Material Suplementario

Sistema de Reconocimiento de Actividad Humana - MHealth

Anexos y Código Fuente

Proyecto Final

Diciembre 2025

Índice

1 Código del Pipeline de Machine Learning

1.1 Módulo de Constantes (constants.py)

```python
from __future__ import annotations
import pathlib

DATASET_URL = "https://archive.ics.uci.edu/static/public/319/mhealth+dataset.zip"

RAW_DIR = pathlib.Path("ml/data/raw")
PROCESSED_DIR = pathlib.Path("ml/data/processed")

# Column names from the official MHEALTH dataset description.
SENSOR_COLUMNS = [
    # Chest Accelerometer
    "acc_chest_x", "acc_chest_y", "acc_chest_z",
    # Two ECG channels
    "ecg_1", "ecg_2",
    # Left ankle accelerometer
    "acc_ankle_x", "acc_ankle_y", "acc_ankle_z",
    # Left ankle gyroscope
    "gyro_ankle_x", "gyro_ankle_y", "gyro_ankle_z",
    # Left ankle magnetometer
    "mag_ankle_x", "mag_ankle_y", "mag_ankle_z",
    # Right arm accelerometer
    "acc_arm_x", "acc_arm_y", "acc_arm_z",
    # Right arm gyroscope
    "gyro_arm_x", "gyro_arm_y", "gyro_arm_z",
    # Right arm magnetometer
    "mag_arm_x", "mag_arm_y", "mag_arm_z",
]

LABEL_COLUMN = "activity"
SUBJECT_COLUMN = "subject"
TIMESTAMP_COLUMN = "timestamp"

ACTIVITY_MAP = {
    0: "Sin clasificar",
    1: "De pie",
    2: "Sentado",
    3: "Acostado",
    4: "Caminando",
    5: "Subiendo escaleras",
    6: "Flexion de cintura",
    7: "Elevacion frontal de brazos",
    8: "Flexion de rodillas",
    9: "Ciclismo",
    10: "Trote",
    11: "Corriendo",
    12: "Saltando",
}

ALL_COLUMNS = SENSOR_COLUMNS + [LABEL_COLUMN]
```

Listing 1: ml/src/mhealth/constants.py

1.2 Módulo de Preprocesamiento (preprocess.py)

```python
from __future__ import annotations
import collections
from typing import Dict, List, Sequence, Tuple
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from .config import Config
from .constants import ACTIVITY_MAP, LABEL_COLUMN, SENSOR_COLUMNS, SUBJECT_COLUMN

def filter_demo_subjects(
    df: pd.DataFrame, excluded: Sequence[int]
) -> Tuple[pd.DataFrame, pd.DataFrame]:
    mask = df[SUBJECT_COLUMN].isin(excluded)
    demo = df[mask].copy()
    remaining = df[˜mask].copy()
    return remaining, demo

def filter_unlabeled_activity(df: pd.DataFrame) -> pd.DataFrame:
    """
    Remove activity 0 (unlabeled/null activity) from dataset.
    This prevents extreme class imbalance issues during training.
    """
    return df[df[LABEL_COLUMN] != 0].copy()

def split_by_subject(
    df: pd.DataFrame, config: Config
) -> Tuple[pd.DataFrame, pd.DataFrame, pd.DataFrame]:
    subjects = sorted(df[SUBJECT_COLUMN].unique())
    rng = np.random.default_rng(config.random_seed)
    rng.shuffle(subjects)
    total = len(subjects)
    val_n = max(1, int(round(total * config.train_val_test_split.val_ratio)))
    test_n = max(1, int(round(total * config.train_val_test_split.test_ratio)))
    train_n = max(1, total - val_n - test_n)

    train_subj = subjects[:train_n]
    val_subj = subjects[train_n : train_n + val_n]
    test_subj = subjects[train_n + val_n : train_n + val_n + test_n]

    train_df = df[df[SUBJECT_COLUMN].isin(train_subj)].copy()
    val_df = df[df[SUBJECT_COLUMN].isin(val_subj)].copy()
    test_df = df[df[SUBJECT_COLUMN].isin(test_subj)].copy()

    return train_df, val_df, test_df

def _window_indices(n_samples: int, window_size: int, step_size: int) -> List[Tuple[int, int]]:
    indices = []
    start = 0
    while start + window_size <= n_samples:
        end = start + window_size
        indices.append((start, end))
        start += step_size
    return indices

def create_windows(
    df: pd.DataFrame,
    window_seconds: float,
    overlap_seconds: float,
    sample_rate_hz: int,
    feature_stats: Sequence[str] | None = None,
) -> pd.DataFrame:
    window_size = int(window_seconds * sample_rate_hz)
    overlap = int(overlap_seconds * sample_rate_hz)
    step = max(1, window_size - overlap)
    rows: List[dict] = []

    for subject_id, group in df.groupby(df[SUBJECT_COLUMN]):
        group = group.sort_values("timestamp")
        idxs = _window_indices(len(group), window_size, step)
        for start, end in idxs:
            window = group.iloc[start:end]
            label_mode = window[LABEL_COLUMN].mode()
            label = int(label_mode.iloc[0]) if not label_mode.empty else None
            feature_row = extract_features(window[SENSOR_COLUMNS], feature_stats=feature_stats)
            feature_row[LABEL_COLUMN] = label
            feature_row[SUBJECT_COLUMN] = subject_id
            rows.append(feature_row)

    return pd.DataFrame(rows)

def extract_features(
    window_df: pd.DataFrame, feature_stats: Sequence[str] | None = None
) -> Dict[str, float]:
    stats = feature_stats or ["mean", "std", "min", "max", "median", "mad", "energy"]
    features: Dict[str, float] = {}
    for col in window_df.columns:
        values = window_df[col].values
        if "mean" in stats:
            features[f"{col}__mean"] = float(np.mean(values))
        if "std" in stats:
            features[f"{col}__std"] = float(np.std(values))
        if "min" in stats:
            features[f"{col}__min"] = float(np.min(values))
        if "max" in stats:
            features[f"{col}__max"] = float(np.max(values))
        if "median" in stats:
            features[f"{col}__median"] = float(np.median(values))
        if "mad" in stats:
            mad = float(np.median(np.abs(values - np.median(values))))
            features[f"{col}__mad"] = mad
        if "energy" in stats:
            energy = float(np.sum(values**2) / len(values))
            features[f"{col}__energy"] = energy
    return features

def build_feature_matrix(df_windows: pd.DataFrame) -> Tuple[pd.DataFrame, pd.Series]:
    feature_cols = [c for c in df_windows.columns if c not in (LABEL_COLUMN, SUBJECT_COLUMN)]
    X = df_windows[feature_cols].copy()
    y = df_windows[LABEL_COLUMN].astype(int).copy()
    return X, y

def fit_scaler(train_features: pd.DataFrame) -> StandardScaler:
    scaler = StandardScaler()
    scaler.fit(train_features)
    return scaler

def transform_features(scaler: StandardScaler, features: pd.DataFrame) -> np.ndarray:
    return scaler.transform(features)

def activity_distribution(preds: Sequence[int]) -> Dict[str, int]:
    counter = collections.Counter(preds)
    return {ACTIVITY_MAP.get(k, str(k)): int(v) for k, v in counter.items()}
```

Listing 2: ml/src/mhealth/preprocess.py

1.3 Módulo de Modelado (modeling.py)

```python
def train_model(
    df: pd.DataFrame,
    config: Config,
    demo_df: pd.DataFrame = None,
) -> Dict[str, object]:
    """
    Train model on provided dataframe.
    """
    set_global_seed(config.random_seed)

    # Remove activity 0 (unlabeled) to prevent class imbalance
    df = filter_unlabeled_activity(df)

    # Verify no demo subjects leaked into training data
    training_subjects = set(df[SUBJECT_COLUMN].unique())
    demo_subjects = set(config.excluded_subjects_demo)
    leaked = training_subjects & demo_subjects
    if leaked:
        raise ValueError(
            f"DATA LEAK DETECTED: Demo subjects {leaked} found in training data!"
        )

    print(f"[SECURITY] Training subjects: {sorted(training_subjects)}")
    print(f"[SECURITY] Excluded subjects (demo): {sorted(demo_subjects)}")
    print(f"[SECURITY] Verification OK: No data leakage")

    train_df_raw, val_df_raw, test_df_raw = split_by_subject(df, config)

    feature_stats = config.features.get("stats")
    train_windows = create_windows(train_df_raw, config.window_seconds,
        config.window_overlap_seconds, config.sample_rate_hz, feature_stats)
    val_windows = create_windows(val_df_raw, config.window_seconds,
        config.window_overlap_seconds, config.sample_rate_hz, feature_stats)
    test_windows = create_windows(test_df_raw, config.window_seconds,
        config.window_overlap_seconds, config.sample_rate_hz, feature_stats)

    # Process demo data if provided
    if demo_df is not None and len(demo_df) > 0:
        demo_df_filtered = filter_unlabeled_activity(demo_df)
        demo_windows = create_windows(demo_df_filtered, config.window_seconds,
            config.window_overlap_seconds, config.sample_rate_hz, feature_stats)
    else:
        demo_windows = pd.DataFrame()

    X_train, y_train = build_feature_matrix(train_windows)
    X_val, y_val = build_feature_matrix(val_windows)
    X_test, y_test = build_feature_matrix(test_windows)
    X_demo, y_demo = build_feature_matrix(demo_windows)

    scaler = StandardScaler()
    clf = RandomForestClassifier(
        n_estimators=config.model.n_estimators,
        max_depth=config.model.max_depth,
        random_state=config.random_seed,
        class_weight=config.model.class_weight,
        n_jobs=-1,
    )

    pipeline = Pipeline([("scaler", scaler), ("clf", clf)])
    pipeline.fit(X_train, y_train)

    metrics = {
        "val": compute_metrics(pipeline, X_val, y_val),
        "test": compute_metrics(pipeline, X_test, y_test),
        "demo": compute_metrics(pipeline, X_demo, y_demo),
        "train": compute_metrics(pipeline, X_train, y_train),
    }

    artifacts = {
        "pipeline": pipeline,
        "feature_columns": list(X_train.columns),
        "metrics": metrics,
        "splits": {
            "train_subjects": sorted(train_df_raw[SUBJECT_COLUMN].unique().tolist()),
            "val_subjects": sorted(val_df_raw[SUBJECT_COLUMN].unique().tolist()),
            "test_subjects": sorted(test_df_raw[SUBJECT_COLUMN].unique().tolist()),
            "demo_subjects": sorted(demo_df[SUBJECT_COLUMN].unique().tolist())
            if demo_df is not None and len(demo_df) > 0
            else config.excluded_subjects_demo,
        },
    }
    return artifacts
```

Listing 3: ml/src/mhealth/modeling.py - Función principal de entrenamiento

2 Código del Backend (FastAPI)

2.1 Endpoints Principales (main.py)

```
from __future__ import annotations
import pathlib
from typing import Annotated
from fastapi import Depends, FastAPI, File, HTTPException, UploadFile
from fastapi.middleware.cors import CORSMiddleware
from .config import load_settings
from .schemas import (
    AggregatePrediction, EvaluateResponse, HealthResponse,
    ModelInfo, PredictResponse, WindowPrediction,
)
from .service import ModelService

settings = load_settings()
try:
    service = ModelService(settings)
except FileNotFoundError:
    service = None

app = FastAPI(
    title="MHealth HAR API",
    version="1.0.0",
    description="API de reconocimiento de actividad humana usando MHealth.",
)

origins = [o.strip() for o in settings.allowed_origins.split(",")]
app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

@app.get("/health", response_model=HealthResponse)
def health() -> HealthResponse:
    return HealthResponse(status="ok")

def _get_service() -> ModelService:
    if service is None:
        raise HTTPException(status_code=500, detail="Modelo no disponible.")
    return service

@app.get("/model-info", response_model=ModelInfo)
def model_info(svc: ModelService = Depends(_get_service)) -> ModelInfo:
    payload = svc.model_info_payload()
    return ModelInfo(**payload)

def _validate_file(file: UploadFile) -> None:
    if not file.filename:
        raise HTTPException(status_code=400, detail="Archivo no proporcionado.")
    if not file.filename.endswith(".log"):
        raise HTTPException(status_code=400, detail="Solo se aceptan archivos .log.")

@app.post("/predict", response_model=PredictResponse)
async def predict(
    file: UploadFile = File(...), svc: ModelService = Depends(_get_service)
) -> PredictResponse:
    _validate_file(file)
    result = svc.predict(file)
    return PredictResponse(**result)

@app.post("/evaluate-log", response_model=EvaluateResponse)
async def evaluate_log(
    file: UploadFile = File(...), svc: ModelService = Depends(_get_service)
) -> EvaluateResponse:
    _validate_file(file)
    result = svc.evaluate(file)
    return EvaluateResponse(
        metrics=result["metrics"],
        predictions=result["predictions"],
        ground_truth=result["ground_truth"],
    )
```

Listing 4: backend/app/main.py

2.2 Esquemas Pydantic (schemas.py)

```python
from __future__ import annotations
from typing import Dict, List, Optional
from pydantic import BaseModel

class HealthResponse(BaseModel):
    status: str

class WindowPrediction(BaseModel):
    window_index: int
    prediction: int
    activity: str
    proba: Dict[str, float]

class AggregatePrediction(BaseModel):
    fraction_per_activity: Dict[str, float]
    mean_proba: Dict[str, float]

class PredictResponse(BaseModel):
    per_window: List[WindowPrediction]
    aggregate: AggregatePrediction

class ModelInfo(BaseModel):
    version: str
    model_type: str
    random_seed: int
    window_seconds: float
    window_overlap_seconds: float
    sample_rate_hz: int
    excluded_subjects_demo: List[int]
    splits: dict
    feature_columns: List[str]
    metrics: Optional[dict]

class EvaluationMetrics(BaseModel):
    accuracy: Optional[float]
    macro_f1: Optional[float]
    confusion_matrix: List[List[int]]

class EvaluateResponse(BaseModel):
    metrics: EvaluationMetrics
    predictions: Optional[List[int]] = None
    ground_truth: Optional[List[int]] = None
```

Listing 5: backend/app/schemas.py

3 Configuración de Docker

3.1 Dockerfile del Backend

```
FROM python:3.11-slim

WORKDIR /app

COPY ml/requirements.txt ./ml-requirements.txt
COPY backend/requirements.txt ./backend-requirements.txt
RUN pip install --no-cache-dir -r ml-requirements.txt -r backend-requirements.txt

COPY config ./config
COPY ml ./ml
COPY backend ./backend

ENV PYTHONPATH="/app/ml/src"

EXPOSE 8000

CMD ["uvicorn", "backend.app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Listing 6: backend/Dockerfile

3.2 Dockerfile del Frontend

```
FROM node:20-alpine AS build
WORKDIR /app
COPY frontend/package*.json ./
COPY frontend/tsconfig*.json ./
COPY frontend/vite.config.ts ./
COPY frontend/.eslintrc.cjs ./
RUN npm install
COPY frontend/. ./
RUN npm run build

FROM node:20-alpine
WORKDIR /app
COPY --from=build /app/dist ./dist
RUN npm install -g serve
EXPOSE 5173
CMD ["serve", "-s", "dist", "-l", "5173"]
```

Listing 7: frontend/Dockerfile

3.3 Docker Compose

```
services:
    backend:
        build:
            context: .
            dockerfile: backend/Dockerfile
        env_file: .env
        ports:
            - "8000:8000"
        volumes:
            - ./config:/app/config
            - ./ml/artifacts:/app/ml/artifacts
        environment:
            - PYTHONPATH=/app/ml/src

    frontend:
        build:
            context: .
            dockerfile: frontend/Dockerfile
        environment:
            - VITE_API_URL=http://localhost:8000
        ports:
            - "5173:5173"
        depends_on:
            - backend
```

Listing 8: docker-compose.yml

4 Lista Completa de Características Extraídas

El modelo utiliza 161 características (23 sensores × 7 estadísticas). A continuación se presenta la lista completa:

Sensor	**Características**
acc_chest_x	mean, std, min, max, median, mad, energy
acc_chest_y	mean, std, min, max, median, mad, energy
acc_chest_z	mean, std, min, max, median, mad, energy
ecg_1	mean, std, min, max, median, mad, energy
ecg_2	mean, std, min, max, median, mad, energy
acc_ankle_x	mean, std, min, max, median, mad, energy
acc_ankle_y	mean, std, min, max, median, mad, energy
acc_ankle_z	mean, std, min, max, median, mad, energy
gyro_ankle_x	mean, std, min, max, median, mad, energy
gyro_ankle_y	mean, std, min, max, median, mad, energy
gyro_ankle_z	mean, std, min, max, median, mad, energy
mag_ankle_x	mean, std, min, max, median, mad, energy
mag_ankle_y	mean, std, min, max, median, mad, energy
mag_ankle_z	mean, std, min, max, median, mad, energy
acc_arm_x	mean, std, min, max, median, mad, energy
acc_arm_y	mean, std, min, max, median, mad, energy
acc_arm_z	mean, std, min, max, median, mad, energy
gyro_arm_x	mean, std, min, max, median, mad, energy
gyro_arm_y	mean, std, min, max, median, mad, energy
gyro_arm_z	mean, std, min, max, median, mad, energy
mag_arm_x	mean, std, min, max, median, mad, energy
mag_arm_y	mean, std, min, max, median, mad, energy
mag_arm_z	mean, std, min, max, median, mad, energy

Cuadro 1: Características extraídas por sensor

5 Dependencias del Proyecto

5.1 Dependencias de Machine Learning (ml/requirements.txt)

```
numpy>=1.24
pandas>=2.0
scikit-learn>=1.3
joblib>=1.3
requests>=2.31
pyyaml>=6.0
```

5.2 Dependencias del Backend (backend/requirements.txt)

```
fastapi>=0.109
uvicorn[standard]>=0.27
pydantic>=2.5
python-multipart>=0.0.6
pyyaml>=6.0
pytest>=7.4
httpx>=0.26
```

5.3 Dependencias del Frontend (frontend/package.json)

```
{
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0"
  },
  "devDependencies": {
    "@types/react": "^18.2.43",
    "@types/react-dom": "^18.2.17",
    "@typescript-eslint/eslint-plugin": "^6.14.0",
    "@typescript-eslint/parser": "^6.14.0",
    "@vitejs/plugin-react": "^4.2.1",
    "eslint": "^8.55.0",
    "eslint-plugin-react-hooks": "^4.6.0",
    "eslint-plugin-react-refresh": "^0.4.5",
    "typescript": "^5.2.2",
    "vite": "^5.0.8"
  }
}
```