Estructura de Datos del Proyecto HAR

Origen, Contenido y Justificación de Uso

Proyecto Human Activity Recognition

1. Origen del Dataset

El dataset proviene del proyecto **UCI HAR Dataset** (Human Activity Recognition Using Smartphones), disponible en:

https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones

Contexto del Experimento

- **Participantes**: 30 voluntarios (19-48 años)
- **Dispositivo**: Samsung Galaxy S II en la cintura
- **Sensores**: Acelerómetro y giroscopio a 50Hz
- **Actividades**: 6 (caminar, subir/bajar escaleras, sentarse, pararse, acostarse)
- **Ventanas**: 2.56 segundos con 50 % overlap (128 lecturas por ventana)

2. Estructura de Directorios

```
data/
+-- train/
|   +-- data_train/           # 9 archivos de senales crudas
|   |   +-- body_acc_x_train.txt
|   |   +-- body_acc_y_train.txt
|   |   +-- body_acc_z_train.txt
|   |   +-- body_gyro_x_train.txt
|   |   +-- body_gyro_y_train.txt
|   |   +-- body_gyro_z_train.txt
|   |   +-- total_acc_x_train.txt
|   |   +-- total_acc_y_train.txt
|   |   +-- total_acc_z_train.txt
|   +-- X_train.txt           # 561 features pre-calculadas (DESCARTADO)
|   +-- y_train.txt           # Etiquetas de actividad (USADO)
|   +-- subject_train.txt     # ID del sujeto (DESCARTADO)
+-- test/
    +-- data_train/           # Misma estructura que train
    +-- X_test.txt            # (DESCARTADO)
    +-- y_test.txt            # (USADO)
    +-- subject_test.txt      # (DESCARTADO)
```

3. ¿Qué Representan las Columnas?

Es importante entender que las **128 columnas no son variables distintas**, sino **lecturas en el tiempo**:

Col 1	Col 2	Col 3	...	Col 127	Col 128
$t = 0.00$s	$t = 0.02$s	$t = 0.04$s	...	$t = 2.52$s	$t = 2.54$s

Interpretación de cada archivo

Cada archivo representa **una señal de un sensor en un eje**:

- `body_acc_x_train.txt`: Aceleración corporal en eje X, 128 lecturas por muestra
- Cada **fila** = una ventana de 2.56 segundos de esa señal
- Cada **columna** = el valor de la señal en ese instante ($t = \frac{columna-1}{50}$ segundos)
- Los valores son numéricos continuos (ej: -0.234, 0.891, 1.023...)

Ejemplo visual de una fila:

```
Fila 1 de body_acc_x_train.txt:
0.012 -0.023 0.045 0.089 0.123 ... (128 valores) ... 0.034 0.012

  ^                                                    ^
t=0.00s                                              t=2.54s
```

Esto forma una **serie temporal** que la CNN analiza para detectar patrones como pasos, oscilaciones o transiciones.

4. Archivos Utilizados

Archivos que SÍ usamos

3.1 Señales Crudas (9 archivos por conjunto)

Cada archivo contiene una matriz de **N filas × 128 columnas**:

- Cada **fila** = una muestra/ventana temporal
- Cada **columna** = un timestep (128 lecturas a 50Hz = 2.56 segundos)

Archivo	Sensor	Descripción
body_acc_x/y/z	Acelerómetro	Aceleración del cuerpo (sin gravedad), separada por filtro Butterworth
body_gyro_x/y/z	Giroscopio	Velocidad angular en cada eje (rad/s)
total_acc_x/y/z	Acelerómetro	Aceleración total (incluye gravedad)

Dimensiones finales:

- Train: 7,352 muestras × 9 canales × 128 timesteps
- Test: 2,947 muestras × 9 canales × 128 timesteps

5. ¿Por qué usar total_acc Y body_acc?

Una pregunta común es: ¿por qué usar tanto la aceleración **con** gravedad como **sin** gravedad? ¿No es redundante?

Respuesta: Capturan información complementaria

total_acc (con gravedad):

- Mide la aceleración total del dispositivo
- La gravedad ($\sim$9.8 m/s^2) siempre apunta hacia abajo
- Revela la **orientación/postura** del dispositivo y del usuario

body_acc (sin gravedad):

- Se obtiene restando la gravedad (filtro Butterworth de paso bajo)
- Mide solo el **movimiento puro** del usuario
- Revela **actividad dinámica**: pasos, oscilaciones, transiciones

Ejemplo práctico por actividad

Actividad	total_acc	body_acc
LAYING (acostado)	Gravedad en eje Z	≈ 0 (sin movimiento)
STANDING (de pie)	Gravedad en eje Y	≈ 0 (sin movimiento)
SITTING (sentado)	Gravedad en eje Y (inclinado)	≈ 0 (sin movimiento)
WALKING (caminando)	Gravedad en eje Y	Oscilaciones rítmicas (pasos)

¿Por qué ambas son necesarias?

1. **Distinguir actividades estáticas**: LAYING, SITTING y STANDING tienen body_acc ≈ 0, pero total_acc revela orientaciones diferentes.
2. **Distinguir actividades dinámicas**: WALKING, UPSTAIRS y DOWNSTAIRS se diferencian por los patrones en body_acc (ritmo, amplitud).
3. **Combinación poderosa**: La CNN usa ambas señales para aprender características que ninguna podría proveer sola.

Analogía: Es como tener una brújula (total_acc te dice la orientación) y un podómetro (body_acc te dice si hay movimiento). Juntos dan una imagen completa.

6. Etiquetas (y_train.txt, y_test.txt)

Archivos de Etiquetas (USADOS)

Un archivo con **N filas × 1 columna**, donde cada valor indica la actividad:

Código	Actividad
1	WALKING
2	WALKING_UPSTAIRS
3	WALKING_DOWNSTAIRS
4	SITTING
5	STANDING
6	LAYING

Nota: Restamos 1 para obtener índices 0-5 (estándar en PyTorch).

7. Archivos Descartados

Archivos que NO usamos

4.1 X_train.txt / X_test.txt

- **Contenido**: 561 features estadísticas pre-calculadas (media, std, max, min, skewness, FFT, etc.)
- **Dimensión**: N × 561
- **Por qué lo descartamos**:
  1. Queremos que la **CNN aprenda sus propias features** directamente de las señales crudas
  2. Usar features pre-calculadas sería "hacer trampa" – el modelo no estaría haciendo extracción de características
  3. El objetivo pedagógico es entender cómo la convolución detecta patrones

4.2 subject_train.txt / subject_test.txt

- **Contenido**: ID del sujeto (1-30) que realizó cada muestra
- **Por qué lo descartamos**:
  1. No queremos que el modelo dependa de quién hace la actividad
  2. El objetivo es clasificar **actividades**, no **personas**
  3. Incluirlo podría causar data leakage si el mismo sujeto aparece en train y test

8. Procesamiento de Datos

8.1. Carga y Apilamiento

1. Cargar cada uno de los 9 archivos de señales como matriz (N × 128)
2. Apilarlos en un tensor 3D: (N × 9 × 128)
3. Cargar etiquetas y restar 1: $y = y_{original} - 1$

8.2. Conversión a Tensores PyTorch

```
X_train = torch.tensor(X_train, dtype=torch.float32)  # (7352, 9, 128)
```

```
y_train = torch.tensor(y_train, dtype=torch.long)     # (7352,)
```

8.3. División Train/Validation

Del conjunto de entrenamiento (7,352 muestras):

- **80 % Train**: 5,881 muestras
- **20 % Validation**: 1,471 muestras

El conjunto de test (2,947) se mantiene separado para evaluación final.

9. Resumen Visual

Archivo	Dimensión	¿Usado?	Razón
body_acc_*.txt	N × 128	**SÍ**	Señales crudas para CNN
body_gyro_*.txt	N × 128	**SÍ**	Señales crudas para CNN
total_acc_*.txt	N × 128	**SÍ**	Señales crudas para CNN
y_train/test.txt	N × 1	**SÍ**	Etiquetas de clase
X_train/test.txt	N × 561	**NO**	Features pre-calculadas
subject_*.txt	N × 1	**NO**	ID de sujeto (irrelevante)

10. Justificación Final

¿Por qué señales crudas en vez de features pre-calculadas?

1. **Deep Learning aprende features**: La ventaja de CNNs es que aprenden automáticamente qué patrones son relevantes. Usar features manuales elimina esta ventaja.
2. **Propósito pedagógico**: Entender cómo los kernels convolucionales detectan patrones temporales (picos, ciclos, transiciones).
3. **Generalización**: Las features aprendidas pueden generalizar mejor a nuevos datos que las features diseñadas manualmente.
4. **End-to-end learning**: El modelo optimiza desde la señal cruda hasta la predicción final, sin pasos intermedios fijos.