Journal of Technical Science & Technologies; ISSN: 2298-0032; e-ISSN:2346-8270; Volume 8, issue 1, 2024

# Fall Detection System based on iOS Smartphone Sensors

Mariam Dedabrishvili

*Affiliated Associated Professor Doctor, International Black Sea University*

mdedabrishvili@ibsu.edu.ge

Natia Mamaiashvili

*Ma, International Black Sea University*

16200165@ibsu.edu.ge

Ioseb Matiashvili

*PhD student, International Black Sea University*

imatiashvili3@gmail.com

## Abstract

Activities of daily living (ADLs) are a crucial aspect of human life, especially in remote health monitoring and fall detection. As smartphones have become an integral part of our daily routines, with their ability to perform complex calculations, connect to the internet, and incorporate various sensors, researchers have been inspired to explore human activity recognition systems. This paper focuses on accelerometer and gyroscope data from iOS-based smartphones. We developed a data collection app to record fall types (e.g., Falling Right, Falling Left) and fall-like activities (e.g., Sitting Fast, Jumping). Volunteers carried smartphones naturally in their pockets during experiments, presenting a challenge of noise but enhancing user comfort. We applied different Machine Learning algorithms (Decision Trees, Random Forest, Logistic Regression, k-Nearest Neighbor, XGBoost, LightGBM, and Neural Networks) to analyze the collected dataset. In contrast to typical studies, our approach replicated real-world smartphone usage. The paper presents and analyzes promising results from the study. Furthermore, we implemented the trained model as a real-time mobile application for potential users. This research illustrates the potential of smartphones in fall detection and opens the way for user-friendly solutions in remote health monitoring.

**Keywords:** Machine Learning, Fall Detection, Data Preprocessing, Data Classification, Smartphone Embedded Sensors.

## Introduction

Human activity recognition (HAR) is a crucial area in machine learning, particularly dedicated to the elderly, and plays a significant role in healthcare, remote monitoring, and surveillance [9]. Activity recognition is essential as it involves collecting data on people's daily lives, which is further monitored and analyzed by computing systems [4]. HAR is highly important in ML but is also complex, posing several challenges that need resolution.

There are various challenging issues based on different conditions, including sensor motion, sensor placement, cluttered backgrounds, and inherent variability in how activities are performed by different individuals, all of which must be addressed to achieve accurate and efficient recognition [7]. Preprocessing and selecting classification techniques are crucial steps to obtain stable and satisfactory results.

One prominent area within HAR is fall detection, as a significant portion of the world's population consists of disabled and elderly individuals over 65 years old who live alone and require assistance and monitoring to prevent serious injuries. This necessitates the use of alerts and signals, which can be achieved, for instance, through sensor-based devices or even smartphones, our everyday assistants.

Numerous published surveys on fall detection discuss various aspects of fall detection models. One of the most challenging aspects is distinguishing falls from daily living activities, which significantly impacts classification accuracy and prediction. Other considerations include providing comfortable conditions for users; in some studies, it was found that tightly attaching sensors to the body can be uncomfortable for wearers [21].

In this paper, we focus on addressing some of the primary challenges in the field, such as achieving high accuracy in algorithms, reducing false alarms, and ensuring user acceptance. We address these issues through an experimental study that involves selecting specific types of falls and fall-like activities, choosing the right sensor-based device, positioning it correctly during data collection and monitoring, applying preprocessing algorithms, and developing appropriate methods for building a learning model. This model is then used to analyze test data to determine the best classification and the optimal balance between training and testing data.

## Related Work

In the modern technology era, there are various possibilities for creating a fall detection system. We can categorize the majority of them into three types: device-based, ambiance sensor-based, and vision-based.

Regarding wearable devices, they rely on acceleration since actual falls are characterized by high accelerations. It should be noted that tri-axial accelerometers, such as those found in wearable sensors or smartphones, have reported promising results [6, 11]. These devices have the advantage of being affordable and user-friendly [15].

Smartphones serve as effective tools for human activity recognition (HAR) and fall detection. The number of smartphone users worldwide has surpassed three billion and is expected to grow by several hundred million in the next few years [22]. Today's world has demonstrated the importance and power of smartphones. They have the incredible capability to collect data and perform calculations using a combination of various sensors and powerful central processors, including separate chips for processing machine learning tasks. Specifically, smartphones come equipped with an embedded Inertial Measurement Unit (IMU), including a triaxial accelerometer, triaxial gyroscope, proximity sensor, digital compass, and barometer. By utilizing these capabilities, we can effectively detect various everyday activities and falls. Furthermore, smartphone capabilities continue to increase each year with more RAM, CPU power, networking capabilities, and more. All of these aspects make smartphones the most convenient devices for developing fall detection systems [15].

Today, Android and iOS systems are essential in the mobile software environment, as they dominate about 99% of the market. [26]. In terms of fall detection systems, both platforms provide high-quality sensor data that can be processed and evaluated by trained models. Many researchers conduct experiments on fall detection using Android-based systems due to its open-source nature and minimal restrictions [8, 15]. In contrast, iOS-based systems for fall detection have limitations due to privacy restrictions, requiring specific permissions for background tasks [5]. iOS follows a more closed approach compared to Android, restricting hardware and software modifications. Despite these limitations, several authors have developed fall detection apps for iPhones. For instance, [3] presents apps to warn users of fall risks based on signal data, and [19] offers an iPhone app that analyzes sensor data for potential falls. Hakim et al. [15] used Android and iOS systems with Matlab Mobile software to implement four machine learning algorithms for fall detection. Their analysis showed that SVM achieved the highest accuracy, predicting fall-like activities with 97% accuracy or better than other algorithms.

## Preliminaries

In this section, we address data preprocessing and classification, which are the two primary components of constructing a Machine Learning (ML) model.

### Data Preprocessing

In today's world, a vast amount of raw or real-world data is generated daily for various specific purposes [13]. In ML, we require this data to be transformed into knowledge using specialized techniques. However, data must first be made understandable for machines, necessitating preparation and processing. Data preprocessing involves determining which tools and techniques can be applied to data and how to apply them to achieve the promised benefits.

This work discusses several approaches to data processing, both in data mining during the ML process and specifically in Human Activity Recognition (HAR) for Fall Detection.

According to Han et al. [16], data preprocessing techniques typically involve four main steps:

- Data Cleaning: Addressing dirty data with missing values, noise, and inconsistencies.
- Data Integration: Combining data into a consistent data store, such as data warehouses.
- Data Transformation: Shaping and formatting data appropriately.
- Data Reduction: Decreasing the volume of the dataset to reduce data size.

These major steps also encompass sub-techniques with various solutions depending on the task, time, or approach. Different data preprocessing methods are also discussed in [1]. In their work, Garcıa et al. [13] consider Data Preprocessing both as a former and latter disciplines as given in Table 1:

**Table 1.** Data Preprocessing Techniques

| Former Disciplines | Latter Disciplines |
|---|---|
| Data Transformation | Feature Selection |
| Data Integration | Instance Selection |
| Data Cleaning | Instance Discretization |
| Data Normalization | |

During our research, we came across [24], which reviews machine learning with the scikit-learn library and Python, including tools for data preprocessing. Yuan et al. [27] detail techniques for processing sensor data to detect human activities in the context of Human Activity Recognition (HAR). They highlight two key steps for better classification results: handling noisy data with windowing techniques and performing feature engineering and model training. For more detailed information on these preprocessing methods and their specific applications, refer to the authors mentioned in [24]. It's important to note that there is a variety of methodologies and tools available, each suited to different tasks and goals, with considerations in terms of speed, accuracy, and dataset size.

### 3.2. Data Classification

Data classification, which follows data preprocessing, involves constructing a classifier model for predicting class labels from categorical or numerical data. The order of these labels is not critical as they are discrete values. In our study, we employed various classification algorithms suitable for sensor-derived numerical data (see Table 2).

**Table 2.** List of ML Algorithms used in the Study

| Names of the Algorithms & their abbreviation | |
|---|---|
| Decision Tree (DT) | XGBoost |
| Random Forest (RF) | LightGBM |
| Logistic regression (LR) | Neural Networks (NN) |
| *k*-Nearest Neighbor (*k*NN) | |

The Decision Tree algorithm, for instance, generates new features by evaluating their value at each iteration. It employs a greedy search with predefined operators, starting with an empty dataset. The algorithm adds or removes attribute-value pairs based on evaluations of class entropy, model complexity, or criteria like the Gini index (1) and Entropy (2) to calculate information gain and split nodes, where $p(C_I)$ is the probability of class in a node.

$$Gini = 1 - \sum_{i-1}^{n} p^2(c_i) \tag{1}$$

$$Entropy = \sum_{i-1}^{n} - p(c_i)\, log_2 p(c_1) \tag{2}$$

Random Forest, a supervised learning algorithm, is a valuable tool for data classification. It creates decision trees on datasets, makes predictions on each set, and returns the best or most popular class by aggregating the votes of these trees [16].

Logistic Regression, a classification algorithm, is typically used for binary outputs where a sample feature belongs to one class or not. It's based on the Sigmoid function, which forms an 'S'-shaped curve when plotted. This classifier transforms values between 0 and 1, making decisions based on the calculated Sigmoid of the weighted features [17]. The Sigmoid function is given in formula (3):

$$y = 1/(1 + e^{-1}) \qquad (3)$$

The *e* denotes the exponential constant and has a value of approximately 2.71828.

The k-Nearest Neighbor (k-NN) Algorithm classifies instances within a dataset based on their proximity to other instances with similar properties [14]. It identifies the k nearest instances to the query instance and determines their class by finding the most frequent class label. Distance measurements between sample points x and y can be calculated using methods like Euclidean Distance (4).

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \qquad (4)$$

XGBoost, or extreme gradient boosting, uses a gradient-boosting decision tree architecture. It constructs new models by predicting the errors of previous models and makes final predictions based on the collected information, aiming to minimize loss like the Gradient Descent framework [10].

LightGBM, another gradient-boosting approach, implements tree-based learning algorithms and is known for its high computational speed. Unlike other tree algorithms that grow horizontally, LightGBM grows vertically or leaf-wise. It selects leaves with significant loss reduction, allowing it to lower loss more efficiently compared to level-wise algorithms [18].

PyTorch, an open-source machine learning library, is widely used in deep learning, artificial intelligence, computer vision, and natural language processing applications. It's primarily a research-focused library that supports the construction of neural networks using the torch.nn package, making it applicable to fall detection as well [23].

A typical neural network (NN) consists of interconnected neurons that produce sequences of real-valued activations. Input neurons are activated by sensors observing the environment, while other neurons receive activation through weighted connections from previously active neurons. The learning process, or the assignment of credits, involves finding weights that enable the NN to exhibit the desired behavior [25], [11].

## Experimental Study of Sensor Data for Fall Detection

This section focuses on constructing the experimental model for fall detection using sensor data. The experimental study encompasses data acquisition, data preprocessing (including data cleaning, integration, transformation, and reduction), data classification, and results demonstration.

Figure 1 illustrates the steps of the experimental model construction using preferred algorithms.

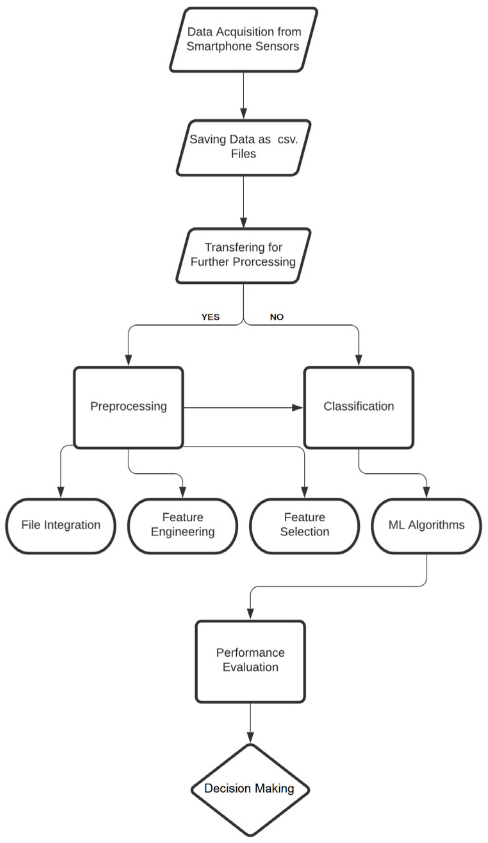


**Figure 1.** Fall Detection Flowchart

### Data Acquisition

In this work, data collection is performed using mobile embedded sensors, which are crucial sources in the data collection process [2]. These sensors are increasingly utilized in various applications as automated tools for data acquisition, facilitating decision-making processes. Sensor data is valuable for applications such as health monitoring, location tracking, activity recognition, and, notably, fall detection [12].

### Experimental Setup

in our study, fall detection experiments were conducted with the participation of seven healthy volunteers who provided full consent before the study. The mean age of the volunteers was 25 years, with an average weight of 60 kg. During data collection, each subject was instructed to intentionally simulate falls and fall-like activities while carrying their iPhone in the right front pocket of their pants without any restrictions or tightness. They performed these actions on a 15 cm thick cushion/mat in a natural living environment. Each activity was timed, with a maximum duration of 10 seconds from start to end, and data was sampled at 20 Hz. The falls and fall-like activities, along with their descriptions, are listed in Table 3, while Table 4 provides a list of engineered features.

**Table 3.** List of Fall Activities

| Fall Activity Reference | Fall Description |
|---|---|
| F1 | Fall forward |
| F2 | Fall backward |
| F3 | Fall towards left |
| F4 | Fall towards right |

application specifically developed for this study, which collected data and detected falls. Visual representations of the application can be seen in Figures 2 and 3 below.

**Table 4.** List of Engineered Features

| Features Description | |
|---|---|
| 1. x, y, z coordinates | 6. Avg of x, y, z and angle values |
| 2. x, y, z angles | 7. x/y, y/x, x/z, z/x etc. |
| 3. Magnitude of the vector | 8. Deviation from avg x, y, z and avg angles |
| 4. Magnitude of the derivative | 9. $1^{st}$ order derivative in time axis for x, y, z values, angles and |
| 5. Angles of the derivative values | magnitude |

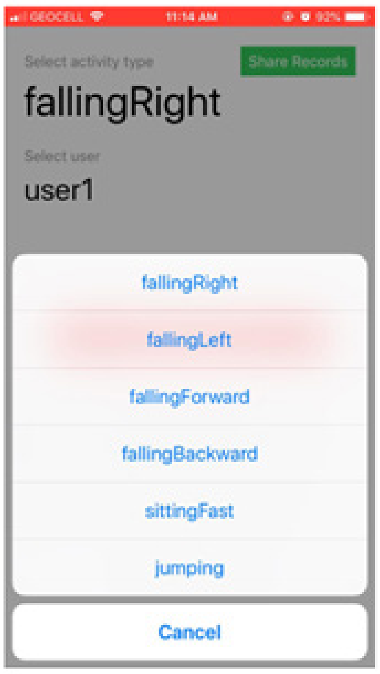
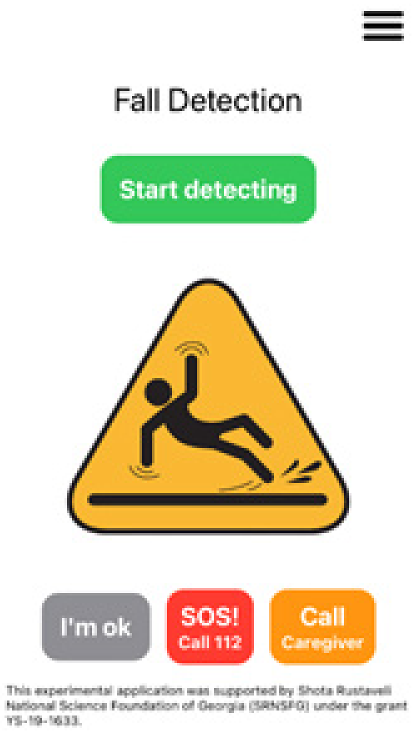


**Figure 2**. Screen of Recording Activities



This experimental application was supported by Shota Rustaveli National Science Foundation of Georgia (SRNSFG) under the grant YS-19-1633.

**Figure 3**. Screen of Fall Detection

### 4.3 Results of Data Preprocessing and Classification

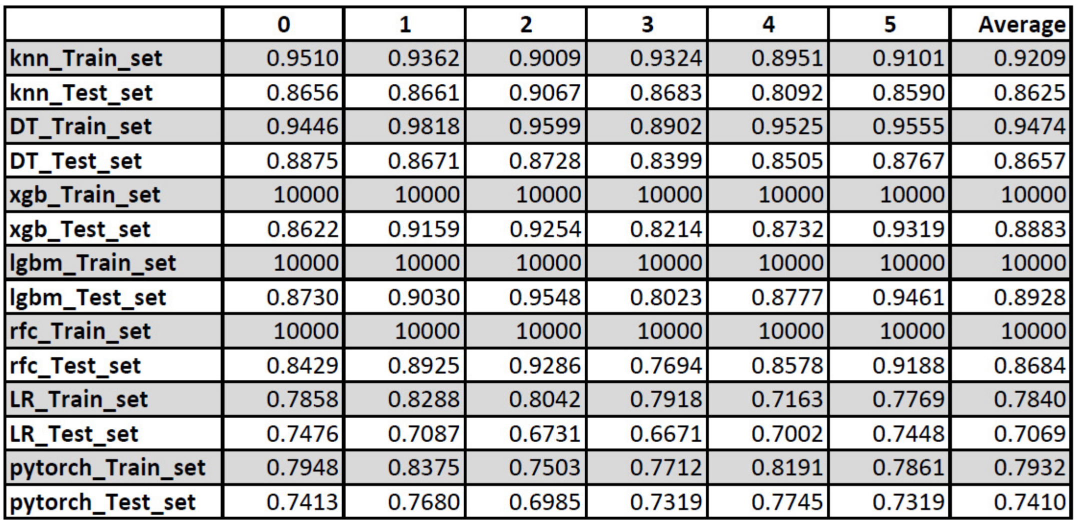
In this study, we used the Python programming language for data preprocessing and classification, given its widespread use in data science. Python is open-source and offers valuable libraries like scikit-learn for scientific computing. We used scikit-learn, along with Pandas and Matplotlib, for data manipulation and visualization.

During our experiments, we collected accelerometer and gyroscope data in separate .csv files for each activity and user. These files were later merged into a single "combined.csv" file. We cleaned the data by filling missing values with means and detecting outliers using Isolation Forest. We also engineered new features like angles and magnitudes and evaluated our models using performance measures based on confusion matrices (Table 5).

We explored various machine learning algorithms, and their performances are shown in Figure 4. For data validation, we employed the train/test split method with different ratios (0.1, 0.15, and 0.2) from scikit-learn. The 0.2 ratio proved to be the best choice. The LGBM classifier demonstrated the highest performance, as seen in Figure 5, with its confusion matrix provided in Figure 6.

**Table 5.** Confusion Matrix-based Performance Measure

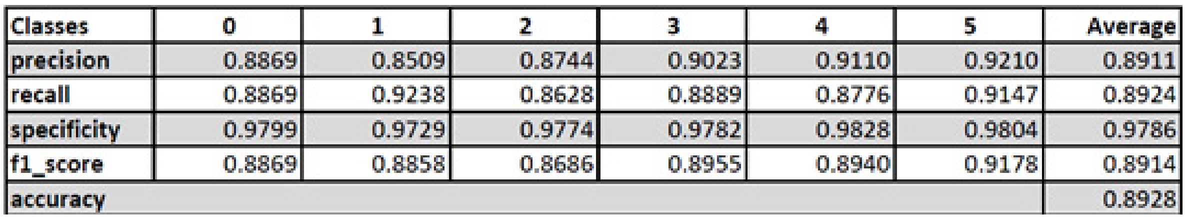
| **Data Class** | **Classified Positive** | **Classified Negative** |
|---|---|---|
| Positive | True Positive (*TP*) | False Negative (*FN*) |
| Negative | False Positive (*FP*) | True Negative (*TN*) |
| | *P=(TP)+(FP)* | *P=(FN)+(FP)* |



| | **0** | **1** | **2** | **3** | **4** | **5** | **Average** |
|---|---|---|---|---|---|---|---|
| **knn_Train_set** | 0.9510 | 0.9362 | 0.9009 | 0.9324 | 0.8951 | 0.9101 | 0.9209 |
| **knn_Test_set** | 0.8656 | 0.8661 | 0.9067 | 0.8683 | 0.8092 | 0.8590 | 0.8625 |
| **DT_Train_set** | 0.9446 | 0.9818 | 0.9599 | 0.8902 | 0.9525 | 0.9555 | 0.9474 |
| **DT_Test_set** | 0.8875 | 0.8671 | 0.8728 | 0.8399 | 0.8505 | 0.8767 | 0.8657 |
| **xgb_Train_set** | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 |
| **xgb_Test_set** | 0.8622 | 0.9159 | 0.9254 | 0.8214 | 0.8732 | 0.9319 | 0.8883 |
| **lgbm_Train_set** | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 |
| **lgbm_Test_set** | 0.8730 | 0.9030 | 0.9548 | 0.8023 | 0.8777 | 0.9461 | 0.8928 |
| **rfc_Train_set** | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 |
| **rfc_Test_set** | 0.8429 | 0.8925 | 0.9286 | 0.7694 | 0.8578 | 0.9188 | 0.8684 |
| **LR_Train_set** | 0.7858 | 0.8288 | 0.8042 | 0.7918 | 0.7163 | 0.7769 | 0.7840 |
| **LR_Test_set** | 0.7476 | 0.7087 | 0.6731 | 0.6671 | 0.7002 | 0.7448 | 0.7069 |
| **pytorch_Train_set** | 0.7948 | 0.8375 | 0.7503 | 0.7712 | 0.8191 | 0.7861 | 0.7932 |
| **pytorch_Test_set** | 0.7413 | 0.7680 | 0.6985 | 0.7319 | 0.7745 | 0.7319 | 0.7410 |

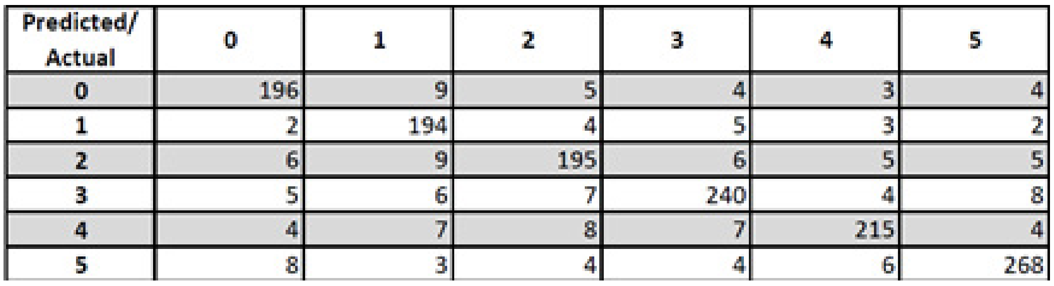
**Figure 4.** ML Algorithm Performances on the Experimental Data, test ratio=0.2

Journal of Technical Science & Technologies; ISSN: 2298-0032; e-ISSN:2346-8270; Volume 8, issue 1, 2024



| Classes | 0 | 1 | 2 | 3 | 4 | 5 | Average |
|---|---|---|---|---|---|---|---|
| precision | 0.8869 | 0.8509 | 0.8744 | 0.9023 | 0.9110 | 0.9210 | 0.8911 |
| recall | 0.8869 | 0.9238 | 0.8628 | 0.8889 | 0.8776 | 0.9147 | 0.8924 |
| specificity | 0.9799 | 0.9729 | 0.9774 | 0.9782 | 0.9828 | 0.9804 | 0.9786 |
| f1_score | 0.8869 | 0.8858 | 0.8686 | 0.8955 | 0.8940 | 0.9178 | 0.8914 |
| accuracy | | | | | | | 0.8928 |

**Figure 5.** Performance Evaluation: Score Matrix for TestSet from *LGBM*



| Predicted/ Actual | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 196 | 9 | 5 | 4 | 3 | 4 |
| 1 | 2 | 194 | 4 | 5 | 3 | 2 |
| 2 | 6 | 9 | 195 | 6 | 5 | 5 |
| 3 | 5 | 6 | 7 | 240 | 4 | 8 |
| 4 | 4 | 7 | 8 | 7 | 215 | 4 |
| 5 | 8 | 3 | 4 | 4 | 6 | 268 |

**Figure 6.** Performance Evaluation: Confusion Matrix for LGBM

## Conclusion

This work outlines the process of constructing a machine learning model for fall detection using sensor data analysis. This process involves three main stages: data acquisition, data preprocessing, and data classification.

The introduction discusses the significance of human activity recognition in daily life and highlights the challenges in the field. It is followed by a review of related work, which explores various methodologies employed in fall detection systems. Additionally, the use of smartphone capabilities for detecting abnormal activities, including falls, is discussed.

The related work section also includes a comparison between two major mobile operating systems, iOS and Android, with an emphasis on privacy issues and limitations. In the Preliminaries section, we delve into the two key components of the fall detection model: data preprocessing and classification.

This comprehensive study covers two main aspects. First, it discusses data preprocessing steps, including data cleaning techniques for handling missing values and noisy data, data integration strategies, data transformation methods for normalization, and data reduction approaches for simplifying the model through dimensionality reduction.

Second, it explores various classification algorithms, such as Decision Tree (DT), Random Forest (RF), Logistic Regression (LR), k-Nearest Neighbor (kNN), XGBoost, LightGBM, and Artificial Neural Networks. These algorithms are discussed in terms of their characteristics and usage in the study.

The study involves the collection and analysis of mobile sensor data from iOS-based devices. During the experiments, accelerometer and gyroscope data are collected using a custom iOS-based mobile application devel-

Journal of Technical Science & Technologies; ISSN: 2298-0032; e-ISSN:2346-8270; Volume 8, issue 1, 2024

oped for this project. The study focuses on four fall activities and two fall-like activities, employing data cleaning techniques such as Mean Value Imputation, Outlier Detection Algorithms, and RobustScaler. For data reduction, both Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) were tested, with LDA showing better performance on our dataset. In terms of feature engineering, magnitude and angle features are selected for data observation. Among the classification algorithms applied in research, LightGBM emerges as the most effective choice. In future work, our focus will be on further refining and optimizing the model's performance for even more accurate fall detection.

**References**

Browne, D., Giering, M., Prestwich, S. (2019). Deep learning human activity recognition. In: AICS.

Dedabrishvili, M. (2020). Effective ways to overcome classification limitations for activities of daily livings (adls). 2020 IEEE 2nd International Conference on System Analysis and Intelligent Computing (SAIC) pp. 1–7.

Dedabrishvili, M., Dundua, B., Mamaiashvili, N. (2021). Smartphone sensor-based fall detection using machine learning algorithms. In: IEA/AIE.

Garcıa, S., Ram´ırez-Gallego, S., Luengo, J., Ben´ıtez, J.M., Herrera, F. (2016). Big data preprocessing: methods and prospects. Big Data Analytics 1(1), 9.

Gent, I.P., Jefferson, C., Kotthoff, L., Miguel, I., Moore, N.C.A., Nightingale, P., Petrie, K. (2010). Learning when to use lazy learning in constraint solving. In: ECAI.

Kambria: (2019). Logistic Regression For Machine Learning and Classification.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In: NIPS.

Majumder, A.J., Zerin, I., Uddin, M., Ahamed, S.I., Smith, R. (2013). smartprediction: a real-time smartphone-based fall risk prediction and prevention system. In: RACS.

Ntanasis, P., Pippa, E., Özdemir, A.T., Barshan, B., Megalooikonomou, V. (2016). Investigation of sensor placement for accurate fall detection. In: MobiHealth.

O'Dea, S.: Mobile Operating System Market Share Worldwide, Mar 2020 - Mar 2021 (2021).