

Universidad ORT Uruguay

Facultad de Ingeniería

Machine learning en producción
Obligatorio

Autores:

Gonzalo de León - 158545

Martin Ljubicic - 154630

Joaquín Oldan - 159579

Docentes:

Federico Zaiter

Matías Sorozabal

2023

Table of Contents

I. Introducción	5
A. Contexto del estudio.....	5
B. Objetivos	5
C. Metodología de investigación	6
II. Fundamentos de Machine Learning	7
A. Definición de Machine Learning	7
B. Tipos de algoritmos de Machine Learning	7
C. Importancia del Machine Learning en la producción	8
III. Arquitectura de la solución	9
A. Recopilación de datos	9
B. Análisis de Datos.....	9
C. Tratamiento de datos.....	10
D. Modelo.....	10
E. Producción (Frontend)	10
IV. Preparación de los datos	12
A. Scrapy	12
B. Exploración de datos.....	14
Balanceo de dataset.....	14
Correlación entre features y target	15
C. Limpieza y preprocesamiento de datos	17
D. Selección de características	18
V. Modelado y entrenamiento	19
A. Selección de algoritmos de Machine Learning.....	19
B. Adaptar datos al modelo	20
C. División de datos	21
D. Entrenamiento del modelo.....	22
Arquitectura del modelo:.....	22
DataLoading y Data augmentation:	23
Carga de datos:	23
Inicialización de pesos y sesgos:	23
Definición de la función de pérdida y algoritmo de optimización:	23
Ciclo de entrenamiento:	24
Evaluación del rendimiento:	24

Versionado de Experimentos y Modelos	25
Grid Search.....	25
VI. Implementación en producción	27
A. Consideraciones de infraestructura	27
FASTAPI	29
Consultas HTTP	29
GRADIO	32
B. Integración del modelo en un sistema de producción.....	33
DOCKER	33
C. Ejemplo de uso	35
VII. Desafíos y recomendaciones	37
A. Desafíos comunes en la implementación de Machine Learning en producción	37
B. Recomendaciones para superar los desafíos	38
VIII. Conclusiones.....	39
A. Resumen de los objetivos alcanzados	39
B. Contribuciones del estudio	39
C. Áreas para futuras investigaciones	39
D. Tabla de cumplimiento.....	41

Glosario

API	
Application Programming Interface.....	
APP	
Application	
ASGI	
Asynchronous Server Gateway Interface.....	
E2E	
End to End	
GUI	
Graphical User Interface	
ML	
Machine Learning.....	
VM	
Virtual Machine.....	

I. Introducción

A. Contexto del estudio

El Machine Learning surge como una necesidad de estudiar los datos obtenidos de cualquier ámbito para obtener información relevante para problemas planteados o para mejorar sistemas. La toma de decisiones automatizada, la automatización de procesos aporta mucho valor agregado en diferentes ámbitos.

Como se pudo observar en las presentaciones realizadas por el equipo de diversos *papers*, existen cada vez más casos de uso exitosos en el área, en la presentación realizada por los integrantes del grupo sobre Michelangelo, se expuso que no solo es importante los resultados directos del aprendizaje, sino la metodología aplicada para llegar a ellos. Esta herramienta proporciona buenos resultados y ayuda a organizar los grupos de trabajo.

La automatización es vista como una oportunidad de mejora para las organizaciones y permitirá redirigir sus esfuerzos a trabajos de mejora constante de estas metodologías, logrando mejorar sus procesos.

En este trabajo se buscará incursionar en estas metodologías para la implementación de un producto que pueda ser llevado a producción haciendo hincapié más en el E2E que en sus diferentes partes.

B. Objetivos

El objetivo general del proyecto consiste en lograr un E2E de un sistema que obtenga datos de una página web de venta de propiedades y que sea capaz de aprender a predecir un atributo de estas, en este caso se apunta a poder distinguir la categoría del inmueble según su valor.

En nuestro caso, esta variable está directamente relacionada con el precio, aunque se podría llegar a pensar en una aplicación de categorías de “valor” en función de las necesidades de quien busca adquirir la vivienda. Esto ya es un desafío más ambicioso y no será el foco del presente trabajo.

Obtener un modelo que logre analizar imágenes y datos tabulares de los elementos de estudio, si bien no hay un enfoque sobre obtener los mejores resultados, si se buscara un mínimo desempeño en esta etapa del modelo, que sirva para obtener resultados bastante aproximados del problema.

Análisis de datos, comprensión de cuales son más relevantes y cuales no, cuales aportan más al target que se busca aprender para luego inferir con buena precisión en datos de testeo.

Desempeño del sistema, velocidad y escalabilidad, un buen sistema debe ser capaz de proporcionar una buena respuesta ante múltiples consultas.

Evaluar el impacto de la solución completa en cuanto a su utilidad real y la aplicabilidad a un caso real de estudio, si bien se puede considerar un objetivo secundario, sería bueno tener una referencia a su utilidad, por su valor agregado a algún proceso determinado para un caso de estudio puntual o general.

Comprender que mejoras se pueden proporcionar a futuro en la aplicación para apuntar a un mercado de clientes reales y su posibilidad de venta.

C. Metodología de investigación

El enfoque del trabajo será con fines académicos, no se analiza un caso de estudio real, aunque luego pueda extrapolarse el trabajo, cambiando la información tratada para atender a un cliente. Se destaca igualmente que, si bien no está enfocado a un cliente, puede ser útil lo realizado en el caso de uso para cierto marco de público objetivo.

El estudio apunta a analizar información de propiedades de la página www.gallito.com.uy, para aprender a clasificarlas en clases de interés dependiendo su precio. Se recopilará información, se analizará y se mostraran resultados de forma amigable al usuario.

Los datos obtenidos se analizan para utilizarlos de la mejor manera en su inyección en el modelo, comprender que tan significativo resultara, y realmente si aportara valor. Datos faltantes también serán analizados para comprender la necesidad de completarlos o no, y así no influir en el aprendizaje del modelo y también luego en la inferencia en producción.

La información recopilada en el aplicativo se considera de interés general y acceso público, y no se exponen datos personales de las cuentas que realizan las publicaciones o datos que puedan ser relacionados a ellas, es necesario revisar el dataset de imágenes y cuidar esta información para completar un trabajo dentro de los marcos éticos adecuados.

El estudio buscara respetar reglas de análisis de datos establecidos, cuidar el data leakage y el training-serving skew que puede repercutir de gran manera en los resultados y desempeños del aplicativo en producción.

Si bien la información de la fuente es obtenida en un momento determinado del mercado de inmuebles, en definitiva, muy relacionada al momento actual del mercado, no se tienen datos históricos, se deberá cuidar las conclusiones a partir de estos datos sesgados en cierta manera.

II. Fundamentos de Machine Learning

A. Definición de Machine Learning

El Machine Learning, o aprendizaje automático, es una rama de la inteligencia artificial que se centra en el desarrollo de algoritmos y técnicas que permiten a las computadoras aprender de manera automática a través de la experiencia y los datos, sin ser programadas explícitamente. En lugar de seguir instrucciones específicas, los algoritmos de Machine Learning son capaces de reconocer patrones, identificar correlaciones y tomar decisiones o realizar predicciones basadas en los datos con los que han sido entrenados.

Este enfoque de aprendizaje automatizado ha demostrado ser eficaz en una amplia gama de aplicaciones, desde el reconocimiento de voz y la visión por computadora hasta la detección de fraudes y la personalización de recomendaciones. El objetivo del Machine Learning es permitir que las máquinas “adquieran conocimiento” y mejoren su rendimiento a medida que se enfrentan a nuevos datos, lo que lo convierte en una herramienta poderosa para abordar problemas complejos y tomar decisiones basadas en información.

B. Tipos de algoritmos de Machine Learning

Existen diversos tipos de algoritmos y enfoques de Machine Learning que se utilizan para abordar diferentes tipos de problemas y tareas. A continuación, algunos de los tipos más comunes de enfoques de Machine Learning:

- **Aprendizaje supervisado (Supervised Learning):** En esta técnica, se entrena al modelo utilizando un conjunto de datos de entrenamiento que incluye ejemplos etiquetados. El objetivo es que el modelo aprenda a predecir o clasificar nuevas instancias en función de las etiquetas conocidas. Los árboles de decisión están en esta categoría, muy explicativos y prácticos.
- **Aprendizaje no supervisado (Unsupervised Learning):** En el aprendizaje no supervisado, no se proporcionan etiquetas en los datos de entrenamiento. El objetivo es descubrir patrones o estructuras ocultas en los datos. Un ejemplo de estos son el clustering.
- **Aprendizaje por refuerzo (Reinforcement Learning):** Se basa en un proceso de toma de decisiones en el que un agente interactúa con un entorno y recibe recompensas o castigos en función de sus acciones. El objetivo es que el agente aprenda a tomar decisiones que maximicen las recompensas a largo plazo. Los ejemplos más característicos se atribuyen a juegos.
- **Aprendizaje profundo (Deep Learning):** El aprendizaje profundo utiliza redes neuronales artificiales con múltiples capas para aprender características y representaciones complejas de los datos. Estas redes pueden realizar tareas de reconocimiento de imágenes, procesamiento de lenguaje natural y traducción. Las redes CNN para tratamiento de imágenes son un ejemplo.

Este trabajo se enfocará en Deep Learning y Aprendizaje supervisado, realizando modelos de CNN + MLP para el tratamiento de imágenes y datos tabulares provistos de los datos adquiridos.

C. Importancia del Machine Learning en la producción

Llevar un algoritmo de Machine Learning a producción es de suma importancia debido a que es el paso final para hacer uso efectivo de los modelos desarrollados y obtener beneficios reales en entornos prácticos. Como puntos importantes hay que destacar:

- Utilización efectiva de los modelos: Se puede aprovechar plenamente el valor y el potencial de los modelos desarrollados. Los modelos pueden aplicarse en tiempo real y en escala, lo que permite automatizar tareas, tomar decisiones y mejorar la eficiencia operativa.
- Toma de decisiones automatizada: Se puede tomar decisiones automatizadas basadas en datos y análisis. Esto puede conducir a una toma de decisiones más rápida, precisa y consistente, evitando errores humanos y mejorando la eficacia general de los procesos.
- Optimización de recursos: Se pueden aprovechar eficientemente los recursos disponibles. Los modelos pueden ayudar a optimizar procesos, mejorar la asignación de recursos, reducir el desperdicio y aumentar la productividad.
- Mejora continua: Permite recopilar datos en tiempo real y retroalimentar los resultados a los modelos de Machine Learning. Esto facilita la mejora continua y la capacidad de adaptación a medida que los modelos se enfrentan a nuevos escenarios y cambios en los datos de entrada.

¡Hay aspectos que hay que cuidar, para lograr un buen sistema!

- Estabilidad y rendimiento: Es esencial garantizar que el algoritmo funcione de manera estable y tenga un rendimiento óptimo en el entorno de producción. Esto implica considerar aspectos como el tiempo de respuesta, la escalabilidad, la eficiencia de los recursos y la capacidad de manejar grandes volúmenes de datos en tiempo real.
- Integración con la infraestructura existente: La integración adecuada del algoritmo en la infraestructura existente es crucial. Esto incluye aspectos como la compatibilidad con los sistemas y tecnologías utilizados, la gestión de datos de entrada y salida, y la seguridad y privacidad estos.
- Monitoreo y mantenimiento: Un algoritmo de Machine Learning en producción debe ser monitoreado de forma continua para asegurar que siga funcionando correctamente y proporcionando resultados precisos. También se debe realizar un mantenimiento regular para actualizar el modelo, reentrenarlo con nuevos datos y abordar posibles desviaciones.
- Evaluación y medición de resultados: Es importante establecer métricas de evaluación para medir el desempeño y el impacto del algoritmo en producción. Esto permitirá evaluar su eficacia, identificar posibles mejoras y demostrar el valor generado a los interesados.

En resumen, la implementación exitosa de un algoritmo de Machine Learning en producción es crucial para aprovechar plenamente los beneficios de los modelos desarrollados y lograr una toma de decisiones automatizada y eficiente.

III. Arquitectura de la solución

Se implementa una solución de varias etapas como se puede observar en el siguiente diagrama:

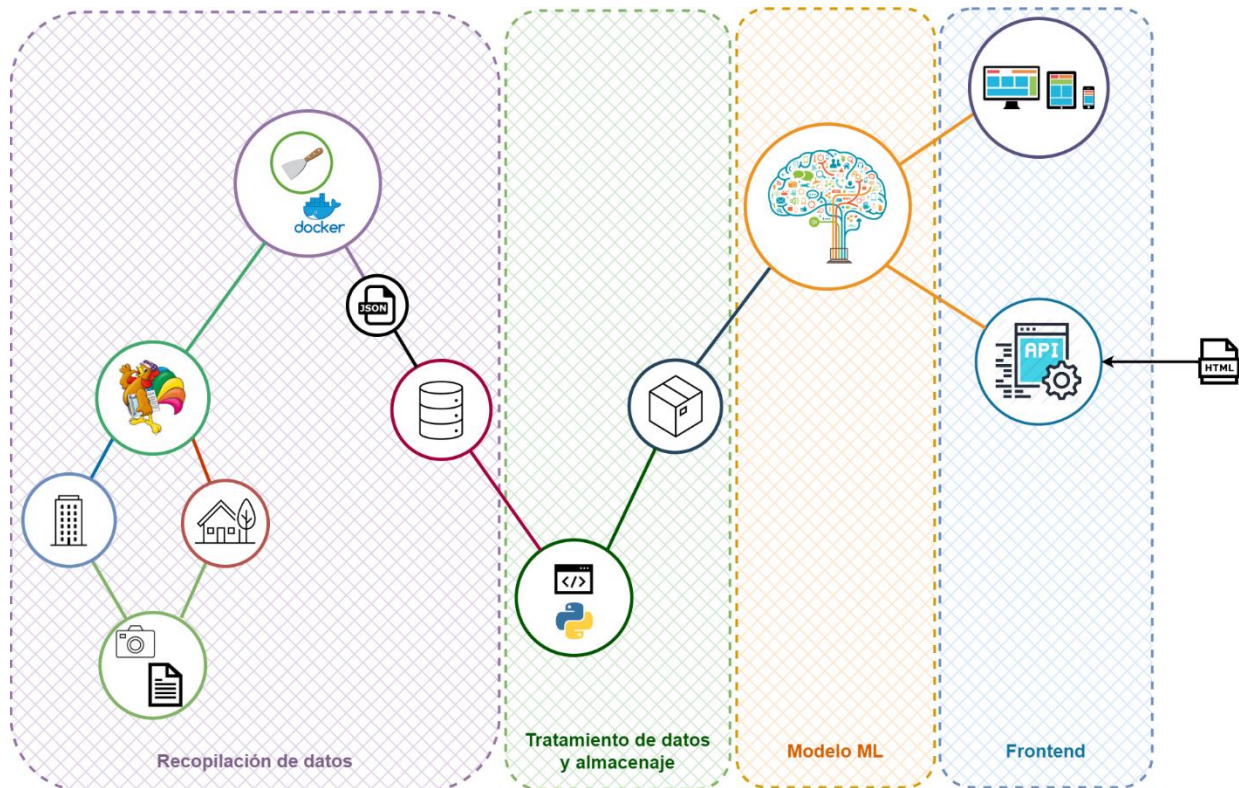


Ilustración 1 - Diagrama de arquitectura

Se dará una breve descripción de cada parte y luego en siguientes capítulos se entrará en detalle.

A. Recopilación de datos

Se implementa una etapa de adquisición de datos utilizando una herramienta de scraper que descarga datos de inmuebles de la página de elgallito.com.uy. Esta etapa se implementa para obtener todos los datos necesarios para el entrenamiento del modelo, y luego en siguientes secciones nos centraremos en analizar esa data y poder entender cuál será útil y cual no.

B. Análisis de Datos

Se ejecuta un análisis sobre los datos obtenidos. Si bien el campo de las propiedades y las variables que afectan su precio son de conocimiento común en cualquier persona que haya embarcado en una compra o alquiler de inmueble, se busca un enfoque más estadístico.

El objetivo del análisis es el de poder caracterizar las cualidades del dataset y entender la relación entre los tipos de datos obtenidos, los predictores y el target, para definir cual predictor es importante para entrenamiento e inferencia del modelo.

C. Tratamiento de datos

La data obtenida es analizada, adaptada en formato, y aceptada o descartada en función de su utilidad. Esta etapa se basa en el conocimiento obtenido de la fase anterior.

D. Modelo

El modelo es implementado en un lenguaje de programación universal y en un formato que puede llevarse a varias plataformas para su funcionamiento. Esta etapa se encargará de obtener la clasificación problema planteada en el desafío a partir de su entrenamiento, evaluación, y posterior puesta en producción.

Se utiliza Weights and Biases, que es una plataforma utilizada en el campo del aprendizaje automático y la ciencia de datos para el seguimiento, la visualización y la colaboración en experimentos y modelos. Proporciona herramientas y servicios que permiten a los investigadores y científicos de datos realizar un seguimiento detallado de los experimentos, registrar métricas, visualizar gráficos y compartir resultados con otros miembros del equipo.

Con Weights and Biases, se puede realizar un seguimiento de los hiperparámetros utilizados en los modelos, registrar métricas de rendimiento durante el entrenamiento, visualizar gráficos y tablas para analizar los resultados.

E. Producción (Frontend)

La puesta en producción del modelo, en otras palabras, servir el modelo, implica que el mismo pueda ser utilizado por los usuarios finales para realizar inferencia. En este caso, a partir de una imagen y de ciertos datos tabulares, obtener el valor del inmueble discretizado en cuatro clases.

- Menor a 100.000 US\$
- Entre 100.000 y 200.000 US\$
- Entre 200.000 y 300.000 US\$
- Más de 300.000 US\$

Es posible acceder al modelo mediante un API, ejecutando consulta GET y POST de HTTP. La inferencia se puede hacer de a un dato por vez o en modalidad batch (más de un dato por consulta).

También es posible acceder mediante un navegador web. En esta modalidad solo se pueden hacer predicciones únicas (una por vez), pero utilizando una GUI.

Para facilitar la puesta en producción del modelo, se optó por empaquetar la aplicación en un contenedor, que como se desarrollara más adelante, está compuesta de diferentes elementos.

Para poder tener acceso a la app por medio de internet, se optó por alojar el contenedor en una VM en la nube de Azure.

IV. Preparación de los datos

A. Scrapy

Para recopilar datos se utiliza Scrapy, es un framework de web scraping de código abierto y alto rendimiento para Python. Proporciona una manera eficiente y escalable de extraer datos de sitios web de manera automatizada.

Con Scrapy, se pueden definir spiders que navegan por las páginas web, extraen datos estructurados y los almacenan en el formato deseado. Su arquitectura modular y su potente motor de procesamiento paralelo permiten realizar scraping de manera eficiente, incluso en sitios web grandes y complejos.

Es importante aclarar que para este obligatorio no se trabajó sobre el código provisto en clase, sino que se comenzó con un spider desde cero.

Como se comentó previamente, para la recolección de datos se usa la fuente www.gallito.com.uy, dentro de la cual se navega a través de la selección de casas y apartamentos a la venta. Para restringir la búsqueda de estas páginas se recurre a los parámetros de “start_urls” y “rules”.

The screenshot shows a web interface for a real estate listing. At the top, there's a navigation bar with links: Inmuebles / Apartamentos / Venta / Montevideo / Parque Batlle. Below this, there are tabs for Galeria, 360°, Video, and Ubicación. The main content area features a large image of a modern apartment building with a 'gallito.com.uy' watermark. To the right of the image, the text reads 'Departamento - Parque Batlle' and 'PISO SEIS. Con exoneración fiscal 100%'. Below this, a price tag shows 'U\$S 141.000'. Further down, there's a form titled 'Consultar al anunciante' with fields for 'Tu email', 'Tu teléfono', and 'Tu consulta', followed by a 'Consultar' button. At the bottom, there's a row of icons representing different features: Apartamento, Venta, Parque Batlle, 1 dormitorio, 1 Baños, and 51 Mts. The footer includes the text 'Inmobiliaria ciudad-inmobiliaria' and a 'Ver Teléfono' button.

Ilustración 2 – Datos relevados

De cada una de las páginas de propiedad, se extrajeron varios valores, en este ejemplo:

Parameter	Value
date	2023-06-10 19:11:31.357829
ID	1
title	Departamento – Parque Battle
Department	Montevideo
Location	Parque Battle
Rooms	1 dormitorio
Bathrooms	1 Baños
building_type	Apartamento
area	51 Mts
price	U\$D 141.000
url	https://www.gallito.com.uy/linda-casa-una-planta-proximo-ruta-8-complejo-a-45-inmuebles-21294865
images	https://imagenes.gallito.com/1024x768/48932647.jpeg
Foreign_id	21294865

Si bien no todos los valores son usados en este proyecto, algunos como `foreign_id` y `date` son tomados para versiones futuras en las cuales se corra periódicamente la colección y re-entrenamiento del modelo para mantenerlo actualizado y combatir el data-drift.

Proceso de Extracción

Para extraer los valores se usan tanto XPath como CSS selector, encontrándose el segundo método mucho más poderoso. Se guarda también el path a las imágenes de cada publicación.

Como posteriormente se guardarán por separado la metadata y las imágenes, es necesario mapear cada ítem extraído (metadata correspondiente a cada publicación) con su imagen. Para esto se opta por agregar un identificador único (ID) a cada ítem y se modifica la clase `customImagePipeline` para que las imágenes se guarden con dicho ID como nombre.

El resultado de la corrida es un archivo csv y una carpeta con las fotografías descargadas.

Corrida

En la corrida se limitó el crawling a 50.000 páginas porque se entiende que proporciona datos suficientes para un entrenamiento. El resultado de la corrida es un archivo csv y una carpeta con las fotografías descargadas, todo guardado en memoria local del sistema donde se ejecuta el proceso de scraping.

Mirando a la nube

Con un fin de poder correr todo el proceso de extracción y limpieza de datos en la nube se configuró el proceso para que una vez terminado el crawl se envíen las fotos y data a un container en Azure para ser consumido después por la limpieza de datos. Además de esto se containerizó también todo el crawler de manera de poder ser ejecutado desde Docker.

Tristemente debido a restricciones de tiempo no fue posible llevar todo el resto del proceso a la nube por lo que quedará para posibles futuros desarrollos.

B. Exploración de datos

Se realiza una exploración de datos tabulares para entender la relación y correspondencia entre variables. Lo que resulta más importante es revisar esto contra el target de este desarrollo, para lograr entender que tanta relación con las feature existe.

En cuanto a las imágenes, es un poco más desafiante realizar una exploración, analizar de que tipo son, de que partes del inmueble, si son ilustrativas o reales, se necesita más apoyo de un evaluador y no es tan directo como para las tabulares.

Balanceo de dataset

Para comenzar, se evalúa que tan balanceadas estén las clases target. Les recordamos que trabajaremos con 4 clases target para clasificar rangos de precio: ("Menor o igual 100.000", "Entre 100.000 y 200.000", "Entre 200.000 y 300.000" y "Mayor a 300.000"). La distribución de datos en clases es la siguiente:

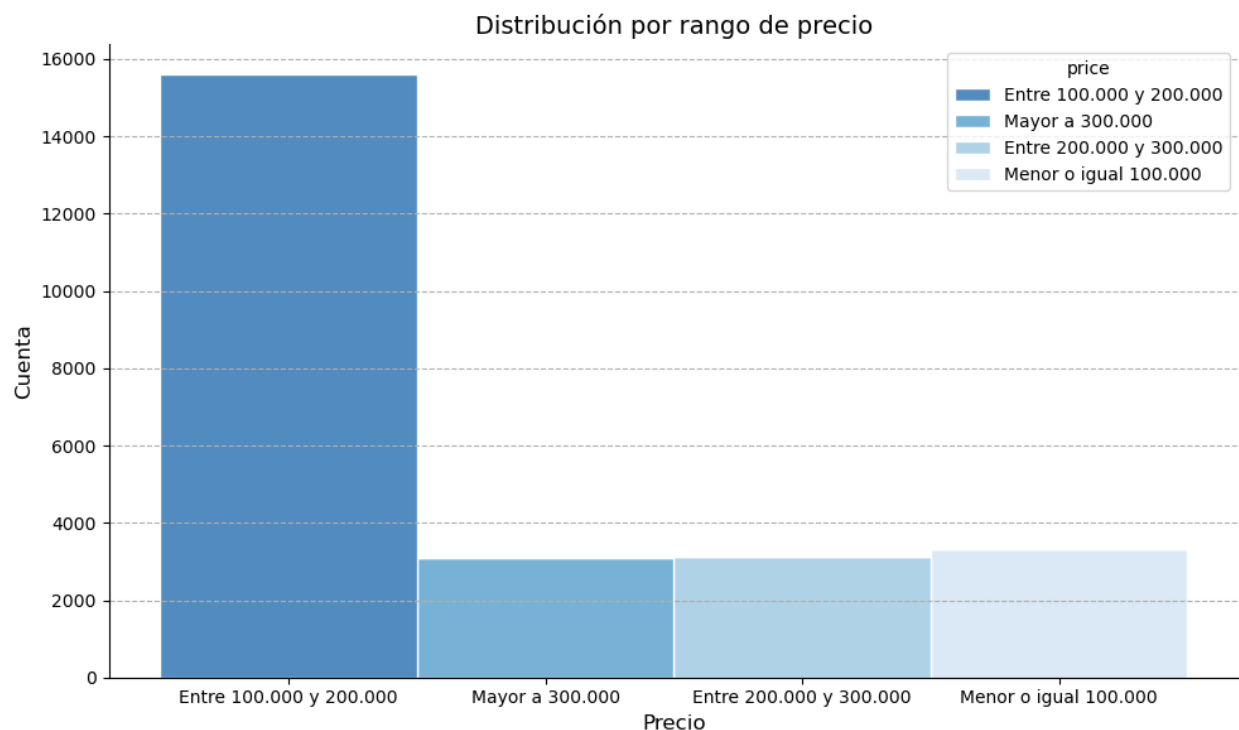


Ilustración 3 – Histograma por clase

Clase	Total muestras	Porcentage	Relación con mayor clase (pesos optim)
Entre 100.000 y 200.000	15600	62.15%	0.07
Menor o igual 100.000	3308	13.17%	0.31
Entre 200.000 y 300.000	3112	12.39%	0.32
Mayor a 300.000	3079	12.26%	0.32

Se puede ver una que le distribución de datos no es uniforme. La clase de “Entre 100.00 y 200.000” representa el 62% de la población de datos, superando a las otras en un factor de más o menos 1 a 5.

Para compensar esto en el entrenamiento del modelo se agregarán al optimizador los pesos descriptos en la última columna.

Correlación entre features y target

Para evaluar si existe razón para descartar alguno de los predictores obtenidos se comienza haciendo un mapa de calor de matriz de correlación:

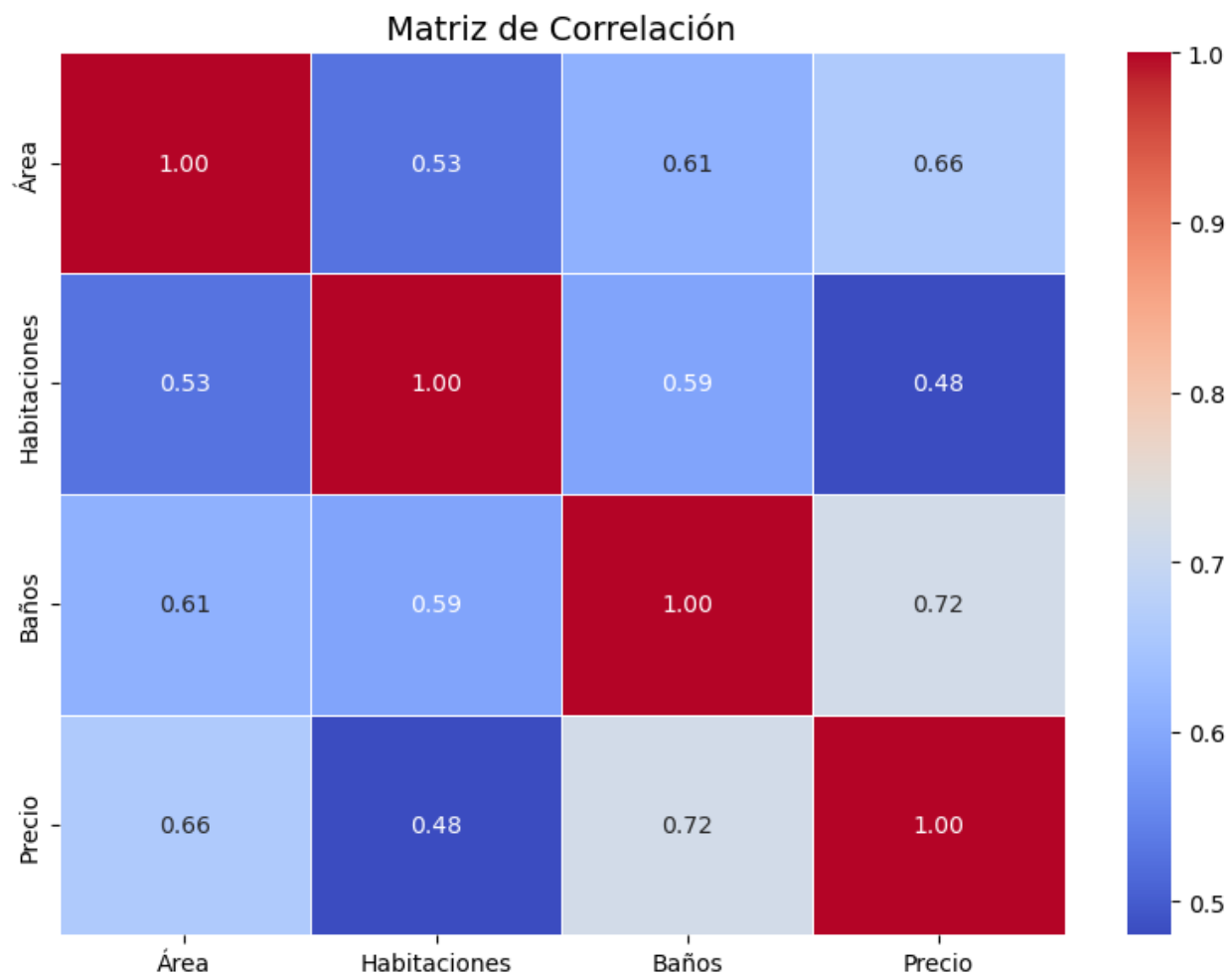


Ilustración 4 – Matriz correlación

De la misma se puede apreciar que los predictores numéricos (ya que departamento y localidad no pueden soportar este análisis) están todos correlacionados con el precio. Es cierto también que todos los predictores están inter-correlacionados. Como es de esperarse en propiedades, más área lleva a más cuartos, usualmente.

Esto lleva a la conclusión de utilizar todos los features obtenidos para el modelo a entrenar y poner en producción.

Departamento/Localidad

Para continuar con el análisis hacemos un scatterplot de los predictores, dejando afuera Departamento/Localidad.

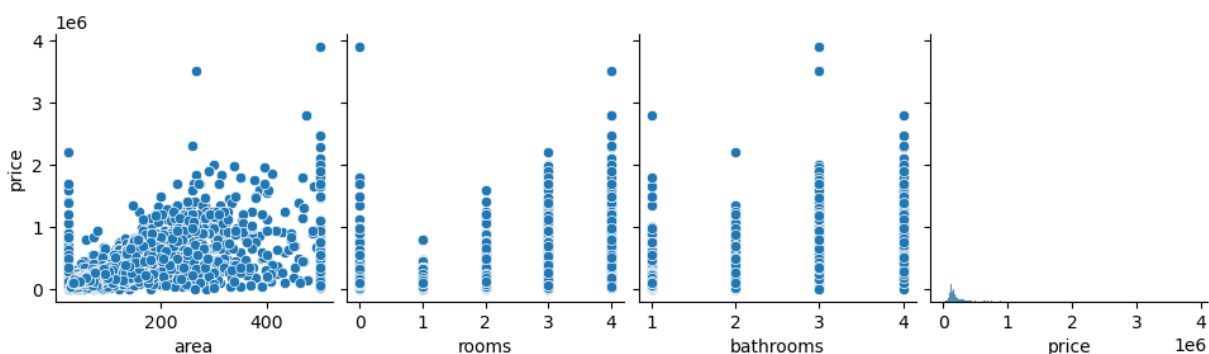


Ilustración 5 – Relación precio con más tabulares

No se puede apreciar de las gráficas una fuerte relación entre el target precio y los predictores. Si, hay una relación de correlación, pero no parece ser muy fuerte. Por otro lado... si discriminamos por Departamento/Localidad de Montevideo/Pocitos, vemos que la correlación se vuelve más fuerte:

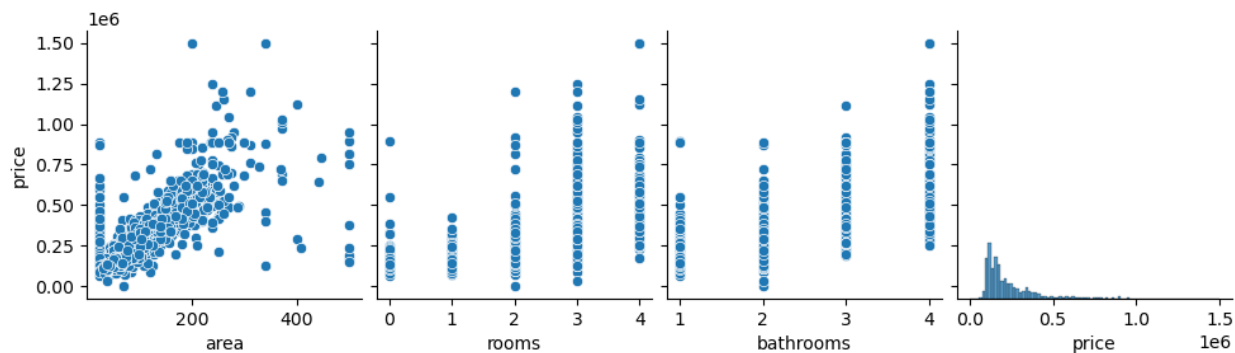


Ilustración 6 – Relación precio con tabulares en Pocitos

Esto es particularmente visible en la gráfica de área en relación con precio, particularmente porque su naturaleza continua (no discreta) es más suave a la vista.

Se concluye por esta fase del análisis que es muy importante también incluir Localidad y Departamento.

Conclusiones:

Las conclusiones del análisis es que todos los predictores tomados (área, cuartos, baños, departamento, localidad) proveen información para predecir precio final y por ende todos han de ser usados.

Se detecta un desbalance en la representación de las clases por lo que será importante compensar este desbalance con alguna técnica, como el uso de pesos en el optimizador.

C. Limpieza y preprocesamiento de datos

Como es sabido, la limpieza de datos es una parte importante y que consume mucho tiempo en un proyecto de este estilo. Para dejar los datos listos para ser procesados y consumidos por el modelo (que consume tanto datos tabulares como imágenes) se ejecuta una serie de validaciones y transformaciones.

Se optó por crear un proceso que ejecute toda la limpieza de todos los datos en batch usando librerías pandas y PIL. Esto con la idea de poder automatizarlo en el futuro a través de alguna herramienta de orquestación.

Los criterios son los siguientes:

Dato	Criterio
bathrooms	Se permiten valores de 1 a 4 baños, todo valor mayor a 3 es 4.
rooms	Los monoambientes ser califican como cero cuartos.
area	El área es mayor a 25 y menor a 500.
building_type	Se define 0 para apartamento y 1 para casa
department/location	Se le asigna identificador, ver abajo
price	Se efectúa una limpieza para pasar a int (valores de dólares, puntos)

Department Location

Para Departamento y Location se trabaja de una forma particular. Se unifican los departamentos zona en único valor, concatenando sus valores (por ejemplo Montevideo y Pocitos queda como montevideopocitos).

Primero se creó una lista de todos los valores posibles de Department y Location. Luego se creó una nueva lista que concatene estas variables (por ejemplo Montevideo y Malvin pasaría a ser montevideomalvin). Esta nueva lista tiene para cada posible dupla queda mapeada a un índice en la lista. Representará al dato conjunto departmentlocation.

Toda la lógica de department/location se nuclea en la clase LocationManager.

Al final de proceso se tiene un nuevo dataset limpio y validado con los datos como el siguiente:

id	area	rooms	bathrooms	building_type	department_location	price
0	166.0	3.0	1	1	16.0	107000

Imágenes

Las imágenes se transforman todas a 256x192px para dejarlas listas para ser ingeridas por la red neuronal.

Datos Faltantes

Durante el proceso, en cualquier línea que no se pueda ejecutar la limpieza o que esta no cumpla con los requerimientos dispuestos, se borrará la línea y posteriormente la foto que la acompaña.

Training-Serving Skew

Para evitar el training-serving skew se utilizan las mismas funciones de limpieza de datos para training y para inferencia.

Data Leakage

Para prevenir el data leakage se crea un único dataset en el cual se genera la partición usando herramientas de PyTorch (que entendemos confiables). Para previr leaking en las imágenes se usó una única imagen por item de la página y se aseguró que cada metadata de los item apunte a la imagen correcta.

D. Selección de características

De los inmuebles se seleccionan determinadas características que luego serán los inputs del modelo.

- Imagen de portada (1024x768)
- Localidad donde se ubica el inmueble (varias, más de 200)
- Cantidad de Habitaciones (0, 1, 2, 3, 4[o más])
- Cantidad de Baños (1,2,3[o más])
- Metros cuadrados (25 a 500)
- Si es casa o apartamento (uno u otro)

La imagen de referencia será la de portada del inmueble, se cree necesario tener un input de este estilo ya que puede aportar información sobre estado, luminosidad, y otros. En una primera aproximación se tomará solamente una imagen, pero podrá establecerse que sea una cantidad mayor. De todas maneras, esa cantidad debería ser fija, para poder luego de limpiar datos, adaptarlos para la etapa de modelado.

La localidad, es un dato importante ya que hay mucha variación en costos debido a este parámetro, hay que trabajar este dato para que logre capturar todas las opciones posibles y luego analizar la mejor forma de inyectarlo en el modelo.

Cantidad de habitaciones, cantidad de baños, Metros cuadrados, son valores numéricos que de cierta forma dan un escalado en los precios de los inmuebles, con una relación que podría considerarse lineal.

Si es casa o apartamento, es un dato no menor, que dará de cierta forma un diferencial entre estas 2 categorías.

V. Modelado y entrenamiento

A. Selección de algoritmos de Machine Learning

Se utiliza un modelo de Deep Learning basado en una CNN para analizar las imágenes, y luego será input de una MLP con el adicional de los datos tabulares comentados en la selección de características.

Red Neuronal Convolutiva (CNN) es un tipo de red neuronal especializada en el procesamiento de datos estructurados en forma de matrices, como imágenes. Se compone de capas convolucionales que aplican filtros para extraer características relevantes de las imágenes y capas de pooling que reducen la dimensionalidad. Las capas convolucionales y de pooling se combinan con capas completamente conectadas para realizar la clasificación o predicción final. Las CNN son altamente eficientes para el procesamiento de imágenes debido a su capacidad para reconocer patrones espaciales y aprender características jerárquicas.

Un Perceptrón Multicapa (MLP) es una red neuronal artificial compuesta por múltiples capas de neuronas, incluyendo una capa de entrada, una o más capas ocultas y una capa de salida. Cada neurona en una capa está conectada a todas las neuronas de la capa anterior y de la siguiente, formando una red profunda. Las neuronas en cada capa realizan cálculos basados en funciones de activación no lineales para transformar los datos de entrada. Los MLP son utilizados en problemas de clasificación y regresión, y pueden aprender relaciones no lineales entre las variables de entrada y la variable objetivo.

El problema para resolver de clasificación en clases de los inmuebles lleva a la utilización de una métrica de error para este tipo de modelo, la Cross Entropy Loss, esta es una función de costo comúnmente utilizada en problemas de clasificación en el campo del Machine Learning. Es especialmente útil cuando se trabaja con modelos de clasificación que producen probabilidades como salida, como las redes neuronales.

Esta función mide la discrepancia entre la distribución de probabilidades predicha por el modelo y la distribución de probabilidades verdaderas (etiquetas) de los datos. Cuanto más diferentes sean estas dos distribuciones, mayor será el valor de la función de pérdida.

En términos más técnicos, para un problema de clasificación binaria, se calcula como la suma de los productos de las etiquetas verdaderas y el logaritmo de las probabilidades predichas de la clase positiva, junto con los productos de las etiquetas complementarias y el logaritmo de las probabilidades predichas de la clase negativa. En problemas de clasificación multiclase, la función se generaliza para incluir todas las clases.

El objetivo del entrenamiento de un modelo es minimizar este parámetro, ajustando los pesos de las neuronas en la red neuronal para mejorar la concordancia entre las predicciones y las etiquetas verdaderas.

Esta no es la única solución para este tipo de problemas, se pudo haber implementado funciones más visuales de clasificación como los árboles de decisión, estos son un tipo de algoritmo de aprendizaje automático utilizado en problemas de clasificación y regresión. Se basan en la estructura de un árbol en el que cada nodo representa una característica o atributo, las ramas representan las posibles decisiones o reglas basadas en los valores de las características, y las hojas representan las etiquetas de clasificación o los valores de regresión.

En el aprendizaje de árboles de decisión, el algoritmo busca encontrar las divisiones o particiones óptimas en los datos de entrenamiento, con el objetivo de maximizar la pureza o la homogeneidad de las clases dentro de cada subconjunto resultante. Estas divisiones se basan en reglas de decisión que se definen mediante pruebas en los valores de las características.

Durante la construcción del árbol, el algoritmo evalúa diversas métricas de calidad para determinar cuál atributo y qué umbral de separación proporcionan la mejor división en cada nodo. Algunas de las métricas comunes utilizadas son la ganancia de información, el índice Gini y la reducción de error cuadrático.

Una vez construido el árbol, se puede utilizar para hacer inferencia, siguiendo el camino desde la raíz hasta una hoja, tomando decisiones basadas en las pruebas de características en cada nodo. En problemas de clasificación, la etiqueta de la hoja final representa la clase predicha, mientras que, en problemas de regresión, el valor de la hoja final es la predicción numérica.

Los árboles de decisión son atractivos debido a su interpretabilidad y facilidad de comprensión. Además, son resistentes al ruido en los datos y pueden manejar características de diferentes tipos. Sin embargo, los árboles de decisión también pueden ser propensos al sobreajuste y pueden tener dificultades para representar relaciones complejas entre variables.

Para mitigar estas limitaciones, existen variantes de árboles de decisión, random forests y gradient boosting trees, que combinan múltiples árboles para mejorar la precisión y generalización del modelo.

B. Adaptar datos al modelo

Se trabajan sobre los datos determinados para ser utilizados en el modelo, previamente discutidos. En una red MLP los datos no numéricos deben ser transformados o codificados de manera adecuada para poder ser utilizados como entradas en la red. Esto se debe a que las neuronas en una red MLP trabajan con valores numéricos y realizan operaciones matemáticas en ellos. La mayoría de este trabajo fue logrado en la capa de data cleaning.

Existen tres datos que son del tipo de clases que requieren otro tratamiento, estos son el target precio y los predictores Departamento-Localidad y Casa/Apartamento.

Se usan las siguientes técnicas:

- One-hot encoding. Esta técnica se utiliza cuando los datos no numéricos son variables categóricas, es decir, tienen un conjunto limitado de categorías distintas. Consiste en convertir cada categoría en un vector binario de longitud igual al número de categorías posibles y se crea

una columna binaria separada para cada categoría, y solo una de las columnas contendrá un valor de 1, mientras que las demás contendrán valores de 0.

- **Embedding:** Es una técnica más avanzada que asigna vectores de valores numéricos de baja dimensión a cada categoría. Esta técnica es particularmente útil cuando los datos no numéricos tienen una gran cantidad de categorías. Los embeddings son aprendidos durante el proceso de entrenamiento de la red, lo que permite que la red capture relaciones complejas entre las categorías y los resultados.

Es importante tener en cuenta que la elección de la técnica de codificación depende del tipo de datos no numéricos y del problema específico que se esté abordando. Cada enfoque tiene sus ventajas y consideraciones, y es necesario evaluar cuál es el más adecuado para el caso en particular.

Se trabajará estas 2 características de la forma:

- **Precio:** Se utiliza One-Hot encoding para discriminar entre las 4 clases en las cuales se separa este target.
- **Casa o Apartamento:** Se utiliza One-Hot encoding, utilizando 2 variables, para indicar de que categoría es el dato. Pero se dejará un único valor, que podrá tomar valor 0 o 1.
- **Departamento/Localidad:** Debido a la cantidad de clases aquí, más de 200, se opta por incluir una capa de Embedding en el modelo, que se irá entrenando a la par de la CNN y MLP.

C. División de datos

La división de conjuntos de datos es una etapa fundamental en el desarrollo de modelos de Machine Learning. Consiste en separar el conjunto de datos en diferentes subconjuntos, con el propósito de utilizarlos de manera adecuada durante las diferentes fases del proceso, como entrenamiento, validación y prueba. Es importante hacerlo con cuidado para evitar data leakage.

La división típica implica tres conjuntos de datos principales:

- **Conjunto de entrenamiento (Training set):** Este conjunto de datos se utiliza para entrenar el modelo. Contiene ejemplos etiquetados que el modelo utilizará para aprender los patrones y relaciones entre las características y las etiquetas objetivo. Se utiliza generalmente alrededor del 70-80% del conjunto de datos total.
- **Conjunto de validación (Validation set):** Después de entrenar el modelo, es esencial evaluar su rendimiento en datos no vistos para ajustar los hiperparámetros y realizar ajustes adicionales. El conjunto de validación se utiliza para este propósito. Permite medir la precisión y el rendimiento del modelo en datos desconocidos y realizar comparaciones entre diferentes configuraciones de modelo. Suele ser del 10-15% del conjunto de datos total.
- **Conjunto de prueba (Test set):** El conjunto de prueba es utilizado para evaluar el rendimiento final y la generalización del modelo después de que se han realizado todas las etapas de entrenamiento y ajuste. Es importante que el conjunto de prueba sea independiente y no se utilice en ninguna etapa previa de desarrollo del modelo, ya que proporciona una evaluación

objetiva y realista de su capacidad para generalizar a datos no vistos. Suele ser del 10-15% del conjunto de datos total.

Es importante destacar que la división de conjuntos de datos debe realizarse de manera aleatoria y estratificada en problemas de clasificación, para garantizar que las clases estén representadas de manera equilibrada en cada subconjunto.

En esta implementación se utiliza para el entrenamiento una división de 80/20 entre datos de entrenamiento y validación. Y se entrena el modelo y observando las métricas calculadas sobre ambos dataset separados para evaluar cuando es un entrenamiento suficiente.

D. Entrenamiento del modelo

Arquitectura del modelo:

El modelo se implementa usando la librería Pytorch. Está formado por una primer capa CNN, seguida de una capa MLP que no solo toma los outputs de la primera capa, sino que los concatena con los datos tabulares de la información de cada ítem. Usando tanto imágenes como datos tabulares como entradas.

La capa CNN está conformada por 4 capas con las siguientes características:

- Kernels de 4x4, Stride 2, Padding 1.
- 3 canales al ingreso (RGB)
- Canalizaciones intermedias de 64, 128, 256, 64.
- Capa de batchnormalization entre convolucionales.

La capa MLP está conformada por 4 capas con las siguientes características:

- La primera capa tendrá como entrada la salida de la CNN más los tabulares.
- Se hace un embedding de la locación
- Las capas irán reduciendo la dimensionalidad desde la entrada a 1024, 512, 128, 64, hasta 4 que será la salida del modelo representando las clases de clasificación.

```
=====
Layer (type:depth-idx)          Param #
=====
MLP_Model                       --
-Embedding: 1-1                  1,040
-Conv2d: 1-2                     6,144
-BatchNorm2d: 1-3                256
-Conv2d: 1-4                    524,288
-BatchNorm2d: 1-5                512
-Conv2d: 1-6                    524,288
-BatchNorm2d: 1-7                256
-Conv2d: 1-8                    65,536
-BatchNorm2d: 1-9                64
-Conv2d: 1-10                   4,096
-BatchNorm2d: 1-11              16
-Conv2d: 1-12                   128
-BatchNorm2d: 1-13              2
-Linear: 1-14                    21,504
-Linear: 1-15                   524,800
-Linear: 1-16                    65,664
-Linear: 1-17                     8,256
=====
```

```
Linear: 1-18 260
=====
Total params: 1,747,110
Trainable params: 1,747,110
Non-trainable params: 0
=====
```

DataLoading y Data augmentation:

Se realiza data augmentation a nivel de imágenes de entrenamiento. Para implementarlo se hace uso de la clase Dataset y Transformation de PyTorch. La manipulación de estas clases permite discriminar entre train y test dataset, haciendo transformaciones en solo el primero.

No se encuentra mayor mejora en la performance del modelo como causa de la augmentación de datos. Se estima que esto es porque es muy difícil para la red encontrar información de importancia en las imágenes.

Entendemos que para que la red pueda sacar valor de las imágenes un entrenamiento y una red distinta serían necesarios.

Carga de datos:

Los datos procesados son cargados para poder entrenar y validar el modelo.

El formato de datos a cargar tendrá la siguiente forma:

[Imagen, [metros, habitación, baños, apto o casa, locacion(emb_dim = 4)], label]

Luego se crearán data loaders que agruparán los datos de manera aleatoria en batches de tamaño solicitado para ser luego ser inyectados en el entrenamiento y validación del modelo.

Inicialización de pesos y sesgos:

Se utiliza un criterio de inicialización aleatoria de pesos y sesgos para el modelo.

Definición de la función de pérdida y algoritmo de optimización:

Se utiliza Adam como optimizador y Cross Entropy Loss como función de error. De esta última se habló en capítulos anteriores, ahora veremos alguna característica del optimizador.

Adam es un algoritmo de optimización popularmente utilizado en el campo del aprendizaje automático para ajustar los parámetros de los modelos durante el proceso de entrenamiento.

Este algoritmo combina elementos del descenso de gradiente estocástico (SGD) con técnicas de estimación de momentos adaptativos. A diferencia del SGD tradicional, Adam mantiene un conjunto de promedios móviles de los gradientes y de las segundas derivadas de los parámetros del modelo. Estos promedios móviles adaptativos se utilizan para ajustar las tasas de aprendizaje de cada parámetro individualmente.

Las principales ventajas de Adam incluyen:

- Eficiencia: Adam combina las ventajas de los algoritmos de descenso de gradiente estocástico y los métodos adaptativos, lo que lo hace eficiente en términos de tiempo de convergencia y requisitos de memoria.
- Adaptabilidad: Adam adapta las tasas de aprendizaje de manera individualizada para cada parámetro, lo que permite un ajuste más preciso y rápido.
- Robustez ante gradientes dispersos: Adam tiene un buen rendimiento en problemas con datos dispersos o con gradientes de baja variabilidad.

A pesar de sus ventajas, Adam también tiene algunas consideraciones como la Sensibilidad a hiperparámetros (tiene muchos para ajustar) o el Tamaño del batch(muy chico o muy grande).

En general, Adam es una opción popular para el entrenamiento de modelos de aprendizaje automático debido a su eficiencia y adaptabilidad.

Ciclo de entrenamiento:

Se utiliza la herramienta W&B para registrar varios entrenamientos variando hiperparámetros como será la cantidad de epochs, el Learning rate, y otros.

Abarcar varias configuraciones de hiperparámetros, evita caer en problemas en el algoritmo de aprendizaje como puede ser problemas de gradiente y aprendizaje lento.

La tasa de aprendizaje es un parámetro crítico en los algoritmos de optimización utilizados en el aprendizaje automático. Define la velocidad a la que un modelo de ML ajusta sus parámetros durante el proceso de entrenamiento.

En términos sencillos, la tasa de aprendizaje determina cuánto deben actualizarse los parámetros del modelo en respuesta a los gradientes calculados durante el entrenamiento. Una tasa de aprendizaje alta permite actualizaciones más grandes en cada iteración, lo que puede llevar a una convergencia más rápida. Por otro lado, una tasa de aprendizaje baja realiza actualizaciones más pequeñas y gradualmente se acerca al óptimo, pero puede llevar más tiempo converger.

Seleccionar una tasa de aprendizaje adecuada es crucial, ya que puede influir en el rendimiento y la eficiencia del modelo. Si la tasa de aprendizaje es demasiado alta, el modelo puede oscilar y no alcanzar un óptimo, o incluso divergir. Por otro lado, si la tasa de aprendizaje es demasiado baja, el modelo puede tardar mucho tiempo en converger o quedar atrapado en un mínimo local subóptimo.

Evaluación del rendimiento:

Al final de cada época o después de completar el entrenamiento, se evalúa el rendimiento del modelo en el conjunto de validación. Se estudian parámetros como son la Loss y la accuracy.

Versionado de Experimentos y Modelos

Se utiliza la aplicación W&B para el seguimiento de todos los entrenamientos realizados y poder sobre el final realizar un chequeo de resultados de buena forma. Quedando guardada en la herramienta los hiperparámetros seleccionados, los resultados del entrenamiento y una copia de los pesos del modelo final producido por tal entrenamiento.

De esta manera tenemos un repositorio centralizado que nos permita hacer un seguimiento de los experimentos para la correcta selección del que tenga la mejor performance.

Abajo se puede apreciar una captura de W&B usado para comparar dos entrenamientos:

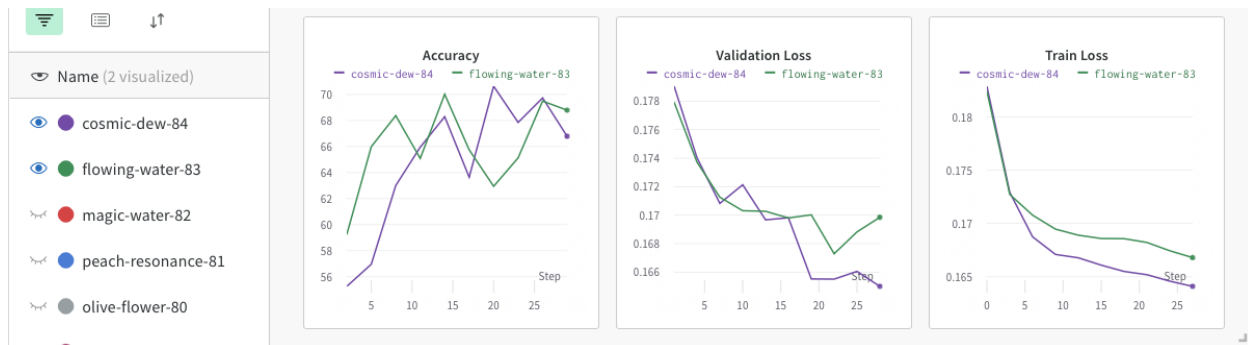


Ilustración 7 – W&B

Aquí se puede apreciar como un modelo queda guardado en W&B dentro de la corrida

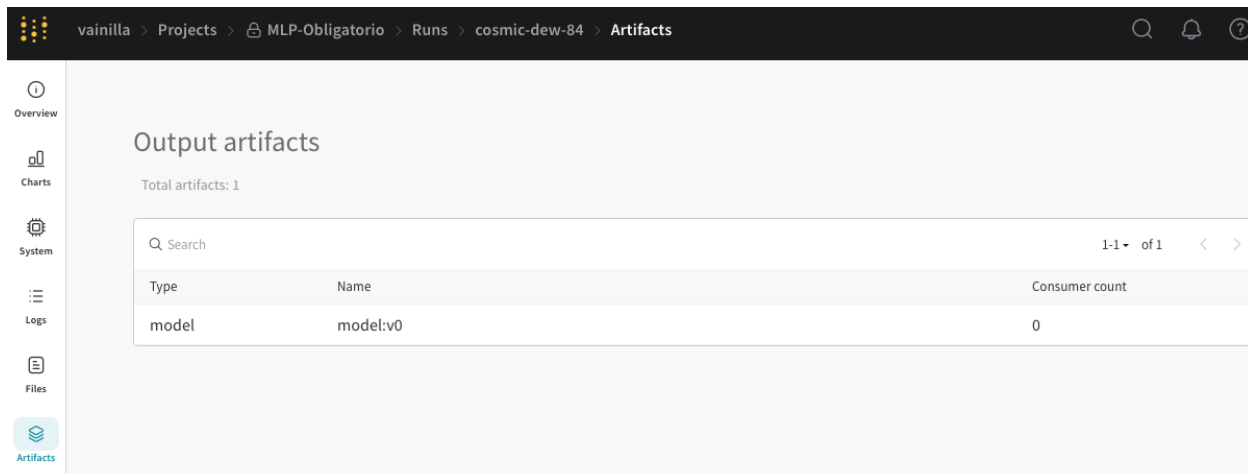


Ilustración 8 – Modelo en W&B

Grid Search

Si bien es posible hacer un tuning de hiperparámetros al comparar diferentes corridas en W&B se quiso implementar un grid search.

Tristemente esto llevó al fracaso. Primero se intentó hacer con W&B ya que se venía trabajando con la plataforma y se tenía experiencia positiva con los “sweeps” (la forma de llamarle a los barridos de parametros). Pero la experiencia positiva había sido trabajando en Keras, en donde fue casi un plug and play. Después de mucho intento nos fue imposible hacerlo andar en Pytorch, es muy difícil de debug usando W&B ya que no muestra los errores, pero entendemos que hay algún tipo de problema en la función forward del Modelo cuando se corre en cpu y un problema con CUDA cuando se corre en GPU.

Se intentó entonces con otra librería Ray Tune, la misma es un poco más bajo nivel que W&B pero opera de una manera similar y ofrece casi las mismas prestaciones, menos las bellas gráficas y centralización de datos en un único repositorio. Esta pudo correr, pero por alguna razón no logramos que corra todas las pruebas que necesitamos y las que corren parecen tener muy malos resultados.

VI. Implementación en producción

A. Consideraciones de infraestructura

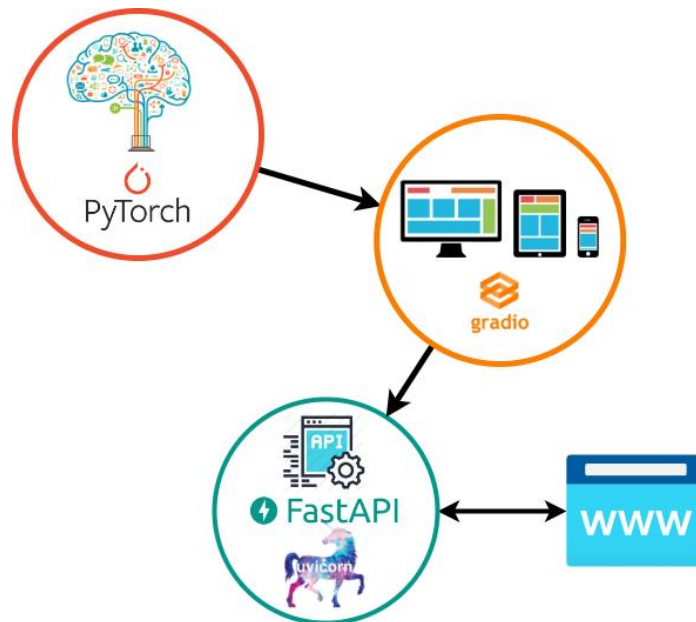


Ilustración 10 - Componentes principales de la APP

Como se mencionó previamente, la APP tiene que ser capaz de exponer el modelo al usuario final, se optó por usar dos mecanismos. El primero de ellos es mediante una API desarrollada con el framework Fastapi, Esta solución está pensada para poder integrar el modelo con otros sistemas que consuman las predicciones y no como una interfaz de usuario amigable a los humanos.

El segundo mecanismo, implica una interfaz gráfica que nos permite interactuar con el modelo y generar predicciones de una forma muy simplificada. Se utilizó el framework Gradio para la construcción de la GUI.

Debido a que la APP es una colección de diferentes clases, es necesario logear con fines de troubleshooting la traza que desencadena una acción a través de todos los componentes involucrados. Se implementa la biblioteca logging de Python para efectuar esta tarea.

El log generado, es global a toda la aplicación, pero se registra que modulo está generando la entrada, de esta forma se obtiene en un solo archivo la traza de todas las acciones.

Ejemplo de log

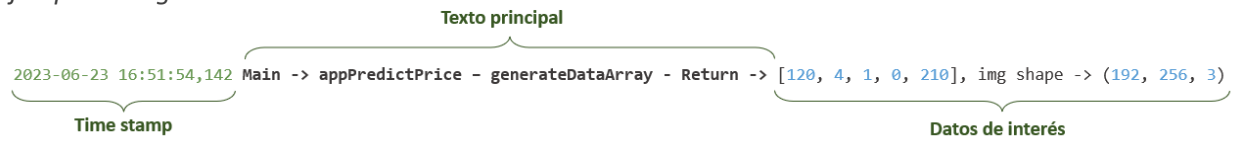


Ilustración 11 – Log example

En el ejemplo, el módulo `appPredictPrice(Gradio)`, la función `generateDataArray` retorno los predictores para hacer la inferencia.

La siguiente imagen, es un diagrama simplificado de los modulos de la APP. Solo se explicarán los elementos más importantes

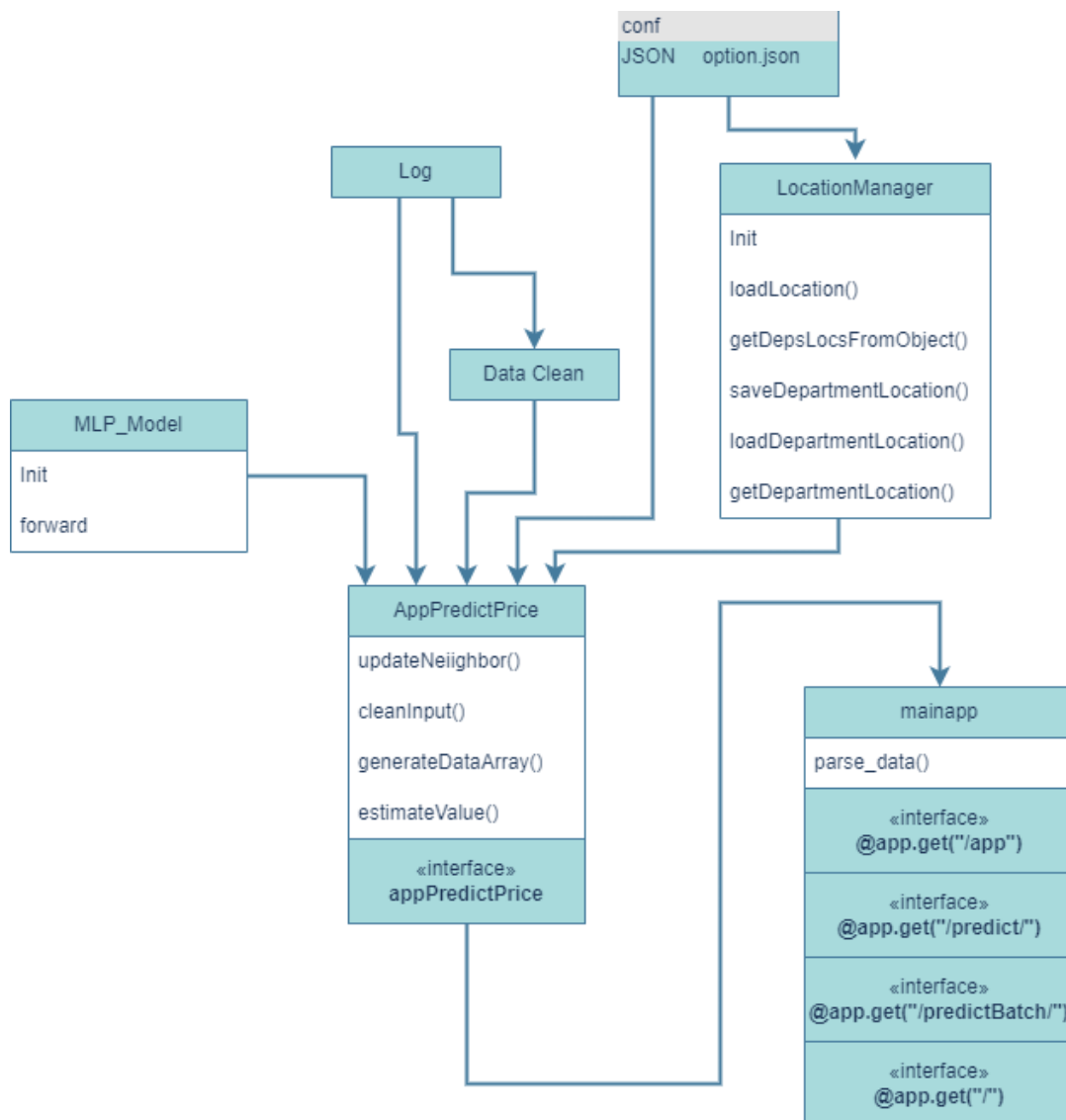


Ilustración 12 – UML simplificado

FASTAPI

Es un framework de desarrollo basado en Python que permite desarrollar API performantes y robustas fáciles de implementar.

En este caso, la API es el núcleo de toda la APP denominada “Mainapp”, además de exponer 3 métodos para hacer consultas http, se encarga de servir a la página web.

La APP es hosteada por Uvicorn, un servidor WEB del tipo ASGI implementado también en Python. Mainapp escucha las consultas http en el puerto 80.

Consultas HTTP

Consulta HTTP	Atributos de entrada	Respuesta
GET /	Ninguno	Página web
GET /app	Ninguno	String con leyenda de la app
POST /predict/	String, Imagen	String con la predicción
POST /predictBatch/	Strings, Imágenes	Strings con las predicciones
GET /docs	Ninguno	Página web con documentación del la API

GET /

Al invocar GET /, que es lo mismo que indicar la URL en un navegador en donde se aloja la APP, se despliega la página WEB.

Fastapi se integra de forma muy simple con Gradio, solo hace falta montar la app grafica (appPredictPrice).

```
Try:
    app = gr.mount_gradio_app(app, appPredictPrice, path=CUSTOM_PATH)
    log.info("Main -> %s - Mount Gradio APP",__name__)
except:
    log.info("Main -> %s - Error to mount Gradio APP",__name__)
```

GET /app

Cumple el rol de un healthcheck, simplemente responde con un string fijo indicando que es una API.

POST /predict/

Método invocado para ejecutar predicciones únicas, requiere un body que contenga la imagen y los datos tabulares de la casa o apartamento que se quiere inferir su precio.

En esta primera versión no se hace control de los datos insertados o si el body de la consulta es correcto.

El método espera un body multipart compuesto por un string (**data**) y la imagen (**image_file**).

Llamamos item al objeto multipat.

Data está compuesto por la concatenación de los datos tabulares separados por “;” en un orden específico.

Data = ["area;rooms;bathrooms;bultdingType; departmentLocation,location"]

Para el caso de un apartamento en Pocitos de 120 m^2 , 3 dormitorios y 1 baño el string seria:

`["120; 3; 1; 0; Montevideo, Pocitos"]`

Image_file es el archivo con la imagen. Solo se aceptan en formato jpg y no hay un mecanismo de validación implementado.

Ejemplo de request:

```
http://cuantovale.ddns.net/predict/
```

Ejemplo de body:

```
-H 'accept: application/json' \  
-H 'Content-Type: multipart/form-data' \  
-F 'data=120;4;1;Apartamento;Montevideo;Pocitos' \  
-F 'image_file=@Casa.jpg;type=image/jpeg'
```

La respuesta que se obtiene es un string con la pseudo probabilidad de a que clase pertenece

```
"message": "[{'Menos de 100K': 2.1662407423406194e-09, 'Entre 100K y 200K': 0.0004232079954817891, 'Entre 200K y 300K': 0.9978289008140564, 'Más de 300K': 0.0017478929366916418}]"
```

POST /predictBatch/

Request similar a predict, solo que acepta más de un item para hacer inferencias.

Si se ejecuta el request con un solo **item**, el resultado es análogo al del request predict. Al igual que en la consulta anterior, no se hace validación de los datos o el body. Si el usuario por ejemplo carga ítems incompletos, el sistema simplemente fallara.

El método espera un body multipart compuesto por un string (**datas**) y las imagenes (**image_files**).

En **datas** el sistema espera los datos tabulares de cada **item**, separando cada conjunto de datos por “,”

$Datas = [data_1, data_2, \dots, data_n]$

Ejemplo de request:

```
http://cuantovale.ddns.net/predictBatch/
```

Ejemplo de body:

```
'http://cuantovale.ddns.net/predictBatch/' \  
-H 'accept: application/json' \  
-H 'Content-Type: multipart/form-data' \  
-F 'datas=120;4;1;Apartamento;Montevideo;Pocitos,460;5;5;Casa;Salto;Salto' \  
-F 'images_files=@apartamento.jpeg;type=image/jpeg' \  
-F 'images_files=@Casa.jpg;type=image/jpeg'
```

Si la consulta es correcta, el sistema retorna tantas predicciones como ítems contenga.

```
"message": "[{'Menos de 100K': 1.5268423192438263e-09, 'Entre 100K y 200K': 0.0002934600634034723, 'Entre 200K y 300K': 0.997514009475708, 'Más de 300K': 0.002192482817918062}, {'Menos de 100K': 0.0, 'Entre 100K y 200K': 0.0, 'Entre 200K y 300K': 1.3158192580010032e-42, 'Más de 300K': 1.0}]"
```

[GET /docs](#)

Fastapi proporciona de forma automática documentación de la API, además de una WEB que permite testear la APP sin necesidad de hacer las consultas HTTP mediante herramientas como Postman o curl.

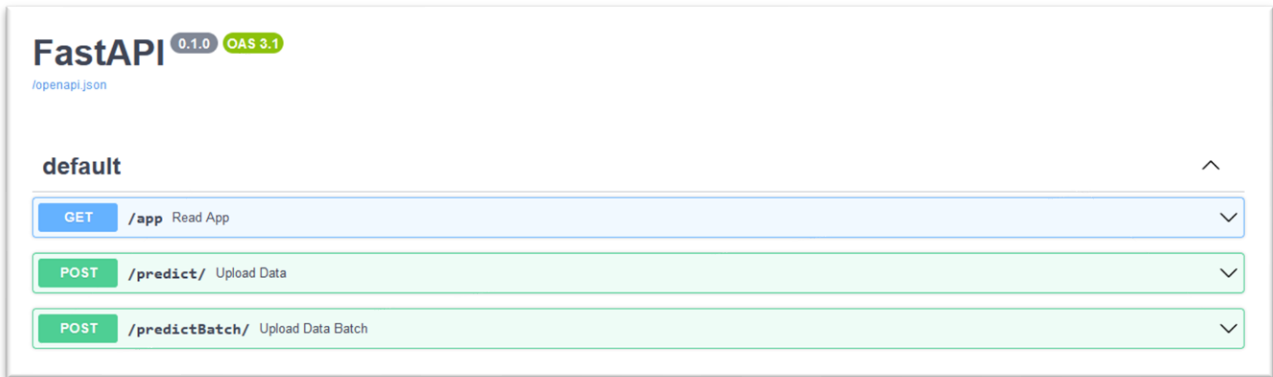


Ilustración 13 – Autodocumentación Fastapi

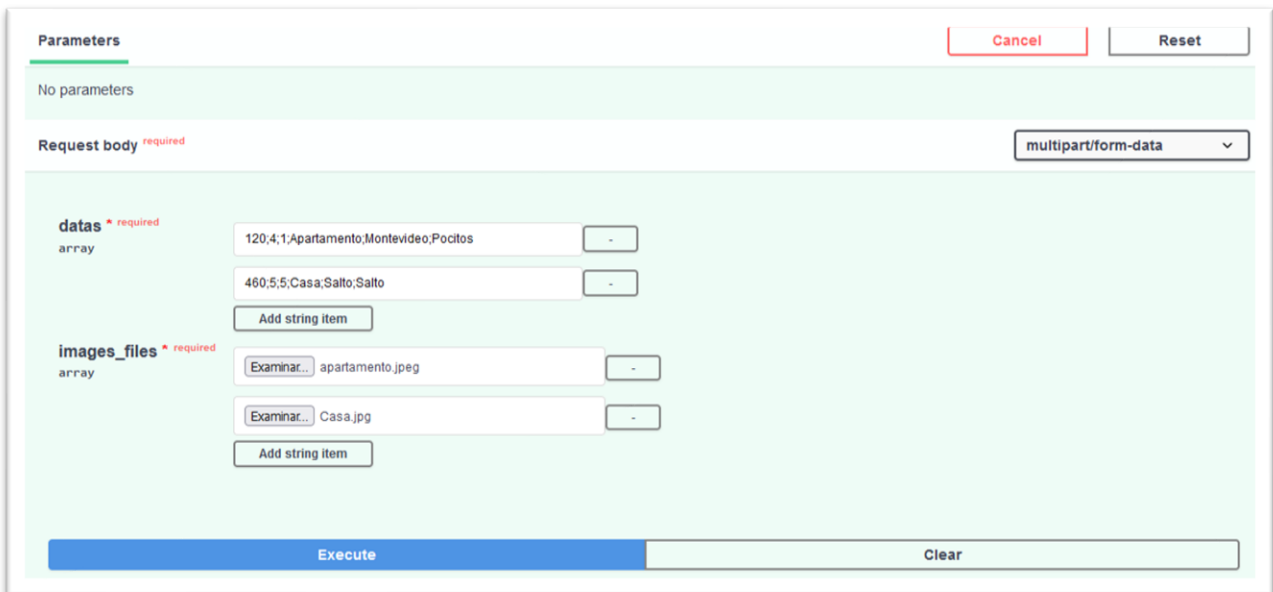


Ilustración 14 – Request – URL/predictBatch/



Ilustración 15 – Salia – URL/predictBatch/

GRADIO

Es un framework que permite generar GUI de forma muy simple y rápida para interactuar con modelos de ML

Como ya se mencionó, la GUI desarrollada en Gradio es invocada por Fastapi en el momento que el usuario accede a la página del modelo <http://cuantovale.ddns.net>.

Aparte de Gradio, se evaluó utilizar Streamlit, pero no convenció al equipo debido a que se intuía que era más compleja de utilizar.

Para evitar incorporar control de datos, se creó una interfaz de usuario basada en “botones”, de esta forma se evita nativamente que el usuario pueda ingresar datos en formatos incorrectos.

La GUI por default tiene datos pre-cargados en los drop-down y radio box que evitan que el usuario omita algún campo. Además, los predictores como departamento y barrio, solo se pueden seleccionar de una lista(drop-down) y las variables numéricas están acotadas a ciertos umbrales utilizados para el entrenamiento del modelo.

El único dato que se puede omitir es la imagen, pero se hace una validación que retorna un mensaje de error en tal caso.

Es posible actualizar las listas de locaciones y barrios disponibles modificando un archivo json (options.json) que es cargado por el sistema al inicio.

Ilustración 16 – WEB_APP

B. Integración del modelo en un sistema de producción

DOCKER

Es una plataforma que permite desplegar aplicaciones dentro de un contenedor con todas las dependencias necesarias para la ejecución de esta.

La construcción de contenedor se hace utilizando dos archivos, el Dockerfile y el requirements.txt. El primero de ellos, tiene las intrusiones de todos los elementos que componen el APP, mientras que el segundo la lista de dependencias.

Ejemplo de Dockerfile

Imagen base →	FROM python:3.11.2
Directorio raíz de la app →	WORKDIR /app/fastapi
Códigos fuentes →	# Load the sources code COPY fastapi /app/fastapi COPY gradio /app/gradio COPY dataclean /app/dataclean COPY log /app/log COPY conf /app/conf COPY modelos /app/modelos COPY dockerfiles/app/requirements.txt /app/fastapi
Instalación de dependencias →	# Install requirements RUN pip3 install -r requirements.txt
Puerto de exposición →	# Port through which the app is exposed EXPOSE 80
Instalación de dependencias →	# Runs our application when the container is started CMD ["uvicorn", "mainapp:app", "--host", "0.0.0.0", "--port", "80", "--reload"]

Ilustración 17 – Dockerfile example

Para poder crear el contenedor solo hace falta ejecutar el comando “docker build” desde el directorio raíz del proyecto (repositorio).

```
docker build -t mainapp:v1 .
```

Posterior a la creación, es necesario iniciar el contenedor

```
docker run -d -p 80:80 -t mainapp:v1
```

Como la App utiliza el puerto 80 para exponer la WEB y la API, es necesario hacer el mapeo del puerto entre el container y host.

Si el entorno no es local. Hay que permitir conexiones sobre puerto 80 del host (crear regla en el firewall).

Por último, y opcional, generar una entrada en los registros DNS, para que se pueda acceder a la APP mediante un nombre de domino, por ejemplo, usando el servicio gratuito de no-ip.

C. Ejemplo de uso

Predicción usando la GUI

Datos : Apartamento en Pocitos 2 Dormitorios, 1 baño y 56 m²

The screenshot shows a web application titled "Estime el valor de su inmueble" on the URL "cuantovale.ddns.net". The interface is for online prediction ("Predicción en línea").

Form Fields:

- Tipo de vivienda:** Radio buttons for "Casa" and "Apartamento" (selected).
- Departamento:** Dropdown menu set to "Montevideo".
- Barrio:** Dropdown menu set to "Pocitos".
- Dormitorios:** Slider set to 2 (range 0 to 5+).
- Baños:** Slider set to 1 (range 1 to 4+).
- Superficie en m2:** Slider set to 56 (range 25 to 500+).

Image Section: A large image of a modern apartment interior is shown, with a label "Image" and edit/delete icons. Below it are "Examples" of other property images.

Buttons: "Estimar valor" and "Limpiar".

Results Section: Titled "Rango de valor estimado", it shows a bar chart for the range "Entre 100K y 200K".

Rango de valor estimado	Porcentaje
Entre 100K y 200K	100%
Entre 200K y 300K	0%
Menos de 100K	0%
Más de 300K	0%

Ilustración 18 - GUI

Predicción usando la API

Datos : Casa en Punta del Este 4 Dormitorios, 3 baño y 150 m²

Nota: El orden de los datos no puede ser modificado.

data * required
 string

150;4;3;Casa;Maldonado;Punta del Este

image_file * required
 string(\$binary)

Examinar... Casa.jpg

Ilustración 19 – API input

Response body

```
{
  "message": "{'Menos de 100K': 0.7082279920578003, 'Entre 100K y 200K': 0.010602781549096107, 'Entre 200K y 300K': 0.2811688482761383, 'Más de 300K': 3.332617666274018e-07}"
}
```



Download

Ilustración 20 – API output

Ejemplo WEB en un móvil

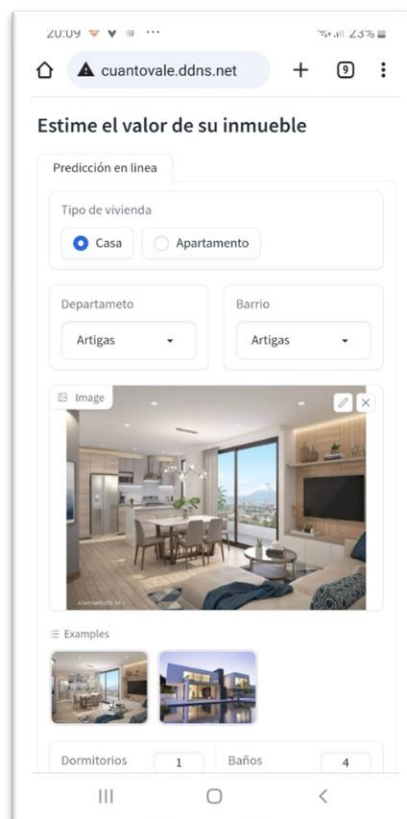


Ilustración 21 - WEB en dispositivo Móvil

VII. Desafíos y recomendaciones

A. Desafíos comunes en la implementación de Machine Learning en producción

El trabajo representa un desafío en su totalidad, poder estructurar cada tramo del pipeline y conectarlos consume tiempo y esfuerzo. Dentro de los desafíos a destacar y como influyeron en este trabajo:

- Escasez o calidad insuficiente de datos: La disponibilidad de datos de calidad y representatividad de estos se acoto a un momento determinado donde se obtuvieron, si bien existe publicaciones de más días, se ajusta a un momento del mercado. Sera necesario monitorear muy bien el modelo, e implementar un mecanismo de reentrenamiento para ajustarlo a variaciones temporales. Además, es importante notar que los datos son subidos por usuarios, por lo que pueden ser incorrectos por sesgos propios de la persona que los carga.
- Preprocesamiento y limpieza de datos: Los datos suelen requerir un procesamiento previo. Esto implica tareas como manejo de valores faltantes, normalización, codificación de variables categóricas, entre otros. En este caso se realizó un trabajo ordenado y conciso sobre los datos disponibles, hay que destacar que faltantes no había muchos; Si se necesitó realizar escalado del área dentro del modelo de metros a hectómetros, para mejorar el entrenamiento.
- Selección y extracción de características: Identificar las características más relevantes y representativas para el modelo es un desafío. No se trabajó extensivamente en este punto, se cree que en la parte de imágenes si se puede realizar un análisis más minucioso y utilizar las más representativas.
- Prevenir data leakage: Las fuentes de “Data Leakage”, como la inclusión de datos del futuro, la filtración de información a través de variables de tiempo o el uso incorrecto de datos durante la validación del modelo son aspectos importantes para cuidar. Las buenas prácticas para evitar la “Data Leakage”, las cuales fueron seguidas en el proyecto, son, separar adecuadamente los conjuntos de entrenamiento (no entreverar fotos y tabulares), evitar información del futuro, mantener la integridad de los datos en el proceso, entrenar el modelo regularmente (a futuro).
- Selección de algoritmos y ajuste de hiperparámetros: Existen una amplia variedad de algoritmos de Machine Learning, cada uno con sus propias características y requisitos. El trabajo se enfocó en un modelo de deeplearning, y si bien se implementó W&B no se trabajó exhaustivamente en el ajuste de hiperparámetros. También puede ajustarse la cantidad de capas de los modelos CNN y MLP.
- Manejo de desbalance de clases: En problemas de clasificación, es común que las clases no estén equilibradas en términos de número de instancias. Esto puede generar sesgos en el rendimiento del modelo. Técnicas de data augmentation son buenas en este punto, o la utilización de pesos en el cálculo de la función de perdida. Esto último fue implementado en este trabajo.
- Interpretación y explicabilidad del modelo: A medida que los modelos de Machine Learning se vuelven más complejos, como las redes neuronales profundas, su interpretación y explicabilidad pueden ser difíciles. Este punto realmente complejo no fue abordado en demasía, trabajar más sobre el modelo podría dejarse en un trabajo a futuro.

- Actualización y mantenimiento del modelo: Los modelos de Machine Learning suelen ser iterativos y requieren actualización y mantenimiento continuo. Como se comentó más arriba, los datos fueron obtenidos en un momento puntual y no se asegura cuanto puedan ser útiles.

B. Recomendaciones para superar los desafíos

En resumen, la implementación exitosa de Machine Learning conlleva una serie de desafíos y consideraciones importantes. Algunos de los desafíos comunes incluyen la disponibilidad y calidad de los datos, la selección del algoritmo adecuado, la optimización de hiperparámetros, el manejo de desbalance de clases y la interpretación de los resultados.

Para superar estos desafíos, es recomendable recopilar y generar más datos, realizar un análisis exhaustivo de los datos, aplicar técnicas de preprocesamiento avanzadas, experimentar con diferentes algoritmos, realizar una búsqueda sistemática de hiperparámetros, considerar técnicas de manejo de desbalance de clases, utilizar técnicas de interpretación de modelos y mantener una monitorización regular del modelo en producción.

En este trabajo no se profundiza mucho en ninguna de las etapas por separado, sino que se prioriza a nivel general un desarrollo completo del pipeline.

Es importante tener en cuenta que cada proyecto de Machine Learning es único y puede requerir enfoques y soluciones específicas.

VIII. Conclusiones

A. Resumen de los objetivos alcanzados

Se logra el objetivo general del proyecto: E2E de un sistema que obtenga datos de una página web de venta de propiedades y que sea capaz de aprender a predecir un parámetro de estas, categorizar el inmueble según su valor.

Se obtiene un modelo que logra analizar imágenes y datos tabulares de los elementos de estudio, si bien no hay un enfoque sobre obtener los mejores resultados, se llega a un mínimo desempeño en esta etapa, y resulta muy útil como aplicación.

Se consigue analizar datos de buena forma, estudiando su distribución, comprendiendo posible problema de desbalance y ajustándose a ello.

El sistema logra implementarse en producción, siendo de muy buena respuesta, y no existiendo diferencias en la forma con que fue entrenado.

Una buena solución completa en cuanto a su utilidad real y la aplicabilidad a un caso real de estudio, si bien se puede considerar un objetivo secundario, sería bueno tener una referencia a su utilidad, por su valor agregado a algún proceso determinado para un caso de estudio puntual o general.

B. Contribuciones del estudio

Las contribuciones principales de este trabajo apuntan a el aprendizaje de llevar un modelo a producción para los integrantes del grupo. La documentación trabajada quedara disponible en un GIT publico pudiendo ser de utilidad para siguientes trabajos.

A nivel académico se considera un trabajo muy bueno, siendo necesario de revisar y afinar ciertos detalles para poder ser llevado como un proyecto comercial a un cliente. Mas adelante se detallará en que se puede trabajar para lograr mejora el producto final.

C. Áreas para futuras investigaciones

Como se comentó, el objetivo final del trabajo era de obtener un sistema E2E con todas sus partes trabajadas y no centrarse en una etapa puntual.

En ese sentido, se puede ir recorriendo estas, y comentando que se puede mejorar, trabajar, para poder llevar esta implementación experimental a una solución comercial y que ajuste a la necesidad de algún cliente potencial.

En lo relacionado con los datos, se puede mejorar la distinción de imágenes en categorías, o evaluar en su aporte real sobre la valoración del inmueble. Como se comentó antes, no se realiza análisis exploratorio sobre estas, y eso es muy probable que sume sesgos luego sobre el modelo. Los inmuebles más caros tienen mejores imágenes en general, más cantidad de imágenes, y esto puede influir.

En los datos tabulares la limitación está en que las categorías habitaciones y baños están topeadas en cierto máximo, podría trabajarse estos datos para estimar por las imágenes si estos valores debieran ser otros según planos o algún otro detalle en las imágenes.

Apuntando al modelo, se puede trabajar mucho más, analizar cambios en el modelo, mejorar el data augmentation; pero lo más importante es indagar en el procesamiento de imágenes obtener mejor información. Posiblemente mejorar el modelo para analizar más imágenes, colocando varias categorías de estas, foto de frente, foto de habitación principal, foto de baño, entre otras.

Dentro de la búsqueda de hiperparámetros puede incluirse estas características del modelo, y sobre eso lograr mejores resultados de 40Loss y Accuracy.

Las categorías establecidas para los precios podrían mejorarse y aumentarlas para tener más información. Podría también adaptarse el modelo, cambiando la función de Loss, para realizar en vez de una clasificación multiclase, una regresión y poder estimar un valor de precio numérico.

A futuro es necesario realizar un seguimiento del rendimiento, revisar métricas específicas para evaluar la calidad y eficacia del modelo a lo largo del tiempo. Los datos de entrada deberán monitorearse, la necesidad de incluir algún parámetro nuevo no tenido en cuenta y que realmente es relevante.

Deberá reentrenarse el modelo cada cierto tiempo, y mediante un procedimiento automatizable lograr la actualización en producción. Para esto lo mejor es actualizar los datos de la plataforma relevada, para contemplar las informaciones más recientes.

Uvicorn permite tener un control de reload continuo, por lo tanto, solo bastaría con cargar las nuevas actualizaciones al contenedor (o que se descarguen de forma periódica) y tocar ("touch") el archivo mainapp, para que se recargue la actualización en caliente, sin necesidad de crear un nuevo contenedor.

En cuanto al modelo en producción, varias mejoras son posibles apuntando a una mejor eficiencia de recursos y automatización. Lograr instanciar un contenedor en un ambiente Cloud para contenedores, con integración Kubernetes para poder tener elasticidad en el crecimiento y alta disponibilidad sería una muy buena primera mejora.

La API podría ser mejorada al incluir limpieza de datos para controlar mejor la ingesta del usuario. Si bien se buscó implementar la clase basemodel para filtrar los datos y el body, se complicó al realizarlo con imágenes y tabulares.

En cuanto a la GUI, sería conveniente que los valores de los umbrales puedan ser modificados desde un archivo de configuración, al igual que las listas desplegables, sin necesidad de modificar el código fuente.

D. Tabla de cumplimiento

Requerimiento	Cumple/No cumple	Referencia
Dataset	Cumple	IV Preparación de Datos
Representación del problema	Cumple	I.E – Objetivos
Ambiente	Cumple	VI.B – Docker
Versionado de código	Cumple	https://github.com/joldan/MLOps.git
Desafíos generales	Cumple	IV.C – Limpieza
Exponer una API con el modelo	Cumple	VI – Fast-API
Plataforma de despliegue	Cumple	3.E – Producción
Requerimientos electivos (al menos 3)	Cumple	
Trazabilidad de ML	Cumple	V.D – Entrenamiento de Modelo
Explicabilidad	No cumple	-
Visualización	Cumple	VI – Gradio
Optimización de modelos	No cumple*	V.D – Grid Search

*El último punto no se cumple pero se hace fuertes intentos.

FIN DEL DOCUMENTO