Proyecto Final: Despliegue de Sistema de Reconocimiento de Actividad Humana (MHealth)

Victor Cornejo
Taller Integrador de Ciencia de Datos

3 de diciembre de 2025

Resumen

Este informe detalla el diseño, desarrollo y despliegue de una aplicación web para el reconocimiento de actividades humanas utilizando el dataset MHealth. El sistema integra un modelo de Machine Learning (Random Forest) dentro de una arquitectura contenerizada con Docker, exponiendo una API REST en Python (Flask) consumida por un Frontend moderno en Vue.js. El proyecto demuestra la viabilidad de transicionar un modelo experimental a un entorno productivo robusto.

Índice

1. Resumen Ejecutivo

El presente proyecto finaliza el ciclo del Taller Integrador de Ciencia de Datos mediante la implementación de un sistema de software completo (End-to-End). El sistema permite a usuarios finales cargar archivos de sensores crudos y obtener clasificaciones automáticas de actividad física. Se ha priorizado la reproducibilidad técnica mediante el uso de contenedores Docker y la usabilidad mediante una interfaz gráfica amigable, cumpliendo con los estándares de ingeniería de software aplicados a la ciencia de datos.

2. Contexto

El conjunto de datos MHealth (Mobile Health) recopila señales de sensores inerciales ubicados en el pecho, muñeca y tobillo de diversos sujetos. En etapas previas, se exploraron estos datos y se entrenaron modelos predictivos. El desafío actual reside en el despliegue, llevar esos modelos estáticos a una aplicación viva que pueda procesar nuevos datos en tiempo real, gestionando la ingesta, limpieza y predicción de forma transparente para el usuario.

3. Arquitectura del Sistema

El sistema adopta un patrón de arquitectura de microservicios, donde el Frontend y el Backend operan en contenedores independientes.

3.1. Diagrama de Arquitectura

La Figura 1 presenta el flujo de datos e interacción entre componentes.

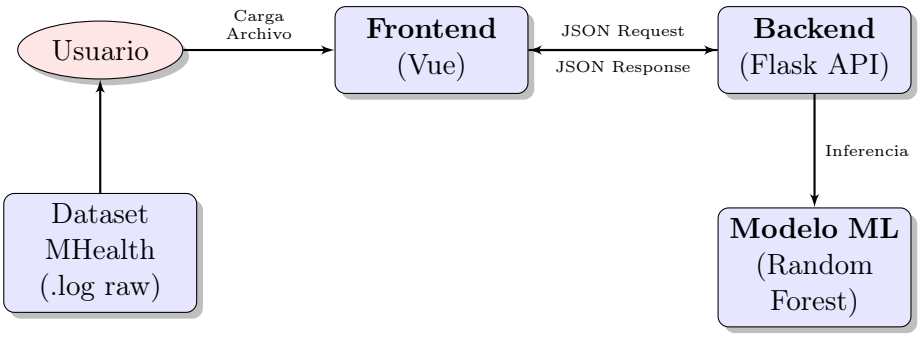

Figura 1: Diagrama de Arquitectura de la Solución.

3.2. Componentes Principales

1. **Cliente Web:** Interfaz de usuario responsable de la captura del archivo y visualización de resultados.
2. **API REST:** Servicio backend que orquesta el pipeline de datos (limpieza, ventaneo) y ejecuta el modelo.
3. **Infraestructura:** Docker Compose gestiona la red interna y los volúmenes necesarios para el funcionamiento.

4. Descripción de Tecnologías Escogidas

4.1. Back End (Python + Flask)

Se seleccionó **Python** por su ecosistema dominante en Ciencia de Datos. **Flask** se utiliza como framework web debido a su ligereza ("micro-framework"), lo que facilita envolver el modelo de `scikit-learn` sin la complejidad innecesaria de frameworks más grandes como Django.

4.2. Front End (Vue.js)

Vue.js fue elegido por su reactividad y facilidad de integración. Permite construir una "Single Page Application"(SPA) que ofrece una experiencia fluida (sin recargas de página) al subir archivos y esperar predicciones asíncronas. Se utilizó **Vite** para un empaquetado optimizado.

5. Descripción de la API

La comunicación entre capas se realiza mediante protocolo HTTP estándar.

5.1. Endpoints

- `GET /health`: Verificación de estado del servicio.
- `POST /detect`: Endpoint principal. Recibe un archivo `multipart/form-data` y retorna un JSON con la clasificación.

Ejemplo de Respuesta JSON:

```
{
  "top_activity": "Trotar (Jogging)",
  "confidence": 92.5,
  "ranking": [
    {"activity": "Trotar", "percentage": 92.5},
    {"activity": "Caminar", "percentage": 7.5}
  ]
}
```

6. Despliegue e Instrucciones de Uso

6.1. Requisitos Previos

- Docker Engine y Docker Compose instalados en el sistema anfitrión.
- Acceso a internet (para descargar las imágenes base la primera vez).

6.2. Pasos de Ejecución

1. Entrenamiento Inicial: El modelo esta presente en el repositorio, pero estos fueron los pasos para generarlo.

```
docker-compose up -d backend
docker-compose exec backend python model_train.py
```

2. Despliegue de la Aplicación: Una vez entrenado el modelo, se levanta el stack completo:

```
docker-compose down
docker-compose up --build
```

El sistema estará disponible en http://localhost:8080.

7. Resultados de Pruebas Básicas

Se definieron casos de prueba para validar la robustez del sistema.

7.1. Caso de Prueba 1: Flujo Exitoso (Happy Path)

Se cargó el archivo `mhealth_subject1.log` conteniendo datos de la actividad.

- **Resultado Esperado:** El sistema identifica la actividad y muestra el gráfico de confianza.
- **Resultado Obtenido:** Predicción. La interfaz mostró la tabla de ranking correctamente.

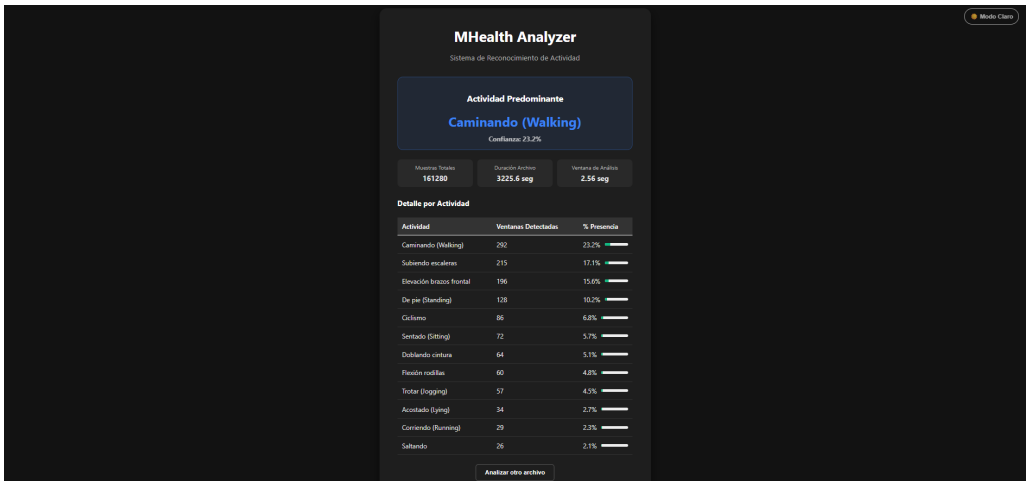

Figura 2: Validación del flujo exitoso.

7.2. Caso de Prueba 2: Archivo Faltante o Inválido

Se intentó enviar la petición sin adjuntar un archivo o enviando un archivo de texto vacío.

- **Comportamiento:** El Backend detecta la ausencia del archivo en el request (`request.files`) y retorna un código HTTP 400.
- **Visualización:** El Frontend captura el error y muestra una alerta roja: "No selected file", sin romper la aplicación.

7.3. Desempeño del Modelo

El tiempo de inferencia promedio para un archivo de 1 minuto de duración (aprox. 3000 muestras) es de $<$ **2 segundos** en un entorno Docker local, lo cual es aceptable para una interacción web asíncrona.

8. Reflexión y Mejoras Futuras

8.1. Rol del Despliegue en Data Science

Este proyecto evidencia que un modelo con alta precisión métrica (Accuracy) no es útil por sí solo. El despliegue requiere habilidades de ingeniería de software (Docker, APIs, Frontend) que son críticas para transformar un experimento en un producto de valor. La reproducibilidad lograda mediante contenedores elimina el problema de "funciona en mi máquina".

8.2. Posibles Mejoras

- **Base de Datos:** Implementar PostgreSQL para guardar un historial de las predicciones realizadas.
- **Segmentación Automática:** Mejorar el algoritmo para detectar múltiples actividades distintas dentro de un mismo archivo continuo, en lugar de asumir una actividad predominante única.

A. Anexos

- **GitHub:** Repositorio