

PRODUCTIVIZACIÓN DEL MODELO

Máster Ciencia de Datos CUNEF

José Carlos Monescillo Calzado

Guía de uso: <https://youtu.be/HaqMEagPfuU>

Introducción

El presente informe detalla el proceso llevado a cabo para la puesta en producción de un modelo de predicción mediante una infraestructura bien organizada y escalable. El trabajo se divide en tres etapas principales:

- **Dockerización del entorno:** en esta fase, se configuró un contenedor Docker que garantiza la reproducibilidad del entorno de desarrollo, incluyendo las dependencias necesarias y configuraciones específicas para ejecutar el modelo y las aplicaciones asociadas.
- **Creación de una API con Flask:** posteriormente, se diseñó una API utilizando Flask para invocar el modelo, lo que permite recibir datos desde un cliente, procesarlos, y devolver predicciones de manera eficiente y automatizada.
- **Desarrollo de un dashboard con Streamlit:** finalmente, se desarrolló un dashboard utilizando Streamlit. Este dashboard permite monitorizar el número de llamadas a la API, clasificar las tipologías de llamadas realizadas, y analizar la distribución de los resultados obtenidos.

1. Dockerización del entorno

En este paso se describe el proceso de dockerización del proyecto de predicción de préstamos. El objetivo principal fue encapsular todas las dependencias y configuraciones del entorno en un contenedor Docker para garantizar que el proyecto se pueda ejecutar de manera consistente en cualquier máquina, independientemente del sistema operativo o configuraciones específicas.

Los principales objetivos de dockerizar el proyecto son varios:

1. Facilitar la portabilidad del proyecto.
2. Asegurar que todas las dependencias necesarias estén incluidas y correctamente configuradas.

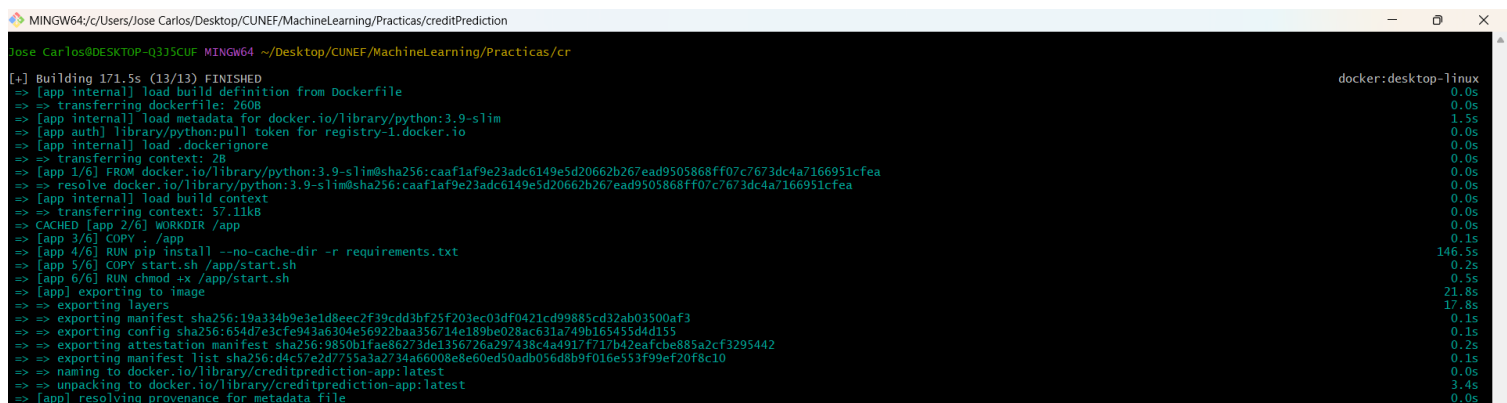
3. Reducir problemas relacionados con inconsistencias en el entorno.
4. Permitir la integración del proyecto con otros servicios o sistemas.

Archivos clave

- **Dockerfile:** este archivo define cómo construir la imagen Docker del proyecto. Incluye instrucciones para:
 - Instalar las dependencias listadas en requirements.txt.
 - Configurar el entorno necesario para ejecutar Flask y Streamlit.
 - Especificar el directorio de trabajo y los comandos de inicio.
- **docker-compose.yml:** este archivo orquesta los servicios necesarios para el proyecto, permitiendo ejecutar múltiples contenedores de manera coordinada. Incluye:
 - Servicio app que:
 - Expone los puertos para la API Flask (5000) y el dashboard de Streamlit (8501).
 - Monta un volumen persistente para almacenar logs.
 - Configura variables de entorno, como `FLASK_ENV=production` para optimizar el rendimiento en producción.

Flujo de Trabajo

1. Construcción de la imagen: el comando `docker compose build` descarga las dependencias y configura el entorno dentro del contenedor.



```
MINGW64~/Users/Jose Carlos/Desktop/CUNEF/MachineLearning/Practicas/creditPrediction
jose Carlos@DESKTOP-Q3J5CUF MINGW64 ~/Desktop/CUNEF/MachineLearning/Practicas/cr
[+] Building 171.5s (13/13) FINISHED
=> [app internal] load build definition from Dockerfile
=> => transferring dockerfile: 260B
=> [app internal] load metadata for docker.io/library/python:3.9-slim
=> [app auth] library/python:pull token for registry-1.docker.io
=> [app internal] load .dockerignore
=> => transferring context: 2B
=> [app 1/6] FROM docker.io/library/python:3.9-slim@sha256:caaf1af9e23adc6149e5d20662b267ead9505868ff07c7673dc4a7166951cfea
=> resolve docker.io/library/python:3.9-slim@sha256:caaf1af9e23adc6149e5d20662b267ead9505868ff07c7673dc4a7166951cfea
=> [app internal] load build context
=> => transferring context: 57.11kB
=> CACHED [app 2/6] WORKDIR /app
=> [app 3/6] COPY . /app
=> [app 4/6] RUN pip install --no-cache-dir -r requirements.txt
=> [app 5/6] COPY start.sh /app/start.sh
=> [app 6/6] RUN chmod +x /app/start.sh
=> [app] exporting to image
=> => exporting layers
=> => exporting manifest sha256:19a334b9e3e1d8e2cf39cdd3bf25f203ec03df0421cd99885cd32ab03500af3
=> => exporting config sha256:654d7e3cfe943a6304e56922baa356714e189be028ac631a749b165455d4d155
=> => exporting attestation manifest sha256:9850b1fae86273de1356726a297438c4a4917f717b42eafcb885a2cf3295442
=> => exporting manifest list sha256:d4c57e2d7755a3a2734a66008e8e60ed50adb056d8b9f016e553f99ef20f8c10
=> => naming to docker.io/library/creditprediction-app:latest
=> => unpacking to docker.io/library/creditprediction-app:latest
=> [app] resolving provenance for metadata file
```

2. Inicio de los servicios: usando `docker compose up`, se inician tanto la API Flask como el dashboard Streamlit, permitiendo su acceso desde el navegador.

```
Jose Carlos@DESKTOP-Q3J5CUF MINGW64 ~/Desktop/CUNEF/MachineLearning/Practicas/creditPrediction (main)
$ docker compose up
[+] Running 1/1
  ✓ Container flask_streamlit_app Recreated
Attaching to flask_streamlit_app
flask_streamlit_app | [2025-01-04 17:24:30 +0000] [7] [INFO] Starting gunicorn 20.1.0
flask_streamlit_app | [2025-01-04 17:24:30 +0000] [7] [INFO] Listening at: http://0.0.0.0:5000 (7)
flask_streamlit_app | [2025-01-04 17:24:30 +0000] [7] [INFO] Using worker: sync
flask_streamlit_app | [2025-01-04 17:24:30 +0000] [9] [INFO] Booting worker with pid: 9
flask_streamlit_app | [2025-01-04 17:24:30 +0000] [10] [INFO] Booting worker with pid: 10
flask_streamlit_app | Collecting usage statistics. To deactivate, set browser.gatherUsageStats to false.
flask_streamlit_app |
flask_streamlit_app | You can now view your Streamlit app in your browser.
flask_streamlit_app |
flask_streamlit_app | URL: http://0.0.0.0:8501
```

3. Detención y limpieza: el comando `docker compose down` detiene los servicios y elimina los contenedores.

```
Jose Carlos@DESKTOP-Q3J5CUF MINGW64 ~/Desktop/CUNEF/MachineLearning/Practicas/creditPrediction (main)
$ docker compose down
[+] Running 2/2
  ✓ Container flask_streamlit_app Removed
  ✓ Network creditprediction_default Removed

Jose Carlos@DESKTOP-Q3J5CUF MINGW64 ~/Desktop/CUNEF/MachineLearning/Practicas/creditPrediction (main)
$ |
```

Este enfoque permite separar la lógica de la aplicación de la configuración del entorno, garantizando reproducibilidad, escalabilidad y facilidad de despliegue.

La dockerización del proyecto garantiza que el entorno sea reproducible en cualquier máquina con Docker instalado.

2. Creación de API con Flask y Gunicorn

La API desarrollada con Flask constituye el backend del proyecto, gestionando las solicitudes de predicción y comunicándose con el modelo de machine learning. Gunicorn, un servidor de aplicaciones WSGI, permite manejar múltiples solicitudes concurrentes, mejorando el rendimiento en producción.

Funcionalidad Principal

- **Endpoints:**
 1. Renderiza un formulario HTML dinámico en la ruta principal (`/`), adaptado a las columnas del modelo.

2. Recibe datos a través de la ruta /predict, procesa los datos con el modelo y devuelve la predicción como JSON.
3. Proporciona una lista de columnas necesarias para la predicción mediante la ruta /columns.

Implementación

- **Modelo y columnas:** el modelo entrenado y las columnas requeridas se cargan desde un archivo predefinido en formato pickle.
- **Registro de actividades:** cada solicitud se registra en un archivo de logs, incluyendo datos de entrada, resultados de predicción y posibles errores. Esto facilita el monitoreo y la depuración.

3. Elaboración del Dashboard con Streamlit

El dashboard actúa como la interfaz gráfica de la aplicación, diseñada para usuarios finales que desean interactuar con el modelo de predicción de manera sencilla e intuitiva.

Funcionalidades

1. Formulario de Predicción:

- Se conecta al backend para enviar datos de entrada.
- Genera dinámicamente campos en el formulario basándose en las columnas requeridas por el modelo.
- Muestra los resultados de la predicción directamente en la interfaz del usuario.

2. Dashboard de Métricas:

- Proporciona estadísticas detalladas, como el número total de solicitudes realizadas, predicciones exitosas y errores registrados.
- Muestra un registro tabular de todas las solicitudes y sus resultados, con detalles como la marca de tiempo, los datos de entrada y la predicción generada.
- Genera gráficos interactivos que permiten visualizar la distribución de las predicciones y otros datos relevantes.

Conclusión

Este proyecto utiliza herramientas actuales para crear una solución práctica y fácil de escalar en el mundo del Machine Learning:

1. **Dockerización:** garantiza la portabilidad y la facilidad de despliegue, permitiendo replicar el entorno en diferentes máquinas de manera uniforme.
2. **API Flask:** proporciona un backend robusto, con gestión eficiente de solicitudes y un diseño extensible para futuras mejoras.
3. **Dashboard Streamlit:** ofrece una interfaz amigable y visualmente atractiva, facilitando la interacción con el modelo de predicción y el análisis de métricas en tiempo real.