Fall detection and monitoring system using machine learning

Saif Asaad Naeem[1], Mustafa Saleh Gouda[2], Hussein Ali Mohsen[3]

Artificial Intelligence Engineering, second year, university Almaaqal, Basra, Iraq

***Abstract*---Monitoring falls in the elderly is a crucial topic that requires attention due to the unpredictability of falls in this age group. Falls are often caused by physical weakness associated with aging, making them more susceptible to falling at any moment. The fall detection system used in this research is inspired by another study, which can be found in the references section. A comparison between the two will be discussed later. This system, known as FDS, collects data from a wearable IoT device and classifies the output to distinguish falls from normal daily activities, prompting immediate medical assistance in case of a fall. In this research, we will classify falls and daily activities using several machine learning algorithms. We will utilize a dataset of acceleration and gyroscope data collected from an accelerometer and rotation sensor. From the acceleration and rotation data, we will extract input features and apply them to supervised machine learning (ML) algorithms, specifically: a decision tree, XGBoost, support vector machine (SVM), Gradient Boosting. The results show that the accuracy of fall detection reaches 94.60%, 98.77%, 88.49%, 98.77%.**

***keywords*----**Fall detection, machine learning, acceleration and gyroscope data, SVM, decision tree, XGBoost, Gradient boosting, SGD, Newton's method, random forest

I. Introduction

According to the World Health Organization, older people represent 20% of the world's population. By 2030, the number of people aged 60 and over is expected to reach 1.4 billion, and by 2050, it is expected to rise from 962 million to 2.1 billion. People over the age of 65 are often at risk of falls, at a rate of about one-third of the population. These figures are alarming. How will these numbers be managed, monitored, and cared for? It is not logical for humans to monitor the health of these numbers. Artificial intelligence must be introduced to replace humans in this matter so that humans can focus on more important things, and AI can take over this difficult task. Therefore, the inspiration for the idea of this project came about, and many ideas and information from another research [research number] were applied, and the differences will be discussed in the following pages. In this research, readings of many movements were recorded. The total number of falling movements and normal activities that were recorded is approximately 40 activities or falls in order for machine learning to recognize all possible movements that can occur to a human being, and this is what distinguishes the work of this research.

II. Hardware Device

A different set of components was used than the research that was relied upon to create the project [1]. Here is a discussion of the differences in tools between the two research projects:

Components of our project	Components of the reference
HC-12	XBee pro
GY-87	ADXL335

The XBee Pro was replaced with the HC-12 module primarily to optimize cost-effectiveness while maintaining reliable data transmission. As for replacing the ADXL335, it was for the purpose of strengthening the machine learning algorithms, as this sensor only gives readings of the person's acceleration, while the GY-87 sensor that was chosen reads readings of the person's acceleration as well as readings of rotational speed, so this gives more features in order for the machine learning algorithms to become stronger and more accurate in distinguishing between falls and normal activities.

The project consists of two devices: a transmitter device called FDS-TX, which the person wears on their chest, and a receiver device called FDS-RX, which is placed next to the computer to receive data in case of recording readings or to receive a fall alarm in case of classification. The diagram shows the two circuits with the pins connections:

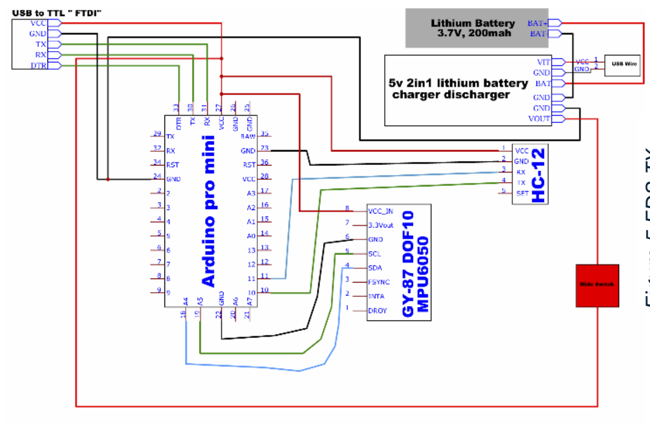


Figure 5 FDS-TX

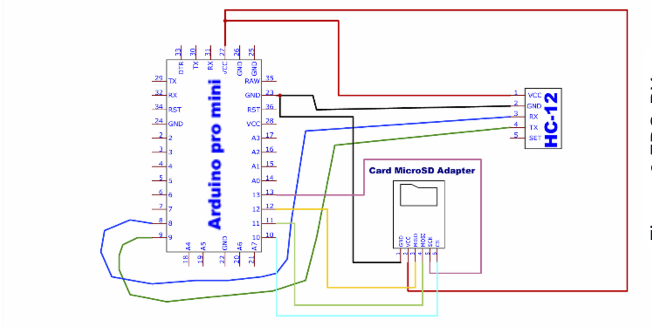


Figure 6 FDS-RX

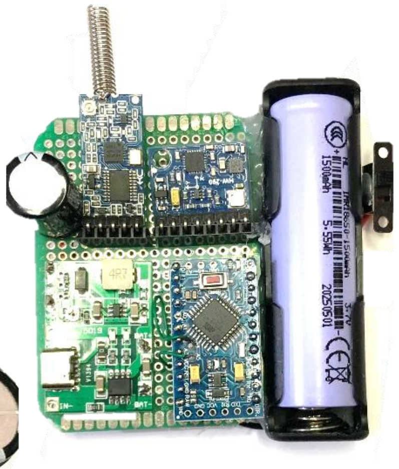


Figure 7 FDS-TX

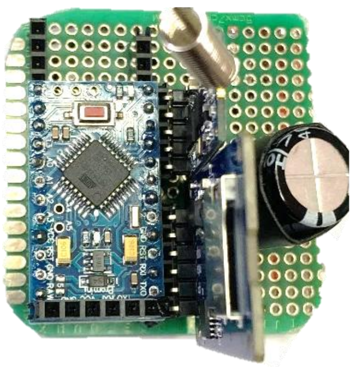


Figure 8 FDS-RX

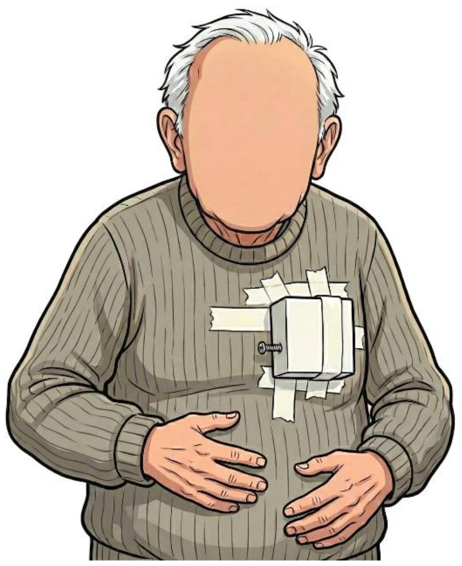


Figure 9 how to wear the FDS-TX

Important note for the errors we encountered in the connection of the components:

it is necessary to place a capacitor 25V 2200uf at the VCC and GND ports of the HC-12 because this piece when receiving or sending data will cause a significant increase in the voltage, which will affect the rest of the parts and the project may not work, so it was done Intensive placement in this place as a voltage tank.

You can get the Arduino codes in the link of the GitHub website at the reference's page [2]

III. Data collection

Readings were recorded for numerous normal activities and falls, **table that includes the cases that were recorded:**

	Fall	Normal activity
1	Falling after 360-Degree rotation	Walking
2	Falling after bending	Jogging
3	Falling after trying to get up from the chair	Running
4	Falling from behind	Jumping
5	Falling from slide backwards	Sleeping activity
6	Falling from the bed on the head	Bending
7	Falling from the bed on the side	Lying down
8	Falling from the front	Sitting
9	Falling from the left side	
10	Falling from the right side	
11	Falling from the stairs	
12	Falling next to the wall	
13	Falling on the back after missing the chair	
14	Falling on the knees	
15	Falling on the pelvis after missing the chair	
16	Falling slowly	
17	Falling while grabbing the chair1	
18	Falling while grabbing the chair2	
19	Falling while holding the wall	
20	Falling while walking	

Although the number of recorded fall scenarios appears higher in the table, the total duration of normal activity data is significantly longer due to the continuous recording nature (30s per activity) versus the momentary nature of falls.

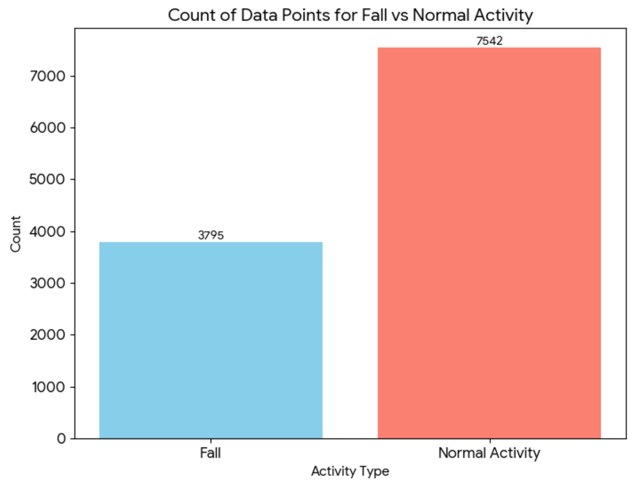


Figure 10 fall and normal activity features dataset size

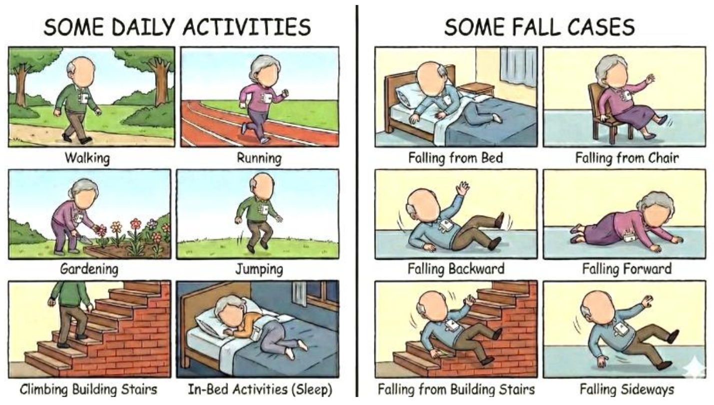


Figure 11 sketches for some of the data we recorded it

The recorded data will be in the form of x, y, z for acceleration and x, y, z for rotational speed. This raw data will not be used directly in machine learning algorithms; instead, features will be extracted from it that the machine learning algorithms can later understand. The GY-87 sensor can also record altitude and temperature readings, but these are not useful for this project because altitude does not change significantly with a normal fall, but it does change significantly with a high fall from a mountain for example. Such a fall would be fatal for an elderly person, so these readings are useless and would affect the bias of the machine learning algorithms, thus weakening the results.

As for temperature readings, there is no need to discuss them, the reason for not recording them is clear.

IV. Features extracting

After collecting and saving the data in an Excel file, we will calculate the magnitude for both the acceleration readings and the rotational speed readings. Then, using the magnitude, we will calculate the features that machine learning algorithms will use to distinguish between the falling state and the normal activity state. The features are as follows: acceleration maximum, acceleration minimum, acceleration mean, gyroscope maximum, gyroscope mean, angle change, acceleration range, acceleration variance.

The mathematics of the features:

Pre-processing: Signal Magnitude

$$Acc_i = \sqrt{a_{x,i^2}^2 + a_{y,i^2}^2 + a_{z,i^2}^2}, \qquad Gyro_i = \sqrt{g_{x,i^2}^2 + g_{Y,i^2}^2 + g_{z,i^2}^2}$$

Statistical Features (on Magnitude Series $x = [x_1, \dots, x_N]$)

$$\text{Mean } (\mu) = \frac{1}{N}\sum_{i=1}^{N} x_i \quad \text{Variance } (\sigma^2) = \frac{1}{N\text{-}1}\sum_{i=1}^{N}(x_i - \mu)^2 \quad \begin{array}{c} Max = \max\{x_1, \dots, x_N\} \\ \text{Min} = \min\{x_1, \dots, x_N\} \\ \text{Range} = Max - Min \end{array}$$

Angle Change

$$Angle\_Chg = \sum_{i=1}^{N} Gyro_i \cdot \Delta t$$

This step and what followed was worked on within PyCharm, and there is a link to the GitHub website containing the data and all the codes used on the references page [website link nu].

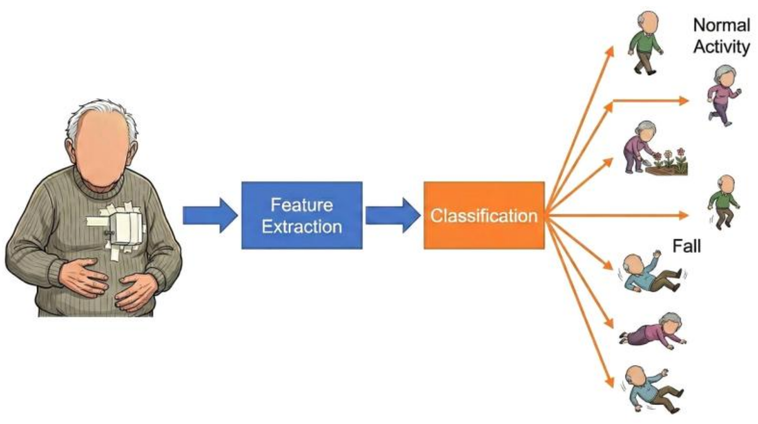
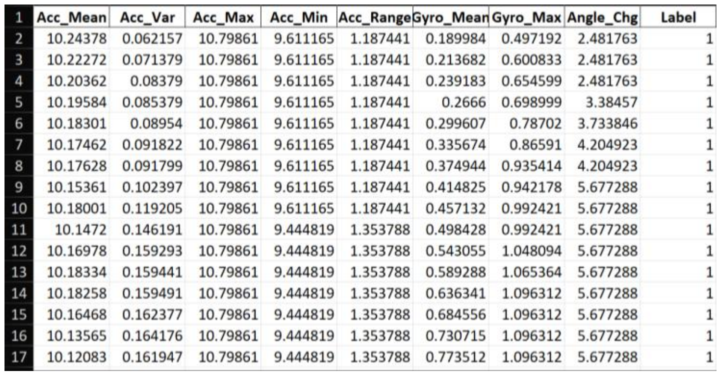


Figure 12 sensor readings path leading to classification

Important note: you can add more features like gyroscope minimum, gyroscope variance, gyroscope range. But they are not necessary because the acceleration at normal situation its value is **1G**, when falling happen its value will **decrease** until **zero**, so acceleration minimum is important to catch the free fall, unlike with gyroscope in normal situation its value is **zero**, so when fall happen the value will **increase**, so the minimum value of gyroscope will **always be zero**, so with that the range of gyroscope will always **equal the maximum gyroscope value**, so adding these features is nothing more than copying the gyroscope maximum value. For the gyroscope variance we didn't use it because it requires heavy calculations (Looping, subtracting the average, squaring, adding, dividing) on the Arduino pro mini, we only choose the acceleration variance because it's important to give the model the ability to distinguish between **"vibration"** from **"movement"**.

To transform the continuous raw data coming from the sensor into meaningful, analyzable units, we used the sliding window technique. Since the fall event lasts a short time (approximately one second), we divided the data stream into overlapping time windows. The window size was set at 20 readings (Window Size = 20 samples). Within each time window, statistical operations were applied to extract features that accurately describe the motion. This technique allowed the model to capture the temporal features of the motion, which is essential for distinguishing falls from other everyday activities**. For more details about the sliding window check the reference we based on [3]**



1	Acc_Mean	Acc_Var	Acc_Max	Acc_Min	Acc_Range	Gyro_Mean	Gyro_Max	Angle_Chg	Label
2	10.24378	0.062157	10.79861	9.611165	1.187441	0.189984	0.497192	2.481763	1
3	10.22272	0.071379	10.79861	9.611165	1.187441	0.213682	0.600833	2.481763	1
4	10.20362	0.08379	10.79861	9.611165	1.187441	0.239183	0.654599	2.481763	1
5	10.19584	0.085379	10.79861	9.611165	1.187441	0.2666	0.698999	3.38457	1
6	10.18301	0.08954	10.79861	9.611165	1.187441	0.299607	0.78702	3.733846	1
7	10.17462	0.091822	10.79861	9.611165	1.187441	0.335674	0.86591	4.204923	1
8	10.17628	0.091799	10.79861	9.611165	1.187441	0.374944	0.935414	4.204923	1
9	10.15361	0.102397	10.79861	9.611165	1.187441	0.414825	0.942178	5.677288	1
10	10.18001	0.119205	10.79861	9.611165	1.187441	0.457132	0.992421	5.677288	1
11	10.1472	0.146191	10.79861	9.444819	1.353788	0.498428	0.992421	5.677288	1
12	10.16978	0.159293	10.79861	9.444819	1.353788	0.543055	1.048094	5.677288	1
13	10.18334	0.159441	10.79861	9.444819	1.353788	0.589288	1.065364	5.677288	1
14	10.18258	0.159491	10.79861	9.444819	1.353788	0.636341	1.096312	5.677288	1
15	10.16468	0.162377	10.79861	9.444819	1.353788	0.684556	1.096312	5.677288	1
16	10.13565	0.164176	10.79861	9.444819	1.353788	0.730715	1.096312	5.677288	1
17	10.12083	0.161947	10.79861	9.444819	1.353788	0.773512	1.096312	5.677288	1

Figure 13 sample of the features [1 = normal]

VI. Fall Detection system framework

The fall detection system relies on an integrated series of data processing and decision-making operations, starting with reading raw signals from the sensor and ending with triggering the alarm if a fall is detected. Figure (14) illustrates the flowchart summarizing this entire process, highlighting the integration between the data collection, feature extraction, and machine learning model (decision tree) stages, which represent the system's brain.

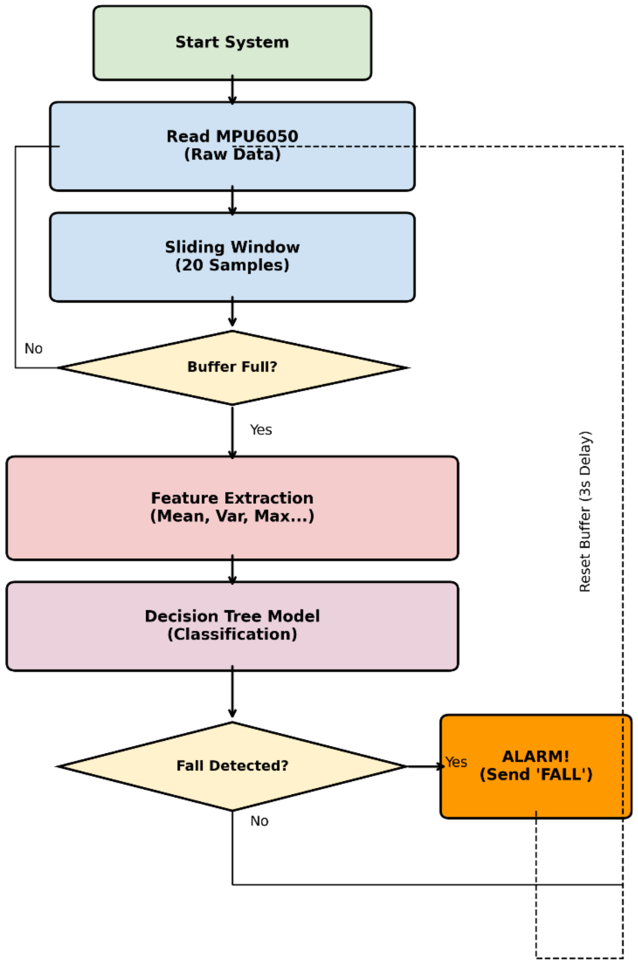


Figure 14 Fall Detection System framework

As illustrated in the figure, the trained Decision Tree Model acts as the heart of the decision-making process, receiving extracted features and classifying the action. In the following sections, we will explain in detail the algorithms that were used and trained, focusing on the Decision Tree Model, which proved superior in this application.

V. Machine learning algorithms

Machine learning enables the system to learn from the data coming out of the sensor. Raw data comes out of the sensor, then features are extracted from it, and based on these features, the algorithms decide whether this is a fall or not. Many algorithms have been used, and many methods have been used to improve these algorithms:

A. Decision tree

A decision tree is a supervised learning algorithm that uses a hierarchical, tree-like structure to make decisions. The algorithm works by iteratively dividing the instance space based on conditional rules (if-else rules) applied to the input feature values, starting from the root node and progressing to the final leaf nodes, which represent the final classification of the movement (fall or normal).

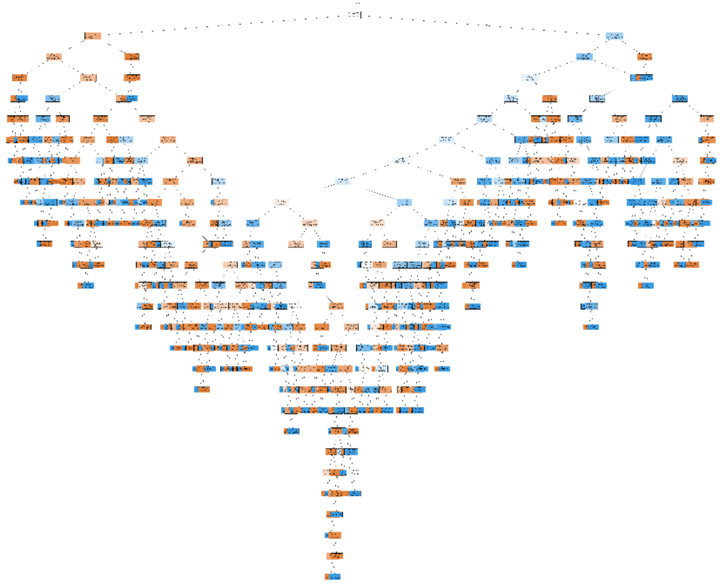


Figure 15 Pruned Decision Tree Structure

Why Decision tree?

The decision tree algorithm was adopted as the definitive model for several key reasons:

- Computational Efficiency: The model's simple logical condition-based nature makes it ideal for running on resource-constrained microcontrollers like the Arduino Pro Mini, as it doesn't require complex matrix multiplication like neural networks.

- Interpretability: The model can be easily translated into C++ arrays, allowing for a better understanding of the physical factors that led to the decision.
- Balanced Performance: Experiments have shown that the tree can efficiently capture the threshold-based nature of fall events.

Training and Improvement Methodology

To ensure the model's accuracy and robustness, we implemented advanced strategies during training:

- Class Weighting: The balanced option was used to address the data imbalance between the number of falls and walks, which increased detection sensitivity (Recall).
- Post-pruning: To avoid the problem of overfitting, we applied the Cost Complexity Pruning technique, where different alpha values were tested and the optimal tree was selected that achieves a balance between accuracy and model size.

Hardware Implementation

To enable the embedded system to implement the algorithm, we developed a special script to extract the tree structure (nodes and branches) and convert it into fixed arrays stored in program memory (PROGMEM) to save random access memory (SRAM), which allowed a complex model to run at a high response speed.

Gradient boosting

Gradient Boosting enhances performance by sequentially correcting errors from previous trees. However, its high computational complexity and memory footprint make it unsuitable for the Arduino Pro Mini. Potential solutions for deploying such complex models on embedded systems are discussed in the 'Future Work' section.

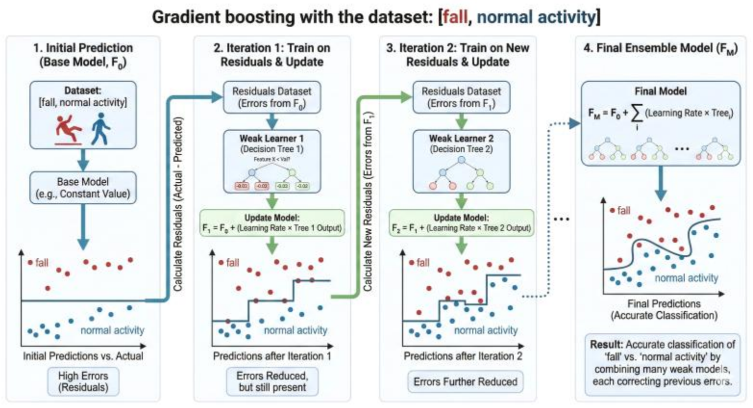

Gradient boosting with the dataset: [fall, normal activity]
1. Initial Prediction (Base Model, F_0)
Dataset: [fall, normal activity]
Base Model (e.g., Constant Value)
fall
normal activity
Initial Predictions vs. Actual
High Errors (Residuals)
Calculate Residuals (Actual - Predicted)
2. Iteration 1: Train on Residuals & Update
Residuals Dataset (Errors from F_0)
Weak Learner 1 (Decision Tree 1)
Update Model: $F_1 = F_0$ + (Learning Rate × Tree 1 Output)
Predictions after Iteration 1
Errors Reduced, but still present
Calculate New Residuals (Errors from F_1)
3. Iteration 2: Train on New Residuals & Update
Residuals Dataset (Errors from F_1)
Weak Learner 2 (Decision Tree 2)
Update Model: $F_2 = F_1$ + (Learning Rate × Tree 2 Output)
Predictions after Iteration 2
Errors Further Reduced
4. Final Ensemble Model (F_M)
Final Model
$F_M = F_0 + \sum_i$ (Learning Rate × Tree_i)
Final Predictions (Accurate Classification)
Result: Accurate classification of 'fall' vs. 'normal activity' by combining many weak models, each correcting previous errors.

Figure 16 Sketches to explain how Gradient boosting works

B. XGBoost

XGBoost (Extreme Gradient Boosting) is one of the most powerful tree-based ensemble learning algorithms, known for its speed and efficiency in handling complex tabular data. It works the same as Gradient Boosting, but the difference is this algorithm uses Regularization, this helps the algorithm to avoid overfitting, and it’s also can handle missing data unlike Gradient Boosting, and for sure it’s much faster than Gradient Boosting. This algorithm was tested in this research due to its global reputation for high accuracy and its ability to handle unbalanced data. The same as Gradient Boosting we used Random search, then Grid search.

C. SVM

Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. It tries to find the best boundary known as hyperplane that separates different classes in the data. It is useful when you want to do binary classification like Fall or Normal activity. The main goal of SVM is to maximize the margin between the two classes. The larger the margin the better the model performs on new and unseen data.

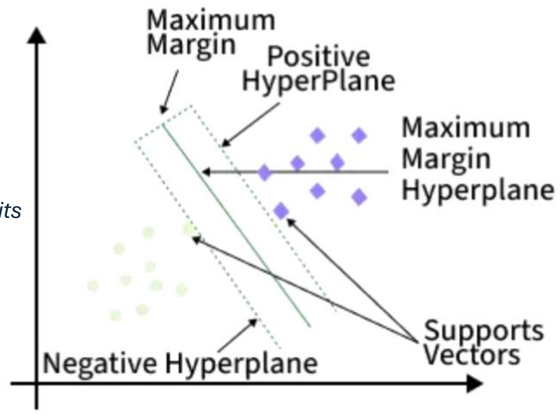


Figure 17 SVM Hyperplane splits the two classes

VII. Results and discussion

After trying many algorithms, the best algorithms for this hardware was pruned Decision tree, the Gradient Boosting and XGBoost was much better than Decision tree, but the Arduino pro mini cannot handle them, for the SVM was the worse Model, this data type hard for the SVM to splits the two classes with this much of features.

Performance of the Proposed Model:

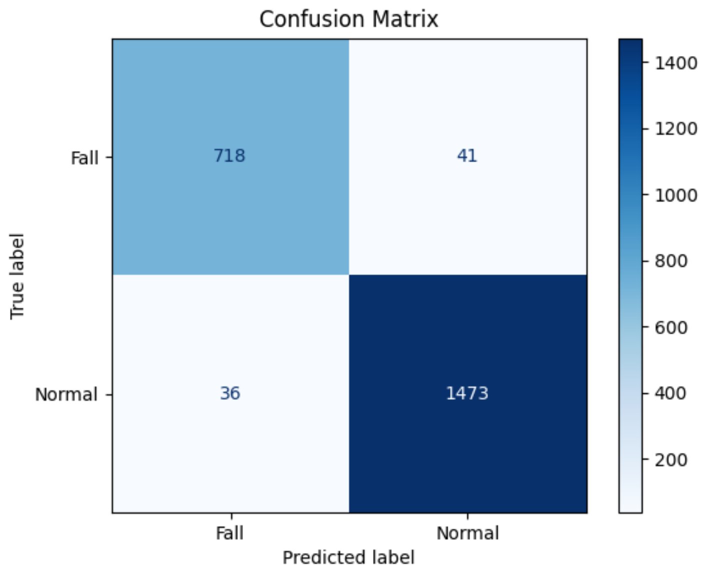


Figure 18 Confusion Matrix of pruned Decision tree

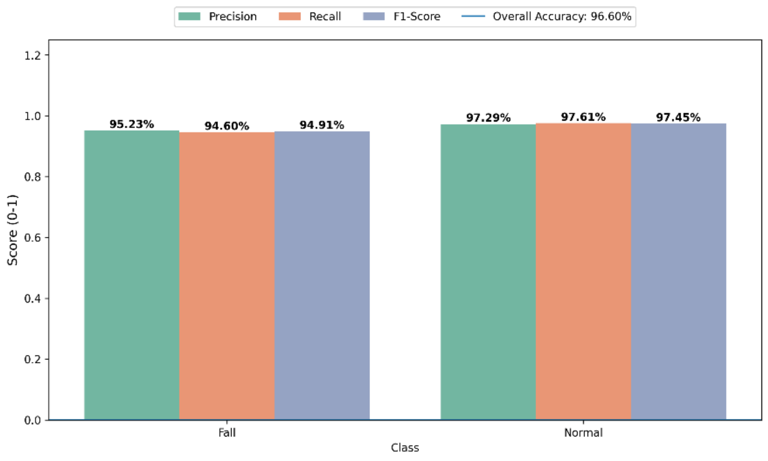
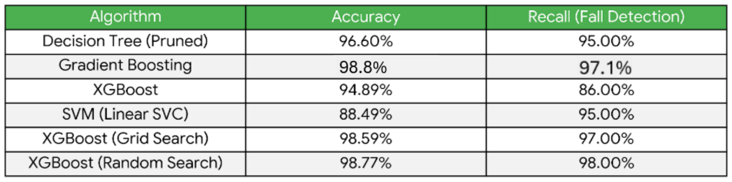


Figure 19 Performance summary of the pruned Decision tree

As shown in figure 18 & 19, the model predicted 94.60% correctly of the Fall testing data, only 5.40% predicted incorrectly (36 incorrect from 754), it maybe looks a big percent, but for Decision tree without Gradient Boosting.

Comparative Analysis:



Algorithm	Accuracy	Recall (Fall Detection)
Decision Tree (Pruned)	96.60%	95.00%
Gradient Boosting	98.8%	97.1%
XGBoost	94.89%	86.00%
SVM (Linear SVC)	88.49%	95.00%
XGBoost (Grid Search)	98.59%	97.00%
XGBoost (Random Search)	98.77%	98.00%

Figure 20 Algorithms comparative

As shown in figure Table (X): A comprehensive comparison of the performance of the machine learning algorithms used. The table illustrates the evolution of the algorithms' performance, where we observe:

- The qualitative leap in XGBoost: The use of optimization techniques (Random Search and then Grid Search) increased XGBoost's accuracy from 98.59% to 98.77%

- The ultimate superiority: With this optimization, XGBoost (Random Search) tops the list of algorithms as the most theoretically accurate model, followed by Gradient Boosting.

- The engineering decision: Despite the numerical superiority of the improved XGBoost, the Decision Tree remains the optimal choice for practical application on the Arduino Pro Mini, as it provides an excellent balance (Recall 95%) with a very low computational cost that is incomparable to the complex requirements of XGBoost.

Project Placement Experiments:

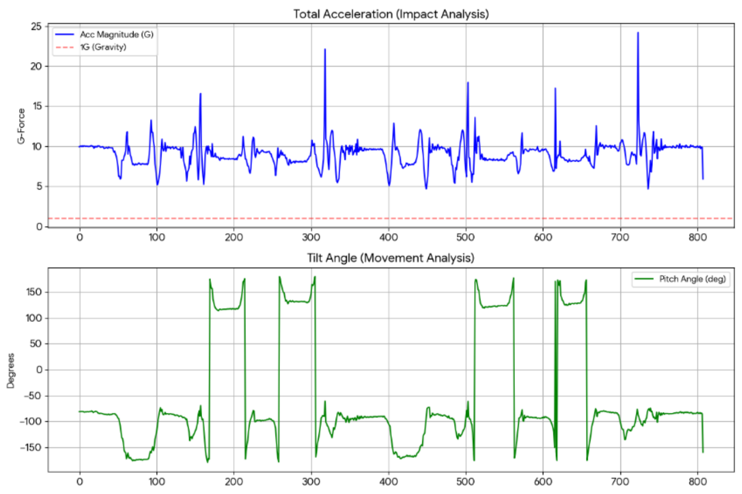


Figure 21 Islamic Prayer readings from the chest (clean)

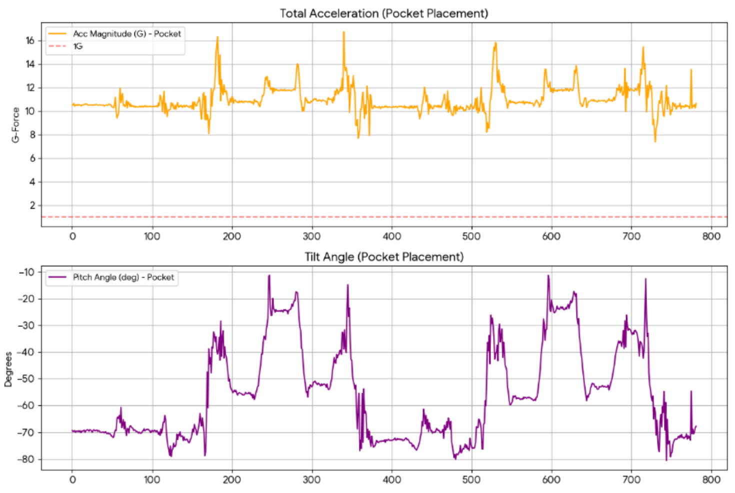


Figure 22 Islamic Prayer readings from the Pocket (clean)

The prayer data (Figure 21) exhibits a distinct dynamic pattern when the sensor is attached to the chest. The transitions between standing, bowing, and prostrating generate significant pitch angle changes, reaching up to 180 degrees, but these changes are characterized by temporal smoothness. In contrast to the acceleration signal, the G-forces remain within moderate ranges (often below 3G) and exhibit broad waveforms, reflecting the body's control over the movement.

This pattern differs radically from the "fall signature," which is characterized by:

1. a sharp, instantaneous change in angle.
2. an immediate spike in acceleration.
3. complete rest.

Therefore, the decision tree model successfully distinguished between prayer and a fall because it learned that a fall is an "impact + rest," while prayer is a "continuous, coordinated movement."

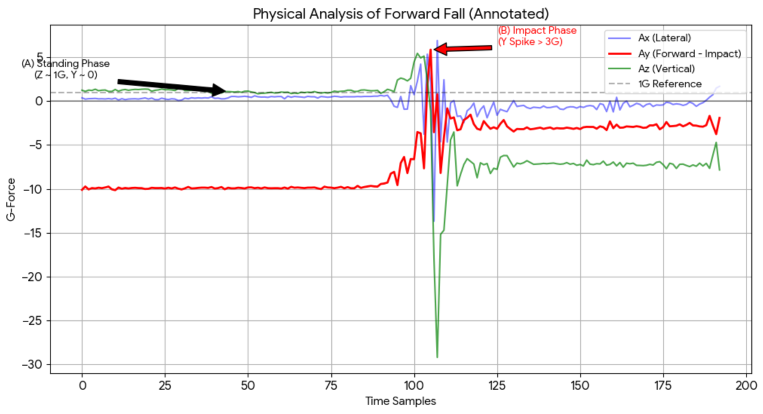
Analyzing prayer data from the pocket position (Figure 22), we found a clear deficiency in motion representation:

Loss of Angular Pattern: The angle change was confined to a very narrow range (~70°) compared to the chest position (~180°), completely obscuring the bowing and prostration movements.

Signal Distortion: The acceleration signal suffered from mechanical noise due to the device's free movement within the pocket, making it extremely difficult to extract clean features.

Conclusion: This comparison confirms that the chest position is the optimal placement for elderly monitoring applications, as it provides comprehensive readings of trunk movement, allowing the algorithm to accurately distinguish between falls and complex activities like prayer, something the pocket position cannot achieve.

Physical Motion Analysis:



Arrow (A) indicates the static phase before the fall, where the vertical forces approach 1G and the horizontal forces are zero. Arrow (B) indicates the moment of impact, where we observe a sharp jump in the forward motion axis (Ay) exceeding 3G, clearly indicating the direction and force of the fall.

IX. Conclusion

This research presented the design and implementation of a comprehensive, low-cost wearable Fall Detection System (FDS) tailored for the elderly. By integrating a GY-87 sensor with an Arduino Pro Mini and utilizing the HC-12 module for long-range communication, we successfully developed a standalone device capable of real-time monitoring.

The core innovation of this study lies in the engineering optimization of machine learning algorithms for embedded systems. While advanced ensemble models like XGBoost demonstrated superior theoretical accuracy (reaching 98.77% with hyperparameter tuning), the Decision Tree algorithm was selected as the optimal solution. It offered a robust balance between high detection sensitivity (95% Recall) and computational efficiency, allowing the model to be deployed directly onto the microcontroller's program memory (PROGMEM) without latency.

Furthermore, the study addressed critical environmental and cultural challenges. Experimental analysis confirmed that placing the sensor on the chest significantly outperforms the pocket placement, providing cleaner signal patterns essential for distinguishing complex daily activities. Notably, the system was trained to successfully differentiate between Islamic prayer movements (prostration and bowing) and actual falls, a feature often overlooked in standard datasets, thereby reducing false alarms significantly.

In conclusion, this project proves that effective fall detection does not necessarily require expensive hardware or heavy processing power. Instead, it relies on the intelligent application of feature extraction, physical signal analysis, and hardware-aware algorithm optimization. Future improvements could include integrating a GPS module for outdoor location tracking and developing a dedicated PCB to further miniaturize the wearable device and replacing the arduino with Raspberry to use XGBoost or Gradient Boosting algorithms.

VIII. Reference

1. Development of Fall Detection Device Using Accelerometer Sensor | IEEE Conference Publication | IEEE Xplore
2. srlj7/Fall-Detection-system-using-machine-learning: A wearable fall detection system for elderly using Arduino and machine learning
3. Fall Detection and Monitoring using Machine Learning: A Comparative Study