Sistema de Reconocimiento de Actividad Humana

Basado en el Dataset MHealth

Proyecto Final - Despliegue de Modelos de Machine Learning

Universidad
Curso: Taller Final

Diciembre 2025

Resumen

Este documento presenta el desarrollo e implementación de un sistema completo de Reconocimiento de Actividad Humana (HAR) utilizando el dataset MHealth. El sistema integra un pipeline de Machine Learning reproducible, una API REST desarrollada con FastAPI y una interfaz web moderna construida con React y TypeScript. El modelo Random Forest implementado alcanza un accuracy del 92.6% y un Macro F1-Score del 92.6% en validación, permitiendo clasificar 12 actividades físicas diferentes a partir de señales de sensores corporales. La arquitectura del sistema sigue principios de diseño modular, facilitando el mantenimiento, la escalabilidad y el despliegue mediante contenedores Docker.

Palabras clave: Reconocimiento de Actividad Humana, Machine Learning, FastAPI, React, MHealth, Series Temporales, Random Forest.

Índice

1 Introducción

1.1 Contexto del Problema

El reconocimiento de actividad humana (HAR, por sus siglas en inglés *Human Activity Recognition*) constituye un área de investigación fundamental en el campo de la computación ubicua y el aprendizaje automático. Esta disciplina tiene aplicaciones directas en salud digital, monitoreo de pacientes, deportes, rehabilitación física y sistemas de asistencia inteligente.

El presente proyecto surge como fase final de un curso donde se trabajó iterativamente con el dataset MHealth, abordando desde la obtención y exploración de datos hasta el modelado y evaluación de algoritmos. Esta etapa final corresponde al **despliegue**, integrando el modelo entrenado en un sistema funcional compuesto por backend, frontend y documentación completa.

1.2 Objetivo General

Desarrollar y desplegar una aplicación web completa para reconocimiento de actividad humana que:

- Integre un modelo de clasificación entrenado sobre el dataset MHealth
- Exponga una API REST para realizar predicciones
- Ofrezca una interfaz de usuario accesible para usuarios no técnicos
- Incluya instrucciones de despliegue y uso reproducibles
- Documente el uso de herramientas de IA generativa

1.3 Objetivos Específicos

1. Implementar un pipeline de preprocesamiento que transforme señales crudas en características estadísticas por ventana temporal
2. Entrenar y evaluar un modelo de clasificación que distinga entre 12 actividades físicas
3. Desarrollar un backend RESTful que permita predicción y evaluación de archivos `.log`
4. Construir un frontend moderno que visualice predicciones y métricas de manera intuitiva
5. Containerizar la aplicación para facilitar su despliegue y reproducibilidad

2 Dataset y Preprocesamiento

2.1 Descripción del Dataset MHealth

El dataset MHealth (*Mobile Health*) fue recolectado por la Universidad Politécnica de Cataluña y está disponible públicamente en el repositorio UCI Machine Learning Repository[1]. Contiene registros de 10 sujetos realizando 12 actividades físicas diferentes mientras portaban sensores corporales.

2.1.1 Características de los Sensores

Se utilizan tres dispositivos de medición inercial ubicados en:

- **Pecho:** Acelerómetro (3 ejes) + 2 canales ECG
- **Tobillo izquierdo:** Acelerómetro, giroscopio y magnetómetro (3 ejes c/u)
- **Brazo derecho:** Acelerómetro, giroscopio y magnetómetro (3 ejes c/u)

En total, cada muestra contiene **23 columnas de sensores** más la etiqueta de actividad. La frecuencia de muestreo es de **50 Hz**.

[1] `https://archive.ics.uci.edu/dataset/319/mhealth+dataset`

2.1.2 Actividades Clasificadas

El sistema reconoce 12 actividades: De pie (1), Sentado (2), Acostado (3), Caminando (4), Subiendo escaleras (5), Flexión de cintura (6), Elevación frontal de brazos (7), Flexión de rodillas (8), Ciclismo (9), Trote (10), Corriendo (11) y Saltando (12).

2.2 Pipeline de Preprocesamiento

Las señales se segmentan en ventanas deslizantes de **5 segundos** con **50% de solapamiento**. Para cada ventana y sensor, se extraen 7 características estadísticas: media, desviación estándar, mínimo, máximo, mediana, MAD y energía, generando $23 \times 7 = \mathbf{161}$ **características** por ventana. Se aplica normalización Z-score ajustada exclusivamente sobre datos de entrenamiento.

2.3 Manejo del Desbalance de Clases

El dataset original incluye una **actividad 0** ("Sin clasificar") que constituye el 93% de las muestras. Se filtran todas las muestras con actividad 0 durante el preprocesamiento, ya que no representa una actividad física significativa y su predominancia impedía el aprendizaje de patrones útiles.

3 Arquitectura del Sistema

3.1 Visión General

El sistema sigue una arquitectura de tres capas claramente separadas: capa de Machine Learning, capa de Backend (API) y capa de Frontend (UI). La Figura 1 ilustra la interacción entre componentes.

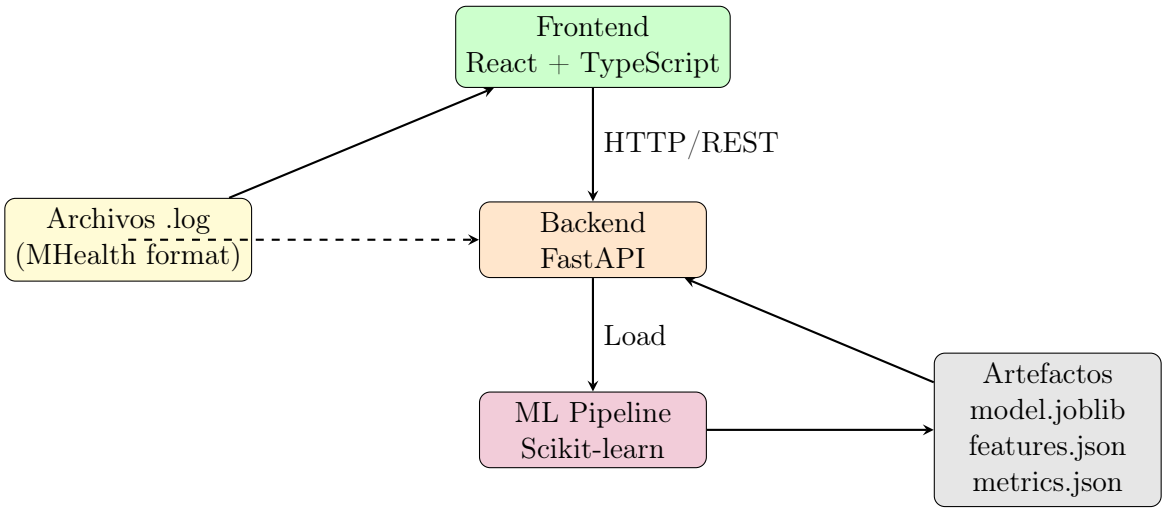

Figura 1: Arquitectura general del sistema MHealth HAR

3.2 Estructura del Repositorio

El proyecto está organizado como un monorepo con tres componentes principales:

- `ml/`: Pipeline de datos y scripts de entrenamiento/evaluación
- `backend/`: Servicio FastAPI con endpoints y lógica de negocio
- `frontend/`: Aplicación React con TypeScript
- `config/`: Configuración centralizada (YAML)
- `prompts/`: Documentación de uso de IA generativa

4 Modelo de Machine Learning

4.1 Selección del Algoritmo

Se seleccionó **Random Forest Classifier** por las siguientes razones:

1. **Robustez:** Maneja bien características correlacionadas y no requiere supuestos distribucionales
2. **Interpretabilidad:** Permite analizar importancia de features
3. **Eficiencia:** Entrenamiento rápido y paralelizable
4. **Generalización:** El ensemble de árboles reduce overfitting
5. **Balance de clases:** Soporta nativamente pesos por clase

4.2 Hiperparámetros

Cuadro 1: Configuración del modelo Random Forest

Parámetro	**Valor**
n_estimators	200
max_depth	None (sin límite)
class_weight	balanced
random_state	42
n_jobs	-1 (todos los cores)

4.3 División de Datos

Para garantizar la ausencia de fuga de datos, la división se realiza **por sujeto**, no por muestra. Esto asegura que las ventanas de un mismo sujeto no aparezcan simultáneamente en conjuntos de entrenamiento y evaluación.

Cuadro 2: División de sujetos para entrenamiento y evaluación

Conjunto	**Sujetos**	**Proporción**
Entrenamiento	4, 5, 6, 10	60%
Validación	3, 9	20%
Test	2, 7	20%
Demo (excluidos)	1, 8	—

Los sujetos 1 y 8 fueron completamente excluidos del entrenamiento, validación y normalización. Estos sujetos sirven como conjunto de demostración para validar el sistema end-to-end en la interfaz web.

4.4 Resultados y Métricas

Cuadro 3: Métricas de rendimiento del modelo

Conjunto	**Accuracy**	**Macro F1-Score**
Entrenamiento	100.0%	100.0%
Validación	92.6%	92.6%
Test	81.3%	81.1%
Demo (sujetos 1, 8)	99.3%	99.1%

El alto rendimiento en el conjunto Demo indica que el modelo generaliza correctamente a sujetos no vistos durante el entrenamiento.

4.5 Impacto de la Eliminación de Actividad 0

La Tabla 4 compara el rendimiento antes y después de filtrar la actividad 0:

Cuadro 4: Comparación de métricas antes y después de filtrar actividad 0

Métrica	**Antes**	**Después**	**Δ Absoluto**	**Δ Relativo**
Accuracy (Val)	71.8%	92.6%	+20.8%	+29.0%
Macro F1 (Val)	13.7%	92.6%	+78.9%	+575.9%
Accuracy (Test)	73.9%	81.3%	+7.4%	+10.0%
Macro F1 (Test)	12.0%	81.1%	+69.1%	+575.8%

5 Backend - API REST

5.1 Tecnologías Utilizadas

- **FastAPI:** Framework web moderno, de alto rendimiento, con validación automática y documentación OpenAPI
- **Pydantic v2:** Validación de datos y serialización
- **Uvicorn:** Servidor ASGI para producción
- **Python 3.11+:** Lenguaje base

5.2 Endpoints de la API

Cuadro 5: Descripción de endpoints de la API

Método	**Endpoint**	**Descripción**
GET	/health	Verifica que el servicio está activo. Retorna `{''status'': ''ok''}`
GET	/model-info	Retorna información del modelo: versión, hiperparámetros, métricas de entrenamiento, sujetos demo
POST	/predict	Recibe archivo .log y retorna predicción por ventana con probabilidades
POST	/evaluate-log	Recibe archivo .log etiquetado y retorna métricas (accuracy, F1, matriz de confusión)

5.3 Formato de Entrada y Salida

- **POST /predict:** Recibe archivo `.log` (23 columnas). Retorna JSON con predicción por ventana (índice, actividad, probabilidades) y resumen agregado (distribución por actividad).
- **POST /evaluate-log:** Recibe archivo `.log` etiquetado (24 columnas). Retorna métricas (accuracy, macro_f1), matriz de confusión, predicciones y ground truth.

El detalle completo de los esquemas JSON se encuentra en el documento de anexos.

5.4 Manejo de Errores

La API implementa validación de entrada y retorna códigos HTTP apropiados:

- **400 Bad Request:** Archivo no proporcionado, formato incorrecto, o archivo sin etiquetas para evaluación
- **500 Internal Server Error:** Modelo no disponible (falta entrenar)

6 Frontend - Interfaz de Usuario

6.1 Tecnologías Utilizadas

- **React 18:** Biblioteca para construcción de interfaces
- **TypeScript:** Tipado estático para mayor robustez
- **Vite:** Bundler moderno con HMR
- **CSS Custom Properties:** Sistema de diseño personalizado

6.2 Funcionalidades Principales

6.2.1 Predicción de Actividades

Permite al usuario cargar un archivo `.log` y obtener:

- Distribución porcentual de actividades detectadas
- Línea temporal con la secuencia de predicciones
- Tabla detallada con predicción y confianza por ventana

6.2.2 Evaluación del Modelo

Permite cargar un archivo `.log` con etiquetas para:

- Calcular accuracy y Macro F1-Score
- Visualizar matriz de confusión interactiva
- Comparar secuencia real vs. predicha en línea temporal

6.3 Diseño de Interfaz

El diseño sigue principios de UI/UX modernos:

- **Minimalismo:** Interfaz limpia sin elementos distractores
- **Feedback visual:** Estados de carga, éxito y error claramente diferenciados
- **Accesibilidad:** Etiquetas descriptivas y colores con suficiente contraste
- **Responsividad:** Adaptación a diferentes tamaños de pantalla
- **Drag & Drop:** Zona de arrastre intuitiva para archivos

7 Despliegue e Instrucciones de Uso

7.1 Requisitos Previos

- Docker y Docker Compose (recomendado)
- Alternativamente: Python 3.11+, Node.js 20+

7.2 Despliegue con Docker (Recomendado)

```
git clone https://github.com/lucckkas/tallerfinal.git && cd tallerfinal
cp .env.example .env
docker compose up --build
# Frontend: http://localhost:5173 | Backend: http://localhost:8000
```

7.3 Despliegue Manual

Backend:

```
python -m venv .venv && source .venv/bin/activate
pip install -r ml/requirements.txt -r backend/requirements.txt
PYTHONPATH=ml/src python ml/train.py --config config/config.yaml
PYTHONPATH=ml/src uvicorn backend.app.main:app --reload --port 8000
```

Frontend:

```
cd frontend && npm install && npm run dev
```

7.4 Verificación

```
curl http://localhost:8000/health  # Verificar backend
pytest backend/tests               # Ejecutar tests
```

8 Pruebas y Resultados

8.1 Pruebas de la API

Se implementaron tests automatizados usando `pytest` y `TestClient` de FastAPI:

Cuadro 6: Cobertura de tests de la API

Test	Descripción	Estado
test_health	Verifica endpoint /health	✓
test_model_info	Verifica endpoint /model-info	✓
test_predict	Verifica predicción con archivo .log	✓
test_evaluate	Verifica evaluación con archivo etiquetado	✓

8.2 Prueba End-to-End

Se validó el flujo completo usando los archivos de los sujetos demo (1 y 8):

1. Subir archivo `mHealth_subject1.log` al frontend
2. Verificar que las predicciones se muestran correctamente
3. Evaluar el archivo y verificar métricas (accuracy $\approx$ 99%)
4. Confirmar que la matriz de confusión se visualiza sin errores

8.3 Resultados de Evaluación en Demo

Al evaluar los sujetos demo que nunca participaron en entrenamiento:

- **Accuracy:** 99.3%
- **Macro F1-Score:** 99.1%
- Matriz de confusión casi diagonal, con mínima confusión entre actividades similares (ej: Trote vs. Corriendo)

9 Uso de Inteligencia Artificial Generativa

9.1 Herramientas Utilizadas

Se utilizó **GitHub Copilot** y modelos de lenguaje (LLMs) como asistente durante el desarrollo. La documentación completa de prompts se encuentra en la carpeta `prompts/`.

9.2 Evolución del Prompt

El desarrollo siguió un proceso iterativo de refinamiento:

1. **Iteración 1:** Solicitud inicial genérica → Faltaba especificación de split y funcionalidad de carga
2. **Iteración 2:** Inclusión de sujetos excluidos para demo → Prompt demasiado complejo
3. **Iteración 3:** Simplificación a un solo modelo y entrada solo por `.log` → Interfaz básica
4. **Iteración 4:** Mejora de UI/UX → Problema con actividad 0
5. **Iteración 5:** Filtrado de actividad 0 → Versión final funcional

9.3 Lecciones Aprendidas

- Los prompts demasiado extensos generan resultados menos enfocados
- La iteración incremental produce mejores resultados que especificar todo de una vez
- Es fundamental revisar y ajustar manualmente el código generado
- La IA acelera el desarrollo pero requiere supervisión experta

10 Reflexión y Mejoras Futuras

10.1 Rol del Despliegue en el Ciclo de Vida de Ciencia de Datos

El despliegue constituye la fase donde el valor del modelo se materializa. Sin una infraestructura de despliegue adecuada:

- Los modelos permanecen como artefactos de investigación sin impacto real
- La brecha entre desarrollo y producción genera inconsistencias
- Los usuarios finales no pueden beneficiarse de los avances técnicos

Este proyecto demuestra cómo integrar ML con desarrollo de software siguiendo prácticas de MLOps: versionado de artefactos, reproducibilidad mediante configuración centralizada, y containerización para consistencia entre entornos.

10.2 Mejoras Futuras

1. **Explicabilidad (XAI):** Incorporar SHAP o LIME para explicar predicciones individuales
2. **Modelos adicionales:** Experimentar con redes neuronales (CNN-1D, LSTM) para comparación
3. **Streaming en tiempo real:** Adaptar la API para procesar flujos continuos de sensores
4. **Autenticación:** Agregar JWT para proteger endpoints sensibles
5. **Monitoreo:** Implementar logging estructurado y métricas de Prometheus
6. **CI/CD:** Pipeline automatizado con GitHub Actions para tests y despliegue

11 Conclusiones

Se desarrolló exitosamente un sistema completo de reconocimiento de actividad humana que integra:

- Un pipeline de ML reproducible con accuracy del 92.6% en validación
- Una API REST robusta con endpoints para predicción y evaluación
- Una interfaz web moderna y profesional accesible para usuarios no técnicos
- Documentación completa de despliegue y uso

El proyecto demuestra la viabilidad de llevar modelos de ML a producción mediante arquitecturas modulares y herramientas modernas de desarrollo. La eliminación de la actividad 0 fue crucial para obtener un modelo funcional, evidenciando la importancia del análisis de datos previo al modelado.

El sistema está listo para uso en demostración y podría extenderse para aplicaciones reales en monitoreo de salud, deportes o rehabilitación con las mejoras propuestas.

Referencias

[1] Banos, O., Garcia, R., Holgado-Terriza, J.A., et al. (2014). mHealthDroid: A Novel Framework for Agile Development of Mobile Health Applications. *Ambient Assisted Living and Daily Activities*.

[2] UCI Machine Learning Repository. MHealth Dataset. `https://archive.ics.uci.edu/dataset/319/mhealth+dataset`

[3] Ramírez, S. (2018). FastAPI Documentation. `https://fastapi.tiangolo.com/`

[4] React Documentation. `https://react.dev/`

[5] Scikit-learn: Machine Learning in Python. `https://scikit-learn.org/`

Anexos

A. Configuración Completa (config.yaml)

```yaml
version: "1.0.0"
random_seed: 42
sample_rate_hz: 50
window_seconds: 5
window_overlap_seconds: 2.5
excluded_subjects_demo:
    - 1
    - 8
train_val_test_split:
    train_ratio: 0.6
    val_ratio: 0.2
    test_ratio: 0.2
features:
    stats: [mean, std, min, max, median, mad, energy]
model:
    type: random_forest
    n_estimators: 200
    max_depth: null
    class_weight: balanced
artifacts:
    dir: ml/artifacts
    model_path: ml/artifacts/model.joblib
    feature_metadata: ml/artifacts/features.json
    metrics: ml/artifacts/metrics.json
    model_info: ml/artifacts/model_info.json
```

Listing 1: Archivo de configuración config/config.yaml

B. Repositorio del Proyecto

El código fuente completo está disponible en:
`https://github.com/lucckkas/tallerfinal.git`