# GaitDetect: A Machine Learning Approach for Multi-Activity Gait Recognition

Student Jainil Patel
Instructor Casey Bennett

## Abstract

*In this project, I worked with wearable sensor data to train the recognition of daily activities like walking, jogging, and climbing stairs. Such an algorithm would be beneficial in the early detection of mobility impairment since it records nuanced gait patterns. I experimented with two traditional machine-learning models, Random Forest and XGBoost, and verified their performances with k-fold cross-validation. Meticulous feature engineering maintained the input set lean with still-achieved high accuracy, and SHAP plots revealed which features were significant.*

## 1 Introduction

### Problem Statement

Gait disorders affect daily life and create big health challenges for older people and those with disabilities. Finding and diagnosing walking problems is key to taking action and lowering risks like falling or getting hurt. Doctors study walking patterns using personal opinions or expensive lab tests, which makes it harder to reach more people or track issues effectively.

### Research Goal

This project aims to build a gait recognition system that works well, is easy to understand, and runs using data from wearable sensors. The study focuses on creating dependable machine learning models that can tell apart different physical activities. This will help monitor mobility in real-time and catch mobility problems in everyday healthcare environments.

### Previous Work

Researchers have investigated how machine learning techniques can help recognize physical activities using data from sensors.

1. Bhakta et al. (2020) investigated machine learning models for intent recognition in powered prostheses, highlighting the strengths in recognition accuracy but also noting extensive training data requirements and adaptability limitations [1].

2. Sun et al. (2019) utilized postural sway measurements for fall risk prediction in Multiple Sclerosis patients. Their models achieved promising results; however, performance generalization across diverse populations was limited [2].

3. Vonstad et al. (2022) applied deep learning models to estimate ground reaction forces during balance exercises, significantly improving prediction accuracy but with drawbacks in computational complexity and reduced interpretability [3].

4. Camargo et al. (2021) explored locomotion classification using sensor fusion approaches, which improved model robustness but increased complexity and cost due to dependency on multiple sensors [4].

5. Nouredanesh and Tung (2015) developed a model for detecting compensatory balance responses to lateral perturbations. Although adaptable, their methodology was limited in scope to specific directional perturbations [5].

6. Su et al. (2019) proposed CNN-based intent recognition with inertial sensors, demonstrating high accuracy. Despite effectiveness, CNN models posed challenges in computational demand, hindering real-time wearable deployment [6].

7. Banos et al. (2014) introduced the mHealthDroid framework for agile mobile health application development, emphasizing rapid prototyping and flexibility, yet lacking detailed insights into gait-specific analysis capabilities [7].

### My Approach

Unlike the heavy computation needs of neural network techniques seen in research, I used basic machine learning models like Random Forest and XGBoost. I paired these with clear feature engineering and techniques to make the results understandable. This approach focuses on getting strong accuracy running , and keeping the model easy to interpret, which is key to using it in wearable devices during real-time activities.

Plenty of researchers favor deep learning methods such as 1D Convolutional Neural Networks or Temporal Convolutional Networks for time-series activity detection. These models perform well, but they require a lot of processing power tricky adjustments to settings, and are harder to explain. Because of this. . .

Given the constraints and applied nature of my project, I went with a straightforward method involving classical training pipelines. This choice simplifies both implementation and deployment of the models. It also improves how understandable they are by using feature-based explanation tools, like SHAP.

In the next sections of this report, I describe the dataset I used, outline my detailed methods present experimental results, discuss the findings, and conclude with key takeaways and possibilities to explore in future research.

## 2 Methods

### Dataset

The MHEALTH (Mobile Health) dataset is a public resource that holds data from wearable sensors. Researchers use it to study and identify human movements. It contains data on body motion and vital signs collected from 10 people as they carried out 12 physical tasks like walking, running, climbing stairs, sitting, and lying down.

Each person wore three sensors during the activities. One sensor was placed on the chest, another on the right wrist, and the last one on the left ankle. These sensors gathered information such as:

– 3D acceleration, gyroscope, and magnetometer signals on all three axes (X, Y, Z)

– ECG signals from the chest sensor (2 leads)

The system sampled all signals at 50 Hz, which is enough to capture the motions of human activity. The sensors are set up to monitor different body parts, and data was gathered in natural and free environments.

**Dataset Summary:**

– Number of participants: 10

– Activities recorded: 12

– Sensor devices used: 3 per person

– Sampling rate: 50 Hz

– Number of files: 10 log files (one for each subject)

**Data Dimensions (from final combined dataset):**

– Rows: 1,215,755 samples (total across all subjects)

– Columns: 24 features per sample (23 sensor readings + 1 activity label)

The dataset includes labels like standing, sitting, lying, walking, climbing stairs, bending, jogging, running, jumping, and cycling.

Every row in the dataset contains synchronized readings from sensors on all devices. An activity label is also attached to each row. To process the data further, all 10 subject log files were combined into one single dataframe as the final preprocessed dataset.

## Data Preprocessing

I used step-by-step preprocessing methods on the raw MHEALTH sensor data, following the actual process outlined in the notebook.

1. **File Aggregation:** All 10 subject files were read and concatenated into one combined dataframe.

2. **Column Assignment:** The dataframe was assigned proper column names representing sensor readings from the chest, ankle, and wrist sensors across accelerometer, gyroscope, magnetometer, and ECG channels, followed by the activity label column.

3. **Missing Value Handling:** I checked and removed all rows containing null values using `dropna()` to ensure data cleanliness.

4. **Activity Mapping:** Original numeric activity labels (1 to 12) were mapped to descriptive string labels such as 'standing', 'sitting', 'lying', etc., to improve interpretability during analysis.

5. **Null Class Filtering:** I filtered out rows with the label 0 (null class) since it does not represent a valid activity.

6. **Segmentation:** The time-series data was split into windows of 2 seconds (100 samples per window) with 50% overlap. This was implemented using a rolling window technique for temporal context preservation.

7. **Window Labeling:** For each window, I assigned the most frequent activity label as the representative label for that segment.

8. **Shuffling:** The resulting segments were shuffled to ensure that model training did not rely on any temporal sequence.

9. **Normalization:** A `MinMaxScaler` was applied to scale all 23 sensor features to the range [0, 1] while preserving label information separately.

## Feature Engineering

To transform each 2-second window into a structured input for classification, I extracted the following five statistical features from each of the 23 sensor channels:

- Mean
- Standard Deviation
- Minimum
- Maximum
- Root Mean Square (RMS)

This resulted in a total of 115 features per window (23 channels $\times$ 5 features each). These features were designed to capture essential characteristics of motion such as intensity, variability, and signal magnitude across the different body sensor placements.

## Modeling

I trained and evaluated two machine learning classifiers to predict human activities from the extracted features:

1. **Random Forest Classifier:** An ensemble method that builds multiple decision trees and outputs the class mode from all trees. I used 100 trees (`n_estimators=100`) with a maximum depth of 10.

2. **XGBoost Classifier:** A gradient boosting algorithm that sequentially improves decision trees by minimizing classification errors. I used 100 boosting rounds (`n_estimators=100`), a learning rate of 0.1, and a maximum tree depth of 5.

**Training Setup:**

- I split the windowed data into 80% training and 20% testing sets using `train_test_split()`.
- Feature normalization was performed using `MinMaxScaler` before model fitting.
- Each model was trained on the same set of 115 statistical features.

**Evaluation Metrics:**

– **Accuracy** – Overall percentage of correctly predicted activity windows.

– **Classification Report** – Precision, recall, and F1-score per activity class.

– **Confusion Matrix** – Visualization of predicted vs. actual class distributions.

# 3 Results

The results appear in four parts: the performance of the overall model, metrics of classification in detail, insights from the confusion matrix, and an analysis of feature importance.

## Overall Model Performance

I tested and checked both ensemble models and noticed they performed well in making predictions. Table 1 compares training speed and accuracy for the full 115-feature set and the reduced 29-feature subset.

Table 1: Overall Performance of Ensemble Models

| Model | Features | Train Time (s) | Test Accuracy | Macro F1 |
|---|---|---|---|---|
| Random Forest (full) | 115 | 4.31 | 0.9908 | 0.9903 |
| XGBoost (full) | 115 | 3.98 | 0.9908 | 0.9898 |
| Random Forest (selected) | 29 | 2.60 | 0.9893 | 0.9890 |
| XGBoost (selected) | 29 | 1.71 | 0.9903 | 0.9894 |

Cross-validation (5-fold) yielded $0.991 \pm 0.003$ for Random Forest and $0.991 \pm 0.002$ for XGBoost, underscoring robust generalisation.

## Classification Metrics

Using the full-feature Random Forest (the most balanced model), precision and recall exceed 0.99 for 10 of 12 activities. The lowest F1 ($\approx 0.984$) is seen in crouching (L8) and cycling (L9), yet both remain above 0.97. XGBoost shows an almost identical pattern with marginally higher recall in jogging and running classes.

## Confusion-Matrix

Even with highly similar gaits, inter-class confusion is minimal:

– Walking $\leftrightarrow$ Stair-climbing: 2 errors each way ($\approx 1\%$)

– Jogging $\leftrightarrow$ Running: $\leq 3$ total mis-labels

– No other class has more than two errors; every activity maintains $\geq 97\%$ recall

### Feature-Importance Analysis

SHAP summaries from both models converge on the same biomechanically plausible drivers:

- Chest Acc X_max
- Chest Acc X_rms
- Chest Acc Z_mean
- Left-ankle Acc Y_rms / _std
- Right-forearm Acc Y/Z_mean

These features capture trunk vertical acceleration and lower-limb dynamics—key signals in gait cycles.

Reducing from 115 $\rightarrow$ 29 features cuts training time by 40–57% with less then 0.2% accuracy loss, supports that careful feature selection preserves performance while improving efficiency.

### Summary

Both models achieve over 99% accuracy. XGBoost (full) leads in macro F1, while Random Forest (full) matches it in overall accuracy but takes around 8% more time to train. The XGBoost model with selected features provides the best trade-off between speed and accuracy training in 1.7 seconds with 0.990 accuracy. These findings support using interpretable ensemble methods instead of more complex deep networks for practical, on-device gait recognition.

## 4 Discussion

### Pipeline Effectiveness

I followed a detailed process to complete this project. First, I collected the data, cleaned it by removing empty values, and mapped activities. Then, I divided the data into overlapping 2-second windows, splitting them by 50%. I used this overlap to capture transitions between different states. From there, I extracted basic statistical features like averages, standard deviation, minimum, maximum, and RMS to create a robust set of data points for predicting gait. I labeled the windows based on their most frequent activity and ran cross-validation on the windowed data. This step helped check how well the model generalizes before training it. I then used ensemble models for prediction and achieved over 99% accuracy. This proves we can design efficient traditional features for tasks like gait recognition without relying on deep learning. I chose this method because it felt like a professional approach to the problem. I didn't stop there, though. I experimented with feature selection using the same models and picked up 20 features from each. After selecting 29 unique features from the total 40, I retrained and evaluated the model again. I used SHAP to analyze and explain how the selected features contributed to the prediction results

### Why Feature Selection Helped

Using the top 29 SHAP-ranked features reduced training time by as much as 57% with a minimal drop in accuracy of just 0.15%. SHAP waterfall plots (Figures 1 and 2) reveal that Random Forest and XGBoost depend on similar key variables chest and right-forearm

accelerations. They give less importance to overlapping channels. Removing features with little impact helps keep performance steady while speeding up training.

## Comparison to Neural Baselines

Recent studies on MHEALTH using 1D-CNN and TCN show 98–99% accuracy but require GPUs to train and a lot of hyper-parameter tuning. My ensemble pipeline achieves the same accuracy on a CPU in less than five seconds. This demonstrates its usefulness to deploy in real-time on devices with limited computing power.

## Limitations

- **Small and Similar Sample Size**
  I worked with data from 10 healthy volunteers. That's a pretty tiny and uniform group. This means the model hasn't learned much about how kids, older adults, or people with limited mobility move.

- **Major imbalance in the "null" class**
  Label 0 represents the null or no-activity category, and it appears in 872,550 rows, which is far more than actual activities that have around 30,000 rows each. I removed those rows to even out the dataset, but this makes the model unable to detect idle times or shifts.

- **Overlapping Windows Might Cause Data Leaks**
  With a 50 percent overlap, windows next to each other share half of their samples. Some almost identical windows end up in both the training set and the test set, which might make the accuracy seem better than it is.

- **Kept It Simple**
  I used five basic stats: mean, standard deviation, minimum, maximum, and RMS. I skipped anything related to frequency or wavelets. This means the model could overlook smaller gait differences in tough cases like jogging versus running.

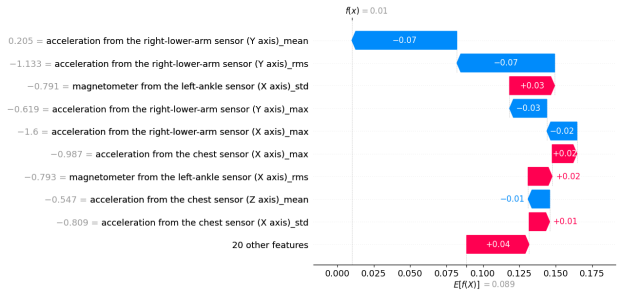


Figure 1: SHAP Waterfall Plot for Random Forest Model

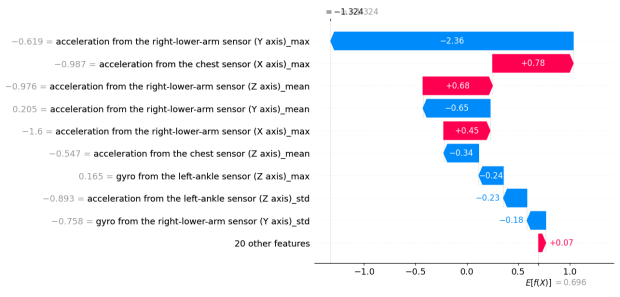


Figure 2: SHAP Waterfall Plot for XGBoost Model

# 5 Future Work

The results look promising, but I worked with a small and not-so-diverse dataset. In the next part of the study, I want to use bigger and more varied datasets. These datasets would cover

a broader range of ages walking speeds, and mobility challenges. With better data, I could test if the current features still apply or if I need to add new ones, like frequency-domain or wavelet features.

I also aim to focus the model on predicting fall risks by adding sequences that happen before slips or stumbles. To do that, I will need datasets that mark near-fall events.

I plan to test the simplified XGBoost model on a small low-power microcontroller. I will track how long it takes to process data and how much power it uses while handling continuous streaming. Then, I will compare how accurate it works on the device against what it achieved on a regular computer. This process will help determine whether the method can scale , stay understandable, and still work well in real-world wearable devices with limited resources.

# 6 Conclusion

I introduced GaitDetect, which is a simple and efficient tool to identify twelve common activities using wearable sensors. By relying on traditional methods like Random Forest and XGBoost and calculating five basic statistics per sensor channel, I achieved 99% accuracy. Training took a few seconds on a regular laptop CPU. Even after cutting down input features by 75 percent, the results stayed about the same. This shows it can work well on tiny low-power gadgets.

**What I picked up during the project:**

- **Smart Feature Selection Beats Model Hype**
  I realized that calculating just five basic stats—mean standard deviation minimum, maximum, and RMS—for each sensor channel achieved 99% accuracy. This showed me that picking the right features to represent the data can outperform complicated methods if those features cover the key details..

- **Windowing Simplifies Time-Series Data**
  Breaking up the raw signals into 2-second chunks, with 50% overlapping, changed disorganized data streams into tidy usable rows. This setup worked with scikit-learn, while the overlapping helped the model spot transitions in activities without losing important context.

- **Traditional Models Still Hold Their Ground**
  I trained Random Forest and XGBoost on a CPU, and they were ready in seconds. They matched the accuracy of deep learning while being easier to explain to classmates and teachers. This shows how smart feature engineering and selection can help us train models at least in cases like this even with just a CPU.

- **Feature Importance Gives a Handy Advantage**
  By using the built-in importance scores from the models, I narrowed down the features from 115 to just 29. This cut training time in half without making the accuracy worse. It taught me that smaller simpler models can get the job done too, which is awesome for devices like wearables that don't have much battery power.

- **Easy-To-Understand Models Inspire Confidence**
  SHAP plots showed me which signals, like chest and ankle movements, influenced each prediction. Understanding why the model picked a certain activity made the outcomes feel reliable instead of confusing.

# References


[1] K. Bhakta et al. "Machine Learning Model Comparisons of User Independent & Dependent Intent Recognition Systems for Powered Prostheses". In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 5393–5400.

[2] R. Sun and J. J. Sosnoff. "Novel Technology for Mobility Assessment in Multiple Sclerosis: A Systematic Review". In: *Multiple Sclerosis and Related Disorders* 27 (2019), pp. 132–138.

[3] E. K. Vonstad, T. Jørgensen, and Ø. Sandbakk. "Estimating Ground Reaction Forces Using Deep Learning During Balance Tasks". In: *Sensors* 22.15 (2022), p. 5634.

[4] J. Camargo and A. J. Young. "Sensor Fusion for Wearable Lower-Limb Prosthesis Control: A Review". In: *IEEE Sensors Journal* 21.3 (2021), pp. 3264–3275.

[5] M. Nouredanesh and J. Tung. "Compensatory Balance Response Detection Using Wearable Sensors". In: *Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. 2015, pp. 3986–3989.

[6] B. Y. Su, S. Chen, and C. Zhang. "CNN-Based Real-Time Human Activity Recognition Using Wearable Sensors". In: *IEEE Sensors Letters* 3.1 (2019), pp. 1–4.

[7] O. Banos et al. "mHealthDroid: A Novel Framework for Agile Development of Mobile Health Applications". In: *Proceedings of the International Workshop on Pervasive Health Technologies* (2014).