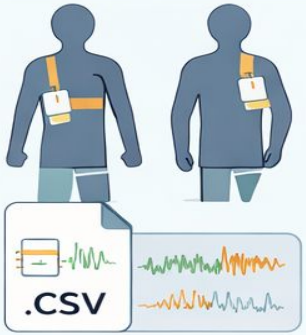
Human Activity Recognition - Dataset HARTH

Giulia Tartaglia Romeo D’angelo

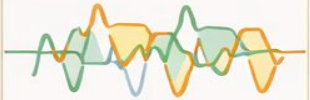
Human Activity Recognition Pipeline

Dataset



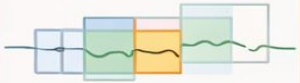
- HARTH Dataset
- 22 subjects
- 2 accelerometers (back and thigh)
- 6 axes (x, y, z per sensor)
- Free-living environment, 50 Hz

Time Segmentation



WINDOW = 100
STEP = 50

Segment signals into overlapped windows of 100 samples (2s) with a step of 50 samples (1s)

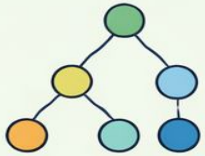


Feature Extraction

- Mean
- Standard Deviation
- Min/Max
- SMA, Energy, etc...

- Compute features like mean, std, min, max, SMA, energy, etc. for each window

Model Training



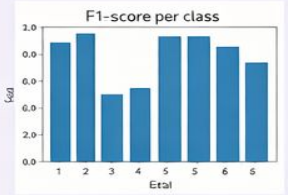
- Gradient Boosted Decision Trees

GBDT

- Train a GBDT classifier using the extracted features from the segmented windows

Evaluation

- Evaluate the model on a test set using accuracy, confusion matrix, and F1-score



Dataset HARTH

- 22 soggetti
- 2 sensori (schiena + coscia)
- 6 assi
- 50 Hz
- contesto free-living

OBIETTIVO: **classificazione multiclasse supervisionata** → associare a ogni segmento temporale del segnale un'attività

```
os.makedirs("harth", exist_ok=True)

with zipfile.ZipFile("harth.zip", "r") as z:
    z.extractall("harth")

csv_files = glob.glob("harth/**/*.csv", recursive=True)
print("CSV trovati:", len(csv_files))
csv_files[:5]
```

```
CSV trovati: 22
['harth/harth/S021.csv',
 'harth/harth/S013.csv',
 'harth/harth/S022.csv',
 'harth/harth/S028.csv',
 'harth/harth/S029.csv']
```

Exploratory data analysis (EDA)

Per capire bene la struttura del dataset abbiamo verificato:

- numero di soggetti
- durata delle registrazioni
- presenza di valori mancanti
- la distribuzione delle attività

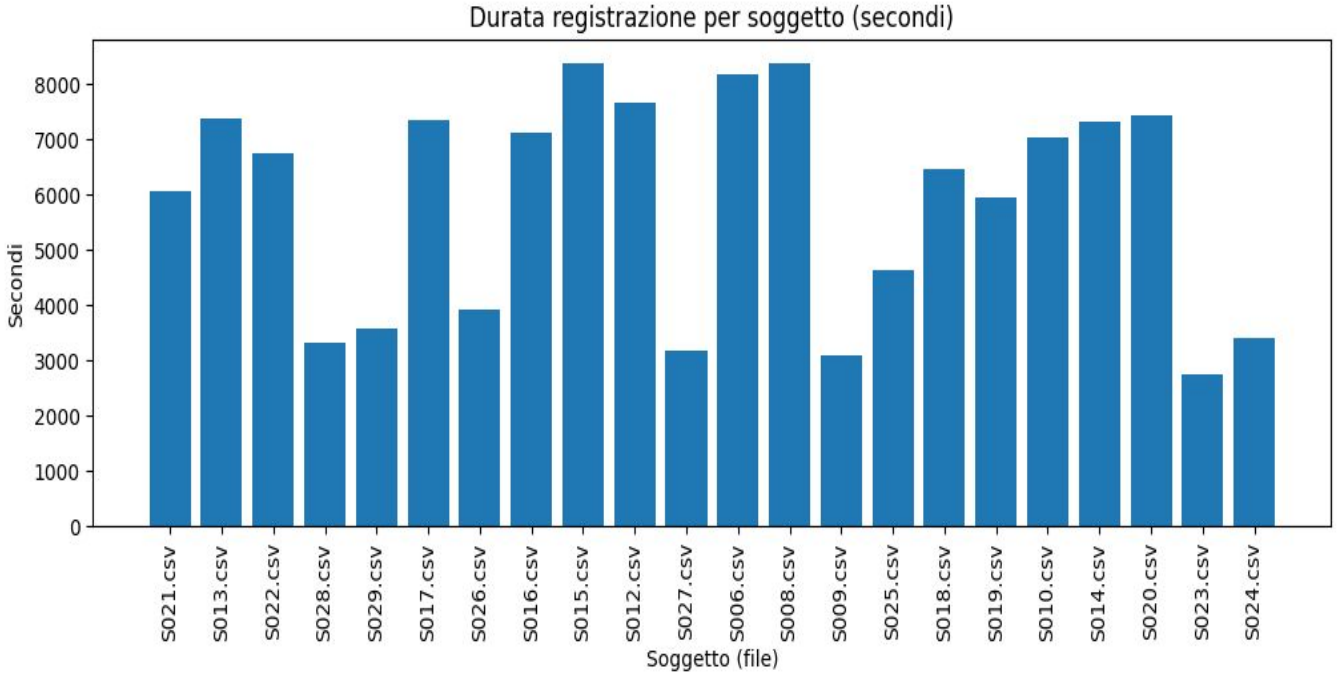
```
CHECK_ALL = False
if CHECK_ALL:
    total_miss = 0
    for fp in csv_files:
        df = pd.read_csv(fp)
        total_miss += int(df.isna().sum().sum())
    print("Missing totali:", total_miss)
```

```
S021.csv missing: 0
S013.csv missing: 0
S022.csv missing: 0
S028.csv missing: 0
S029.csv missing: 0
```

```
SR = 50  # Hz (dato dal dataset)

n_subjects = len(csv_files)
total_rows = 0

for fp in csv_files:
    df = pd.read_csv(fp, usecols=["label"])
    total_rows += len(df)

print("Numero soggetti:", n_subjects)
print("Totale campioni (righe):", total_rows)
print("Sampling rate:", SR, "Hz")
print("Durata totale (min):", total_rows / SR / 60)
```

```
Numero soggetti: 22
Totale campioni (righe): 6461328
Sampling rate: 50 Hz
Durata totale (min): 2153.776
```

Durata registrazione per soggetto



- calcola e visualizza quanto dura la registrazione di ciascun soggetto
- Risponde alla domanda : ***“Tutti i soggetti contribuiscono la stessa quantità di dati?”*** → dal grafico risposta NO
- alcuni soggetti hanno registrazioni molto lunghe
- altri molto più brevi

- Per questo → ***split per soggetto*** (per evitare bias e data leakage)

→ Dato che le durate per soggetto sono diverse, è fondamentale separare train e test per soggetto, altrimenti i soggetti più lunghi dominerebbero entrambi i set

Preprocessing

Successivamente all'EDA abbiamo:

- rimosso colonne non informative come il timestamp
- uniformato il tipo delle label
- escluso le attività di cycling inattivo

OBIETTIVO: permette di semplificare il problema + rende il dataset più coerente con l'obiettivo di riconoscere attività motorie

```
if "timestamp" in df.columns:
    df = df.drop(columns=["timestamp"])
df["label"] = df["label"].astype(int)

# rimozione inattive (opzionale)
if REMOVE_INACTIVE:
    df = df[~df["label"].isin(INACTIVE)].reset_index(drop=True)
```

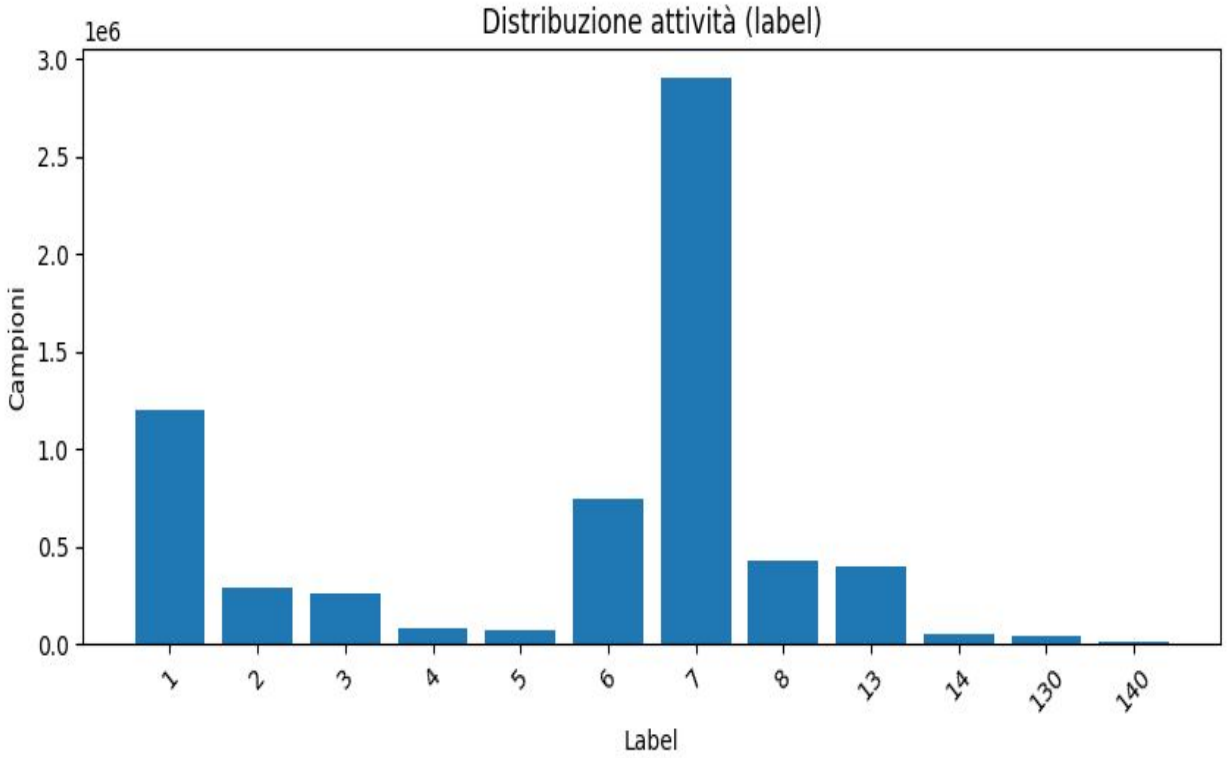
Distribuzione attività (label)

EDA post-preprocessing

- sbilanciamento delle classi :
  - walking, standing hanno più campioni
  - running, stairs sono rare
- dobbiamo gestire questo sbilanciamento

modello deve imparare da pochi esempi per alcune classi e più errori sulle classi rare

- accuracy non è sufficiente → F1 score e confusion matrix



Segnali grezzi

com'è fatto il dato che il modello riceve?; come si comporta il segnale nel tempo?

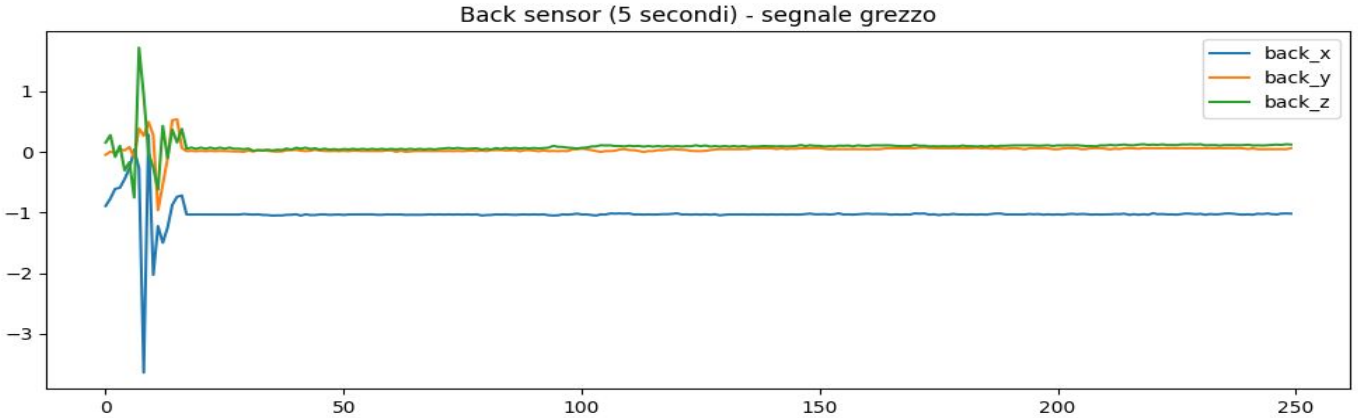
- *vediamo 5 secondi di segnale accelerometrico grezzo sui tre assi*
- *segnale è continuo, rumoroso e varia nel tempo*

Tabella:

Esempio soggetto: S021.csv

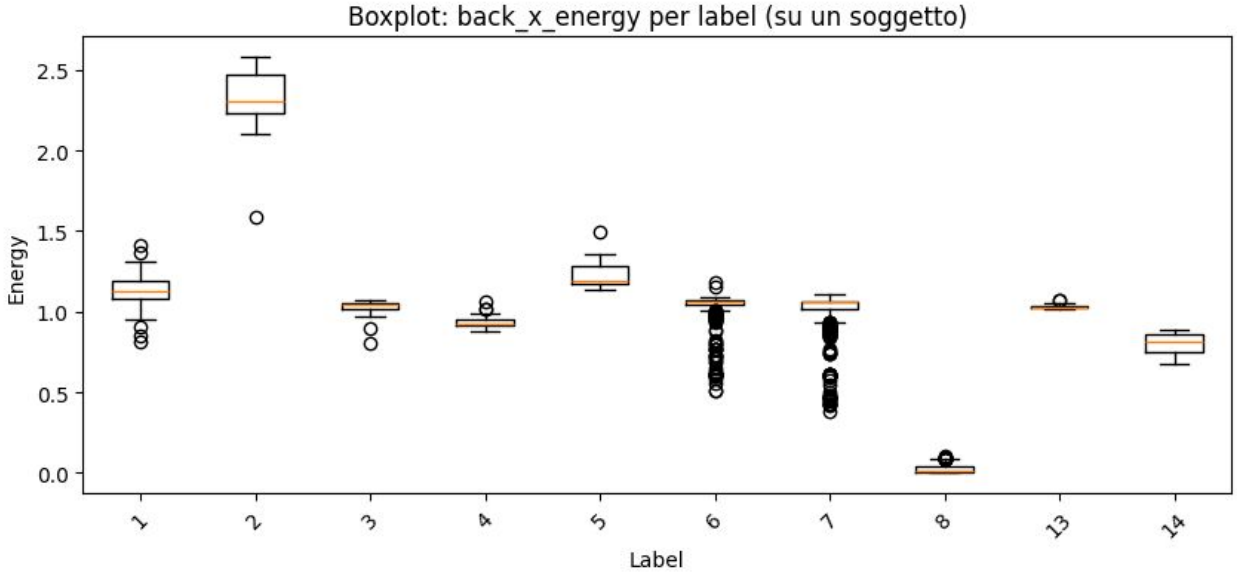
	back_x	back_y	back_z	thigh_x	thigh_y	thigh_z	label
count	302247.000000	302247.000000	302247.000000	302247.000000	302247.000000	302247.000000	302247.000000
mean	-0.929234	0.004081	-0.079903	-0.467381	0.150108	0.483539	5.860634

Grafico:



EDA guidata su finestre temporali - *Energia del segnale*

- **Finestre temporali**: 2 s (100 campioni), senza overlap
- **Feature esplorativa**: energia = mean(x^2) su `back_x`
- **Obiettivo**: verificare separabilità delle attività



Segmentazione temporale e costruzione del dataset

- **Sliding windows**: 2 s (100 campioni), overlap 50%
- **Finestre pure**: scartiamo transizioni di attività
- **Output**: una finestra → un esempio di classificazione

```
WINDOW = 100   # 2 secondi
STEP   = 50    # 1 secondo (overlap 50%)
```

```
# windowing
n = len(df)
for start in range(0, n - WINDOW + 1, STEP):
    w = df.iloc[start:start+WINDOW]

    # tieni solo finestre pure
    if w["label"].nunique() != 1:
        continue

    X_list.append(extract_features(w))
    y_list.append(int(w["label"].iloc[0]))
    g_list.append(subj)
```

Feature engineering

- Prendo ogni finestra di segnale e la trasformi in una riga di numeri
- **DA** 2 secondi di segnale (100 punti × 6 assi) **A** pochi numeri che riassumono quel segnale
- modello lavora su una tabella e non su un segnale continuo
- Statistiche per asse: mean, std, min, max, energy
- Feature di movimento complessivo (SMA)

```python
def extract_features(w: pd.DataFrame) -> np.ndarray:
    feats = []
    for col in SENSOR_COLS:
        x = w[col].to_numpy(dtype=np.float32)
        feats.extend([
            float(x.mean()),
            float(x.std()),
            float(x.min()),
            float(x.max()),
            float(np.mean(x**2))  # energy
        ])
    back = w[["back_x","back_y","back_z"]].to_numpy(dtype=np.float32)
    thigh = w[["thigh_x","thigh_y","thigh_z"]].to_numpy(dtype=np.float32)
    feats.append(float(np.mean(np.abs(back).sum(axis=1))))   # back_SMA
    feats.append(float(np.mean(np.abs(thigh).sum(axis=1))))  # thigh_SMA
    return np.array(feats, dtype=np.float32)
```

Split per soggetto

- Train/Test = 80% / 20%
- Split basato sul soggetto - GroupShuffleSplit
- Nessun data leakage

```
from sklearn.model_selection import GroupShuffleSplit

gss = GroupShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
train_idx, test_idx = next(gss.split(X, y, groups=groups))

X_train, X_test = X[train_idx], X[test_idx]
y_train, y_test = y[train_idx], y[test_idx]
```

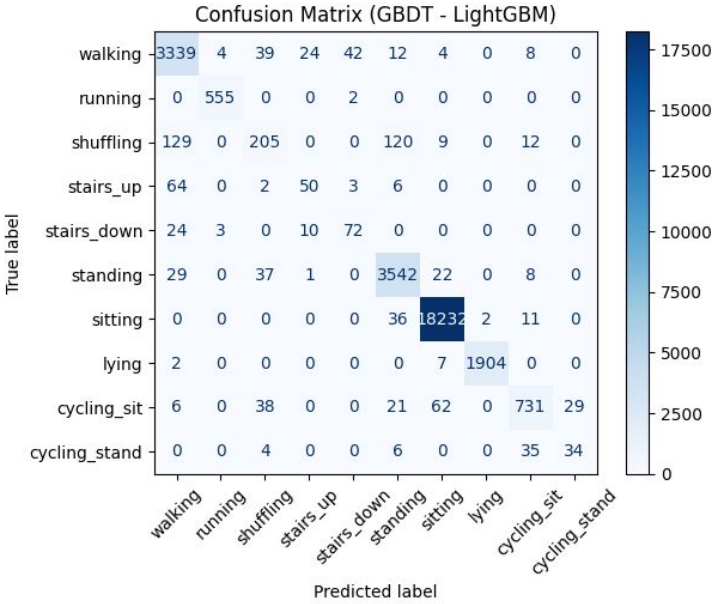
Modello di Classificazione: GBDT

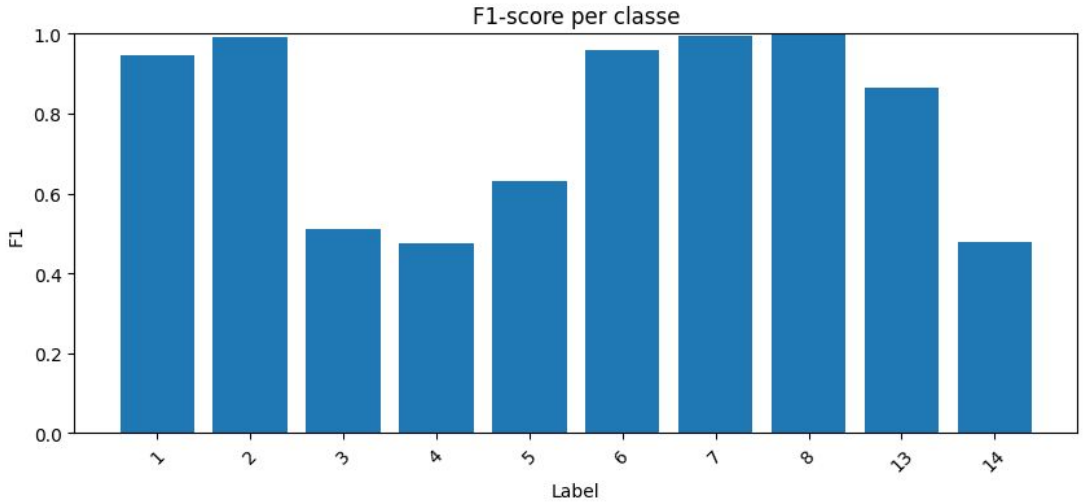
- Modello non lineare su feature tabellari
- Addestramento su train, test su soggetti mai visti
- Buon compromesso tra accuratezza ed interpretabilità

```
!pip install -q lightgbm

import lightgbm as lgb

gbdt = lgb.LGBMClassifier(
    n_estimators=500,
    learning_rate=0.05,
    num_leaves=63,
    subsample=0.9,
    colsample_bytree=0.9,
    random_state=42
)

gbdt.fit(X_train, y_train)
y_pred = gbdt.predict(X_test)
```

Valutazione modello

- Controllo quanto bene il modello riconosce attività su soggetti mai visti
- misuro quanto è bravo e dove sbaglia





Valutazione modello