Step Counting Project Documentation

Vikrant Singh

February 21, 2025

1 Challenges Faced

1. **Lack of Labeled Data:**
   One of the primary challenges was that the dataset did not provide any ground truth labels for the step counts of the left and right feet. To address this, we developed a custom strategy that involved filtering the raw sensor signals using a Butterworth low-pass filter and then applying a peak detection algorithm. The computed peaks were used as surrogate labels for left and right step counts, enabling a semi-supervised approach to model training.

2. **Inconsistent Step Detection in Combined Data:**
   Initially, when processing the entire DataFrame without separation, the peak detection algorithm sometimes detected only a single peak for an event that should have represented both a left and a right step. This led to discrepancies (e.g., 34 steps for the right foot versus 21 for the left), even though normal human gait is nearly synchronized (with a maximum difference of about two steps). To resolve this, we split the data into separate DataFrames for left and right, applying peak detection to each subset individually. This resulted in more consistent and reliable step counts.

3. **Inconsistent Time Format:**
   The raw sensor data contained inconsistent time formats; some entries included sub-second precision while others did not, and occasionally there were missing time values (NaT). To ensure uniformity, we standardized the time column using `pd.to_datetime` with the specified format (`%Y-%m-%d %H:%M:%S.%f`) and proper error handling. This step was essential for correctly sorting the data and computing time differences.

4. **Varying Sampling Frequencies:**
   Different files had varying sampling frequencies. Since the peak detection algorithm relies on the `distance` parameter (defined in samples) to separate distinct step events, discrepancies in sampling rate could lead to inconsistent peak detection. Initially, we considered dynamically adjusting the distance parameter based on each file's sampling frequency. However, after evaluation, we opted to use a consistent fixed value (e.g., 50 samples) for all files to ensure comparability across the dataset.

2 Methodology Overview

The goal of this project is to accurately count the number of steps taken by the left and right foot from raw sensor data. The methodology consists of the following main steps:

1. **Data Preprocessing:**

   - Load the raw sensor data from JSON files.
   - Convert the `time` column to a datetime format using:

```
df['time'] = pd.to_datetime(df['time'], errors='coerce',
                            format='%Y-%m-%d %H:%M:%S.%f')
```

- Sort the DataFrame by the time column.
- Drop rows with missing critical values (i.e., time, accelerometer, and gyroscope readings).

2. **Feature Computation and Filtering:**
   - Compute the acceleration magnitude:

$$\texttt{acc\_mag} = \sqrt{ax^2 + ay^2 + az^2}$$

   - Estimate the sampling frequency (fs) using the median time difference between consecutive samples.
   - Apply a Butterworth low-pass filter to the acceleration magnitude to obtain the filtered signal (`acc_mag_filt`). The filter is defined as:

```
nyq = 0.5 * fs
normal_cutoff = cutoff / nyq
if normal_cutoff >= 1:
    normal_cutoff = 0.99
b, a = butter(order, normal_cutoff, btype='low', analog=False)
acc_mag_filt = filtfilt(b, a, acc_mag)
```

3. **Step Counting via Peak Detection:**
   - Use the filtered acceleration signal to detect peaks that correspond to step events.
   - Since the project demands separate step counts for the left and right foot, the data is split based on the metadata (i.e., `side` field).
   - The peak detection function is defined as follows:

```
def count_steps(df, fileName, prominence=0.2, distance=None, desired_interval=0.3):
    if distance is None:
        dt_median = median(df['delta_time'] where >0)
        fs = 1 / dt_median
        distance = int(fs * desired_interval)
    left_df = filter(df where metadata.side == 'L')
    right_df = filter(df where metadata.side == 'R')
    left_peaks = find_peaks(left_df['acc_mag_filt'], prominence, distance)
    right_peaks = find_peaks(right_df['acc_mag_filt'], prominence, distance)
    return (count(left_peaks), count(right_peaks))
```

   - Here, `desired_interval` (set to 0.3 seconds) is used to compute the minimum number of samples between peaks based on the sampling frequency.

4. **Result Saving:**
   - For each file, save the measurement ID, start time, end time, left step count, and right step count in a JSON file.

3 Algorithm

Below is a high-level pseudocode representation of the methodology:

Algorithm 1 Step Counting Algorithm

procedure ProcessFile($file$)
    $df \leftarrow$ Read JSON file as DataFrame
    Convert `time` column using `pd.to_datetime(..., format=%Y-%m-%d %H:%M:%S.%f)`
    Sort df by `time`
    Drop rows with missing `time, ax, ay, az, gx, gy, gz`
    Compute `delta_time` as time differences between consecutive rows
    Compute acceleration magnitude: $acc\_mag \leftarrow \sqrt{ax^2 + ay^2 + az^2}$
    Compute sampling frequency: $fs \leftarrow 1/\text{median}(delta\_time)$
    Apply Butterworth low-pass filter to acc_mag to obtain acc_mag_filt
    return df
end procedure

procedure CountSteps(df, $prominence$, $desired\_interval$)
    $dt\_median \leftarrow \text{median}(df.delta\_time)$
    $fs \leftarrow 1/dt\_median$
    $distance \leftarrow \lfloor fs \times desired\_interval \rfloor$
    $left\_df \leftarrow \{\text{rows in } df \mid \texttt{metadata.side} = \text{'L'}\}$
    $right\_df \leftarrow \{\text{rows in } df \mid \texttt{metadata.side} = \text{'R'}\}$
    $L \leftarrow$ FIND_PEAKS($left\_df.acc\_mag\_filt$, $prominence$, $distance$)
    $R \leftarrow$ FIND_PEAKS($right\_df.acc\_mag\_filt$, $prominence$, $distance$)
    return $(|L|, |R|)$ ▷ Counts of left and right steps
end procedure

procedure Main
    for all $file \in$ Dataset **do**
        $df \leftarrow$ ProcessFile($file$)
        $(left, right) \leftarrow$ CountSteps(df, $prominence = 0.2$, $desired\_interval = 0.3$)
        Save measurement ID, start time, end time, $left$, $right$ in JSON format
    end for
end procedure

4 Model Architectures

To predict left and right step counts from raw sensor data, we explored two primary approaches: sequential models that process time-series features directly, and aggregated-feature models that rely on summary statistics of the sensor data.

4.1 Sequential Models

These models directly process the 13-dimensional time-series features extracted from the sensor data. The features include:

- **Raw accelerometer readings:** ax, ay, az.
- **Raw gyroscope readings:** gx, gy, gz.
- **Filtered acceleration magnitude:** acc_mag_filt.
- **Gyroscope magnitude:** $gyro\_mag$.
- **Rate-of-change of gyroscope data:** gx_diff, gy_diff, gz_diff.
- **Time delta:** Δt.

- **Side indicator:** Encoded as 0 (left) and 1 (right).

The following four sequential models were implemented:

a) **UnifiedCountLSTM:** A two-layer LSTM network with dropout (0.2) and a two-layer fully-connected network with a ReLU activation. This model was chosen for its capacity to capture long-term dependencies in the time-series data.

b) **SimpleLSTM:** A simplified single-layer LSTM that outputs predictions directly from the last timestep without intermediate activations, reducing model complexity.

c) **CNN-LSTM:** A hybrid model that first applies a 1D convolution (with max pooling) to extract local features and then feeds the result into an LSTM to capture sequential dependencies. This combination leverages both spatial and temporal feature extraction.

d) **UnifiedCountGRU:** A GRU-based network with two layers and dropout (0.2). GRUs are known for faster convergence and fewer parameters, making them an attractive alternative.

4.2 Aggregated-Feature Models

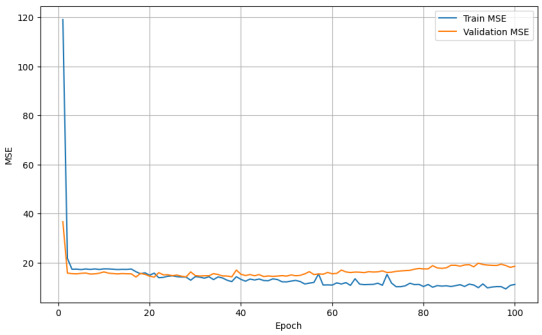
In this approach, the time-series data is aggregated into a fixed-length feature vector by computing summary statistics (mean, standard deviation, maximum, and minimum) for each sensor channel, as well as the measurement duration. In total, 35 features were computed. Two regression models were then employed:

a) **Random Forest Regressor:** Chosen for its robustness and ability to capture non-linear relationships.

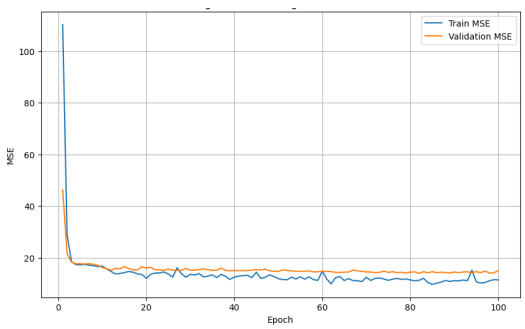
b) **XGBoost Regressor:** An advanced gradient boosting algorithm that effectively models complex interactions in structured data.

5 Training, Validation, and Learning Curves

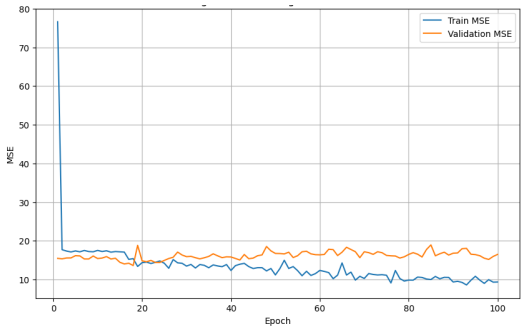
For the sequential models, the dataset was partitioned into training, validation, and test sets. The models were trained for 100 epochs using the Mean Squared Error (MSE) loss function and the Adam optimizer. The learning curves (training vs. validation MSE) for the four sequential models are presented below in a 2x2 grid.



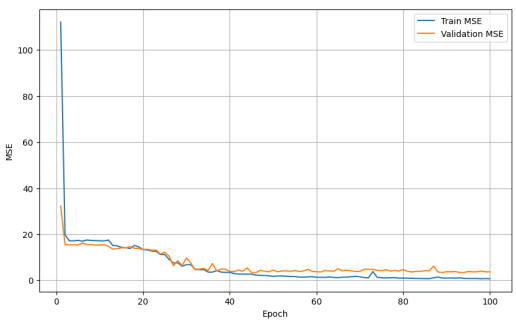
(a) UnifiedCountLSTM



(b) SimpleLSTM



(c) CNN-LSTM



(d) GRU

Figure 1: Learning curves (Training vs. Validation MSE) for the four sequential models.

6 Results and Final Conclusion

The learning curves for the sequential models indicated that the **GRU model achieved the lowest validation error** (final validation MSE ≈ 3.68), suggesting it was the most effective at capturing temporal dependencies in the sensor data. The UnifiedCountLSTM and CNN-LSTM models exhibited higher training and validation errors, while the SimpleLSTM model, despite its reduced complexity, did not outperform the GRU.

In the aggregated-feature approach, the **Random Forest regressor** achieved a Mean Absolute Error (MAE) of **2.055** on the test set, and the **XGBoost** regressor achieved an **MAE of 2.214**. These results demonstrate that while sequential models exploit the full temporal dynamics of the data, aggregated-feature models using summary statistics can also provide competitive performance.