

Universidad ORT Uruguay
Facultad de Ingeniería

Obligatorio - Machine Learning en **Producción**

Autores:

Dario Perla - 76706
Jose Rodriguez - 288624

Profesores:

Matías Sorozabal
Federico Zaiter

Julio 2023

Índice

Índice	2
Introducción	3
Descripción	3
Requerimientos mínimos	3
Crear Dataset	4
Representación del problema	5
AMBIENTE	6
Versionado del código	7
Desafíos Generales	8
Exponer Api	8
Plataforma de despliegue	8
Requerimientos electivos	10
Implementación de técnicas opcionales:	10
Ajuste Hyperparameters	10
Gradio	10
Experimentos	11
Entregables	13
Conclusiones	14
Bibliografia	14

Introducción

Descripción

El objetivo del obligatorio es llevar a la práctica los conceptos vistos durante el curso construyendo un sistema de Machine Learning de principio a fin.

La tarea consiste en desarrollar un sistema de clasificación que combine imágenes y datos tabulares para resolver un problema de clasificación binaria o multiclase. El proyecto incluye la construcción del dataset, la selección de características, la optimización del modelo y la exposición de una API para realizar predicciones online y batch. Se va a prestar especial atención a que se tomen las consideraciones necesarias para prevenir los problemas comunes al desarrollar sistemas de ML que van ser llevados a producción e interactuar con la realidad.

Requerimientos mínimos

- Crear Dataset
- Representación del problema
- Ambiente
- Versionado de código
- Desafíos generales
- Exponer Api
- Plataforma de despliegue

Crear Dataset

El Dataset será una composición de imágenes ya catalogadas de casas y apartamentos.

La creación del Dataset se realiza a partir del práctico realizado en clase, donde se planteaba la extracción de las imágenes del Gallito Luis de las secciones casas y apartamentos.

- <https://www.gallito.com.uy/inmuebles/apartamentos>
- <https://www.gallito.com.uy/inmuebles/casas>

Se utilizará Scrapy una biblioteca de código abierto la cual es un marco de aplicación para rastrear sitios web y extraer datos estructurados.

Se utiliza una rutina spider que permite realizar varios requerimientos independientes (en nuestro caso se bajan varias imágenes en simultáneo), las cuales deben ser controladas para evitar abusos sobre el sitio que nos provee las imágenes.

La recolección de imágenes se realiza hacia un storage de Azure, en donde quedarán disponibles para la siguiente fase del procesamiento.

El scraper lo que realiza es para cada imagen que encuentra la sube al Blob de Azure pero lo sube a una carpeta específica y al nombre le agrega la etiqueta y el id de la foto.

Ejemplo:

: “Apartamento/Apartamento_12713194_110acc04351e362095696cba4b8c0d5b25e8cf96.jpg”

Entendemos que quizás deberíamos tener mas imagenes en el dataset y para ello quizás hubieramos tenido que usar mas data augmentation pero entendimos que el modelo no era el tema sino su despliegue desde end-to-end

Representación del problema

El problema a resolver es el clasificar las imágenes entre apartamento o casas.

Las imágenes son JPG, de 256*256 en RGB.

Para los entrenamientos se utilizará los datos del gallito luis de las secciones casas y apartamentos.

Se expondrá una Url, el la cual se le podrá ingresar imágenes y se obtendrá el porcentaje con la probabilidad de cada clase de datos (si es casa o apartamento).

Para nuestro trabajo utilizaremos un Spider para bajar la información y crear el dataset.

En “colab”, se cargará la información y se normalizara, ejecutando luego los procedimientos de entrenamiento

Dentro de ese estudio se evaluarán varios modelos, y se utilizará las técnicas mencionadas en clase para mejorarlo.

El modelo ya creado y optimizado se despliega en un docker en una Virtual machine de Azure.

Además el modelo también es publicado en “Gradio” para ser consumido por el cliente final.

El objetivo final es que el modelo pueda hacer inferencia y poder decirnos si dada una foto se está ante una casa o un apartamento.

Ambiente

Para realizar el obligatorio utilizamos las siguientes dependencias para el scraper y que básicamente fueron las que se vieron en los laboratorios:

- scrapy
- Pillow
- Azure

Utilizando pip tools se manejan las otras dependencias generadas por las primeras tres y que se encuentran en requirements.in

En el caso de la api y docker tuvimos problemas a la hora de hacer el deploy ya que encontramos errores en las dependencias y seguramente fue porque hubo actualizaciones en estos días y la api había dejado de funcionar al hacer deploy con el último modelo . Para ello hicimos manualmente la selección de dependencias y el resultado fue :

- Pillow
- numpy==1.22
- fastapi
- uvicorn
- typing-extensions
- tensorflow
- keras==2.12.0
- python-multipart

Aprendimos de primera mano que el tema de dependencias puede ser complejo ya que literalmente dejó de funcionar de un día para el otro.

Versionado del código

Para versionar el código utilizamos git el cual se encuentra en:

- URL Git: <https://github.com/jrodriguezshaw/mlprodoobl>

Ahí fuimos guardando tanto el código como los notebooks que se usaron en Colab. Desde Colab te permite grabar los notebook directamente en git. Y utilizamos dos ramas siendo main la final.

Desafíos Generales

En el caso para prevenir data leakage tomamos la decisión de separar el dataset en tres sets de train, validation y test y aplicar las transformaciones que fueren necesarias después de hacer el split y no antes ya que eso si se hace antes hay una filtración de información en la cual el modelo está aprendiendo que el conjunto de pruebas tiene cierta información asociada con los datos de entrenamiento y, por lo tanto, puede rendir bien en el conjunto de pruebas debido a que en realidad fue entrenado correctamente en el conjunto de entrenamiento. En definitiva no se aplica ningún cambio al dataset entero

Exponer Api

Se expuso una api en la dirección: <http://4.246.188.146:8080/predict> y a la cual le pegamos desde la test que se encuentra en la carpeta serving dentro del repo.

Para exponer la api utilizamos:

- Azure
- Reglas en azure para poder pegarle desde afuera
- VM con ubuntu y docker
- Se usó un contenedor con docker que contiene la api expuesta en el puerto 8080
- El modelo seleccionado fue el modelo 3.

Plataforma de despliegue

La plataforma de despliegue fue en azure en donde como mencionamos anteriormente se expuso la api. El entrenamiento se realizó la mayoría en colab pero también utilizamos un notebook en azure para probar los dos entornos y donde notamos que en azure el procesamiento fue mucho más lento que en Colab y seguramente esto fue a la hora de elegir el cómputo en azure ya que como es paga elegimos menos recursos.

También notamos que no es fácil el salto entre distintos entornos ya que según el kernel de Azure las cosas dejaban de funcionar y no es tan fácil como copiar y pegar desde Colab a Azure.

Requerimientos electivos

Implementación de técnicas opcionales:

Ajuste Hyperparameters

Utilizamos Grid Search para lograr mejorar los hiperparametros.

Ver

Notebook: Entrenamiento_GRIDSEARCH_obl.ipynb

Gradio

Utilizamos Gradio para tener un entorno gráfico basándonos en el modelo 3 y el código se encuentra en el notebook entregado como parte del trabajo.

```
def classify_image(image):
    image = np.expand_dims(image, axis=0)
    prediction = model3.predict(image)

    # Obtiene las etiquetas de las clases
    class_labels = ['Casa', 'Apartamento']


    # Devuelve un diccionario con las etiquetas de las clases y sus probabilidades correspondientes
    return {class_labels[i]: float(prediction[0][i]) for i in range(2)}
```

```
import gradio as gr

iface = gr.Interface(
    fn=classify_image, # la función que hace la clasificación
    inputs=gr.inputs.Image(shape=(256, 256)), # el tipo de entrada que espera tu modelo
    outputs=gr.outputs.Label(num_top_classes=3), # el tipo de salida que produce tu modelo
)
iface.launch(share=True)
```

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
Running on public URL: <https://aca56549a08bec7c54.gradio.live>

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from Terminal to deploy to Spaces (<https://huggingface.co/spaces/gradio/deploys>)

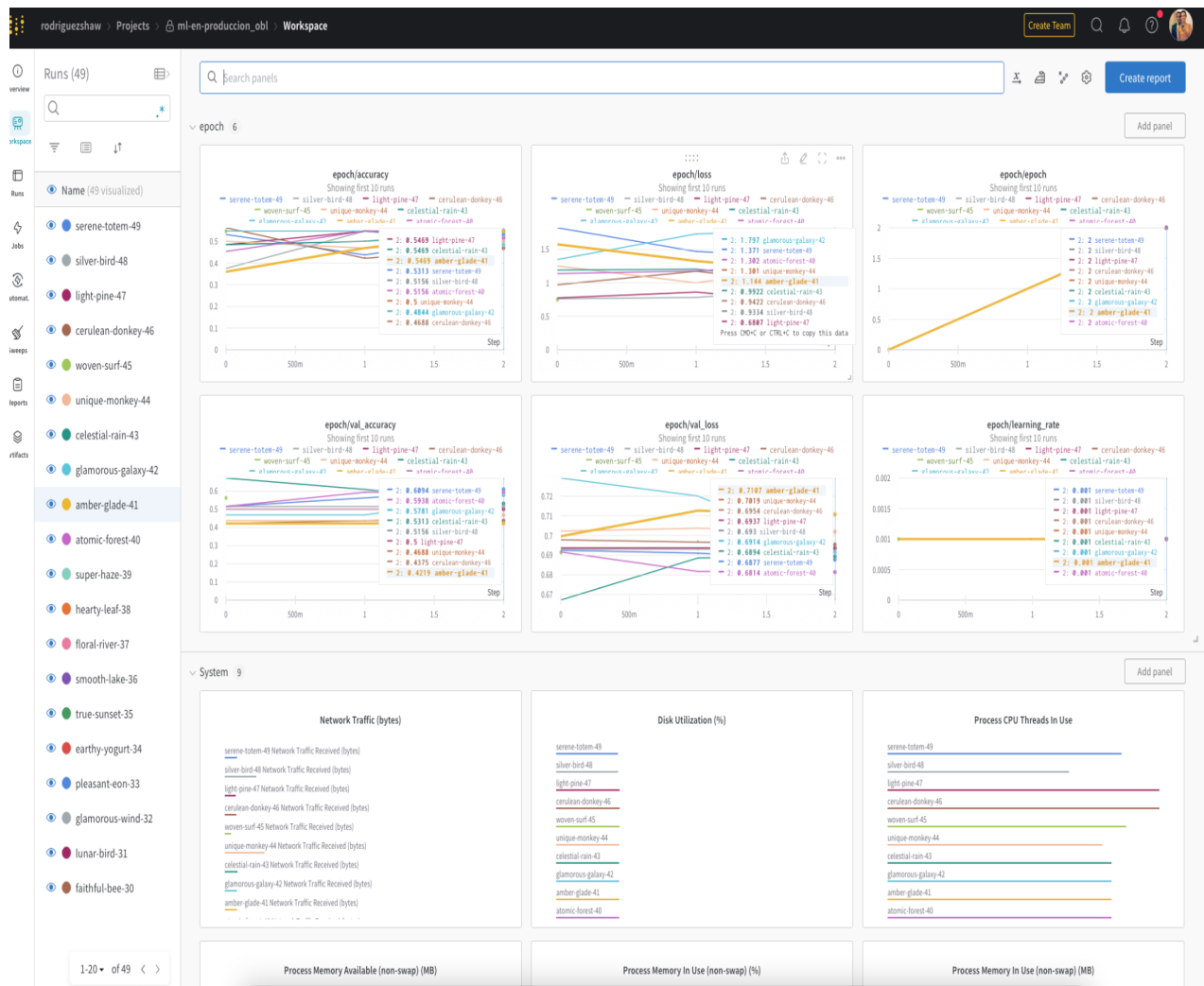


The Gradio interface displays a house image on the left and classification results on the right. The results show 'Casa' with a 51% probability and 'Apartamento' with a 49% probability. Below the results is an 'Avisar' button. At the bottom of the interface are 'Limpiar' and 'Enviar' buttons.

Class	Probability
Casa	51%
Apartamento	49%

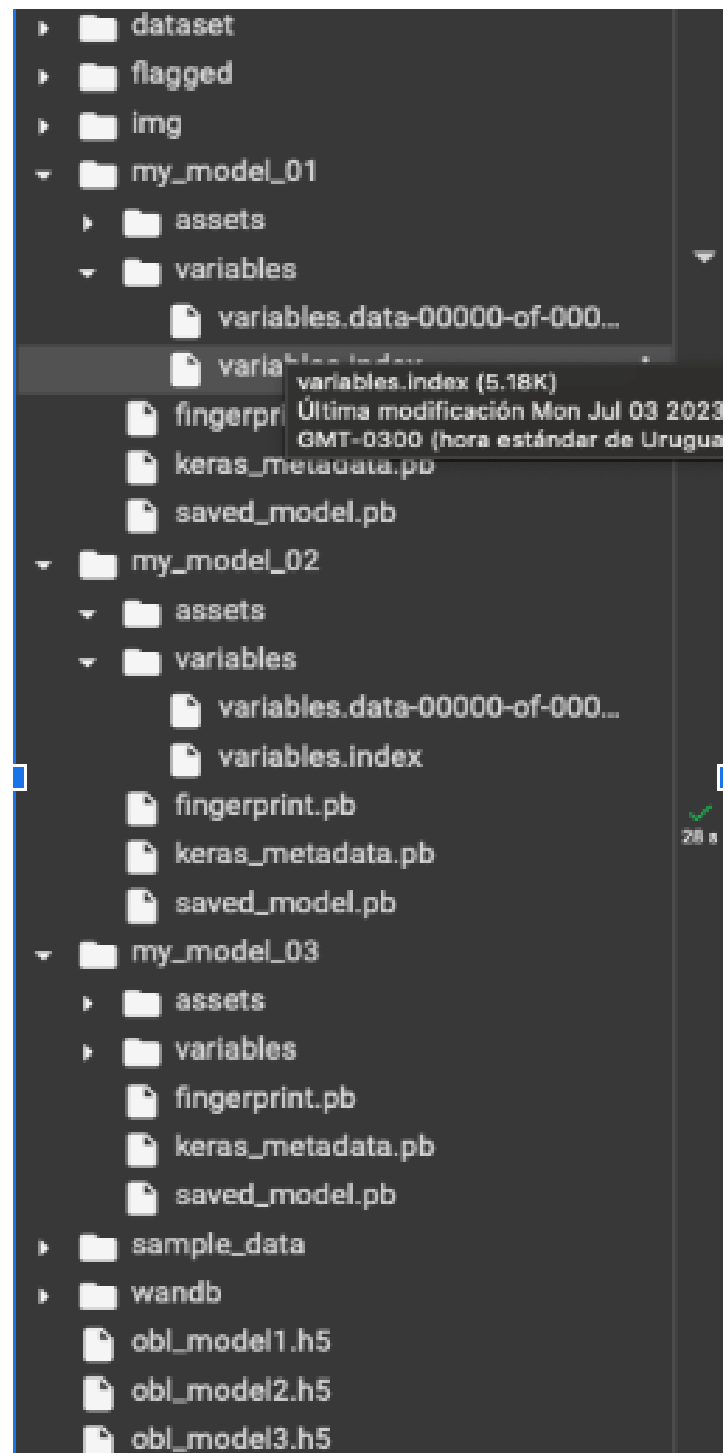
Experimentos

Usamos wandb para hacer seguimiento a las corridas y para guardar los artefactos de esas corridas. Las corridas se realizaron tanto desde Colab como desde Azure y todo funciona de maravilla la integración con la herramienta es muy simple y transparente. Lamentablemente no pudimos bajar los artefactos creados en las ejecuciones pero dejamos un link para entrar en azure por si se precisara ver. Si se entregan los artefactos de los modelos 1,2 y 3.



	rodriguezshaw > Projects > ml-en-production_obl > Table																				
Overview	Runs (49)																				
NonSpace	<div> <input type="text"/> Create Team </div> <div> Filter Sort Group Tag Move Create Sweep </div>																				
	Name (49 visualized)	Status	Notes	User	Tags	Created	Runtime	Sweep	activation_	batch_size	dropout	epoch	loss	metric	optimizer	epoch/accu	epoch/epoc	epoch/learr	epoch/loss	epoch/val_	epoch/val_
Runs	- serene-totem-4	Running	Add notes	rodrigu		44s ago	38s	-	relu	128	0.7079	3	categorica	accuracy	adam	0.5313	2	0.001	1.371	0.6094	0.6877
Jobs	- silver-bird-48	Finished	Add notes	rodrigu		1m ago	44s	-	relu	128	0.7961	3	categorica	accuracy	adam	0.5156	2	0.001	0.9334	0.5156	0.693
Automat.	- light-pine-47	Finished	Add notes	rodrigu		17m ago	55s	-	relu	128	0.4122	3	categorica	accuracy	adam	0.5469	2	0.001	0.6807	0.5	0.6937
	- cerulean-dontke...	Finished	Add notes	rodrigu		18m ago	57s	-	relu	128	0.4739	3	categorica	accuracy	adam	0.4688	2	0.001	0.9422	0.4375	0.6954
	- woven-surf-45	Finished	Add notes	rodrigu		19m ago	29s	-	relu	128	0.4149	3	categorica	accuracy	adam	0.5469	0	0.001	0.7497	0.5625	0.6914
Sweeps	- unique-monkey...	Finished	Add notes	rodrigu		31m ago	53s	-	relu	128	0.3483	3	categorica	accuracy	adam	0.5	2	0.001	1.301	0.4688	0.7019
Reports	- celestial-rain-43	Finished	Add notes	rodrigu		32m ago	50s	-	relu	128	0.4057	3	categorica	accuracy	adam	0.5469	2	0.001	0.9922	0.5313	0.6894
Artifacts	- glamorous-gala...	Finished	Add notes	rodrigu		33m ago	57s	-	relu	128	0.0735	3	categorica	accuracy	adam	0.4844	2	0.001	1.797	0.5781	0.6914
	- amber-glade-41	Finished	Add notes	rodrigu		34m ago	55s	-	relu	128	0.5565	3	categorica	accuracy	adam	0.5469	2	0.001	1.144	0.4219	0.7107
	- atomic-forest-40	Finished	Add notes	rodrigu		35m ago	57s	-	relu	128	0.03834	3	categorica	accuracy	adam	0.5156	2	0.001	1.302	0.5938	0.6814
	- super-haze-39	Finished	Add notes	rodrigu		36m ago	49s	-	relu	128	0.7661	3	categorica	accuracy	adam	0.5781	2	0.001	1.573	0.4063	0.7064
	- hearty-leaf-38	Finished	Add notes	rodrigu		37m ago	54s	-	relu	128	0.6439	3	categorica	accuracy	adam	0.5469	2	0.001	1.251	0.5	0.6938
	- floral-river-37	Finished	Add notes	rodrigu		38m ago	49s	-	relu	128	0.6885	3	categorica	accuracy	adam	0.5469	2	0.001	1.068	0.3594	0.6993
	- smooth-lake-36	Finished	Add notes	rodrigu		39m ago	53s	-	relu	128	0.6508	3	categorica	accuracy	adam	0.625	2	0.001	0.9095	0.5	0.6955
	- true-sunset-35	Finished	Add notes	rodrigu		40m ago	49s	-	relu	128	0.0391	3	categorica	accuracy	adam	0.6406	2	0.001	1.279	0.4688	0.7165
	- earthy-yogurt-34	Finished	Add notes	rodrigu		41m ago	48s	-	relu	128	0.6898	3	categorica	accuracy	adam	0.4844	2	0.001	1.49	0.5	0.6934
	- pleasant-eon-33	Finished	Add notes	rodrigu		42m ago	58s	-	relu	128	0.1271	3	categorica	accuracy	adam	0.4375	2	0.001	1.3	0.5	0.6982
	- glamorous-wind...	Finished	Add notes	rodrigu		43m ago	53s	-	relu	128	0.6525	3	categorica	accuracy	adam	0.5156	2	0.001	1.249	0.4844	0.7057
	- lunar-bird-31	Finished	Add notes	rodrigu		44m ago	55s	-	relu	128	0.451	3	categorica	accuracy	adam	0.4688	2	0.001	1.394	0.5156	0.6931
	- faithful-bee-30	Finished	Add notes	rodrigu		45m ago	56s	-	relu	128	0.03016	3									

Pruebas realizadas



Entregables

Se entrega como parte del trabajo:

- Notebooks
 - Entrenamiento_obl.ipynb : contiene el entrenamiento completo de los modelos
 - Entrenamiento_GRIDSEARCH_obl.ipynb : Se realiza sobre el modelo3 para obtener los mejores hyperparametros.
 - Copia_de_entrenamiento_obl.ipynb ; Este notebook se dejo para mostrar el uso de versionados de github, ya que al resto se le cambio el nombre para las entrega final.
- Documento de la tarea
 - ObligatorioMLP.docx : Documentacion actual
 - ObligatorioMLP.pdf : Version pdf
- Enlaces
 - Api : <http://4.246.188.146:8080/predict>
 - Gradio: <https://aca56549a08bec7c54.gradio.live/>
 - Git: <https://github.com/jrodriguezshaw/mlprodobl>
 - Azure:
<https://jr288624.eastus.instances.azureml.ms/lab/tree/Users/JR288624/obl1>

Obs: En el caso de Gradio iremos actualizando el link para que permanezca activo para su corrección.

Conclusiones

1. Creemos haber cumplido el propósito del obligatorio en cuanto a tener un proyecto completo, desde la recopilación de datos, hasta la puesta en producción.
2. Observaciones:
 - a. Observamos que la predicciones no son buenas, lo cual es esperable ya que el dataset de entrada si bien tiene casa y apartamentos como clasificador original, se observan fotos de muchos elementos en común, habitaciones internas, jardines, piscinas, las cuales son válidas tanto para casas como apartamentos.
 - b. Se realizó el análisis de duplicados, pero se basó en el Id de la imagen, y quizás cuando en realidad debimos analizar la imagen en sí, ya que la misma foto puede ser publicada por varias inmobiliarias por ejemplo, donde cada una tiene un id distinto.
 - c. El grid search fue limitado, ya que Colab nos limitaba la cantidad de tiempo asignado al análisis.
 - d. Azure,
 - e. Tuvimos problemas con la suscripción
 - f. Tuvimos problemas aleatorios que no logramos identificar. En donde el mismo código a veces funcionaba y a veces daba error.

Bibliografía

- Tareas prácticos del curso
- <https://docs.scrapy.org/>
 - <https://docs.scrapy.org/en/latest/topics/media-pipeline>.
 - <https://docs.scrapy.org/en/latest/topics/item-pipeline.html>
 - <https://docs.scrapy.org/en/latest/topics/settings.html>
- <https://learn.microsoft.com/en-us/azure>
- Pillow
- <https://stackoverflow.com/questions/74017936/download-images-from-azure-storage>
- [https://azuresdkdocs.blob.core.windows.net/\\$web/python/azure-storage-blob/12.0.0b5/index.html](https://azuresdkdocs.blob.core.windows.net/$web/python/azure-storage-blob/12.0.0b5/index.html)
- <https://learn.microsoft.com/en-us/python/api/azure-storage-blob/azure.storage.blob.containerclient?view=azure-python>
- <https://shahzaibchadhar.medium.com/how-to-use-scrapy-for-image-download-using-pipelines-in-python-37be4cce6c18>
- <https://towardsdatascience.com/what-is-data-leakage-and-how-can-it-be-avoided-in-machine-learning-eb435a27c3e3>
- <https://machinelearningmastery.com/data-preparation-without-data-leakage/>
- <https://machinelearningmastery.com/how-to-normalize-center-and-standardize-images-with-the-imagedatagenerator-in-keras/>
- <https://gradio.app/>