Reconocimiento de Actividad Humana (MHealth) – Informe

Resumen

Aplicación web que recibe archivos `.log` del dataset MHealth y devuelve la actividad humana predicha por un modelo entrenado (RandomForest). Incluye backend FastAPI, frontend HTML/Bootstrap y despliegue con Docker Compose. Se validan automáticamente separador, columnas y presencia de Label/Subject; se devuelven errores claros para un usuario no técnico.

Contexto y objetivo

- Dataset: MHealth; series temporales de sensores corporales.
- Problema: reconocer actividad humana en ventanas de tiempo.
- Objetivo del despliegue: entregar una app usable (backend+frontend) que procese logs reales del dataset y muestre resultados claros.

Arquitectura

```
[Frontend HTML/Bootstrap] --> POST /detect (FastAPI)
                              GET /health
                                   |
                                   v
                        [Pipeline de validación]
                                   |
                                   v
                    [model_pipeline.pkl (joblib)]
```

- Contenedores: `backend` (uvicorn) y `frontend` (nginx) en la misma red Docker.
- CORS habilitado para consumo local (8080 → 8000).

Backend

- Framework: FastAPI.
- Endpoints:
  - `GET /health` → {"status":"ok","message":"backend listo"}
  - `POST /detect` (form-data, clave `file`):
    - Validaciones: separador `\t` o espacios; 23 columnas de señales + Label/Subject opcional; archivo no vacío; decodificación UTF-8/Latin-1.
    - Respuesta 200:

```
{
  "prediction": [...],
  "predicted_labels": ["Walking", ...],
  "samples": N,
```

```
    "preprocessing": {
      "separator": "\\t",
      "columns_detected": 24,
      "label_column": "Label"
    },
    "status": "complete"
  }
```

    - Errores 400: archivo vacío, separador no detectado, número de columnas inválido.
- Manejo de errores: try/except → HTTPException 400 o 500.
- Mapeo de clases:
    - 0 Standing, 1 Jogging, 2 Running, 3 Walking, 4 Upstairs, 5 Downstairs, 6 Sitting, 7 Lying, 8 Cycling.
- Dependencias clave: `fastapi`, `uvicorn`, `pandas`, `numpy==1.23.5`, `scikit-learn==1.0.2`, `python-multipart`, `joblib`.

Pipeline y modelo

- `process_log_file`:
    - Detecta separador (`\t` o espacios).
    - Asume 23 features exactas (verificadas con `feature_names_in_` del modelo).
    - Si hay 24 columnas, intenta clasificar la última como Label (valores enteros en rango) o Subject (entero constante).
    - Devuelve DataFrame con 23 columnas y metadatos de preprocesamiento.
- Modelo: `model_pipeline.pkl` (StandardScaler + RandomForestClassifier, entrenado con scikit-learn 1.0.2).

Frontend

- HTML + Bootstrap servido por nginx.
- Flujo:
    1. Selección de archivo `.log`.
    2. Botón “Enviar”.
    3. Spinner “Procesando…”.
    4. Alertas de error (detalle de backend).
    5. Resumen: ventanas procesadas, actividades detectadas, separador, última columna.
    6. Conteo por actividad y actividad dominante.
    7. JSON completo (scroll).
- Detecta host backend: `localhost:8000` en local o `backend:8000` en Docker.

Despliegue

- Comando principal: `docker-compose up --build`
    - Frontend: http://localhost:8080
    - Backend (docs): http://localhost:8000/docs
- Sin Docker (rápido):

```
cd backend
pip install -r requirements.txt
uvicorn app:app --reload  # backend en 8000
# abrir frontend/index.html en navegador
```

- Notas: el modelo (`backend/model/model_pipeline.pkl`) está en .gitignore; colocarlo manualmente antes de levantar.

Pruebas rápidas

- Salud:

```
curl http://localhost:8000/health
```

- Detección con log de ejemplo:

```
curl -X POST -F "file=@test_logs/subject1.log" http://localhost:8000/detect
```

- Casos de error (espera 400 con mensaje claro):
  - Archivo vacío.
  - Archivo con separador no detectable.
  - Archivo con ≠23/24 columnas.

Resultados observados

- Ejemplo real (archivo MHealth):
  - `samples`: 161280
  - `separator`: `\t`
  - `columns_detected`: 24, `label_column`: "Label"
  - Actividades dominantes: Lying/Sitting (según el log usado).

Riesgos y mejoras futuras

- Riesgo: versión de sklearn/numpy distinta al entrenar → incompatibilidad del pickle.
- Mejora frontend: colapsar JSON; agregar validación de extensión antes de enviar.
- Mejora backend: limitar tamaño de archivo y tiempo de ejecución; añadir logging estructurado.
- CI: agregar test rápido `test_pipeline.py` (shape y validación de columnas).

Uso de IA generativa

- Carpeta `prompts/` con los prompts utilizados para asistencia en código y documentación.

Referencias

- Dataset MHealth.

- Documentación FastAPI, scikit-learn (persistencia de modelos), Docker Compose.