UTRECHT UNIVERSITY

Department of Information and Computing Science

---

**Applied Data Science master thesis**

# Exploring the association between remote monitored physical activity and amyotrophic lateral sclerosis: insights from machine learning

**First examiner:**

Ruben van Eijk

**Candidate:**

Xia Wu

**Second examiner:**

Boudewijn Sleutjes

**In cooperation with:**

UMC Utrecht

3 July, 2024

**Abstract**

Amyotrophic lateral sclerosis (ALS) is a fatal motor neural disease with a highly variable presentation among patients. Diagnosing ALS typically takes over a year from the initial onset of symptoms, often delaying the use of assistive devices like wheelchairs and complicating disease management and treatment. Assessment inconsistencies, due to variations in ALS Functional Rating Scale-Revised (ALSFRS-R) versions, recall bias, and periodic assessments, underscore the need for objective methods to quantify disease progression.

Recent studies have demonstrated that accelerometer-based physical activity monitoring might objectively evaluate disease severity and track progression in ALS patients. Digital biomarkers from accelerometers might enable earlier detection of health status changes, offering clinicians valuable insights into daily functioning to aid treatment. Machine learning research has increasingly linked physical activity with various diseases, offering valuable insights for ALS progression studies.

Our study investigates the relationship between accelerometer-based physical activity features and walking abilities in ALS patients using explainable artificial intelligence techniques. We conducted a detailed analysis to understand how each feature distinguished between categories. Our findings indicate that digital biomarkers extracted from accelerometer data can effectively distinguish ALS patients' walking abilities at different scales using XGBoost. Particularly, bout-based features highlight the model's ability to capture both gross and fine motor functions. Patients with normal walking function exhibit higher levels of moderate-to-vigorous and vigorous activities, while those with impaired walking spend more time in light activities.

In conclusion, accelerometry-derived features, analyzed through machine learning methods, can differentiate walking functions in ALS patients. However, further research is needed to optimize model performance, validate markers in practical applications, explore alternative thresholds, and refine labeling criteria to fully assess functional degradation.

# Contents

# 1. Introduction

Amyotrophic lateral sclerosis (ALS) is a devastating neurodegenerative disorder characterized by inexorable progression. It profoundly affects motor function and quality of life over time, ultimately leading to death by respiratory failure a few years after onset of first symptoms [1]. Despite its incurable nature, supportive and adjunctive therapies offer some relief from symptoms, potentially improving the disease course [2]. Effective utilization of these therapies necessitates healthcare professionals to possess a thorough understanding of patients' disease progression. The ALS Functional Rating Scale-Revised (ALSFRS-R) typically plays an important role in monitoring and managing disease progression by evaluating various behavioral functions affected by ALS [3]. However, the languages and variations in ALSFRS-R versions, the divergent training programs from different certifying organizations and the limitation in detecting small changes in physical functioning can lead to inconsistencies in assessments, which compromise the quality and conclusiveness of the data collected [4, 5, 6].

In particular, the broad use of digital platforms and mobile applications for self-assessment by patients makes it essential to establish objective methods for quantifying disease progression and collection of high-quality data via remote digital tools. Recent studies have shown that accelerometer-based physical activity monitoring is an objective method for evaluating disease severity and measuring disease progression in ALS patients [7, 8]. The accelerometer-derived vertical movement index, has been validated as a highly discriminatory outcome measure for predicting future disease progression rates and strongly associated with overall survival time in patients with ALS [9]. Accelerometer-based outcomes also exhibit excellent correlations with King's staging in ALS patients [7, 9, 10]. These findings highlight the potential of accelerometer-based outcomes to model individual disease courses and predict survival outcomes, similar to the functionality of ALSFRS-R scale [11, 12, 13].

Given the continuous and objective nature of accelerometer-based monitoring, it offers the advantage of enabling continuous home monitoring [14]. This capability allows for the detection and early treatment of changes

in health status, providing clinicians with valuable insights into patients' day-to-day functioning. Understanding patients' body movement and physical activity levels throughout the day is critical, particularly in the context of neurodegenerative or chronic diseases [15].

In recent years, there has been a growing body of machine learning research focused on exploring the relationship between physical activity and various diseases, such as Parkinson's disease, low back pain, and dementia [16, 17, 18]. In the context of ALS, Gupta et al. demonstrated that high response rates for at-home data collection can be achieved using supervised machine learning models [19]. Similarly, Vieira et al. showcased the feasibility of using digital outcomes alongside machine learning to predict self-assessed ALSFRS-R [20]. Both of them found limb movement did not correlate well with bulbar and respiratory function, but can have a better association in gross and fine motor function.

Building on this foundation, our study aims to further analyze how digital biomarkers derived from accelerometry data relate to differences in walking ability among ALS patients, distinguishing between those with mild to serious changes and those with no changes during daily life. By leveraging explainable artificial intelligence techniques, we seek to evaluate the importance of these features and gain a deeper understanding of the key factors driving the classification outcomes. This comprehensive approach not only enhances our understanding of ALS severity in movement aspect, but also holds the potential to inform more timely, personalized and effective treatment strategies.

# 2. Data

## 2.1 Data description

The patient cohort for this study was derived from two prospective cohort studies conducted at the University Medical Centre Utrecht (UMCU) in the Netherlands. Participants were recruited through the Treatment Research Initiative to Cure ALS (TRICALS) database and the UMCU biobank for motor neuron diseases. Patients were eligible for inclusion based on the following criteria: (a) Patients were required to be over the age of 18. (b) Patients were required to have a diagnosis of possible, probable (laboratory- supported), or definite amyotrophic lateral sclerosis (ALS), according to the revised El Escorial criteria [21]. (c) Patients diagnosed with progressive muscular atrophy (PMA) were also included. (e) All participants provided written informed consent.

## 2.2 Data preparation

### 2.2.1 Accelerometer data

The accelerometer data utilized in this study were obtained from prior studies. Patients were provided with the ActiGraph GT9X Link (ActiGraph LLC, Pensacola, FL), a compact (0.5 × 3.5 × 1 cm), lightweight (14 g) tri-axial accelerometer device. They return the device either by mailed or handed it over during visits. Patients were instructed to wear the ActiGraph on the right hip in the anterior axillary line using a belt clip during waking hours for a continuous period of 3 to 7 days. The device was set to collect data at a sampling rate of 30 Hz.

The accelerometer data were distributed to students through SURFfilesender, a secure file-sharing service that stores files on servers in the Netherlands for up to 21 days. This service is recommended by UMC Utrecht and complies with European privacy legislation (GDPR). Students received the data in GT3X file format, comprising a total of 89 files from 89 patients.

### 2.2.2 Metadata

An Excel spreadsheet provided supplementary data on patients' starting and stopping wear times, as well as their walking scale. The "walking scale" was estimated using the item 8 of the ALSFRS-R, assessing walking functionality ranging from 0 (no movement in leg) to 4 (no problems) [3] and served as the classification label for this study. All data in the spreadsheet were anonymized to protect patient privacy. Importantly, students did not have access to any demographic information about the patients.

## 2.3 Data processing

### 2.3.1 Raw data segmentation

The accelerometer data for each patient was segmented into 24-hour spans to represent daily activities. Due to the varying start and stop times of measurements for each patient, the initial step involved selecting valid time spans based on the wear times recorded in the metadata file. Next, any missing data within these spans were identified and filled with zeros, ensuring continuity and indicating periods of no movement. Subsequently, Hees, a raw-based non-wear algorithm [22], was employed to detect wear and non-wear times. Only days with at least 8 hours of wear time were considered and one continuous time span representing a day's activity. Days with less than 8 hours of data were excluded from the analysis. Figure 1 shows the different number of valid data days for each patient, illustrating the variability in the amount of data collected per patient.

To ensure fair comparison between different patients, 2 days' data from each patient were included in the analysis. Consequently, 3 patients who had fewer than 2-day data and 1 patient who did not have any days with more than 8 hours of wear time were excluded. For patients with more than 2 days, 2 days were randomly sampled. Figure 2-a graphically illustrates the process of raw data segmentation.

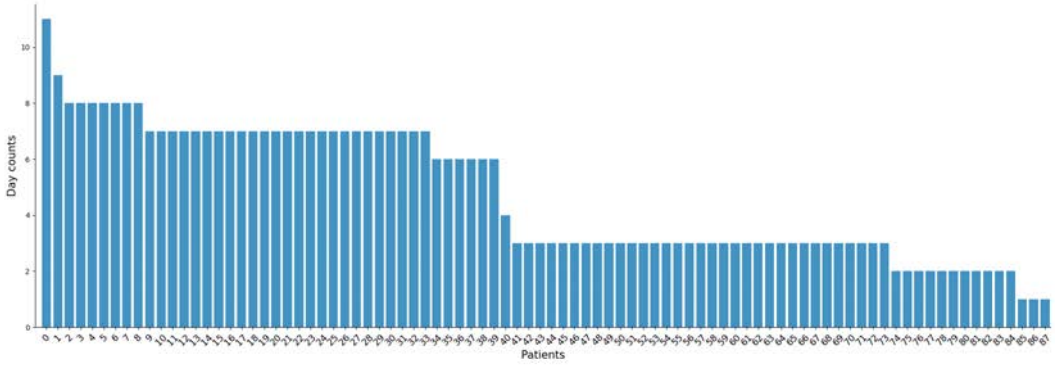



Figure 1 The amount of data collected per patient

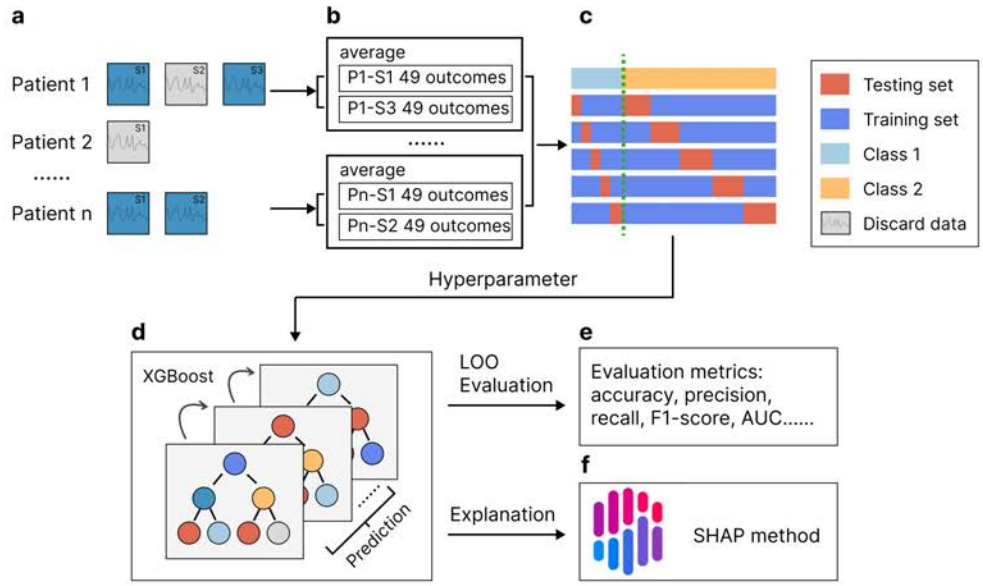



Figure 2 The workflow of preprocessing and modelling

### 2.3.2 Accelerometer-based features

The raw accelerometer data underwent summarization into activity counts to quantify physical activity. Each sample within a 1-second epoch window was assigned an activity count based on the vector magnitude of the triaxial data. This activity count was calculated as the square root of the sum of the squares of the triaxial data, i.e., $\sqrt{x^2 + y^2 + z^2}$. To remove or limit accelerations likely attributed to non-human body movements, frequency filtering (using a 7th order IIR filter) and amplitude thresholds were applied [23].

Based on extensive literature review, 49 features were selected (Table 1) [18, 19, 24, 25]. Features such as aggregate, maximum, standard deviation, mean and ratio duration time of each activity bout in a day, were computed to reflect the average daily activity of a patient. To estimate bout time in

different activity levels, we used a cutoff method [26]. Sedentary activity is defined as less than 100 counts per minute. Light activity is fewer than 760 counts per minute. Moderate-to-vigorous activity is below 2020 counts per minute, and vigorous activity is considered to be 2020 counts per minute or more.

Features regarding each activity bout count were also calculated based on activity bouts, offering detailed insights into the characteristics of each bout type, including sedentary, light, moderate-to-vigorous, and vigorous activities, across the day. This analysis helps in understanding how activity is distributed across different intensity levels and the overall contribution of each activity type to the daily activity profile. Apart from bout-based features, minute-based features such as mean, standard deviation, coefficient of variation, minimum, maximum, and percentiles (25th, 50th, and 75th) offer a granular view of activity distribution on a minute-by-minute basis throughout the day. Spectral analysis involved examining the frequency domain of the accelerometer data and extracting features representing average power within the 0.5 Hz to 5 Hz frequency range across multiple 2-second windows of accelerometer data using Fast Fourier Transform (FFT). This feature serves to quantify both the intensity and variability of physical movements captured by the accelerometer.

To compare data between different patients fairly, two days from each patient were included in the analysis. Averaging the features over these two days for each patient resulted in a single observation per patient, which can maintain sample independence.

Table 1 Feature description

| category | Feature name | Description and method |
|---|---|---|
| amount of activity | Aggregate duration time | The accumulated time of each activity type (sedentary, light, moderate-to-vigorous, and vigorous) bouts over a single day. |
| | Max duration time | The duration of the longest bout for each activity type (sedentary, light, moderate-to-vigorous, and vigorous) The accumulated time of each activity type (sedentary, light, moderate-to-vigorous, and vigorous) bouts over a single day. |
| | Standard deviation (sd) duration time | The standard deviation of the time spent in consecutive bouts for each activity type (sedentary, light, moderate-to-vigorous, and vigorous) The accumulated time of each activity type (sedentary, light, moderate-to-vigorous, and vigorous) bouts over a single day. |
| | Mean duration time | The average duration of consecutive bouts for each activity type (sedentary, light, moderate-to-vigorous, and vigorous) The accumulated time of each activity type (sedentary, light, moderate-to-vigorous, and vigorous) bouts over a single day. |
| | Ratio duration time | The ratio of the accumulated time of each activity type (sedentary, light, moderate-to-vigorous, and vigorous) bouts to the length of one segment. |
| Activity intensity | Aggregate count | The accumulated count of individual bouts for each activity type (sedentary, light, moderate-to-vigorous, and vigorous) recorded over a single day. |
| | Max count | The highest count recorded for a single bout of each activity type (sedentary, light, moderate-to-vigorous, and vigorous) within a specified time segment. |
| | Standard deviation (sd) count | The standard deviation of the counts for each bout of each activity type (sedentary, light, moderate-to-vigorous, and vigorous) recorded over a single day. |
| | Mean count | The average value of the counts for each bout of activity type (sedentary, light, moderate-to-vigorous, and vigorous) recorded over a single day. |

| | |
|---|---|
| ratio count | The ratio of the accumulated counts of each type of activity bout (sedentary, light, moderate-to-vigorous, and vigorous) to the total number of counts recorded within one segment. |
| Minute-based mean count | The average of activity counts recorded within each 1-minute window throughout a day. |
| Minute-based standard deviation (sd) count | The standard deviation of the activity counts recorded within a 1-minute window throughout a day. |
| Minute-based coefficient of variation (cv) count | The coefficient of variation of the activity counts recorded within a 1-minute window throughout a day. |
| Minute-based max count | The maximum of activity counts recorded within a 1-minute window throughout a day. |
| Minute-based min count | The minimum of activity counts recorded within a 1-minute window throughout a day. |
| Minute-based p25 count | The 25th percentile of activity counts recorded within a 1-minute window throughout a day. |
| Minute-based p50 count | The 50th percentile of activity counts recorded within a 1-minute window throughout a day. |
| Minute-based p75 count | The 75th percentile of activity counts recorded within a 1-minute window throughout a day. |
| Spectral | Cumulative power in the 0.5-5 Hz frequency band |

# 3. Method

## 3.1 XGboost classifier

XGBoost short for extreme gradient boosting, is a powerful algorithm for data classification and prediction [27]. It has gained popularity for its outstanding performance in various machine learning challenges and scientific projects [28]. It offers several advantageous properties: (i) It incorporates feature selection during training by assigning weights to individual features based on their importance in predicting the target variable. (ii) It is a flexible and robust non-parametric model capable of handling a variety of data types, including numerical and categorical features, making it suitable for diverse data distributions. (iii) As an ensemble learning method, it is more robust to noise compared to a single decision tree. (iv) It implements regularization techniques such as L1 and L2 regularization to prevent overfitting and improve generalization performance.

The input of the model is <S, L>, where S represents the physical activity feature vectors of patient i and L represents the corresponding label. Each physical activity feature vector, Si, consists of k elements, where k is 49 features in this case. The labels, L, denote whether a patient's walking ability is dichotomized as "normal walking" or "problem walking" based on their ALSFRS-R scale, where a label of 1 indicates normal walking, and 0 denotes problem walking.

Before inputting the features into the model, they are normalized using z-score normalization. Due to the imbalance in feature measures (20 normal walking subjects and 65 subjects with problem walking), an adaptive synthetic oversampling approach (ADASYN) is applied to the training set to create a balanced dataset [29].

The dataset is initially divided using StratifiedKFold into 5 folds, each containing 25 samples. Within these folds, there are approximately 6 samples classified as normal walking and 19 as problem walking. During cross-validation, one fold serves as the test set while the remaining four folds are combined to form the training set.

Following this partitioning, RandomizedSearchCV is employed to explore a range of hyperparameter values for XGBoost. This process involves randomly sampling combinations of hyperparameters and evaluating each combination using 5-fold cross-validation. The goal is to identify the optimal hyperparameter settings that maximize the model's performance, specifically aiming for high accuracy across all classes in a balanced manner.

The approach incorporates Leave-One-Out (LOO) cross-validation to assess the final model's performance. In each iteration of LOO cross-validation, one sample is designated as the test set, while the remaining 84 samples are used to train the model. Within this training phase, ADASYN is applied to balance the dataset by generating synthetic samples for the minority class, enhancing the model's ability to learn from imbalanced data. Following training, predictions are generated using the model on the test sample, and these predicted values are compared against the actual labels to compute evaluation metrics.

## 3.2 Evaluation metrics

Accuracy, sensitivity, specificity, precision, F1-score, and AUC were calculated to evaluate the performance of the classification. In this study, normal walking was considered as the positive case and problem walking was the negative case. Correct predictions of normal walking and problem walking patients are called true positives (TP) and true negatives (TN), respectively. Incorrect classifications of problem walking patients as normal walking or of normal walking patients as problem walking, are called false positives (FP) and false negatives (FN) respectively.

Accuracy was the proportion of all the correct classification results.

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad (1)$$

Sensitivity represents the proportion of positive cases that are correctly assigned (true positive rate).

$$sensitivity = \frac{TP}{TP + FN} \quad (2)$$

Specificity refers to the rate of correctly predicted negative cases in all negative cases (true negative rate).

$$specificity = \frac{TN}{TN + FP} \tag{3}$$

Precision is the ratio of the correctly predicted positive cases in all predicted positive cases.

$$precision = \frac{TP}{TP + FP} \tag{4}$$

F1-score is the harmonic mean (average) of the precision and sensitivity.

$$F1 = 2\,\frac{precision \times sensitivity}{precision + sensitivity} \tag{5}$$

The receiver operating characteristic (ROC) curve was calculated to evaluate the performance of XGboost. The Y-axis of this curve represents the true positive rate (sensitivity) and the X-axis means false positive rate (specificity). The overall classification performance of XGboost was evaluated by the area under the ROC curve (AUC). AUC ranges in value from 0 to 1, where 0.5 indicates random guessing, and 1 indicates perfect performance.

## 3.3 Feature importance

To evaluate the influence of each feature on the classifier's prediction, the SHapley Additive exPlanations (SHAP) approach will be employed. This method, derived from coalitional game theory, provides insights into how to fairly distribute the "payout" among the features [30]. By computing Shapley values, features in non-linear models like XGboost can be ranked based on their importance. These values quantify feature importance and reveal the direction of their impact on the model prediction. SHAP fosters understanding by providing precise explanations for each dataset observation, reinforcing confidence when key variables align with human domain knowledge and reasonable expectations.

# 4. Results

85 patients with 49 accelerometer features were selected as the input for the classifiers. Figure 3 displays the confusion matrix, from which the accuracy, sensitivity, specificity, precision and the F1-score were calculated to evaluate the model's performance metrics.

Using XGBoost, the two groups were classified with a high accuracy of 76.47%. The specificity score reveals that 76.92% of the samples are correctly classified as problem walking patients, while 23.08% are identified as normal walking patients. The sensitivity score shows that 75% of the samples are correctly assigned to the normal walking group, with 25% misclassified as problem walking. These scores indicate that the model is effective at identifying true positive and true negative cases but also makes some false positive and false negative errors. The F1-score, which considers both false positives and false negatives by computing their harmonic mean, was 60%. This suggests that the model has a moderate ability to correctly identify positive cases and balance false positives and false negatives, although the precision is relatively low. In figure 4, the AUC indicates that the model has a 75% chance of correctly distinguishing between problem walking and normal walking patients.


Figure 3 Confusion matrix

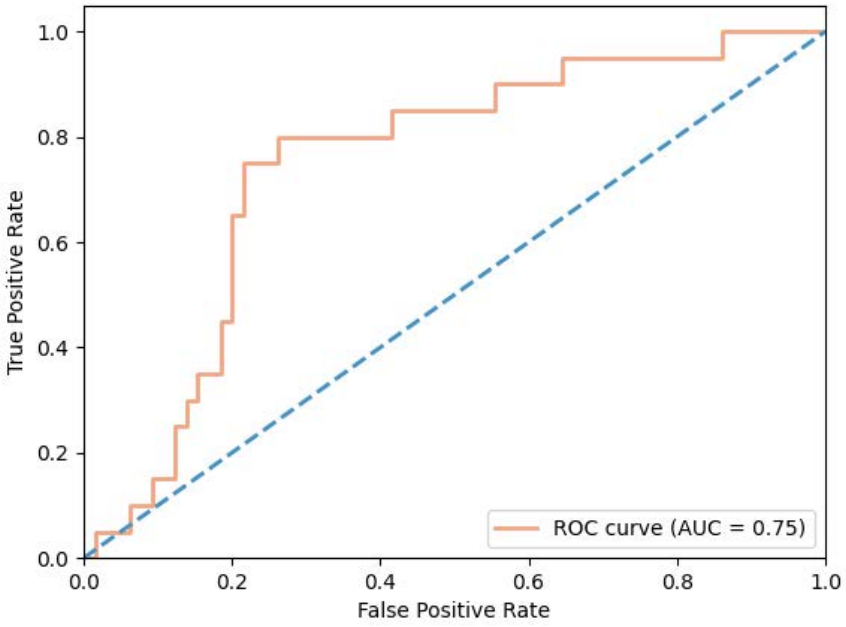



Figure 4 The receiver operating characteristic (ROC) curve is shown in orange. The area under the curve (AUC) represents the overall performance of the classifier. The blue dashed line represents a random classifier, where the proportion of correctly classified normal walking samples is equal to the proportion of incorrectly classified problem walking samples.

In Figure 5, a density scatter plot of SHAP values for each feature highlights their impact on the model output. Features are sorted by the sum of the SHAP value magnitudes across all samples. The plot reveals that the moderate-to-vigorous standard deviation (sd) count is the most impactful feature for the model's prediction. Small values of the moderate-to-vigorous sd count (marked with blue dots) contribute to negative model predictions (more likely to be classified as problem walking), whereas higher values in this feature (marked with red dots) contribute to positive model predictions (more likely to be classified as normal walking). This indicates that patients with changes in walking ability tend to have small variations in moderate-to-vigorous activity counts. Furthermore, the pattern in the mean duration time of the light activity reveals that higher values in this feature (indicating more time spent in light activity) lead to a higher likelihood of patients being classified as having problem walking.

The distribution of SHAP values for the moderate-to-vigorous sd count is also informative. Many low values have very high negative SHAP values, indicating that small variations in moderate-to-vigorous activity strongly contribute to negative predictions, suggesting a higher likelihood of

problem walking classification. In contrast, many high values are centered around a small range of low positive SHAP values, indicating that normal walking individuals with variations in moderate-to-vigorous activity have a less pronounced impact on predictions.

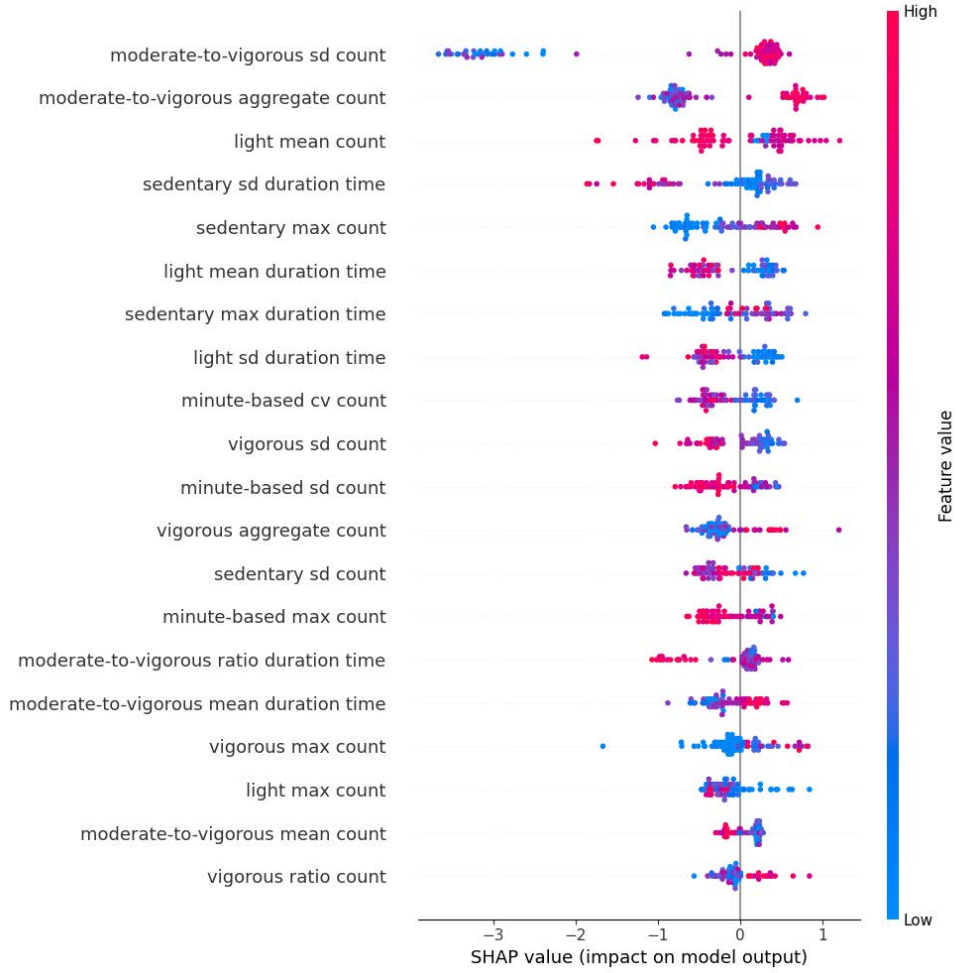



Figure 5 SHAP summary plot

Appendix 1 further illustrates the impact of each feature on the model's predictions and identifies critical threshold values distinguishing problem walking subjects from normal walking subjects. SHAP values above the y=0 lead to predictions of normal walking. For moderate-to-vigorous activity, all features except for ratio duration time and mean count show that higher values increase the likelihood of predicting normal walking. Similarly, for vigorous activity, all features except for sd count indicate that higher values increase the probability of predicting normal walking. In contrast, for sedentary activity, higher values in all features except for max count and max duration time increase the likelihood of predicting problem walking. Also, for light activity and minute-based features, higher values in these features decrease the likelihood of predicting normal walking, thereby increasing the probability of predicting problem walking.

# 5. Discussion and Conclusion

This research highlights the utility of objective physical performance monitoring using accelerometry in the ALS population. By understanding the importance of these features, we can provide an objective method to quantify disease progression and comprehend the interactions among these features. This approach has the potential to significantly enhance our ability to monitor and manage ALS.

Based on bout-based, minute-based, and spectral outcomes, using XGBoost, the two groups were classified with relatively high accuracy, sensitivity, and specificity. The precision is 50%, indicating that the model produces 50% false positive predictions and 50% true positive predictions. However, the F1-score of 60% suggests that the model has a moderate ability to correctly identify positive cases and balance false positives and false negatives. The AUC of 75% is consistent with the AUC achieved by the multi-layer perceptron models in Vieira et al.'s approach to predicting ALSFRS-R scores for walking function. This result confirms that digital biomarkers extracted from accelerometer data can effectively distinguish ALS patients' walking abilities at different scales using XGBoost. Additionally, as item 8 (walking) is a gross motor function, this finding reveals that accelerometer data can capture the relationship between physical activity and gross motor functions. This aligns with previous research by Gupta et al. and Vieira et al., further validating the use of accelerometers in assessing gross motor functions in ALS patients.

After verifying the feasibility of the model, we conducted a more detailed analysis of how each feature distinguished between categories, which is less commonly addressed in other methods. It was observed that higher values in moderate-to-vigorous and vigorous activity are more likely to predict patients without problem walking compared to those with problem walking. Interestingly, features such as moderate-to-vigorous mean count, moderate-to-vigorous ratio duration time, and vigorous sd count show higher values for patients with problem walking. This means normal

walking patients may often have a large number of short bouts of activity, which can result in the total activity count being spread across many bouts, leading to a lower mean count. In contrast, problem walking patients may have fewer but more intense bouts of activity, resulting in higher mean counts. For instance, patients with no ambulatory capacity, not even with assistance, still exhibit some periods of moderate-to-vigorous and vigorous activity every day. This implies that even those with severe walking difficulties engage in some level of intense activity. However, whether this movement is autonomous and related to walking remains to be verified by collecting more data.

Conversely, problem walking patients exhibit high values in light activity and minute-based features. On the one hand, higher light mean count and light mean duration time indicate that problem walking patients spend more time engaged in light activity. These light activities may include tasks such as dressing, hygiene, turning in bed, or adjusting bed clothes, which can be considered fine-motor functions. Thus, bout-based features applied in the machine learning model are able to effectively predict not only the functional assessment of skills that require strength but also the fine motor functions at different walking stages of ALS patients. This comprehensive approach highlights the model's ability to capture a wide range of motor functions, providing valuable insights into the patient's overall motor abilities.

On the other hand, higher values in standard deviation, coefficient of variation, and max count suggest more variability in their activity levels. This variability is also evident in sedentary behavior. Patients with problem walking do not engage in long and highly active rest periods (indicated by small max counts and max duration times), and their resting patterns are highly variable (indicated by large standard deviations in duration time and count). This variability suggests that problem walking patients have many rest periods of differing lengths and activity levels. This observation is consistent with the label’s partition, where problem walking functionality ranges widely from 0 (no movement in the leg) to 3 (minor ambulatory difficulties).

The variability in sedentary, light and short period activities among

patients with a problem walking scale from 0 to 3 means the model can detect these differences effectively. In other words, the model is able to identify subtle variations in motor activity that correlate with the severity of walking difficulties. Clinically, this variability can be crucial for tailoring individualized treatment plans and monitoring disease progression.

Several limitations of this study need to be addressed. Firstly, the samples with abnormal walking patterns include three different severity levels. This indicates a wide variability among patients, especially those with no ambulatory capacity. Tracking disease progression in this subgroup of patients could potentially be enhanced by measuring the severity of other functionalities, such as using voice samples. Secondly, although this study provides important insights into the potential of accelerometer-derived features, the limited sample size prevents a thorough understanding due to the heterogeneity in patients. The variation in individual patient characteristics and disease progression stages may not be fully captured in a small sample, leading to less generalizable results. Future research with larger, more diverse cohorts is necessary to validate these findings and enhance the robustness of the model, ensuring it can accommodate the wide range of motor function variations present in the ALS population. Lastly, it should be noted that this study established sedentary, light, moderate-to-vigorous, and vigorous physical activity thresholds based on Matthews's theory [26]. Future research could explore alternative methods for threshold selection to optimize feature extraction and improve model performance.

In conclusion, using machine learning methods, features derived from accelerometry can to some extent differentiate the walking function of ALS patients. We observed that patients with normal walking function exhibit higher levels of moderate-to-vigorous and vigorous activities, whereas those with impaired walking spend more time in light physical activities. However, to optimize the model's performance and ensure the validity of these markers in practical applications, further research is needed. This includes exploring alternative thresholds, utilizing additional datasets, addressing dataset size limitations, and refining labeling criteria to comprehensively assess functional degradation.

# References

1. Oskarsson, B., Gendron, T. F., & Staff, N. P. (2018). Amyotrophic Lateral Sclerosis: An Update for 2018. Mayo Clinic proceedings, 93(11), 1617–1628. https://doi.org/10.1016/j.mayocp.2018.04.007

2. Orsini, M., Oliveira, A. B., Nascimento, O. J., Reis, C. H., Leite, M. A., de Souza, J. A., Pupe, C., de Souza, O. G., Bastos, V. H., de Freitas, M. R., Teixeira, S., Bruno, C., Davidovich, E., & Smidt, B. (2015). Amyotrophic Lateral Sclerosis: New Perpectives and Update. Neurology international, 7(2), 5885. https://doi.org/10.4081/ni.2015.5885

3. Cedarbaum, J. M., Stambler, N., Malta, E., Fuller, C., Hilt, D., Thurmond, B., & Nakanishi, A. (1999). The ALSFRS-R: a revised ALS functional rating scale that incorporates assessments of respiratory function. Journal of the Neurological Sciences, 169(1–2), 13–21. https://doi.org/10.1016/s0022-510x(99)00210-5

4. Maier, A., Boentert, M., Reilich, P., Witzel, S., Petri, S., Großkreutz, J., Metelmann, M., Lingor, P., Cordts, I., Dorst, J., Zeller, D., Günther, R., Hagenacker, T., Grehl, T., Spittel, S., Schuster, J., Ludolph, A., Meyer, T., & MND-NET consensus group (2022). ALSFRS-R-SE: an adapted, annotated, and self-explanatory version of the revised amyotrophic lateral sclerosis functional rating scale. Neurological research and practice, 4(1), 60. https://doi.org/10.1186/s42466-022-00224-6

5. Bedlack, R. S., Vaughan, T., Wicks, P., Heywood, J., Sinani, E., Selsov, R., Macklin, E. A., Schoenfeld, D., Cudkowicz, M., & Sherman, A. (2016). How common are ALS plateaus and reversals?. Neurology, 86(9), 808–812. https://doi.org/10.1212/WNL.0000000000002251

6. Van Eijk, R. P. A., de Jongh, A. D., Nikolakopoulos, S., McDermott, C. J., Eijkemans, M. J. C., Roes, K. C. B., & van den Berg, L. H. (2021). An old friend who has overstayed their welcome: the ALSFRS-R total score as primary endpoint for ALS clinical trials. Amyotrophic lateral sclerosis & frontotemporal degeneration, 22(3-4), 300–307. https://doi.org/10.1080/21678421.2021.1879865

7. Van Eijk, R. P. A., Bakers, J. N. E., Bunte, T. M., De Fockert, A. J., Eijkemans, M. J., & Van Den Berg, L. H. (2019). Accelerometry for remote monitoring of physical activity in amyotrophic lateral sclerosis: a longitudinal cohort study. Journal of Neurology, 266(10), 2387–2395. https://doi.org/10.1007/s00415-019-09427-5

8. Strączkiewicz, M., Karas, M., Johnson, S. A., Burke, K., Scheier, Z., Royse, T. B., Calcagno, N., Clark, A., Iyer, A., Berry, J., & Onnela, J. (2024). Upper limb movements as digital biomarkers in people with ALS. EBioMedicine, 101, 105036. https://doi.org/10.1016/j.ebiom.2024.105036

9. Van Unnik JWJ, Meyjes M, Janse van Mantgem MR, Van den Berg LH, Van Eijk RPA. (2024). Remote monitoring of amyotrophic lateral sclerosis using wearable sensors detects differences in disease progression and survival: a prospective cohort study. EBioMedicine, 5:103:105104. doi: 10.1016/j.ebiom.2024.105104

10. Balendra, R., Jones, A., Jivraj, N., Knights, C., Ellis, C., Burman, R., Turner, M. R., Leigh, P. N., Shaw, C. E., & Al-Chalabi, A. (2014). Estimating clinical stage of amyotrophic lateral sclerosis from the ALS Functional Rating Scale. Amyotrophic Lateral Sclerosis and Frontotemporal Degeneration/Amyotrophic Lateral Sclerosis & Frontotemporal Degeneration, 15(3–4), 279–284. https://doi.org/10.3109/21678421.2014.897357

11. Maier, A., Boentert, M., Reilich, P., Witzel, S., Petri, S., Großkreutz, J., Metelmann, M., Lingor, P., Cordts, I., Dorst, J., Zeller, D., Günther, R., Hagenacker, T., Grehl, T., Spittel, S., Schuster, J., Ludolph, A. C., & Meyer, T. (2022). ALSFRS-R-SE: an adapted, annotated, and self-explanatory version of the revised amyotrophic lateral sclerosis functional rating scale. Neurological Research and Practice, 4(1). https://doi.org/10.1186/s42466-022-00224-6

12. Prell, T., Gaur, N., Steinbach, R., Witte, O. W., & Großkreutz, J. (2020). Modelling disease course in amyotrophic lateral Sclerosis: pseudo-longitudinal insights from cross-sectional health-related quality of life data. Health and Quality of Life Outcomes, 18(1). https://doi.org/10.1186/s12955-020-01372-6

13. Gordon, P. H., Cheung, Y. K., & Kimura, F. (2006). Progression rate of ALSFRS-R at time of diagnosis predicts survival time in ALS. Neurology, 67(7), 1314–1315. https://doi.org/10.1212/01.wnl.0000243812.25517.87

14. Yang, C., & Hsu, Y. (2010). A review of Accelerometry-Based Wearable Motion Detectors for Physical activity monitoring. Sensors, 10(8), 7772–7788. https://doi.org/10.3390/s100807772

15. Mathie, M., Coster, A. C., Lovell, N. H., & Celler, B. G. (2004b). Accelerometry: providing an integrated, practical method for long-term, ambulatory monitoring of human movement. Physiological Measurement, 25(2), R1–R20. https://doi.org/10.1088/0967-3334/25/2/r01

16. Mei, J., Desrosiers, C., & Frasnelli, J. (2021). Machine Learning for the Diagnosis of Parkinson's Disease: A Review of Literature. Frontiers in aging neuroscience, 13, 633752. https://doi.org/10.3389/fnagi.2021.633752

17. Zheng, X., Reneman, M. F., Preuper, R. H. S., Otten, E., & Lamoth, C. J. (2023). Relationship between physical activity and central sensitization in chronic low back pain: Insights from machine learning. Computer methods and programs in biomedicine, 232, 107432. https://doi.org/10.1016/j.cmpb.2023.107432

18. Iaboni, A., Spasojevic, S., Newman, K., Schindel Martin, L., Wang, A., Ye, B., Mihailidis, A., & Khan, S. S. (2022). Wearable multimodal sensors for the detection of behavioral and psychological symptoms of dementia using personalized machine learning models. Alzheimer's & dementia (Amsterdam, Netherlands), 14(1), e12305. https://doi.org/10.1002/dad2.12305

19. Gupta, A. S., Patel, S., Premasiri, A., & Vieira, F. (2023). At-home wearables and machine learning sensitively capture disease progression in amyotrophic lateral sclerosis. Nature communications, 14(1), 5080. https://doi.org/10.1038/s41467-023-40917-3

20. Vieira, F. G., Venugopalan, S., Premasiri, A. S., McNally, M., Jansen, A., McCloskey, K., Brenner, M. P., & Perrin, S. (2022). A machine-learning based objective measure for ALS disease severity. NPJ digital medicine, 5(1), 45. https://doi.org/10.1038/s41746-022-00588-8

21. Brooks, B. R., Miller, R. G., Swash, M., & Munsat, T. L. (2000). World

Federation of Neurology Research Group on Motor Neuron Diseases. El Escorial Revisited: Revised criteria for the diagnosis of amyotrophic lateral sclerosis. Amyotrophic Lateral Sclerosis and Other Motor Neuron Disorders, 1(5), 293–299. https://doi.org/10.1080/146608200300079536

22. Syed, S., Morseth, B., Hopstock, L. A., & Horsch, A. (2020). Evaluating the performance of raw and epoch non-wear algorithms using multiple accelerometers and electrocardiogram recordings. Scientific reports, 10(1), 5866. https://doi.org/10.1038/s41598-020-62821-2

23. Neishabouri, A., Nguyen, J., Samuelsson, J., Guthrie, T., Biggs, M., Wyatt, J., Cross, D., Karas, M., Migueles, J. H., Khan, S., & Guo, C. C. (2022). Quantification of acceleration as activity counts in ActiGraph wearable. Scientific reports, 12(1), 11958. https://doi.org/10.1038/s41598-022-16003-x

24. Ellis, K., Kerr, J., Godbole, S., Staudenmayer, J., & Lanckriet, G. (2016). Hip and Wrist Accelerometer Algorithms for Free-Living Behavior Classification. Medicine and science in sports and exercise, 48(5), 933–940. https://doi.org/10.1249/MSS.0000000000000840

25. Zheng, X., Reneman, M. F., Preuper, R. H. S., Otten, E., & Lamoth, C. J. (2023). Relationship between physical activity and central sensitization in chronic low back pain: Insights from machine learning. Computer methods and programs in biomedicine, 232, 107432. https://doi.org/10.1016/j.cmpb.2023.107432

26. Matthews, C. E., Keadle, S. K., Troiano, R. P., Kahle, L., Koster, A., Brychta, R., Van Domelen, D., Caserotti, P., Chen, K. Y., Harris, T. B., & Berrigan, D. (2016). Accelerometer-measured dose-response for physical activity, sedentary time, and mortality in US adults. The American journal of clinical nutrition, 104(5), 1424–1432. https://doi.org/10.3945/ajcn.116.135129

27. Chen, T.Q. and Guestrin, C. (2016) Xgboost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, 13-17 August 2016, 785-794. https://doi.org/10.1145/2939672.2939785

28. Gertz, M., Große-Butenuth, K., Junge, W., Maassen-Francke, B.,

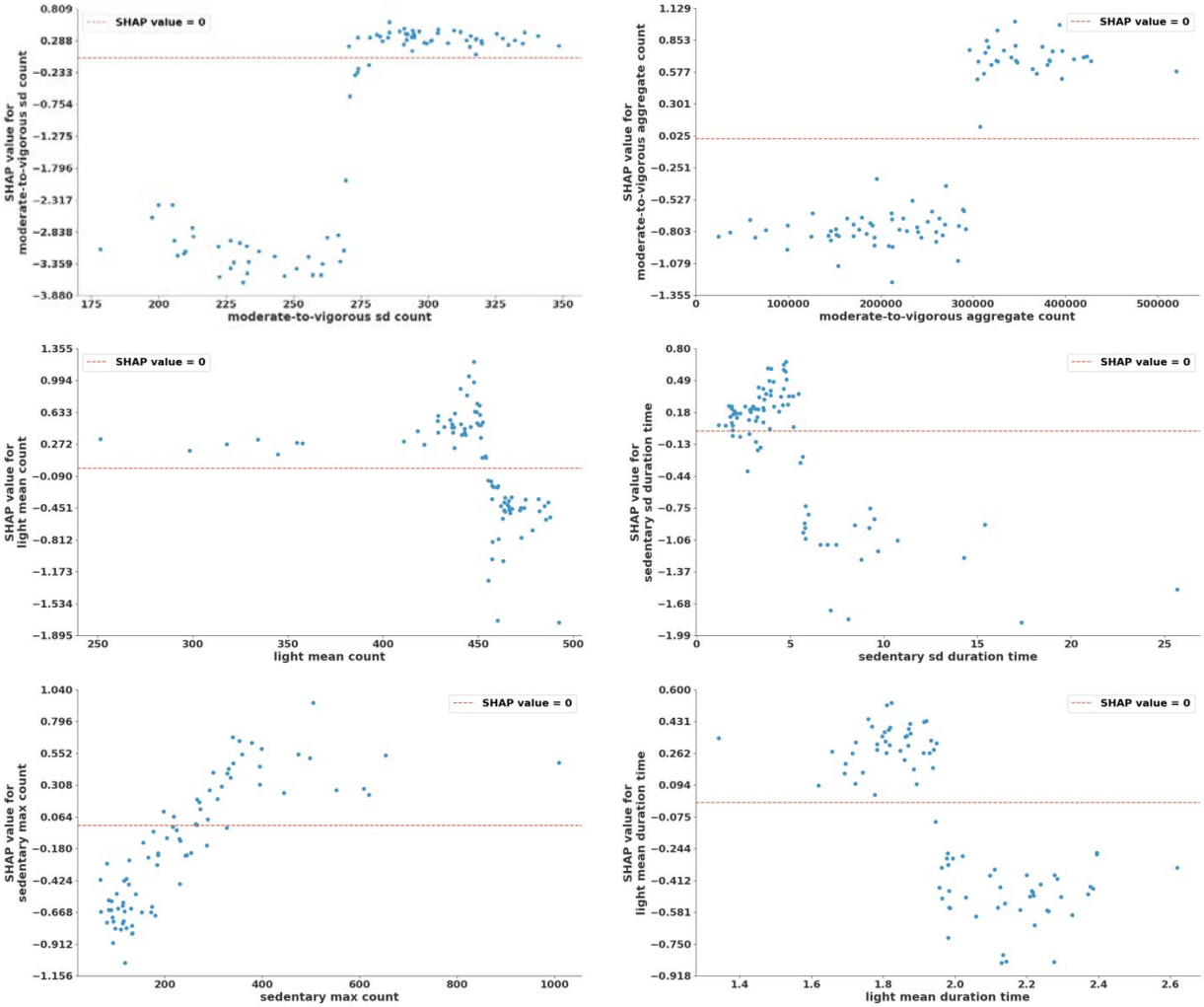
Renner, C., Sparenberg, H., & Krieter, J. (2020). Using the XGBoost algorithm to classify neck and leg activity sensor data using on-farm health recordings for locomotor-associated diseases. Computers and Electronics in Agriculture, 173, 105404. https://doi.org/10.1016/j.compag.2020.105404

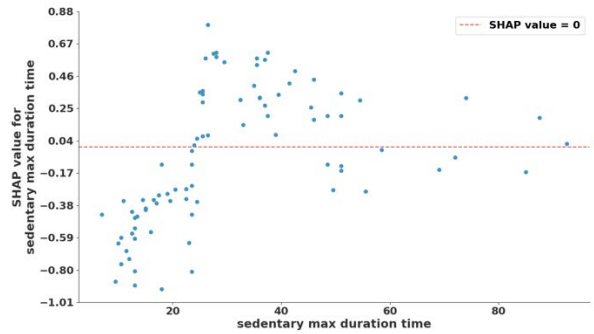
29. Haibo He, Yang Bai, Garcia, E. A., & Shutao Li. (2008). ADASYN: Adaptive synthetic sampling approach for imbalanced learning. 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), 1322-1328. https://doi.org/10.1109/IJCNN.2008.4633969

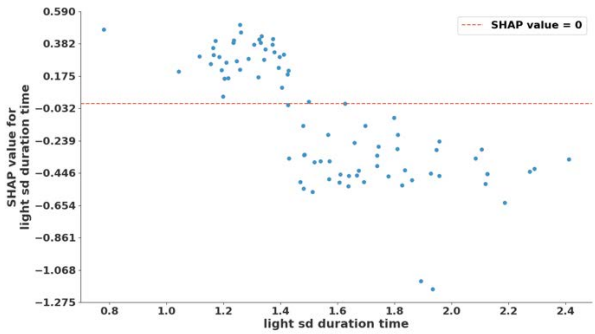
30. Lundberg, S. M., & Lee, S. (2017). A unified approach to interpreting model predictions. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1705.07874

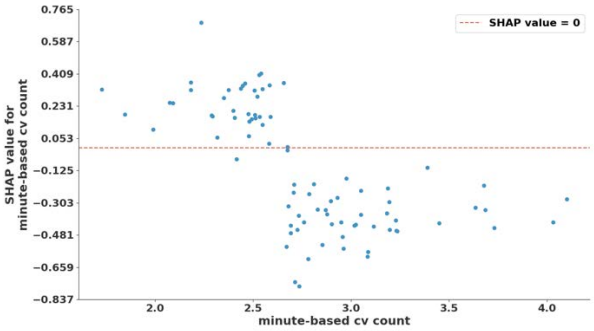
# Appendixes

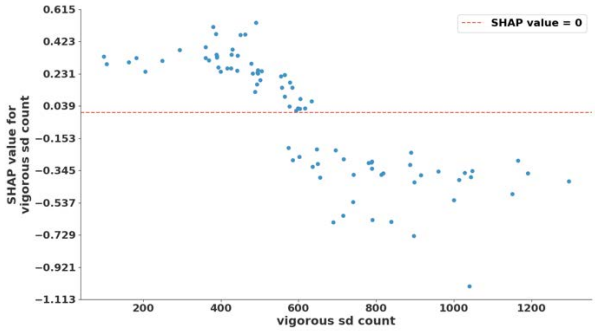
## Appendix 1 SHAP dependence plots

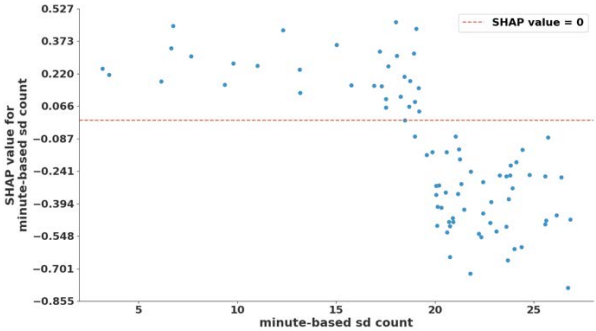


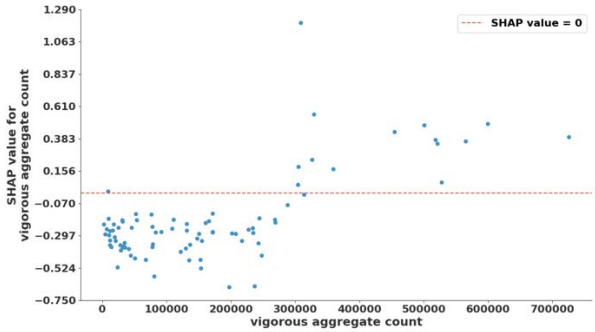

SHAP value = 0
SHAP value for sedentary max duration time
sedentary max duration time

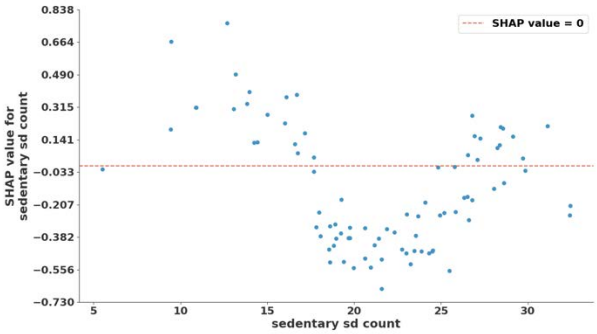

SHAP value = 0
SHAP value for light sd duration time
light sd duration time

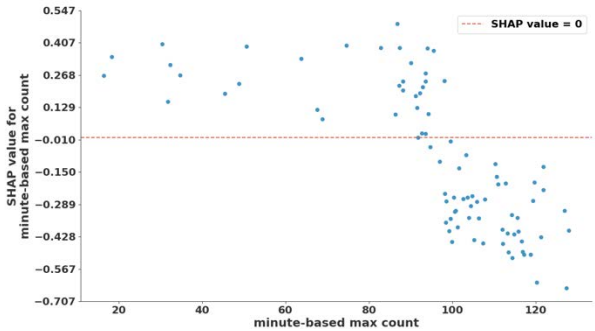

SHAP value = 0
SHAP value for minute-based cv count
minute-based cv count

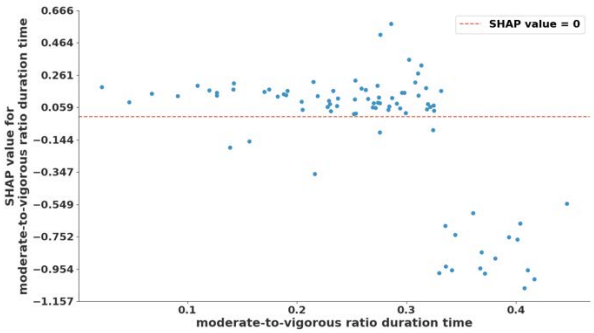

SHAP value = 0
SHAP value for vigorous sd count
vigorous sd count

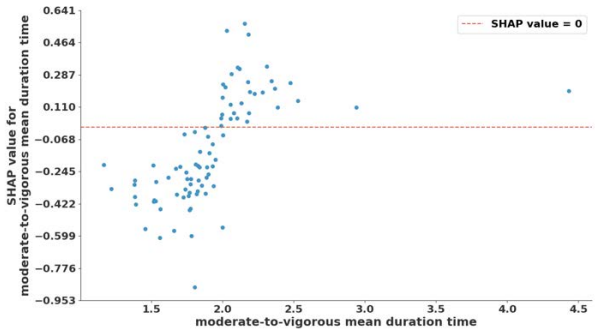

SHAP value = 0
SHAP value for minute-based sd count
minute-based sd count

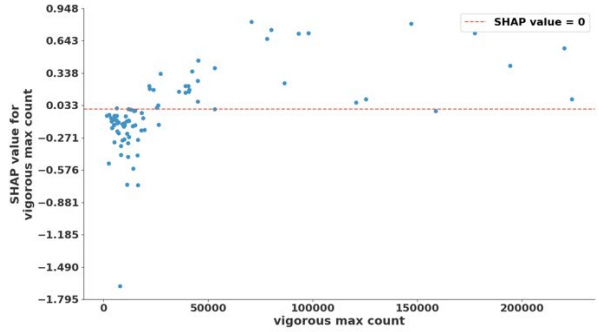

SHAP value = 0
SHAP value for vigorous aggregate count
vigorous aggregate count

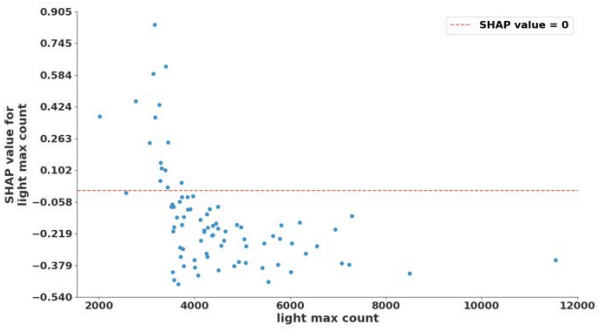

SHAP value = 0
SHAP value for sedentary sd count
sedentary sd count

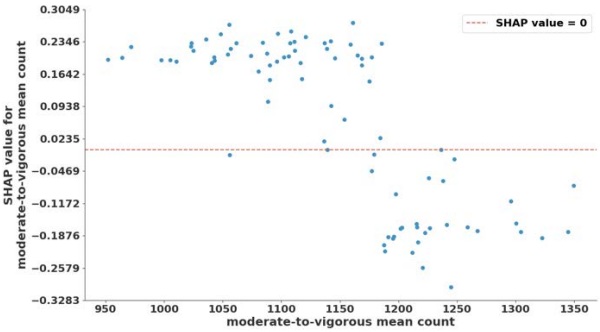

SHAP value = 0
SHAP value for minute-based max count
minute-based max count

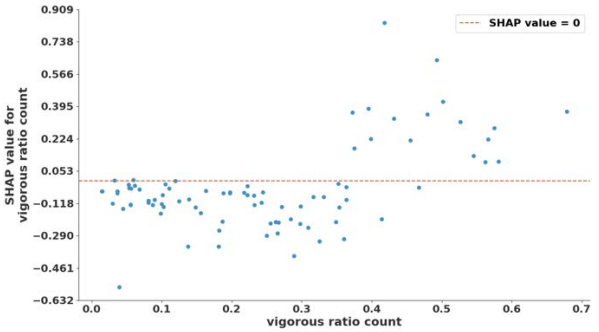

SHAP value = 0
SHAP value for moderate-to-vigorous ratio duration time
moderate-to-vigorous ratio duration time


SHAP value = 0
SHAP value for moderate-to-vigorous mean duration time
moderate-to-vigorous mean duration time


SHAP value = 0
SHAP value for vigorous max count
vigorous max count
0.948
0.643
0.338
0.033
−0.271
−0.576
−0.881
−1.185
−1.490
−1.795
0
50000
100000
150000
200000


SHAP value = 0
SHAP value for light max count
light max count
0.905
0.745
0.584
0.424
0.263
0.102
−0.058
−0.219
−0.379
−0.540
2000
4000
6000
8000
10000
12000


SHAP value = 0
SHAP value for moderate-to-vigorous mean count
moderate-to-vigorous mean count
0.3049
0.2346
0.1642
0.0938
0.0235
−0.0469
−0.1172
−0.1876
−0.2579
−0.3283
950
1000
1050
1100
1150
1200
1250
1300
1350


SHAP value = 0
SHAP value for vigorous ratio count
vigorous ratio count
0.909
0.738
0.566
0.395
0.224
0.053
−0.118
−0.290
−0.461
−0.632
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

# Appendix 2 Converting GT3X files to csv files

```
library(read.gt3x)
library(PhysicalActivity)
library(ggplot2)
library(lubridate)
library(dplyr)

# Set the directory where the GT3X files are located
gt3x_dir <- "self-define directory path"

# List all GT3X files in the directory
gt3x_files <- list.files(gt3x_dir, pattern = "\\.gt3x$", full.names = TRUE)

for (file in gt3x_files) {
  # Read the GT3X file
  data <- read.gt3x(file)
  raw_data <- as.data.frame(data)
  raw_data$time_index <- attributes(data)$time_index
  # Convert the data to CSV format
  csv_file <- sub("\\.gt3x$", ".csv", file)
  write.csv(raw_data, csv_file, row.names = FALSE)
  # Print message
  cat("Converted", file, "to CSV format:", csv_file, "\n")
}
```

# Appendix 3 Accelerometer data preprocessing

```
import pandas as pd
import os
import glob
import matplotlib.pyplot as plt
import numpy as np
import re
from hees_2013 import hees_2013_calculate_non_wear_time

def missing_process(selected_df):
    # calculate the difference between current second and previous second
    ts = [ts_elem.to_pydatetime() for ts_elem in selected_df['time']]
    ts = [t.strftime('%Y-%m-%d %H:%M:%S') for t in ts]
    ts = pd.to_datetime(ts).unique()
    ts = pd.DataFrame(ts, columns=['time'])
    ts['diff'] = ts['time'].diff().dt.total_seconds()

    # if difference is not 1, record the missing second
    missing_sec = []
    for i in range(1, len(ts)):
        if ts['diff'][i] != 1:
            missing_sec.append(pd.date_range(start=ts['time'][i - 1],
end=ts['time'][i], freq='S')[1:-1])
    missing_sec = [item for sublist in missing_sec for item in sublist]

    missing_sec_list = [value for sublist in [[elem] * 30 for elem in
missing_sec] for value in sublist]
    timestamp_microseconds = [int(sec.timestamp() * 1e6) for sec in
missing_sec_list]

    # interpolate 30 sample datetimes in one second
    initial_value = [3.3333, 3.3334, 3.3333]
    interpolation_values = []
    cumulative_sum = 0
    missing_result = []

    for i in range(0, len(timestamp_microseconds)):
        if i % 30 == 0:
            interpolation_values.append(0)
        else:
```

```
            index = (i % 3)
            interpolation_values.append(initial_value[index - 1])

    for i in range(len(timestamp_microseconds)):
        cumulative_sum = interpolation_values[i] + cumulative_sum
        if i % 30 == 0:
            cumulative_sum = 0
        next_microsecond = timestamp_microseconds[i] + cumulative_sum * 10000
        missing_result.append(next_microsecond)

    missing_result = pd.to_datetime(missing_result, unit='us')
    missing_result = pd.DataFrame({'time': missing_result})
    missing_result['X'] = 0
    missing_result['Y'] = 0
    missing_result['Z'] = 0

    complete_sec = pd.concat([selected_df[['time', 'X', 'Y', 'Z']],
missing_result], ignore_index=True)
    complete_sec = complete_sec.sort_values(by='time').reset_index(drop=True)

    return complete_sec


# Directory containing the CSV files
directory = 'self-define directory path'


# Get a list of file paths for all CSV files in the directory
csv_files = glob.glob(os.path.join(f'{directory}', "*.csv"))

# Read metadata
metadata = pd.read_excel('(self-define directory path)/metadata.xlsx')
metadata['tSTART'] = pd.to_timedelta(metadata['tSTART'])
metadata['tSTOP'] = pd.to_timedelta(metadata['tSTOP'])


# Iterate over each CSV file
for file_path in csv_files:
    # Extract the name of the CSV file
    csv_name = os.path.splitext(os.path.basename(file_path))[0]
    num = re.findall(r'\d+', csv_name)
```

```
    if len(num) > 1:
        num = num[1]
    else:
        num = num[0]
    csv_name_new = f'ID{num}'

    # select start recording time and stop recording time based on the record
in metadata
    if csv_name_new in metadata['ID'].values:
        index = metadata[metadata['ID'] == csv_name_new].index[0]
        start_date = metadata['START'][index]
        start_time = metadata['tSTART'][index]
        stop_date = metadata['STOP'][index]
        stop_time = metadata['tSTOP'][index]

    start_date_time = start_date + start_time
    stop_date_time = stop_date + stop_time



    # Read the CSV file into a DataFrame
    df = pd.read_csv(file_path)
    df['time'] = pd.to_datetime(df['time'])
    selected_df = df[(df['time'] >= start_date_time) & (df['time'] <=
stop_date_time)].reset_index(drop=True)

    # checking missing seconds and filling to continuous time series
    complete_df = missing_process(selected_df)
    complete_np = complete_df[['X', 'Y', 'Z']].values

    # checking wear and non-wear time
    non_wear_detect = hees_2013_calculate_non_wear_time(data=complete_np,
hz=30)
    complete_df['wear_or_not'] = non_wear_detect
    start_index = None
    spans = []

    # Loop through the DataFrame to find consecutive spans of 1 in column
'wear_or_not'
    for i in range(len(complete_df)):
        if complete_df.loc[i, 'wear_or_not'] == 1:
            if start_index is None:
```

```
                start_index = i
        else:
            if start_index is not None:
                spans.append((start_index, i - 1))
                start_index = None
        print(i)

    # If the last span goes till the end of the DataFrame
    if start_index is not None:
        spans.append((start_index, len(complete_df) - 1))

    # select one day's time equal or larger than 8 hours
    data_time = [sublist[1] - sublist[0] for sublist in spans]
    valid_time = 30 * 60 * 60 * 8  # 8 hours data
    data_time = np.array(data_time)
    day_index = np.where(data_time >= valid_time)[0]


    # save results based on one day
    for i in day_index:
        start = spans[i][0]
        stop = spans[i][1]
        complete_df.loc[start:stop].to_csv(f'(self-define directory path)
/{csv_name_new}_{i}.csv', index=False)

        plt.plot(complete_df['X'][start:stop])
        plt.savefig(f'(self-define directory path)/{csv_name_new}_{i}.png')
        plt.close()
```

# Appendix 4 Randomly selecting 2 day's data

```
import numpy as np
import os
import glob
import matplotlib.pyplot as plt
from collections import Counter
import random
import csv

random.seed(0)

directory = 'self-define directory path'

# Get a list of file paths for all CSV files in the directory
csv_files = glob.glob(os.path.join(self-define directory path, "*.csv"))

# Extract the name of the CSV file
csv_name = []
for file_path in csv_files:
    csv_name.append(os.path.splitext(os.path.basename(file_path))[0])

count_csv = Counter([item.split('_')[0] for item in csv_name])

# select individuals that have equal and more than 2 segments
selected_names = []
for name, count in count_csv.items():
    if count >= 2:
        selected_names.append(name)

selected_data = []
for name in selected_names:
    for item in csv_name:
        if item.split('_')[0] == name:
            selected_data.append(item)
        else:
            pass

# randomly select 2 segments
grouped_data = {}
```

```
for item in selected_data:
    prefix = item.split('_')[0]
    if prefix not in grouped_data:
        grouped_data[prefix] = []
    grouped_data[prefix].append(item)

selected_samples = []
for prefix, items in grouped_data.items():
    selected_sample = random.sample(items, 2)
    selected_samples.append(selected_sample)
flattened_list = [element for sublist in selected_samples for element in
sublist]

# save to file
csv_file = f'(self-define directory path)/(self-define name).csv'

with open(csv_file, 'w', newline='') as file:

    writer = csv.writer(file)
    writer.writerow(flattened_list)

print("CSV saves successful:", csv_file)

# plot the number segments of each individuals
count_csv = sorted(count_csv.items(), key=lambda x: x[1], reverse=True)

names = [item[0] for item in count_csv]
counts = [item[1] for item in count_csv]

plt.figure(figsize=(22, 10))
plt.bar(names, counts, color="#4393C3")
plt.xlabel('Patients')
plt.ylabel('Day counts')
plt.xticks(ticks=np.arange(len(names)), labels=np.arange(len(names)))
ax = plt.gca()
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.xlim(-1, len(names) - 0.5)
plt.savefig("(self-define name).png")
plt.show()
```

# Appendix 5 Feature extraction

```
import numpy as np
import pandas as pd
import os
import glob
import re
import extract_ac
import matplotlib.pyplot as plt
import datetime
import numpy.fft as fft
import csv

directory = 'self-define directory path'

# Get a list of file paths for all CSV files in the directory
csv_name = pd.read_csv('(self-define directory path)/(self-define name).csv')
csv_files = glob.glob(os.path.join('self-define directory path', "*.csv"))

# initial time column name
time_column = 'time'

def get_counts_csv(
    raw,
    freq: int,
    epoch: int,
    fast: bool = True,
    verbose: bool = False,
    time_column: str = None,
):

    if time_column is not None:
        ts = pd.to_datetime(raw[time_column])
        time_freq = str(epoch) + "S"
        ts = ts - pd.to_timedelta(epoch / 2, unit='s')
        ts = ts.dt.round(time_freq)
        ts = ts.unique()
        ts = pd.DataFrame(ts, columns=[time_column])

    raw_clean = raw[["X", "Y", "Z"]]
    if verbose:
        print("Converting to array", flush=True)
    raw_clean = np.array(raw_clean)
    if verbose:
        print("Getting Counts", flush=True)
    counts = extract_ac.get_counts(raw_clean, freq=freq, epoch=epoch,
fast=fast)
    del raw_clean
    counts = pd.DataFrame(counts, columns=["Axis1", "Axis2", "Axis3"])
    counts["AC"] = (
        counts["Axis1"] ** 2 + counts["Axis2"] ** 2 + counts["Axis3"] ** 2
    ) ** 0.5

    ts = ts[0: counts.shape[0]]
    if time_column is not None:
        counts = pd.concat([ts, counts], axis=1)
    return counts
```

```
# feature: %active
def active(counts):
    counts_minute = counts.groupby([counts[time_column].dt.date,
counts[time_column].dt.hour, counts[time_column].dt.minute]).sum()

    counts_minute_ts = counts_minute.index.values
    counts_minute_ts = [datetime.datetime.combine(i[0], datetime.time(i[1],
i[2])) for i in counts_minute_ts]
    counts_minute_ts = [t.strftime('%Y-%m-%d %H:%M') for t in
counts_minute_ts]
    counts_minute[time_column] = pd.to_datetime(counts_minute_ts)
    counts_minute = counts_minute.reset_index(drop=True)

    counts_minute['type'] = [1 if c < 100 else (2 if c < 760 else (3 if c <
2020 else 4)) for c in counts_minute['AC']]

    counts_minute['active'] = ['inactive' if t == 1 else 'active' for t in
counts_minute['type']]


    return counts_minute
def active_stat(count_min):
    active_size = count_min.groupby('active').size()
    active_percent = active_size['active'] / active_size.sum()
    return active_percent


# feature: met
def met(counts):
    # The coefficient of variation of the counts per 10 s
    i = 0
    ac_10_cv = []
    ac_10 = []
    while i < len(counts):
        j = i + 10
        ac_10.append(counts['AC'][i:j].sum())
        ac_10_std = counts['AC'][i:j].std()
        ac_10_mean = counts['AC'][i:j].mean()
        if ac_10_mean == 0:
            ac_10_cv.append(0)
        else:
            ac_10_cv.append((ac_10_std / ac_10_mean) * 100)
        i = j
    ac_10_cv = pd.DataFrame({'ac10': ac_10, 'cv10': ac_10_cv})

    # The met of the counts per 10 s
    met_10 = []
```

```
    for c in range(len(ac_10_cv)):
        if ac_10_cv['ac10'][c] <= 8:
            met_10.append(1)
        elif ac_10_cv['ac10'][c] > 8 and ac_10_cv['cv10'][c] <= 10:
            met_10.append(2.294275 * (np.exp(0.00084679 *
ac_10_cv['ac10'][c])))
        elif ac_10_cv['ac10'][c] > 8 and ac_10_cv['cv10'][c] > 10:
            ln_ac10 = np.log(ac_10_cv['ac10'][c])
            met_10.append(0.749395 + (0.716431 * ln_ac10) - (0.179874 *
(ln_ac10 ** 2)) + (0.033173 * (ln_ac10 ** 3)))

    met_10 = pd.DataFrame(met_10)

    # The met of the counts per day
    met_1d = met_10.mean().values[0]
    return met_1d

# feature: daily_vm
def daily_VM(counts):
    # daily VM
    counts_day_std = counts['AC'].std()
    counts_day_mean = counts['AC'].mean()
    vm = np.log(counts_day_std * counts_day_mean + 1)
    return vm

# check if the all the labels in the dictionary. If not, adding the missed
label and set value to 0
def missing_key(dict):
    if len(dict) != 4:
        missing_keys = [key for key in [1, 2, 3, 4] if key not in dict.keys()]
        for i in range(len(missing_keys)):
            dict[missing_keys[i]] = 0
    return dict

# feature:
# The accumulated/max/standard deviation/mean/ratio time of each activity type
# (sedentary, light, moderate-to-vigorous, and vigorous) bouts within one
segment.
def activity_bout_time(counts_minute):
    activity_arrays = {}
    # Iterate over the types
```

```
    for type_name, type_data in counts_minute.groupby('type'):
        # Sort the type_data by the time_column
        type_data =
type_data.sort_values(by=time_column).reset_index(drop=False)

        # Initialize lists to store continuous duration intervals
        bout_intervals = []
        current_interval = [type_data.iloc[0]['index']]

        # Iterate over the rows in the type_data
        for i in range(1, len(type_data)):
            # Check if the current row's time is continuous with the previous
row
            if (type_data.iloc[i][time_column] - type_data.iloc[i -
1][time_column]).total_seconds() <= 60:
                current_interval.append(type_data.iloc[i]['index'])
            else:
                bout_intervals.append(current_interval)
                current_interval = [type_data.iloc[i]['index']]

        # Append the last interval
        bout_intervals.append(current_interval)

        # Store the continuous intervals in the activity_arrays dictionary
        activity_arrays[type_name] = bout_intervals

    aggregate_minutes_each_time = {}
    max_minutes_each_time = {}
    mean_minutes_each_time = {}
    ratio_minutes_each_time = {}
    std_minutes_each_time = {}
    for type_name in activity_arrays:
        agg_bout_len = []
        for bout in activity_arrays[type_name]:
            bout_len = len(bout)
            agg_bout_len.append(bout_len)
        aggregate_minutes_each_time[type_name] = np.sum(agg_bout_len)
        max_minutes_each_time[type_name] = np.max(agg_bout_len)
        std_minutes_each_time[type_name] = np.std(agg_bout_len)
        mean_minutes_each_time[type_name] = np.sum(agg_bout_len) /
len(agg_bout_len)
```

```
        ratio_minutes_each_time[type_name] = np.sum(agg_bout_len) /
len(counts_minute)

    aggregate_minutes_each_time = missing_key(aggregate_minutes_each_time)
    max_minutes_each_time = missing_key(max_minutes_each_time)
    std_minutes_each_time = missing_key(std_minutes_each_time)
    mean_minutes_each_time = missing_key(mean_minutes_each_time)
    ratio_minutes_each_time = missing_key(ratio_minutes_each_time)

    return aggregate_minutes_each_time, max_minutes_each_time,
std_minutes_each_time, mean_minutes_each_time, ratio_minutes_each_time

# The accumulated/max/standard deviation/mean/ratio count of each activity
type
# (sedentary, light, moderate-to-vigorous, and vigorous) bouts within one
segment.
def activity_bout_count(counts_minute):
    activity_arrays = {}
    # Iterate over the types
    for type_name, type_data in counts_minute.groupby('type'):
        # Sort the type_data by the time_column
        type_data =
type_data.sort_values(by=time_column).reset_index(drop=False)

        # Initialize lists to store continuous duration intervals
        bout_intervals = []
        current_interval = [type_data.iloc[0]['AC']]

        # Check if the current row's time is continuous with the previous row
        # Calculate each bout counts
        for i in range(1, len(type_data)):
            if (type_data.iloc[i][time_column] - type_data.iloc[i -
1][time_column]).total_seconds() <= 60:
                current_interval.append(type_data.iloc[i]['AC'])
            else:
                bout_intervals.append(current_interval)
                current_interval = [type_data.iloc[i]['AC']]

        # Append the last interval
        bout_intervals.append(current_interval)
```

```
        # Store the continuous intervals in the activity_arrays dictionary
        activity_arrays[type_name] = bout_intervals

    aggregate_minutes_each_activity = {}
    max_minutes_each_activity = {}
    mean_minutes_each_activity = {}
    ratio_minutes_each_activity = {}
    std_minutes_each_activity = {}
    for type_name in activity_arrays:
        agg_bout_len = []
        mean_bout_len = []
        for bout in activity_arrays[type_name]:
            agg_bout_len.append(np.sum(bout))
            mean_bout_len.append(np.sum(bout)/len(bout))
        aggregate_minutes_each_activity[type_name] = np.sum(agg_bout_len)
        max_minutes_each_activity[type_name] = max(agg_bout_len)
        mean_minutes_each_activity[type_name] = np.mean(mean_bout_len)
        std_minutes_each_activity[type_name] = np.std(mean_bout_len)
        ratio_minutes_each_activity[type_name] = np.sum(agg_bout_len) /
counts_minute['AC'].sum()

    aggregate_minutes_each_activity =
missing_key(aggregate_minutes_each_activity)
    max_minutes_each_activity = missing_key(max_minutes_each_activity)
    mean_minutes_each_activity = missing_key(mean_minutes_each_activity)
    std_minutes_each_activity = missing_key(std_minutes_each_activity)
    ratio_minutes_each_activity = missing_key(ratio_minutes_each_activity)
    return aggregate_minutes_each_activity, max_minutes_each_activity,
mean_minutes_each_activity, std_minutes_each_activity,
ratio_minutes_each_activity


# feature:
# mean, standard deviation, coefficient of variation,
# minimum, maximum, 25th, 50th and 75th percentile.
def basic_stats(counts):
    counts_minute_group = counts.groupby([counts[time_column].dt.date,
counts[time_column].dt.hour, counts[time_column].dt.minute])

    mean_counts_minutes = counts_minute_group.mean()
    std_counts_minutes = counts_minute_group.std()
```

```
    cv_counts_minutes = std_counts_minutes / mean_counts_minutes
    max_counts_minutes = counts_minute_group.max()
    min_counts_minutes = counts_minute_group.min()
    p25_counts_minutes = counts_minute_group.quantile(0.25)
    p50_counts_minutes = counts_minute_group.quantile(0.50)
    p75_counts_minutes = counts_minute_group.quantile(0.75)
    p90_counts_minutes = counts_minute_group.quantile(0.90)
    p95_counts_minutes = counts_minute_group.quantile(0.95)


    # Resample data to 5-minute intervals and calculate sum of acceleration
values
    agg_counts_minutes = counts_minute_group.sum()
    count = 0
    agg_counts_5minutes = []
    while (count < len(agg_counts_minutes)):

agg_counts_5minutes.append(agg_counts_minutes['AC'][count:count+5].max())
        count = count + 5

    max_counts_day_5min = np.mean(agg_counts_5minutes)

    mean_counts_day = mean_counts_minutes['AC'].mean()
    std_counts_day = std_counts_minutes['AC'].mean()
    cv_counts_day = cv_counts_minutes['AC'].mean()
    max_counts_day = max_counts_minutes['AC'].mean()
    min_counts_day = min_counts_minutes['AC'].mean()
    p25_counts_day = p25_counts_minutes['AC'].mean()
    p50_counts_day = p50_counts_minutes['AC'].mean()
    p75_counts_day = p75_counts_minutes['AC'].mean()
    p90_counts_day = p90_counts_minutes['AC'].mean()
    p95_counts_day = p95_counts_minutes['AC'].mean()

    return mean_counts_day, std_counts_day, cv_counts_day, max_counts_day,
min_counts_day, p25_counts_day, p50_counts_day, p75_counts_day,
p90_counts_day, p95_counts_day, max_counts_day_5min


# feature:
# Angular Features: roll, pitch, and yaw angles
# roll = tan-1(y, z), pitch = tan-1(x, z) and yaw = tan-1(y, x).
```

```
# The average (avgroll, avgpitch, avgyaw) and standard deviation (sdroll,
sdpitch, sdyaw) of these angles were computed over the window
def angular_features(raw_df):
    roll_counts_second = pd.DataFrame(np.arctan2(raw_df['Y'], raw_df['Z']))
    pitch_counts_second = pd.DataFrame(np.arctan2(raw_df['X'], raw_df['Z']))
    yaw_counts_second = pd.DataFrame(np.arctan2(raw_df['Y'], raw_df['X']))

    roll_counts_second[time_column] = pd.to_datetime(raw_df[time_column])
    pitch_counts_second[time_column] = pd.to_datetime(raw_df[time_column])
    yaw_counts_second[time_column] = pd.to_datetime(raw_df[time_column])

    roll_counts_minute =
roll_counts_second.groupby([roll_counts_second[time_column].dt.date,
roll_counts_second[time_column].dt.hour,
roll_counts_second[time_column].dt.minute])
    pitch_counts_minute =
pitch_counts_second.groupby([pitch_counts_second[time_column].dt.date,
pitch_counts_second[time_column].dt.hour,
pitch_counts_second[time_column].dt.minute])
    yaw_counts_minute =
yaw_counts_second.groupby([yaw_counts_second[time_column].dt.date,
yaw_counts_second[time_column].dt.hour,
yaw_counts_second[time_column].dt.minute])

    roll_counts_minute_mean = roll_counts_minute.mean()
    pitch_counts_minute_mean = pitch_counts_minute.mean()
    yaw_counts_minute_mean = yaw_counts_minute.mean()

    roll_counts_minute_std = roll_counts_minute.std()
    pitch_counts_minute_std = pitch_counts_minute_mean.std()
    yaw_counts_minute_std = yaw_counts_minute_mean.std()

    roll_counts_day_mean = roll_counts_minute_mean.mean().values[0]
    pitch_counts_day_mean = pitch_counts_minute_mean.mean().values[0]
    yaw_counts_day_mean = yaw_counts_minute_mean.mean().values[0]

    roll_counts_day_std = roll_counts_minute_std.mean().values[0]
    pitch_counts_day_std = pitch_counts_minute_std.mean()
    yaw_counts_day_std = yaw_counts_minute_std.mean()

    return roll_counts_day_mean, pitch_counts_day_mean, yaw_counts_day_mean,
```

```
roll_counts_day_std, pitch_counts_day_std, yaw_counts_day_std

# feature: Fast Fourier Transform (FFT)
def fft_features(raw_df):
    sample_frequency = 30  # 30 hz
    window_size = 60  # 2 seconds
    # fft_frequency = fft.fftfreq(window_size, 1 / sample_frequency)
    half_hz = 1
    five_hz = 10

    amp = np.sqrt(raw_df['X']**2 + raw_df['Y']**2 + raw_df['Z']**2)
    power_spectral_list = []
    i = 0
    while i < len(amp)-60:
        j = i + 60
        fft_results = fft.fft(amp[i:j])[0:int(window_size/2)]
        power = np.abs(fft_results)**2
        power[0] = 0
        max_fre = fft.fftfreq(window_size, 1 /
sample_frequency)[np.argmax(power)]
        accumulated_power = np.sum(power[half_hz:five_hz])
        power_spectral_list.append(accumulated_power)
        i = j
    return np.mean(power_spectral_list)

# Iterate over each CSV file to calculate features
final_feature_table = []
c = 0
for file_path in csv_files:
    # Extract the name of the CSV file
    csv_name = os.path.splitext(os.path.basename(file_path[0]))[0]

    # Read the CSV file into a DataFrame
    df = pd.read_csv(file_path[0])

    # get counts per second
    counts = get_counts_csv(raw=df, freq=30, epoch=1, verbose=False,
time_column=time_column)

    # feature: %active
    counts_minute = active(counts)
```

```
    percent_active_day = active_stat(counts_minute)

    # feature: MET
    met_day = met(counts)

    # feature: daily vm
    daily_vm = daily_VM(counts)

    # feature: daily A1
    daily_A1 = np.log(counts['Axis3'].std() + 1)

    # feature: VMI
    vmi = np.log(counts['Axis3'] + 1).std()

    # feature: agg, max, mean, std, ratio of bout time
    aggregate_minutes_each_time, max_minutes_each_time, std_minutes_each_time,
mean_minutes_each_time, ratio_minutes_each_time =
activity_bout_time(counts_minute)

    # feature: agg, max, mean, std, ratio of bout counts
    aggregate_minutes_each_activity, max_minutes_each_activity,
mean_minutes_each_activity, std_minutes_each_activity,
ratio_minutes_each_activity = activity_bout_count(counts_minute)

    # feature: basic stats
    mean_counts_day, std_counts_day, cv_counts_day, max_counts_day,
min_counts_day, p25_counts_day, p50_counts_day, p75_counts_day,
p90_counts_day, p95_counts_day, max_counts_day_5min = basic_stats(counts)

    # feature: angular
    roll_counts_day_mean, pitch_counts_day_mean, yaw_counts_day_mean,
roll_counts_day_std, pitch_counts_day_std, yaw_counts_day_std =
angular_features(df)

    # feature: Fast Fourier Transform (FFT)
    fft_feature_day = fft_features(df)

    # final feature table
    features = [csv_name, percent_active_day, met_day, daily_vm, daily_A1,
vmi,
                aggregate_minutes_each_time[1],
```

```
aggregate_minutes_each_time[2], aggregate_minutes_each_time[3],
aggregate_minutes_each_time[4],
                max_minutes_each_time[1], max_minutes_each_time[2],
max_minutes_each_time[3], max_minutes_each_time[4],
                std_minutes_each_time[1], std_minutes_each_time[2],
std_minutes_each_time[3], std_minutes_each_time[4],
                mean_minutes_each_time[1], mean_minutes_each_time[2],
mean_minutes_each_time[3], mean_minutes_each_time[4],
                ratio_minutes_each_time[1], ratio_minutes_each_time[2],
ratio_minutes_each_time[3], ratio_minutes_each_time[4],
                aggregate_minutes_each_activity[1],
aggregate_minutes_each_activity[2], aggregate_minutes_each_activity[3],
aggregate_minutes_each_activity[4],
                max_minutes_each_activity[1], max_minutes_each_activity[2],
max_minutes_each_activity[3], max_minutes_each_activity[4],
                std_minutes_each_activity[1], std_minutes_each_activity[2],
std_minutes_each_activity[3], std_minutes_each_activity[4],
                mean_minutes_each_activity[1], mean_minutes_each_activity[2],
mean_minutes_each_activity[3], mean_minutes_each_activity[4],
                ratio_minutes_each_activity[1],
ratio_minutes_each_activity[2], ratio_minutes_each_activity[3],
ratio_minutes_each_activity[4],
                mean_counts_day, std_counts_day, cv_counts_day,
max_counts_day, min_counts_day, p25_counts_day, p50_counts_day,
p75_counts_day, p90_counts_day, p95_counts_day, max_counts_day_5min,
                roll_counts_day_mean, pitch_counts_day_mean,
yaw_counts_day_mean, roll_counts_day_std, pitch_counts_day_std,
yaw_counts_day_std,
                fft_feature_day]
    final_feature_table.append(features)

    c += 1
    print(csv_name, c)

final_feature_table = np.array(final_feature_table)

# save to file
csv_file = '(self-define directory path)/(self-define name).csv'

with open(csv_file, 'w', newline='') as file:
    writer = csv.writer(file, delimiter=',')
```

```
    for row in final_feature_table:
        writer.writerow(row)

print("CSV saves successfully:", csv_file)
```

## Appendix 6 Averaging 2 day's data

```
import numpy as np
import pandas as pd

# read files and IDs
directory = '(self-define directory path)'
df = pd.read_csv('(self-define directory path)/(self-define name).csv',
header=None)
df = df.rename(columns={0: 'ID'})
df['ID_prefix'] = df['ID'].str.split('_').str[0]

# import metadata to get labels
metadata = pd.read_excel('(self-define directory path)/metadata.xlsx')
metadata['label'] = [0 if v == 4 else 1 for v in metadata['ITEM8']]

id_names = np.unique([name.split('_')[0] for name in df['ID']])
id_labels = pd.DataFrame(columns=['ID', 'label'])
for id in id_names:
    id_labels = id_labels.append(metadata[metadata['ID'] == id][['ID',
'label']], ignore_index=True)


# Iterate over each CSV file to calculate mean
df_avg = df.groupby('ID_prefix').mean().reset_index(drop=False)
df_avg_all = pd.merge(df_avg, metadata[['ID', 'label']], left_on='ID_prefix',
right_on='ID', how='inner')
df_avg_all = df_avg_all.drop(columns=['ID_prefix']).reset_index(drop=True)
df_avg_all.to_csv('(self-define directory path)/(self-define name).csv',
index=False)
```

## Appendix 7 Modelling process

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import LeaveOneOut, StratifiedKFold
from sklearn.metrics import accuracy_score, precision_score, f1_score,
```

```
confusion_matrix, recall_score, auc, roc_curve
from sklearn.model_selection import RandomizedSearchCV
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import randint, uniform
from xgboost import XGBClassifier
from imblearn.over_sampling import ADASYN
import shap
import seaborn as sns
import matplotlib.ticker as mticker



# balance classes
def resampling(X, y):
    sm = ADASYN(random_state=0)
    X_res, y_res = sm.fit_resample(X, y)
    return X_res, y_res

# read data
directory = '(self-define directory path)'
df = pd.read_csv('(self-define directory path)/(self-define name).csv')
df['label'] = df['label'].replace({0: 1, 1: 0})
X = df.drop(['label', 'ID', '57', '58', '59', '60', '61', '62', '55', '54',
'56',
             '1', '2', '3', '4', '5'], axis=1)
y = df['label']


# Normalization
scaler = StandardScaler()
X_sca = scaler.fit_transform(X)


# setting hyperparameters for further tuning
param_distributions = {
    'n_estimators': randint(50, 3000),
    'max_depth': randint(1, 6),
    'eta': uniform(0.01, 0.3),
    'min_child_weight': randint(1, 10),
    'reg_alpha': uniform(0, 1),
```

```
}

# Hyperparameter Tuning using RandomizedSearchCV with StratifiedKFold
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
random_search = RandomizedSearchCV(XGBClassifier(scale_pos_weight=10,
random_state=0),
                                   param_distributions,
                                   n_iter=50,
                                   cv=cv,
                                   scoring='f1_micro',
                                   random_state=0,
                                   n_jobs=-1,
                                   return_train_score=True)

# Final Model Training
x_, y_ = resampling(X_sca, y)
random_search.fit(x_, y_)
best_parameters = random_search.best_params_

clf = XGBClassifier(
    scale_pos_weight=10,
    n_estimators=best_parameters['n_estimators'],
    max_depth=best_parameters['max_depth'],
    eta=best_parameters['eta'],
    min_child_weight=best_parameters['min_child_weight'],
    reg_alpha=best_parameters['reg_alpha'],
    random_state=0
)


# Final Evaluation using LOO
loo = LeaveOneOut()
y_true = []
y_pred = []
y_pred_prob = []
shap_values = []
for train_index, test_index in loo.split(X_sca):
    X_train, X_test = X_sca[train_index], X_sca[test_index]
    y_train, y_test = y[train_index], y[test_index]

    X_train, y_train = resampling(X_train, y_train)
```

```
    y_train = np.array(y_train)
    # training model
    clf.fit(X_train, y_train)

    # predict classes
    y_pred.append(clf.predict(X_test)[0])
    y_true.append(y_test.values[0])
    # predict probabilities
    y_pred_prob.append(clf.predict_proba(X_test)[:,1])

    # Create a SHAP explainer object
    explainer = shap.TreeExplainer(clf)
    # Calculate SHAP values for a sample of data
    shap_values.append(explainer.shap_values(X_test)[0])



# Drawing the SHAP plots
shap_values = np.array(shap_values)
feature_names=['sedentary aggregate duration time', 'light aggregate duration
time', 'moderate-to-vigorous aggregate duration time', 'vigorous aggregate
duration time',
               'sedentary max duration time', 'light max duration time',
'moderate-to-vigorous max duration time', 'vigorous max duration time',
               'sedentary sd duration time', 'light sd duration time',
'moderate-to-vigorous sd duration time', 'vigorous sd duration time',
               'sedentary mean duration time', 'light mean duration time',
'moderate-to-vigorous mean duration time', 'vigorous mean duration time',
               'sedentary ratio duration time', 'light ratio duration time',
'moderate-to-vigorous ratio duration time', 'vigorous ratio duration time',
               'sedentary aggregate count', 'light aggregate count',
'moderate-to-vigorous aggregate count', 'vigorous aggregate count',
               'sedentary max count', 'light max count', 'moderate-to-vigorous
max count', 'vigorous max count',
               'sedentary sd count', 'light sd count', 'moderate-to-vigorous
sd count', 'vigorous sd count',
               'sedentary mean count', 'light mean count', 'moderate-to-
vigorous mean count', 'vigorous mean count',
               'sedentary ratio count', 'light ratio count', 'moderate-to-
vigorous ratio count', 'vigorous ratio count',
               'minute-based mean count', 'minute-based sd count', 'minute-
```

```
based cv count', 'minute-based max count', 'minute-based min count', 'minute-
based p25 count', 'minute-based p50 count', 'minute-based p75 count',
               'spectral']
X.columns = list(range(shap_values.shape[1]))
# SHAP feature importance measured as the mean absolute Shapley values
# shap_bar = shap.summary_plot(shap_values, X_sca,
#                   plot_type="bar",
#                   plot_size=[15, 15],
#                   feature_names=feature_names,
#                   max_display=20,
#                   show=False)
# # Shap summary plot
# shap_dot = shap.summary_plot(shap_values, X_sca,
#                   plot_type="dot",
#                   plot_size=[15, 15],
#                   feature_names=feature_names,
#                   max_display=20,
#                   show=False)
# plt.savefig('plot/shap_bar.png')
# plt.savefig('plot/shap_dot.png')


# SHAP dependence plots
for feature in X.columns:
    fig, ax = plt.subplots(figsize=(14, 8))
    shap_dependence = shap.dependence_plot(feature,
                          shap_values,
                          X,
                          show=False,
                          feature_names=feature_names,
                          interaction_index=None,
                          ax=ax)
    ax.set_xlabel(ax.get_xlabel(), fontsize=18, fontweight='bold')
    ax.set_ylabel(ax.get_ylabel(), fontsize=18, fontweight='bold')
    scatter = ax.collections[0]
    scatter.set_sizes([40])  # Set point size
    scatter.set_color('#4393C3')  # Set point color
    ax.axhline(y=0,
               color='#D6604D',
               linestyle='--',
               label='SHAP value = 0')
    plt.legend(loc='best', prop={'size': 16, 'weight': 'bold'})
    y_min, y_max = ax.get_ylim()
    num_ticks = 10
    yticks = np.linspace(y_min, y_max, num_ticks)
    ax.set_yticks(yticks)
    ax.tick_params(axis='x', labelsize=16)
    ax.tick_params(axis='y', labelsize=16)
    for tick in ax.get_xticklabels():
        tick.set_fontweight('bold')
    for tick in ax.get_yticklabels():
```

```
        tick.set_fontweight('bold')

    plt.savefig(f'{feature_names[feature]}.png')
    # plt.show()

# Calculating evaluation metrics
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
cm = confusion_matrix(y_true, y_pred)
tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
specificity = tn / (tn + fp)
print(f'acc:{accuracy:.4f}; precision:{precision:.4f}; recall:{recall:.4f}; specificity:{specificity:.4f};
f1:{f1:.4f}; cm: {cm}')


# plot confusion matrix
cm_counts = ['{0:0.0f}'.format(value) for value in cm.flatten()]
row_sums = cm.sum(axis=1, keepdims=True)
cm_percentage = cm / row_sums * 100

# Create labels for the heatmap annotations with percentage values
cm_percentage_new = np.array([["{:.2f}%".format(value) for value in row] for row in cm_percentage])
labels = [f'{v1}\n{v2}' for v1, v2 in zip(cm_counts, cm_percentage_new.flatten())]
labels = np.asarray(labels).reshape(2,2)
ax = sns.heatmap(cm_percentage, annot=labels, fmt='', cmap='Blues', cbar_kws={'format': '%.0f%%'})
cbar = ax.collections[0].colorbar
cbar.ax.yaxis.set_major_formatter(mticker.PercentFormatter(xmax=100))
ax.set(xlabel='Predicted label', ylabel='True label')
ax.set_xticklabels(['Problem walking', 'Normal walking'])
ax.set_yticklabels(['Problem walking', 'Normal walking'])
# plt.savefig('(self-define directory path)/(self-define name).png')

# Plot ROC curve
fpr, tpr, thresholds = roc_curve(y_true, y_pred_prob)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='#F4A582', lw=2, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='#4393C3', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
# plt.savefig('(self-define directory path)/(self-define name).png')
```