todo centralizado en un sistema de registro de modelos.

Continuando con el flujo del proceso, se registra la información del código fuente, es decir, la ubicación donde se crearon los códigos para la generación del modelo, siendo esta información cargada en una fuente de datos en línea. Este momento marca la viabilidad de realizar la integración continua, que se lleva a cabo mediante la misma herramienta de versionado de código Git, estableciendo así un flujo de trabajo continuo en el desarrollo del modelo.

La fase de integración continua se realiza a través de la misma herramienta de repositorio de código que respalda el versionado del código GitHub. A continuación, se automatiza el proceso de entrenamiento del modelo, descrito en la fase de producción, que contribuye a una automatización progresiva. En este enfoque, el entrenamiento se configura automáticamente en intervalos regulares o después de ciertas tareas específicas, generando de manera continua registros de modelos en desarrollo.

En otras palabras, cada vez que se inicia un entrenamiento, se registra un nuevo modelo, listo para su implementación posterior. Este modelo entrenado se despliega y se expone a través de una API, facilitando el acceso desde diversos dispositivos, ya sea una computadora o un teléfono celular. Además, se crea una aplicación web que permite a los usuarios cargar imágenes simples para realizar predicciones sobre la presencia o ausencia de plagas.

El proceso se complementa con un servicio de monitoreo diseñado para supervisar, registrar y verificar la información de los distintos modelos. Este monitoreo también se utiliza para analizar el rendimiento de los modelos, y en caso de un bajo rendimiento según las métricas establecidas, se activa un mecanismo de notificación.

Cuando el sistema detecta un bajo rendimiento, se dispara una notificación, generalmente a través de correo electrónico, que se envía directamente a un científico de datos. Este profesional evalúa la situación y determina los procesos a seguir para modificar y mejorar el rendimiento del modelo. El ciclo de integración continua, entrenamiento automatizado y monitoreo proactivo contribuyen a mantener y mejorar constantemente la eficacia de los modelos en producción.

Un proceso de entrenamiento de un modelo debe ser escalable, colaborativo y reproducible. Los principios, herramientas y técnicas que garantizan que la construcción de modelos sea escalable, colaborativo y reproducible se conoce como MLOps. El MLOps sería llevar cualquier modelo de machine learning desde su etapa de desarrollo a la etapa de producción y que se encuentre disponible para un usuario final, y a su vez sea confiable y eficiente.

Dentro de las características del MLOps confiable y eficiente se encuentran:

- Buenas prácticas para crear código (variables y funciones de métodos adecuados para que se entienda el código elaborado).
- Control de versiones (código, datos y modelos de ML).
- Pruebas automáticas (sobre el flujo del sistema de ML para evitar fallas en el modelo que es el usado por los usuarios finales).
- Reproducibilidad (ayuda a obtener y replicar los mismos resultados con los datos y modelos anteriores).

- Documentación (ayuda a entender como fue realizado el código).
- Monitoreo (verifica cambios que pasan con el tiempo al modelo).

A partir de estas características y específicamente después de realizar varias iteraciones para ajustar la parametrización de los modelos, se busca determinar cuál es el modelo óptimo que será implementado en producción. Durante este proceso de búsqueda, se versiona cada uno de los modelos en consideración y se compara su rendimiento para identificar el más eficiente. Cabe señalar que, se llevan a cabo pruebas automáticas en todo el modelo, contribuyendo así a la detección y prevención de posibles fallos en el flujo del modelo para ponerlo en producción.

Las pruebas se realizan al modelo siempre y cuando no sea demasiado complejo de reentrenar o validar. En particular, se destaca que, debido a la naturaleza misma de los algoritmos, es más lógico y eficiente entrenar el modelo de regresión logística en comparación con modelos más exigentes computacionalmente, como el modelo YOLOv8. Incluso con el primero, es posible utilizar únicamente la CPU para obtener un resultado, mientras que el segundo requiere el uso simultáneo de CPU y GPU para funcionar correctamente.

En el contexto de la reproducibilidad, se refiere a la capacidad de tener los datos en un punto específico, lo que significa que se puede rastrear qué datos se utilizaron exactamente para entrenar un modelo en particular. La reproducibilidad es esencial para replicar resultados en diferentes momentos en el tiempo. Este proceso se ve facilitado mediante una documentación que ayuda a comprender las acciones realizadas, especialmente en relación con el código de la solución implementada. La documentación proporciona la transparencia necesaria para entender el enfoque adoptado y los resultados obtenidos en cada fase del proceso.

Tabla 6: Elementos de trabajo con MLOps

Elementos	**Características**	**Acciones**
Análisis de datos	El análisis de datos se refiere al proceso de inspeccionar, limpiar, transformar y modelar datos con el objetivo de descubrir información útil, llegar a conclusiones y respaldar la toma de decisiones.	Durante esta etapa, se aborda la tarea de comprender la naturaleza de los datos, especialmente en el contexto de la identificación de plagas. Se busca discernir la presencia o ausencia de plagas, examinando la estructura y características de los datos. Se destacan acciones específicas, como la identificación de dimensiones desiguales en los datos, donde algunas muestras presentan mayor cantidad de píxeles en el ancho que en la altura. Ante esta diversidad, se afronta el desafío de establecer uniformidad en las dimensiones, asegurando coherencia en el conjunto de datos.
Feature Store (almacén de funciones)	Se almacenan las transformaciones que se le han aplicado a los datos, para después entrenar los modelos.	En el modelo de la regresión logística se remueve el fondo de las imágenes y después se redimensionan a un valor indicado de 640 x 640 pixeles.

Tabla 6: Elementos de trabajo con MLOps

Elementos	**Características**	**Acciones**
Versionado del código	En donde se guarda todo el código generado, y se sube a internet.	Facilitar una integración adecuada de la solución realizada para los modelos tanto de regresión logística como Yolo V8.
Integración continua	Se refiere a la práctica de fusionar y validar regularmente los cambios de código en un repositorio compartido.	Permitir que diferentes miembros de un equipo de trabajo, pueden realizar una integración correcta de los códigos que se van agregando a un repositorio.
Versionado del modelo	Permite cambiar entre modelos en tiempo real o monitorear diferentes modelos	Se verifica cómo se van comportando los modelos dependiendo a parámetros que se le vayan cambiando y determinar cuál entre ellos es el mejor y poder después de haberlo identificado colocarlo en producción.
Registro de modelos	Una vez que se ha entrenado un modelo, se almacena en un sistema de registro de modelos junto con sus metadatos.	Selección de parámetros por modelo a registrar, como ciclos de entrenamiento y numero de imágenes a emplear para la regresión logística. Cada una de estos parámetros, junto con el modelo entrenado, es agregado de manera integral al registro del modelo. La práctica de agregar estos metadatos al registro del modelo se realiza de manera continua, generando así versiones sucesivas del modelo a medida que se lleva a cabo el proceso de validación.

Tabla 6: Elementos de trabajo con MLOps

Elementos	**Características**	**Acciones**
Model serving	Desplegar el modelo para que pueda ser usado por los usuarios, por medio de API y/o aplicación.	Sea en un computador, en una tablet o en un celular, el acceso al modelo se facilita para cualquier usuario, ya sea un individuo común o incluso un científico de datos. Se desarrolla una API y una aplicación con la finalidad de que el usuario final, pueda observar el comportamiento del modelo. La API y la aplicación permite pasar datos de prueba o datos reales, permitiendo así la validación de la existencia o no de plagas en un aguacate Hass.
Monitoreo de modelos	Se deben monitorear los modelos, para detectar cambios en el desempeño o puesta en producción. Esto es debido a que los modelos en producción pueden degradarse con el tiempo por los cambios en los datos de entrada, fallas en el sistema o fallas al momento de la generación del modelo.	Se lleva a cabo una monitorización continua para identificar cualquier cambio que pueda surgir en el rendimiento de un modelo. Este cambio puede manifestarse a través de variaciones en el rendimiento comparándose con una métrica global, así como errores de los servicios. Estas discrepancias pueden originarse por posibles fallas en diversas etapas del proceso.
CI / CD	Garantizan que el código se fusione frecuentemente donde se realicen compilaciones y pruebas automatizadas en un repositorio central.	En el proceso de MLOps, implica no solo probar el código, sino los modelos resultantes y desplegar el modelo en una API y/o una aplicación para los usuarios finales.

Tabla 6: Elementos de trabajo con MLOps

Elementos	**Características**	**Acciones**
Almacén de metadatos	El registro es fundamental para la reproducibilidad. Se pueden registrar todos los parámetros pasados a un modelo, como las métricas después de evaluarlo o registros del hardware usado.	Se permite el registro de información de los modelos, debido a que es parte esencial para la reproducibilidad. Esto implica la retención de información, específica para el modelo, como los parámetros y las iteraciones realizadas, así como la cantidad de datos empleados durante el proceso de entrenamiento de un modelo, entre otros. Este registro es para garantizar la capacidad de reproducir exactamente las condiciones bajo las cuales se desarrolló un modelo en particular.

Fuente: Elaboración propia

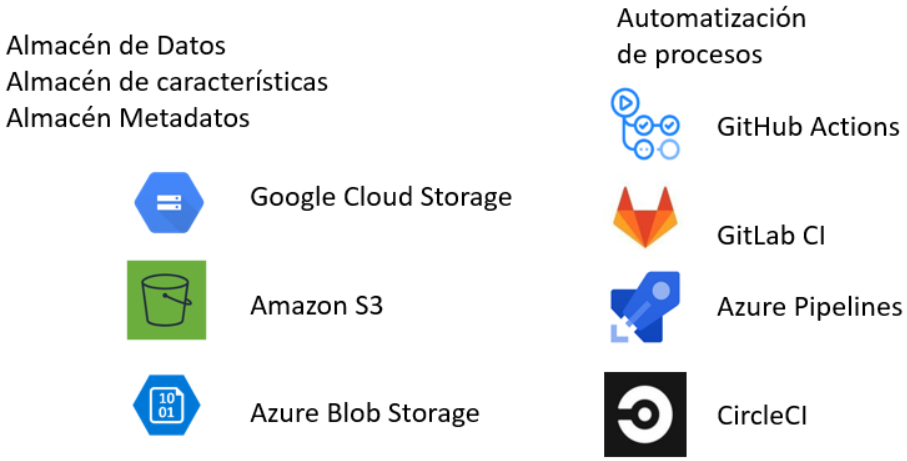
Profundizando en el ciclo del MLOps, se retoma la referencia previa a la integración continua y despliegue continuo. Estas dos fases trabajan en conjunto para asegurar que el código se fusiona de manera regular y que las implementaciones son compiladas y automatizadas en un repositorio central. La integración continua se centra en fusionar el código con frecuencia, implementando compilaciones automatizadas que son almacenadas en un repositorio central.

En cuanto al proceso, no se limita únicamente a probar el código; también implica evaluar los modelos resultantes. Este paso incluye el despliegue del modelo en una aplicación, permitiendo que los usuarios finales consuman los resultados. La entrega continua va más allá, implicando la implementación de cambios en el código en un entorno de pruebas o producción. Esta fase facilita despliegues rápidos para corregir fallos o implementar cambios en un modelo, posibilitando su incorporación en un entorno de producción.

Para esta investigación se optó por seleccionar herramientas Open Source y así contar con una solución a bajo costo, y estas se integraron con Google Cloud Platform (GCP) para aprovechar características específicas. Aunque GCP es un servicio de pago, se ha utilizado la capa gratuita que abarca diferentes servicios sin límite de tiempo, y si no se sobrepasa esa capacidad no habrá cobros adicionales.

Como se viene mencionando, cada etapa del ciclo de vida del MLOps se cumple, total o parcialmente, mediante el uso de herramientas específicas (ver figura 31, figura 4, figura 5 e figura 6). Este enfoque garantiza una implementación efectiva y personalizada del ciclo de vida del MLOps, adaptándolo a las necesidades y requisitos específicos del proyecto.

Figura 31: Herramientas para almacenamiento y automatización de procesos



Fuente: Elaboración propia

Para esta implementación de MLOps se utilizó Jupyter. El repositorio de código es GitHub, que es una plataforma líder en el desarrollo colaborativo de software que proporciona servicios de alojamiento de repositorios Git, permitiendo a los miembros de un equipo trabajar juntos en proyectos, gestionar versiones de código y facilitar la colaboración distribuida.

Figura 32: Herramientas utilizadas en el proyecto de MLOps para plagas en aguacate Hass

Fuente: Elaboración propia

Las herramientas seleccionadas, como se mencionó una de ellas es Google Cloud Storage, elegida por su capa gratuita limitada, lo que permite manipular eficazmente este almacenamiento. Además, Google Cloud ofrece un crédito de alrededor de $300 dólares para utilizar diversas funciones de Google, lo cual resulta beneficioso para el análisis de datos.

Otra herramienta clave es Visual Studio Code, que desempeñó un papel esencial en las transformaciones mencionadas. Fue especialmente útil para la transformación de los cuadernos de Jupyter a Python, proporcionando una interfaz intuitiva para manipular el código directamente en formato Python. Esto facilitó la gestión del código en un repositorio utilizando Git, donde se lleva a cabo el versionado de todo el código y lo realizado en el desarrollo.

Figura 33: Herramientas utilizadas en el proyecto de MLOps para plagas en aguacate Hass

Fuente: Elaboración propia

Para el registro de modelos, se optó por MLFlow, una herramienta elegida por su eficacia en la gestión de versiones de modelos. En el ámbito del versionado de datos, se seleccionó la herramienta DVC, que facilita el almacenamiento de los datos por versiones. Para la construcción de aplicaciones web, se utilizó Streamlit. Para la implementación de la API, tanto local como remotamente, se empleó FastAPI. Es importante señalar que esta parte del proyecto, aunque extensa, es una extensión de la actividad 2 del objetivo tres, que implica la identificación de las herramientas empleadas para desarrollar la metodología MLOps en el proyecto.

Figura 34: Herramientas utilizadas en el proyecto de MLOps para plagas en aguacate Hass

Fuente: Elaboración propia

En lo que respecta al monitoreo del modelo, se eligió realizar pruebas con Pytest, una dependencia de Python que evalúa el código y ejecuta pruebas. Este proceso de evaluación se lleva a cabo de manera programada a diario a las 11 PM GMT-5 utilizando GitHub Actions, una herramienta de automatización de procesos.

La elección del horario específico, las 11 PM, se estableció para automatizar la ejecución de tareas en un momento menos propenso a la manipulación activa de modelos, especialmente por parte de los científicos de datos. No obstante, este horario podría ajustarse según necesidades y criterios específicos.

En cuanto al almacenamiento de características, se utilizó Google Cloud Storage, empleando sus capacidades para almacenar las transformaciones aplicadas a las imágenes en el modelo de regresión logística. Cabe mencionar que, aunque se optó por este modelo para implementar la metodología MLOps debido a su menor consumo de recursos computacionales y tiempos de respuesta más rápidos en comparación con el modelo Yolo, esta metodología también se podría adaptar para Yolo según los requisitos específicos del modelo.

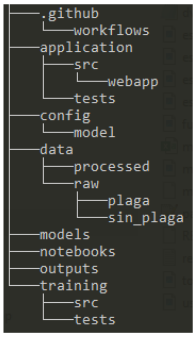
Estas herramientas, cuidadosamente seleccionadas, ofrecieron una solución integral que abarcó desde el almacenamiento y análisis de datos hasta la gestión efectiva del código, de los datos, de los modelos y sus versiones, contribuyendo así al éxito del proyecto en el ámbito de la ingeniería de software.

8.3.2. Integración con herramientas MLOps

Después de haber identificado las herramientas utilizadas para implementar MLOps. La sección se centra en el uso de estas herramientas y destaca el código disponible en un enlace proporcionado, organizado en una estructura específica de carpetas basada en una plantilla de ciencia de datos de cookiecutter[10], la cual es una dependencia de Python. El enlace a continuación proporciona acceso al código completo del proyecto, organizado de manera estructurada en carpetas con GitHub[11].

Figura 35: Estructura de las carpetas del proyecto

- **.github:**
  - **workflows:** Almacén de los archivos de extensión Yaml, que realizan las tareas automatizadas.
- **application:**
  - **src:** Códigos de la API con FastAPI.
    - **webapp:** Códigos de la Aplicación con Streamlit.
- **config:** Archivos de configuración.
- **data:** Almacén de datos (raw - datos sin procesar, processed - datos procesados)
- **models:** Almacén de modelos ya entrenados.
- **notebooks:** Almacén de notebooks para analizar visualmente los resultados.
- **outputs:** Guarda las configuraciones que se tuvieron por cada ejecución de un modelo.
- **training:**
  - **src:** Códigos de Python para el desarrollo del modelo.
  - **test:** Códigos de prueba para el modelo.



Fuente: Elaboración propia

[10] La plantilla se encuentra en el siguiente enlace de acceso público en GitHub aquí.
[11] Las carpetas y el código se encuentran en el siguiente enlace de acceso público en GitHub aquí.

La estructura de carpetas utilizada es esencial para comprender la lógica detrás de la implementación. Uno de los componentes de esta estructura es el "workflows", que almacena los archivos con extensiones Yaml y actúa como la base de las tareas automatizadas. Este punto será abordado de manera más detallada en el objetivo 4. Este enfoque en la organización y el propósito de las carpetas proporciona una visión clara de la implementación subyacente.

Para realizar la ejecución local y obtener la estructura completa de carpetas desde el enlace proporcionado, se sigue un proceso que implica la clonación del repositorio desde GitHub. Una vez obtenida la información localmente, el código disponible puede ejecutarse para aquellos que lo necesiten. Todas las dependencias necesarias para el normal funcionamiento del proyecto localmente están detalladas en el archivo dev-requirements.txt[12].

Las dependencias del proyecto para trabajar de forma local se encuentran en la raíz del proyecto, en un archivo llamado dev-requirements.txt. Para instalar las diferentes dependencias seria con el comando:

```
pip install -r dev-requirements.txt
```

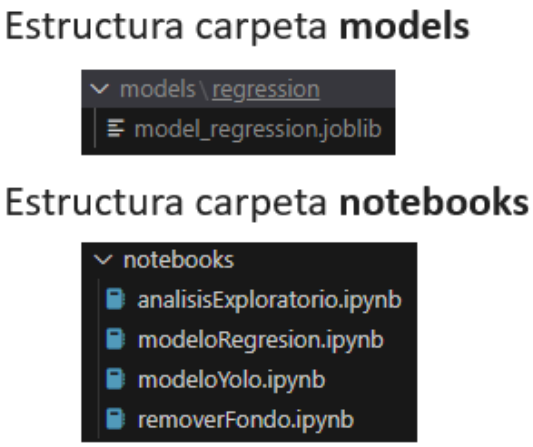
Cuando se clona el proyecto, se obtienen todos los archivos y carpetas en la ubicación local deseada. Una vez completada la clonación, se puede acceder a la estructura completa del proyecto. Este proyecto ha sido diseñado considerando todas las dependencias necesarias para su ejecución.

En cuanto a la estructura de carpetas, se destaca la carpeta "models", que desempeña un papel crucial en el almacenamiento del modelo entrenado. En este contexto, el modelo de regresión logística, se almacena en formato "joblib", una dependencia de Python diseñada para guardar modelos de Machine Learning de manera eficiente, que facilita su obtención y reproducibilidad.

Adicionalmente, se tiene la carpeta "notebooks", la cual contiene los códigos asociados del modelo de regresión logística mencionado anteriormente en formato jpynb de Jupyter Notebook. En este sentido, los mismos códigos utilizados para el objetivo dos del proyecto se encuentran aquí, lo que facilita la revisión y comprensión de las implementaciones.

[12]Las dependencias para trabajar con el proyecto localmente se encuentran aquí.

Figura 36: Estructura de carpeta models y notebooks



Fuente: Elaboración propia

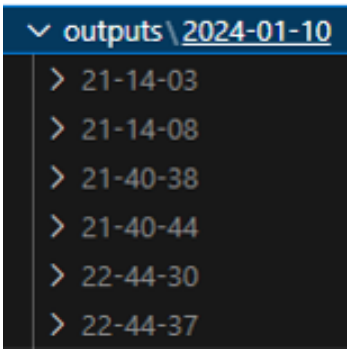
Cabe resaltar que estos códigos no solo representan lo realizado del modelo de regresión logística, sino que también abarcan los procedimientos aplicados en la implementación del modelo Yolo V8. La inclusión de ambas implementaciones se encuentra en la carpeta "notebooks" ofreciendo una visión completa y organizada de las soluciones realizadas, permitiendo una fácil referencia y reproducción de los procesos aplicados.

En pos de simplificar y preservar la integridad de las configuraciones durante el entrenamiento o evaluación de un modelo, se implementó una estrategia que evita la manipulación directa de los valores dentro del código. Para lograr esto, se incorporó un framework opensource para Python llamado Hydra, el cual facilita la gestión de configuraciones al externalizarlas en archivos dedicados.

En lugar de codificar valores directamente, se optó por utilizar archivos de configuración, aprovechando la funcionalidad proporcionada por Hydra. Esta herramienta permite centralizar todas las configuraciones en un archivo específico, que contiene los parámetros que definen la configuración para la ejecución de un programa. Estos archivos de configuración adoptan el formato YAML, proporcionando una estructura clara y legible, la adopción de archivos de configuración no solo mejora la legibilidad del código al eliminar valores directos, sino que también facilita la adaptabilidad del programa a diferentes contextos y escenarios.

Otro aspecto es que, con Hydra, cada vez que se evalúa un modelo, las configuraciones se guardan automáticamente en una carpeta designada llamada "outputs". En esta carpeta, cada ejecución del modelo genera automáticamente un subdirectorio con la fecha, hora, minutos y segundos precisos en los que se llevó a cabo. Este enfoque asegura un registro detallado de las configuraciones utilizadas en cada ejecución, lo que no solo facilita la reproducibilidad, sino que también permite un seguimiento de los parámetros específicos utilizados en diferentes ejecuciones.

Figura 37: Estructura carpeta outputs

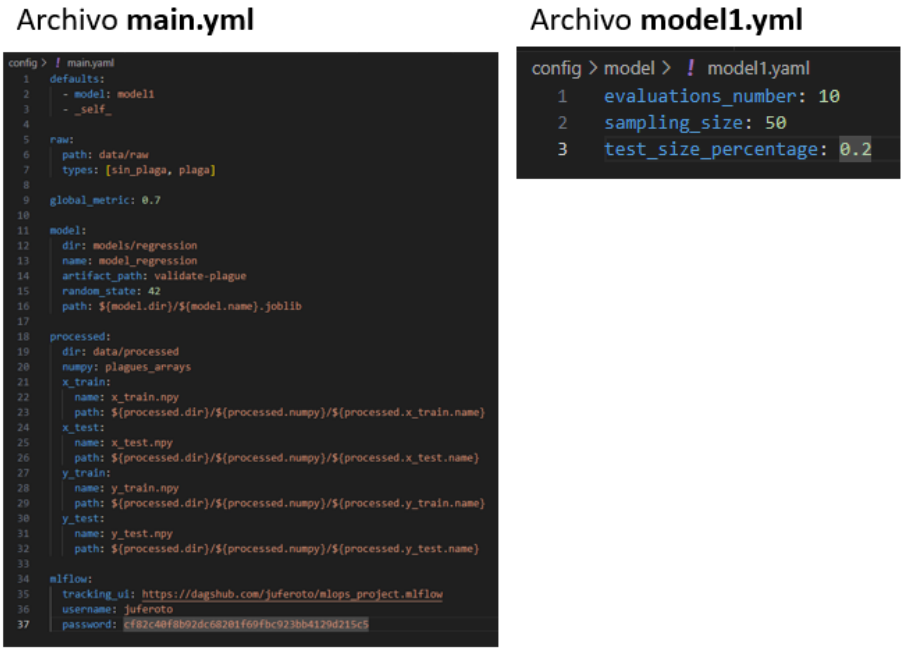


Fuente: Elaboración propia

Este enfoque ofrece una trazabilidad exhaustiva, beneficiando a aquellos que requieren información detallada sobre las ejecuciones del modelo, y también contribuye al registro y versionado de los modelos. Cada ejecución crea un marcador temporal, proporcionando una forma sistemática y organizada de rastrear las diferentes versiones del modelo. Además, se extiende al versionado de los datos, lo que añade un nivel adicional de control y seguimiento en cada ejecución de un modelo.

En el archivo main.yaml, se guardan todas las configuraciones que se tienen para el proyecto, como las variables al seleccionar el modelo por defecto a trabajar, que en este caso es el modelo de regresión logística. Como también, la ruta en donde se encuentran ubicadas las imágenes procesadas de la información del cultivo del aguacate Hass, las cuales pueden variar en términos de tener o no tener fondo. Todas estas imágenes se almacenan en la carpeta "data/raw" y se guardan los dos tipos mencionados en el objetivo uno: las imágenes con plaga y las imágenes sin plaga.

Figura 38: Configuraciones del archivo main.yaml y del archivo model1.yaml



Fuente: Elaboración propia

Es en esta sección donde se define la ubicación específica para almacenar el modelo de regresión. La carpeta designada para este propósito es la carpeta "models", y se asigna un nombre único a dicho modelo. El "artifact path" es un componente manejado internamente por la herramienta MLFlow, para identificar el espacio de trabajo en donde se van almacenar los distintos modelos.

Además, se configura el "random state" para la división de conjuntos de entrenamiento y prueba. Aunque esta división se realiza de manera aleatoria, se garantiza que, aunque sea de manera aleatoria, la semilla (random state) permanezca constante. Este enfoque asegura que, al ejecutar el código en diferentes máquinas locales o remotas, se obtenga consistentemente la misma información en términos de conjuntos de datos de entrenamiento y prueba.

En la ruta especificada "data/processed", es donde se almacena la información ya procesada, es decir, las transformaciones que se realizan al modelo, como la redimensión de las imágenes, la normalización y la eliminación de fondos de las imágenes.

Durante este proceso, la información se redimensiona, normaliza y se eliminan los fondos de las imágenes, acciones fundamentales en la solución del modelo de regresión lineal. Además, se establecen diferentes conjuntos de entrenamiento y pruebas utilizando las variables disponibles en el archivo "model1".

Además, se tienen las configuraciones de acceso de la herramienta MLFlow para realizar el versionado de datos. La función específica de esta configuración se explicará detalladamente en el objetivo cuatro. En resumen, MLFlow desempeña un papel crucial en el seguimiento y versionado de los datos, proporcionando un registro detallado de las diferentes iteraciones y cambios realizados durante el proceso de entrenamiento de un modelo. La configuración detallada no solo garantiza la consistencia en la preparación de datos y la separación de conjuntos, sino que también establece las bases para el seguimiento exhaustivo del versionado de datos a lo largo del proyecto.

Como se mencionó anteriormente, fue necesario reestructurar por completo el modelo de regresión logística, trasladándolo desde cuadernos de Jupyter a extensión Python. Esta transformación se llevó a cabo con el objetivo de simplificar el proceso, especialmente para poder llevar a cabo la implementación de tareas automatizadas. La reorganización del código implicó la división en archivos separados, cada uno destinado a cumplir una función específica. Los archivos son:

- **evaluate_model.py**: Aquí se tiene una función que evalúa el modelo que se acaba de generar después de haberlo procesado y entrenado. Se hace con los conjuntos de prueba generados de la función que está en el archivo process.py[13].

- **helper.py**: Aquí se tiene una función que se crea con el propósito de registrar información para la herramienta de registro de modelos MLFlow[14].

- **main.py**: Aquí se tiene una función encargada de ejecutar paso a paso todo el proceso de generación del modelo de ML (procesamiento de las imágenes, entrenar y evaluar el modelo)[15].

- **preprocessor.py**: Esta es una clase auxiliar que se contiene las funciones para remover el fondo de las imágenes del aguacate y para cambiar el tamaño de las imágenes (normalizar las imágenes) a valores entre 640 X 640[16].

- **process.py**: Aquí se tiene una función que llama a la clase auxiliar preprocessor.py para realizar el procesamiento de las imágenes y así generar los diferentes conjuntos de entrenamiento y pruebas que se van a usar para entrenar el modelo[17].

- **train_model.py**: Aquí se tiene una función que se encarga de entrenar el modelo, con los conjuntos de entrenamiento generados de la función que esta en el archivo process.py[18].

[13] El código del archivo evaluate_model.py se encuentra aquí.
[14] El código del archivo helper.py se encuentra aquí.
[15] El código del archivo main.py se encuentra aquí.
[16] El código del archivo preprocessors.py se encuentra aquí.
[17] El código del archivo process.py se encuentra aquí.
[18] El código del train_model.py se encuentra aquí.

Para entrenar un modelo, se puede realizar mediante la línea de comandos utilizando el siguiente comando en Python:

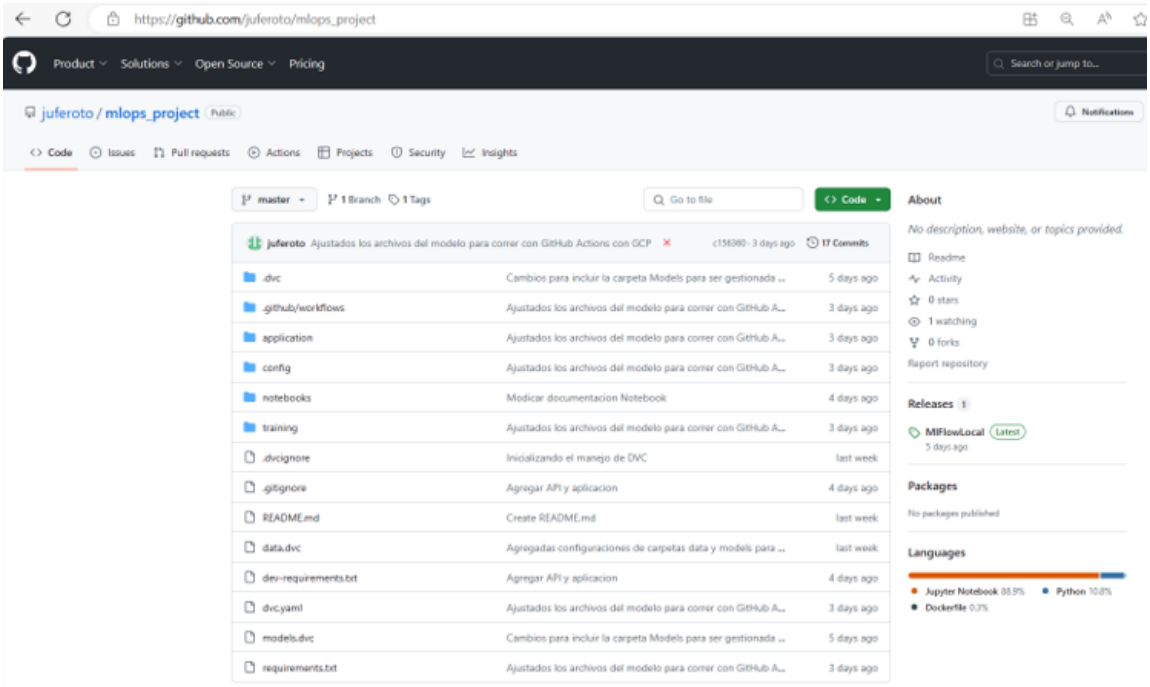
```
python training\src\main.py
```

Este comando dirige la ejecución al archivo principal main.py ubicado en el directorio 'training', iniciando el paso a paso de las operaciones correspondientes, para generar un modelo de regresión logística. Este proceso abarca desde el procesamiento de imágenes hasta el entrenamiento y la evaluación del modelo.

Adicionalmente, durante el proceso se almacenan en el espacio dedicado de MLFlow las métricas resultantes del modelo recién entrenado. Junto con estas métricas, se guarda otra información relevante, como los parámetros utilizados en el modelo. En este caso, se registra el número de evaluaciones, el porcentaje de la prueba y el tamaño del conjunto de datos. Todos estos detalles se conservan para facilitar el registro del modelo utilizando la herramienta de MLFlow, como se mencionó anteriormente.

En relación a como se versiona el código, se realiza por medio de GitHub, la cual es una plataforma web que proporciona servicios para el control de versiones y la colaboración en el desarrollo de software.

Figura 39: Herramienta para versionado de código - GitHub

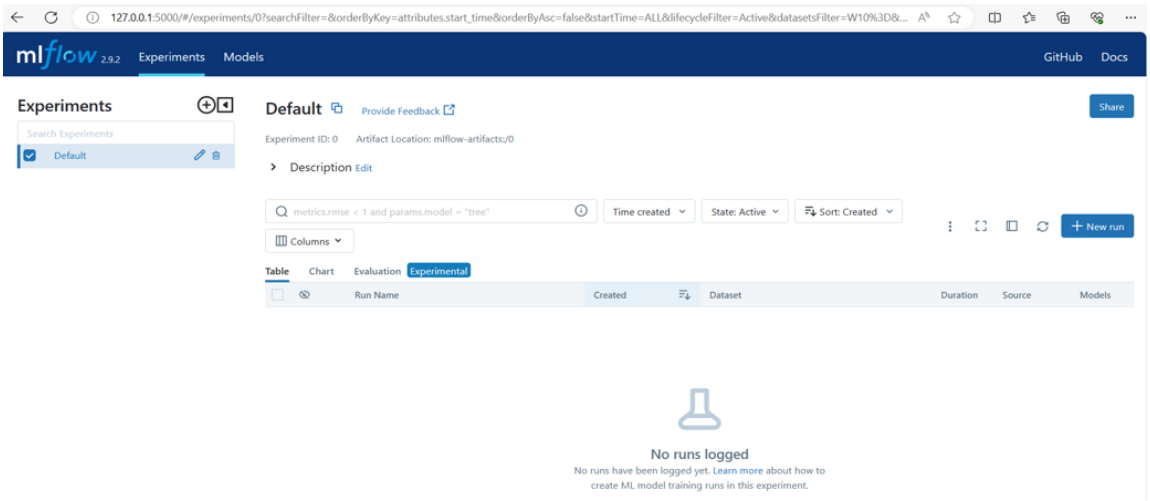


Fuente: Elaboración propia

Ahora, en relación a como se versiona el modelo de regresión logística, pasamos a la herramienta MLOps que estamos aplicando, y esta es MLFlow. Esta es la herramienta elegida para versionar modelos y realizar trazabilidad de experimentos, así como para el registro de modelos. MLFlow es integral para el desarrollo y la gestión de proyectos de aprendizaje automático, facilitando el seguimiento, la reproducción y el registro, así como la trazabilidad de los experimentos. Ofrece una solución unificada que abarca todo el ciclo del MLOps.

Sin embargo, es importante destacar que algunas funcionalidades de MLFlow son de pago. Por esta razón, únicamente estamos utilizando la parte gratuita de ML Flow, específicamente la que se enfoca en el registro de modelos y trazabilidad de experimentos. Estas características permiten almacenar y validar los modelos localmente o incluso en la nube, integrándose si se requiere con otras herramientas.

Figura 40: Herramienta para versionar modelos y trazabilidad de experimentos en MLFlow



Fuente: Elaboración propia

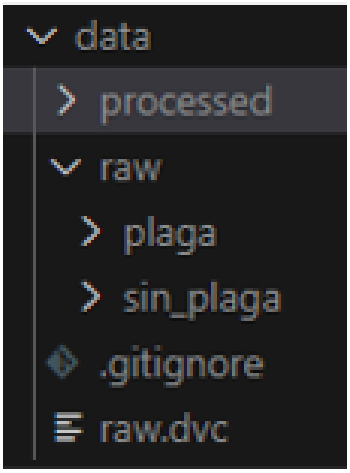
En la figura 40 se describe la interfaz de MLFlow, donde se presentan de manera clara y organizada cada uno de los experimentos que están en ejecución. Además, se exhiben los modelos que han sido registrados.

Todas las configuraciones y modelos pueden almacenarse en una máquina o servidor local. Más adelante, en el objetivo cuatro, exploraremos la posibilidad de integrar el MLFlow con una herramienta por internet. Por el momento, se ha optado por la solución local, donde todos los modelos, configuraciones y datos se guardan de manera local.

Con la integración de las herramientas seleccionadas para la metodología MLOps, se cuenta con la presencia de DVC, una herramienta que simplifica la gestión de versiones en proyectos de ciencia de datos y aprendizaje automático. DVC proporciona un control de versiones específico para datos, lo que facilita la colaboración, garantiza la reproducibilidad y favorece la integración con otras herramientas de machine learning.

Esta herramienta al permite el versionado de datos, es decir, la capacidad de transitar de un punto a otro, es especialmente útil cuando se ejecuta un código o un modelo. DVC posibilita conocer exactamente cuáles fueron los datos específicos utilizados en una ejecución particular. Así, al ejecutar un modelo y utilizar la versión de datos con DVC, es posible obtener una visión precisa de los datos involucrados en esa ejecución específica.

Figura 41: Estructura de la carpeta "data" del proyecto



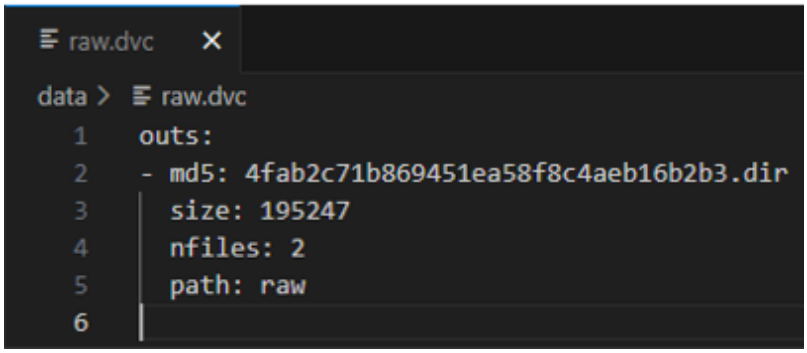
Fuente: Elaboración propia

Para este proyecto en particular, se implementó la siguiente estructura de carpetas para gestionar los datos. En la carpeta "data", se encuentra el subdirectorio "raw", que alberga todos los datos procesados o sin procesar, clasificados en carpetas clave como "sin plaga" y "con plaga".

En la sección "processed", como se detalla en el archivo de configuraciones "main.yaml", se encuentran los valores ya procesados y separados en conjuntos de entrenamiento y pruebas. Esta configuración es esencial para la gestión eficiente de DVC.

La estructura de archivos utilizada para el control de versiones de datos se presenta como un ejemplo en pantalla. Automáticamente, al agregar versionado a la carpeta "raw", se genera un archivo como el que se muestra aquí (ver figura 42), el cual es esencial para el versionado de datos con DVC. Este proceso garantiza la trazabilidad y el control de versiones de los datos que se requieren para la ejecución de un modelo.

Figura 42: Estructura del archivo para realizar control de versiónes de los datos a la carpeta "data/raw"



Fuente: Elaboración propia

La herramienta DVC, se utiliza comúnmente para registrar el estado de los datos, es decir, en qué punto se encuentran los datos. Estos datos pueden transitar por diferentes estados a lo largo de las diversas etapas del proceso para generar un modelo de regresión logística[19].

En el contexto de esta investigación, el procedimiento sigue varias etapas. Por ejemplo, existe una etapa en la que los datos se encuentran sin procesar. Luego, se pasa a una etapa donde los datos son procesados y se almacenan en su estado modificado. Posteriormente, se avanza a la etapa de entrenamiento del modelo, y después de completar esta fase, se obtiene un modelo ya entrenado. Por lo tanto, la herramienta DVC permite gestionar eficazmente los diferentes estados de los datos a medida que avanzan a través de las distintas etapas del proceso.

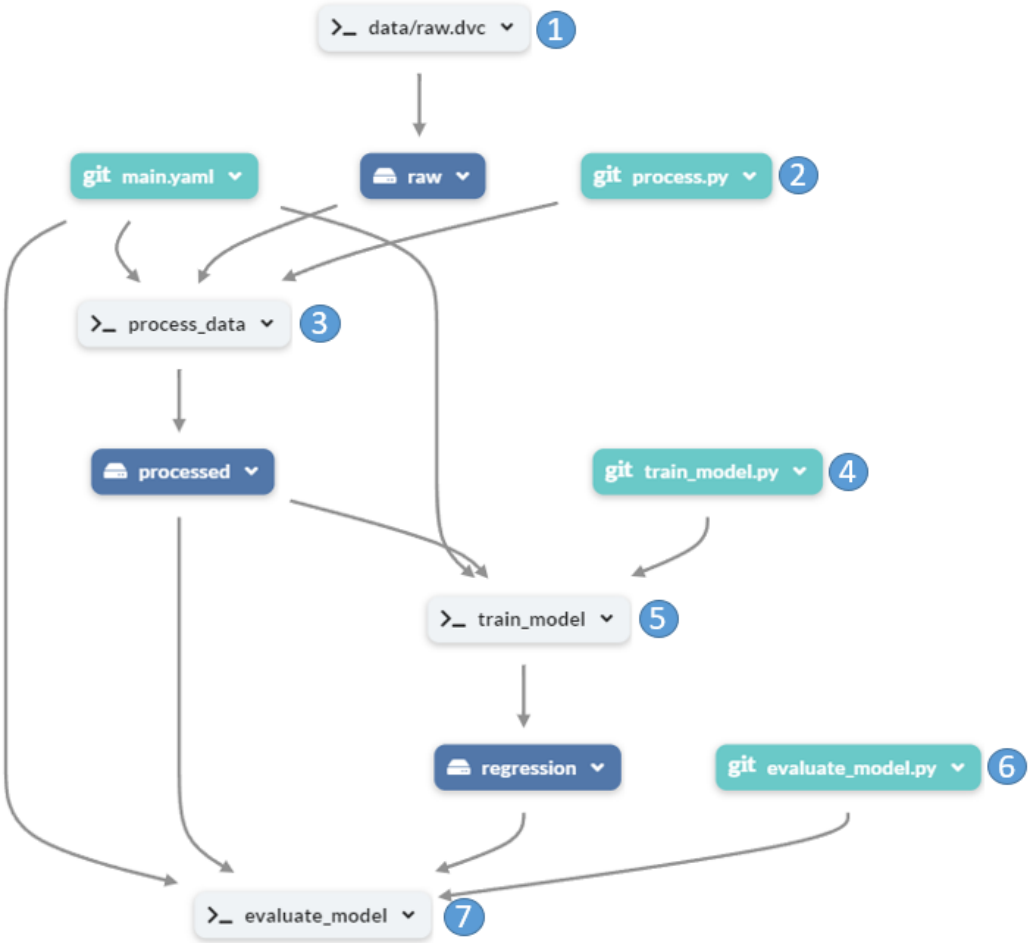
[19]La estructura de archivos de los datos del proyecto con DVC se encuentra aquí.

La estructura que se presenta aquí (ver figura 43) es extraída directamente de la herramienta DagsHub que será detallada en el objetivo cuatro. Esta herramienta integra tanto MLFlow para el versionado de modelos, DVC para el versionado de datos y GitHub para el versionado de código.

Estas son las seis etapas de los datos que se están manejando:

1. Estado inicial de los datos (datos sin procesar).
2. Carpeta donde se ubican los datos, archivo de configuración y script para procesar los datos.
3. Estado que se va a realizar para el proceso de los datos (process_data).
4. Carpeta donde se guardan los datos ya procesados y se separan el conjunto de entrenamiento y prueba. También está el script para entrenar el modelo.
5. Estado que se va a realizar para el entrenamiento del modelo (train_model).
6. Carpeta donde se guarda el modelo.
7. Estado que se va a realizar para la evaluación del modelo (evaluate_model).

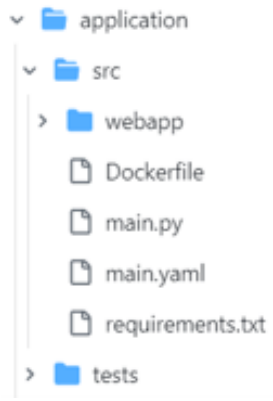
Figura 43: Estado de los datos del proyecto con DVC



Fuente: Elaboración propia

La otra herramienta que es utilizada para la creación de API es FastAPI, la cual es un marco web moderno y rápido para desarrolladores que desean construir APIs rápidas, seguras y bien documentadas con Python[20]. Su enfoque basado en tipos y su compatibilidad con estándares modernos hacen que el desarrollo de APIs sea una tarea eficiente y robusta.

Figura 44: Estructura de la carpeta de la API del proyecto



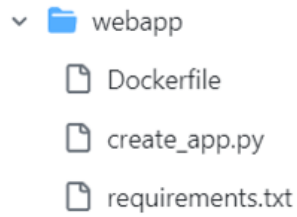
Fuente: Elaboración propia

Aquí se presenta la estructura de carpetas de la API del proyecto. El archivo Docker, en particular, forma parte del objetivo 4. El código empleado para generar la API está contenido en el archivo "main.py". En este código, se expone un método HTTP - GET que sirve para obtener una respuesta después de que un usuario suba una imagen, para validar contra el modelo seleccionado para ser usado en la API.

Para la creación de aplicaciones web, se utiliza la herramienta Streamlit, una dependencia de Python que simplifica la creación de aplicaciones web interactivas diseñadas para la visualización de datos y prototipos[21].

[20] El código que genera la API se encuentra en el siguiente enlace en GitHub aquí,
[21] El código que genera la aplicación se encuentra en el siguiente enlace en GitHub aquí

Figura 45: Estructura de la carpeta de la aplicación del proyecto



Fuente: Elaboración propia

Del contenido sobre el archivo Dockerfile se mencionara en el objetivo 4 y el archivo requirements.txt son las dependencias de Python que necesita la aplicación para funcionar.

Dentro del código desarrollado para generar una aplicación destinada a ser consumida por los usuarios finales. Streamlit simplifica este proceso, ofreciendo una herramienta poderosa para desarrolladores que buscan crear rápidamente aplicaciones web interactivas, en este caso, enfocadas en datos.

Por otro lado, para iniciar MLFlow desde una máquina local, se utiliza el comando

```
mlflow server
```

Al ejecutar este comando desde la línea de comandos, se genera una dirección HTTP local que permite su uso y acceso directo desde el propio computador o servidor local[22]. Para hacer uso apropiado del MLFlow en local se debe cambiar el archivo de configuraciones "main.yaml" del proyecto, colocando la dirección HTTP del MLFlow en el puerto local 5000.

[22]El video en donde se muestra el uso de MLFlow de manera local se encuentra aquí

8.3.3. Despliegue en entorno de pruebas

Se procede con el despliegue del entorno de pruebas, y en este contexto se reentrena el modelo de regresión logística, se lanza la aplicación y la API, y se demuestra el funcionamiento de la herramienta MLFlow, todo en un entorno local.

Para iniciar la API desde una máquina local, basta con ejecutar el siguiente comando:

```
python application\src\main.py
```

A su vez, para iniciar la aplicación desde una máquina local se debe ejecutar el siguiente comando:

```
streamlit run application\src\webapp\create_app.py
```

Además, también se deberá lanzar MLFlow utilizando el comando mencionado previamente[23].

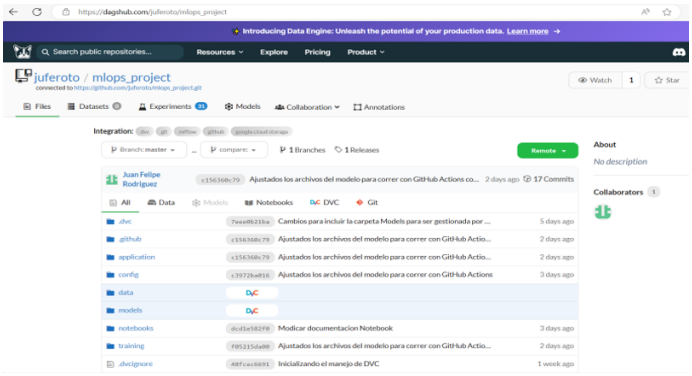
8.4. Uso de MLOps mediante despliegue en un ambiente controlado, con la capacidad de monitorear y mejorar continuamente el rendimiento del modelo

8.4.1. Despliegue en Cloud

La plataforma DagsHub se presenta como una solución integral específicamente diseñada para la ingeniería de software, fusionando de manera efectiva las funcionalidades esenciales de control de versiones del código, versionado de datos y seguimiento de experimentos. Este enfoque unificado facilita la administración de proyectos en los campos de aprendizaje automático, desde la fase inicial de experimentación hasta la colaboración en equipo y la entrega de resultados.

[23]El video del despliegue en entorno de pruebas y los comandos empleados se encuentra aquí.

Figura 46: Plataforma que integra el versionado del código, los datos y los modelos en DagsHub



Fuente: Elaboración propia

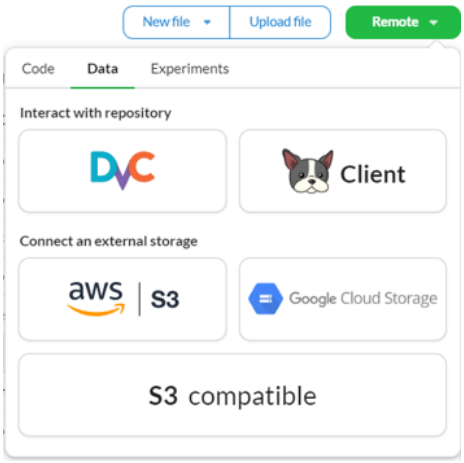
Al acceder a la interfaz de DagsHub mediante el enlace detallado[24], se inicia el proceso para utilizar esta plataforma. Para dar inicio a su uso, es posible crear una cuenta de forma gratuita. Esta suscripción no tiene ningún costo y permite gestionar proyectos públicos sin restricciones. El límite máximo para el espacio de almacenamiento es de 100 gigabytes, proporcionando así un entorno accesible para el manejo de datos en proyectos específicos.

La configuración del proyecto sigue la misma estructura que se detalló en el objetivo 3, tal como se describe en la figura 46. Al crear una cuenta en DagsHub, esta se vincula automáticamente con la cuenta Github utilizada para el versionado del código. Posteriormente, al compartir el enlace, se posibilita a otros usuarios acceder a la configuración del proyecto de manera sencilla. Este enfoque facilita la colaboración y el intercambio de información entre los miembros del equipo alineando la integración del proyecto de manera eficiente.

Cuando se busca incorporar almacenamiento externo, se presentan dos opciones específicas: AWS (Amazon S3) y Google Cloud (Google Storage). En este proyecto en particular, se optó por utilizar Google Cloud Storage (ver figura 47), debido a que en su capa gratuita abarca 5 GB por mes sin límite de tiempo. Al hacer clic en la opción correspondiente, se despliegan las configuraciones necesarias, permitiendo así realizar las acciones pertinentes para integrar y agregar el almacenamiento en la nube de Google al proyecto. Este proceso se simplifica mediante una interfaz intuitiva que guía al usuario a través de las opciones necesarias para una configuración eficaz.

[24]El acceso al DagsHub del proyecto se encuentra aquí.

Figura 47: Opciones para configurar un almacén de datos en DagsHub

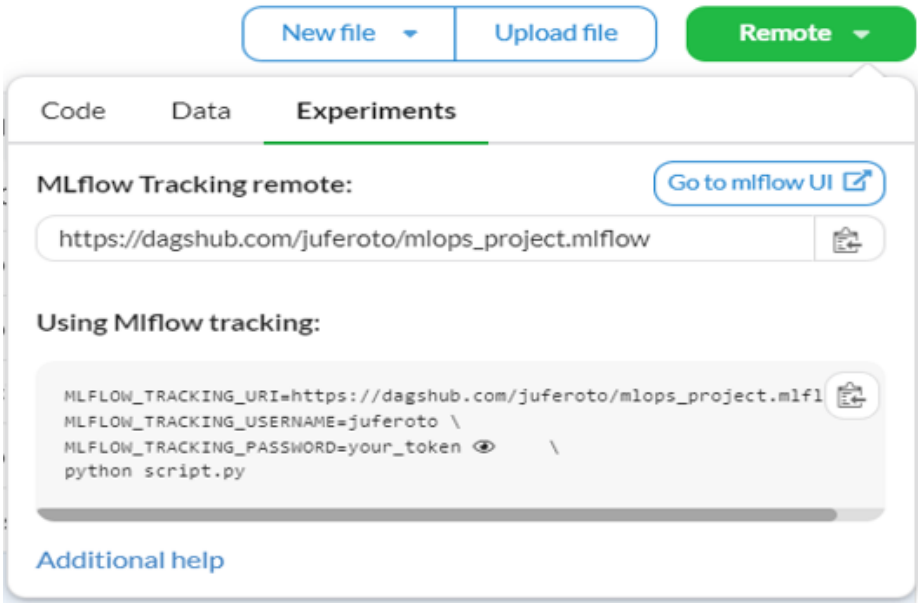


Fuente: Elaboración propia

Al seleccionar la opción "Remote", se habilita la capacidad de visualizar los experimentos realizados con MLFlow (ver figura 48). Es importante destacar que, en este contexto, la visualización no ocurre a nivel local, sino que se realiza de forma remota a través de la plataforma proporcionada por DagsHub con la "Go to mlflow UI". Esto implica que los datos y resultados almacenados en MLFlow se encuentran accesibles de manera remota, gracias a la integración facilitada por DagsHub.

Para contextualizar y entender cómo se integra MLFlow en la nube con DagsHub, es esencial considerar los ajustes necesarios en el archivo principal "main.yaml" de configuraciones. Al hacer la comparación, se identifica que en la configuración de MLFlow para trabajar en la nube, se requiere la adición de la URL que indica la ubicación específica del proyecto proporcionado por DagsHub, como el usuario y contraseña que provee DagsHub. Estos ajustes permiten que el proyecto se conecte con los recursos en la nube de MLFlow, facilitando así la gestión y visualización remota de experimentos y datos asociados con el proyecto realizado.

Figura 48: Opciones para usar remotamente MLFlow



Fuente: Elaboración propia

El enlace específico del proyecto que provee DagsHub para MLflow es https://dagshub.com/juferoto/mlops_project.mlflow. Este enlace es el que se debe sustituir en el archivo de configuraciones "main.yaml"[25].

Siguiendo con el despliegue en la nube, con el objetivo de asegurar un funcionamiento sin contratiempos tanto para la aplicación como para la API, es crucial determinar las dependencias necesarias. Por esta razón, se genera un archivo llamado requirements.txt., Este archivo se crea para enumerar todas las dependencias esenciales[26].

La instalación de estas dependencias se lleva a cabo mediante el comando

```
pip install -r requirements.txt
```

[25]Dentro del siguiente video se demuestra la gestión remota de la herramienta MLflow, siguiendo un proceso similar al realizado localmente. La diferencia clave radica en que esta vez se realiza de manera remota. El enlace para acceder al video se encuentra disponible en YouTube aquí.

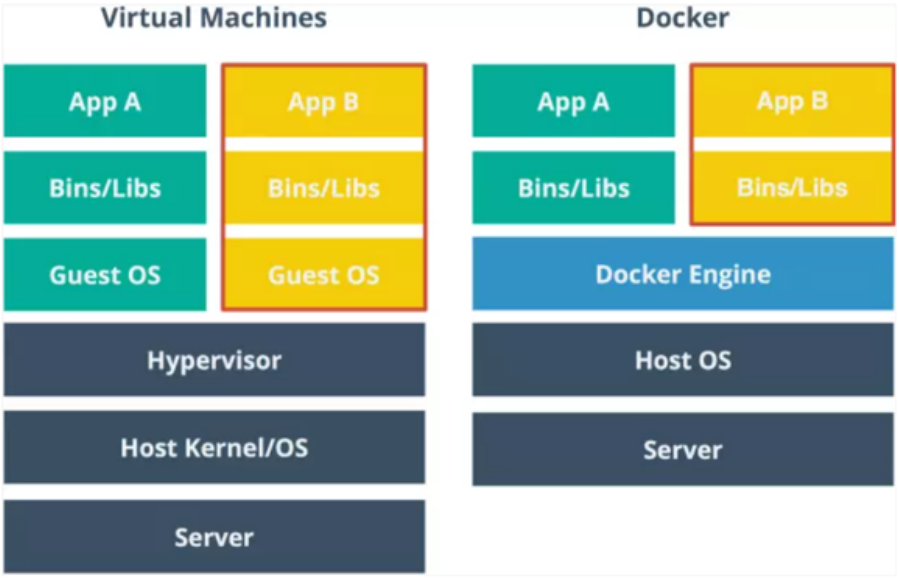
[26]Las dependencias para asegurar el funcionamiento de la aplicación y la API localmente se encuentran aquí.

Este proceso garantiza que todas las herramientas y dependencias necesarias se instalen de manera coherente, proporcionando así las condiciones apropiadas para el correcto funcionamiento de la aplicación y la API en la nube.

Para desplegar la aplicación y hacerla accesible en línea, una opción destacada es el uso de contenedores. Estos, según su descripción, son una tecnología de virtualización a nivel del sistema operativo que posibilita empacar y distribuir aplicaciones junto con sus dependencias y configuraciones.

La utilización de contenedores, especialmente a través de la conocida herramienta Docker, simplifica significativamente el proceso de desarrollo, implementación y gestión de aplicaciones. Docker se ha consolidado como una de las soluciones más utilizadas en este ámbito, ofreciendo una solución eficiente y coherente para el empaquetado, distribución y ejecución de aplicaciones (Orovengua, 2016).

Figura 49: Diversas maneras de despliegue



Fuente: Orovengua (2016)

Continuando con lo planteado por Orovengua (2016), existen diversas maneras de llevar a cabo despliegues en entornos de nube, siendo dos de ellas notables: el uso de máquinas virtuales y la implementación de contenedores con Docker. La distinción clave entre ambas radica en que, mientras cada aplicación en máquinas virtuales requiere un sistema operativo virtualizado en su totalidad (utilizando, por ejemplo, un gigabyte de memoria RAM), los contenedores de Docker optimizan el rendimiento al aprovechar el kernel, es decir, el núcleo de la máquina real. Estos contenedores únicamente cargan en la memoria las bibliotecas o dependencias necesarias para ejecutar la aplicación, evitando la duplicación de recursos. Esto resulta en una eficiencia considerable, ocupando solo alrededor del 20 % del espacio en comparación con una máquina virtual de 1 gigabyte o 1024 megabytes. En este caso, el contenedor utiliza solo 200 megabytes de capacidad, ofreciendo así una solución más eficaz y económica en términos de recursos.

Para emplear Docker se requiere de un archivo esencial en el contexto del uso de contenedores. Cabe destacar que, al optar por emplear contenedores con Docker, se requiere la creación de un archivo denominado Dockerfile[27]. En este archivo se incorporan las configuraciones necesarias para el despliegue de una aplicación.

Figura 50: Configuración del archivo Dockerfile para desplegar la API con FastAPI y Docker

Dockerfile

application > src > Dockerfile > ...

```
1   FROM tiangolo/uvicorn-gunicorn:python3.11-slim
2
3   WORKDIR /app
4   ADD . /app
5
6   RUN python -m venv venv
7
8   RUN /bin/bash -c "source venv/bin/activate"
9
10  RUN pip install -r requirements.txt
```

Fuente: Elaboración propia

[27]El código del archivo Dockerfile para la API se encuentra en GitHub aquí.

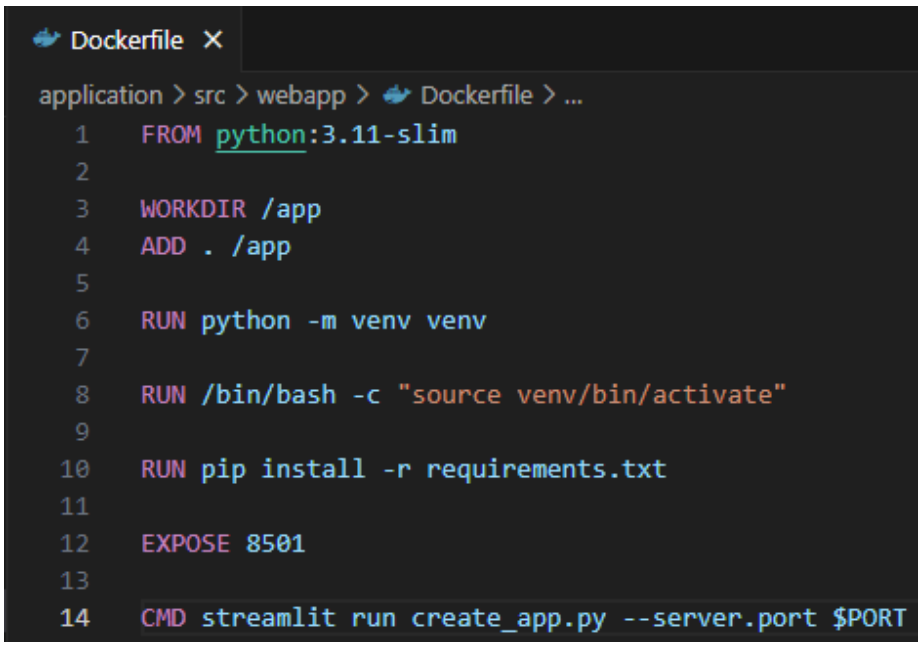
El código que se presenta en la figura 50 tiene como propósito desplegar la API. En este fragmento, se especifica la descarga de la versión necesaria de Python y de Uvicorn, que se utiliza para habilitar FastAPI, la cual expone la API para su acceso desde internet.

Adicionalmente, el código también detalla la carpeta de destino donde se copiará la aplicación. Se establece la creación de un entorno virtual, su activación, y la instalación de las dependencias necesarias para ejecutar la API de manera efectiva.

Un archivo Dockerfile también es esencial para el despliegue de la aplicación. Mientras el Dockerfile previo estaba diseñado para desplegar la API, este nuevo archivo está destinado a implementar la aplicación con Streamlit, una herramienta que simplifica la creación de la aplicación en general[28].

[28]El código del archivo Dockerfile para la aplicación se encuentra en GitHub aquí

Figura 51: Configuración del archivo Dockerfile para desplegar la aplicación con Streamlit y Docker



Fuente: Elaboración propia

En este contexto, se vuelve a crear el archivo Dockerfile (ver figura 51), especificando la imagen necesaria de Python. Se detalla la carpeta de destino donde se copiará el proyecto. El proceso implica la creación y activación de un entorno virtual específico, la instalación de dependencias cruciales, la exposición del puerto designado para acceder a la aplicación, y la ejecución de un comando que permite visualizar la aplicación desde internet.

Para lograr lo anterior, es necesario emplear contenedores como medio de empaquetamiento y almacenamiento de toda la información necesaria. Una vez que se haya encapsulado la información dentro de un contenedor, el siguiente paso implica transferir dicho contenedor a una herramienta o proveedor de servicios en la nube para facilitar el despliegue. En este caso, como se detalló en el tercer objetivo, se optó por utilizar Google Cloud Platform (GCP) como proveedor de servicios en la nube.

GCP ofrece una amplia gama de servicios y herramientas que abarcan desde plataformas complejas hasta infraestructuras personalizables. Esta diversidad brinda a los desarrolladores la flexibilidad de seleccionar la solución que mejor se ajuste a sus necesidades específicas. En el contexto de este proyecto en particular, se eligieron tres servicios de GCP que se consideran idóneos para satisfacer los requisitos establecidos. Los servicios seleccionados para realizar despliegues con GCP en la nube para el proyecto son:

- **Google Cloud Storage**: Se utiliza para el almacenamiento y recuperación de datos.
- **Google Artifact Registry**: Se usa para la gestión de artefactos (contenedores) de software del proyecto.
- **Google Cloud Run**: Se utiliza para desplegar y ejecutar los contenedores de manera administrada y escalable.

La selección de estos servicios fue posible gracias a la extensa variedad que GCP proporciona, permitiendo así una adaptación precisa a las demandas del proyecto. Estos servicios específicos se eligieron para garantizar una implementación correcta de la solución propuesta. Tanto Google Artifact Registry, como, Google Cloud Run fueron los que se usaron para la parte de los contenedores.

Antes de poder aprovechar las opciones ofrecidas anteriormente, es necesario establecer configuraciones y accesos adecuados para hacer uso de sus servicios. Este proceso es aplicable a los tres componentes esenciales de Google que se emplearon (ver anexo 1).

Las URLs que genera Cloud Run son usualmente estáticas por cada contenedor a desplegar. En ese sentido, la URL para hacer uso de la API es **https://api-plague-predict-6gzk55suba-uc.a.run.app/docs** y la URL para hacer uso de la aplicación es **https://web-plague-predict-6gzk55suba-uc.a.run.app**. Para hacer uso de estas, se debe contar previamente a que los contenedores se hayan creado y desplegado correctamente.

A continuación, se presenta en un video como se ve la API y la aplicación en la nube, con el modelo que ha sido seleccionado mediante MLFlow y se ha marcado para estar en producción[29].

[29]El video correspondiente a este proceso está disponible en YouTube aquí

8.4.2. Monitoreo sobre el modelo

Para realizar el monitoreo del modelo se emplea GitHub Actions que permite una gestión eficiente y automatizada de todo el ciclo de vida del modelo para llevarlo a producción, desde la integración de cambios hasta la entrega y despliegue en la infraestructura de GCP.

En el contexto específico del proyecto se han definido tres archivos clave: **validate_deploy_app.yaml**, **validate_model.yaml** y **validate_model_automatically.yaml**. Cada uno de estos archivos desempeña un papel en la validación del modelo y despliegue de la API y la aplicación.

En particular, el archivo **validate_deploy_app.yaml**[30]. tiene la responsabilidad de gestionar la validación del despliegue de la aplicación. Este proceso es esencial para garantizar que la aplicación se despliegue de manera correcta y coherente en el entorno de Google Cloud Run[31].

Dentro de GitHub Actions, la estructura se organiza en "Jobs", cada uno de los cuales contiene pasos específicos a seguir. En este caso particular, se ilustrará el contenido del workflow llamado "Desplegar API y aplicación en remoto".

El "Job" que es responsable de desplegar la API y la aplicación, se inicia con la denominación general del archivo. En este contexto, se identifica como "Desplegar API y aplicación". Dentro de este Job, se definen varios pasos, y el primero de ellos es la acción de "Checkout", que básicamente implica obtener la totalidad del proyecto almacenado en GitHub. La finalidad de este paso inicial es asegurar que el entorno de ejecución cuente con la versión más reciente y completa del proyecto antes de proceder con las tareas de despliegue de la API y la aplicación.

[30]La estructura del archivo validate_deploy_app.yaml se encuentra aquí.

[31]La estructura completa y detallada de cada uno de estos archivos se encuentra disponible en el enlace proporcionado, ofreciendo una visión profunda de cómo se gestionan las validaciones y despliegues en el marco del proyecto de ingeniería de datos. El código se encuentra en GitHub aquí

Continuando con el análisis del archivo **validate_deploy_app.yaml**, se incorpora otro paso fundamental: la configuración del ambiente. En este caso, se solicita cargar una versión específica de Python, en este ejemplo, la versión 3.11. Además, se lleva a cabo la instalación de los paquetes necesarios. Este paso es crucial para preparar el ambiente con las herramientas y dependencias adecuadas que se han mencionado anteriormente. La selección y configuración de la versión de Python es esencial para garantizar la compatibilidad con la aplicación y la API que se desplegarán.

Para llevar a cabo el despliegue en la nube, es imprescindible contar con el archivo Dockerfile. Se recuerda que este archivo corresponde tanto para el despliegue de la API y la aplicación. Este paso asegura que los elementos necesarios estén presentes y correctamente configurados para la implementación en cloud.

La autenticación se convierte en otro componente esencial de este proceso. Este archivo se autentica tanto con el servidor de GCP como con su componente Artifact Registry, que es la ubicación donde se almacenan los contenedores. Esta autenticación es necesaria para garantizar la autorización adecuadas al interactuar con los servicios de Google Cloud, y especialmente con el lugar donde residen las imágenes de los contenedores.

Posteriormente, el proceso implica la construcción y la carga de la imagen. Es decir, se toma el archivo Docker previamente configurado y se genera la imagen correspondiente. Esta imagen se sube al Artifact Registry mencionado anteriormente. Este paso final es para asegurar que la imagen esté disponible y lista para el despliegue en Google Cloud Run.

Al realizar el despliegue de la API, llevamos a cabo una serie de pasos específicos. En primer lugar, se procede a construir y subir la imagen al Artifact Registry. En segundo lugar, se ejecuta un paso adicional para desplegar la aplicación en Google Cloud. Tanto para la API como para la aplicación en sí, es esencial seguir cuidadosamente los comandos establecidos, garantizando así la disponibilidad y la integridad tanto de la API como de la aplicación.

Aquí se presentan configuraciones basadas en la documentación oficial de GitHub para llevar a cabo el despliegue en GitHub Actions. Estas configuraciones se ajustan a las prácticas recomendadas por GitHub para la implementación de flujos de trabajo automatizados.

Estas configuraciones, extraídas directamente de la documentación oficial, proporcionan una base sólida para la ejecución eficiente de acciones en GitHub. Al seguir estas directrices, se asegura la coherencia y la conformidad con las prácticas recomendadas.

Cabe señalar que, el archivo anterior **validate_deploy_app.yaml** se activa automáticamente cada vez que se realiza un "push", es decir, cuando se modifica el algoritmo del modelo o alguna de las partes del proyecto. Este escenario puede surgir debido a la identificación de nuevas adiciones o ajustes de parámetros esenciales. En situaciones donde el modelo almacenado en la nube no coincide con el correcto, es necesario realizar un nuevo entrenamiento y generar una versión actualizada. Este proceso de reentrenamiento también implica que el código subyacente ha experimentado modificaciones.

Además, este desencadenante automático también ocurre cuando se manipulan datos que se deben cargar. En tal caso, se realiza un "push" hacia la rama principal, que en este contexto es la rama Master en GitHub.

Este enfoque automatizado garantiza que cualquier cambio en el código del modelo o en los datos sea detectado y desencadene el proceso de validación. La ejecución del flujo de trabajo de GitHub Actions asegura la consistencia, la actualización del modelo y el despliegue del mismo en un entorno local, manteniendo la coherencia entre el código, los datos y la implementación realizada.

Para validar el modelo y la aplicación, se emplea un enfoque similar en GitHub Actions, donde se define un workflow con el nombre "Validar modelo y aplicación en local" en el archivo **validate_model.yaml**[32]. Este archivo contiene distintos "jobs", cada uno con sus respectivos pasos detallados para llevar a cabo la validación.

Cuando se trata de validar el proyecto en el contexto de Git, ya no se realiza directamente en la rama Master. En su lugar, se adopta un enfoque más colaborativo, donde la validación se lleva a cabo al crear una nueva rama. Esta práctica busca asegurar la integridad de los datos y la calidad del modelo antes de incorporarlo a la rama principal.

La validación sobre el modelo y la aplicación se realiza mediante un *pull request*, este paso adicional se implementa con el propósito de evaluar el estado en que se encuentren los datos, el desempeño del modelo y el despliegue de la aplicación en local, antes de fusionarse con la rama principal. Esta estrategia es relevante en entornos de trabajo colaborativo, donde distintas personas pueden estar modificando diferentes secciones del proyecto.

La incorporación del *pull request* permite a los científicos de datos validar de manera individual cómo se comporta su modelo con nuevos datos o cambios específicos. Esta práctica facilita la detección temprana de posibles problemas y proporciona una oportunidad para ajustar y mejorar antes de fusionar los cambios en la rama principal.

En ese sentido, en el contexto de la validación del modelo del archivo **validate_model.yaml**, se detalla el procedimiento para evaluar el comportamiento del modelo antes de fusionarlo con la rama principal. En este escenario, se crea una rama auxiliar para la validación. Se inicia un *pull request* en esta rama para observar cómo se comporta el proyecto con nuevos datos o cambios específicos. Si la evaluación resulta satisfactoria y todos los aspectos son adecuados, entonces el científico de datos puede proceder a fusionar los cambios con la rama principal.

[32]La estructura del archivo validate_model.yaml se encuentra aquí.

El último archivo, denominado **validate_model_automatically.yaml**[33], ha sido creado con la finalidad de ejecutar periódicamente un workflow específico llamado "Validar modelo y aplicación automáticamente".

El workflow se ejecuta de manera automática según una programación establecida mediante un cron. Los *cron* son convenciones para tareas programadas, donde los campos indican, en orden, minutos, horas, mes y día de la semana (ver figura 52). En este caso particular, se ha definido que la tarea programada se ejecute todos los días a las 11 PM GMT-5. Se ha seleccionado este horario considerando que no todas las personas estarán conectadas o disponibles a esa hora, lo cual optimiza la ejecución automática y periódica del workflow.

Figura 52: Regla de cron jobs

```
# ┌──────────── minute (0 - 59)
# │ ┌──────────── hour (0 - 23)
# │ │ ┌──────────── day of the month (1 - 31)
# │ │ │ ┌──────────── month (1 - 12)
# │ │ │ │ ┌──────────── day of the week (0 - 6)
# │ │ │ │ │
# │ │ │ │ │
# │ │ │ │ │
# * * * * *
```

Fuente: Elaboración propia

Es importante señalar que este archivo se centra en realizar pruebas tanto al modelo como al despliegue de la aplicación como se muestra en la figura 53. La secuencia de pasos establecidos en este archivo asegura la ejecución efectiva de estas pruebas automáticas de manera programada.

[33]La estructura del archivo validate_model_automatically.yaml se encuentra aquí.

Figura 53: Proceso de validación del modelo y aplicación automáticamente

Fuente: Elaboración propia

Para detectar un fallo durante la ejecución de las pruebas, ya sea al evaluar el modelo o al intentar desplegar la aplicación localmente, se han establecido las pruebas utilizando *pytest*. En caso de que alguna de estas pruebas falle, se activa otro "Job" denominado "Enviar correo en caso de falla" (ver figura 54). Este "Job" tiene la función de notificar sobre la detección de un fallo en alguna de las dos tareas.

Figura 54: Estructura de notificación por correo en caso de fallo del archivo validate_model_automatically.yaml

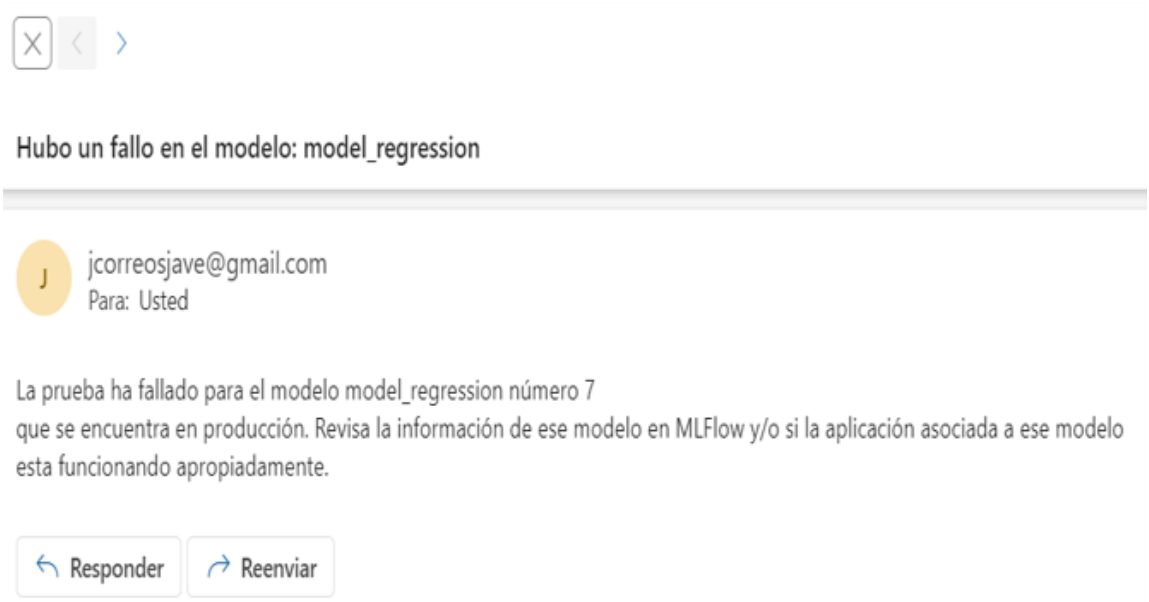
```yaml
notify:
  name: Enviar correo en caso de falla
  needs: test_model
  runs-on: ubuntu-latest
  if: failure()
  steps:
    - name: Genera y envía el correo de la falla
      uses: dawidd6/action-send-mail@v3
      if: ${{needs.test_model.outputs.validate_model == 'False' || needs.test_model.outputs.validate_app == 'False'}}
      with:
        server_address: smtp.gmail.com
        server_port: 465
        secure: true
        username: ${{ secrets.EMAIL_USERNAME }}
        password: ${{ secrets.EMAIL_PASSWORD }}
        subject: 'Hubo un fallo en el modelo: ${{needs.test_model.outputs.model}}'
        to: juferoto@hotmail.com
        from: jcorreosjave@gmail.com
        body: |
          La prueba ha fallado para el modelo ${{needs.test_model.outputs.model}} número ${{needs.test_model.outputs.version}}
          que se encuentra en producción. Revisa la información de ese modelo en MLFlow y/o si la aplicación asociada a ese modelo
          esta funcionando apropiadamente.
```

Fuente: Elaboración propia

La implementación de las pruebas en *pytest* ofrece una evaluación específica para identificar cualquier inconveniente tanto en el modelo como en el despliegue de la aplicación. La activación del "Job" sirve como mecanismo de alerta temprana, proporcionando la información necesaria en el caso de que alguna de las tareas no se ejecute conforme a lo esperado[34], situación que es enviada a un correo electrónico especificando el nombre y el número del modelo que se encuentra en producción.

[34]Se manda a ejecutar periódicamente un workflow de Github Actions que reevalúa el desempeño del modelo y envía un correo electrónico en caso de contar con un valor inferior predefinido. Esto se envía a la persona a cargo del modelo para que lo verifique, y después ejecute un despliegue con el modelo que considere está en buenas condiciones, para ser usado desde la API y aplicación.

Figura 55: Estructura del correo recibido



Fuente: Elaboración propia

Como se viene señalando, las pruebas se desarrollaron con *pytest*, siendo éste una dependencia de Python. La prueba con *pytest* al momento de evaluar el modelo, verifica 3 métricas (accuracy, precision y recall) establecidas a un valor del 70 %, en donde si se cuenta con valores inferiores en una de las tres, se envía una alerta al científico de datos (por correo), para que así proceda a revisar el estado del modelo. Esto se hace ejecutando una tarea programada diaria, que es ejecutada cada día a las 11 PM GMT-5 usando GitHub Actions.

Aquí se presenta la estructura de los archivos que se emplearon específicamente para realizar la evaluación del desempeño del modelo[35]. Entre ellos, se destaca un archivo llamado "get_variables.py"[36], cuya finalidad es obtener variables necesarias para el proceso, siendo utilizado en GitHub Actions. Este archivo proporciona información como el nombre del modelo y la versión del modelo, que se utilizarán en caso de fallas en las pruebas de rendimiento del modelo y despliegue.

[35]La estructura de archivos para evaluar el desempeño del modelo se encuentra en GitHub aquí
[36]La estructura del archivo get_variables.py se encuentra aquí.

Esta organización de archivos refleja una estrategia bien definida para evaluar el desempeño del modelo en el contexto de la ciencia de datos e ingeniería de software. La modularidad y la claridad en la estructura facilitan la gestión y ejecución efectiva de las pruebas, contribuyendo a un proceso adecuado.

Además, el archivo principal **test_train_model.py**[37] para evaluar el rendimiento del modelo contiene una funcionalidad clave. En este documento, se obtiene el modelo de MLFlow, que representa la versión actualmente en producción. Asimismo, se accede a los conjuntos de datos de prueba, y finalmente, se realiza la prueba de rendimiento, utilizando el modelo de regresión logística.

La obtención del modelo de MLFlow es un paso crítico, ya que garantiza que se esté evaluando la versión en producción, asegurando consistencia en el proceso de evaluación de rendimiento.

Adicionalmente, después de obtener las predicciones con el modelo obtenido desde MLFlow, se procede a calcular diversas métricas de evaluación. Estas métricas incluyen, entre otras, la exactitud (accuracy), la precisión (precision) y la sensibilidad (recall).

Este proceso inicia con la obtención de las predicciones a partir del modelo adquirido. Posteriormente, se realizan cálculos precisos para evaluar el rendimiento del modelo. Las métricas específicas mencionadas, como accuracy, precision y recall, proporcionan una visión del comportamiento del modelo en función de sus predicciones.

Asimismo, se compara con la variable que representa la métrica global previamente mencionada y se establece en un valor objetivo del 70 %. Esta variable actúa como un umbral de referencia, proporcionando un criterio claro para la evaluación del rendimiento del modelo. Al establecerla en 70 %, se indica un nivel deseado de desempeño en la salida del modelo.

[37] La estructura del archivo test_train_model.py se encuentra aquí.

Para finalizar se muestra la evaluación del despliegue de la aplicación con la estructura del archivo **test_service.py** para verificar el comportamiento del servicio.

En este código se tiene como objetivo evaluar el comportamiento del servicio, indicando que la aplicación se ha ejecutado correctamente. La lógica implementada verifica si la API ha sido cargada exitosamente y si está generando los resultados esperados. La evaluación se basa en una condición, donde si la condición se cumple, se considera que la prueba ha sido exitosa[38]. En caso contrario, si la condición no se satisface, la prueba falla y se activa la notificación por correo electrónico del archivo **validate_model_automatically.yaml**.

El proceso de monitorear métricas y ajustar el modelo para reentrenamiento si es necesario y así evitar la degradación del modelo y mostrar nuevamente en la nube la API y aplicación con el nuevo modelo seleccionado, se evidencia desde el video preparado para su acceso[39].

[38] El código para evaluar el despliegue de la aplicación se encuentra en GitHub aquí.

[39] Se muestra la notificación recibida por correo, para luego monitorear el modelo y seleccionar uno distinto o realizar reentrenamiento, debido a que hubo una degradación del modelo. Además, se muestra nuevamente la API y aplicación en la nube con el nuevo modelo seleccionado, en el siguiente enlace: https://youtu.be/U2DqNOixHWw

9. Conclusiones

En relación con la implementación de técnicas de procesamiento de imágenes para extraer características relevantes y mejorar la capacidad del modelo de Machine Learning en el reconocimiento y detección de las plagas *Stenoma catenifer* y *heilipus lauri* en el cultivo de aguacate Hass a partir de imágenes capturadas en campo se tomó el código diseñado en Jupyter para el análisis de las imágenes del cultivo de aguacate Hass, clasificadas en las categorías "con Plaga" y "sin Plaga", ajustando una herramienta efectiva para la inspección visual detallada de las características distintivas entre ambas clasificaciones. Este enfoque simplifica la tarea de análisis exploratorio de datos, revelando que el 93,28 % de las imágenes presentan plagas, mientras que el 6,72 % se consideran saludables, lo que indica un desbalanceo significativo de aproximadamente 1 imagen saludable por cada 14 imágenes con plagas. La distribución por dimensiones muestra que la mayoría de las imágenes tienen una resolución de 4,000 píxeles por ancho y 3,000 píxeles de alto.

Como consecuencia de tener modelos ML con alto desbalanceo puede significar que el modelo tendrá dificultades para generalizar y reconocer correctamente las imágenes sanas en datos nuevos y no vistos, resultando en una alta tasa de falsos negativos, donde imágenes sanas pueden ser incorrectamente etiquetadas como afectadas por plagas.

El procesamiento de la segmentación de imágenes, aplicando un algoritmo de redes neuronales convolucionales como Yolo, demuestra la eficacia del Deep Learning en la identificación precisa de plagas. La aplicación de polígonos para señalar las áreas afectadas permite una interpretación precisa por parte del algoritmo.

La fase de pre-procesamiento, destacando las zonas afectadas por la plaga, simplifica la tarea del algoritmo y mejora su capacidad de interpretación. En conjunto, estos procesos contribuyen a la creación de modelos confiables en la detección de plagas en cultivos de aguacate Hass.

Para el desarrollo y entrenamiento de un modelo de Machine Learning utilizando técnicas apropiadas de preprocesamiento y selección de características, así como algoritmos de aprendizaje supervisado o no supervisado, para lograr una detección de las plagas *Stenoma catenifer* y *heilipus lauri* en el cultivo de aguacate Hass se evidenció que, en el muestreo con validación cruzada (cross-validated subsampling) se realiza repetidamente a la clase mayoritaria para equilibrar las clases y se evalúa el modelo en cada iteración utilizando la clase minoritaria, donde es necesario promediar los resultados obtenidos en todas las iteraciones. Este enfoque es especialmente útil cuando se trata con conjuntos de datos desequilibrados, donde una clase tiene muchos más datos (imágenes) que otra. Al realizar varias iteraciones y calcular promedios, se busca obtener una evaluación más robusta del rendimiento del modelo, ya que cada iteración puede tener una muestra diferente de la clase mayoritaria.

En el proceso de selección de algoritmos para abordar el problema de detección de plagas en cultivos de aguacate Hass, se optó por la regresión logística y Yolo (You Only Look Once) V8. La regresión logística se utiliza para clasificación binaria, asignando probabilidades a la presencia o ausencia de plagas en las imágenes. Por otro lado, Yolo V8 es un algoritmo de clasificación que determina la probabilidad de la presencia de objetos (plagas) en una imagen.

El preprocesamiento desempeñó un papel crucial en la normalización del tamaño de las imágenes, reduciendo la complejidad computacional y asegurando la consistencia en las entradas del modelo. La normalización también evita problemas asociados con la variabilidad en el tamaño de las imágenes, facilitando el análisis pixel por pixel.

La selección y aplicación de estos algoritmos, junto con las técnicas de preprocesamiento y evaluación, demuestran una aproximación analítica y cuidadosa para abordar la información para la detección de plagas.

Tras el entrenamiento de modelos para la detección de plagas, la evaluación y comprensión de su desempeño se volvieron fundamentales. En el caso de la regresión logística, el análisis se centró en la matriz de confusión, permitiendo entender cómo el modelo clasifica los datos de prueba. Al calcular promedios de estas matrices tras 10 iteraciones, se obtuvo una precisión del 93.4 %, un recall del 90.4 %, y una exactitud del 92.5 %. Estos resultados demuestran una efectividad destacada en la identificación precisa de casos positivos y negativos, mostrando que el modelo tiene una capacidad sólida para predecir la presencia o ausencia de plagas.

Por otro lado, la implementación de Yolo V8 simplificó el proceso al centrarse solo en las imágenes con plagas. La evaluación por épocas reveló un progreso notable en la precisión del modelo, superando el 95 % en la época 9. Este enfoque optimizado demuestra eficiencia al dirigir la atención a instancias relevantes, mejorando significativamente la capacidad del modelo para clasificar imágenes con precisión.

Tanto la regresión logística como Yolo han demostrado ser herramientas efectivas para la detección de plagas, con sus respectivos enfoques y métricas de evaluación proporcionando información valiosa sobre el rendimiento de los modelos en distintas situaciones y condiciones.

En el desarrollo de una metodología MLOps que permita la integración, automatización y monitoreo del modelo de Machine Learning diseñado para el reconocimiento y control de las plagas *Stenoma catenifer* y *heilipus lauri* en el cultivo de aguacate Hass, se planteó un proceso dividido en dos etapas fundamentales: desarrollo y producción. En la fase de desarrollo, alineada con los objetivos iniciales, se centra en el preprocesamiento de datos, abordando tareas como normalización y redimensionamiento. La implementación del MLOps introduce el versionado del código y datos, permitiendo el control de cambios y la trazabilidad. Destaca la iniciación del versionado de datos, almacenando características específicas, como los parámetros críticos del modelo de regresión logística.

En la fase de producción, se identifican procesos clave, desde guardar modelos hasta exponerlos a través de una API, facilitando el acceso desde diversos dispositivos. La aplicación web permite a los usuarios cargar imágenes para predicciones. La integración continua se automatiza, registrando modelos en cada entrenamiento y desplegándolos para su implementación posterior. El monitoreo proactivo y notificaciones ante bajo rendimiento aseguran un mantenimiento constante y mejora de la eficacia de los modelos.

En el ámbito del monitoreo del modelo, se implementó una estrategia sólida utilizando Pytest, una dependencia de Python que permite evaluar y ejecutar pruebas de manera eficiente. Asimismo, con la automatización proporcionada por GitHub Actions permitió el proceso integral que abarca todo el flujo de trabajo.

La elección de Google Cloud Storage para el almacenamiento de los datos desempeñó un papel fundamental en la ejecución eficiente del ciclo de MLOps, específicamente en el modelo de regresión logística. Esta decisión se basó en las capacidades de su capa gratuita para almacenar las transformaciones aplicadas a las imágenes, proporcionando una solución para la gestión de datos.

La selección del modelo de regresión logística sobre el modelo Yolo se basó en consideraciones de eficiencia computacional y tiempos de respuesta más rápidos. Aunque esta elección específica se alineó con los requisitos del proyecto, es importante destacar que la metodología empleada es flexible y adaptable. La misma estrategia puede extenderse al modelo Yolo según las necesidades y criterios específicos del propio modelo.

En relación con la integración continua se encontró que, la integración de GitHub, MLFlow y DVC proporciona una robusta solución para el control de versiones y la trazabilidad de código, datos y modelos. Esta combinación permite gestionar eficientemente diferentes estados de datos a lo largo del ciclo de vida del proyecto, facilitando la reproducibilidad y la comprensión de los experimentos.

Por otro lado, la elección de FastAPI para la creación de una API agrega un componente ágil y bien documentado al desarrollo. Se destaca por su velocidad, proporcionando un entorno moderno para la construcción de una API de manera eficiente. Además, el uso de *Streamlit* para el desarrollo de aplicaciones web ofrece una solución potente y fácil de implementar, simplificando la creación de aplicaciones interactivas centradas en datos, agilizando el proceso y permitiendo una rápida visualización de la información para los usuarios finales.

Uso de MLOps mediante despliegue en un ambiente controlado, con la capacidad de monitorear y mejorar continuamente el rendimiento del modelo, se puede señalar que DagsHub, como plataforma integral, permite una solución diseñada específicamente para abordar las complejidades en los modelos de despliegue o MLOps. Su integración de funciones como el control de versiones del código (GitHub), la gestión de los datos (DVC), el seguimiento de datos y experimentos (MLFlow), refleja un enfoque holístico hacia la gestión de proyectos de aprendizaje automático. Al consolidar estas capacidades críticas en una única plataforma, DagsHub facilita un entorno colaborativo para equipos multidisciplinares como científicos de datos e ingeniería de software.

La sincronización entre el control de versiones del código y el seguimiento de datos ofrece una visión unificada del desarrollo, permitiendo un rastreo preciso de los cambios en el código y sus implicaciones en los datos. Además, la capacidad de realizar un seguimiento detallado de los experimentos facilita la evaluación y mejora continua de modelos de aprendizaje automático.

La incorporación de contenedores, emerge como un componente relevante para el proceso de la información, simplificando de manera significativa el desarrollo, implementación y gestión de aplicaciones. Docker, al consolidarse como una solución líder, muestra coherencia en el empaquetado, distribución y ejecución de aplicaciones en este ámbito.

La adopción de contenedores facilita la creación de entornos reproducibles, eliminando inconsistencias entre distintas etapas del desarrollo y asegurando una implementación uniforme en diferentes entornos. La portabilidad inherente de los contenedores amplifica la flexibilidad y agilidad en el despliegue de aplicaciones en este proyecto de detección de plagas para el cultivo del aguacate Hass.

Los servicios de Google Cloud Platform (GCP), como Google Cloud Storage, Google Artifact Registry y Google Cloud Run, ofrecen beneficios característicos para un ecosistema de datos dinámico, como lo son las imágenes establecidas para detectar las plagas en el aguacate Hass. Google Cloud provee una solución escalable y segura para almacenar y gestionar datos, permitiendo una fácil integración con otros servicios de GCP y facilitando el acceso a los datos.

A través de los archivos creados **validate_deploy_app.yml**, **validate_model.yml** y **validate_model_automatically.yml**, se evidencia la importancia de la gestión de tareas automatizadas con GitHub Actions. En conjunto, estos archivos ofrecen una visión completa de cómo se gestionan las validaciones y despliegues en el proyecto. La atención detallada a la validación del modelo, la estructura para el despliegue y la automatización de actualizaciones reflejan las prácticas adecuadas al momento de aplicar la metodología MLOps, asegurando la coherencia, fiabilidad y eficiencia en el desarrollo y despliegue de una API o aplicación.

En la evaluación del modelo y despliegue del modelo, este enfoque centrado en el desempeño del modelo en relación a una métrica global permite una evaluación del rendimiento del modelo que se encuentre en producción. La generación de esta métrica es esencial para comprender la eficacia del modelo y tomar decisiones informadas sobre su viabilidad.

Esta práctica en la definición de una o varias métricas refleja un enfoque cuantitativo en la evaluación del rendimiento en el ámbito de ciencia de datos. La claridad en la especificación de estos criterios facilita la toma de decisiones basadas en métricas bien definidas.

10. Futuros Cambios

1. La implementación de parámetros para asignar a un científico de datos designado a cada modelo a evaluar representa una práctica clave, debido a que se logra una asignación clara de responsabilidades. Las notificaciones a través de correos electrónicos dirigidos a científicos específicos agilizan la comunicación y permite una respuesta más rápida ante posibles problemas identificados durante la evaluación o el despliegue.

2. La automatización del despliegue de modelos constituye un enfoque estratégico en la ciencia de datos, ofreciendo una mejora continua en el rendimiento de los modelos. La implementación de un nuevo "workflow" que evalúe y seleccione automáticamente el mejor modelo basándose en métricas predefinidas claras refleja una práctica analítica avanzada. Este proceso no solo agiliza la toma de decisiones, sino que también garantiza la implementación del modelo más apropiado en producción sin la intervención humana.

   De allí que, el cambio automatizado entre modelos, donde el modelo seleccionado se despliega en producción mientras que el modelo anterior se archiva, demuestra un enfoque dinámico y adaptativo. La capacidad de evaluar continuamente el rendimiento de los modelos en producción y realizar cambios automáticos aumenta la agilidad y la eficacia del sistema en su conjunto.

3. Desacoplar el código para que permita manejar diferentes modelos de ML, y siga la metodología MLOps. Es relevante el desacoplamiento del código, para permitir la gestión de diversos modelos de machine learning (ML) y siguiendo la metodología MLOps, representa un enfoque estratégico y moderno para el desarrollo y despliegue de modelos. Al separar el código de la lógica específica del modelo, se logra una modularidad esencial que facilita la incorporación y gestión eficiente de diferentes modelos.

   La adopción de la metodología MLOps refleja una orientación hacia la integración continua y la entrega continua (CI/CD) en el ciclo de vida de los modelos de ML. Este enfoque promueve la automatización, la colaboración y la escalabilidad, contribuyendo a una implementación robusta y ágil.

11. Referencias Bibliográficas

Referencias

Agapito, A., Cibrián, L., Gutiérrez, R., Ruiz, J., López, C., and Rueda, P. (2022). Phytophthora cinnamomi rands en aguacate. Revista Mexicana de Ciencias Agricolas, 1(28):331–341. https://dialnet.unirioja.es/descarga/articulo/8624636.pdf.

Aimacaña, C. and Columba, G. (2021). Análisis comparativo de algoritmos de machine learning para la detección de plagas en los cultivos representativos de la sierra ecuatoriana. Tesis de pregrado, Universidad Central del Ecuador, Ecuador. http://www.dspace.uce.edu.ec/bitstream/25000/24228/1/UCE-FING-ISI-AIMACA%C3%91A%20JEFFERSON-COLUMBA%20ALEXANDER.pdf.

Arley, O. y Llano, R. (2016). Sistemas de información enfocados en tecnologías de agricultura de precisión y aplicables a la caña de azúcar, una revisión. Revista Ingenierías Universidad de Medellín, 15(28):103–124.

Ávila, P. (2022). Entrenamiento de redes neuronales para la identificación de plagas en cultivos de café. Tesis de grado, Universidad Pedagogica, Colombia. http://repository.pedagogica.edu.co/bitstream/handle/20.500.12209/18306/entrenamiento%20redes%20neuronales.pdf?sequence=4&isAl.

Bastian, M. (2023). Yolov8 shows the enormous possibilities of computer vision. https://the-decoder.com/yolov8-shows-the-enormous-possibilities-of-computer-vision/.

Carabalí, M., Caicedo, V., and Holguín, M. (2022). Guía para el reconocimiento y manejo de las principales plagas de aguacate cv. hass en colombia. Instituto Colombiana Agropecuario, Colombia. https://editorial.agrosavia.co/index.php/publicaciones/catalog/download/226/208/1422-1?inline=1.

Castañeda, V., Guerrero, M., Renteros, P., and Villanueva, M. (2021). Detección de nutrientes del suelo y planta, y pestes en campos de cultivo de banano orgánico con machine learning. Tesis de grado, Universidad Piura, Peru. https://pirhua.udep.edu.pe/bitstream/handle/11042/5204/T_IME_2104.pdf?sequence=1&isAllowed=y.

Contreras, C., Medina, D., Acevedo, J., and Guevara, I. (2022). Metodología de desarrollo de técnicas de agrupamiento de datos usando aprendizaje automático. Revista Tecnura, 26(72):42–58. http://www.scielo.org.co/pdf/tecn/v26n72/0123-921X-tecn-26-72-42.pdf.

Cruz, L., Caamal, C., Pat., F., and Reza, S. (2022). Competitividad de las exportaciones de aguacate hass de méxico en el mercado mundial. Revista Mexicana de Ciencias Agrícolas, 13(2):355–362. http://cienciasagricolas.inifap.gob.mx/index.php/agricolas/article/view/2885/4723.

Departamento Administrativo Nacional de Estadística (DANE) (2016). Cultivo del aguacate hass (persea americana mill; persea nubigena var. guatemalensis x persea americana var. drymifolia), plagas y enfermedades durante la temporada de lluvias. Boletín Mensual Insumos y Factores asociados a la Producción Agropecuaria, (50):1–102. https://www.dane.gov.co/files/investigaciones/agropecuario/sipsa/Bol_Insumos_ago_2016.pdf.

Díaz, V., Caicedo, V., and Carabalí, M. (2017). Ciclo de vida y descripción morfológica de heilipus lauri boheman (coleoptera: Curculionidae) en colombia. Revista Acta Zoológica Mexicana, 33(2):231–242. https://www.scielo.org.mx/pdf/azm/v33n2/2448-8445-azm-33-02-00231.pdf.

García, F. and Ulloa, L. (2022). Implementación de modelo machine learning aplicado al estudio de enfermedades de café en el centro de investigación sacha wiwa, perteneciente a la parroquia gasaganga, cantón la maná, provincia de cotopaxi. Tesis, Universidad Técnica de Cotopaxi, Ecuador. http://repositorio.utc.edu.ec/bitstream/27000/8981/1/UTC-PIM-000530.pdf.

Géron, A. (2019). Hands-on machine learning with scikit-learn, keras, and tensorflow: Concepts, tools, and techniques to build intelligent systems. pages 35–84. O'Reilly Media, Inc., 2nd edition.

Instituto Colombiano Agropecuario (2012). Manejo fitosanitario del cultivo del aguacate hass. mediadas para la temporada invernal. Ministerio de Agricultura y Desarrollo Rural, Colombia.

Jara, E., Giral, D., and Martínez, S. (2016). Implementación de algoritmos basados en máquinas de soporte vectorial (svm) para sistemas eléctricos: revisión de tema. Revista Tecnura, 20(48):149–170. https://www.redalyc.org/pdf/2570/257046835012.pdf.

Monsalve, S. (2021). Agricultura de precisión en la predicción de la merma de defectos en cultivos de banano. Tesis especialización, Universidad de Antioquia, Antioquia. https://bibliotecadigital.udea.edu.co/bitstream/10495/20593/10/SebastianMonsalve_2021_AgriculturaPrecisi%C3%B3n.pdf.

Neptune (2024). Mlops landscape in 2024: Top tools and platforms. https://neptune.ai/blog/mlops-tools-platforms-landscape.

Organización de las Naciones Unidas para la Alimentación y la Agricultura (2021). Perspectivas de la agricultura y del desarrollo rural en las américas: una mirada hacia américa latina y el caribe 2021-2022. pages 87–107. CEPAL, FAO, IICA. Costa Rica. http://repositorio.cepal.org/bitstream/handle/11362/47208/CEPAL-FAO21-22_es.pdf?sequence=1&isAllowed=y.

Organización de las Naciones Unidas para la Alimentación y la Agricultura (FAOSTAT) (2021). Hacia una agricultura sostenible y resiliente en américa latina y el caribe - análisis de siete trayectorias de transformación exitosas. Organización de las Naciones Unidas para la Alimentación y la Agricultura–FAO, Chile. https://www.fao.org/3/cb4415es/cb4415es.pdf.

Orovengua, J. (2016). Docker, virtualizada aplicaciones con contenedores. Linux Party. https://www.linuxparty.es/115-docker/9544-docker-virtualiza-aplicaciones-con-contenedores.html.

Palacios, T., Ramírez, A., Uribe, G., Granados, E., Romero, C., and Valdez, C. (2011). La palomilla barrenadora del aguacate *Stenoma catenifer* walsingham (lepidoptera: Elachistidae) en querétaro, méxico. Revista Acta Zoológica Mexicana, 27(2). https://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S0065-17372011000200021.

Pedro, V., Reynaldos, G., Ureta, A., and Cortez, P. (2021). Generalidades del machine learning y su aplicación en la gestión sanitaria en servicios de urgencia. Revista Médica de Chile, 149(2):248–254. https://www.scielo.cl/scielo.php?script=sci_arttext&pid=S0034-98872021000200248.

Proyecto Colombia Mide (2021). Informe ejecutivo: Estudio sobre las necesidades y brechas de la calidad en la cadena productiva de aguacate hass y plan de acción. antioquía y su zona de influencia. Instituto Nacional de Metrología INM, Programa de la Unión Europea, Bogotá. https://colombiamide.inm.gov.co/wp-content/uploads/2021/05/Informe-Aguacate-Hass_VFinal_20210506.pdf.

Rehman, A. (2022). Converting a custom dataset from coco format to yolo format. https://medium.com/red-buffer/converting-a-custom-dataset-from-coco-format-to-yolo-format-6d98a4fd43fc.

Reyes, A., A., C., López, F., Mendoza, B., and Secundino, J. (2022). Cambio climático y producción de aguacate. Revista de la Universidad de Chapingo. https://www.researchgate.net/publication/357662730_Cambio_climatico_y_produccion_de_aguacate.

Rivero, D. (2022). Investigación de mercado y aplicación práctica de mlops: machine learning operations. Tesis maestría, Universidad ORT Uruguay, Uruguay. https://dspace.ort.edu.uy/bitstream/handle/20.500.11968/4836/Material%20completo.pdf?sequence=1&isAllowed=y.

Salamanca, R. and Castro, E. (2021). Técnicas de aprendizaje automático aplicadas en los sistemas de predicción. Revista Tecnología Investigación y Academia TIA, 8(1):39–53.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Journal of Neural Networks, (61):85–117. https://www2.econ.iastate.edu/tesfatsi/DeepLearningInNeuralNetworksOverview.JSchmidhuber2015.pdf.

Tovar, M. (2019). Diseño e implementación de un aplicativo móvil para realizar la detección temprana de la enfermedad de la sigatoka negra en los cultivos de plátanos. Tesis de maestría, Universidad Tecnológica de Pereira, Colombia. https://repositorio.utp.edu.co/server/api/core/bitstreams/5de73e8c-547d-4d43-bcf9-bff99d8ea88c/content.

Valenzuela, C. (2022). Detección y clasificación de enfermedades en el tomate mediante deep learning y computer vision. Tesis de maestria, Universidad Nacional de la Plata, Argentina. http://sedici.unlp.edu.ar/bitstream/handle/10915/139770/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y.

Vela, C. (2012). La industria del software: Una experiencia de empresas, gobiernos y universidades en uruguay y ecuador. pages 21–78. FLACSO, Ecuador. https://biblio.flacsoandes.edu.ec/libros/digital/52735.pdf.

Visengeriyeva, L., Kammer, A., Bär, I., Kniesz, A., and Michael Plöd, M. (2020). Why you might want to use machine learning. https://ml-ops.org/content/motivation.

Zafra, G. (2006). Tipos de investigación. Revista Científica General José María Córdova, 4(4):13–14. https://www.redalyc.org/pdf/4762/476259067004.pdf.

Zapata, C., Carabalí, M., and Arango, L. (2022). Distribución espacial del daño de *Heilipus lauri* (coleoptera: Curculionidae) y *Stenoma catenifer* (lepidoptera: Elachistidae) en aguacate *Persea americana* var. *Hass*.

12. Glosario de Términos

Big Data: Se refiere a los datos que son tan grandes o complejos que es difícil o imposible procesarlos con los métodos tradicionales.

Machine Learning: Sub-área de la inteligencia artificial que consiste en desarrollar algoritmos con la capacidad de aprender y mejorar a través de experiencias, sin necesidad de programarlas explícitamente.

MLOps: Es una extensión de la metodología DevOps que busca incluir los procesos de aprendizaje automático y ciencia de datos en la cadena de desarrollo y operaciones para hacer que el desarrollo del ML sea más confiable y productivo.

Feature Store: Validación y preparación de los datos, donde una capa de gestión de datos para el aprendizaje automático que permite compartir y descubrir funciones y crear canalizaciones de aprendizaje automático más eficaces.

Código de fuente: Colección de líneas de texto, escritas en un lenguaje de programación, que guían el proceso de ejecución de un programa. Estas instrucciones, que son comprensibles por humanos, están redactadas por un programador.

Model Registry: Modelo requerido para los atributos se les asigna un clave para distinguirlos de los demás registros. Relación (el vínculo que define la dependencia entre varias entidades).

Pipeline: Tuberías virtuales se crean para segmentar los datos y, de este modo, incrementar el rendimiento de un sistema digital.

Plagas: Cualquier ser vivo que resulta perjudicial para otro ser vivo de interés para el ser humano.