UTS UNIVERSITY OF TECHNOLOGY SYDNEY

Development of a Machine Learning Based Fall Detection System for the Elderly and Disabled

by Farhan Ahnaf Rashid

Thesis submitted in fulfilment of the requirements for the degree of

Master of Engineering (Research)

under the supervision of Dr. Xiaoying Kong

University of Technology Sydney
Faculty of Faculty of Engineering and Information Technology

December 2021

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, Farhan Ahnaf Rashid declare that this thesis, is submitted in fulfilment of the requirements for the award of Master of Engineering (Research), in the Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Production Note:
Signature removed
prior to publication.

Signature of Student: Farhan Ahnaf Rashid

Date: 20 December 2021

ACKNOWLEDGMENT

I wish to express my deepest gratitude to my first principal supervisor: Assoc. Prof. Dr. Kumbesan Sandrasegaran (Retired), my present principal supervisor: Dr. Xiaoying Kong and co-supervisor: Dr. Gengfa Fang for their encouragement to start this work and for the opportunity to be a member of the inspiring research group. Their continuous support and constructive criticism have been precious during this works.

My thanks go to all non-academic staffs of SEDE HDR team of UTS for their valuable communication during the COVID-19 pandemic.

My warmest thanks belong to my parents AKM Harun Ar Rashid and Prof. Dr. Fazilatun Nessa for their confidence in me and for being always so supportive and interested in my work and well-being and for providing unfailing support to finish this work. I wish to extend thanks to my mother for her inspiring suggestions during writing this thesis.

RELATED PUBLICATION/ RESEARCH PAPERS PRESENTED AT CONFERENCES

Farhan Ahnaf Rashid, Kumbesan Sandrasegaran and Xiaoying Kong, "Simulation of SisFall Dataset for Fall Detection Using MATLAB Classifier Algorithms". In 12th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP'2021), December 10-12, 2021, Xian, China. (Paper published in the conference proceeding).

Farhan Ahnaf Rashid, Kumbesan Sandrasegaran and Xiaoying Kong, "Simulation of MobiFall Dataset for Fall Detection Using MATLAB Classifier Algorithms" In 14th International Conference on Developments in eSystems Engineering (DeSE 2021), December 7-10, 2021, Sharjah, UAE. (Paper published in the conference proceeding).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ADL	Activities of Daily Living/Life
ANN	Artificial Neural Network
DBN	Dynamic Bayesian Network
DOF	Degrees of freedom
DSP	Digital Signal Processing
DT	Decision Tree
FPGA	Feld programmable gate array
EMG	Surface electromyography
GPS	Global Positioning System
HMM	Hidden Markov Model
I2C	Inter-integrated circuit
KNN	k-Nearest Neighbors
ML	Machine learning
MLP	Multilayer perceptron
NB	Naïve Bayes
NN	Neural Network
RFID	Radio Frequency Identification Devices
RPROP	Resilient Backpropagation
STPRtool	Statistical Pattern Recognition Toolbox
SVM	Support Vector Machine

ABSTRACT

Fall accidents from accidental injury are considered one of the significant global public health concerns, and the largest proportion of fatal accidents are experienced by elderly people. Currently, there is a demand for creating an effective machine learning-based fall detection system that is portable at a low cost.

Hence, this project is aimed at undertaking a research study on the various fall detection and navigation systems designed for the elderly and disabled and developing an effective machine learning-based low-cost fall detection system. The methodology included a study on the various fall detection systems as well as general features used in machine learning for fall and ADL (Activities of Daily Living) classifications. The most suitable potential combination of machine learning algorithms that will provide the best accuracy, precision, sensitivity, specificity and lowest training time were developed via simulation models using MATLAB. The SisFall and MobiFall datasets were used for both classification and testing for input data in simulations. SisFall dataset used ADXL345 and MMA8451Q accelerometer model, while the MobiFall dataset used LSM330DLC inertial module from a Samsung Galaxy S3 smartphone.

Up to 24 algorithms were simulated, including Decision Trees, Naïve Bayes Classifiers, Support Vector Machines, KNN and available Ensemble Classifiers. Four experiments were done, with Experiment 1 using the ADXL345 accelerometer with 5-fold cross-validation, Experiment 2 using 10-fold cross-validation, Experiment 3 using the MMA8451Q accelerometer with 10-fold cross-validation and Experiment 4 using the MobiFall Dataset with 5-fold cross-validation. The classifier models were run ten times, and each iteration was saved for further processing. Classifiers showing the most accuracy (up to 99%) in both training and testing phases included Quadratic SVM, Cubic SVM, Medium Gaussian SVM and Fine KNN.

A stacked ensemble method was simulated as well utilizing classifiers such as Medium Gaussian SVM, Cubic SVM and Fine KNN. SisFall and MobiFall Datasets were further used to train and test the classifier system. Initial testing and training had shown an improvement in accuracy (up to 99% in binary classifications) when compared to the system using individual classifiers. Results had shown a remarkable increase in precision, sensitivity and accuracy when classifying data in a binary classification system compared to classifying the data based on the specific categories (the various types of falls and

ADLs). Hence an effective system model was developed and trained to identify the data between both Fall Events and ADLs as well as separately identifying the actual type of Fall/ADLs.

CHAPTER 1

1 INTRODUCTION

Since the inception of human history, children have always outnumbered the elderly population. However, in modern times, as technology advances, people over the age of 65 are beginning to outnumber children, with the highest proportion being in developed countries. It was reported that 9.3% of the world's population were of age 65 or over in 2020 and this figure was predicted to rise to 16% by 2050 [1]. In addition, 80% of the elderly will live in low and middle-income nations by 2050 as well. This is due to various factors that follow the advancement of technology and causes changes such as an increase in life expectancy and the increasing significance of chronic health conditions such as mental health disorders and diabetes [1], [2].

This increase in ageing populations also brought forth an increasing interest regarding the issues on disability, considering the risk of disability increases with age. Around 15% of the world's population (based on 2010 estimates) is considered to have some manner of disability [3]. A disability condition prevents the person from interacting with the environment in the same way as an average person. A person on a wheelchair or a blind person requires external assistance to simply walk or interact with the environment. To assist these people, research has been done on improving the quality of life for the disabled and the elderly, especially monitoring the condition of the user or detecting any anomalies such as accidental falling.

1.1 Fall Event

A fall is defined as "an event which results in a person coming to rest inadvertently on the ground or floor or other lower-level" [4], [5]. Fall injuries can be classified to be either fatal or non-fatal (which is the majority of fall-related injuries). Fall accidents are a significant public health issue considering it is the highest cause of global "accidental or unintentional injury deaths" (after road traffic injuries). One of the critical risk factors in this type of injury is dependent on various factors such as age, gender and health of the person, which is why elderly people aged 65 and above suffer the highest number of fatal falls [4], [5].

There has been extensive research done in the field of fall detection as early as the 1970s [5]. Considering the different approaches taken in fall detection, some areas are still needed to be improved upon, and more research and techniques are needed to be adapted. Currently, there are various systems designed to detect a Fall Event. A typical system is shown in Figure 1.1. However, there is a demand to develop a robust and optimal system, and this is why fall detection is one of the most researched topics in elderly care. In fact, a noteworthy cause of nursing home admissions is falls and the fear of falling [6].

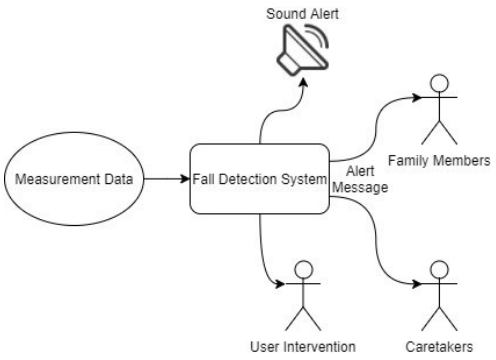


Figure 1.1: Example of a fall detection system [7]

1.2 Motivation

Considering the advances in technology and society, family dynamics change in cities, leaving the elderly population with fewer options for care [8]. To assist their everyday lives without burdening the working-age population, there is a rising demand to research a technical solution. As fall accidents are a significant public health issue, fall detection is one of the most researched fields considering today's technology [6].

Similarly, research has also been done on designing a navigation system for the disabled and elderly to track the location of the user. Considering the fact that the elderly sometimes have a lacking memory or are visually impaired, there can be cases where the person can find himself or herself in a new environment without guidance. A navigation system can offer guidance and alert the dedicated personnel or caretaker to the location [9].

The Global Positioning System (GPS) used satellites to detect the location of the user, hence being useful in outdoor conditions. However, indoor navigation is a developing field with research being done to design a reliable system. This type of system can be

deployed in commercial areas such as large airports to assist the elderly and disabled in case of emergencies [10].

1.3 Aims, objectives and significance

1.3.1 Aims

The aim of this project is to undertake a comprehensive research study on the various fall detection and navigation systems designed for the elderly and disabled design and develop a fall detection system. These aims can be addressed via the focusing on these research questions:

- What potential combination of Machine Learning (ML) algorithm for an accelerometer-based system will provide the best accuracy, sensitivity (recall), specificity and precision?
- In what ways can the classifier system be improved in terms of portability, memory, cost and energy consumption?

1.3.2 Objectives

The given aims can be achieved by separating tasks into the following objectives:

- Investigate research done on fall detection systems and navigation systems for the elderly and disabled and compare the performance of the fall detection system with and without machine learning being applied.
- Assess the performance, strengths and weaknesses of machine learning algorithms used in various systems.
- Investigate and demonstrate the best combination of machine learning methods for a fall detection system in terms of complexity, energy efficiency and real-time performance.
- Develop the fall detection classifier model using the chosen algorithms that can be simulated and tested.
- Evaluate the results presented by the new system model with comparison to existing published works.

1.3.3 Significance

A comprehensive research study/review on the various machine learning algorithms used in fall detection will grant a comprehensive knowledge on the performances of the various

machine learning algorithms used in fall detection as well as the performance, strengths and weaknesses of machine learning techniques used in various systems and identifying the most effective, less complex, accurate and portable system at a low cost. The successful development of the fall detection and navigation system using the chosen algorithms will demonstrate an effective low cost and portable solution which can be simulated and tested. Additionally, an improvement on the model to allow evaluation of the performance of other fall detection systems via fall detection datasets will exhibit an acceptable and practical model for fall detection dataset simulations.

CHAPTER 2

2 LITERATURE REVIEW

Extensive work has been done in this field by various studies. Various fall detection methods were investigated and approached by different papers. Two prominent techniques used for fall detection are rule-based methods and machine learning. Rule-based methods set up “rules” to differentiate between Activities of Daily Living (ADLs) and Fall Events based on inputs such as tilt angles and vector magnitudes from accelerometers [7]. This approach is computationally efficient, uses less energy, and is simple to incorporate into an embedded system. The fundamental issue with such systems is that precisely determining the reference or predetermined value is challenging [11], [12]. Studies found that such approaches were affected by occlusions, resolution difficulties, misclassification, and sensors' sensitivity to undesired input, all of which contributed to a higher false alarm rate. Such a classification system cannot manage the technical challenges that conventional methods encounter. As a result, an effective classification method is required that can extract the required characteristics while minimizing undesirable data and noise [11], [13].

On the other hand, machine learning algorithms use given data to train the system to identify Fall Events and ADLs, which are then used to detect actual fall events [14]. The advancement of technology with expanding stored datasets also brought about many advances in machine learning algorithms, and research is being done to improve and develop the technology and its applications [15]. Traditional approaches possess flaws, such as low camera resolution, undesired characteristics in sensor data, and misclassification by threshold-based methods, which can be addressed using machine learning-based classification. Considering the benefits of machine learning, numerous fall detection studies have implemented algorithms such as ANN, KNN, and SVM to categorise Fall Events and ADLs. Some existing studies use data collection equipment such as sensors and cameras to determine the performance of fall detection algorithms. Acceleration-based fall detection methods that depend on threshold-based algorithms have produced a significant number of false alarms and have failed to accurately extract characteristics from the signal data. Ambient sensors and equipment have also been discovered to be sensitive to undesirable vibrations, noise, and interference from the

surroundings. Threshold-based methods failed to distinguish between useful signals and noise, resulting in performance difficulties, while machine learning methods have been observed to resolve these problems and improve accuracy [11]. In addition, the comparisons reveal that the machine learning-based classifications performed better than the threshold-based classifications with a large quantity of training data [11], [13].

Overall, the machine learning method is more advanced and results in higher detection rates. Although there have been reported challenges with applying these approaches due to the requirement of high mathematical abilities and greater system resources, they are now the emerging trend, following the lower adaptability of the thresholding methods. Furthermore, no technique has gained widespread acceptance, and each study proposes a unique strategy among the many machine learning methods [16]. The different methods that have been published to date to detect falling events include image processing approach, using location sensors, smartphones sensors, accelerometers and wristbands and smartwatches. Some methods are combined in research to investigate the accuracy and speed of Fall Event detection.

2.1 Fall detection methods

There are multiple methods for detecting ADLs or Fall Events which have been researched by various studies as well as being available commercially. However, the most common methods used in such systems are as follows:

- Image processing approach
- Location sensor method
- Accelerometer in smartphones
- Wristband and smartwatches
- RFID technology
- Surface electromyography technology
- Low-cost accelerometer

These types of methods have been briefly described, and some of the research on these types of fall detection have been looked at and discussed in the following sections.

2.1.1 Image processing approach

The fall detection system uses cameras and 3D depth sensors for fall detection and ADL classifications. An example of this is shown in Figure 2.1.

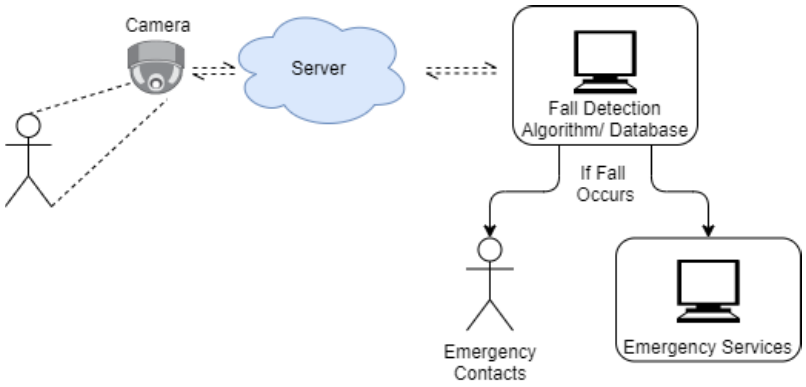


Figure 2.1: Example of a fall detection system

Yang and Lin [17] used depth image analysis to detect multiple subjects overlapping each other. The system was also designed to be usable in low-light environments such as indoor lighting. The principal characteristic for the fall detection algorithm was using a "central line" of the human profile to acquire the tilt angle of the person. Any crouching events and fall events were detected via measuring the vertical velocity of the person. This gave the advantage of people detection even if the target was partially blocked by obstacles, in addition to detecting multiple people even in low-light scenarios. This method was effective in differentiating users crouching and actual fall events, even during moments where multiple persons had been detected and overlapped. Accuracy was reported to be 94.31%, and the system had been stated to be robust and suitable for homes and public areas.

Yang and Lin [17] stated that the algorithm could be applied for other purposes using cameras of different models having the required depth resolution and depth detection distance. To provide privacy, the detection was based on handling illumination changes and identity protection.

In addition, more investigations were undertaken to study different non-invasive technologies for privacy. Planinc and Kampel [18] used audio, cameras (as 2D sensors) and Kinect as a 3D depth sensor for fall detection. The algorithm was implemented in C/C++. Fall detection using the Kinect was tested on 72 videos containing 40 falls and 32 ADLs, and the results were compared with the method using audio and cameras. For

privacy and the protection of the elderly, only depth data from the videos from the sensors were analysed, and edge detection algorithms were applied. Testing showed that the system's algorithm using depth data had outperformed other vision-based algorithms as well as algorithms based on audio information. The algorithm using world coordinates outperformed the version using image coordinates with an accuracy of 95.8%. However, the robustness of their system needed to be improved and tracking issues as well as using a combination of different approaches to fall detection was required to be researched further. Other researchers have used neural networks to detect Fall Events using multi-camera and 360-degree video feeds. The model fuses the prediction scores of the two neural networks using a weighted fusion to achieve the final output after two neural networks were trained independently on footage of Fall Events and ADLs [19]. Other studies have proposed similar techniques by implementing convolutional neural networks with RGB-D Depth Sensor Cameras [20].

There are several drawbacks to this method, considering the system is complex and expensive to set up. The location for camera installations is done indoors, such as in nursing homes and hospitals. Issues for this method include the resolution of the cameras, the distance between camera and target, the privacy of users as well as a situation when a person goes out of the camera's range, and the complex and expensive system [21]. As discussed in studies using RGBD cameras, the privacy aspect also used 3D depth to detect common activities such as standing, sitting and fall events [22].

2.1.2 Location sensor method

This method was designed by Luštrek *et al.* [6] and used tags that allow sensors to detect the positions and locations of the tags worn by the user at different parts of the body. A general example of the system is shown in Figure 2.2. Radio sensors are used to detect the location and posture of the user in the room as well as detecting Fall Events. Using multiple location tags increases the accuracy of the fall detection system.

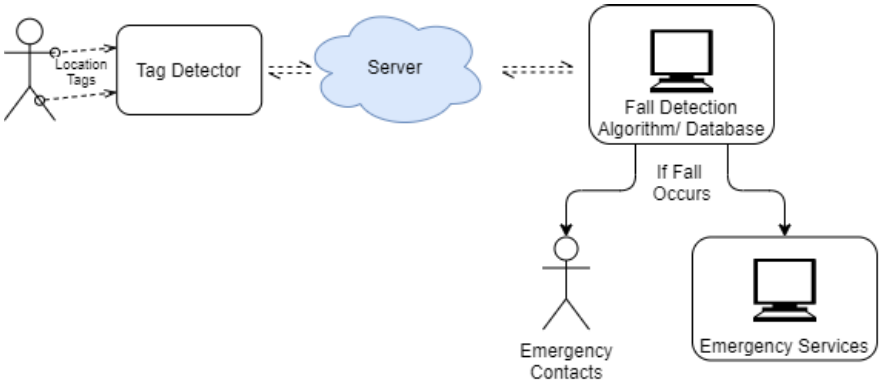


Figure 2.2: Example of a fall detection system

The variance of the acceleration vector length, the minimum, maximum and average acceleration along the y-axis, the orientation and the speed of change along the z-axis were selected as system inputs. The classifier was trained in scenarios with activities of daily living (ADL) and different types of Fall Events by five different users undergoing five repetitions. Machine Learning method used for user activity classification includes Bayesian inference for ADL types and Hidden Markov Model to remove "infeasible activity transitions". To minimise errors, a fall occurrence was declared when both classifiers showed a Fall Event.

A triaxial accelerometer was used for fall detection for comparison, using a threshold-based fall detection algorithm. The length of the acceleration vector is extracted; however, the direction of the acceleration was ignored. The average orientation of the user was measured with a location tag. If an acceleration was detected and orientation was not upright for ten seconds after the acceleration, a Fall Event was declared. The location-based system had given higher accuracy (94.7 %) than accelerometer-based methods (81.8 % with threshold, orientation and context). Luštrek *et al.* [6] noted the contextual information on location in the apartment during a potential Fall Event proved to be advantageous.

However, the required equipment is expensive compared to usual accelerometer-based methods. The tags are inconvenient, and the user can fail to accurately equip the tags in the correct manner, and this can lead to faulty readings, so they are not a very practical solution [6].

2.1.3 Accelerometer in smartphones

Modern smartphones are equipped with a wide range of sensors, such as accelerometers and gyroscopes. A fall detection system using this method uses the accelerometer, tilt sensor for fall detection and ADL detection. The location of the user is detected via the WiFi and built-in GPS location sensor. One advantage of using this system is that the portability of the system allows usage both outdoors and indoors [21]. Hence, research has been done on using these sensors to detect the user falling in real-time and use the smartphone to notify the administrator of such an event via calls or messages [23].

Lee *et al.* [23] designed a real-time fall detection system that used the acceleration sensor and tilt sensor built into a smartphone to detect Fall Events. The system used real-time location tracking using Google's Map and Google's 3D mapping service. A threshold-based fall-detection algorithm was used using the tilt sensor in the smartphone as well. When a Fall Event was detected, the algorithm used the phone's location tracking system (Google Map) to notify the administrator of the event and its location for emergency intervention. Experimental results show an accuracy within nine metres, and an error range from 12:00 pm to 6:00 pm was comparatively greater to other times in the day. The accuracy is excellent in any area due to the visual effects provided by Google Maps. Advantages of these features included swift intervention to assist the user when a Fall Event is detected. Additional wireless equipment had been used as well to track the user indoors with a one-metre error range. The overall system had achieved excellent results in both speed and accuracy. However, the precision of the smartphone's tilt sensor and fall detection algorithm needed to be improved and more research was required on minimising battery consumption. Testing should also be expanded for more age groups as well. Su and Twu [24] tested a system detection system by placing the smartphone on the waist. Data blurring was implemented by weighted moving average, and three features (impact, stillness, and weightlessness) were used to identify Fall Events via the threshold-based method.

Nevertheless, due to different smartphones having different models of accelerometers and gyroscopes, the developed software and algorithm's performance and accuracy will not be the same for each smartphone model. This can also be affected by incoming and outgoing calls [21]. The user may not have a smartphone with them at all times while

indoors. Furthermore, the accelerometer can be exploited to give sensitive information about the user, such as general activities performed by the user.

This was investigated in Das *et al.* [25] whether smartphones present a security threat to the user and whether sensors in the phone can be used by other people for malicious purposes. An Android-based platform was used in this study. Results indicate internal security to be inadequate. The sensors in the smartphone could be potentially used for malicious purposes as the sensor data could be used to identify user activities. Das *et al.* [25] also reported the accelerometer might be used to detect key presses and text without the user's knowledge.

2.1.4 Wristband and smartwatches

Recent trends have shown wrist wearables to be increasingly popular with people due to their simplicity and convenience. A fall detection system using an accelerometer built inside a wrist device/smartwatch. The location of the user is detected via the built-in Bluetooth Module/GPS location sensor. Wristbands and watches are the least intrusive and will interfere the least during the user's daily activities [21].

Kostopoulos *et al.* [26] designed a threshold-based fall detection system using a smartwatch. This system considered post-fall movements of the user and the trained fall pattern to detect Fall Events. An indoor navigation system was designed using the smartwatch's Bluetooth to provide the location of the user for monitoring. The system had successfully shown an accuracy of 96.01%. However, this system is restricted to indoors, and a GPS needs to be integrated with the system for outdoor use. Other works such as Gjoreski *et al.* [27] showed that equipping the system performs best on the ankle, knee and waist and non-dominant wrist with 85% accuracy on the wrist.

Horng and Chen [28] have also implemented a system with a smartwatch to monitor and identify Fall Events and sensors for indoor positioning as well as outdoors. Android-based platform and Amazon cloud server were set up to monitor the user. The system used a threshold classifier based on fuzzy theory to detect falls. During a Fall Event, the caregiver is notified via WiFi with details, including the location of the fall and the caregiver's identification, as well as notifications to the caregiver's app via the server. This system showed an overall 93.75% in sensitivity and 97.5% in specificity.

Disadvantages of this system are that the wrist of a user in a position of susceptible to high movement as people use their hands to do everyday tasks. Activities of Daily Living (ADLs activities) cause the sensor accuracy to be lowered [21], and a navigation system is yet to be implemented in some studies [27]. Hence, the system finds it hard to differentiate and sense whether a movement is a Fall Event or an ADL activity due to the random hand movements [27].

2.1.5 RFID technology

This fall detection system uses Passive RFID tags placed in the walls in a room, creating a tag array. RFID Antenna is used as the signal source as well as a sensor to detect the signal strength. A person's figure can be detected by analysing the signal strength.

Kaudki and Surve [29] presented a radio frequency identification based device-free activity recognition system using passive RFID tags. Passive RFID tags are placed in the surrounding walls in a room, creating a tag array, and an RFID Antenna acts as the transmitter and receiver of the radio signals. [29] stated that the system is unobtrusive, low cost and maintenance-free. Data from the RFID tags is processed for each segment in the array. Extracted features are then used for classification via dictionary learning and KNN classifier. Any Fall Events will cause the system to send a message or alarm the caretaker. The MATLAB R2016a software was used in designing the algorithm. Studies have proposed the usage of these systems in mental health facilities as well. An ensemble system was developed based on Linear Regression, Decision Tree, Random Forest, and XGBoost [30].

Disadvantages of this system include not being able to identify complex activities and being similar to the image processing method; the system is not portable and cannot be used outdoors.

2.1.6 Surface electromyography technology

This system uses the user's muscle behaviour (lower limb) to detect Fall Events via surface electromyography. Surface electromyography (sEMG) probes are used to monitor the muscles behaviour to enable a faster recognition of Fall Events and ADLs.

Leone, Rescio and Siciliano [31] developed a fall detection system via lower limb surface electromyography. The user's muscle behaviour was chosen as it may allow a Fall Event recognition. Four wireless surface electromyography (sEMG) probes were used to

monitor the lower limb muscles behaviour. Machine learning techniques were applied to overcome any disadvantages of well-known threshold-based approaches. Markov Random Field-based Fisher-Markov selector was applied for reduction of EMG signal complexity and execution time. The Linear Discriminant Analysis (LDA) classifier was used for ADL classification and fall detection to have an acceptable balance between the system capabilities, classification accuracy and costs for computations. The first phase of the data processing included a reduction in user movement noise signals and environmental noise via band-pass filtering. A USB receiver was used to store and process the data on a standalone PC. The system was developed using MATLAB and was tested via 15 participants to create a dataset holding more than 300 ADLs and 90 Fall Events. The results were obtained via a ten-fold validation scheme and showed sensitivity and specificity to be 89.1% and 87.1%, respectively.

However, similar to the location sensor method, the probes are inconvenient, and the user can fail to accurately equip the probes at the lower limb. This can lead to faulty readings, so this method is not practical enough to be implemented at this stage.

2.1.7 Low-cost accelerometer

This is one of the most commonly researched methods for fall detection. Studies were done on fall detection and using a GPS modem to monitor the user activity with a call centre for medical monitoring of the patient. Pérolle *et al.* [32] used a biaxial accelerometer with Activity monitoring being performed through a neural network. The system analysed the inputs from the biaxial accelerometer, classified and log the user's activity and detected any Fall Events. The monitoring report was sent to the call centre once a day.

Hsieh *et al.* [33] used a tri-axial accelerometer (ADXL 325) for the fall detection system. The proposed machine learning-based fall detection algorithm used multi-SVM with linear, quadratic or polynomial kernel function and KNN classifier. In the training phase, the classification model was trained by KNN classifier and SVM, which was then used to distinguish between falls and ADLs. Eight kinds of falling postures and seven types of daily activities arranged in the experiment were used to investigate the performance of the machine learning-based fall detection algorithm. The results of feature selection and classification evaluation were obtained by k-fold cross-validation method. The emulated falls were performed in ten subjects, and the results showed that the algorithm could fulfil

the requirements of adaptability and flexibility for the individual differences. The k-nearest neighbor (KNN) method with 0.1 second window size was found to have the highest accuracy of 96.26%.

Similarly, Dinh and Struck [34] developed a real-time fall detection algorithm that used machine learning methods for Fall Event classification and used a single triaxial accelerometer attached to the waist. The developed algorithm for classification used spherical coordinates to represent the intensity of the acceleration and two angles describing the orientation. A fuzzy inference system (FIS) calculated the movement activity from the transformed signal components, and the output of the FIS was recorded and classified by an ANN. The ANN is a multilayer perceptron (MLP) where Resilient Backpropagation (RPROP) is used as the training algorithm. The network had an input layer of 640 neurons, two hidden layers consisting of 30 and 20 neurons respectively, and an output layer with one neuron. For training of the neural network, training patterns were recorded and interpreted manually with 0 (not a Fall Event) and 1 (Fall Event). The system demonstrated a lower detection rate for slow falls (collapsing) and had shown an average precision of 90.3%. This was comparable to existing research using knowledge-based methods. Dinh and Struck [34] noted that slow falls (due to heart attack) could be detected by analysing the movement pattern over time.

Nevertheless, the system needed to be improved in detecting Fall Events by collapsing via adding additional rules to the system. However, this may increase calculation resources. The precision needed to be improved, and a post-fall analysis was required to be done. In addition, the classification of the severity of the Fall Event was not attempted in this study.

The system designed by Rescio, Leone and Siciliano [35], used a tri-axial MEMS (Micro Electro-Mechanical System) accelerometer, FPGA for computing and Zigbee module for wireless communication. Data was sent to an external computing platform for data analysis at 10Hz. One Class Support Vector Machine (SVM) was used to classify all Fall Events. The SVM classifier was trained using 40 falls and 50 ADLs, and 210 falls, and 150 ADLs have been used for testing (taken from the dataset). The kernel functions in the table were analysed via the MATLAB tool STPRtool. The sensitivity indicates the capacity to detect a fall, and specificity is the capacity to detect only the Fall Events. The polynomial kernel showed the best results and is used in order to limit the computational

workload. Simulations showed good functionality in controlled environments with high accuracy (97.7% sensitivity). Various performance metrics of different kernel functions in One-Class SVM were compared. The best results were the polynomial function and Gaussian Radial Basis Function. The polynomial kernel is chosen to reduce the computational workload. A study of posture was not made, but only posture changing analysis was used, limiting the computational cost as well. However, there is a further need to test the system in actual situations. Research must be done on various models of accelerometers for the robustness of the system and design for additional embedded mobile solutions (such as FPGA or DSP).

Focusing on the elderly staying at home alone, studies have been done using 3-DOF (degrees of freedom) accelerometers for both posture recognition and fall detection. Having just accelerometers removed the need to use multiple modules and have one 3-DOF accelerometer to sense the position and posture of the user. The system proposed by Van Thanh *et al.* [21] used I2C (inter-integrated circuit) interface in the connection between ADXL345 and MCU. The system used both "posture recognition module" and "vertical velocity estimation" to enhance the accuracy of the system by checking twice with an interval of 0.5 s. If a fall is confirmed, the state of the user will be checked twice, and the location of the user will be detected. The system was reported to have a sensitivity and accuracy of 100%. A message system was designed to indicate the fall positioning of the user via the GSM/GPS modem to the required personnel and attempt to can the designated number (such as relatives) to inform them of the incident [21].

Some studies have used a gyroscope in conjunction with accelerometers. Jian He *et al.* [36] designed a fall detection system having a triaxial accelerometer and gyroscope integrated in a wearable vest. This system had integrated additional functions such as GPS capabilities for monitoring, sending alerts and messaging. The system detects the reluctant acceleration and angular velocity of ADLs. The data is sent to a smartphone via Bluetooth for processing. The kNN algorithm and sliding window analysed the input data and detected any Fall Events. Falls were classified by acceleration, and angular velocity and a set of 20 accelerations and 20 angular velocities were used for classification. It can be calculated that the accuracy is 97.7%. The sensitivity and specificity are 94% and 99%, respectively. Jian He *et al.* [36] noted that there was a reduction in both false positives and false negatives and an increase in accuracy in using both the accelerometer and gyroscope. Weka, which integrates with various machine learning algorithms for data

mining, was used to compare *k*NN to other algorithms using the same training and experimental data. *k*NN algorithm has the most precision, but the Naive Bayes algorithm had the most sensitivity of 95.2%, but had a lower specificity compared to *k*NN. The precision of ANN, on the other hand was about 97.5%.

Multiple accelerometers had been used in different studies, such as the system designed by Mostarac *et al.* [37] for patients. Two triaxial accelerometers with ZigBee transceiver was used for monitoring patients and detecting Fall Events. ZSTAR wireless sensing triple-axis board was used due to its low power consumption and portability. The sensors were located on the top and bottom of the patient's clothes, and data was sent to local receivers. The receivers sent the data to the server for analysis and storage. If a Fall Event is detected, the local health care service is alerted. Personal computers were used to browse the database collected at the server. This algorithm was able to distinguish between falls (forward, backward and fall into a sitting position) and ADLs. Different outputs from the sensor for a period of six hours was simulated Fall Events and ADLs. While the system successfully monitored user activities such as sitting and falling with 100% accuracy, this number of user activities is too low for an accurate assessment of the accuracy of the system. More Fall Events and more complex ADLs are needed to be simulated for more accurate results.

Some studies, such as the system designed by Gjoreski, Lustrek and Gams [38] had used multiple accelerometers in detecting Fall Events. [38] used nine placements of up to four sensors on the waist, chest, thigh and ankle to allow fall detection via detecting the posture of the user. Machine Learning method was used for the classification model. This was used for posture recognition to classify data from the accelerometers and predict the user's posture. This was trained with 11 people. Several commonly used classification algorithms were analysed, including Naïve Bayes, SVM, J48 and Random Forest. The algorithm that achieved the best results for almost all postures was Random Forest. Using two accelerometers on both chest and waist gave an accuracy of 90% in identifying user postures, with the chest accelerometer being better in recognising postures [38]. Using 3 accelerometers was found to be adequate to accurately detect all fall events except a slow fall. The results were compared to similar results from a system using ultrawideband location tags using scenarios with events that may be difficult to different differentiate between a Fall Event and ADL. The location-based system was useful in detecting slow

falls compared using one accelerometer. The system was comparable to two accelerometers (accuracy of 93%). Using two accelerometers on both chest and waist gave an accuracy of 90% (better than location-based system) in identifying user postures with the chest accelerometer being better in recognising postures. Using all four accelerometers increased the accuracy to 99%. Gjoreski, Lustrek and Gams [38] showed that accelerometers are superior to location sensors in posture recognition. The chest and waist were the most common places to put an accelerometer and a comparison of the two revealed that they perform very similarly, with the chest having a small advantage. However, the system performance can be improved by contextual information about the location where a potential fall is taking place, since problems occurred in using one accelerometer while distinguishing some postures. Additionally, it was very hard for accelerometers to distinguish slowly falling down and lying on the bed using only acceleration signals. More work has to be done on detecting transitional activities such as standing up or sitting down as well as achieving this posture recognition using one accelerometer.

Nguyen Gia *et al.* [39] used an IoT (Internet of Things) system of a wearable sensor node containing an accelerometer and gyroscope in an IoT based fall detection system. Results showed that the system was energy-efficient, lightweight and flexible. However, the accuracy of the system was not stated, as the focus of the research was on minimising system energy consumption. Sousa *et al.* [40] implemented a low-power portable system via a tri-axis accelerometer placed on the waist for protracted usage. A threshold-based algorithm was used with minimal false alarms (97.7% specificity) and a sensitivity of 92.6%.

A disadvantage of the accelerometer method was that using a low-cost accelerometer could cause lower precision in detecting a Fall Event. In studies using more than one accelerometer, wearing them is the inconvenient and uncomfortable for the user.

2.1.8 Calculations for processing accelerometer output

The signal output for accelerometer-based fall detection systems have three phases: start of the fall, impact, aftermath and posture. Falls in the lateral direction have a higher falling velocity but falls from a bed does not have any [41]. The accelerometer output during a Fall Event is shown in Figure 2.3 [42]:

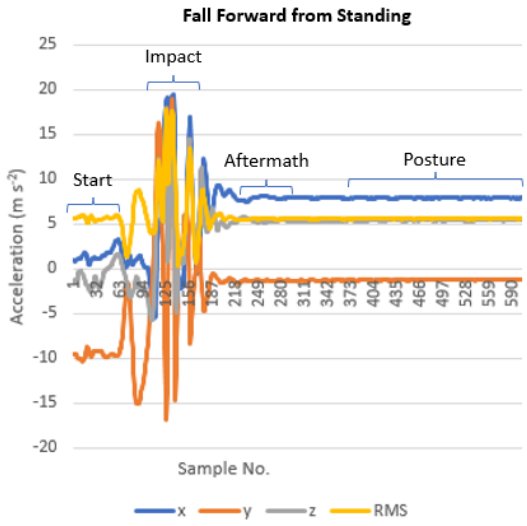


Figure 2.3: Signal output of an accelerometer during a Fall Event

Signal output during a Fall Event can be divided into the following phases:

1) **Start**: The user loses starts to drop towards the ground with the body accelerating and reaching maximum speed just before impact. At this stage, the root mean square (RMS) of acceleration will be less than 1 g.

2) **Impact**: The user impacts the ground with the RMS increasing rapidly until reaching a peak.

3) **Aftermath**: After the impact, the user stays immobile for a short period of time, with the RMS showing a flat trend.

4) **Posture**: After the period of immobility, may change posture by getting up. Conversely, the user may remain still in a different position compared to the initial orientation. Hence, the accelerometer output for each axes will be different from the initial output [41].

In calculations, the signal magnitudes a_x, a_y and a_z represents the acceleration in x, y and z axis, respectively. The magnitude of the signal vector had been calculated by using Equation 1 [23]:

$$magnitude = \sqrt{{a_x}^2 + {a_y}^2 + {a_z}^2} \qquad (1)$$

The magnitude value increases when there are large movements. This can occur during a Fall Event, and this increase in value can be analysed to detect a fall. Considering acceleration vector $a = [a_x \quad a_y \quad a_z]$, the orientation of the user is calculated by

working out the angle φ. The angle φ between the acceleration vector and the z-axis is given by Equation 2 [6], [14], [36], [38]:

$$\cos\varphi = \frac{a_z}{\sqrt{{a_x}^2+{a_y}^2+{a_z}^2}} \quad (2)$$

The following terms are used to describe and compare the effectiveness of the fall detection systems:

- **Sensitivity/Recall:** Sensitivity is the capacity of the system to detect falls and corresponds to the ratio of true positives to the total number of falls.
- **Specificity:** Specificity is the capacity of the system to detect falls only when they occur.
- **Accuracy:** Accuracy corresponds to the correct differentiation between falls and non-falls.
- **Precision:** Precision is the proportion of actual falls compared to the overall number of falls detected.

The sensitivity/recall, specificity, precision and accuracy of the system can be calculated with Equations 3 to 6, respectively [35], [43], [44]:

$$sensitivity/recall = \frac{TP}{TP+FN} \quad (3)$$

$$specificity = \frac{TN}{TN+FP} \quad (4)$$

$$precision = \frac{TP}{TP+FP} \quad (5)$$

$$accuracy = \frac{TP+TN}{TP+FN+TN+FP} \quad (6)$$

where,

True-positive (TP): a fall happens, and the algorithm detects it.

False-positive (FP): a fall does not occur, and the algorithm reveals a fall.

True-negative (TN): a daily event is performed, and the algorithm does not detect it

False-negative (FN): a fall occurs, but the algorithm does not detect it.

Some studies have used spherical coordinates (r, ϕ, θ) to detect Fall Events. This is converted from Cartesian coordinates (x, y, z) via Equations 7 to 9 [34]:

$$r = \sqrt{x^2 + y^2 + z^2} \tag{7}$$

$$\phi = \begin{cases} \arccos\left(x/\sqrt{x^2 + y^2}\right) & , y > 0 \\ 2\pi - \arccos\left(x/\sqrt{x^2 + y^2}\right) & , y \leq 0 \end{cases} ; \quad \phi = [0,\ 2\pi] \tag{8}$$

$$\theta = \arccos(z/r)\,; \theta = [0, \pi] \tag{9}$$

2.2 Navigation systems

A navigational system is defined as "a device that has the capability of knowing your current position and allows you to determine your destination" [45]. Various systems have been designed for many years to help and assist the elderly and disabled, such as those that are contained within a wheelchair having severely impaired motion skills. Research was done to develop systems that had autonomous navigation in crowded areas to navigate narrow spaces such as entering a restroom using obstacle avoidance and wall following algorithms [10].

A navigation system for a wheelchair used the iris movement and image processing for navigation which was designed using MATLAB [46]. A user interface was also designed via LabVIEW to allow the user to control wheelchair movements.

An indoor location detection was designed by Marco *et al.* [47] with ZigBee and ultrasound positioning, which provided extensive coverage and robustness even in crowded areas. The system was successfully tested in a residence for disabled people. An alarm system was implemented to warn people of risk areas and situations where they should not stay or where need a caretaker is needed. This system was useful in detecting abnormal or risky patterns like wandering in the middle of the night. An accelerometer and a button were also integrated into the device that the users wear, which made it possible to detect falls and make emergency calls. This system also used guiding messages (as voice and text messages) to assist in navigation.

Some methods such as the image processing approach, location sensor method and RFID technology are inherently based indoors. Hence, location detection can be used easily with the fall detection algorithm. For example, camera locations can be used to provide contextual information for fall detection in the systems designed in [17] and [18]. Location sensor method, such as [6] and the RFID system in [29] similarly, has built-in functionality to detect and monitor the user indoors.

Fall detection systems involving wristbands and smartwatches have location tracking features via the built-in Bluetooth Module and GPS location sensor [21]. Similarly, modern smartphones have built-in sensors; the location of the user can be detected via the WiFi and built-in GPS location sensor, such as the system in [23]. An advantage of using this system is that the portability of the system allows usage for both outdoors and indoors [21].

Accelerometer-based fall detection system does not have any built-in navigational system. However, studies have incorporated capabilities such as using ZigBee transceivers used for monitoring patients indoors in [37]. Other studies, such as the system in [21] and [36] has integrated GPS capabilities for outdoor monitoring, sending alerts and messaging. This results in less restriction to the user as the user is not needed to be confined inside.

Some authors have developed and tested the method of setting up cameras inside rooms and other sensors such as accelerometers to monitor any possible indoor falls as well as using GPS sensors to provide location inside and outside the domestic environment [48].

2.3 Commercially available fall detection systems

There are various types of fall detection systems available commercially. These range from pendants to straps and smartwatches to detect falls. Some of the multiple systems commercially available to the public are shown in Table 2.1 below.

Table 2.1: Examples of commercially available fall detection systems

Fall Detection System	Features	Price
Smartfall Blue Fall Pendant [49]	• By pushing the button on the pendant, the user may send an alarm call to the Customer Care Centre from anywhere in the house. • If the user is unable to reach the button, the Smartfall contains sensors that detect a fall and immediately send an emergency call. • For fall detection, the pendant should be worn around the neck. • The device has easily cancellable notifications to reduce false alarms. • The device battery can be replaced. • The alarm device has a range of up to 300 metres.	AUD 49.95 per month

Apple Watch SE [50], [51]	• The device is used on the wrist. • If the watch detects a hard fall, it will tap on the wrist, sound an alarm after the user is immobile for a minute and display an alert. • Alerts can be manually dismissed. • The device will call emergency services and contacts' number with the location of the fall. • The Health app automatically records Fall Events. • The system has an international emergency call features. • The watch has sensors for heart rate. • The device may detect high-impact ADLs as a Fall Event. • The device has GPS, and some models have cellular access.	Starting from AUD 429
Tunstall Gem4 [49]	• The device can be worn around the neck. • The user can use the Emergency button to create an alarm call and connect to the Customer Care Centre. • The user can manually terminate false alarms and calls. • The device has GPS tracking to locate the user. • The device has a charging base via which emergency calls can be initiated. • Voice instructions and vibration feedback are available on the device.	AUD 250
LiveLife Mobile Alarm [52]	• The device can be worn around the neck. • Automatic fall detection with loudspeaker and microphone available in the pendant. • The device uses GPS via Google Maps. • The device has a two-way voice communication system and sends emergency calls and messages with the user location to up to six contacts.	Starting from AUD 527
4G NHA Life Alarm [53]	• Mobile-based alarm with cellular features and GPS • The pendant features a microphone and loud speaker • The device can be used in the form of wristbands, clips, and lanyards. • When the emergency button is touched, or a fall is detected, this device will send a call and message to up to 8 contacts with the user location. • The user and contact can have two-way voice communication.	Starting from AUD 495
Personal 3G Mobile Life Alarm with GPS and Fall Detector [54]	• The device has cellular services as part of the alarm. • The device can be worn either around the neck, on wristbands, or via belt clips. • In an instance of Fall Event or if the user presses the emergency button, the device will call and send a message with the user location to up to 5 contacts. • The user and contact can have two-way voice communication. • The fall detection system can be manually adjusted or turned off.	Starting from AUD 440

mCareWatch [55]	• The device is used on the wrist. • The software platform connects to the smartwatch, which has cellular access. • The device has GPS and Wi-Fi access as well to disclose the user's position via the caregiver's online site and mobile app. • If the device senses a Fall Event, it prompts the user if they are fine. In lack of any reply, the device will notify the emergency contacts. • This function depletes the battery and necessitates more regular device recharging.	Starting from AUD 499

2.4 Machine learning algorithms

Several machine learning algorithms were used alongside the various types of fall detection systems. Some methods used machine learning algorithms such as Neural Network (NN) [32], [34], Support Vector Machine (SVM) [35], [38], Dynamic Bayesian network (DBN) [56], Decision Tree (DT) [57], Random Forest [38], Hidden Markov Model (HMM) [6], [58], k-Nearest Neighbor (kNN) [36], [59] and Naïve Bayes (NB) [36], [38], [60] to detect falls and accurately classify user activities from sensor data. Some studies had used multiple algorithms for analysis. Gjoreski *et al.* [38] analysed multiple algorithms such as Naïve Bayes, Random Forest, KNN and SVM. Hsieh et al. [38] used multi-SVM with linear, quadratic or polynomial kernel function and kNN classifier for the fall detection system. Conversely, studies such as Ojetola *et al.* [57] used one algorithm such as C4.5 algorithm (a Decision Tree Classifier) to create a Decision Tree using vector magnitudes and raw data. This is used to differentiate between a Fall Event and ADLs. In this section the following technical terms are commonly used in machine learning algorithms:

- **Kernel:** A kernel is a method of transforming input data as input into the required format in a higher-dimensional space or "featured space". For SVM, the non-linear data can be solved with a linear classifier using a kernel function [61].
- **Nodes:** This term is used in the context of both Decision Tree and ANN. For Decision Trees, a node is formed when a Decision Tree tries to split the data from a group into separate groups, which is determined by a set of requirements [62]. For ANN, the multiple inputs, outputs and hidden layers have a random number of neurons or nodes. The nodes are organised into layers, and a single node from a layer is connected to several other nodes in the next layer [15], [63], [64].

- **Hyperplane:** A hyperplane is a three-dimensional plane used to divide the data into two sections [65].

2.4.1 Support Vector Machine (SVM)

SVM is defined as "a classifier method that performs classification tasks by constructing hyperplanes in a multidimensional space that separates cases of different class labels" [66], [67]. An iterative training algorithm is used to minimise the error function. SVM algorithms use a kernel function to map the non-linear input data to a higher dimensional space for analysis [56]. The support vectors are the data points closest to the hyperplane. These points are considered to maximise the margin for the hyperplane, as shown in Figure 2.4 [68].

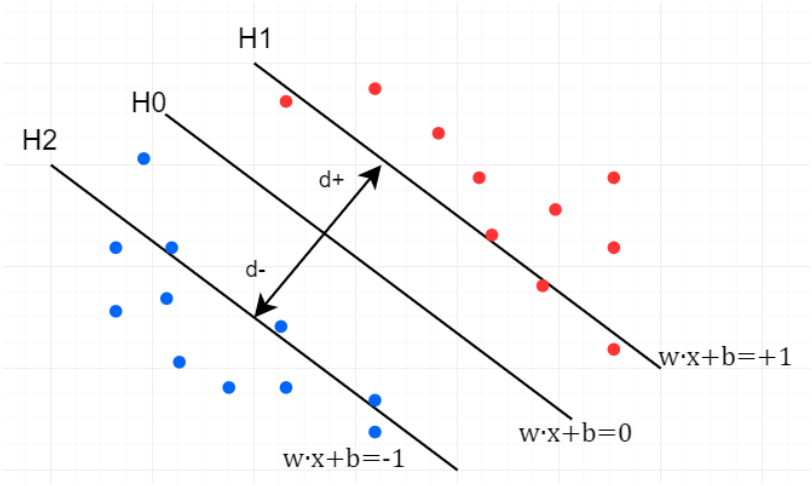


Figure 2.4: Graph showing the hyperplanes

As shown in Equations 10 and 11, Hyperplane H is such that:

$$w \cdot x_i + b \geq +1 \; when \; y_i = +1 \qquad (10)$$

$$w \cdot x_i + b \leq -1 \; when \; y_i = -1 \qquad (11)$$

H_1 and H_2 are the planes and described in Equations 12 and 13:

$$H_1 : w \cdot x_i + b = +1 \qquad (12)$$

$$H_2 : w \cdot x_i + b = -1 \qquad (13)$$

The points on the planes H_1 and H_2 are the tips of the support vectors and the plane H_0 is the median in between where $w \cdot x_i + b = 0$. The margin of a separating hyperplane is the line consisting of d+ and d– [68].

Studies such as Rescio, Leone and Siciliano [13] had designed the fall detection system based on One-Class Support Vector Machine (OC-SVM) as it was reported to be less "computationally intensive" than other algorithms such as a neural network.

2.4.2 Artificial Neural Network (ANN)

ANN is a supervised algorithm that contains multiple inputs, outputs and hidden layers with a random number of neurons or nodes. The nodes are organised into layers, and a single node from a layer is connected to several other nodes in the next layer, as shown in Figure 2.5. When the node receives input data, it is multiplied by a number or "weights" that is assigned to the node. The resulting data is sent to the next layer if the value exceeds a threshold, and the next layer goes through a similar process until the data exits the output node. When an ANN is trained, all weights and thresholds are set to random values and are adjusted until the inputs give the desired outputs [15], [63], [64]. Studies such as the system designed by Pérolle *et al.* [32] used a biaxial accelerometer with fall detection with activity monitoring being performed through a neural network.

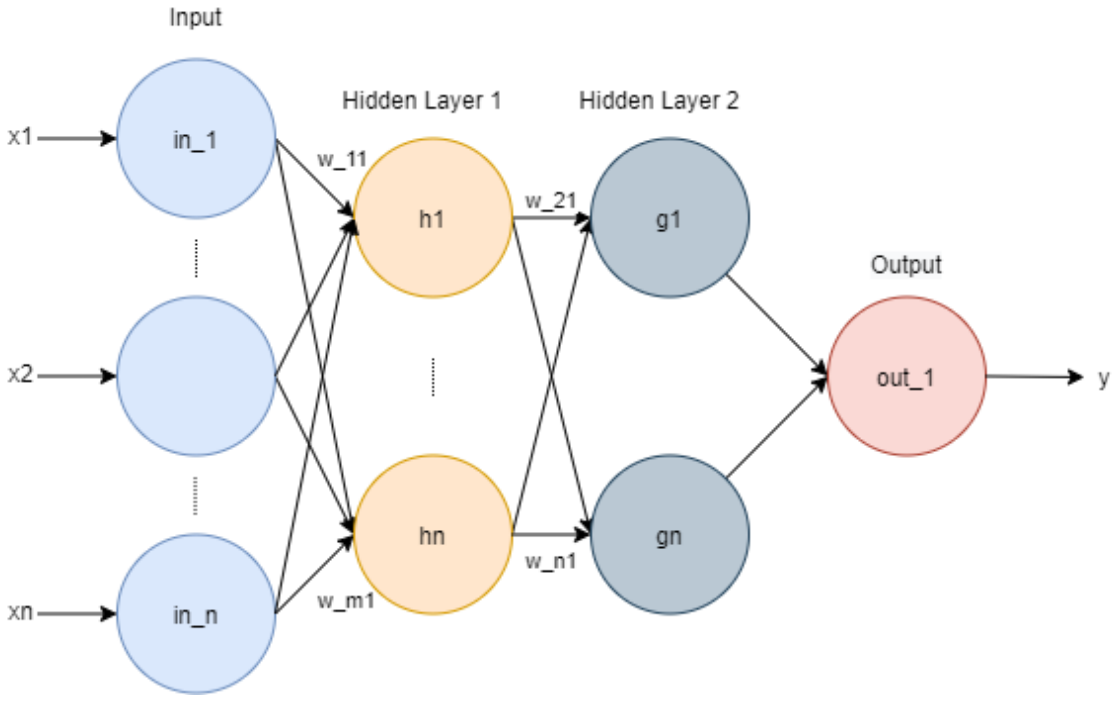


Figure 2.5: Artificial Neural Network

2.4.3 Naïve Bayesian Network

The Naïve Bayes method is a probabilistic algorithm based on Bayes' theorem. The algorithm assumes that all input data (I_n) are independent of each other, even if they are from the same class (C), as shown in Figure 2.6 [69].

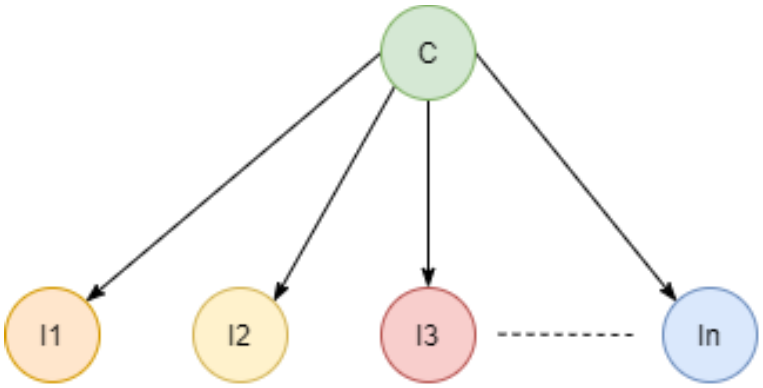


Figure 2.6: Naïve Bayesian method

During the training phase, the algorithm is trained to calculate the probability of the outcome, depending on the training data. In the testing phase, the test data is entered, and the algorithm calculates the posterior probability for all trained classes. The class that has the highest probability is the output [15]. As shown in Equation 14, the variable y is the class variable which represents if it is suitable, given the conditions while variable X represents the parameters/features [70], [71]:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)} \tag{14}$$

X is given as $X = (x_1, x_2, x_3, \dots, x_n)$, where $x_1, x_2, x_3, \dots, x_n$ represent the features.

By substituting for **X** and expanding using the chain rule (Equation 15):

$$P(y|x_1, x_2, x_3, \dots, x_n) = \frac{P(x_1|y),P(x_2|y),\dots,P(x_n|y)P(y)}{P(X)} \tag{15}$$

For all entries in the dataset, to find the class y with maximum probability (Equation 16):

$$y = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y) \tag{16}$$

Studies such as Li *et al.* [60] used Naïve Bayes for classification of ADLs in their fall detection system.

2.4.4 Random Forest method

Random forest is a classifier comprised of various decision trees. Training data is used to "grow" Decision Trees [72]. In a DT, a dataset is entered into a node that has features or criterion that split the data into subsets. Each subset is led via "branches" to other nodes (shown in Figure 2.7), and the process continues until the end-nodes, or "leaves", is reached. This is considered as the classification for the dataset [73].

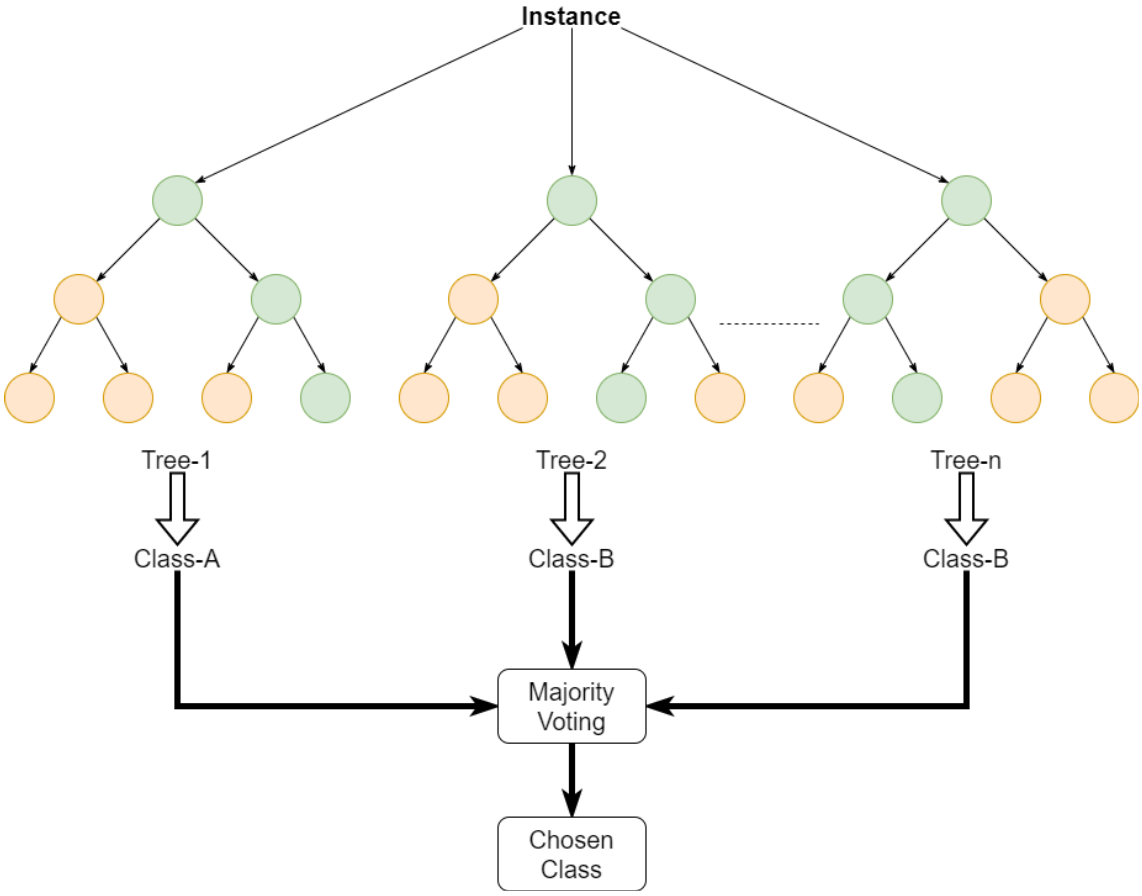


Figure 2.7: Random Forest method

A Random Forest grows numerous decision trees in the training phase, with each tree having the training data as input. The algorithm then accumulates all the classification predictions from DTs and selects the prediction that gives the most significant number of votes. This is then given as the output for the result. [74], [75].

For example, two child nodes in a binary tree (shown in Figure 2.8) can be assumed to as shown in Equation 17 [75]:

$$ni_j = w_j C_j - w_{left(j)} C_{left(j)} - w_{right(j)} C_{right(j)} \qquad (17)$$

where,

ni_j = the importance of node j

w_j = weighted number of samples reaching node j

C_j = the impurity value of node j

left (j) = child node from left split on node j

right (j) = child node from right split on node j

The importance for each feature on a decision tree is then calculated in Equation 18:

$$fi_j = \frac{\sum_{(j:node\ j\ splits\ on\ feature\ i)} ni_j}{\sum_{(k \in all\ nodes)} ni_k} \quad (18)$$

where,

fi_j = the importance of feature i

ni_j = the importance of node j

The importance of the final feature is its average over all the trees. The sum of the feature's importance value can be shown by Equation 19 [75]:

$$RFfi_i = \frac{\sum_{(j \in all\ trees)} normfi_{ij}}{T} \quad (19)$$

$RFfi_i$ = the importance of feature i calculated from all trees in the Random Forest model

$normfi_{ij}$ = the normalised feature importance for i in tree j

T = total number of trees

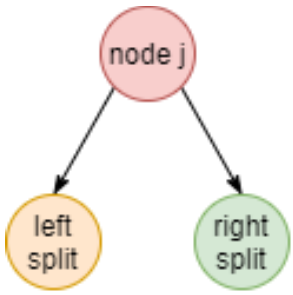


Figure 2.8: Binary Tree

Many studies have used this classifier in designing fall detection systems. Luštrek *et al.* [6] used Random Forest classifier for user's ADL classification and fall detection.

2.4.5 k-Nearest Neighbors (KNN) algorithm

KNN algorithm considers the distances of inputs and groups close datapoints together into k groups. This is done by calculating the Euclidean Distance between data points "d", using Equation 20 [76]. New sample data are classified depending on the closeness of its neighbouring groups [77].

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \qquad (20)$$

The above formula takes in n number of dimensions or features in machine learning. The data point, which is located at the minimum distance from the test point, is assumed to belong to the same class (shown in Figure 2.9).

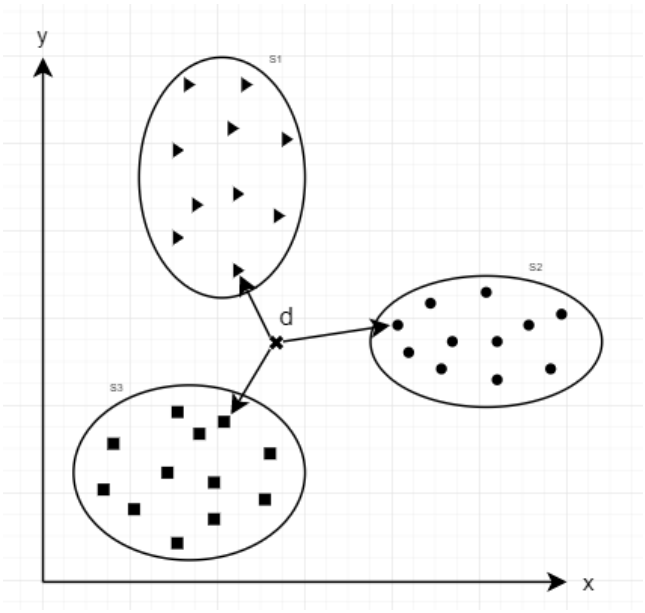


Figure 2.9: KNN example as a diagram showing the data classes and datapoint "d"

Studies such as Zhang *et al.* [59] used KNN for the classification of ADLs and detection of Fall Events. Jian He *et al.* [36] also primarily used KNN as the classifier in their fall detection system and compared the results to other algorithms such as Naive Bayes and ANN via Weka, a software used with various machine learning algorithms for data mining.

2.4.6 Hidden Markov Model (HMM)

The HMM is based on supplementing the Markov chain. A Markov chain is defined as "a model that shows the probabilities of sequences of random variables, states, each of which can take on values from some set." These sets may consist of tags, words or symbols representing a variable [78]. A Markov chain makes a very strong assumption that if the future in the sequence is to be predicted, the states before the current one have no impact on the future outcomes except via the current state [79]. Unlike the regular Markov process, hidden Markov model states are stated to not be directly visible to the observer. The observer could only obtain the value of output, which is dependent on the value of the state, as shown in Figure 2.10 [58], [80].

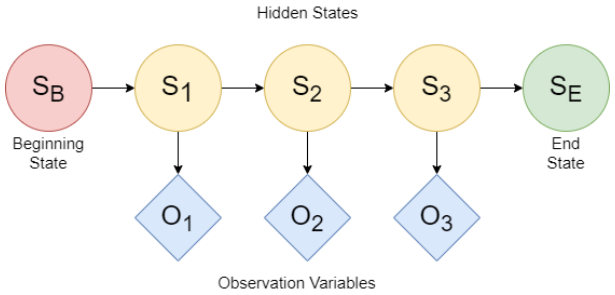


Figure 2.10: Hidden Markov Model

Wang *et al.* [58] designed a fall detection system using a tri-axial accelerometer worn at the waist that used HMM for ADL and Fall Event classification. Other studies such as Luštrek *et al.* [6] used HMM to remove "infeasible activity transitions" during ADL classifications.

CHAPTER 3

3 METHODOLOGY

A methodology can be described as "a system of practices, techniques, procedures and rules used by those who work in a discipline" [81]. In this context there are multiple ways a project can be tackled and different methodologies exist to guide the project along and aid in tackling any setbacks that can arise during the project's duration [82]. An instance on how this model of this project was structured is shown in Figure 3.1.

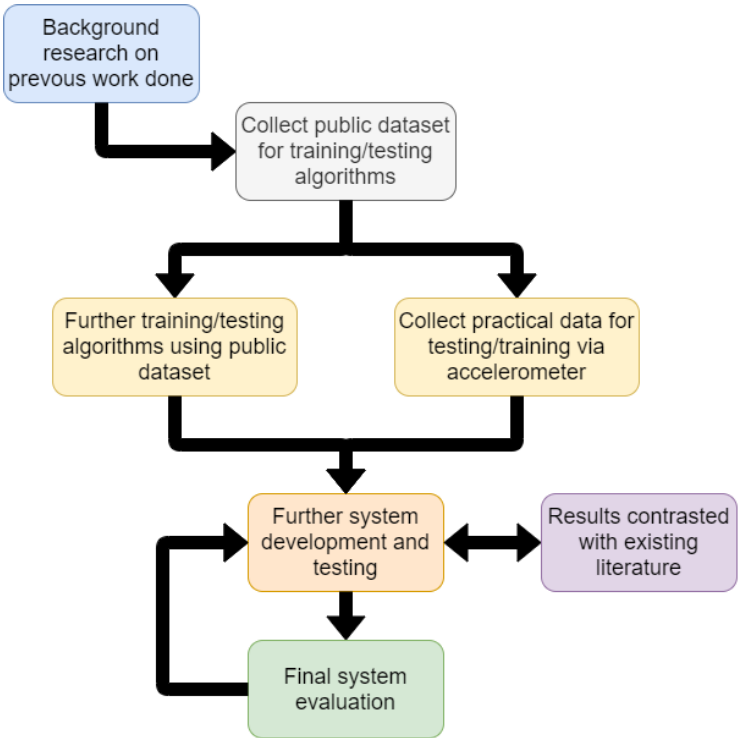


Figure 3.1: Flowchart of the project

In this model, tasks were dependent on the tasks done in the previous step. The design of the system and the coding of the software was implemented in steps. Each step was noted, and the technology used was needed to be understood to proceed to the next step. Future modifications can be attempted to improve the project based on this model.

3.1 Requirements

At the first step in the model, all requirements and information on the project were researched, followed by the analysis of the methods of system designs. This includes all

information on fall detection systems (from the literature review) and the algorithms used in the system.

3.2 System design

Keeping existing models in consideration, a proposed model of a fall detection is shown in Figure 3.2. The signal from the accelerometer is processed to a readable form and undergoes feature selection. The data is then fed into the classifiers for classification and tested for ADLs and Fall Events.

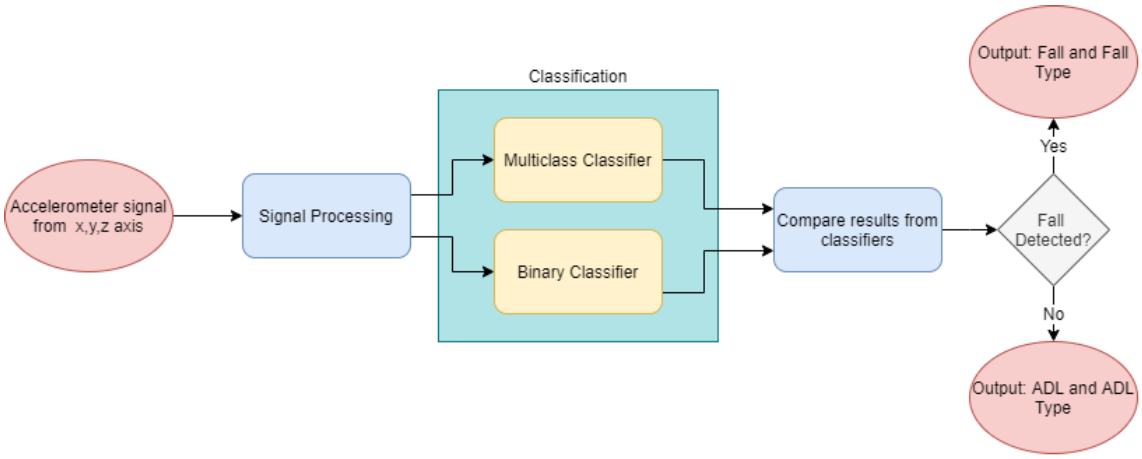


Figure 3.2: Diagram of the proposed system

As shown in Figure 3.3, the data was acquired from input devices such as accelerometers and gyroscopes. This was then put through pre-processing such as filters (if available) to remove noise from the inputs and to extract the required features. The features extracted from the inputs was passed to the next stage to calibrate the system. If the system is already calibrated, the trained machine learning algorithms will use the features to identify and classify Fall Events and ADLs.

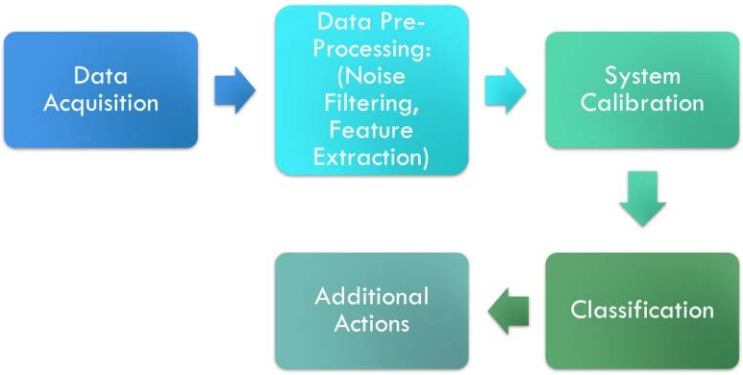


Figure 3.3: System data flow for the system

3.3 Dataset acquisition for simulation

Before the actual implementation of the fall detection system, the various machine-learning techniques were simulated to identify the most accurate and efficient algorithm or combination of algorithms. For this project, available machine-learning algorithms in MATLAB was used for simulation to acquire the best combination of algorithms for the software. Simulations required input data, and, in this scenario, datasets were used as input for the system and algorithm to simulate all events and ADLs. Such datasets include the SisFall [83], and Mobifall [84] datasets. The publicly available datasets shown in Table 3.1 were considered for classification and testing for the best viability of the ML algorithms. However, for the purposes of these experiments, SisFall and MobiFall datasets were chosen to be used as input data for simulations.

Table 3.1: Available public datasets considered for simulation

Dataset	Source	Samples	ADL	Fall	Type	Placement
SisFall	[83]	4510	19	15	2 accelerometers and 1 gyroscope	Waist
MobiFall	[84]	3200	12	4	Accelerometer and gyroscope in smartphone	Thigh

3.3.1 SisFall dataset

The SisFall database consisted of 4,510 files, with each file containing a single activity. There were a total of 34 recorded and measured activities (19 ADLs and 15 Fall Events).

Fall Event and ADL data were acquired via three sensors (2 accelerometers and 1 gyroscope) at a frequency sample of 200 Hz. The two accelerometer models were:

- ADXL345, which has a resolution of 13 bits and a range of −16g to +16g
- MMA8451Q, which has a resolution of 14 bits and a range of −8g to +8g

Test participants for this dataset include adults between 19 and 30 years old and elderly people between 60 and 75 years old. The ADLs and Fall Events that were identified and classified are shown in Table 3.2 and Table 3.3, respectively [83]. An example of the raw data for activity D01 in text file format is shown in Figure 3.4. Each row represents an instance of reading taken from the ADXL34 accelerometer, MMA8451Q accelerometer and the ITG3200 gyroscope. The first three columns represent the x, y and z-axis readings for the ADXL34 accelerometer, the following three columns represent the x, y and z-axis readings for the MMA8451Q accelerometer and similarly, the last three columns represent the x, y and z-axis reading from the ITG3200 gyroscope.

Table 3.2: ADL categories [83]

D01	The person walks at a slower pace
D02	The person walks at a faster pace
D03	The person jogs at a slower pace
D04	The person jogs at a faster pace
D05	The person goes up and down the stairs at a slower pace
D06	The person goes up and down the stairs at a faster pace
D07	The person sits on a chair (of half-height), waits and stands up (at a slower pace)
D08	The person sits on a chair (of half-height), waits and stands up (at a faster pace)
D09	The person sits on a chair (of low-height), waits and stands up (at a slower pace)
D10	The person sits on a chair (of low-height), waits and stands up (at a faster pace)
D11	The person tries to get up and falls on chair from a sitting position
D12	Person lies down from a sitting position slowly, waits and then gets back to previous position
D13	Person lies down from a sitting position faster, waits and then gets back to previous position
D14	From lying on back position, switches to lateral position, waits and returns to previous position
D15	Person bends knees at a slower pace, from a standing position and returns to previous position
D16	Person bends (without knees) at a slower pace, from standing and returns to previous position
D17	The person sits in a car from standing position, waits and returns to previous position
D18	The person trips (but does not fall) while walking
D19	The person jumps softly (but does not fall)

Table 3.3: Fall Event categories [83]

F01	The person falls in forward direction while walking (due to a slip)
F02	The person falls in backward direction while walking (due to a slip)
F03	The person falls sideways while walking (due to a slip)
F04	The person falls in forward direction while walking (due to a stumble)
F05	The person falls in forward direction while jogging (due to a stumble)
F06	The person falls in vertical direction while walking (due to fainting)
F07	The person falls, uses table to try to moderate the fall while walking (due to fainting)

F08	The person falls in forward direction, while attempting to rise up
F09	The person falls in a sideway direction, while attempting to rise up
F10	The person falls in forward direction, while attempting to sit down
F11	The person falls in backward direction, while attempting to sit down
F12	The person falls in a sideway direction, while attempting to sit down
F13	The person falls in a forward direction from a sitting position (due to sleeping/fainting)
F14	The person falls in a backward direction from a sitting position (due to sleeping/fainting)
F15	The person falls in a sideway direction from a sitting position (due to sleeping/fainting)

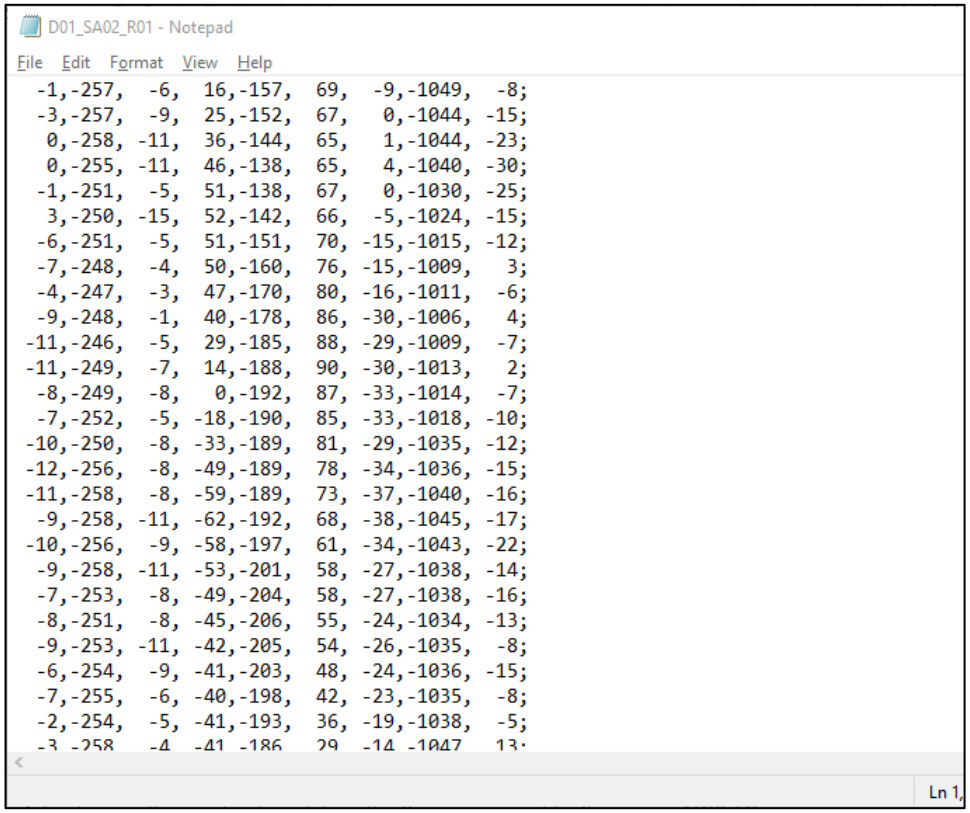


Figure 3.4: Raw data for walking slowly (D01)

3.3.2 MobiFall dataset

MobiFall dataset consists of 66 subjects with more than 3200 trials. The dataset contains 4 different types of falls, 12 different ADLs. Data were acquired via smartphone (Samsung Galaxy S3). LSM330DLC inertial module (3D accelerometer and gyroscope) used to capture the motion data. ADLs and Fall Events were categorised as shown in Table 3.4 and Table 3.5, respectively [84]. Raw data acquired for falls with front knees lying using an accelerometer is presented in Figure 3.5. The last three columns are the data reading taken from the smartphone accelerometer in the x, y and z-axis, respectively.

Table 3.4: ADL categories [84]

STD	The person stands with precise movements
WAL	The person walks regularly
JOG	The person jogs regularly
JUM	The person jumps regularly
STU	The person goes up the stairs
STN	The person goes down the stairs
SCH	The person sits on a chair
CSI	The person sits in a car from standing position
CSO	The person gets out of a car from sitting position

Table 3.5: Fall Event categories [84]

FOL	Person falls in forward direction from standing position, but moderates fall via hands
FKL	The person falls in forward direction from standing position, but knees hit ground first
BSC	The person falls in backward direction from standing position
SDL	The person falls in lateral direction from standing position

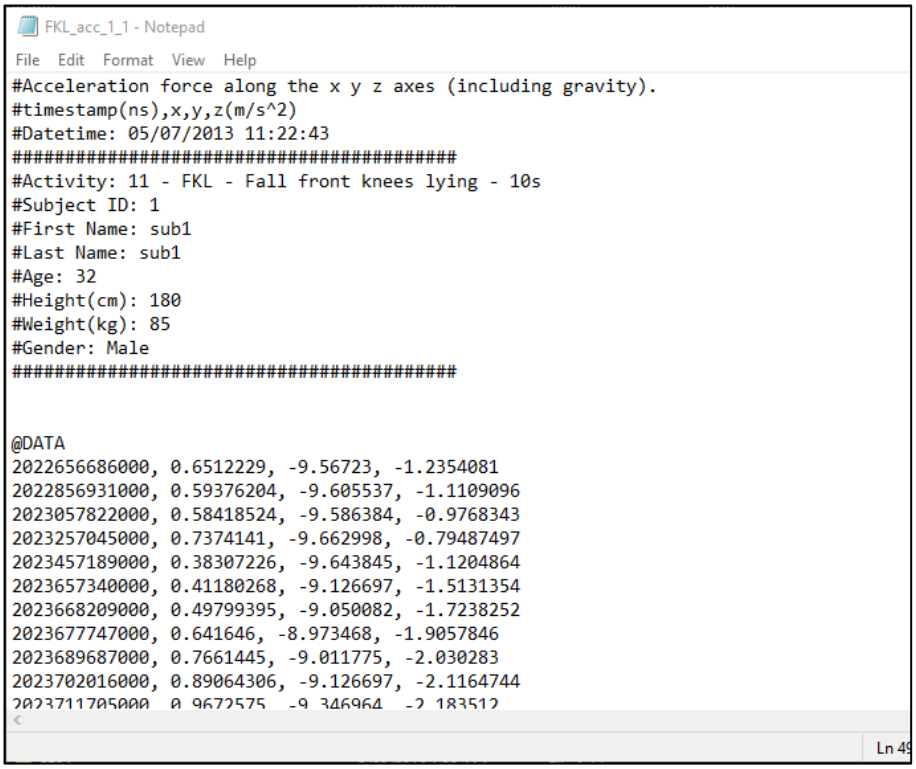


Figure 3.5: Raw data for fall with front knees lying (accelerometer)

3.4 Dataset feature selection

As described in Section 2.2.1, general features were chosen to be used for fall detection and ADL detection and classification. These features were calculated and extracted from the input data and were used to train the ML Models:

- Position Axis: The accelerometer vector in the direction of x, y and z-axis
- Maximum: Maximum Value of the Acceleration Vector in an instance of the simulated ADL/Fall Event
- Minimum: Minimum Value of the Acceleration Vector in an instance of the simulated ADL/Fall Event
- Mean: Mean Value of the Acceleration Vector in an instance of the simulated ADL/Fall Event
- Variance/Standard Deviation: The variance/dispersion of the data
- Orientation: The angle of the user/ the angle the user is facing in their current position

Accelerometer-based fall detection systems detect the signal output from starting the fall, aftermath and posture. The magnitude of the signal vector had been calculated by using Equation 21 [23]:

$$magnitude = \sqrt{(A_x)^2 + \left(A_y\right)^2 + (A_z)^2} \qquad (21)$$

The value of the magnitude increases when there are large movements. This can occur during a Fall Event, and this increase in value can be analyzed to detect a fall. Considering acceleration vector a = [a_x a_y a_z], the orientation of the user is calculated by working out the angle φ. The angle φ between the acceleration vector and the z-axis is given by Equation 22 [6], [38]:

$$\cos\varphi = \frac{a_z}{\sqrt{{a_x}^2+{a_y}^2+{a_z}^2}} \qquad (22)$$

The sensitivity/recall (capacity of the system to detect falls), specificity (capacity of the system to detect Fall Events only when they occur) and accuracy (correct differentiation between falls and non-falls) of the system were calculated with Equations 23 to 26 respectively [35], [43], [44]:

$$sensitivity/recall = \frac{TP}{TP+FN} \qquad (23)$$

$$specificity = \frac{TN}{TN+FP} \qquad (24)$$

$$precision = \frac{TP}{TP+FP} \qquad (25)$$

$$accuracy = \frac{TP+TN}{TP+FN+TN+FP} \quad (26)$$

3.5 Algorithms selected for simulation

The algorithms available in the Classification Learner App in MATLAB was used with the datasets to assess the accuracy, precision, and recall of each algorithm. Up to 24 algorithms were simulated, including Decision Trees, Discriminant Analysis, Naïve Bayes Classifiers, SVM, Gaussian SVM, KNN and available Ensemble Classifiers. The subtypes of the investigated algorithms are shown in Table 3.6.

Table 3.6: List of algorithms investigated in this study

Algorithms					
Decision Trees	**SVM**	**KNN**	**Ensemble**	**Naïve Bayes**	**Discriminant**
Fine Tree	Linear SVM	Fine KNN	Boosted Trees (AdaBoost)	Gaussian Naïve Bayes	Linear Discriminant
Medium Tree	Quadratic SVM	Medium KNN	Bagged Trees	Kernel Naïve Bayes	Quadratic Discriminant
Coarse Tree	Cubic SVM	Coarse KNN	Subspace Discriminant		
	Fine Gaussian SVM	Cosine KNN	Subspace KNN		
	Medium Gaussian SVM	Cubic KNN	RUSBoosted Trees		
	Coarse Gaussian SVM	Weighted KNN			

3.6 Cross-validation

In each set of experiments, the classification was validated using k-fold cross-validation. Cross-validation is a method for assessing ML models by training multiple ML models on the subsets of available input data and evaluating them on a complementary data subset [85].

In k-fold cross-validation, input data is split into k subsets of data (also known as folds). ML model is trained on all but one (k-1) of the subsets and then evaluated by the model on the subset that was not used for training. This method is replicated k times, each time with a separate subset reserved for analysis (and exempted from training) [85]. An

example of 5-fold cross-validation is shown in Figure 3.6. For instance, for iteration 1, Fold 1 is used as the test data to test the trained algorithm, while Folds 2 to 5 are used as the training data to train the algorithm for testing. In the next iteration, Fold 2 is used as the test data, while Folds 1, 3 to 5 are used for the training data. This process continues for five iterations in total, and the average performance is considered when processing the final accuracy of the classification.

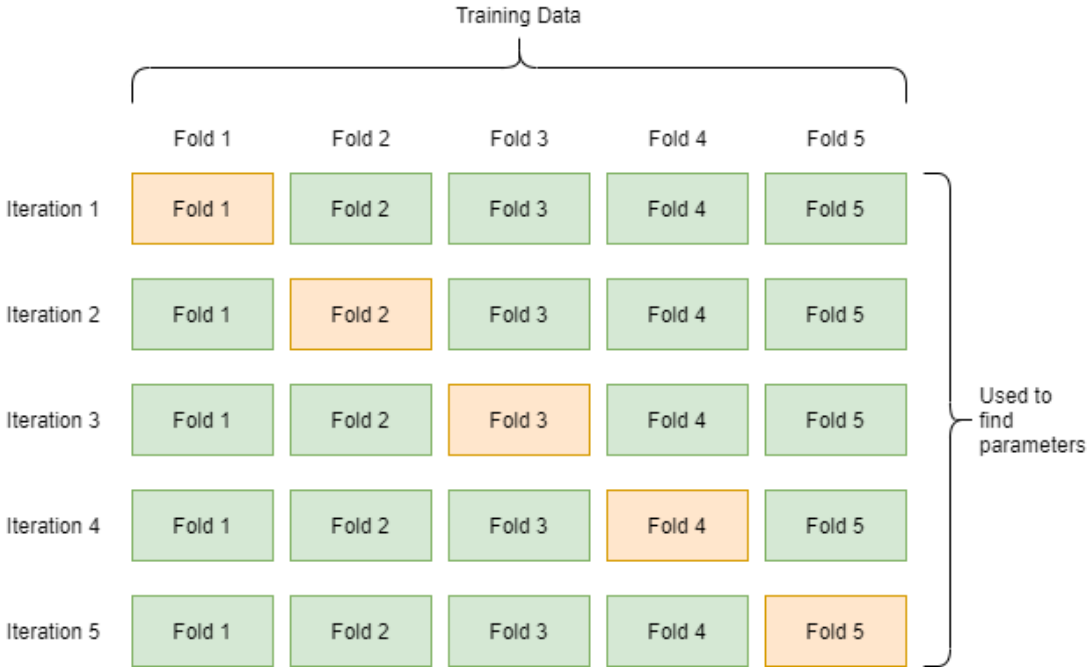


Figure 3.6: Example of 5-fold cross-validation

3.7 Experiment for algorithm simulation

The experiments were divided based on the dataset, validation type, and accelerometer used in each instance. The dataset used for the experiments is divided into a "training set" (80% of the overall data) and a "testing set" (20% of the overall data). Using the training set of data, each experiment was repeated ten times and the mean and standard deviation for the values of recall, precision, accuracy and time taken were calculated and compared. The trained classifier was then run using the testing set of data.

The simulation experiment details devices, resolution, ranges, classification numbers, validation types and features used are shown in Table 3.7. Experiments 1.1 to 1.4 used the SisFall dataset with the ADXL345 accelerometer using 5-fold cross-validation. Similarly, Experiments 2.1 to 2.4 used the same dataset and accelerometer but used 10-fold cross-validation. Experiments 3.1 to 3.4 used the same dataset as well with the

MMA8451Q accelerometer using 10-fold cross-validation. Experiments 4.1 to 4.4 used the MobiFall dataset with the LSM330DLC inertial module using 5-fold cross-validation.

Table 3.7: Simulation experiments done

No	Dataset	Device	Resolution	Range	Classification No	Validation Type	Featu-res
1.1	SisFall	Accelerometer: ADXL345	13 bits	-16g to +16g	19 (ADLs)	5-Fold Cross Validation	10
1.2	SisFall	Accelerometer: ADXL345	13 bits	-16g to +16g	19 (ADLs)	5-Fold Cross Validation	16
1.3	SisFall	Accelerometer: ADXL345	13 bits	-16g to +16g	34 (19 ADLs and 15 Falls)	5-Fold Cross Validation	16
1.4	SisFall	Accelerometer: ADXL345	13 bits	-16g to +16g	2 (ADL and Fall)	5-Fold Cross Validation	16
2.1	SisFall	Accelerometer: ADXL345	13 bits	-16g to +16g	19 (ADLs)	10-Fold Cross Validation	10
2.2	SisFall	Accelerometer: ADXL345	13 bits	-16g to +16g	19 (ADLs)	10-Fold Cross Validation	16
2.3	SisFall	Accelerometer: ADXL345	13 bits	-16g to +16g	34 (19 ADLs and 15 Falls)	10-Fold Cross Validation	16
2.4	SisFall	Accelerometer: ADXL345	13 bits	-16g to +16g	2 (ADL and Fall)	10-Fold Cross Validation	16
3.1	SisFall	Accelerometer: MMA8451Q	14 bits	−8g to +8g	19 (ADLs)	10-Fold Cross Validation	10
3.2	SisFall	Accelerometer: MMA8451Q	14 bits	−8g to +8g	19 (ADLs)	10-Fold Cross Validation	16
3.3	SisFall	Accelerometer: MMA8451Q	14 bits	−8g to +8g	34 (19 ADLs and 15 Falls)	10-Fold Cross Validation	16
3.4	SisFall	Accelerometer: MMA8451Q	14 bits	−8g to +8g	2 (ADL and Fall)	10-Fold Cross Validation	16
4.1	MobiFall	LSM330DLC inertial module (Samsung Galaxy S3)	12 bits	−16g to +16g	9 (ADLs)	5-Fold Cross Validation	10
4.2	MobiFall	LSM330DLC inertial module (Samsung Galaxy S3)	12 bits	−16g to +16g	9 (ADLs)	5-Fold Cross Validation	16
4.3	MobiFall	LSM330DLC inertial module (Samsung Galaxy S3)	12 bits	−16g to +16g	13 (9 ADLs and 4 Falls)	5-Fold Cross Validation	16

4.4	MobiFall	LSM330DLC inertial module (Samsung Galaxy S3)	12 bits	−16g to +16g	2 (ADL and Fall)	5-Fold Cross Validation	16

3.8 Dataset extraction for training

The required features used for classification were extracted from the SisFall and Mobifall dataset. Figure 3.7 shows the data for ADXL345 accelerometer being extracted. The first step includes data files being initialised for reading. The text files were then read and transferred into arrays (as shown in Figure 3.8). The mean, maximum, minimum, average, orientation and standard deviation were extracted and calculated for each ADL and Fall and tabulated together.

An example of variables extracted for the ADL D01 in a table and tabulating all data together for ADXL345 is presented in Figure 3.9 and Figure 3.10, respectively.

```
folder = 'D:\Users\faras\D-Documents\UTS\Datasets\New folder\SisFall_dataset_Test\Train\D01/';

% Read the accelerometer data files
files = dir([folder,'*.txt']);
num_files = length(files);
dataFiles = zeros(1,num_files);
```

Figure 3.7: An instance of text files for D01 to be read being initialised

```
dataFiles(i) = fopen([folder files(i).name],'r');
data = fscanf(dataFiles(i),'%d\t,%d\t,%d\t,%d\t,%d\t,%d\t,%d\t,%d\t,%d;\n',[9,inf]);
noisy_x(1,:) = ((2*156.96)/(2^13))*data(1,:);
noisy_y(1,:) = ((2*156.96)/(2^13))*data(2,:);
noisy_z(1,:) = ((2*156.96)/(2^13))*data(3,:);
```

Figure 3.8: Code to read data from text

```
table_D01 = table(Mean_x, Mean_y, Mean_z, Max_x, Max_y, Max_z, Min_x,...
    Min_y, Min_z, std_x, std_y, std_z, AvgAcc, cos_AvgAcc_x, cos_AvgAcc_y, cos_AvgAcc_z, Catl, Cat_Bl);
```

Figure 3.9: An instance of variables extracted for the ADL D01 in a table

```
%%
ADL_80_per_ADXL345_table = [table_D01; table_D02; table_D03; table_D04;...
    table_D05; table_D06; table_D07; table_D08; table_D09; table_D10;...
    table_D11; table_D12; table_D13; table_D14; table_D15; table_D16;...
    table_D17; table_D18; table_D19];

%%
save('extracted_80_per_ADL_ADXL345_dataset.mat','table_D01','table_D02',...
'table_D03','table_D04','table_D05','table_D06','table_D07','table_D08',...
'table_D09','table_D10','table_D11','table_D12','table_D13','table_D14',...
'table_D15','table_D16','table_D17','table_D18','table_D19',...
'ADL_80_per_ADXL345_table');
```

Figure 3.10: Tabulating all data together for ADXL345

Similarly, the data for MMA8451Q accelerometer and MobiFall dataset was extracted, and accelerometer data files were read using the set of instructions shown in Figure 3.11. The mean, maximum, minimum, average, orientation and standard deviation was extracted and calculated for each ADL and Fall and tabulated together. This set of data was stored separately from data from the ADXL345 accelerometer. Figure 3.12 shows the code to save the data for further processing.

```
dataFiles(i) = fopen([folder files(i).name],'r');
data = fscanf(dataFiles(i),'%d\t,%d\t,%d\t,%d\t,%d\t,%d\t,%d\t,%d\t,%d;\n',[9,inf]);
noisy_x = ((2*9.81*8)/(2^14))*data(7,:);
noisy_y = ((2*9.81*8)/(2^14))*data(8,:);
noisy_z = ((2*9.81*8)/(2^14))*data(9,:);
```

Figure 3.11: Code to read data from text

```
%%

ADL_80_per_MMA8451Q_table = [A1{1};A1{2};A1{3};A1{4};A1{5};A1{6};A1{7};A1{8};...
    A1{9};A1{10};A1{11};A1{12};A1{13};A1{14};A1{15};A1{16};A1{17};A1{18};A1{19}];

%%

save('extracted_80_per_ADL_MMA8451Q_dataset.mat', 'A1', 'ADL_80_per_MMA8451Q_table');
```

Figure 3.12: Tabulating all data together for MMA8451Q

3.8.1 Training data classification

3.8.1.1 Initiating simulation algorithm

The classifier algorithm was initiated via the Classification Learner App in MATLAB. The generated classifier algorithm was then modified to run ten times, and each iteration was saved for further processing. Figure 3.13 shows the code for data to be extracted and classified. This section of the code stayed consistent for all ML algorithms in these simulations.

```
% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'Mean_x', 'Mean_y', 'Mean_z', 'Max_x', 'Max_y', 'Max_z', 'Min_x', 'Min_y', 'Min_z', 'AvgAcc'};
predictors = inputTable(:, predictorNames);
response = inputTable.Cat1;
```

Figure 3.13: Initialise predictors and response for classification

3.8.1.2 Setting up classifiers

3.8.1.2.1 Decision Tree

Figure 3.14 shows the setting on which the Decision Tree classifier for the particular instance was set up. The code shown sets up the Fine Tree classifier. Similarly, Medium Tree classifier has 'MaxNumSplits' variable set to 20, and the Coarse Tree classifier has 'MaxNumSplits' variable set to 4.

The predict function was set up with the trained model being "trainedClassifier". This was used in other test data to measure the accuracy, precision and recall of the trained model (Figure 3.14 and Figure 3.15). After training the classification model, the results were cross-validated as shown in Figure 3.16, with the overall accuracy of the model being calculated. This process was repeated for each iteration of the code.

```
% Train a classifier
% This code specifies all the classifier options and trains the classifier.
classificationTree = fitctree(...
    predictors, ...
    response, ...
    'SplitCriterion', 'gdi', ...
    'MaxNumSplits', 100, ...
    'Surrogate', 'off', ...
    'ClassNames', categorical({'D01'; 'D02'; 'D03'; 'D04'; 'D05'; 'D06';...
    'D07'; 'D08'; 'D09'; 'D10'; 'D11'; 'D12'; 'D13'; 'D14'; 'D15'; 'D16'; 'D17'; 'D18'; 'D19'}));
```

Figure 3.14: Setting up classification type and properties

```
% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
treePredictFcn = @(x) predict(classificationTree, x);
trainedClassifier.predictFcn = @(x) treePredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.RequiredVariables = {'AvgAcc', 'Max_x', 'Max_y',...
    'Max_z', 'Mean_x', 'Mean_y', 'Mean_z', 'Min_x', 'Min_y', 'Min_z'};
trainedClassifier.ClassificationTree = classificationTree;
```

Figure 3.15: Setting up the predict function

```
% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationTree, 'KFold', 5);

% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
```

Figure 3.16: An instance of k-fold cross-validation

3.8.1.2.2 Naïve Bayes

Figure 3.17 shows the setting on which the Naïve Bayes classifier for the particular instance was set up. The code shown sets up the Gaussian Naïve Bayes classifier. The numerical predictors as part of the 'repmat' function were set to 'Normal'. Similarly, Kernel Naïve Bayes classifier has the numerical predictors as part of the 'repmat' function was set to 'Kernel'.

Similar to Figure 3.15, the predict function was set up with the trained model being "trainedClassifier" in other test data to measure the accuracy, precision and recall of the trained model. After training the classification model, the results were cross-validated as shown in Figure 3.16, with the overall accuracy of the model being calculated. This process was repeated each iteration of the code.

```
distributionNames =  repmat({'Normal'}, 1, length(isCategoricalPredictor));
distributionNames(isCategoricalPredictor) = {'mvmn'};

if any(strcmp(distributionNames,'Kernel'))
    classificationNaiveBayes = fitcnb(...
        predictors, ...
        response, ...
        'Kernel', 'Normal', ...
        'Support', 'Unbounded', ...
        'DistributionNames', distributionNames, ...
        'ClassNames', categorical({'D01'; 'D02'; 'D03'; 'D04'; 'D05';...
        'D06'; 'D07'; 'D08'; 'D09'; 'D10'; 'D11'; 'D12'; 'D13'; 'D14'; 'D15'; 'D16'; 'D17'; 'D18'; 'D19'}));
else
    classificationNaiveBayes = fitcnb(...
        predictors, ...
        response, ...
        'DistributionNames', distributionNames, ...
        'ClassNames', categorical({'D01'; 'D02'; 'D03'; 'D04'; 'D05';...
        'D06'; 'D07'; 'D08'; 'D09'; 'D10'; 'D11'; 'D12'; 'D13'; 'D14'; 'D15'; 'D16'; 'D17'; 'D18'; 'D19'}));
end
```

Figure 3.17: Setting up classification type and properties

3.8.1.2.3 SVM

Figure 3.18 shows the setting on which the SVM classifier for a particular instance was set up. The code shown sets up the linear SVM classifier. The hyperparameters for 'KernelFunction' was set to 'linear''. Similarly, the hyperparameters for 'KernelFunction' was set to 'polynomial' with 'PolynomialOrder' set to 2 for Quadratic SVM (Figure 3.19). Cubic SVM classifier for 'KernelFunction' was set to 'polynomial' with 'PolynomialOrder' set to 3.

Fine Gaussian SVM required the hyperparameters for 'KernelFunction' set to 'gaussian' with 'KernelScale' set to 0.79, as seen in Figure 3.20. Similarly, Medium Gaussian SVM and Course Gaussian SVM had the hyperparameters for 'KernelFunction' set to 'gaussian' with 'KernelScale' set to 3.2 and 13, respectively.

As was shown in Figure 3.15, the predict function was similarly set up with the trained model being “trainedClassifier” in other test data to measure the accuracy, precision and recall of the trained model. After training the classification model, the results were cross-validated, with the overall accuracy of the model being calculated. This process was repeated each iteration of the code.

```
template = templateSVM(...
    'KernelFunction', 'linear', ...
    'PolynomialOrder', [], ...
    'KernelScale', 'auto', ...
    'BoxConstraint', 1, ...
    'Standardize', true);
classificationSVM = fitcecoc(...
    predictors, ...
    response, ...
    'Learners', template, ...
    'Coding', 'onevsone', ...
    'ClassNames', categorical({'D01'; 'D02'; 'D03'; 'D04'; 'D05'; 'D06'; 'D07';...
    'D08'; 'D09'; 'D10'; 'D11'; 'D12'; 'D13'; 'D14'; 'D15'; 'D16'; 'D17'; 'D18'; 'D19'}));
```

Figure 3.18: Setting up Linear SVM classification type and properties

```
template = templateSVM(...
    'KernelFunction', 'polynomial', ...
    'PolynomialOrder', 2, ...
    'KernelScale', 'auto', ...
    'BoxConstraint', 1, ...
    'Standardize', true);
```

Figure 3.19: Setting up Quadratic SVM classification type and properties

```
template = templateSVM(...
    'KernelFunction', 'gaussian', ...
    'PolynomialOrder', [], ...
    'KernelScale', 0.79, ...
    'BoxConstraint', 1, ...
    'Standardize', true);
```

Figure 3.20: Setting up Fine Gaussian SVM classification type and properties

3.8.1.2.4 KNN

Figure 3.21 shows the setting on which the KNN classifier for a particular instance was set up. The code shown sets up the linear SVM classifier. The hyperparameters were adjusted depending on the distance and number of nearest neighbours. For Fine KNN, the ‘Distance’ was set to ‘Euclidean’ and ‘NumNeighbors’ was set to 1. Similarly, Medium KNN and Course KNN required the ‘Distance’ to be set to ‘Euclidean’ with ‘NumNeighbors’ set to 10 and 100, respectively. Additionally, Weighted KNN had the

'NumNeighbors' set to 10 and the distance weight function 'DistanceWeight' set to 'squaredinverse'.

Similarly, Cosine KNN required the hyperparameters for 'Distance' set to 'Cosine' with 'NumNeighbors' set to 10 (Figure 3.22). Cubic KNN had the hyperparameters for 'Distance' set to 'Minkowski' with 'Exponent' set to 3 (Figure 3.23).

The predict function was set up with the trained model being "trainedClassifier" similar to Figure 3.15. After training the classification model, the results were cross-validated as shown in Figure 3.16, with the overall accuracy of the model being calculated. This process was repeated each iteration of the code.

```
classificationKNN = fitcknn(...
    predictors, ...
    response, ...
    'Distance', 'Euclidean', ...
    'Exponent', [], ...
    'NumNeighbors', 1, ...
    'DistanceWeight', 'Equal', ...
    'Standardize', true, ...
    'ClassNames', categorical({'D01'; 'D02'; 'D03'; 'D04'; 'D05'; 'D06'; 'D07';...
    'D08'; 'D09'; 'D10'; 'D11'; 'D12'; 'D13'; 'D14'; 'D15'; 'D16'; 'D17'; 'D18'; 'D19'}));
```

Figure 3.21: Setting up Fine KNN classification type and properties

```
classificationKNN = fitcknn(...
    predictors, ...
    response, ...
    'Distance', 'Cosine', ...
    'Exponent', [], ...
    'NumNeighbors', 10, ...
    'DistanceWeight', 'Equal', ...
    'Standardize', true, ...
    'ClassNames', categorical({'D01'; 'D02'; 'D03'; 'D04'; 'D05'; 'D06'; 'D07';...
    'D08'; 'D09'; 'D10'; 'D11'; 'D12'; 'D13'; 'D14'; 'D15'; 'D16'; 'D17'; 'D18'; 'D19'}));
```

Figure 3.22: Setting up Cosine KNN classification type and properties

```
classificationKNN = fitcknn(...
    predictors, ...
    response, ...
    'Distance', 'Minkowski', ...
    'Exponent', 3, ...
    'NumNeighbors', 10, ...
    'DistanceWeight', 'Equal', ...
    'Standardize', true, ...
    'ClassNames', categorical({'D01'; 'D02'; 'D03'; 'D04'; 'D05'; 'D06'; 'D07';...
    'D08'; 'D09'; 'D10'; 'D11'; 'D12'; 'D13'; 'D14'; 'D15'; 'D16'; 'D17'; 'D18'; 'D19'}));
```

Figure 3.23: Setting up Cubic KNN classification type and properties

3.8.2 Saving results

After running the experiment, the relevant results and the trained classifier was saved for further processing and comparison. These results include the overall accuracy, the time elapsed during the simulation, the predicted classes for the simulation and the Actual classes that were simulated. An instance of this code is shown in Figure 3.24. The saved results were then extracted, the false positive, false negative, recall, precision and specificity for each activity was calculated and tabulated for further comparison and analysis. An instance of the tabulated results for Experiment 1.1 is shown in Figure 3.25.

```
elapsedTime = toc;

Predict_Cat = validationPredictions;
Actual_Cat = categorical(ADL_80_per_ADXL345_table.Cat1);

newStr = join(["Experiment 1/Exp_1_Ex_1_",i,".mat"],'');
Saved_Mat = convertStringsToChars(newStr);

save(Saved_Mat,'Predict_Cat','Actual_Cat','trainedClassifier','validationAccuracy','elapsedTime');
clearvars -except trainingData ADL_80_per_ADXL345_table
```

Figure 3.24: Saved results of the simulation

C_End

PLOTS | VARIABLE | VIEW

New from Selection | Open | Print | Rows: 1 | Columns: [] | Insert | Delete | Transpose | Sort

VARIABLE | SELECTION | EDIT

24x7 cell

	1	2	3	4	5	6	7
1	"Fine Tree"	"60.51±0.97"	"60.98±1"	"98.19±0.06"	"0.605±0.01"	"67.68±1.05"	"0.44±0.812"
2	"Medium Tree"	"49.49±0.48"	"47.07±1.51"	"97.71±0.03"	"0.466±0.006"	"59.29±0.61"	"0.123±0.005"
3	"Coarse Tree"	"24.52±0.24"	"15.34±0.65"	"96.04±0.02"	"0.174±0.007"	"29.55±0.29"	"0.105±0.005"
4	"Linear Discriminant"	"57.41±0.55"	"59.11±0.84"	"97.98±0.03"	"0.573±0.006"	"64.03±0.47"	"0.191±0.24"
5	"Quadratic Discriminant"	"64.32±0.71"	"70.76±1.71"	"98.49±0.03"	"0.651±0.01"	"73.06±0.58"	"0.181±0.044"
6	"Gaussian Naïve Bayes"	"58.35±0.84"	"59.26±0.67"	"97.87±0.02"	"0.579±0.007"	"62±0.43"	"0.296±0.131"
7	"Kernel Naïve Bayes"	"64.26±0.78"	"64.97±0.61"	"98.28±0.03"	"0.636±0.006"	"69.29±0.57"	"3.753±0.413"
8	"Linear SVM"	"67.29±0.64"	"73.08±2.6"	"98.54±0.02"	"0.682±0.009"	"73.97±0.42"	"6.456±0.598"
9	"Quadratic SVM"	"74.6±0.86"	"76.35±0.98"	"98.87±0.04"	"0.752±0.009"	"79.8±0.76"	"6.522±0.139"
10	"Cubic SVM"	"75.59±0.69"	"77.64±1.15"	"98.95±0.03"	"0.763±0.009"	"81.27±0.51"	"6.542±0.047"
11	"Fine Gaussian SVM"	"61.86±0.45"	"66±0.3"	"98.57±0.03"	"0.623±0.004"	"74.49±0.54"	"7.029±0.298"
12	"Medium Gaussian SVM"	"69.04±0.61"	"73.14±2.74"	"98.77±0.03"	"0.694±0.007"	"78.06±0.57"	"6.27±0.148"
13	"Coarse Gaussian SVM"	"55.06±0.45"	"53.8±0.46"	"98.11±0.03"	"0.536±0.004"	"66.3±0.54"	"6.091±0.064"
14	"Fine KNN"	"76.53±1.08"	"79.04±1.25"	"99.09±0.04"	"0.773±0.012"	"83.76±0.64"	"0.184±0.235"
15	"Medium KNN"	"63.19±0.99"	"65.23±2.05"	"98.41±0.05"	"0.631±0.012"	"71.7±0.95"	"0.117±0.014"
16	"Coarse KNN"	"45.71±0.53"	"44.86±0.65"	"97.47±0.04"	"0.432±0.006"	"54.97±0.64"	"0.131±0.011"
17	"Cosine KNN"	"60.75±0.72"	"63.26±1.7"	"98.31±0.04"	"0.606±0.009"	"69.95±0.64"	"0.17±0.199"
18	"Cubic KNN"	"61.59±0.87"	"65.38±1.08"	"98.32±0.05"	"0.616±0.009"	"70.09±0.83"	"0.195±0.015"
19	"Weighted KNN"	"73.12±0.98"	"77.4±1.95"	"99.04±0.05"	"0.736±0.012"	"82.97±0.84"	"0.118±0.014"
20	"Boosted Trees (AdaBoo...	"54.33±0.68"	"55.59±2.21"	"98.02±0.04"	"0.531±0.009"	"64.79±0.65"	"2.669±0.219"
21	"Bagged Trees"	"69.85±0.55"	"73.01±0.9"	"98.76±0.02"	"0.703±0.006"	"77.87±0.41"	"2.226±0.127"
22	"Subspace Discriminant"	"53.21±0.57"	"51.8±1.89"	"97.81±0.03"	"0.517±0.005"	"61.05±0.51"	"1.838±0.162"
23	"Subspace KNN"	"72.42±0.92"	"75.97±1.42"	"98.9±0.04"	"0.731±0.01"	"80.35±0.72"	"1.879±0.03"
24	"RUSBoosted Trees"	"64.26±1.24"	"60.76±0.74"	"97.99±0.03"	"0.598±0.007"	"63.79±0.6"	"2.325±0.026"

Figure 3.25: An instance of the tabulated results for Experiment 1.1

3.9 Dataset extraction for testing

The required data used for classification during testing were extracted from the SisFall and Mobifall dataset. Figure 3.26 shows the data for ADL activity D01 of the ADXL345 accelerometer being extracted. Similar to extracting the training data, the first step includes data files being initialised for reading. The text files were then read and transferred into arrays (as shown in Figure 3.27). The mean, maximum, minimum, average, orientation and standard deviation were extracted and calculated for each ADL and Fall and tabulated together (shown in Figure 3.28 and Figure 3.29). An example of variables extracted for the ADL D01 in a table and tabulating all data together for ADXL345 is presented in Figure 3.30 and Figure 3.31, respectively.

```
%D01----------------------------------------------------------------------

folder = 'D:\Users\faras\D-Documents\UTS\Datasets\New folder\SisFall_dataset_Test\Train\Test_Set\D01/';

% READ THE ACCELEROMETER DATA FILES
files = dir([folder,'*.txt']);
numFiles = length(files);
dataFiles = zeros(1,numFiles);
```

Figure 3.26: An instance of text files for D01 to be read

```
for i=1:1:numFiles
    dataFiles(i) = fopen([folder files(i).name],'r');
    data = fscanf(dataFiles(i),'%d\t,%d\t,%d\t,%d\t,%d\t,%d\t,%d\t,%d\t,%d;\n',[9,inf]);
    noisy_x(1,:) = ((2*156.96)/(2^13))*data(1,:);
    noisy_y(1,:) = ((2*156.96)/(2^13))*data(2,:);
    noisy_z(1,:) = ((2*156.96)/(2^13))*data(3,:);
```

Figure 3.27: Code to read data from text

```
Mean_x(i,:) = mean(noisy_x,2);
Mean_y(i,:) = mean(noisy_y,2);
Mean_z(i,:) = mean(noisy_z,2);

Max_x(i,:)= max(noisy_x,[],2);
Max_y(i,:)= max(noisy_y,[],2);
Max_z(i,:)= max(noisy_z,[],2);

Min_x(i,:)= min(noisy_x,[],2);
Min_y(i,:)= min(noisy_y,[],2);
Min_z(i,:)= min(noisy_z,[],2);
```

Figure 3.28: The mean, maximum, minimum extracted for the ADL D01 in an array

```
if Mean_x(i,:) < 0
cos_AvgAcc_x(i,:) = -acosd(-Mean_x(i,:)/AvgAcc(i,:));
else
cos_AvgAcc_x(i,:) = acosd(Mean_x(i,:)/AvgAcc(i,:));
end

if Mean_y(i,:) < 0
cos_AvgAcc_y(i,:) = -acosd(-Mean_y(i,:)/AvgAcc(i,:));
else
cos_AvgAcc_y(i,:) = acosd(Mean_y(i,:)/AvgAcc(i,:));
end

if Mean_z(i,:) < 0
cos_AvgAcc_z(i,:) = -acosd(-Mean_z(i,:)/AvgAcc(i,:));
else
cos_AvgAcc_z(i,:) = acosd(Mean_z(i,:)/AvgAcc(i,:));
end

std_x(i,:) = std(noisy_x,[],2);
std_y(i,:) = std(noisy_y,[],2);
std_z(i,:) = std(noisy_z,[],2);
```

Figure 3.29: Calculating the orientation of the signal and standard deviation

```
Cat1 = categorical (cellstr(Cat));
Cat_B1 = categorical (cellstr(Cat_B));
table_D01 = table(Mean_x, Mean_y, Mean_z, Max_x, Max_y, Max_z, Min_x,...
    Min_y, Min_z, std_x, std_y, std_z, AvgAcc, cos_AvgAcc_x, cos_AvgAcc_y, cos_AvgAcc_z, Cat1, Cat_B1);
```

Figure 3.30: Variables extracted for the ADL D01 in a table

```
ADL_20_per_ADXL345_table = [table_D01; table_D02; table_D03; table_D04;...
    table_D05; table_D06; table_D07; table_D08; table_D09; table_D10;...
    table_D11; table_D12; table_D13; table_D14; table_D15; table_D16;...
    table_D17; table_D18; table_D19];

%%
save('extracted_20_per_ADL_ADXL345_dataset.mat','table_D01','table_D02',...
'table_D03','table_D04','table_D05','table_D06','table_D07','table_D08',...
'table_D09','table_D10','table_D11','table_D12','table_D13','table_D14',...
'table_D15','table_D16','table_D17','table_D18','table_D19',...
'ADL_20_per_ADXL345_table');
```

Figure 3.31: Tabulating all data together for ADXL345

Similar to extracting the training data for the MMA8451Q accelerometer and MobiFall dataset, accelerometer data files were read using the set of instructions similar to that shown in Figure 3.32 and Figure 3.33. The mean, maximum, minimum, average,

orientation and standard deviation was extracted and calculated for each ADL and Falls, and tabulated together. Figure 3.34 shows the code to save the data for further processing.

```
for no = 1:1:numArrays

    if no < 10

        newStr = join(["D:\Users\faras\D-Documents\UTS\Datasets\New folder\SisFall_dataset_Test\Train\Test_Set\D0",no,"/"],'');
        folder = convertStringsToChars(newStr);

    else
        newStr = join(["D:\Users\faras\D-Documents\UTS\Datasets\New folder\SisFall_dataset_Test\Train\Test_Set\D",no,"/"],'');
        folder = convertStringsToChars(newStr);
    end

    % READ THE ACCELEROMETER DATA FILES
    files = dir([folder,'*.txt']);
    numFiles = length(files);
    dataFiles = zeros(1,numFiles);
```

Figure 3.32: An instance of text files to be read

```
for i=1:1:numFiles
    dataFiles(i) = fopen([folder files(i).name],'r');
    data = fscanf(dataFiles(i),'%d\t,%d\t,%d\t,%d\t,%d\t,%d\t,%d\t,%d\t,%d;\n',[9,inf]);
    noisy_x = ((2*9.81*8)/(2^14))*data(7,:);
    noisy_y = ((2*9.81*8)/(2^14))*data(8,:);
    noisy_z = ((2*9.81*8)/(2^14))*data(9,:);
```

Figure 3.33: Code to read data from text for MMA8451Q

```
ADL_20_per_MMA8451Q_table = [A1{1};A1{2};A1{3};A1{4};A1{5};A1{6};A1{7};A1{8};...
    A1{9};A1{10};A1{11};A1{12};A1{13};A1{14};A1{15};A1{16};A1{17};A1{18};A1{19}];

%%

save('extracted_20_per_ADL_MMA8451Q_dataset.mat', 'A1', 'ADL_20_per_MMA8451Q_table');
```

Figure 3.34: Tabulating all data together for MMA8451Q

3.9.1 Testing data classification

3.9.1.1 Initiating simulation algorithm for testing

The classifier algorithm was previously initiated via the Classification Learner App in MATLAB. The generated classifier algorithm was then modified to run ten times, and each iteration was saved for further processing. The best three most accurate classifiers are then tested via the “testing set” of data. Figure 3.35 shows the code for data for classifying test set of data for Experiment 2.4 (a binary classification).

```
Predict_Cat = trainedClassifier.predictFcn(ADLFall_20_per_test_ADXL345_table);
Actual_Cat = categorical(ADLFall_20_per_test_ADXL345_table.Cat_B1);

%confusion matrix struct
[cm, order] = confusionmat(Actual_Cat,Predict_Cat)
[m,n] = size(cm);
```

Figure 3.35: Test set data used for the trained classifier

3.9.1.2 Extracting the results

Figure 3.36 shows part of the code used to extract the results of the tested trained classifier for further processing. As can be seen in Figure 3.36, the values of recall, precision and specificity are calculated by extracting the true-positive (TP), true-negative (TN), false-positive (FP) and false-negative values (FN) from the confusion matrix (cm). This process was repeated in each iteration of the code.

```
for i=1:1:m
    for j=1:1:n
        total_obs(no,:) = total_obs(no,:) + cm(i,j);
        if i == no
            FN(no,:) = FN(no,:) + cm(i,j);
        end
        if j == no
            FP(no,:) = FP(no,:) + cm(i,j);
        end
    end
end

FP(no,:) = FP(no,:) - TP(no,:);
FN(no,:) = FN(no,:) - TP(no,:);
TN(no,:) = total_obs(no,:) - cm(no,no) - FP(no,:) - FN(no,:);


recall(no,:) = TP(no,:)/(TP(no,:) + FN(no,:));
precision(no,:) = TP(no,:)/(TP(no,:) + FP(no,:));
specificity(no,:) = TN(no,:)/(TN(no,:) + FP(no,:));
```

Figure 3.36: Part of code for calculating recall, precision and specificity

3.9.2 Saving results

After running the experiment, the relevant results were saved for further processing and comparison. These results include the overall accuracy, the time elapsed during the simulation, the predicted classes for the simulation and the actual classes that were simulated. An instance of this code is shown in Figure 3.37. The saved results can then be extracted and tabulated for further comparison and analysis. An instance of the tabulated results for Experiment 1.1 is shown in Figure 3.38.

```
save (Saved_Mat, 'accuracy', 'Predict_Cat','Actual_Cat','trainedClassifier','validationAccuracy','elapsedTime',...
    'cm', 'order', 'm', 'n', 'TP', 'TN', 'FP',...
    'FN', 'recall', 'precision', 'specificity', 'total_obs', 'F1_score',...
    'macro_av_recall', 'macro_av_precision', 'macro_av_specificity', 'macro_av_F1_score');
```

Figure 3.37: Code for saving the results of the simulation

```
C_End{meh,2} = join([round(total_recall,2),"±",round(std_array(1),2)],'');
C_End{meh,3} = join([round(total_precision,2),"±",round(std_array(2),2)],'');
C_End{meh,4} = join([round(total_specificity,3),"±",round(std_array(3),3)],'');
C_End{meh,5} = join([round(total_F1_score,3),"±",round(std_array(4),3)],'');
C_End{meh,6} = join([round(total_accuracy,2),"±",round(std_array(5),2)],'');
C_End{meh,7} = join([round(total_elapsedTime,3),"±",round(std_array(6),3)],'');
```

Figure 3.38: Code for tabulated results of the simulation

3.10 Ensemble method

The ensemble method is used as a means to combine several ML models by training an additional model. This is referred to as a stacked ensemble model, or stacked learner, as depicted in Figure 3.39. Base models for this method were trained, tested and compared to create a stacked ensemble classifier. The base model code is shown in Figure 3.40 to 3.43. The classifier algorithm was initiated as shown in Figure 3.41, which shows the setting on which the SVM classifier for a particular instance was set up. The code shown sets up the Gaussian SVM classifier. This type of SVM required the hyperparameters for 'KernelFunction' to be set to 'gaussian' with 'KernelScale' set to 4. Cubic SVM classifier for 'KernelFunction' was set to 'polynomial' with 'PolynomialOrder' set to 3. Similarly, the hyperparameters for 'KernelFunction' was set to 'polynomial' with 'PolynomialOrder' set to 2 for Quadratic SVM.

Figure 3.42 shows the setting on which the KNN classifier for a particular instance was set up. For Fine KNN, the 'Distance' was set to 'Euclidean', and 'NumNeighbors' was set to 1. Figure 3.42 also shows the setting in which the Decision Tree classifier for the particular instance was set up. The code shown sets up the Fine Tree classifier. Similarly, Fine Tree classifier has 'MaxNumSplits' variable set to 100.

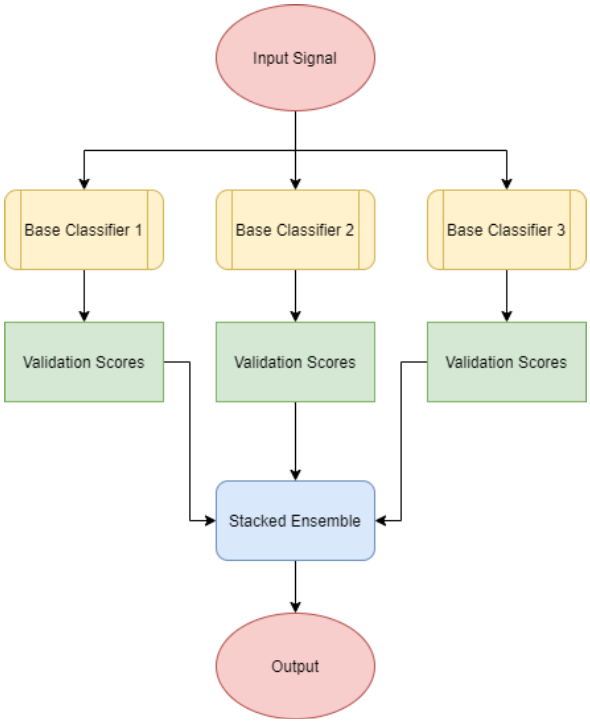


Figure 3.39: Stacked Ensemble Classifier

```
tic
inputTable = trainingData;
predictorNames = {'Mean_x', 'Mean_y', 'Mean_z', 'Max_x', 'Max_y', 'Max_z',...
    'Min_x', 'Min_y', 'Min_z', 'std_x', 'std_y', 'std_z', 'AvgAcc',...
    'cos_AvgAcc_x', 'cos_AvgAcc_y', 'cos_AvgAcc_z'};

predictors = inputTable(:, predictorNames);
response = inputTable.Cat_B1;
```

Figure 3.40: Part of base classifier code (I)

```
% SVM with Gaussian kernel
rng('default') % For reproducibility
mdls{1} = fitcsvm(...
    predictors, ...
    response, ...
    'KernelFunction', 'gaussian', ...
    'PolynomialOrder', [], ...
    'KernelScale', 4, ...
    'BoxConstraint', 1, ...
    'Standardize', true, ...
    'ClassNames', categorical({'ADL'; 'Fall'}));

% SVM with polynomial kernel
rng('default')
mdls{2} = fitcsvm(predictors,response,'KernelFunction','polynomial','PolynomialOrder', 3, ...
    'Standardize',true,'KernelScale','auto','BoxConstraint', 1);

rng('default')
mdls{3} = fitcsvm(predictors,response,'KernelFunction','polynomial','PolynomialOrder', 2, ...
    'Standardize',true,'KernelScale','auto','BoxConstraint', 1);
```

Figure 3.41: Part of base classifier code (II)

```
%KNN
rng('default')
mdls{4} = fitcknn(...
    predictors, ...
    response, ...
    'Distance', 'Euclidean', ...
    'Exponent', [], ...
    'NumNeighbors', 1, ...
    'DistanceWeight', 'Equal', ...
    'Standardize', true);

% % Decision tree
rng('default')
mdls{5} = fitctree(...
    predictors, ...
    response, ...
    'SplitCriterion', 'gdi', ...
    'MaxNumSplits', 100, ...
    'Surrogate', 'off', ...
    'ClassNames', categorical({'ADL'; 'Fall'}));
```

Figure 3.42: Part of base classifier code (III)

```
% Naive Bayes
rng('default')
mdls{6} = fitcnb(trainingData,'Cat_Bl');
%
```

Figure 3.43: Part of base classifier code (IV)

Input is trained via the k-fold cross-validated predictions of the base models, as to reduce overfitting. A 'cvpartition' object was created and transferred to a 'crossval' function of each model (Figure 3.44). The stacked ensemble classifier was trained based on the variations found on the results from the different algorithms and improves the accuracy upon them (Figure 3.45). As shown in Figure 3.46, after being trained, the classifier and base models were tested for comparison.

```
N = numel(class);
Scores = zeros(size(trainingData,1),N);
cv = cvpartition(trainingData.Cat_Bl,"KFold",5);
for ii = 1:N
    val_c = crossval(class{ii},'cvpartition',cv);
    [~,s] = kfoldPredict(val_c);
    Scores(:,ii) = s(:,val_c.ClassNames=='Fall');
end
```

Figure 3.44: K-fold cross-validation scores

```
t = templateTree();
stacked_class = fitcensemble(Scores,trainingData.Cat_Bl,'Method','Bag','Learners',t)
```

Figure 3.45: Stacked Ensemble initialization

```
%Iterate over the base models to the compute predicted labels, scores, and loss values
label = [];
score = zeros(size(testingData,1),N);
mdlLoss = zeros(1,numel(class));
for i = 1:N
    [lbl,s] = predict(class{i},testingData);
    label = [label,lbl];
    score(:,i) = s(:,val_c.ClassNames=='Fall');
end

%Attach the predictions from the stacked ensemble to label and stacked_class
[lbl,s] = predict(stacked_class,score);
label = [label,lbl];

%Concatenate the score of the stacked ensemble to the scores of the base models
score = [score,s(:,1)];

name = {'SVM-Gaussian','SVM-Cubic','SVM-Quadratic','KNN','Decision tree','Naive Bayes','Stacked'};
```

Figure 3.46: Classifier Testing

3.11 Visualizing results with confusion matrix

The recall and precision results of each classifier simulated in the experiments can be visualised by using a confusion matrix. This is a table that is used to describe the performance of a classification on a set of test data for which the true values are known. As seen in the Figure 3.47, the rows correspond to the true class, while the columns correspond to the predicted class.

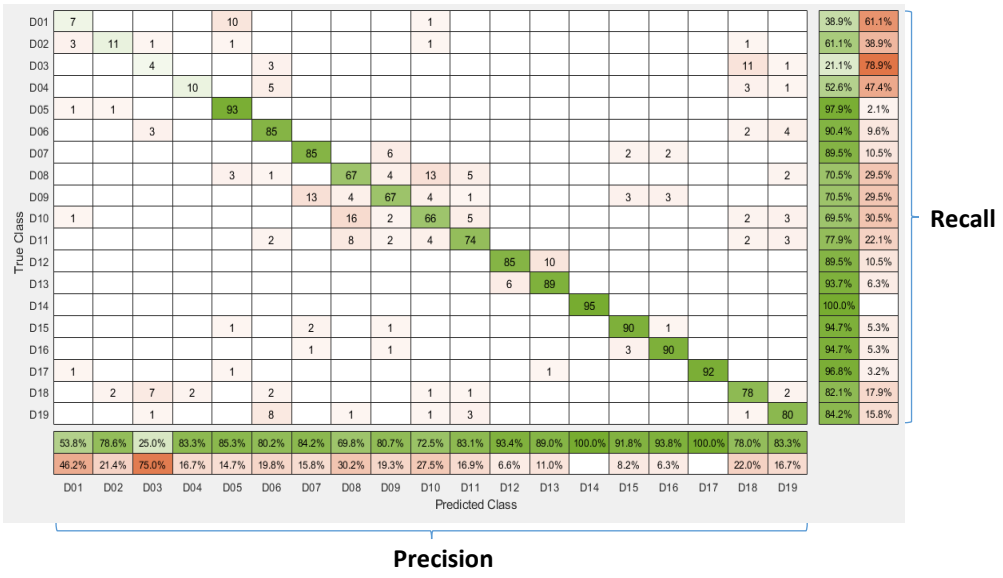


Figure 3.47: Confusion matrix example

CHAPTER 4

4 SIMULATION RESULTS AND DISCUSSION

4.1 Experiment 1 (using SisFall's ADXL345 accelerometer model and 5-fold cross-validation)

4.1.1 Experiment 1.1 (classifying multiple ADLs using 10 features)

The SisFall dataset was used for this set of experiments. The ADXL345 accelerometer model was used and has a resolution of 13 bits, and a range of −16g to +16g. In this model, 19 ADLs were classified, using 5-fold cross-validation and a total of 10 features, including Maximum (x, y, z axis), Minimum (x, y, z axis), Mean (x, y, z axis) and Position Vector (Magnitude).

There were 24 algorithms available in the classification learner app that were used in Experiment 1.1, and the simulation results were tabulated and presented in Table 4.1 with Recall, Precision, and Accuracy in percentages and Training Time in seconds.

Amongst 24 algorithms used in Experiment 1.1, Fine KNN has shown the best recall, precision and accuracy of 76.53%, 79.04% and 83.76%, respectively. The classifier had a training time of 0.184 seconds. Cubic SVM and Weighted KNN also showed good results in comparison with Fine KNN except having higher Training Time of 6.542 seconds of Cubic SVM. A confusion matrix for an instance of the experiment done using Fine KNN and Cubic SVM is also shown in Figure 4.1 and 4.2, respectively.

Table 4.1: Results from Experiment 1.1 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Decision Trees					
Fine Tree	60.51±0.97	60.98±1	0.605±0.01	67.68±1.05	0.44±0.812
Medium Tree	49.49±0.48	47.07±1.51	0.466±0.006	59.29±0.61	0.123±0.005
Coarse Tree	24.52±0.24	15.34±0.65	0.174±0.007	29.55±0.29	0.105±0.005
Discriminant					
Linear Discriminant	57.41±0.55	59.11±0.84	0.573±0.006	64.03±0.47	0.191±0.24
Quadratic Discriminant	64.32±0.71	70.76±1.71	0.651±0.01	73.06±0.58	0.181±0.044
Naïve Bayes					
Gaussian Naïve Bayes	58.35±0.84	59.26±0.67	0.579±0.007	62±0.43	0.296±0.131
Kernel Naïve Bayes	64.26±0.78	64.97±0.61	0.636±0.006	69.29±0.57	3.753±0.413
SVM					

Linear SVM	67.29±0.64	73.08±2.6	0.682±0.009	73.97±0.42	6.456±0.598
Quadratic SVM	74.6±0.86	76.35±0.98	0.752±0.009	79.8±0.76	6.522±0.139
Cubic SVM	75.59±0.69	77.64±1.15	0.763±0.009	81.27±0.51	6.542±0.047
Fine Gaussian SVM	61.86±0.45	66±0.3	0.623±0.004	74.49±0.54	7.029±0.298
Medium Gaussian SVM	69.04±0.61	73.14±2.74	0.694±0.007	78.06±0.57	6.27±0.148
Coarse Gaussian SVM	55.06±0.45	53.8±0.46	0.536±0.004	66.3±0.54	6.091±0.064
KNN					
Fine KNN*	76.53±1.08	79.04±1.25	0.773±0.012	83.76±0.64	0.184±0.235
Medium KNN	63.19±0.99	65.23±2.05	0.631±0.012	71.7±0.95	0.117±0.014
Coarse KNN	45.71±0.53	44.86±0.65	0.432±0.006	54.97±0.64	0.131±0.011
Cosine KNN	60.75±0.72	63.26±1.7	0.606±0.009	69.95±0.64	0.17±0.199
Cubic KNN	61.59±0.87	65.38±1.08	0.616±0.009	70.09±0.83	0.195±0.015
Weighted KNN	73.12±0.98	77.4±1.95	0.736±0.012	82.97±0.84	0.118±0.014
Ensemble/Miscellaneous					
Boosted Trees (AdaBoost)	54.33±0.68	55.59±2.21	0.531±0.009	64.79±0.65	2.669±0.219
Bagged Trees	69.85±0.55	73.01±0.9	0.703±0.006	77.87±0.41	2.226±0.127
Subspace Discriminant	53.21±0.57	51.8±1.89	0.517±0.005	61.05±0.51	1.838±0.162
Subspace KNN	72.42±0.92	75.97±1.42	0.731±0.01	80.35±0.72	1.879±0.03
RUSBoosted Trees	64.26±1.24	60.76±0.74	0.598±0.007	63.79±0.6	2.325±0.026

*Algorithm with highest recall, precision and accuracy

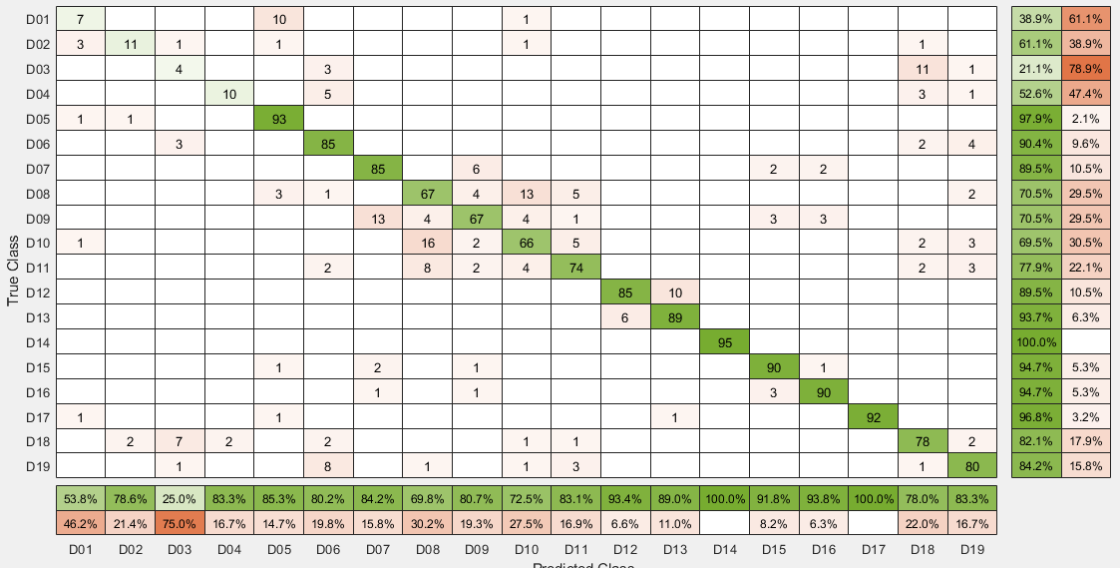


Figure 4.1: Confusion matrix for Fine KNN

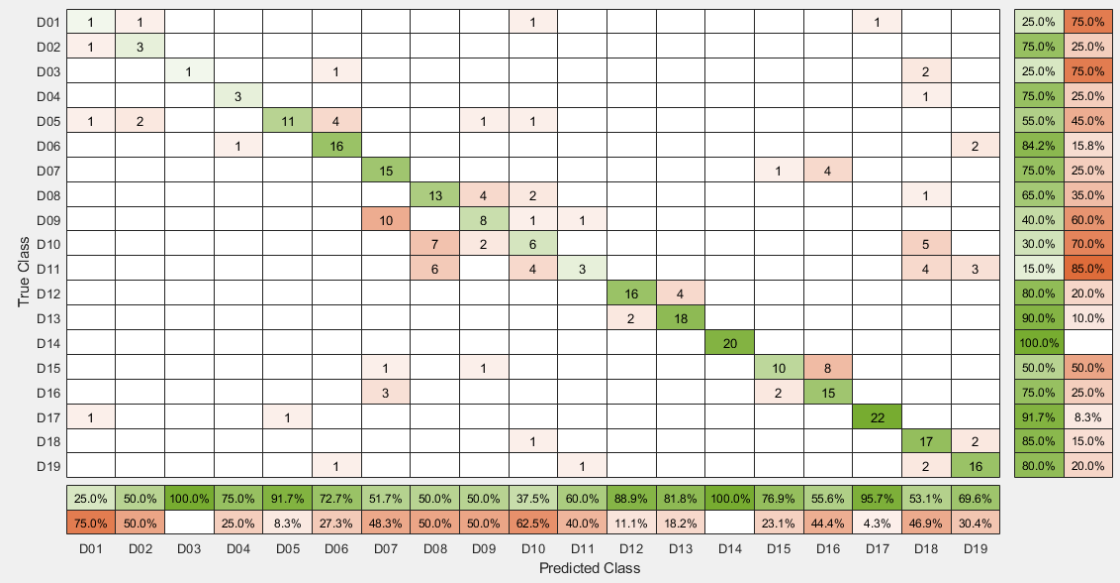


Figure 4.2: Confusion matrix for Cubic SVM

4.1.2 Experiment 1.2 (classifying multiple ADLs using 16 features)

Similar to Experiment 1.1, 19 ADLs were classified, and 5-fold cross-validation was used in Experiment 1.2. However, in this experiment, a total of 16 features were used. This includes Maximum (x, y, z axis), Minimum (x, y, z axis), Mean (x, y, z axis), Position Vector (Magnitude), Orientation (against x, y, z axis) and Standard Deviation (x, y, z axis values). The simulation results of Experiment 1.2 are tabulated in Table 4.2.

In this experiment, also 24 algorithms were used to generate simulation results, and only Quadratic Discriminant did not produce any results. Amongst all of the 23 algorithms of Experiment 1.2, Cubic SVM has shown the best recall, precision and accuracy of 87.68% and 90.61%, respectively. The algorithm had a training time of 6.216 seconds. Quadratic SVM also showed similar results of achieving a recall and precision of 87.1% and 89.42%, respectively and an overall accuracy of 89.87% with a similar training time of 6.348 seconds. However, in terms of faster training time, while having similar recall, precision and accuracy, Fine KNN was the most accurate at 88.85%, with a training time of only 0.198 seconds. A confusion matrix for an instance of the experiment done using Cubic SVM and Fine KNN is also shown in Figures 4.3 and 4.4, respectively.

Table 4.2: Results from Experiment 1.2 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Decision Tree					
Fine Tree	76.03±0.63	76.53±0.81	0.761±0.007	79.39±0.44	0.566±1.124
Medium Tree	63.09±1.01	63.75±1.81	0.606±0.01	69.21±0.52	0.134±0.009
Coarse Tree	24.67±0.21	16.77±0.92	0.183±0.009	29.73±0.25	0.112±0.004
Discriminant					
Linear Discriminant	76.72±0.45	78.11±0.71	0.768±0.005	78.34±0.37	0.305±0.58
Naïve Bayes					
Gaussian Naïve Bayes	74.19±0.62	74.24±0.62	0.734±0.006	75±0.44	0.329±0.12
Kernel Naïve Bayes	76.82±0.74	79.23±0.64	0.77±0.006	79.45±0.43	5.796±0.399
SVM					
Linear SVM	81.58±0.64	84.41±0.55	0.825±0.006	83.6±0.31	6.749±0.745
Quadratic SVM	87.1±0.96	89.42±0.94	0.879±0.01	89.87±0.63	6.348±0.09
Cubic SVM*	87.68±0.86	90.21±0.88	0.885±0.009	90.61±0.56	6.216±0.035
Fine Gaussian SVM	64.2±0.43	68.53±0.29	0.648±0.004	77.3±0.52	6.656±0.206
Medium Gaussian SVM	82.22±0.59	86.91±0.73	0.832±0.007	86.29±0.39	6.084±0.098
Coarse Gaussian SVM	63.81±0.48	67.69±2.34	0.637±0.005	74.61±0.52	5.859±0.183
KNN					
Fine KNN	85.62±0.86	88.95±1.04	0.866±0.009	88.85±0.32	0.198±0.242
Medium KNN	70±0.77	75.11±2.08	0.707±0.008	75.25±0.75	0.121±0.017
Coarse KNN	44.48±0.43	43.74±0.58	0.426±0.005	53.56±0.49	0.134±0.005
Cosine KNN	67.58±0.87	71.14±1.02	0.672±0.011	74.13±0.49	0.121±0.009
Cubic KNN	67.31±1.05	72.26±2.25	0.68±0.012	73.1±0.64	0.47±0.023
Weighted KNN	79.25±0.65	85±3.03	0.801±0.008	85.33±0.54	0.129±0.007
Ensemble/Miscellaneous					
Boosted Trees (AdaBoost)	68.44±0.85	70.59±2.32	0.677±0.011	76.47±0.74	3.55±0.242
Bagged Trees	82.38±0.94	85.51±1.13	0.833±0.01	85.49±0.69	2.311±0.227
Subspace Discriminant	74.72±0.43	74.02±0.39	0.741±0.004	76.46±0.44	1.958±0.12
Subspace KNN	78.07±1.07	83.45±1.61	0.794±0.012	84.54±0.55	2.152±0.138
RUSBoosted Trees	77.11±0.91	75.63±0.98	0.754±0.009	77.18±0.68	2.5±0.128

*Algorithm with highest recall, precision and accuracy

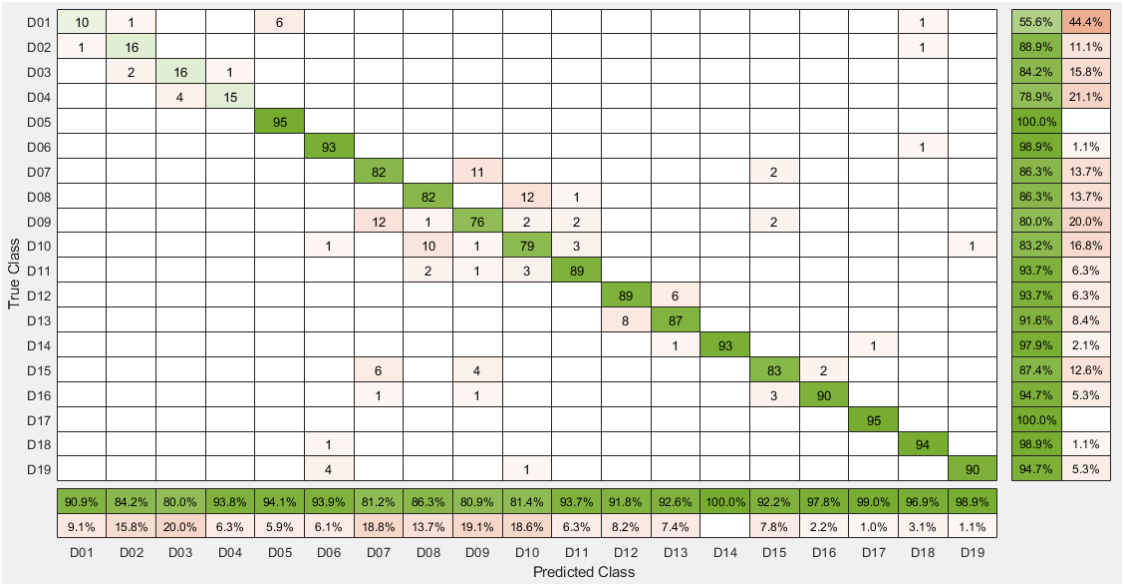


Figure 4.3: Confusion matrix for Cubic SVM

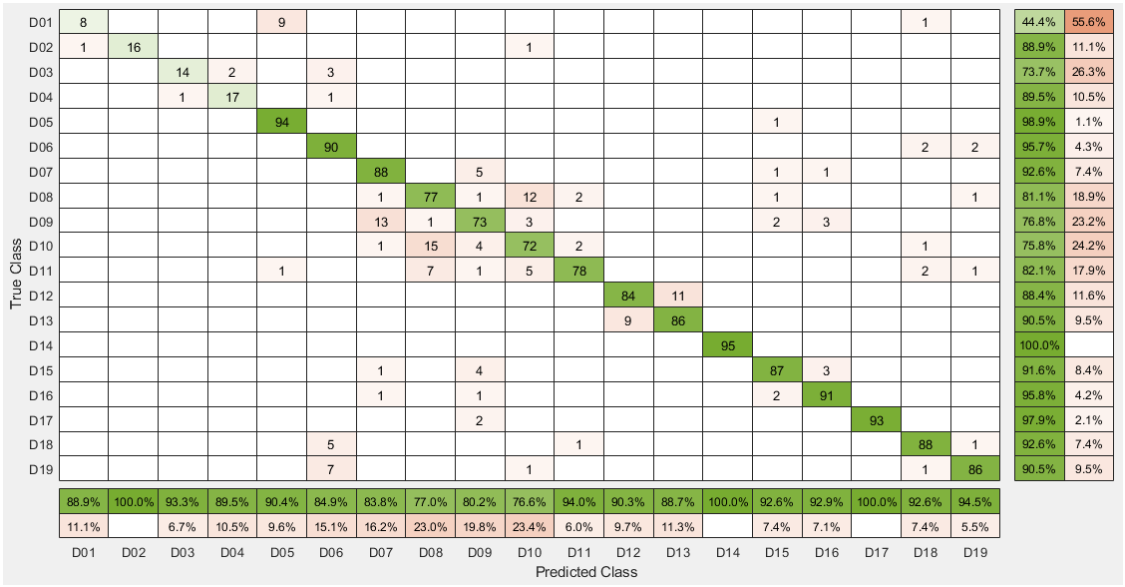


Figure 4.4: Confusion matrix for Fine KNN

4.1.3 Experiment 1.3 (classifying multiple ADLs and Falls using 16 features)

Experiment 1.3 shows the ability of the algorithm to classify both ADLs and Fall Events. 19 ADLs and 15 types of Falls were classified using 5-fold cross-validation and 16 features. In this case, both ADLs and Fall Events were classified together, and the results are tabulated in Table 4.3.

In this experiment, 23 algorithms were used to generate simulated results. Cubic SVM has shown the best recall and precision of 76.71% and 78.47%, respectively and overall accuracy of 77.17%. However, the algorithm had a longer training time of 21.87 seconds.

Similarly, Quadratic SVM also showed similar results of achieving a recall and precision of 76.72% and 78.47%, respectively and overall accuracy of 76.85% with a similar training time of 21.957 seconds. In terms of faster training time, Fine KNN was the most accurate having an accuracy of 71.59% in only 0.168 seconds. A confusion matrix for an instance of the experiment done using Cubic SVM and Fine KNN is also shown in Figures 4.5 and 4.6, respectively.

Table 4.3 Results from Experiment 1.3 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Decision Tree					
Fine Tree	59.57±0.82	60.61±0.97	0.59±0.008	59.77±0.81	0.278±0.13
Medium Tree	39.84±0.63	39.37±1.16	0.367±0.008	42.44±0.72	0.203±0.01
Coarse Tree	13.8±0.3	6.26±0.9	0.071±0.002	15.22±0.34	0.166±0.01
Discriminant					
Linear Discriminant	56.1±0.45	60.09±1.41	0.562±0.006	56.8±0.3	0.174±0.06
Naïve Bayes					
Gaussian Naïve Bayes	60.76±0.38	61.22±0.59	0.599±0.004	59.54±0.44	0.504±0.06
Kernel Naïve Bayes	63.61±0.26	64.64±0.63	0.629±0.003	63.48±0.22	17.246±0.34
SVM					
Linear SVM	70.52±0.36	73.12±0.37	0.713±0.004	70.25±0.41	20.546±0.62
Quadratic SVM	76.72±0.35	78.47±0.42	0.773±0.004	76.85±0.3	21.957±0.19
Cubic SVM*	76.71±0.39	78.6±0.4	0.773±0.004	77.17±0.27	21.87±0.29
Fine Gaussian SVM	54.27±0.53	68.58±1.28	0.579±0.006	59.86±0.59	21.75±1.01
Medium Gaussian SVM	71.8±0.51	75.04±0.39	0.726±0.005	72.83±0.44	21.385±0.48
Coarse Gaussian SVM	57.39±0.44	60.04±1.04	0.573±0.004	61.95±0.46	21.117±0.25
KNN					
Fine KNN	71.35±0.4	73.01±0.63	0.718±0.005	71.59±0.33	0.168±0.067
Medium KNN	57.83±0.45	59.81±0.34	0.581±0.004	60.01±0.37	0.16±0.005
Coarse KNN	38.38±0.54	38.32±0.81	0.364±0.006	42.41±0.62	0.223±0.103
Cosine KNN	57.63±0.66	59.55±1.23	0.575±0.008	59.99±0.34	0.162±0.012
Cubic KNN	55.75±0.47	57.85±0.46	0.559±0.004	58.19±0.57	1.488±0.129
Weighted KNN	67.64±0.59	70.51±1.35	0.68±0.005	69.9±0.45	0.192±0.1
Ensemble/Miscellaneous					
Boosted Trees (AdaBoost)	47.34±0.77	44.93±2.98	0.423±0.009	50.82±0.67	6.338±0.244
Bagged Trees	72.39±0.63	73.7±0.67	0.726±0.006	72.77±0.5	4.05±0.159
Subspace Discriminant	51.86±0.29	52±1.09	0.506±0.004	53.5±0.27	2.938±0.126
Subspace KNN	63.91±0.58	66.01±0.81	0.644±0.006	65.44±0.42	2.667±0.166
RUSBoosted Trees	57.18±0.64	53.86±1.26	0.53±0.006	55.21±0.51	4.046±0.069

*Algorithm with highest recall, precision and accuracy

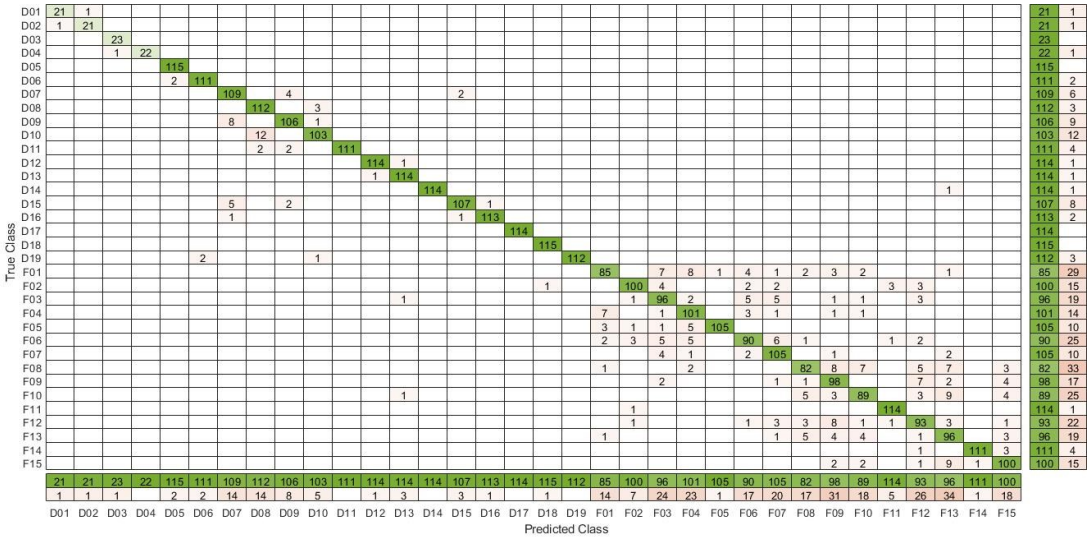


Figure 4.5: Confusion matrix for Cubic SVM

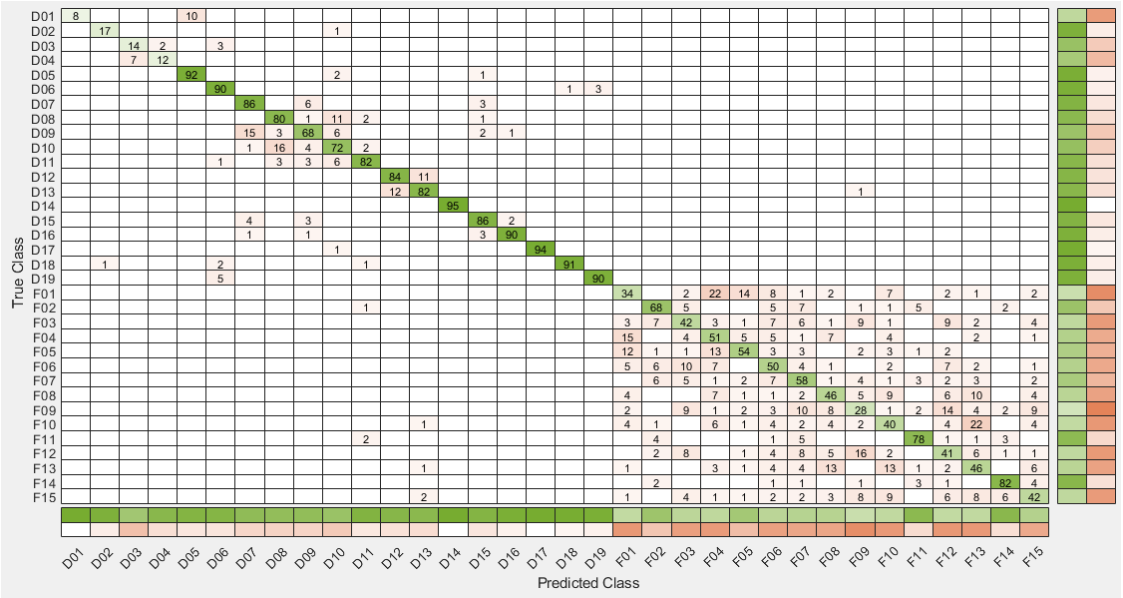


Figure 4.6: Confusion matrix for Fine KNN

4.1.4 Experiment 1.4 (classifying only an instance of ADL and Fall using 16 features)

Experiment 1.4 shows the ability of the algorithm to classify both ADLs and Fall Events as binary classification. ADLs and Falls were classified using 5-fold cross-validation and a total of 16 features. In this experiment, 23 algorithms have been used to produce simulated results. All the algorithms had been shown to perform well with an accuracy of over 90%, as tabulated in Table 4.4.

With a slight advantage, Fine KNN was most accurate in terms of shorter training time and gave 99.72 % accuracy with a training time of 0.165 seconds. Similarly, Cubic SVM gave 99.64% accuracy with a training time of 0.354 seconds, and Quadratic SVM gave 99.67% accuracy with a training time of 0.35 seconds. However, all algorithms had shown similar training time and similar accuracy to each other in comparison to other experiments. A confusion matrix for an instance of the experiment done using Fine KNN is also shown in Figure 4.7.

Table 4.4: Results from Experiment 1.4 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Decision Trees					
Fine Tree	98.54±0.17	98.54±0.17	0.985±0.002	98.54±0.17	0.116±0.008
Medium Tree	98.36±0.2	98.36±0.2	0.984±0.002	98.36±0.2	0.112±0.004
Coarse Tree	95.81±0.17	95.84±0.17	0.958±0.002	95.82±0.17	0.095±0.004
Discriminant					
Linear Discriminant	96.89±0.07	96.86±0.07	0.969±0.001	96.87±0.07	0.183±0.233
Quadratic Discriminant	98.69±0.05	98.68±0.05	0.987±0	98.68±0.05	0.131±0.036
Naïve Bayes					
Gaussian Naïve Bayes	96.78±0.09	96.77±0.09	0.967±0.001	96.73±0.1	0.156±0.145
Kernel Naïve Bayes	97.43±0.09	97.4±0.09	0.974±0.001	97.4±0.09	3.889±0.87
SVM					
Linear SVM	97.77±0.06	97.77±0.06	0.978±0.001	97.77±0.06	0.683±0.866
Quadratic SVM	99.67±0.04	99.68±0.04	0.997±0	99.67±0.04	0.35±0.014
Cubic SVM	99.64±0.06	99.64±0.06	0.996±0.001	99.64±0.06	0.354±0.011
Fine Gaussian SVM	98.35±0.11	98.33±0.11	0.983±0.001	98.31±0.11	1.526±0.063
Medium Gaussian SVM	99.61±0.05	99.62±0.05	0.996±0	99.62±0.05	0.385±0.017
Coarse Gaussian SVM	97.73±0.09	97.7±0.09	0.977±0.001	97.71±0.09	0.491±0.032
KNN					
Fine KNN*	99.72±0.06	99.73±0.06	0.997±0.001	99.72±0.06	0.165±0.114
Medium KNN	99.39±0.05	99.42±0.05	0.994±0.001	99.4±0.05	0.146±0.015
Coarse KNN	97.21±0.1	97.18±0.09	0.972±0.001	97.17±0.1	0.169±0.007
Cosine KNN	99.54±0.04	99.56±0.03	0.995±0	99.55±0.04	0.136±0.009
Cubic KNN	99.45±0.09	99.48±0.09	0.995±0.001	99.46±0.09	1.385±0.023
Weighted KNN	99.55±0.07	99.57±0.07	0.996±0.001	99.56±0.07	0.145±0.019
Ensemble/Miscellaneous					
Bagged Trees	99.12±0.1	99.13±0.11	0.991±0.001	99.13±0.1	1.949±0.048
Subspace Discriminant	95.93±0.07	95.94±0.06	0.959±0.001	95.86±0.07	1.893±0.117

Subspace KNN	99.58±0.07	99.59±0.07	0.996±0.001	99.59±0.07	2.251±0.247
RUSBoosted Trees	98.41±0.23	98.4±0.23	0.984±0.002	98.4±0.23	2.755±0.21

*Algorithm with highest recall, precision and accuracy

Figure 4.7: Confusion matrix for Fine KNN

4.2 Experiment 2 (using SisFall's ADXL345 accelerometer model and 10-fold cross-validation)

4.2.1 Experiment 2.1 (classifying multiple ADLs using 10 features)

Experiment 2 was done with the same parameters as Experiment 1. However, 10-fold cross-validation was used instead of 5-fold cross-validation. As in Experiment 1.1, the ADXL345 accelerometer was used, with 19 ADLs being classified using 10 features. There were 24 algorithms available in the classification learner app, and the simulation results were tabulated and presented in Table 4.5 with Recall, Precision, and Accuracy in percentages and Training Time in seconds.

Amongst 24 algorithms used in Experiment 2.1, the Fine KNN algorithm was the best in respect of accuracy (84.64%), precision (80.34%) and recall (77.85%) values. The classifier had a training time of 0.172 seconds. Quadratic and Cubic SVM also showed similar results in comparison with Fine KNN except having higher Training Time of

13.368 and 13.583 seconds, respectively. A confusion matrix for an instance of the experiment done using Fine KNN is also shown in Figure 4.8.

Table 4.5: Results from Experiment 2.1 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Decision Trees					
Fine Tree	61.57±1.32	62.23±1.28	0.6170±0.0125	68.04±0.96	0.218±0.010
Medium Tree	49.54±0.37	46.80±1.31	0.4645±0.0052	59.41±0.56	0.223±0.075
Coarse Tree	24.41±0.11	15.43±0.57	0.1749±0.0040	29.41±0.13	0.181±0.017
Discriminant					
Linear Discriminant	57.76±0.30	59.81±0.62	0.5782±0.0036	64.39±0.24	0.253±0.105
Quadratic Discriminant	66.06±0.54	71.40±0.71	0.6707±0.0059	73.73±0.41	0.302±0.019
Naïve Bayes					
Gaussian Naïve Bayes	58.56±0.39	59.34±0.38	0.5801±0.0038	61.97±0.22	0.403±0.042
Kernel Naïve Bayes	64.83±0.56	66.43±0.99	0.6422±0.0054	69.81±0.40	6.953±0.371
SVM					
Linear SVM	68.36±0.59	75.52±0.55	0.6944±0.0062	74.81±0.25	12.844±0.190
Quadratic SVM	75.78±0.68	77.36±0.71	0.7638±0.0064	80.87±0.50	13.368±0.182
Cubic SVM	77.02±0.49	79.19±0.54	0.7785±0.0051	82.38±0.44	13.583±0.390
Fine Gaussian SVM	62.93±0.18	66.64±0.16	0.6326±0.0017	75.78±0.22	13.955±0.135
Medium Gaussian SVM	70.06±0.61	75.63±3.75	0.7050±0.0078	78.83±0.58	12.729±0.206
Coarse Gaussian SVM	55.71±0.32	54.39±0.34	0.5428±0.0030	67.08±0.38	13.109±0.428
KNN					
Fine KNN*	77.85±0.52	80.34±0.55	0.7867±0.0053	84.64±0.43	0.172±0.012
Medium KNN	64.69±0.68	66.41±1.10	0.6458±0.0073	73.47±0.58	0.181±0.007
Coarse KNN	46.60±0.37	45.15±0.48	0.4407±0.0043	56.13±0.49	0.228±0.021
Cosine KNN	63.02±0.27	67.15±1.29	0.6316±0.0038	71.96±0.42	0.179±0.016
Cubic KNN	63.23±0.43	65.87±1.32	0.6318±0.0058	71.77±0.60	0.293±0.019
Weighted KNN	74.16±0.74	78.44±0.94	0.7472±0.0073	83.93±0.50	0.197±0.016
Ensemble/Miscellaneous					
Boosted Trees (AdaBoost)	55.00±0.57	54.23±0.94	0.5376±0.0060	65.71±0.64	5.430±0.106
Bagged Trees	71.05±0.77	74.05±1.22	0.7148±0.0082	79.15±0.54	5.153±0.099
Subspace Discriminant	52.94±0.49	51.61±2.38	0.5142±0.0052	61.10±0.42	3.987±0.143
Subspace KNN	72.78±0.92	76.28±1.22	0.7340±0.01	80.85±0.61	4.064±0.017
RUSBoosted Trees	65.49±0.53	61.59±0.46	0.6078±0.0053	64.87±0.52	4.824±0.285

*Algorithm with highest recall, precision and accuracy

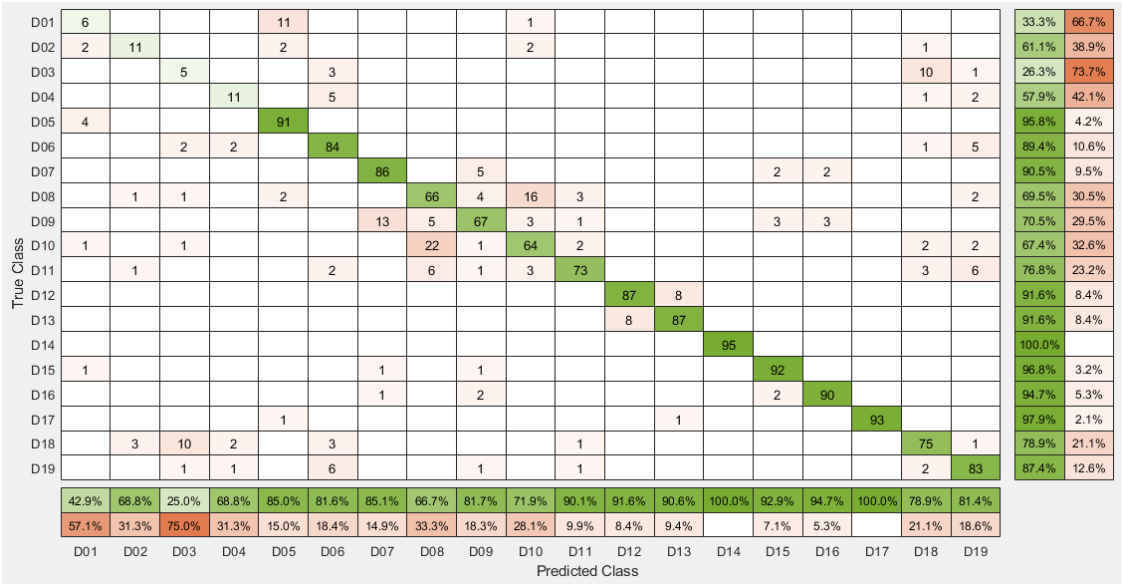


Figure 4.8: Confusion matrix for Fine KNN

4.2.2 Experiment 2.2 (classifying multiple ADLs using 16 features)

Experiment 2.2 used the same parameters as Experiment 1.2, except 10-fold cross-validation was used. As in Experiment 1.2, ADXL345 was used with 19 ADLs were classified and a total of 16 features. The results are tabulated in Table 4.6.

Amongst all the algorithms simulated, Cubic SVM gave the best performance, having a recall, precision and accuracy of 88.95%, 90.89% and 91.58%, respectively. Similarly, Quadratic SVM gave an accuracy of 90.53% with a training time of 14.1414 seconds, and Fine KNN had 89.97% accuracy with a training time of 0.25 seconds. A confusion matrix for Cubic SVM is also shown in Figure 4.9.

Table 4.6: Results from Experiment 2.2 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Decision Trees					
Fine Tree	76.94±0.65	77.11±0.58	0.7690±0.0056	80.28±0.47	0.276±0.023
Medium Tree	63.10±0.81	63.74±2.20	0.6054±0.0093	69.43±0.48	0.2454±0.0080
Coarse Tree	24.23±0.13	18.31±0.76	0.1976±0.0059	29.21±0.15	0.217±0.014
Discriminant					
Linear Discriminant	76.89±0.41	78.42±0.95	0.7694±0.0043	78.48±0.32	0.26±0.22
Naïve Bayes					
Gaussian Naïve Bayes	74.09±0.48	74.41±0.55	0.7329±0.0051	74.67±0.36	0.55±0.10
Kernel Naïve Bayes	77.47±0.35	79.77±0.53	0.7773±0.0037	79.81±0.34	12.17±0.42
SVM					
Linear SVM	82.99±0.45	85.90±0.35	0.8394±0.0040	84.25±0.35	13.6772±0.3004
Quadratic SVM	87.92±0.42	89.88±0.30	0.8868±0.0038	90.53±0.39	14.1414±0.3759
Cubic SVM*	88.95±0.78	90.89±0.69	0.8964±0.0076	91.58±0.45	13.1848±0.1660
Fine Gaussian SVM	65.36±0.27	69.07±0.24	0.6574±0.0024	78.70±0.32	12.8105±1.0714

Medium Gaussian SVM	83.65±0.33	87.61±0.27	0.8467±0.0030	87.34±0.30	12.4999±0.0756
Coarse Gaussian SVM	65.30±0.37	71.16±2.06	0.6554±0.0046	75.61±0.27	12.7435±0.5413
KNN					
Fine KNN	87.23±0.46	90.62±0.48	0.8821±0.0043	89.97±0.45	0.25±0.20
Medium KNN	71.40±0.32	76.62±1.46	0.7230±0.0036	76.56±0.22	0.200±0.034
Coarse KNN	44.82±0.52	43.59±0.59	0.4292±0.0053	53.92±0.63	0.217±0.038
Cosine KNN	69.01±0.49	72.45±0.71	0.6888±0.0056	75.51±0.48	0.211±0.041
Cubic KNN	69.07±0.64	74.20±2.05	0.6993±0.0068	74.45±0.64	0.623±0.059
Weighted KNN	80.59±0.55	87.80±0.97	0.8155±0.0048	86.48±0.36	0.200±0.039
Ensemble/Miscellaneous					
Boosted Trees (AdaBoost)	68.80±0.71	70.92±1.03	0.6811±0.0086	76.91±0.49	6.57±0.35
Bagged Trees	83.46±0.64	86.29±0.71	0.8434±0.0063	86.40±0.49	4.66±0.21
Subspace Discriminant	75.32±0.29	75.09±1.76	0.7476±0.0034	77.04±0.22	4.14±0.14
Subspace KNN	79.30±0.55	84.71±1.20	0.8071±0.0065	85.37±0.32	4.15±0.17
RUSBoosted Trees	76.75±0.43	75.60±0.74	0.7511±0.0062	76.86±0.60	5.62±0.11

*Algorithm with highest recall, precision and accuracy

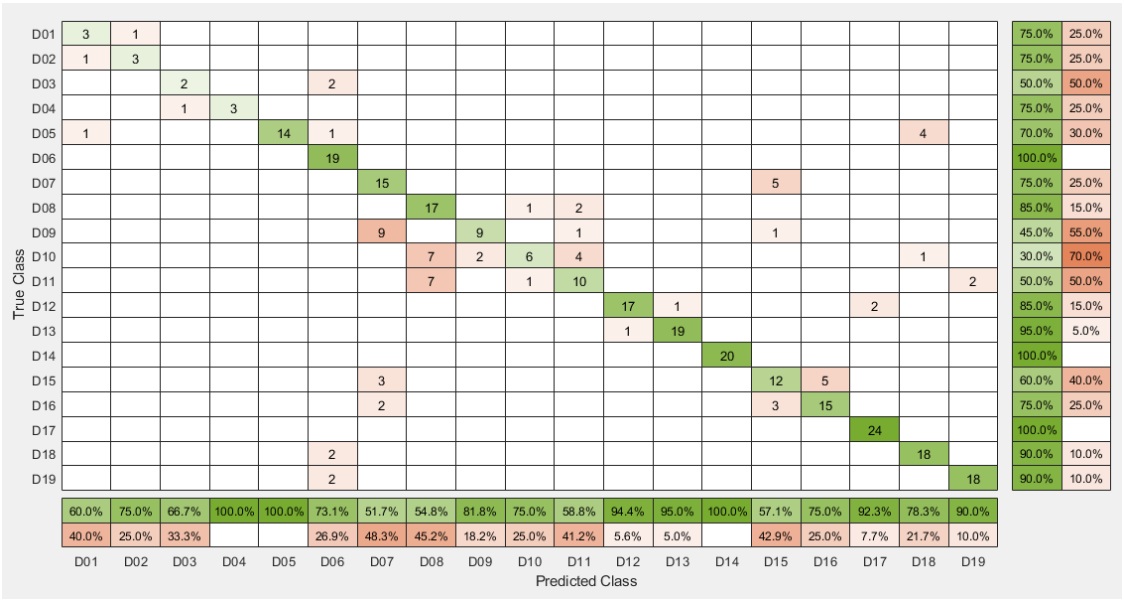


Figure 4.9: Confusion matrix for Cubic SVM

4.2.3 Experiment 2.3 (classifying multiple ADLs and Falls using 16 features)

Similar to Experiment 1.3, Experiment 2.3 used the same parameters, except using 10-fold cross-validation. As in Experiment 1.3, 19 ADLs and 15 Fall Events were classified using a total of 16 features. The results are tabulated in Table 4.7.

For this experiment, Cubic SVM has shown the best recall and precision of 77.87% and 79.71%, respectively and an overall accuracy of 78.14%. However, the algorithm had a longer training time of 40.426 seconds. A confusion matrix for Cubic SVM is also shown in Figure 4.10. Quadratic SVM also showed similar results of achieving a recall and

precision of 77.70% and 79.13%, respectively and an overall accuracy of 77.75% with a similar training time of 40.450 seconds. Medium Gaussian SVM had also shown a similar result. However, in terms of faster training time (less than one second), Fine KNN was the most accurate having an accuracy of 72.58% in only 0.241 seconds.

Table 4.7: Results from Experiment 2.3 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Decision Trees					
Fine Tree	59.19±0.61	60.62±1.07	0.5868±0.0067	59.17±0.60	0.488±0.088
Medium Tree	39.14±0.46	38.64±0.69	0.3616±0.0047	41.77±0.42	0.368±0.015
Coarse Tree	14.44±0.21	7.34±1.19	0.0839±0.0024	15.93±0.23	0.293±0.016
Discriminant					
Linear Discriminant	56.24±0.30	60.20±1.49	0.5633±0.0038	57.14±0.29	0.275±0.025
Naïve Bayes					
Gaussian Naïve Bayes	60.93±0.24	61.61±0.33	0.6011±0.0025	59.73±0.19	0.781±0.050
Kernel Naïve Bayes	64.03±0.38	64.96±0.51	0.6325±0.0036	63.88±0.28	20.908±0.223
SVM					
Linear SVM	70.87±0.30	73.44±0.28	0.7160±0.0028	70.58±0.32	39.390±0.797
Quadratic SVM	77.70±0.33	79.13±0.29	0.7818±0.0032	77.75±0.30	40.450±0.250
Cubic SVM*	77.87±0.58	79.71±0.51	0.7847±0.0057	78.14±0.45	40.426±0.157
Fine Gaussian SVM	56.12±0.28	71.40±0.29	0.5984±0.0029	61.84±0.31	42.330±0.155
Medium Gaussian SVM	72.97±0.34	75.96±0.33	0.7371±0.0033	73.93±0.31	38.655±0.432
Coarse Gaussian SVM	58.45±0.33	62.73±1.33	0.5849±0.0041	62.72±0.27	38.479±0.453
KNN					
Fine KNN	72.37±0.36	74.11±0.34	0.7279±0.0033	72.58±0.34	0.241±0.066
Medium KNN	59.48±0.55	61.73±1.05	0.5978±0.0052	61.86±0.46	0.235±0.009
Coarse KNN	39.84±0.47	41.37±1.60	0.3797±0.0049	43.89±0.58	0.291±0.058
Cosine KNN	59.40±0.64	61.63±0.61	0.5935±0.0069	61.89±0.51	0.237±0.010
Cubic KNN	57.57±0.53	59.43±0.38	0.5775±0.0051	59.95±0.45	1.712±0.106
Weighted KNN	68.79±0.34	72.57±1.00	0.6916±0.0036	71.07±0.26	0.272±0.065
Ensemble/Miscellaneous					
Boosted Trees (AdaBoost)	47.34±0.25	44.05±1.61	0.4182±0.0030	50.81±0.26	12.154±0.564
Bagged Trees	73.61±0.34	75.00±0.50	0.7384±0.0036	73.83±0.40	7.887±0.344
Subspace Discriminant	52.17±0.18	52.15±0.35	0.5086±0.0024	53.86±0.19	4.919±0.103
Subspace KNN	64.82±0.69	67.18±1.00	0.6529±0.0075	66.24±0.65	5.145±0.169
RUSBoosted Trees	57.30±0.54	54.02±0.92	0.5318±0.0054	55.25±0.60	7.914±0.216

*Algorithm with highest recall, precision and accuracy

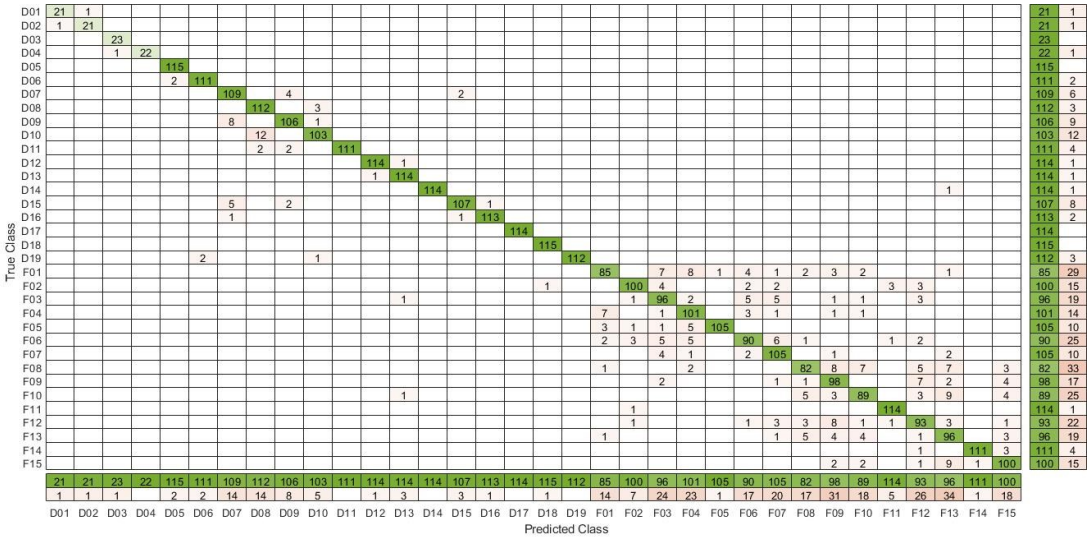


Figure 4.10: Confusion matrix for Cubic SVM

4.2.4 Experiment 2.4 (classifying only an instance of ADL and Fall using 16 features)

Similar to Experiment 1.4, Experiment 2.4 classified both ADLs and Fall Events as a binary classification. ADLs and Falls were classified using 10-fold cross-validation and 16 features. In this experiment, 23 algorithms have been used to produce simulated results. All the algorithms had been shown to perform well, as tabulated in Table 4.8.

With a slight advantage, Fine KNN is most accurate in terms of shorter training time and gave 99.72 % accuracy with a training time of 0.201 seconds. Similarly, Cubic SVM gave 99.67% accuracy with a training time of 0.700 seconds, and Quadratic SVM gave 99.70% accuracy with a training time of 0.702 seconds. However, all algorithms had shown similar training time and accuracy to each other compared to other experiments. An example of the confusion matrix for Fine KNN is shown in Figure 4.11.

Table 4.8: Results from Experiment 2.4 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Decision Trees					
Fine Tree	98.54±0.21	98.53±0.21	0.9853±0.0021	98.54±0.21	0.195±0.011
Medium Tree	98.28±0.13	98.27±0.13	0.9827±0.0013	98.28±0.13	0.184±0.010
Coarse Tree	95.80±0.16	95.83±0.16	0.9581±0.0016	95.82±0.16	0.153±0.006
Discriminant					
Linear Discriminant	96.87±0.07	96.84±0.07	0.9684±0.0007	96.84±0.07	0.201±0.061
Quadratic Discriminant	98.71±0.03	98.70±0.03	0.9871±0.0003	98.71±0.03	0.216±0.022
Naïve Bayes					
Gaussian Naïve Bayes	96.78±0.05	96.76±0.05	0.9673±0.0005	96.73±0.05	0.171±0.054

Kernel Naïve Bayes	97.47±0.05	97.44±0.05	0.9744±0.0005	97.44±0.05	4.159±0.213
SVM					
Linear SVM	97.85±0.05	97.85±0.05	0.9785±0.0005	97.85±0.05	0.864±0.068
Quadratic SVM	99.69±0.07	99.70±0.07	0.9970±0.0007	99.70±0.07	0.702±0.029
Cubic SVM	99.67±0.04	99.68±0.04	0.9967±0.0004	99.67±0.04	0.700±0.021
Fine Gaussian SVM	98.47±0.05	98.44±0.05	0.9843±0.0005	98.43±0.05	3.215±0.078
Medium Gaussian SVM	99.62±0.02	99.62±0.02	0.9962±0.0002	99.62±0.02	0.781±0.064
Coarse Gaussian SVM	97.84±0.07	97.82±0.07	0.9783±0.0007	97.83±0.07	1.002±0.053
KNN					
Fine KNN*	99.71±0.03	99.73±0.03	0.9972±0.0003	99.72±0.03	0.201±0.050
Medium KNN	99.45±0.04	99.47±0.04	0.9946±0.0004	99.46±0.04	0.197±0.007
Coarse KNN	97.34±0.07	97.32±0.07	0.9731±0.0007	97.31±0.07	0.238±0.020
Cosine KNN	99.59±0.04	99.61±0.04	0.9960±0.0004	99.60±0.04	0.202±0.015
Cubic KNN	99.50±0.06	99.53±0.05	0.9951±0.0006	99.51±0.06	1.644±0.037
Weighted KNN	99.56±0.05	99.58±0.05	0.9957±0.0005	99.57±0.05	0.200±0.010
Ensemble/Miscellaneous					
Boosted Trees (AdaBoost)	99.10±0.12	99.11±0.12	0.9911±0.0012	99.11±0.12	4.134±0.350
Bagged Trees	99.12±0.08	99.12±0.08	0.9912±0.0008	99.12±0.08	3.648±0.364
Subspace Discriminant	95.91±0.09	95.93±0.09	0.9585±0.0010	95.85±0.10	3.279±0.096
Subspace KNN	99.62±0.06	99.63±0.06	0.9963±0.0006	99.63±0.06	4.063±0.178
RUSBoosted Trees	98.43±0.25	98.43±0.25	0.9843±0.0025	98.43±0.25	4.996±0.120

*Algorithm with highest recall, precision and accuracy

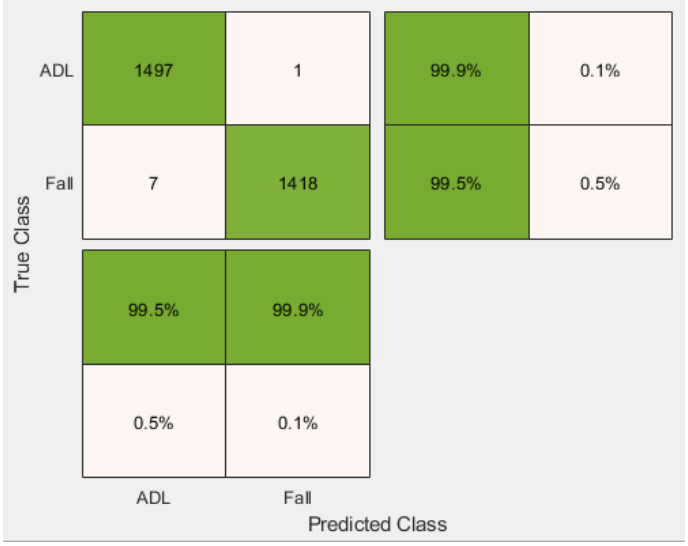


Figure 4.11: Confusion matrix for Fine KNN

4.3 Experiment 3 (using SisFall's MMA8451Q accelerometer model and 10-fold cross-validation)

4.3.1 Experiment 3.1 (classifying multiple ADLs using 10 features)

Experiment 3 was run with the same parameters as Experiment 2. However, the MMA8451Q accelerometer was used instead of the ADXL345 accelerometer. As in

Experiment 2.1, 19 ADLs were classified using 10 features. This includes Maximum (x, y, z axis), Minimum (x, y, z axis), Mean (x, y, z axis) and Position Vector (Magnitude). There were 24 algorithms used for simulation, and the simulation results were tabulated and presented in Table 4.9 with Recall, Precision, and Accuracy in percentages and Training Time in seconds.

Amongst 24 algorithms used in Experiment 2.1, the Fine KNN algorithm was the best overall in respect of accuracy (85.23%), precision (80.82%) and recall (78.72%) values. The classifier had a training time of 0.176 seconds. Weighted KNN has similar with an accuracy of 84.11% and a training time of 0.169 seconds. Cubic SVM also showed similar results (84.01% accuracy) in comparison with Fine KNN except having a higher Training Time of 11.973 seconds. A confusion matrix for an instance of the experiment done using Fine KNN is also shown in Figure 4.12.

Table 4.9: Results from Experiment 3.1 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Decision Trees					
Fine Tree	64.50±0.82	64.77±0.90	0.6446±0.0084	70.83±0.61	0.268±0.125
Medium Tree	51.93±0.34	51.21±0.86	0.5006±0.0035	61.66±0.38	0.233±0.040
Coarse Tree	23.21±0.06	17.30±0.14	0.1661±0.0015	27.96±0.07	0.189±0.025
Discriminant					
Linear Discriminant	60.34±0.58	61.68±0.60	0.6048±0.0061	66.31±0.36	0.205±0.063
Quadratic Discriminant	67.94±0.54	72.49±1.08	0.6898±0.0064	75.45±0.40	0.310±0.032
Naïve Bayes					
Gaussian Naïve Bayes	61.99±0.75	61.80±0.46	0.6075±0.0059	63.99±0.26	0.409±0.044
Kernel Naïve Bayes	68.26±0.61	69.77±1.42	0.6752±0.0070	71.96±0.38	6.856±0.675
SVM					
Linear SVM	69.41±0.40	73.99±1.02	0.7059±0.0055	75.83±0.37	11.662±0.249
Quadratic SVM	77.50±0.53	79.48±0.57	0.7824±0.0054	82.87±0.39	12.520±0.386
Cubic SVM	78.58±0.55	81.23±0.82	0.7961±0.0062	84.01±0.53	11.973±0.076
Fine Gaussian SVM	63.21±0.23	66.93±0.26	0.6344±0.0022	76.11±0.28	12.624±0.048
Medium Gaussian SVM	72.62±0.48	80.33±0.78	0.7370±0.0050	80.55±0.48	11.542±0.253
Coarse Gaussian SVM	56.48±0.22	54.68±0.25	0.5510±0.0021	68.01±0.27	11.450±0.201
KNN					
Fine KNN*	78.72±0.64	80.82±0.85	0.7946±0.0068	85.23±0.46	0.176±0.033

Medium KNN	66.38±0.57	69.82±1.31	0.6680±0.0065	74.13±0.44	0.180±0.006
Coarse KNN	49.22±0.48	48.51±0.50	0.4701±0.0053	59.27±0.54	0.187±0.010
Cosine KNN	63.28±0.51	67.81±0.59	0.6345±0.0054	72.13±0.41	0.163±0.010
Cubic KNN	63.98±0.71	65.90±1.09	0.64±0. 0074	72.56±0.60	0.255±0.012
Weighted KNN	75.21±0.80	80.97±1.65	0.7622±0.0089	84.11±0.59	0.169±0.009
Ensemble/Miscellaneous					
Boosted Trees (AdaBoost)	57.36±0.63	57.59±0.51	0.5654±0.0061	67.11±0.60	5.192±0.301
Bagged Trees	73.09±0.89	76.57±1.29	0.7371±0.0091	80.30±0.57	4.245±0.023
Subspace Discriminant	54.93±0.51	55.05±2.78	0.5344±0.0066	62.64±0.50	3.257±0.092
Subspace KNN	74.27±0.33	77.82±0.91	0.7512±0.0034	81.13±0.43	3.369±0.114
RUSBoosted Trees	65.27±0.92	62.28±0.70	0.6098±0.0084	65.26±0.84	4.504±0.045

*Algorithm with highest recall, precision and accuracy

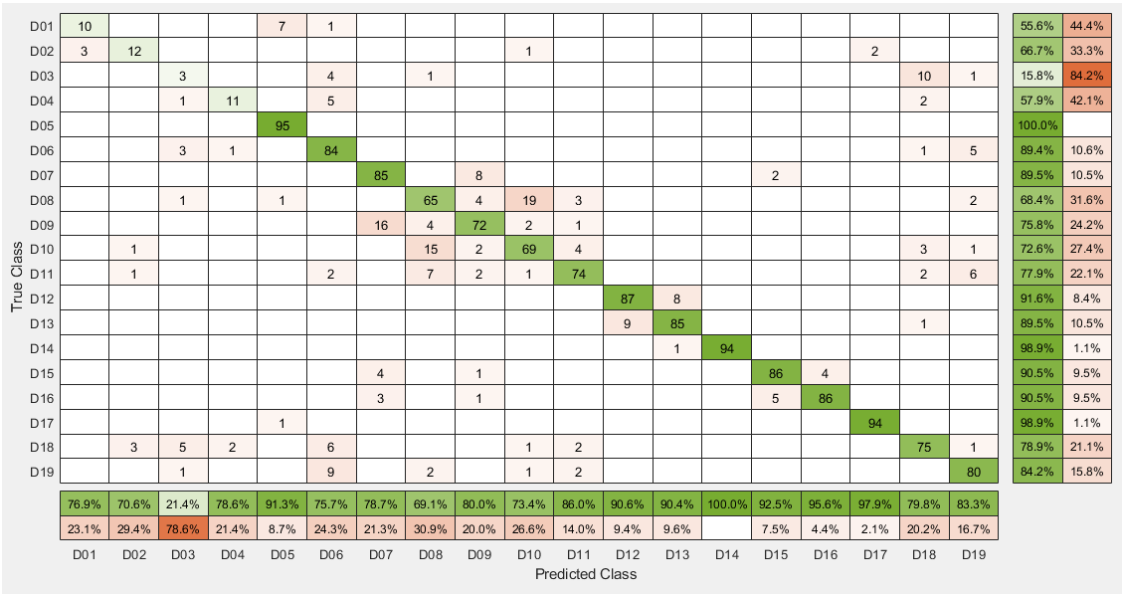


Figure 4.12: Confusion matrix for Fine KNN

4.3.2 Experiment 3.2 (classifying multiple ADLs using 16 features)

Experiment 3.2 used the same parameters as Experiment 2.2, except the MMA8451Q accelerometer was used. As in Experiment 2.2, 19 ADLs were classified, and a total of 16 features were used. The results are tabulated in Table 4.10.

Amongst all the algorithms simulated, Cubic SVM gave the best performance, having a recall, precision and accuracy of 90.52%, 91.77% and 92.08%, respectively. Similarly, Quadratic SVM gave an accuracy of 91.7% with a training time of 12.955 seconds, and

Fine KNN had 90.21% accuracy with a training time of 0.243 seconds. A confusion matrix for an instance of the experiment done using Cubic SVM is also shown in Figure 4.13.

Table 4.10: Results from Experiment 3.2 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Decision Trees					
Fine Tree	76.91±0.9	76.39±0.89	0.765±0.009	80.05±0.76	0.285±0.036
Medium Tree	65.35±0.66	62.81±1.92	0.618±0.007	71.04±0.33	0.242±0.025
Coarse Tree	24.27±0.08	17.82±0.43	0.195±0.003	29.24±0.09	0.194±0.012
Discriminant					
Linear Discriminant	77.1±0.34	79.17±0.51	0.775±0.004	78.6±0.29	0.272±0.282
Naïve Bayes					
Gaussian Naïve Bayes	75.85±0.29	75.84±0.52	0.751±0.003	75.89±0.23	0.536±0.194
Kernel Naïve Bayes	77.97±0.32	79.5±0.71	0.777±0.005	79.73±0.36	10.268±0.14
SVM					
Linear SVM	82.51±0.48	85.24±0.44	0.833±0.005	84.45±0.31	12.368±0.716
Quadratic SVM	89.63±0.52	91.15±0.46	0.902±0.005	91.7±0.44	12.955±0.18
Cubic SVM*	90.52±0.62	91.77±0.56	0.91±0.006	92.08±0.38	13.059±0.364
Fine Gaussian SVM	63.87±0.27	69.44±0.18	0.65±0.002	76.91±0.32	13.588±0.135
Medium Gaussian SVM	84.31±0.32	88.67±0.23	0.856±0.003	87.33±0.29	12.945±1.004
Coarse Gaussian SVM	67.45±0.42	74.19±0.31	0.682±0.004	76.55±0.27	12.063±0.227
KNN					
Fine KNN	88.49±0.3	90.18±0.38	0.892±0.003	90.21±0.23	0.243±0.222
Medium KNN	71.22±0.77	78.41±1.44	0.721±0.008	76.3±0.65	0.162±0.005
Coarse KNN	43.15±0.46	42.91±0.43	0.421±0.004	51.95±0.53	0.193±0.016
Cosine KNN	69.14±0.73	72.88±1.16	0.688±0.009	75.25±0.47	0.174±0.017
Cubic KNN	69.62±0.64	74.61±1.02	0.704±0.007	74.68±0.54	0.562±0.033
Weighted KNN	82.13±0.44	85.93±1.14	0.829±0.005	87.46±0.44	0.188±0.042
Ensemble/Miscellaneous					
Boosted Trees (AdaBoost)	70.46±0.46	70.83±0.99	0.692±0.005	76.81±0.51	5.799±0.153
Bagged Trees	84.4±0.8	87.26±0.84	0.854±0.008	86.96±0.7	4.401±0.089
Subspace Discriminant	75.55±0.25	74.85±0.36	0.75±0.003	77.18±0.26	3.659±0.159
Subspace KNN	77.56±0.81	84.76±1.1	0.79±0.01	84.6±0.45	3.531±0.176
RUSBoosted Trees	77.37±0.6	76.49±0.72	0.761±0.007	77.82±0.57	4.608±0.038

*Algorithm with highest recall, precision and accuracy

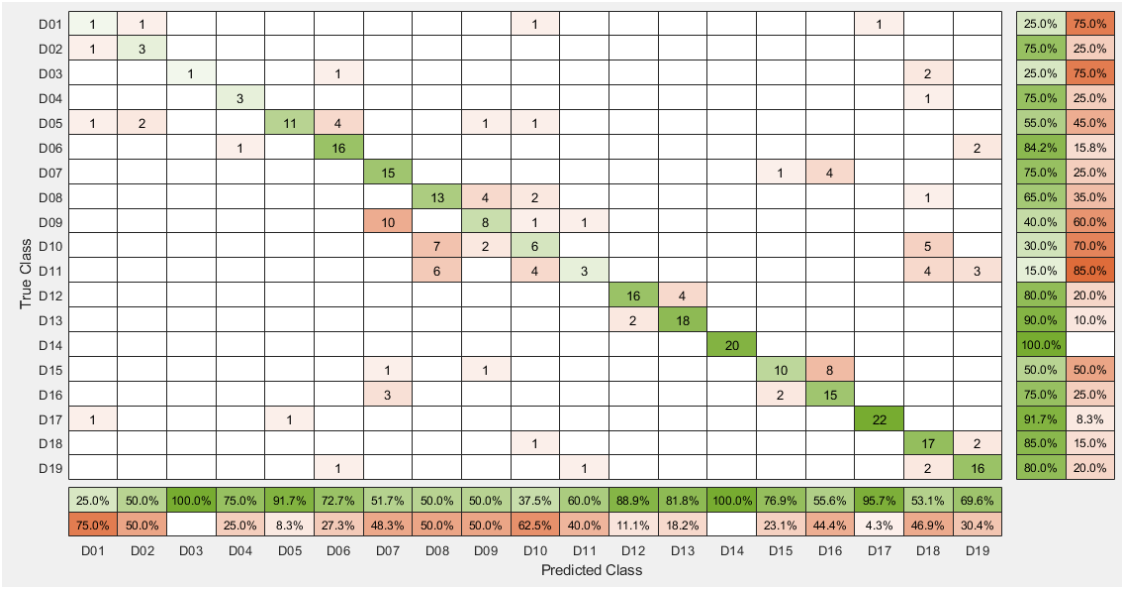


Figure 4.13: Confusion matrix for Cubic SVM

4.3.3 Experiment 3.3 (classifying multiple ADLs and Falls using 16 features)

Experiment 3.3 used the same parameters as Experiment 2.3, except the MMA8451Q accelerometer was used. As in Experiment 2.3, 19 ADLs and 15 Falls were classified via a total of 16 features. The results are tabulated in Table 4.11.

For this experiment, Cubic SVM has shown the best recall and precision of 78.62% and 80.18%, respectively and an overall accuracy of 78.43%. However, the algorithm had a longer training time of 42.807 seconds. Quadratic SVM also showed similar results of achieving a recall and precision of 78.43% and 79.94%, respectively and overall accuracy of 78.42% with a similar training time of 43.028 seconds. Similarly, Medium Gaussian SVM had shown a similar recall, precision, accuracy and training time. However, in terms of faster training time (less than one second), Fine KNN was the most accurate having an accuracy of 73.11% in only 0.254 seconds. The confusion matrix for an instance of the experiment done using Cubic SVM is also shown in Figure 4.14.

Table 4.11: Results from Experiment 3.3 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Decision Trees					
Fine Tree	59.67±0.47	60.66±0.71	0.591±0.004	59.64±0.58	0.489±0.117
Medium Tree	40.59±0.73	40.59±0.77	0.383±0.007	42.03±0.51	0.358±0.011
Coarse Tree	14.47±0.07	7.64±0.7	0.087±0.002	15.95±0.08	0.307±0.024
Discriminant					
Linear Discriminant	59.15±0.39	61.24±1.56	0.59±0.004	58.83±0.36	0.254±0.018
Naïve Bayes					

Gaussian Naïve Bayes	62.4±0.28	62.35±0.34	0.613±0.003	60.99±0.27	0.733±0.022
Kernel Naïve Bayes	64.61±0.2	65.34±0.26	0.633±0.002	64.14±0.1	19.005±0.353
SVM					
Linear SVM	71.84±0.4	74.09±0.32	0.723±0.004	71.76±0.29	40.925±0.782
Quadratic SVM	78.43±0.3	79.94±0.36	0.788±0.003	78.42±0.26	43.028±0.67
Cubic SVM*	78.62±0.42	80.18±0.32	0.791±0.004	78.43±0.26	42.807±0.424
Fine Gaussian SVM	55.48±0.34	70.21±1.23	0.592±0.003	61.16±0.4	42.707±1.131
Medium Gaussian SVM	74.24±0.41	77.14±0.36	0.75±0.004	74.69±0.44	38.172±0.336
Coarse Gaussian SVM	59.84±0.23	63.93±1.17	0.6±0.003	64.09±0.25	39.262±1.054
KNN					
Fine KNN	73.32±0.31	74.17±0.33	0.736±0.003	73.11±0.29	0.254±0.093
Medium KNN	60.57±0.42	64.62±1.24	0.609±0.005	62.66±0.37	0.234±0.014
Coarse KNN	39.42±0.41	40.24±0.86	0.382±0.004	43.57±0.44	0.273±0.014
Cosine KNN	59.81±0.36	61.85±0.4	0.595±0.004	61.94±0.33	0.272±0.086
Cubic KNN	58.88±0.28	62.58±1.01	0.59±0.003	61.3±0.22	1.752±0.192
Weighted KNN	70.46±0.38	74.06±0.35	0.71±0.004	72.13±0.33	0.264±0.087
Ensemble/Miscellaneous					
Boosted Trees (AdaBoost)	46.95±0.41	46.61±1.86	0.421±0.006	51.02±0.29	12.386±0.585
Bagged Trees	73.68±0.58	75.03±0.57	0.74±0.006	73.97±0.52	7.903±0.384
Subspace Discriminant	55.43±0.27	57.06±0.2	0.548±0.003	56.41±0.29	4.886±0.098
Subspace KNN	64.13±0.39	67.92±0.57	0.647±0.004	66.32±0.36	4.796±0.068
RUSBoosted Trees	57.62±0.58	54.63±1.33	0.535±0.005	55.84±0.47	7.378±0.124

*Algorithm with highest recall, precision and accuracy

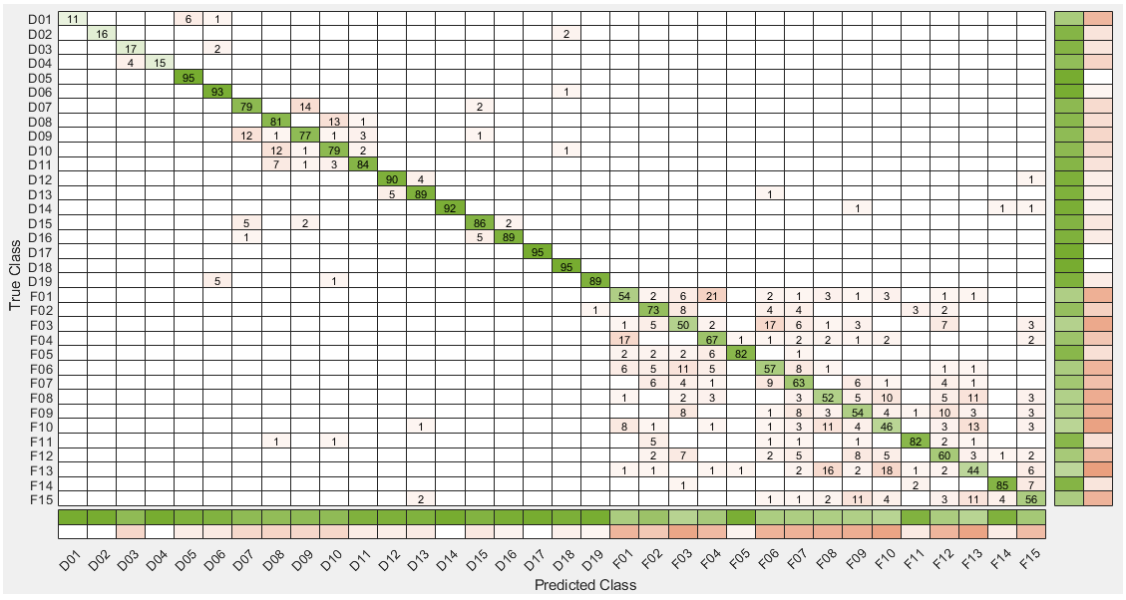


Figure 4.14: Confusion matrix for Cubic SVM

4.3.4 Experiment 3.4 (classifying only an instance of ADL and Fall using 16 features)

Experiment 3.4 shows the ability of the algorithm to classify both ADLs and Fall Events as binary classification. Similar to Experiment 2.4, ADLs and Falls were classified using 10-fold cross-validation and a total of 16 features. All the algorithms had been shown to perform well, as shown in Table 4.12. With a slight advantage, Fine KNN was most accurate in terms of shorter training time and gave 99.59% accuracy with a training time of 0.239 seconds. Medium Gaussian SVM was the most accurate, giving 99.65% accuracy with a training time of 0.805 seconds and Quadratic SVM gave 99.64% accuracy with a training time of 0.638 seconds. However, all algorithms had shown similar training time and similar accuracy to each other in comparison to other experiments. A confusion matrix for Medium Gaussian SVM is shown in Figure 4.15.

Table 4.12: Results from Experiment 3.4 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Decision Trees					
Fine Tree	98.37±0.13	98.38±0.14	0.984±0.0014	98.38±0.14	0.172±0.006
Medium Tree	98.31±0.08	98.3±0.08	0.983±0.0008	98.31±0.08	0.172±0.013
Coarse Tree	96.37±0.13	96.35±0.13	0.964±0.0013	96.36±0.13	0.154±0.013
Discriminant					
Linear Discriminant	96.97±0.08	96.94±0.08	0.969±0.0008	96.95±0.08	0.255±0.277
Quadratic Discriminant	98.78±0.03	98.75±0.04	0.988±0.0004	98.76±0.04	0.187±0.032
Naïve Bayes					
Gaussian Naïve Bayes	96.63±0.08	96.62±0.07	0.966±0.0008	96.57±0.08	0.207±0.166
Kernel Naïve Bayes	97.59±0.08	97.56±0.09	0.976±0.0009	97.56±0.09	3.863±0.32
SVM					
Linear SVM	98.28±0.05	98.29±0.05	0.983±0.0005	98.29±0.05	0.881±0.223
Quadratic SVM	99.64±0.02	99.64±0.02	0.996±0.0002	99.64±0.02	0.638±0.034
Cubic SVM	99.62±0.04	99.62±0.04	0.996±0.0004	99.62±0.04	0.645±0.02
Fine Gaussian SVM	98.14±0.09	98.12±0.09	0.981±0.001	98.1±0.1	3.092±0.159
Medium Gaussian SVM*	99.65±0.04	99.65±0.04	0.997±0.0004	99.65±0.04	0.805±0.079
Coarse Gaussian SVM	98.09±0.05	98.08±0.05	0.981±0.0005	98.08±0.05	0.918±0.135
KNN					
Fine KNN	99.59±0.02	99.6±0.02	0.996±0.0002	99.59±0.02	0.239±0.194
Medium KNN	99.39±0.07	99.41±0.06	0.994±0.0007	99.4±0.07	0.186±0.008
Coarse KNN	98.22±0.05	98.2±0.05	0.982±0.0005	98.21±0.05	0.211±0.007
Cosine KNN	99.46±0.07	99.47±0.07	0.995±0.0007	99.47±0.07	0.188±0.019
Cubic KNN	99.35±0.07	99.38±0.07	0.994±0.0007	99.36±0.07	1.478±0.036
Weighted KNN	99.48±0.04	99.5±0.04	0.995±0.0004	99.49±0.04	0.19±0.018
Ensemble/Miscellaneous					
Bagged Trees	99.14±0.1	99.14±0.1	0.991±0.001	99.14±0.1	3.256±0.029
Subspace Discriminant	95.94±0.12	95.94±0.12	0.959±0.0012	95.88±0.12	2.972±0.114
Subspace KNN	99.43±0.06	99.44±0.06	0.994±0.0006	99.43±0.06	3.519±0.136

RUSBoosted Trees	98.25±0.14	98.24±0.15	0.982±0.0015	98.24±0.15	4.644±0.293

*Algorithm with highest recall, precision and accuracy

Figure 4.15: Confusion matrix for Medium Gaussian SVM

4.4 Experiment 4 (using MobiFall's LSM330DLC model and 10-fold cross-validation)

4.4.1 Experiment 4.1 (classifying multiple ADLs using 10 features)

Experiment 4 used the MobiFall Dataset using the accelerometer LSM330DLC inertial module (Samsung Galaxy S3). 9 ADLs were classified in this experiment by using 5-fold cross-validation and a total of 10 features. In this experiment, 23 algorithms were used to produce simulated results shown in Table 4.13. Of all the algorithms given, Fine KNN gave the best performance in terms of training time (0.125 seconds). Cubic SVM gave an accuracy of 87.97% and a training time of 1.305 seconds, and Quadratic SVM has similar with an accuracy of 88.08% and a training time of 1.352 seconds. The confusion matrix for an instance of the experiment done using Fine KNN is also shown in Figure 4.16.

Table 4.13: Results from Experiment 4.1 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Decision Trees					
Fine Tree	72.17±2.58	71.28±2.57	0.71±0.02	76.24±2.23	0.45±1.011
Medium Tree	72.3±4.27	71.61±2.73	0.71±0.03	75.75±1.84	0.083±0.009
Coarse Tree	43.86±2.01	46.6±6.39	0.4±0.03	58.65±1.59	0.075±0.004
Discriminant					
Linear Discriminant	81.48±1.19	81.39±1.29	0.81±0.01	82.78±1.03	0.168±0.256
Naïve Bayes					
Gaussian Naïve Bayes	78.02±1.58	78.25±2.13	0.78±0.02	81.05±1.36	0.206±0.209

Kernel Naïve Bayes	73.94±1.84	79.99±1.95	0.76±0.02	79.17±1.34	1.627±0.235
SVM					
Linear SVM	78.1±2.21	86.21±1.37	0.81±0.02	83.91±1.36	1.626±0.72
Quadratic SVM*	84.52±2.08	89.43±2.02	0.86±0.02	88.08±1.57	1.352±0.147
Cubic SVM	84.47±1.61	89.17±1.51	0.86±0.02	87.97±0.85	1.305±0.021
Fine Gaussian SVM	47.06±1.67	61.11±1.06	0.47±0.02	62.56±1.82	1.262±0.029
Medium Gaussian SVM	82.12±1.33	90.64±1.21	0.85±0.02	87.22±1.03	1.277±0.145
Coarse Gaussian SVM	59.42±0.77	61.16±1.03	0.59±0.01	73.76±0.86	1.208±0.026
KNN					
Fine KNN	81.43±1.61	84.24±1.63	0.82±0.02	85.94±1.45	0.125±0.134
Medium KNN	58.62±1.56	66.24±8.19	0.59±0.02	71.62±2	0.082±0.006
Coarse KNN	33.1±1	29.73±1.26	0.28±0.01	47.07±1.49	0.083±0.004
Cosine KNN	57.3±1.94	62.51±5.49	0.57±0.03	68.72±1.64	0.088±0.031
Cubic KNN	56.27±1.81	59.95±5.43	0.56±0.02	69.21±1.59	0.095±0.016
Weighted KNN	76.59±1.77	85.55±1.47	0.79±0.02	82.37±1.26	0.086±0.015
Ensemble/Miscellaneous					
Boosted Trees (AdaBoost)	75.58±2.44	76.34±2.54	0.76±0.02	81.58±1.48	1.822±0.362
Bagged Trees	74.81±2.19	75.86±2.65	0.75±0.02	78.98±1.4	1.482±0.206
Subspace Discriminant	77.7±1.08	84.55±2.15	0.79±0.01	81.47±0.87	1.525±0.138
Subspace KNN	80.93±2.01	87.36±2.81	0.83±0.02	84.59±2.3	1.398±0.042
RUSBoosted Trees	78.07±2	73.37±1.63	0.74±0.02	78.01±1.28	1.832±0.169

*Algorithm with highest recall, precision and accuracy

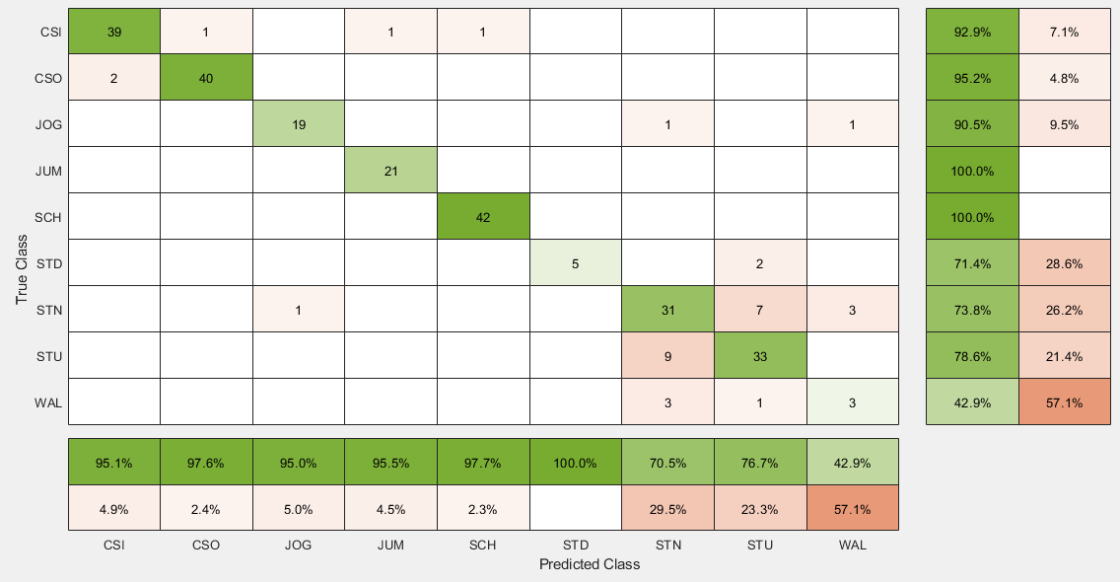


Figure 4.16: Confusion matrix for Fine KNN

4.4.2 Experiment 4.2 (classifying multiple ADLs using 16 features)

In Experiment 4.2, 9 ADLs were classified, and 5-fold cross-validation was used using a total of 16 features. In this experiment, 23 algorithms have been used to produce simulated results shown in Table 4.14. Of all the algorithms given, Fine KNN gave the

best performance in terms of training time with 83.98% accuracy. Cubic SVM gave an accuracy of 88.87% and a training time of 1.327 seconds, and Quadratic SVM has similar with an accuracy of 88.65% and a training time of 1.406 seconds. The confusion matrix for an instance of the experiment done using Fine KNN is also shown in Figure 4.17.

Table 4.14: Results from Experiment 4.2 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Decision Trees					
Fine Tree	34.95±1.06	35.1±1.09	0.35±0.01	76.77±1.17	0.096±0.03
Medium Tree	35.63±1.46	35.61±0.94	0.35±0.01	78.65±1.89	0.081±0.001
Coarse Tree	21.28±0.83	22.86±3.22	0.2±0.01	59.44±0.93	0.08±0.001
Discriminant					
Linear Discriminant	38.7±0.91	39.11±0.68	0.39±0.01	85.71±0.69	0.096±0.031
Naïve Bayes					
Gaussian Naïve Bayes	80.77±1.29	84.2±1.52	0.82±0.01	82.89±1.12	0.179±0.023
Kernel Naïve Bayes	76.46±2.42	82.34±2.6	0.78±0.02	81.05±1.2	2.354±0.047
SVM					
Linear SVM	79.37±1.96	89.28±1.21	0.82±0.02	85.94±1.34	1.452±0.32
Quadratic SVM	84.66±1.91	90.26±1.08	0.87±0.02	88.65±0.75	1.406±0.031
Cubic SVM	83.62±2.35	89.74±1.49	0.86±0.02	88.87±0.73	1.327±0.041
Fine Gaussian SVM	49.29±0.93	62.21±0.69	0.5±0.01	64.51±1.05	1.294±0.029
Medium Gaussian SVM*	83.68±0.92	91.92±1.03	0.86±0.01	89.59±0.79	1.284±0.024
Coarse Gaussian SVM	60.08±1.06	60.14±0.86	0.59±0.01	74.06±1.06	1.261±0.048
KNN					
Fine KNN	75.32±1.34	81.05±1.79	0.77±0.02	83.98±1.07	0.09±0.028
Medium KNN	57.46±1.26	66.56±1.07	0.58±0.01	69.47±1.19	0.085±0.003
Coarse KNN	33.39±0.73	31.93±0.46	0.26±0.01	47.44±1.03	0.085±0.002
Cosine KNN	58.81±1.83	59.85±2.09	0.58±0.02	69.77±1.75	0.086±0.005
Cubic KNN	54.39±1.44	57.33±1.82	0.55±0.02	65.34±1.82	0.093±0.003
Weighted KNN	73.39±1.94	82.99±5.42	0.75±0.02	82.44±1.65	0.084±0.003
Ensemble/Miscellaneous					
Boosted Trees (AdaBoost)	75.69±2.72	77.06±4.09	0.76±0.03	81.2±1.77	1.601±0.029
Bagged Trees	75.53±3.46	80.93±2.9	0.77±0.03	80.15±1.43	1.513±0.173
Subspace Discriminant	81.43±1.44	84.63±1.42	0.82±0.01	84.47±0.94	1.41±0.024
Subspace KNN	74.63±0.99	78.95±1.61	0.76±0.01	78.68±1.31	1.457±0.024
RUSBoosted Trees	81.11±1.78	79.76±1.26	0.79±0.01	80.94±1.36	1.629±0.06

*Algorithm with highest recall, precision and accuracy

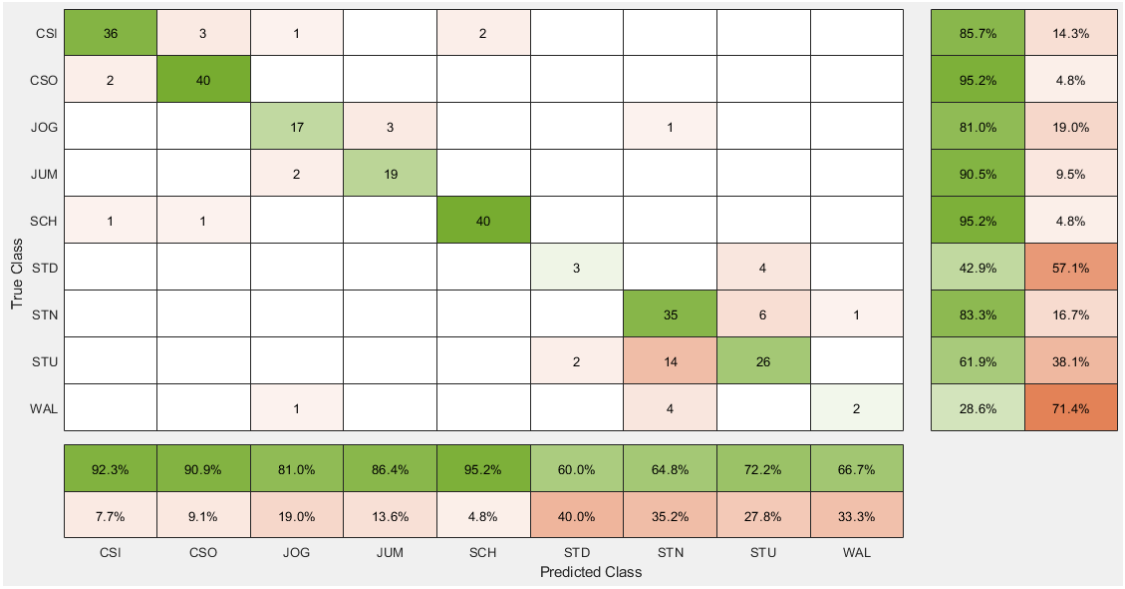


Figure 4.17: Confusion matrix for Fine KNN

4.4.3 Experiment 4.3 (classifying multiple ADLs and Falls using 16 features)

Experiment 4.3 demonstrated the ability of the algorithm to classify both ADLs and Fall Events. 9 ADLs and 4 Fall Events were classified, and a total of 16 features were used. The results are tabulated in Table 4.15.

Of all the algorithms given, Quadratic SVM gave the best performance, and Fine KNN is the most accurate for a shorter training time. Fine KNN gave 81.13% accuracy with a training time of 0.083 seconds, and Quadratic SVM gave 85.89% accuracy with a training time of 2.716 seconds. Cubic SVM had shown similar training time and similar accuracy to Quadratic SVM. The confusion matrix for an instance of the experiment done using Fine KNN is also shown in Figure 4.18.

Table 4.15: Results from Experiment 4.3 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Decision Trees					
Fine Tree	68.71±1.8	69.17±2.93	0.69±0.02	68.83±1.57	0.092±0.012
Medium Tree	62.75±3.34	63.86±3.11	0.62±0.03	62.65±2.31	0.083±0.003
Coarse Tree	31.45±0.84	23.58±1.19	0.25±0.01	36.58±0.67	0.079±0.003
Discriminant					
Linear Discriminant	76.97±2	77.94±1.97	0.77±0.02	77.94±1.06	0.084±0.007
Naïve Bayes					
Gaussian Naïve Bayes	75.09±0.75	77.52±1.46	0.76±0.01	74.01±0.92	0.197±0.006
Kernel Naïve Bayes	74.81±0.65	78.72±1.44	0.76±0.01	76.66±0.59	3.141±0.025
SVM					
Linear SVM	76.9±1.47	85.85±1.53	0.79±0.02	82.11±1.02	2.615±0.054
Quadratic SVM	83.56±0.67	88.18±0.94	0.85±0.01	85.89±0.48	2.767±0.034
Cubic SVM	82.84±1.25	88.09±0.92	0.85±0.01	85.55±0.64	2.716±0.022

Fine Gaussian SVM	43.44±1.19	72.08±1.64	0.48±0.01	53.85±1.22	2.702±0.041
Medium Gaussian SVM*	81.44±1.29	86.26±2.58	0.83±0.02	84.11±0.77	2.63±0.068
Coarse Gaussian SVM	61.28±1.33	63.04±1.33	0.61±0.01	71.54±1.18	2.613±0.019
KNN					
Fine KNN	76.16±0.46	80.47±1.8	0.77±0.01	81.13±0.59	0.083±0.002
Medium KNN	60.7±1.43	65.53±2.17	0.61±0.01	68.77±1.5	0.086±0.003
Coarse KNN	39.04±0.72	37.69±3.62	0.32±0.01	49.17±0.97	0.085±0.003
Cosine KNN	59.62±1.83	61.55±1.86	0.6±0.02	66.98±1.88	0.088±0.005
Cubic KNN	55.78±1.37	58.7±2.91	0.56±0.01	64.68±1.56	0.12±0.004
Weighted KNN	75.11±1.67	82.29±4.4	0.76±0.02	80.63±1.38	0.084±0.004
Ensemble/Miscellaneous					
Boosted Trees (AdaBoost)	69.89±1.68	71.3±2.39	0.7±0.02	71.62±1.54	1.807±0.059
Bagged Trees	76.61±2.35	80.24±2.21	0.78±0.02	78.89±2.08	1.57±0.036
Subspace Discriminant	73.32±1.51	75.77±3.73	0.73±0.02	75.81±1.17	1.439±0.023
Subspace KNN	72.83±1.4	75.47±1.57	0.74±0.01	74.86±0.97	1.475±0.016
RUSBoosted Trees	77.52±1.69	75.61±1.66	0.76±0.01	76.62±2.02	1.816±0.051

*Algorithm with highest recall, precision and accuracy

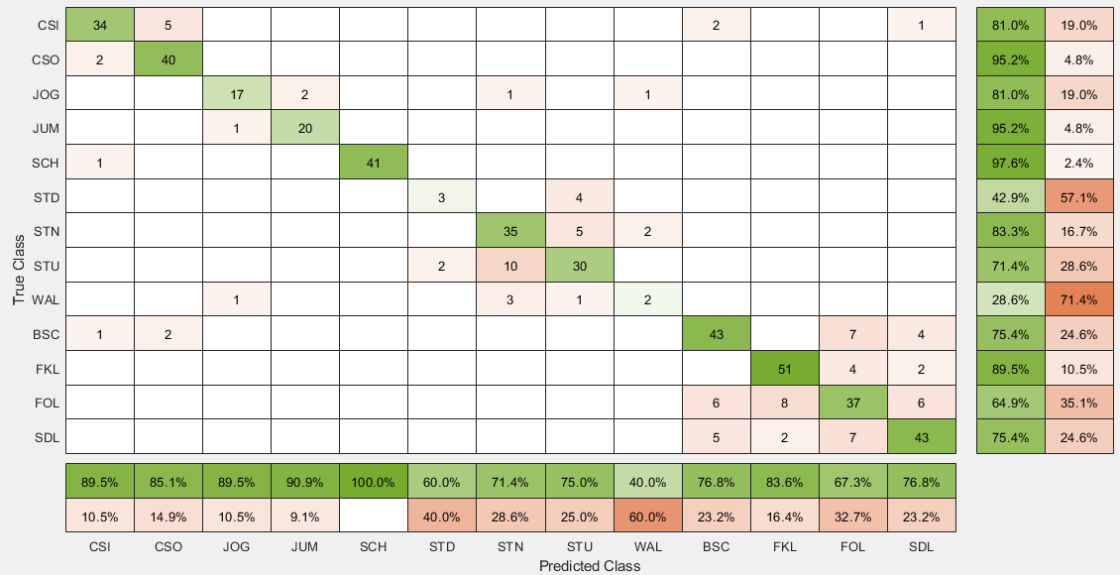


Figure 4.18: Confusion matrix for Fine KNN

4.4.4 Experiment 4.4 (classifying only an instance of ADL and Fall using 16 features)

Experiment 4.4 classified both ADLs and Fall Events as a binary classification. ADLs and Falls were classified using 5-fold cross-validation and a total of 16 features. All the algorithms had been shown to perform well, as shown in Table 4.16.

With a slight advantage, Medium Gaussian SVM was the most accurate and gave 99.13% accuracy with a training time of 0.133 seconds. Boosted Trees had comparatively higher accuracy as well, giving 99.11% accuracy, but with a longer training time of 4.134 seconds. However, all algorithms had shown similar training time and accuracy to each

other compared to other experiments. A confusion matrix for Medium Gaussian SVM is shown in Figure 4.19.

Table 4.16: Results from Experiment 4.4 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Decision Trees					
Fine Tree	94.96±0.8	94.95±0.77	0.95±0.01	94.98±0.77	0.081±0.007
Medium Tree	95.07±0.87	94.99±0.84	0.95±0.01	95.04±0.84	0.082±0.005
Coarse Tree	92.47±1.04	92.79±1	0.93±0.01	92.65±1.01	0.079±0.003
Discriminant					
Linear Discriminant	98.57±0.16	98.66±0.17	0.99±0	98.62±0.16	0.138±0.189
Quadratic Discriminant	98.71±0.03	98.7±0.03	0.99±0	98.71±0.03	0.216±0.022
Naïve Bayes					
Gaussian Naïve Bayes	96.64±0.43	96.44±0.45	0.97±0	96.52±0.45	0.11±0.062
Kernel Naïve Bayes	96.41±0.28	96.16±0.28	0.96±0	96.21±0.29	0.729±0.056
SVM					
Linear SVM	98.82±0.19	98.82±0.23	0.99±0	98.83±0.21	0.173±0.123
Quadratic SVM	98.58±0.32	98.61±0.3	0.99±0	98.6±0.31	0.143±0.009
Cubic SVM	98.45±0.2	98.46±0.2	0.98±0	98.46±0.2	0.138±0.008
Fine Gaussian SVM	93.06±0.73	93.03±0.64	0.93±0.01	92.53±0.78	0.157±0.005
Medium Gaussian SVM*	99.13±0.14	99.12±0.14	0.99±0	99.13±0.14	0.133±0.004
Coarse Gaussian SVM	98.38±0.19	98.32±0.21	0.98±0	98.36±0.2	0.138±0.01
KNN					
Fine KNN	98.55±0.33	98.56±0.35	0.99±0	98.56±0.34	0.162±0.248
Medium KNN	97.29±0.5	97.09±0.53	0.97±0.01	97.17±0.53	0.086±0.008
Coarse KNN	95.19±0.48	95.1±0.5	0.95±0	95.16±0.49	0.087±0.002
Cosine KNN	96.89±0.38	96.67±0.4	0.97±0	96.74±0.41	0.086±0.005
Cubic KNN	96.43±0.51	96.24±0.53	0.96±0.01	96.32±0.53	0.12±0.005
Weighted KNN	98.75±0.13	98.69±0.14	0.99±0	98.72±0.14	0.085±0.005
Ensemble/Miscellaneous					
Boosted Trees (AdaBoost)	99.1±0.12	99.11±0.12	0.99±0	99.11±0.12	4.134±0.35
Bagged Trees	96.4±0.57	96.66±0.53	0.97±0.01	96.54±0.54	1.539±0.192
Subspace Discriminant	98.27±0.23	98.23±0.25	0.98±0	98.26±0.24	1.43±0.279
Subspace KNN	98.21±0.28	98.1±0.28	0.98±0	98.16±0.28	1.541±0.074
RUSBoosted Trees	98.43±0.25	98.43±0.25	0.98±0	98.43±0.25	4.996±0.12

*Algorithm with highest recall, precision and accuracy

Figure 4.19: Confusion matrix for Medium Gaussian SVM

4.5 Brief comparison of Experiments 1 to 4

A summary of the noted experimental results from Table 4.1 to 4.16 can be seen in Table 4.17 for comparison. It was observed that in terms of recall, precision and accuracy, the most suitable classifiers are Cubic SVM, Quadratic SVM, Medium Gaussian SVM and Fine KNN. However, in terms of training time, Fine KNN had shown to have the least time taken for training. Combining both Fine KNN and Cubic KNN for classification can potentially bring further accurate results.

Table 4.17: Comparison of classifiers for given simulations

No. of Experiments	Best Algorithm	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Experiment 1.1	Fine KNN	76.53±1.08	79.04±1.25	0.773±0.012	83.76±0.64	0.184±0.235
Experiment 1.2	Cubic SVM	87.68±0.86	90.21±0.88	0.885±0.009	90.61±0.56	6.216±0.035
Experiment 1.3	Cubic SVM	76.71±0.39	78.6±0.4	0.773±0.004	77.17±0.27	21.87±0.289
Experiment 1.4	Fine KNN	99.72±0.06	99.73±0.06	0.997±0.001	99.72±0.06	0.165±0.114
Experiment 2.1	Fine KNN	77.85±0.52	80.34±0.55	0.7867±0.0053	84.64±0.43	0.172±0.012
Experiment 2.2	Cubic SVM	88.95±0.78	90.89±0.69	0.8964±0.0076	91.58±0.45	13.1848±0.1660
Experiment 2.3	Cubic SVM	77.87±0.58	79.71±0.51	0.7847±0.0057	78.14±0.45	40.426±0.157
Experiment 2.4	Fine KNN	99.71±0.03	99.73±0.03	0.9972±0.0003	99.72±0.03	0.201±0.050
Experiment 3.1	Fine KNN	78.72±0.64	80.82±0.85	0.7946±0.0068	85.23±0.46	0.176±0.033
Experiment 3.2	Cubic SVM	90.52±0.62	91.77±0.56	0.91±0.006	92.08±0.38	13.059±0.364
Experiment 3.3	Cubic SVM	78.62±0.42	80.18±0.32	0.791±0.004	78.43±0.26	42.807±0.424

Experiment 3.4	Medium Gaussian SVM	99.65±0.04	99.65±0.04	0.997±0.0004	99.65±0.04	0.805±0.079
Experiment 4.1	Quadratic SVM	84.52±2.08	89.43±2.02	0.86±0.02	88.08±1.57	1.352±0.147
Experiment 4.2	Medium Gaussian SVM	83.68±0.92	91.92±1.03	0.86±0.01	89.59±0.79	1.284±0.024
Experiment 4.3	Quadratic SVM	83.56±0.67	88.18±0.94	0.85±0.01	85.89±0.48	2.767±0.034
Experiment 4.4	Medium Gaussian SVM	99.13±0.14	99.12±0.14	0.99±0	99.13±0.14	0.133±0.004

4.6 Testing simulation results and comparison with trained results

4.6.1 Experiment 1 (using SisFall's ADXL345 model and 5-fold cross-validation)

4.6.1.1 Experiment 1.1 (classifying multiple ADLs using 10 features)

Previously trained top three results from Experiment 1.1 are shown in Table 4.18. In this experiment Fine KNN had shown the best recall, precision and accuracy of 76.53%, 79.04% and 83.76%, respectively with a training time of 0.184 seconds. Cubic and Weighted KNN also showed good results in comparison with Fine KNN except having higher Training Time of 6.542 seconds.

However, in testing, the results as shown in Table 4.18, Cubic SVM showed the best recall, precision and accuracy of 63.99%, 67.64% and 67.08%, respectively, with Fine KNN showing a lower accuracy of 61.76%. In respect of training time, they had a shorter training time in testing than their corresponding time in training. A confusion matrix for an instance of the experiment done using Fine KNN testing is shown in Figure 4.20.

Table 4.18: Top trained and tested results from Experiment 1.1 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Experiment 1.1 (Training)*					
Cubic SVM	75.59±0.69	77.64±1.15	0.763±0.009	81.27±0.51	6.542±0.047
Fine KNN	76.53±1.08	79.04±1.25	0.773±0.012	83.76±0.64	0.184±0.235
Weighted KNN	73.12±0.98	77.4±1.95	0.736±0.012	82.97±0.84	0.118±0.014
Experiment 1.1 (Testing)**					
Cubic SVM	63.99±0	67.64±0	0.628±0	67.08±0	0.109±0.005
Fine KNN	57.41±0	58.77±0	0.559±0	61.76±0	0.158±0.407

Weighted KNN	58.98±0	55.56±0	0.565±0	66.14±0	0.024±0.054

*In Training 80% of the SisFall dataset was used
**In Testing 20% of the SisFall dataset was used

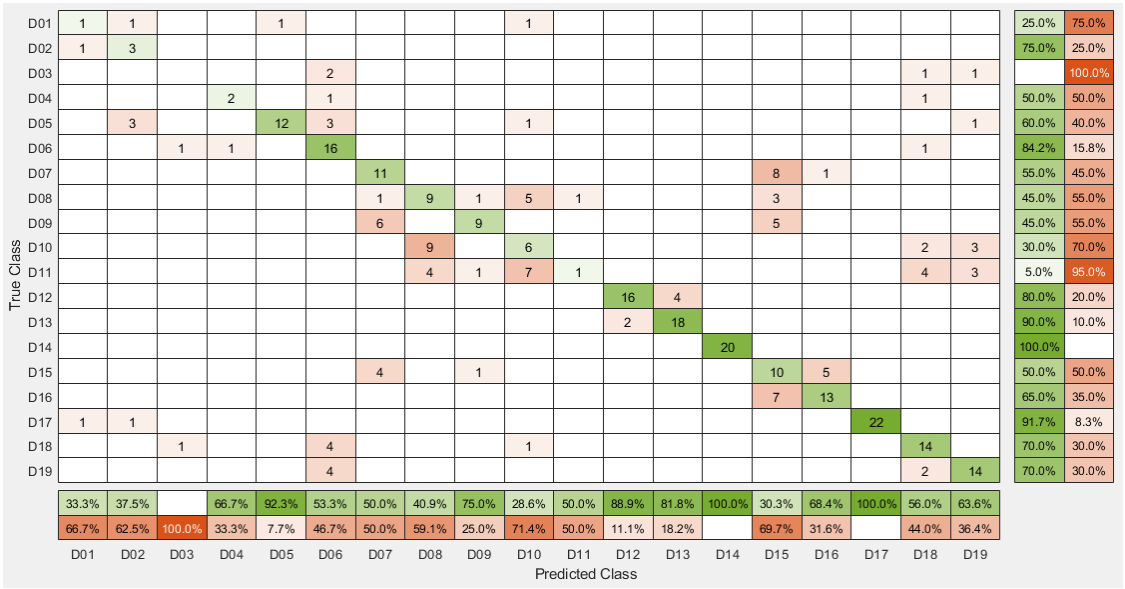


Figure 4.20: Confusion matrix for tested Fine KNN (Experiment 1.1)

4.6.1.2 Experiment 1.2 (classifying multiple ADLs using 16 features)

Similar to Experiment 1.1, 19 ADLs were classified, and 5-fold cross-validation was used in Experiment 1.2. However, a total of 16 features were used. Previously trained results are shown in Table 4.19. In this experiment, the top-trained algorithm with the highest accuracy was Cubic SVM (90.61%), Quadratic SVM (89.87%) and Fine KNN (88.85%), respectively. In terms of faster training time, Fine KNN showed the lowest time of only 0.198 seconds needed to be trained. The testing simulation results of the same algorithm are shown in Table 4.18. In testing, Cubic SVM also showed the best recall, precision and accuracy of 75%, 77.85% and 76.49%, respectively. A confusion matrix for an instance of the experiment done using Cubic SVM for testing is also shown in Figure 4.21. Quadratic SVM also showed similar results of achieving a recall and precision of 73.42% and 75.01%, respectively and overall accuracy of 74.61% with a shorter training time of 0.107 seconds. Fine KNN showed a lower accuracy of 71.47% comparatively but had a faster training time of 0.016 seconds.

Table 4.19: Top trained and tested results from Experiment 1.2 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Experiment 1.2 (Training)*					
Quadratic SVM	87.1±0.96	89.42±0.94	0.879±0.01	89.87±0.63	6.348±0.09
Cubic SVM	87.68±0.86	90.21±0.88	0.885±0.009	90.61±0.56	6.216±0.035
Fine KNN	85.62±0.86	88.95±1.04	0.866±0.009	88.85±0.32	0.198±0.242
Experiment 1.2 (Testing)**					
Quadratic SVM	73.42±0	75.01±0	0.728±0	74.61±0	0.107±0.003
Cubic SVM	75±0	77.85±0	0.748±0	76.49±0	0.109±0.003
Fine KNN	70.79±0	76.47±0	0.712±0	71.47±0	0.016±0.027

*In Training 80% of the SisFall dataset was used
**In Testing 20% of the SisFall dataset was used

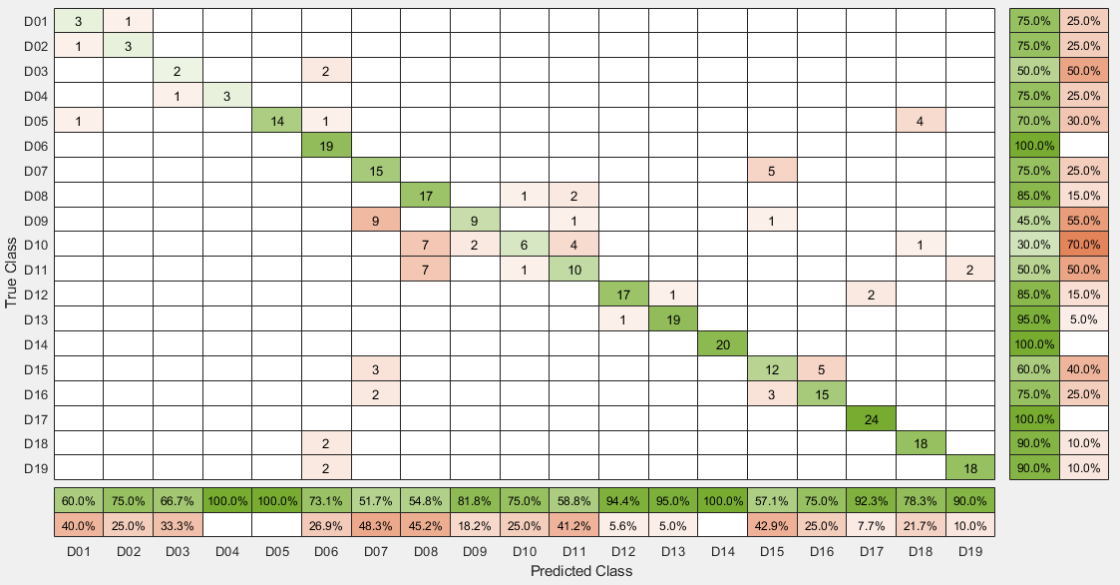


Figure 4.21: Confusion matrix for tested Cubic SVM (Experiment 1.2)

4.6.1.3 Experiment 1.3 (classifying multiple ADLs and Falls using 16 features)

In Experiment 1.3, the studied algorithms were able to classify both 19 ADLs and 15 types of Fall Events together using 5-fold cross-validation. The top three trained and tested results from the simulation of Experiment 1.3 are shown in Table 4.20.

Table 4.20: Top trained and tested results from Experiment 1.3 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Experiment 1.3 (Training)*					
Quadratic SVM	76.72±0.35	78.47±0.42	0.773±0.004	76.85±0.3	21.957±0.19
Cubic SVM	76.71±0.39	78.6±0.4	0.773±0.004	77.17±0.27	21.87±0.29
Fine KNN	71.35±0.4	73.01±0.63	0.718±0.005	71.59±0.33	0.168±0.067
Experiment 1.3 (Testing)**					
Quadratic SVM	63.65±0	64.22±0	0.627±0	62.72±0	0.106±0.001
Cubic SVM	62.9±0	65.46±0	0.627±0	62.56±0	0.106±0.003
Fine KNN	55.96±0	57.61±0	0.558±0	54.29±0	0.006±0.001

*In Training 80% of the SisFall dataset was used
**In Testing 20% of the SisFall dataset was used

In training results, Cubic SVM showed the highest precision (78.6%) and accuracy (77.17%) with a longer training time of 21.87 seconds in comparison with Fine KNN and Quadratic SVM; however, in testing, Quadratic SVM showed the best recall, precision and accuracy of 63.65%, 64.22% and 62.72% respectively. Cubic SVM also showed very similar results of achieving a recall and precision of 62.9% and 65.46%, respectively and overall accuracy of 62.56%. Fine KNN showed a comparatively lower accuracy of 54.29% but had a faster training time of 0.006 seconds. A confusion matrix for an instance of the experiment done using Cubic SVM for testing is shown in Figure 4.22.

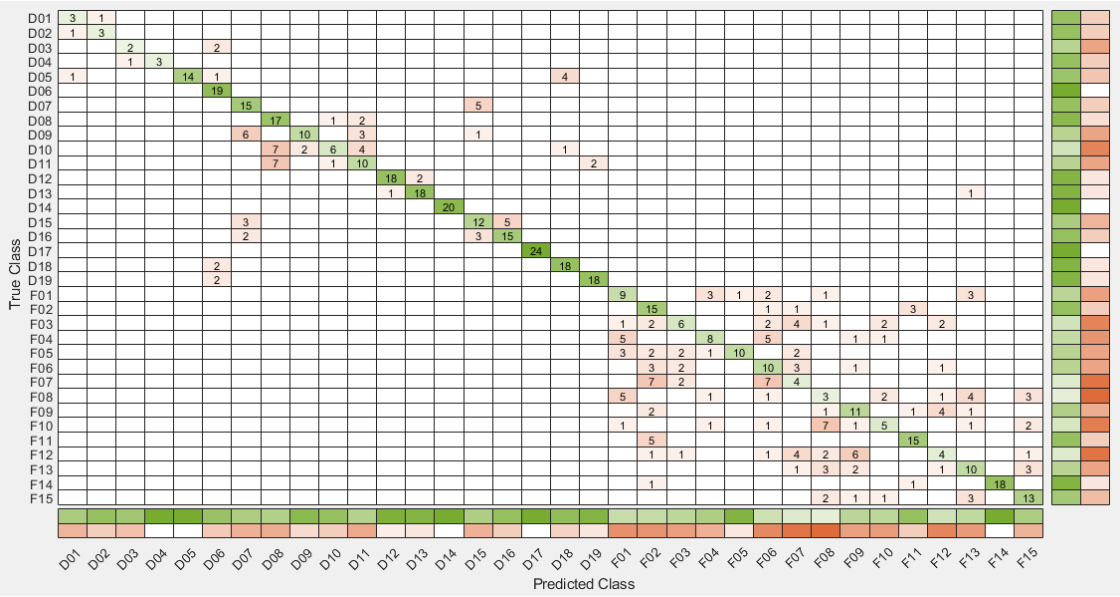


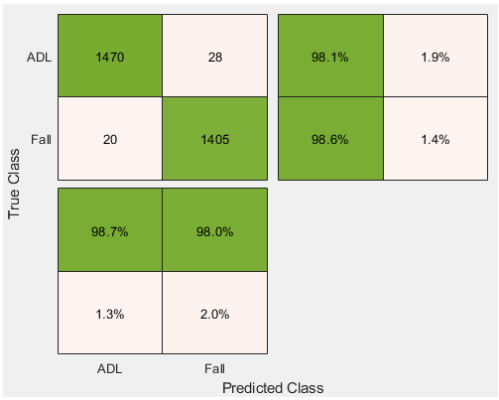
Figure 4.22: Confusion matrix for tested Cubic SVM (Experiment 1.3)

4.6.1.4 Experiment 1.4 (classifying only an instance of ADL and Fall using 16 features) Through this experiment, algorithms were able to classify different types of both ADLs and Fall Events as a binary classification. 5-fold cross-validation and a total of 16 features were used for the classification of ADLs and Falls Events. The top three highest accurate trained and tested results of Experiment 1.4 are shown in Table 4.21. In training, Fine KNN exhibited the highest accuracy (99.72%) with a short training time of 0.165 seconds, whereas in testing, Fine KNN showed 99.19% accuracy with the shorter training time (0.011 seconds). Quadratic SVM was the most accurate at 99.68%, with a training time of 0.004 seconds. Cubic SVM was also similar, having an accuracy of 99.53% and a training time of 0.005 seconds. Overall, all the studied algorithms exhibited higher recall, precision and comparatively shorter training time in testing than their corresponding time in training. A confusion matrix for an instance of the experiment done using Fine KNN for both training and testing is shown in Figure 4.23.

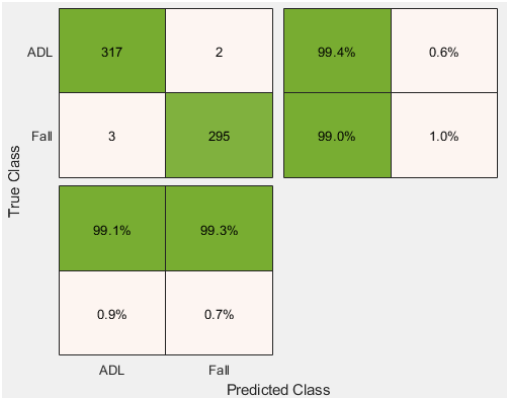
Table 4.21: Top trained and tested results from Experiment 1.4 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Experiment 1.4 (Training)*					
Quadratic SVM	99.67±0.04	99.68±0.04	0.997±0	99.67±0.04	0.35±0.014
Cubic SVM	99.64±0.06	99.64±0.06	0.996±0.001	99.64±0.06	0.354±0.011
Fine KNN	99.72±0.06	99.73±0.06	0.997±0.001	99.72±0.06	0.165±0.114
Experiment 1.4 (Testing)**					
Quadratic SVM	99.66±0	99.69±0	0.997±0	99.68±0	0.004±0.001
Cubic SVM	99.52±0.05	99.53±0.05	0.995±0.001	99.53±0.05	0.005±0.005
Fine KNN	99.18±0	99.19±0	0.992±0	99.19±0	0.011±0.007

*In Training 80% of the SisFall dataset was used
**In Testing 20% of the SisFall dataset was used



Trained Fine KNN



Tested Fine KNN

Figure 4.23: Confusion matrix for trained and tested Fine KNN (Experiment 1.4)

4.6.2 Experiment 2 (using SisFall's ADXL345 model and 10-fold cross-validation)

4.6.2.1 Experiment 2.1 (classifying multiple ADLs using 10 features)

In Experiment 2.1, 10-fold cross-validation was used instead of 5-fold cross-validation for classification, and other parameters were the same as Experiment 1.1. The previously trained top three algorithms with the highest accuracy and the tested results for the same algorithm are shown in Table 4.22. Cubic SVM, Fine and Weighted KNN exhibited the highest percentage of accuracy, precision and recall in training, respectively. Amongst them, Fine KNN was the most accurate (84.64%) and lowest training time (0.172 seconds). Cubic SVM showed higher accuracy in testing, about 67.07% compared to Fine KNN (61.76%) and Weighed KNN (66.16%), respectively. However, Weighted and Fine KNN had a training time of less than 0.1 seconds. A confusion matrix for an instance of the experiment done using Fine KNN for testing is shown in Figure 4.24.

Table 4.22: Top trained and tested results from Experiment 2.1 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Experiment 2.1 (Training)*					
Cubic SVM	77.02±0.49	79.19±0.54	0.7785±0.0051	82.38±0.44	13.583±0.390
Fine KNN	77.85±0.52	80.34±0.55	0.7867±0.0053	84.64±0.43	0.172±0.012
Weighted KNN	74.16±0.74	78.44±0.94	0.7472±0.0073	83.93±0.50	0.197±0.016
Experiment 2.1 (Testing)**					
Cubic SVM	63.99±0	67.64±0	0.628±0	67.08±0	0.111±0.005
Fine KNN	57.41±0	58.77±0	0.559±0	61.76±0	0.006±0.001
Weighted KNN	58.98±0	55.56±0	0.565±0	66.14±0	0.007±0.002

*In Training 80% of the SisFall dataset was used
**In Testing 20% of the SisFall dataset was used

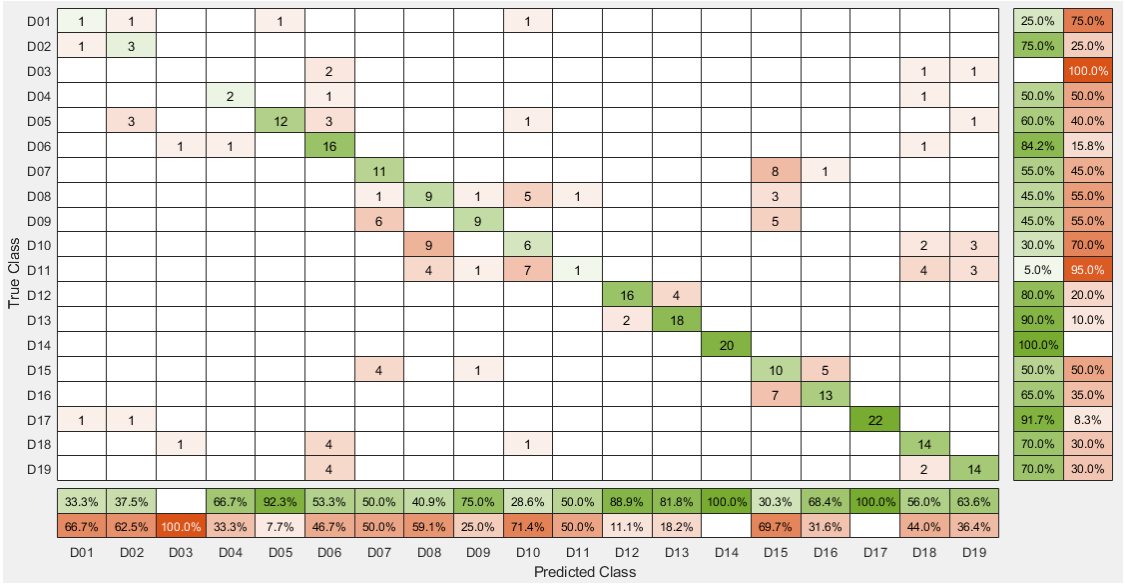


Figure 4.24: Confusion matrix for tested Fine KNN (Experiment 2.1)

4.6.2.2 Experiment 2.2 (classifying multiple ADLs using 16 features)

The test parameters used in Experiment 2.2 and Experiment 2.1 for classification were similar, except 10-fold cross-validation was used. The previously trained results of Experiment 2.2 and tested results for the same algorithms are shown in Table 4.23. In training, Cubic SVM exhibited the highest accuracy (91.58%) than Quadratic SVM (90.53%) and Fine KNN (89.97%), respectively. However, a longer training time was required for Quadratic and Cubic SVM. The same trend was also observed in testing, where Cubic SVM has also shown the best recall, precision and accuracy of 75%, 77.85% and 76.49%, respectively. Quadratic SVM also showed similar results of achieving a recall and precision of 73.42% and 75.01%, respectively and overall accuracy of 74.61% with a shorter training time of 0.106 seconds. Fine KNN showed a comparatively lower accuracy of 71.47% but had a faster training time of 0.007 seconds. A confusion matrix for Cubic SVM for testing is shown in Figure 4.25.

Table 4.23: Top trained and tested results from Experiment 2.2 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Experiment 2.2 (Training)*					
Quadratic SVM	87.92±0.42	89.88±0.30	0.8868±0.0038	90.53±0.39	14.1414±0.3759
Cubic SVM	88.95±0.78	90.89±0.69	0.8964±0.0076	91.58±0.45	13.1848±0.1660
Fine KNN	87.23±0.46	90.62±0.48	0.8821±0.0043	89.97±0.45	0.25±0.20
Experiment 2.2 (Testing)**					
Quadratic SVM	73.42±0	75.01±0	0.728±0	74.61±0	0.106±0.002
Cubic SVM	75±0	77.85±0	0.748±0	76.49±0	0.108±0.002
Fine KNN	70.79±0	76.47±0	0.712±0	71.47±0	0.007±0.001

*In Training 80% of the SisFall dataset was used
**In Testing 20% of the SisFall dataset was used

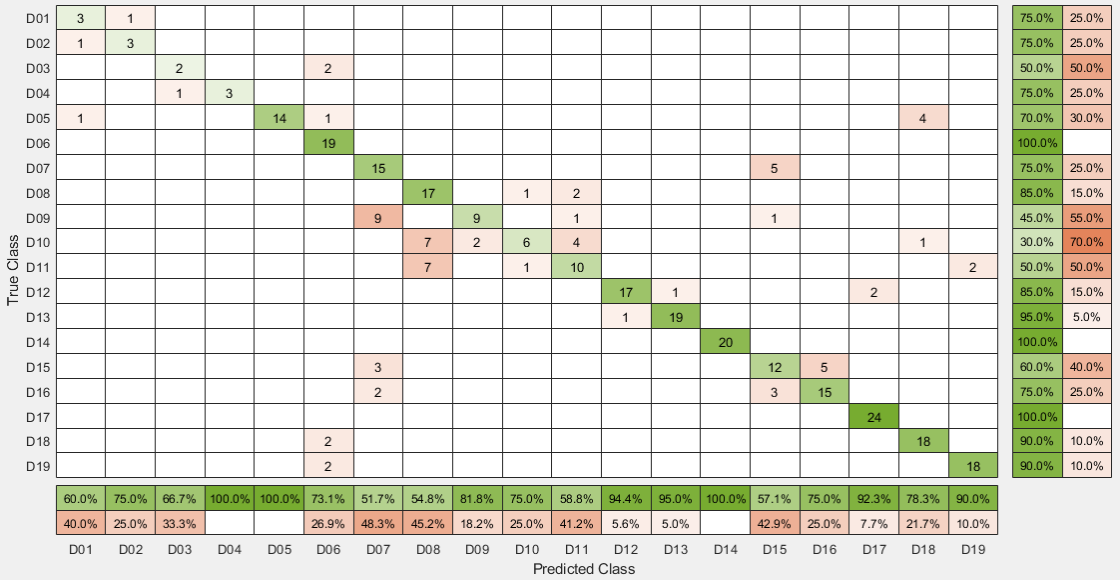


Figure 4.25: Confusion matrix for tested Cubic SVM (Experiment 2.2)

4.6.2.3 Experiment 2.3 (classifying multiple ADLs and Falls using 16 features)

The parameters used in Experiment 2.3 were similar to Experiment 1.3 except for using 10-fold cross-validation. The top three trained algorithms from this experiment were tested, and the results are shown in Table 4.24. In training, Cubic SVM exhibited the highest accuracy (77.87%) than Quadratic (77.75%) and Gaussian SVM (73.93%). In testing, the accuracy was slightly lower than the training; however, Quadratic SVM showed the best recall, precision and accuracy of 63.65%, 64.22% and 62.72%, respectively. Cubic SVM also showed very similar results of achieving a recall and

precision of 62.9% and 65.46%, respectively and overall accuracy of 62.56%, and Medium Gaussian SVM has shown similar results as well with an accuracy of 59.48%. In respect of training time, all the algorithms had the longest training time in training whereas they had a shorter training time in testing. A confusion matrix for Cubic SVM for testing is shown in Figure 4.26.

Table 4.24: Top trained and tested results from Experiment 2.3 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Experiment 2.3 (Training)*					
Quadratic SVM	77.70±0.33	79.13±0.29	0.7818±0.0032	77.75±0.30	40.450±0.250
Cubic SVM	77.87±0.58	79.71±0.51	0.7847±0.0057	78.14±0.45	40.426±0.157
Medium Gaussian SVM	72.97±0.34	75.96±0.33	0.7371±0.0033	73.93±0.31	38.655±0.432
Experiment 2.3 (Testing)**					
Quadratic SVM	63.65±0	64.22±0	0.627±0	62.72±0	0.382±0.004
Cubic SVM	62.9±0	65.46±0	0.627±0	62.56±0	0.384±0.008
Medium Gaussian SVM	59.49±0	60.89±0	0.586±0	59.48±0	0.412±0.008

*In Training 80% of the SisFall dataset was used
**In Testing 20% of the SisFall dataset was used

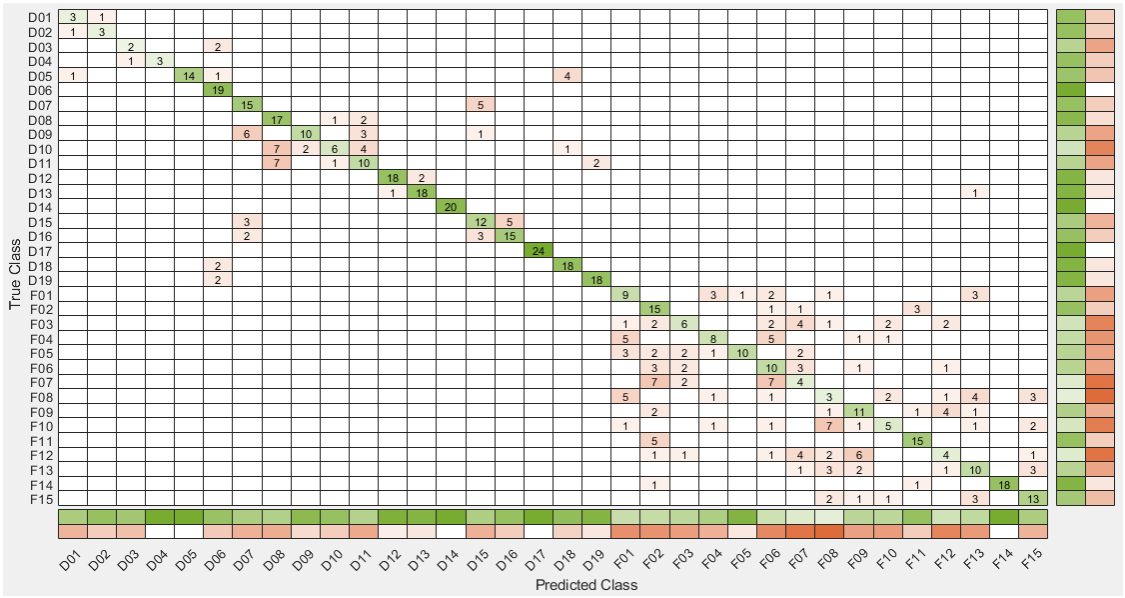


Figure 4.26: Confusion matrix for tested Cubic SVM (Experiment 2.3)

4.6.2.4 Experiment 2.4 (classifying only an instance of ADL and Fall using 16 features)

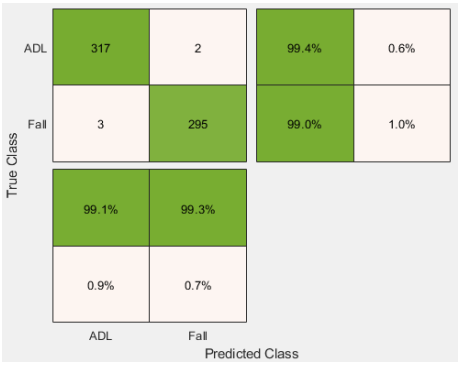
Experiment 2.4 classified both ADLs and Fall Events as a binary classification in a similar way to Experiment 1.4. A total of 16 features and 10-fold cross-validation was used for the classification of different types of ADLs and Fall Events. The trained simulation results of Experiment 2.4 and tested results for Quadratic SVM, Cubic SVM and Fine KNN are presented in Table 4.25. In training, all the investigated algorithms exhibited very good accuracy, recall and precision. The topmost accurate result was for Fine KNN (99.72 %), and the other two also exhibited 99.72% (Quadratic SVM) and 99.67% (Cubic SVM) accuracy, respectively. Fine KNN also had the shortest training time of 0.201 seconds. Testing results (shown in Table 4.25) have also shown very similar results, with Quadratic SVM being the most accurate at 99.68% with a training time of 0.004 seconds. Cubic SVM was also similar, having an accuracy of 99.5% and a similar training time of 0.005 seconds. Fine KNN is most accurate for shorter training time and gave 99.19% accuracy with a training time of 0.009 seconds. A confusion matrix for Cubic SVM for both training and testing is shown in Figure 4.27.

Table 4.25: Top trained and tested results from Experiment 2.4 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Experiment 2.4 (Training)*					
Quadratic SVM	99.69±0.07	99.70±0.07	0.9970±0.0007	99.70±0.07	0.702±0.029
Cubic SVM	99.67±0.04	99.68±0.04	0.9967±0.0004	99.67±0.04	0.700±0.021
Fine KNN	99.71±0.03	99.73±0.03	0.9972±0.0003	99.72±0.03	0.201±0.050
Experiment 2.4 (Testing)**					
Quadratic SVM	99.66±0	99.69±0	0.997±0	99.68±0	0.004±0
Cubic SVM	99.49±0.05	99.5±0.05	0.995±0.001	99.5±0.05	0.005±0.002
Fine KNN	99.18±0	99.19±0	0.992±0	99.19±0	0.009±0.001

*In Training 80% of the SisFall dataset was used
**In Testing 20% of the SisFall dataset was used

Trained Fine KNN



Tested Fine KNN

Figure 4.27: Confusion matrix for trained and tested Fine KNN (Experiment 2.4)

4.6.3 Experiment 3 (using SisFall's MMA8451Q model and 10-fold cross-validation)

4.6.3.1 Experiment 3.1 (classifying multiple ADLs using 10 features)

In Experiment 3, the MMA8451Q accelerometer was used instead of the ADXL345 accelerometer, and the other parameters were the same as with Experiment 2. In previously trained results, the most accurate algorithm was Cubic SVM, Fine and Weighed KNN. Amongst them, Fine KNN showed the highest accuracy (85.23%). Cubic SVM and Weighed KNN also exhibited almost similar accuracy of 84.01% and 84.11%, respectively. In respect of training time, Weighed KNN had the shortest training time (0.169 seconds), whereas Cubic SVM had the longest training time (11.973 seconds). The trained and tested simulation results of Experiment 3.1 are shown in Table 4.26. However, it was observed in testing, Cubic SVM showed the best recall, precision and accuracy of 63.06%, 61.53% and 64.89%, respectively, with Fine KNN showing slightly lower values comparatively with 63.64% accuracy, but having a faster training time of 0.006 seconds. A confusion matrix for an instance of the experiment done using Fine KNN for testing is shown in Figure 4.28.

Table 4.26: Top trained and tested results from Experiment 3.1 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Experiment 3.1 (Training)*					
Cubic SVM	78.58±0.55	81.23±0.82	0.7961±0.0062	84.01±0.53	11.973±0.076
Fine KNN	78.72±0.64	80.82±0.85	0.7946±0.0068	85.23±0.46	0.176±0.033
Weighted KNN	75.21±0.80	80.97±1.65	0.7622±0.0089	84.11±0.59	0.169±0.009
Experiment 3.1 (Testing)**					

Cubic SVM	63.06±0	61.53±0	0.608±0	64.89±0	0.105±0.001
Fine KNN	62.02±0	63.75±0	0.609±0	63.64±0	0.006±0.001
Weighted KNN	59.42±0	61.53±0	0.587±0	63.01±0	0.007±0.001

*In Training 80% of the SisFall dataset was used
**In Testing 20% of the SisFall dataset was used

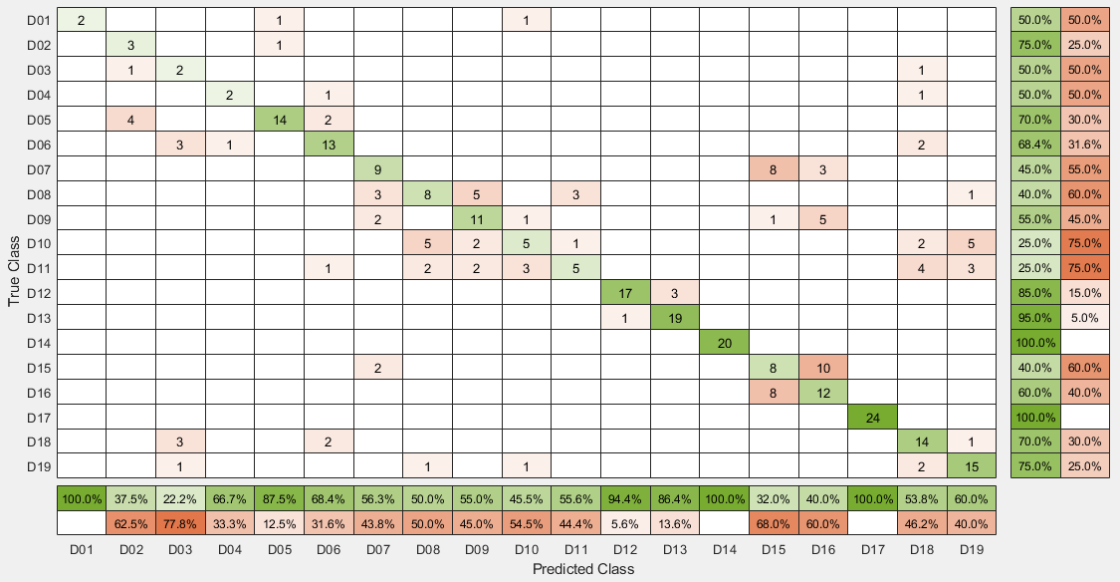


Figure 4.28: Confusion matrix for tested Fine KNN (Experiment 3.1)

4.6.3.2 Experiment 3.2 (classifying multiple ADLs using 16 features)

The parameters used for Experiment 3.2 and Experiment 2.2 were the same, except the MMA8451Q accelerometer was used. In trained results, Quadratic SVM, Cubic SVM and Fine were the topmost accurate algorithms and exhibited 91.7%, 92.08% and 90.21% accuracy, respectively. Fine KNN had the shortest training time of 0.243 seconds, and Quadratic and Cubic SVM had a longer training time. The tested and trained simulation results of Experiment 3.2 are shown in Table 4.27.

In testing, Cubic SVM also showed the best recall, precision and accuracy of 72.37%, 74.65% and 74.61%, respectively. Quadratic SVM also showed similar results of achieving a recall and precision of 71.32% and 73.37%, respectively and overall accuracy of 73.35% with a shorter training time of 0.107 seconds. Fine KNN showed the worst results comparatively with an accuracy of 64.89% but having a faster training time of 0.006 seconds. A confusion matrix for an instance of the experiment done using Cubic SVM for testing is shown in Figure 4.29.

Table 4.27: Top trained and tested results from Experiment 3.2 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Experiment 3.2 (Training)*					
Quadratic SVM	89.63±0.52	91.15±0.46	0.902±0.005	91.7±0.44	12.955±0.18
Cubic SVM	90.52±0.62	91.77±0.56	0.91±0.006	92.08±0.38	13.059±0.364
Fine KNN	88.49±0.3	90.18±0.38	0.892±0.003	90.21±0.23	0.243±0.222
Experiment 3.2 (Testing)**					
Quadratic SVM	71.32±0	73.37±0	0.715±0	73.35±0	0.107±0.004
Cubic SVM	72.37±0	74.65±0	0.724±0	74.61±0	0.108±0.003
Fine KNN	63.16±0	64.57±0	0.629±0	64.89±0	0.006±0.001

*In Training 80% of the SisFall dataset was used
**In Testing 20% of the SisFall dataset was used

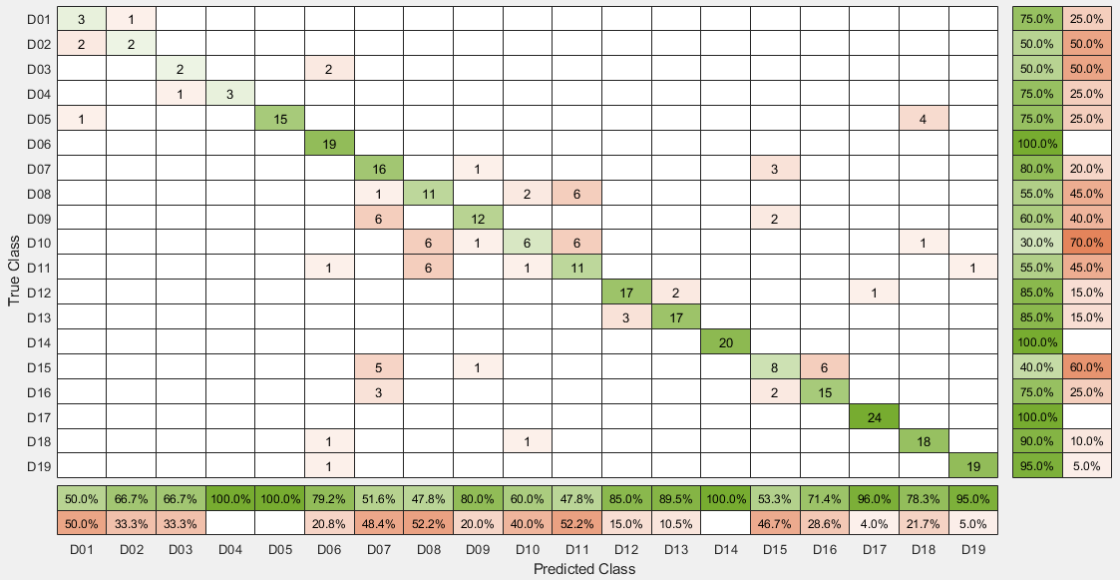


Figure 4.29: Confusion matrix for tested Cubic SVM (Experiment 3.2)

4.6.3.3 Experiment 3.3 (classifying multiple ADLs and Falls using 16 features)

In Experiment 3.3, the MMA8451Q accelerometer was used, and the other parameters were the same as Experiment 2.3. Experiment 3.3 was able to classify both ADLs and 15 Falls via a total of 16 features. In previously trained results, Cubic SVM, Quadratic SVM and Medium Gaussian SVM showed the highest accuracy of 78.43%, 78.42% and 74.69%, respectively. All three also showed higher recall, precision and a longer training time. However, in terms of faster training time (less than one second), Fine KNN was the most accurate having an accuracy of 73.11% in only 0.254 seconds (Table 4.11). In testing,

Quadratic SVM showed the best recall, precision and accuracy of 61.59%, 63.97% and 61.75%, respectively. Cubic SVM also showed very similar results of achieving a recall and precision of 59.64% and 62.19%, respectively and an overall accuracy of 59.64%. All three had the longest training time of about 0.388 to 0.42 seconds. The results of tested and trained algorithms are shown in Table 4.28. The confusion matrix for an instance of the experiment done using Cubic SVM for testing is shown in Figure 4.30.

Table 4.28: Top trained and tested results from Experiment 3.3 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Experiment 3.3 (Training)*					
Quadratic SVM	78.43±0.3	79.94±0.36	0.788±0.003	78.42±0.26	43.028±0.67
Cubic SVM	78.62±0.42	80.18±0.32	0.791±0.004	78.43±0.26	42.807±0.424
Medium Gaussian SVM	74.24±0.41	77.14±0.36	0.75±0.004	74.69±0.44	38.172±0.336
Experiment 3.3 (Testing)**					
Quadratic SVM	61.59±0	63.97±0	0.618±0	61.75±0	0.39±0.015
Cubic SVM	59.64±0	62.19±0	0.596±0	59.64±0	0.388±0.007
Medium Gaussian SVM	57.72±0	60.15±0	0.573±0	58.18±0	0.42±0.019

*In Training 80% of the SisFall dataset was used
**In Testing 20% of the SisFall dataset was used

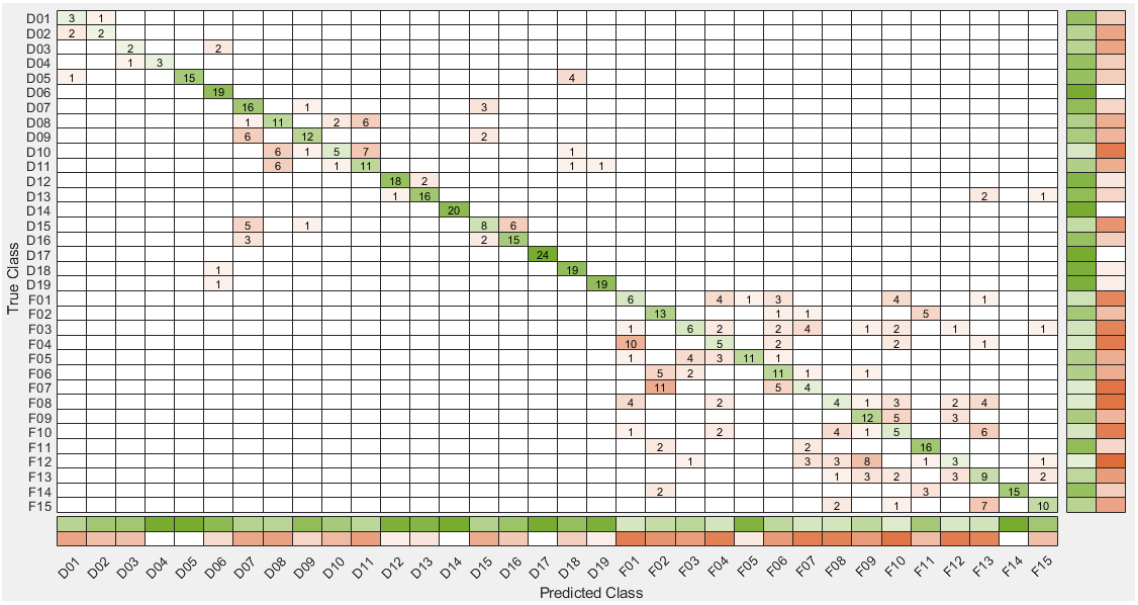


Figure 4.30: Confusion matrix for tested Cubic SVM (Experiment 3.3)

4.6.3.4 Experiment 3.4 (classifying only an instance of ADL and Fall using 16 features) Experiment 3.4 shows the ability of the algorithm to classify both ADLs and Fall Events as binary classification. The parameters used for Experiment 2.4 and Experiment 3.4 were similar, and in this experiment, a total of 16 features and 10-fold cross-validation were used for the classification of different types of ADLs and Fall Events. The previously trained results showed the top three accurate algorithms as Quadratic, Cubic and Medium Gaussian SVM, respectively, and they exhibited about 99.62 to 99.65% accuracy. In a similar fashion, all three algorithms exhibited the highest recall and precision. They had a shorter training time, less than 1 second. Fine KNN was also most accurate in terms of shorter training time and gave 99.59% accuracy with a training time of 0.239 seconds (Table 4.12). The trained and tested parameters of Experiment 3.4 are shown in Table 4.29. Testing results have shown very similar results, with Medium Gaussian SVM being the most accurate at 99.84% with a training time of 0.004 seconds. The confusion matrix for an instance of the experiment done using Medium Gaussian SVM for testing is shown in Figure 4.31. Quadratic SVM and Cubic SVM were also similar, having an accuracy of 99.71% and 99.68%, respectively.

Table 4.29: Top trained and tested results from Experiment 3.4 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Experiment 3.4 (Training)*					
Quadratic SVM	99.64±0.02	99.64±0.02	0.996±0.0002	99.64±0.02	0.638±0.034
Cubic SVM	99.62±0.04	99.62±0.04	0.996±0.0004	99.62±0.04	0.645±0.02
Medium Gaussian SVM	99.65±0.04	99.65±0.04	0.997±0.0004	99.65±0.04	0.805±0.079
Experiment 3.4 (Testing)**					
Quadratic SVM	99.71±0.07	99.71±0.07	0.997±0.001	99.71±0.07	0.004±0.001
Cubic SVM	99.68±0.15	99.67±0.15	0.997±0.002	99.68±0.15	0.004±0.001
Medium Gaussian SVM	99.84±0	99.83±0	0.998±0	99.84±0	0.004±0

*In Training 80% of the SisFall dataset was used
**In Testing 20% of the SisFall dataset was used

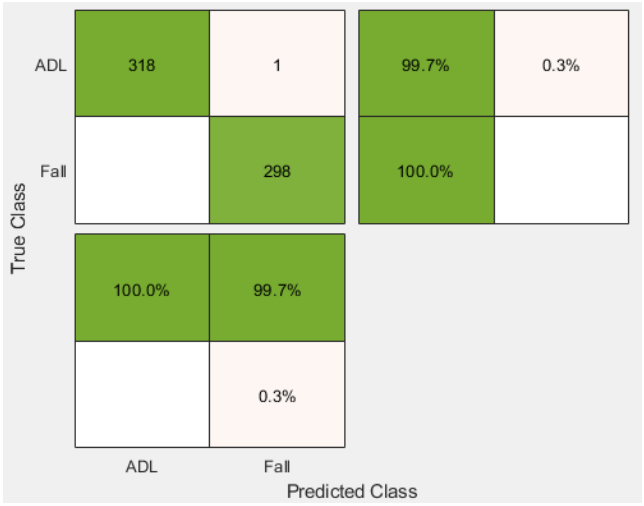


Figure 4.31: Confusion matrix for tested Cubic SVM (Experiment 3.4)

4.6.4 Experiment 4 (using MobiFall's LSM330DLC model and 10-fold cross-validation)

4.6.4.1 Experiment 4.1 (classifying multiple ADLs using 10 features)

Experiment 4 is done using the MobiFall dataset and accelerometer LSM330DLC inertial module (Samsung Galaxy S3). In this experiment, 9 ADLs were classified by using 5-fold cross-validation. From the previously trained results of Experiment 4.1, the top three accurate algorithms were Quadratic, Cubic and Gaussian SVM with 87.22% to 88.08% accuracy. In testing, the Medium Gaussian SVM algorithm was the best overall in respect of accuracy (69.74%), precision (70.94%) and recall (72.22%) values. Cubic SVM showed lower results (53.95% accuracy) in comparison. The topmost trained and tested simulation results of Experiment 4.1 are shown in Table 4.30. A confusion matrix for an instance of the experiment done using Medium Gaussian SVM for testing is shown in Figure 4.32.

Table 4.30: Top trained and tested results from Experiment 4.1 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Experiment 4.1 (Training)*					
Quadratic SVM	84.52±2.08	89.43±2.02	0.86±0.02	88.08±1.57	1.352±0.147
Cubic SVM	84.47±1.61	89.17±1.51	0.86±0.02	87.97±0.85	1.305±0.021
Medium Gaussian SVM	82.12±1.33	90.64±1.21	0.85±0.02	87.22±1.03	1.277±0.145
Experiment 4.1 (Testing)**					

Quadratic SVM	65.74±0	71.12±0	0.638±0	65.79±0	0.024±0.002
Cubic SVM	56.48±0	68.12±0	0.544±0	53.95±0	0.023±0.001
Medium Gaussian SVM	72.22±0	70.94±0	0.693±0	69.74±0	0.024±0.001

*In Training 80% of the MobiFall dataset was used
**In Testing 20% of the MobiFall dataset was used

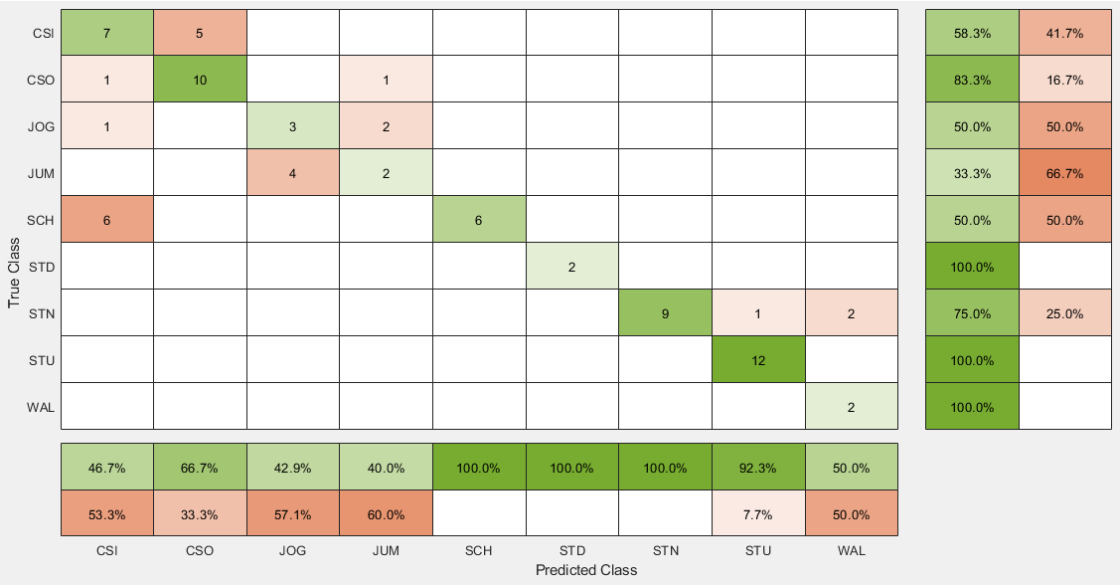


Figure 4.32: Confusion matrix for tested Medium Gaussian SVM (Experiment 4.1)

4.6.4.2 Experiment 4.2 (classifying multiple ADLs using 16 features)

Experiment 4.2 used the same parameters as Experiment 2.2 as well. Previously trained top three algorithms with the highest accuracy and the tested simulation results for the same algorithm are presented in Table 4.31. In testing, Medium Gaussian SVM showed the best recall, precision and accuracy of 72.22%, 75.87% and 72.37%, respectively, in comparison with the trained results. Quadratic SVM also showed similar results of achieving a recall and precision of 69.44% and 70.9%, respectively and overall accuracy of 69.74% with a shorter training time. The confusion matrix for an instance of the experiment done using Medium Gaussian SVM for testing is shown in Figure 4.33.

Table 4.31: Top trained and tested results from Experiment 4.2 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Experiment 4.2 (Training)*					
Quadratic SVM	84.66±1.91	90.26±1.08	0.87±0.02	88.65±0.75	1.406±0.031

Cubic SVM	83.62±2.35	89.74±1.49	0.86±0.02	88.87±0.73	1.327±0.041
Medium Gaussian SVM	83.68±0.92	91.92±1.03	0.86±0.01	89.59±0.79	1.284±0.024
Experiment 4.2 (Testing)**					
Quadratic SVM	69.44±0	70.9±0	0.667±0	69.74±0	0.023±0.001
Cubic SVM	67.59±0	70.9±0	0.651±0	67.11±0	0.023±0.001
Medium Gaussian SVM	72.22±0	75.87±0	0.694±0	72.37±0	0.025±0.001

*In Training 80% of the MobiFall dataset was used
**In Testing 20% of the MobiFall dataset was used

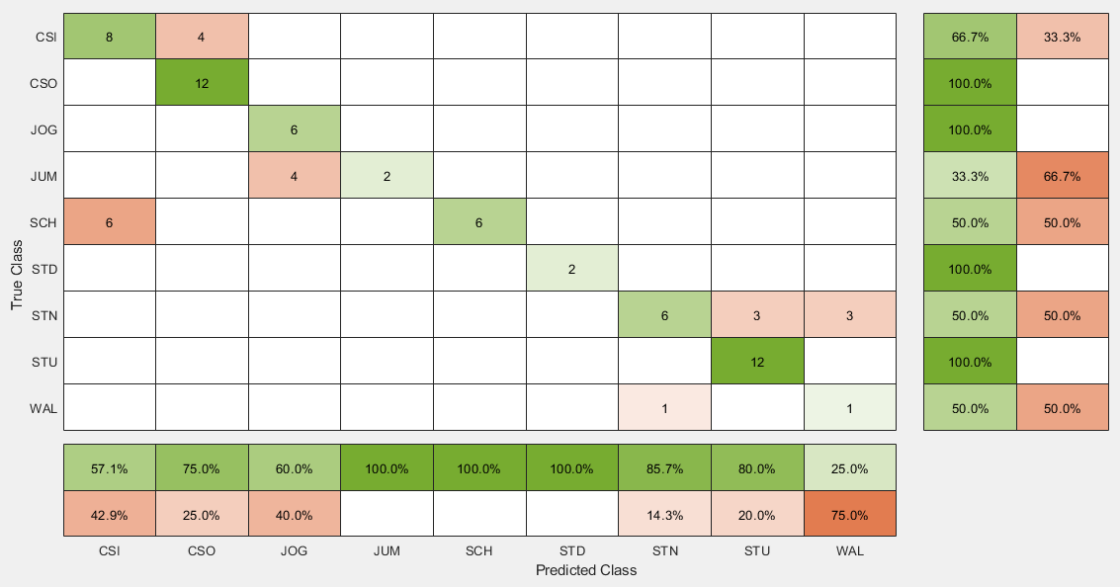


Figure 4.33: Confusion matrix for tested Medium Gaussian SVM (Experiment 4.2)

4.6.4.3 Experiment 4.3 (classifying multiple ADLs and Falls using 16 features)

Previously trained top three results for Experiment 4.3 are shown in Table 4.32. The confusion matrix for an instance of the experiment done using Medium Gaussian SVM for testing is shown in Figure 4.34. In testing (shown in Table 4.31), Medium Gaussian SVM showed the best accuracy at 77.94%. In this case, Quadratic SVM showed very similar results of achieving a recall and precision of 75% and 78%, respectively and overall accuracy of 76.47%.

Table 4.32: Top trained and tested results from Experiment 4.3 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Experiment 4.3 (Training)*					
Quadratic SVM	83.56±0.67	88.18±0.94	0.85±0.01	85.89±0.48	2.767±0.034

Cubic SVM	82.84±1.25	88.09±0.92	0.85±0.01	85.55±0.64	2.716±0.022
Medium Gaussian SVM	81.44±1.29	86.26±2.58	0.83±0.02	84.11±0.77	2.63±0.068
Experiment 4.3 (Testing)**					
Quadratic SVM	75±0	78.34±0	0.73±0	76.47±0	0.049±0.002
Cubic SVM	71.92±0	74.99±0	0.704±0	73.53±0	0.051±0.002
Medium Gaussian SVM	76.28±0	79.05±0	0.743±0	77.94±0	0.058±0.011

*In Training 80% of the MobiFall dataset was used
**In Testing 20% of the MobiFall dataset was used

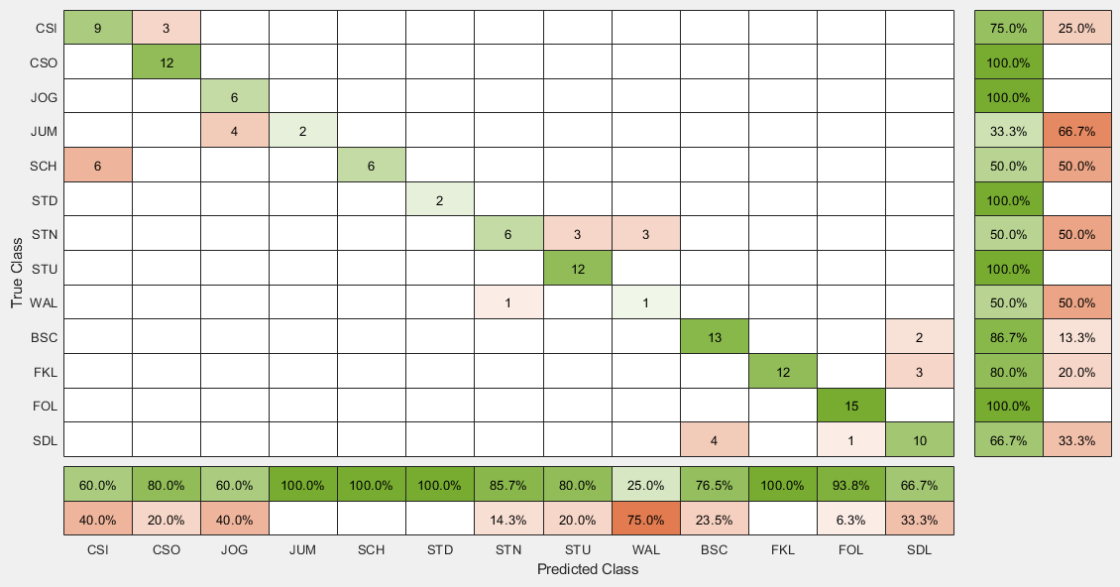


Figure 4.34: Confusion matrix for tested Medium Gaussian SVM (Experiment 4.3)

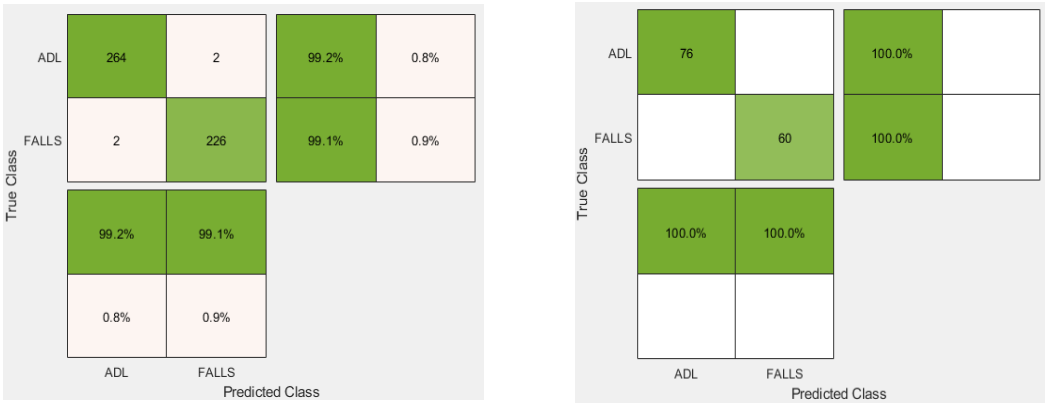
4.6.4.4 Experiment 4.4 (classifying only an instance of ADL and Fall using 16 features)

Experiment 4.4 shows the ability of the algorithm to classify both ADLs and Fall Events as binary classification. In this experiment, different types of ADLs and Falls were classified using 5-fold cross-validation, and a total of 16 features were used. In training, Linear SVM, Medium Gaussian SVM and Boosted Trees had shown shorter training time and highest accuracy (98.83% to 99.13%) in comparison to other experiments. The top three results from the Experiment 4.4 are shown in Table 4.33. Testing results have shown very similar results, with Medium Gaussian SVM being the most accurate at 100% with a shorter training time. However, Boosted Trees had shown a very low accuracy of 51.47%. The confusion matrix for an instance of the experiment done using Medium Gaussian SVM for both training and testing is shown in Figure 4.35.

Table 4.33: Top trained and tested results from Experiment 4.4 (mean ± SD; n = 10)

ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
Experiment 4.4 (Training)*					
Linear SVM	98.82±0.19	98.82±0.23	0.99±0	98.83±0.21	0.173±0.123
Medium Gaussian SVM	99.13±0.14	99.12±0.14	0.99±0	99.13±0.14	0.133±0.004
Boosted Trees (AdaBoost)	99.1±0.12	99.11±0.12	0.99±0	99.11±0.12	4.134±0.35
Experiment 4.4 (Testing)**					
Linear SVM	97.19±0	96.9±0	0.97±0	97.06±0	0.005±0.002
Medium Gaussian SVM	100±0	100±0	1±0	100±0	0.026±0.066
Boosted Trees (AdaBoost)	46.05±0	41.18±0	0.435±0	51.47±0	0.037±0.005

*In Training 80% of the MobiFall dataset was used
**In Testing 20% of the MobiFall dataset was used



Trained Medium Gaussian SVM Tested Medium Gaussian SVM

Figure 4.35: Confusion matrix for trained and tested Medium Gaussian SVM (Experiment 4.4)

4.7 Results achieved using the ensemble classifier

The available classifier systems were combined in an ensemble classifier. This system combines output from various algorithms and re-classifies between the classes to improve recall, precision and accuracy. For SisFall dataset, 10-fold cross-validation is used in this case. In particular, the Bagged Decision Tree method is used for this model. As can be seen in Table 4.34, the Ensemble Classifier can provide up to 99.62% accuracy. For results taken from inputs from the ADXL345 accelerometer, Figure 4.36 shows the confusion matrix of a trained stacked ensemble of Med. Gaussian. Cubic and Quadratic

SVM, Fine KNN and Decision Tree. Figure 4.37 shows the confusion matrix of the tested stacked ensemble. Similarly, Figure 4.38 shows the confusion matrix of the trained stacked ensemble of Cubic SVM and Fine KNN, while Figure 4.39 shows the confusion matrix of the tested stacked ensemble classifier. As shown in Table 4.34, decreasing the number of base models for the ensemble system resulted in a slight decrease in the recall, precision and accuracy. However, a slight decrease in training time was recorded as well.

Similarly, for results taken from inputs from the MMA8451Q accelerometer, Figure 4.40 shows the confusion matrix of a trained stacked ensemble of Med. Gaussian. Cubic and Quadratic SVM, Fine KNN and Decision Tree, while Figure 4.41 shows the confusion matrix of the tested stacked ensemble. Figure 4.42 shows the confusion matrix for a trained stacked ensemble of Cubic SVM and Fine KNN, and Figure 4.43 shows the confusion matrix tested stacked ensemble classifier. Similar to the results from the ADXL345 accelerometer model, the MMA845Q model results had shown a slight decrease in the recall, precision and accuracy in decreasing the number of base models (comparing No. 4 with No. 5). However, using Med. Gaussian. and Cubic SVM had shown similar results to No. 4 with an additional decrease in training time as well.

Table 4.34: Trained results from SisFall dataset

No.	ML Model	Recall (%)	Precision (%)	F1 Score	Accuracy (%)	Training Time (sec)
ADXL345						
1	SVMs, KNN, Decision Tree, Naïve Bayes	99.62	99.63	0.996	99.62	10.34
2	SVMs and KNN	99.49	99.49	0.995	99.49	9.25
3	SVMs	99.45	99.45	0.995	99.45	9.64
MMA8451Q						
4	SVMs, KNN, Decision Tree, Naïve Bayes	99.42	99.42	0.994	99.42	10.38
5	SVMs and KNN	99.39	99.38	0.994	99.38	9.68
6	SVMs	99.42	99.42	0.994	99.42	9.57

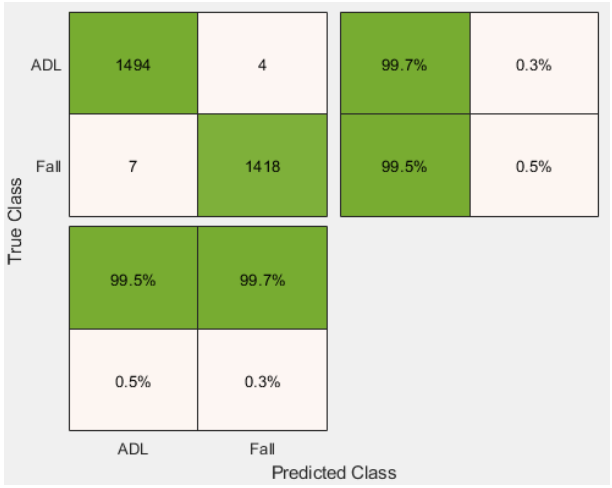


Figure 4.36: Confusion matrix trained stacked ensemble (No.1)

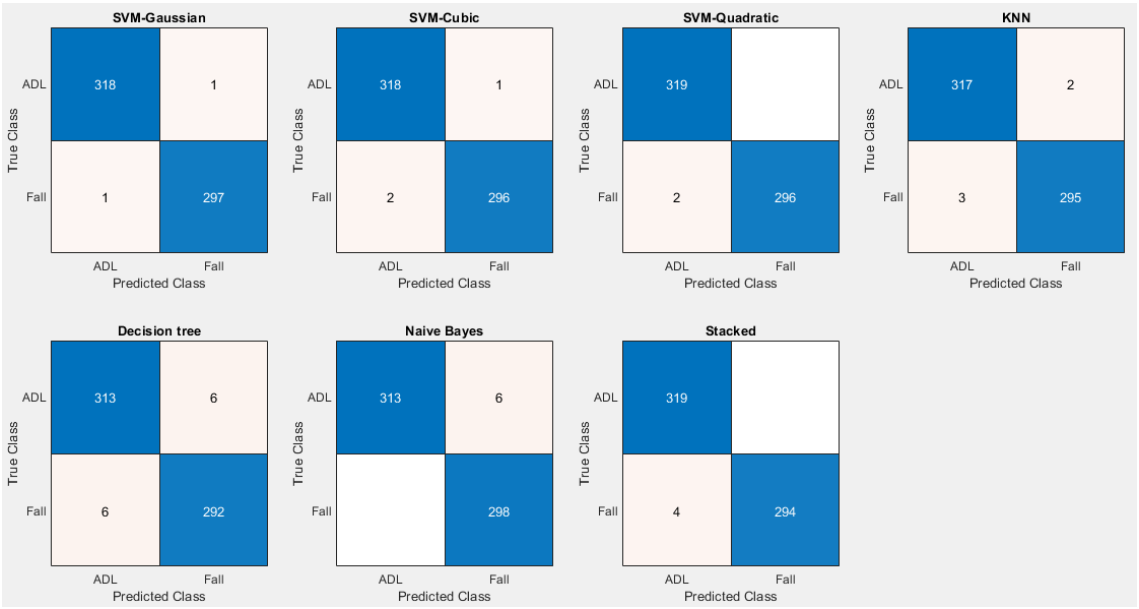


Figure 4.37: Confusion matrix of tested stacked ensemble (No.1)

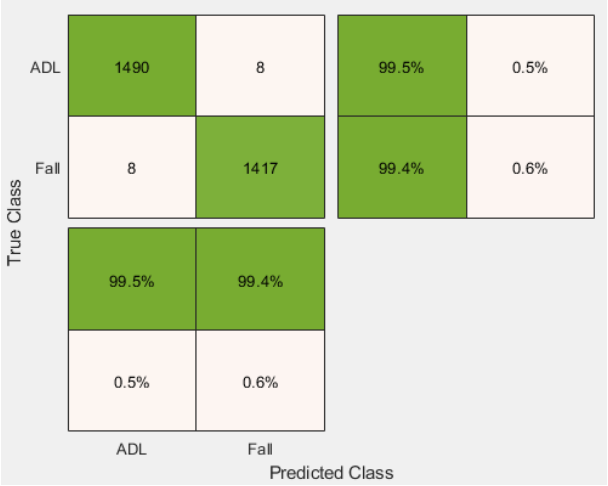


Figure 4.38: Confusion matrix of trained stacked ensemble (No.3)

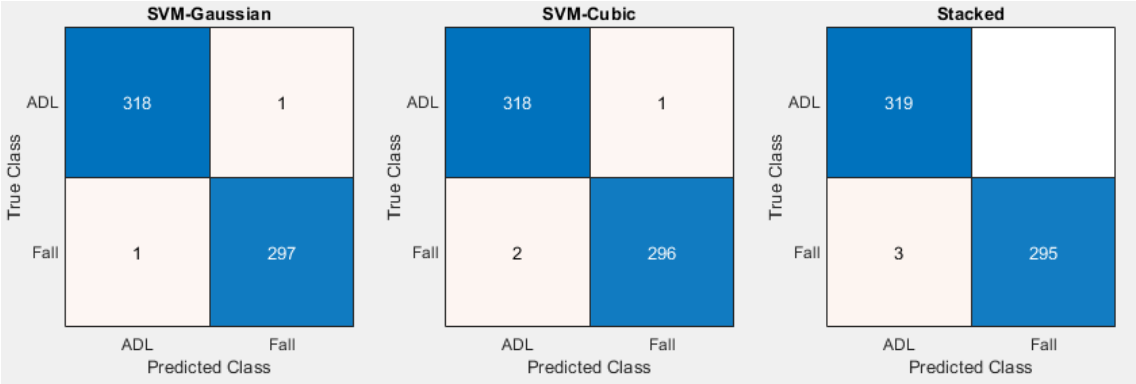


Figure 4.39: Confusion matrix of tested stacked ensemble (No.3)

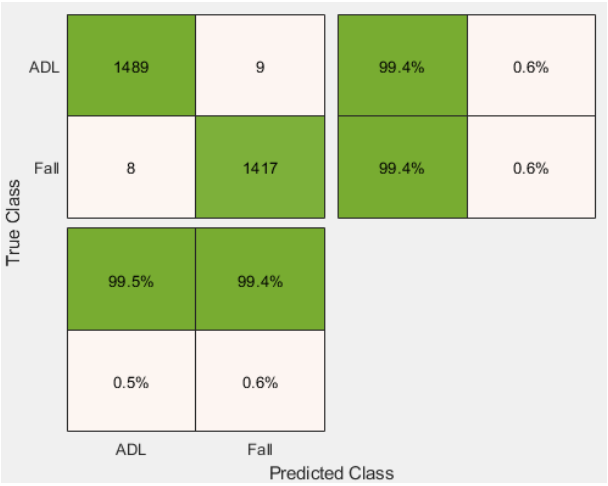


Figure 4.40: Confusion matrix trained stacked ensemble (No.4)

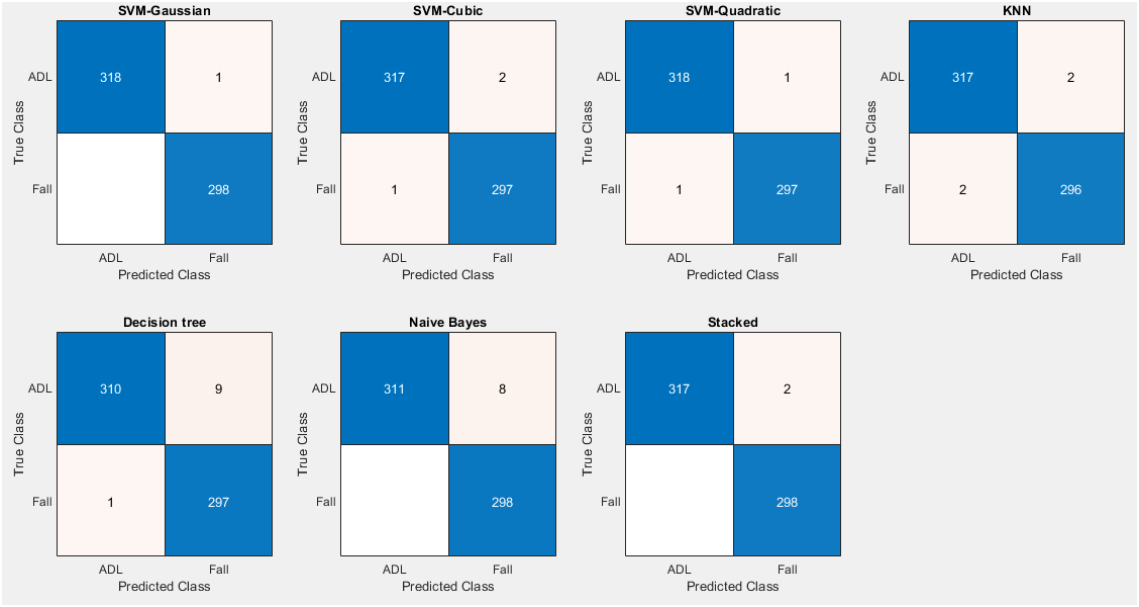


Figure 4.41: Confusion matrix of tested stacked ensemble (No.4)

Figure 4.42: Confusion matrix trained stacked ensemble (No.6)

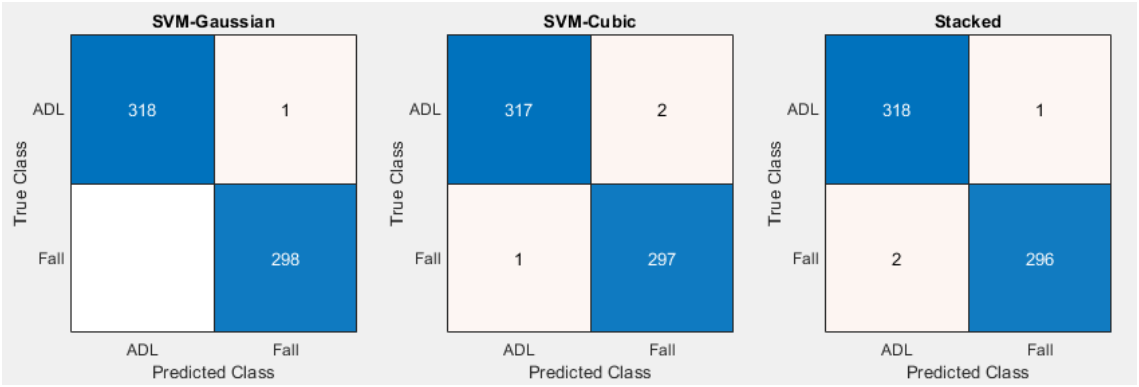


Figure 4.43: Confusion matrix of tested stacked ensemble (No.6)

CHAPTER 5

5 CONCLUDING DISCUSSION AND FUTURE SUGGESTIONS

Falling is a dangerous event faced mainly by elderly people. To take preventive measures from falls, tremendous work has been done on the development of a fall detection system using machine learning algorithms. This work also aimed to develop a fall detection system utilizing several algorithms that will be able to detect falls with the highest accuracy. In this instance, two public datasets such as SisFall and MobiFall were used to train up to 24 algorithms and test the algorithms with the highest recall, precision and accuracy. Seventeen experiments were designed to train the classifiers and test for the most accurate algorithms. Tables 5.1 and 5.2 show the overall best results for the classifiers trained and tested.

Table 5.1: Comparison of the best trained and tested classifiers of Experiment 1 and Experiment 2 for given simulations

No. of Experiments	Best Algorithm	Precision (%)	Accuracy (%)	Training Time (sec)
Experiment 1.1				
Training	Fine KNN	79.04±1.25	83.76±0.64	0.184±0.235
Testing	Cubic SVM	67.64±0	67.08±0	0.109±0.005
Experiment 1.2				
Training	Cubic SVM	90.21±0.88	90.61±0.56	6.216±0.035
Testing	Cubic SVM	77.85±0	76.49±0	0.109±0.003
Experiment 1.3				
Training	Cubic SVM	78.6±0.4	77.17±0.27	21.87±0.289
Testing	Quadratic SVM	64.22±0	62.72±0	0.106±0.001
Experiment 1.4				
Training	Fine KNN	99.73±0.06	99.72±0.06	0.165±0.114
Testing	Quadratic SVM	99.69±0	99.68±0	0.004±0.001
Experiment 2.1				
Training	Fine KNN	80.34±0.55	84.64±0.43	0.172±0.012
Testing	Cubic SVM	67.64±0	67.08±0	0.111±0.005
Experiment 2.2				
Training	Cubic SVM	90.89±0.69	91.58±0.45	13.1848±0.1660
Testing	Cubic SVM	77.85±0	76.49±0	0.108±0.002
Experiment 2.3				
Training	Cubic SVM	79.71±0.51	78.14±0.45	40.426±0.157
Testing	Quadratic SVM	64.22±0	62.72±0	0.382±0.004
Experiment 2.4				
Training	Fine KNN	99.73±0.03	99.72±0.03	0.201±0.050
Testing	Quadratic SVM	99.69±0	99.68±0	0.004±0

Table 5.2: Comparison of the best trained and tested classifiers of Experiment 3 and Experiment 4 for given simulations

No. of Experiments	Best Algorithm	Precision (%)	Accuracy (%)	Training Time (sec)
Experiment 3.1				
Training	Fine KNN	80.82±0.85	85.23±0.46	0.176±0.033
Testing	Cubic SVM	61.53±0	64.89±0	0.105±0.001
Experiment 3.2				
Training	Cubic SVM	91.77±0.56	92.08±0.38	13.059±0.364
Testing	Cubic SVM	74.65±0	74.61±0	0.108±0.003
Experiment 3.3				
Training	Cubic SVM	80.18±0.32	78.43±0.26	42.807±0.424
Testing	Quadratic SVM	63.97±0	61.75±0	0.39±0.015
Experiment 3.4				
Training	Medium Gaussian SVM	99.65±0.04	99.65±0.04	0.805±0.079
Testing	Medium Gaussian SVM	99.83±0	99.84±0	0.004±0
Experiment 4.1				
Training	Quadratic SVM	89.43±2.02	88.08±1.57	1.352±0.147
Testing	Medium Gaussian SVM	70.94±0	69.74±0	0.024±0.001
Experiment 4.2				
Training	Medium Gaussian SVM	91.92±1.03	89.59±0.79	1.284±0.024
Testing	Medium Gaussian SVM	75.87±0	72.37±0	0.025±0.001
Experiment 4.3				
Training	Quadratic SVM	88.18±0.94	85.89±0.48	2.767±0.034
Testing	Medium Gaussian SVM	79.05±0	77.94±0	0.058±0.011
Experiment 4.4				
Training	Medium Gaussian SVM	99.12±0.14	99.13±0.14	0.133±0.004
Testing	Medium Gaussian SVM	100±0	100±0	0.026±0.066

For SisFall dataset, it was observed that in terms of recall, precision and accuracy, the most suitable classifiers were Fine KNN, Cubic SVM and Quadratic SVM. Based on different experimental parameters, Fine KNN showed the highest accuracy of 88.37% in Experiment 1.1, 99.72% in Experiment 1.4, 84.64% in Experiment 2.1, 99.72% in Experiment 2.4 and 85.23% in Experiment 3.1, respectively. Cubic SVM also achieved the highest accuracy in a number of studied experiments as 90.61% in Experiment 1.2, 77.17% in Experiment 1.3, 91.58% in Experiment 2.2, 78.14% in Experiment 2.3, 92.08% in Experiment 3.2 and 78.43% in Experiment 3.3. Medium Gaussian SVM achieved the highest accuracy (99.65%) in Experiment 3.4.

In comparison, several authors used the SisFall dataset to develop the Fall detection system as [64] used the deep-learning-based accelerometer signal enhancement model and gave 97.34% accuracy from SVM. Cho and Yoon [86] used three public datasets, including SisFall on a fall detection system based on a one-dimensional convolutional neural network (CNN) and raw data of SisFall provided 61.37% accuracy. Cahoolessur and Rajkumarsingh [87] utilized SisFall datasets with an XGBoost algorithm and obtained an accuracy of 96%. A fall detection system developed by [88] used SisFall dataset based on non-linear classification feature with Kalman filter and achieved an accuracy of about 99.4%.

In this study, during testing the trained results for the SisFall dataset, it was observed that Fine KNN, Cubic SVM and Quadratic SVM exhibited the highest accuracy of over 99% in Experiments 1.4, 2.4 and 3.4, respectively. SVM performance varies depending on the kernel and parameters used for classification. As multiclass classifications are more complex in comparison to binary classifications, non-linear SVM has shown to have higher flexibility and performance as a large number of data points can processing be in higher dimensions [44]. However, Fine KNN had shown to have the least time taken for training in terms of training time. KNN has also shown to be robust at processing noisy data as well as having a general fast training time [89]. Thus, a combination of Fine KNN and Cubic SVM for classification is expected to give further accurate results.

For MobiFall dataset, various researchers had used the dataset with several methods to detect Fall Events and ADLs. KNN-based systems focused on binary classification achieved comparatively higher accuracy [90], [91]. Five classifiers were used by [90] for fall detection. Amongst the studied algorithms, KNN showed the highest accuracy at 87.5%. Similarly, a KNN-based system designed by [91] achieved accuracy up to 98% by utilizing the MobiFall dataset for training and testing with binary classification. In another study, [92] developed a threshold-based system that used the MobiFall dataset and reached up to 99.44% accuracy with binary classification (there is no further ADL and fall classification).

In comparison, trained results of Quadratic SVM gave the highest accuracy of 88.08% and 85.89% in Experiment 4.1 and 4.3, respectively, whereas Medium Gaussian SVM gave 89.59% accuracy in Experiment 4.2 and 99.13% in Experiment 4.4, respectively. In testing results with MobiFall dataset, Medium Gaussian SVM was the most accurate and

gave 100% accuracy in one instance. Fine KNN had shown to have comparatively less accuracy, with the highest being at 98.56% in Experiment 4.4. For Experiment 4.3, MobiFall Dataset results have been shown to give a higher recall, precision and accuracy on average, compared to results from the SisFall dataset in Experiment 4.4, due to a higher number of misclassifications between Fall Events.

In terms of cross-validation, Experiments 1 and 2 showed that higher k-fold cross-validation resulted in a slightly higher recall, precision and accuracy. Experiment 1 had used 5-fold cross-validation, and Experiment 2 had used 10-fold cross-validation. It was recorded in Experiment 1.1 that Fine KNN had a lower recall, precision and accuracy of 76.53%, 79.04% and 83.76% respectively, compared to Experiment 2, which had a comparatively higher recall, precision and accuracy of 77.85%, 80.34% and 84.64% respectively. However, it was observed that for testing, the results were very similar with both Experiments 1.1 and 2.1 having recall, precision and accuracy of 57.41% 58.77% and 61.76%, respectively.

Regarding feature selection for classification, Experiments 1.2, 2.2, 3.2 and 4.2 used 16 features and showed a higher recall, precision and accuracy compared to Experiments 1.1, 2.1, 3.1, and 4.1, which used 10 features. For example, in Experiment 1.1, Fine KNN had lower recall, precision and accuracy of 76.53%, 79.04% and 83.76% respectively, compared to Experiment 1.2, which had a comparatively higher recall, precision and accuracy of 85.62%, 88.95%, and 88.85% respectively. A similar trend has been observed in all the other experiments conducted with higher features.

Applying binary classification in a stacked ensemble classifier had shown varying results depending on the specific base classifier model being used for the system. While a trained stacked ensemble of Med. Gaussian. Cubic and Quadratic SVM, Fine KNN and Decision Tree was shown to have the highest accuracy for both ADXL345 and MMA8451Q models, lowering the number of base models to Medium Gaussian and Cubic SVM had shown to decrease the training time. Accuracy was observed to have decreased for ADXL345 results; however, MMA8451Q results have shown minimal effect in results. While accuracy between individual classifier models may vary, the reduction in result variance in using a stacked ensemble classifier system has the advantage of improving the overall quality of classification [93], [94]. It was noted that results during training had shown an overall higher recall, precision and accuracy compared to testing results. There

are multiple factors relating to this discrepancy. A factor that affects this variation of results can be postulated to be due to the differences in the datapoint distribution between the training and testing sets as there was a difference between the number of sampled datapoints in the testing and the training dataset. It seemed that conducting experiments with a larger training dataset would provide more accurate and precise results.

In conclusion, a potential combination of Machine Learning (ML) algorithms for an accelerometer-based system that will provide the best accuracy, sensitivity (recall), specificity and precision has been designed by simulating the training and testing phases of classifiers. It was observed that the majority of times, Fine KNN, Medium Gaussian, Quadratic and Cubic SVM gave the best results for most of the experiments. Additionally, testing the trained classifier with test data had shown variations in accuracy, recall and precision being lower, compared to the results given when the trained classifier was testing via cross-validation. This was partly due to a comparatively lower set of data for testing. Testing results in Experiments 1.4, 2.4, 3.4, 4.4 and using a stacked ensemble method had also confirmed that if the classifier system was also trained for differentiating only between Fall Events and ADLs, most of the algorithms were able to distinguish between the two classes and achieve a high recall, precision and accuracy while having a faster training time. In simulations, this faster training time for both training and testing phases of the experiments was assumed to be an indication of lower usage of memory and energy consumption.

Significant contributions at this current stage include a comprehensive simulation model, system design and data based on simulation available in MATLAB with the SisFall and MobiFall datasets. Due to the current COVID-19 pandemic, research was focused on the simulation aspect of the designed system. The current developed combined ensemble classifier is to be applicable to practical fall detection modules focused on portability (smartphones and accelerometers). Based on the current available portable devices, Android-based smartphones are generally cheaper compared to Apple iOS-based smartphones and such Android-based devices tend to be more accessible to the general public globally [95], [96]. Hence a developed system, if applied to such devices, results in a lower cost and will be more affordable to the public.

The classifiers can be further optimized for future use. However, future experiments are required to be done in form of testing other datasets for comparison as well as practical

testing. For the development and testing of the classifiers, other datasets are also viable for testing, including DLR [97] UMAFall [98] and TST [99] datasets. In addition, there is more work to be done in applying the combination of classifiers using the current results for the fall detection system in a Python or Android-based system. Such portable accelerometer-based systems need to be further optimised in terms of memory usage and overall energy consumption when actively monitoring and processing ADLs and Fall Events.

6 APPENDIX

6.1 The following shows an instance of the code for training Fine KNN for Experiment 2.4

```
fclose all;
clear

load('extracted_80_per_train_ADLFall_ADXL345__dataset.mat',
'ADLFall_80_per_train_ADXL345_table')
trainingData = ADLFall_80_per_train_ADXL345_table;

for i=1:1:10
tic



% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'Mean_x', 'Mean_y', 'Mean_z', 'Max_x', 'Max_y',
'Max_z', 'Min_x', 'Min_y', 'Min_z', 'std_x', 'std_y', 'std_z',
'AvgAcc', 'cos_AvgAcc_x', 'cos_AvgAcc_y', 'cos_AvgAcc_z'};
predictors = inputTable(:, predictorNames);
response = inputTable.Cat_B1;
isCategoricalPredictor = [false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false, false];

% Train a classifier
% This code specifies all the classifier options and trains the
classifier.
classificationKNN = fitcknn(...
    predictors, ...
    response, ...
    'Distance', 'Euclidean', ...
    'Exponent', [], ...
    'NumNeighbors', 1, ...
    'DistanceWeight', 'Equal', ...
    'Standardize', true, ...
    'ClassNames', categorical({'ADL'; 'Fall'}));

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
knnPredictFcn = @(x) predict(classificationKNN, x);
trainedClassifier.predictFcn = @(x)
knnPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.RequiredVariables = {'AvgAcc', 'Max_x', 'Max_y',
'Max_z', 'Mean_x', 'Mean_y', 'Mean_z', 'Min_x', 'Min_y', 'Min_z',
'cos_AvgAcc_x', 'cos_AvgAcc_y', 'cos_AvgAcc_z', 'std_x', 'std_y',
'std_z'};
trainedClassifier.ClassificationKNN = classificationKNN;
trainedClassifier.About = 'This struct is a trained model exported
from Classification Learner R2019b.';

% Extract predictors and response
```

```
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'Mean_x', 'Mean_y', 'Mean_z', 'Max_x', 'Max_y',
'Max_z', 'Min_x', 'Min_y', 'Min_z', 'std_x', 'std_y', 'std_z',
'AvgAcc', 'cos_AvgAcc_x', 'cos_AvgAcc_y', 'cos_AvgAcc_z'};
predictors = inputTable(:, predictorNames);
response = inputTable.Cat_B1;
isCategoricalPredictor = [false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false, false];

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationKNN,
'KFold', 10);

% Compute validation predictions
[validationPredictions, validationScores] =
kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun',
'ClassifError');

%Added

elapsedTime = toc;

Predict_Cat = validationPredictions;
Actual_Cat = categorical(ADLFall_80_per_train_ADXL345_table.Cat_B1);

newStr = join(["Experiment 5/Exp_5_Ex_14_",i,".mat"],'');
Saved_Mat = convertStringsToChars(newStr);

save(Saved_Mat,'Predict_Cat','Actual_Cat','trainedClassifier','validat
ionAccuracy','elapsedTime');
clearvars -except trainingData ADLFall_80_per_train_ADXL345_table
end
```

6.2 The following shows an instance of the code for training Cubic SVM for Experiment 4.1

```
fclose all;
clear

load('extracted_80_per_ADL_MobiFall_Acc_dataset.mat',
'MobiFall_ADL_80_per_acc_table')
trainingData = MobiFall_ADL_80_per_acc_table;

for i=1:1:10
tic
inputTable = trainingData;
predictorNames = {'Mean_x', 'Mean_y', 'Mean_z', 'Max_x', 'Max_y',
'Max_z', 'Min_x', 'Min_y', 'Min_z', 'AvgAcc'};
predictors = inputTable(:, predictorNames);
response = inputTable.acc_cat;
isCategoricalPredictor = [false, false, false, false, false, false,
false, false, false, false];

% Train a classifier
% This code specifies all the classifier options and trains the
classifier.
template = templateSVM(...
    'KernelFunction', 'polynomial', ...
    'PolynomialOrder', 3, ...
    'KernelScale', 'auto', ...
    'BoxConstraint', 1, ...
    'Standardize', true);
classificationSVM = fitcecoc(...
    predictors, ...
    response, ...
    'Learners', template, ...
    'Coding', 'onevsone', ...
    'ClassNames', categorical({'CSI'; 'CSO'; 'JOG'; 'JUM'; 'SCH';
'STD'; 'STN'; 'STU'; 'WAL'}));

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
svmPredictFcn = @(x) predict(classificationSVM, x);
trainedClassifier.predictFcn = @(x)
svmPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.RequiredVariables = {'AvgAcc', 'Max_x', 'Max_y',
'Max_z', 'Mean_x', 'Mean_y', 'Mean_z', 'Min_x', 'Min_y', 'Min_z'};
trainedClassifier.ClassificationSVM = classificationSVM;
trainedClassifier.About = 'This struct is a trained model exported
from Classification Learner R2019b.';
trainedClassifier.HowToPredict = sprintf

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'Mean_x', 'Mean_y', 'Mean_z', 'Max_x', 'Max_y',
'Max_z', 'Min_x', 'Min_y', 'Min_z', 'AvgAcc'};
predictors = inputTable(:, predictorNames);
```

```
response = inputTable.acc_cat;
isCategoricalPredictor = [false, false, false, false, false, false,
false, false, false, false];

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationSVM,
'KFold', 5);

% Compute validation predictions
[validationPredictions, validationScores] =
kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun',
'ClassifError');


%Added

elapsedTime = toc;

Predict_Cat = validationPredictions;
Actual_Cat = categorical(MobiFall_ADL_80_per_acc_table.acc_cat);

newStr = join(["Experiment 1/Exp_1_Ex_10_",i,".mat"],'');
Saved_Mat = convertStringsToChars(newStr);

save(Saved_Mat,'Predict_Cat','Actual_Cat','trainedClassifier','validat
ionAccuracy','elapsedTime');
clearvars -except trainingData MobiFall_ADL_80_per_acc_table
end
```

6.3 The following shows an instance of the code for saving the training results into readable form and extracting the results for external processing

```
%%
%Save stats from trained classifier

fclose all;
clear

for k=1:1:24
    for l=1:1:10
        newStr = join(["Experiment 5/Exp_5_Ex_",k,"_",l,".mat"],'');
        newStr1 = join(["Experiment
5/Exp_5_Ex_",k,"_",l,"_stats.mat"],'');

        Load_Mat = convertStringsToChars(newStr);
        Saved_Mat = convertStringsToChars(newStr1);

        load(Load_Mat);

        %confusion matrix struct
        [cm, order] = confusionmat(Actual_Cat,Predict_Cat);
        [m,n] = size(cm);

        macro_av_recall = 0;
        macro_av_precision = 0;
        macro_av_specificity = 0;
        macro_av_F1_score = 0;

        for no = 1:1:2

            %Initialize variables
            TN(no,:) = 0;
            TP(no,:) = cm(no,no);
            total_obs(no,:) = 0;
            FN(no,:) = 0;
            FP(no,:) = 0;

            recall(no,:) = 0;
            precision(no,:) = 0;
            specificity(no,:) = 0;
            F1_score(no,:) = 0;

            for i=1:1:m
                for j=1:1:n
                    total_obs(no,:) = total_obs(no,:) + cm(i,j);
                    if i == no
                        FN(no,:) = FN(no,:) + cm(i,j);
                    end
                    if j == no
                        FP(no,:) = FP(no,:) + cm(i,j);
                    end
                end
            end
```

```
            FP(no,:) = FP(no,:) - TP(no,:);
            FN(no,:) = FN(no,:) - TP(no,:);
            TN(no,:) = total_obs(no,:) - cm(no,no) - FP(no,:) -
FN(no,:);

            recall(no,:) = TP(no,:)/(TP(no,:) + FN(no,:));
            precision(no,:) = TP(no,:)/(TP(no,:) + FP(no,:));
            specificity(no,:) = TN(no,:)/(TN(no,:) + FP(no,:));

            recall(isnan(recall))=0;
            precision(isnan(precision))=0;
            specificity(isnan(specificity))=0;

            F1_score(no,:) = (2 * recall(no,:) * precision(no,:)) /
(recall(no,:) + precision(no,:));

            F1_score(isnan(F1_score))=0;

            recall(isnan(recall))=0;
            precision(isnan(precision))=0;

            macro_av_recall = macro_av_recall + recall(no,:);
            macro_av_precision = macro_av_precision + precision(no,:);
            macro_av_specificity = macro_av_specificity +
specificity(no,:);
            macro_av_F1_score = macro_av_F1_score + F1_score(no,:);
        end

        macro_av_recall = macro_av_recall/2;
        macro_av_precision = macro_av_precision/2;
        macro_av_specificity = macro_av_specificity/2;
        macro_av_F1_score = macro_av_F1_score/2;


        save (Saved_Mat, 'cm', 'order', 'm', 'n', 'TP', 'TN', 'FP',...
            'FN', 'recall', 'precision', 'specificity', 'total_obs',
'F1_score',...
            'macro_av_recall', 'macro_av_precision',
'macro_av_specificity', 'macro_av_F1_score');
    end
end

%%

%For Exporting

clear

meh = 24;

for no = 1:1:10

    newStr = join(["Experiment
5/Exp_5_Ex_",meh,"_",no,"_stats.mat"],'');
    Load_Mat = convertStringsToChars(newStr);
    load(Load_Mat);

    newStr1 = join(["Experiment 5/Exp_5_Ex_",meh,"_",no,".mat"],'');
```

```
    Load_Mat1 = convertStringsToChars(newStr1);
    load(Load_Mat1, 'validationAccuracy', 'elapsedTime');

    array_av(no,1) = macro_av_recall*100;
    array_av(no,2) = macro_av_precision*100;
    array_av(no,3) = macro_av_specificity*100;
    array_av(no,4) = macro_av_F1_score*100;
    array_av(no,5) = validationAccuracy*100;
    array_av(no,6) = elapsedTime;
end
```

6.4 The following shows an instance of the code for saving the testing results into readable form and extracting the results for external processing

```
%%
%Save stats from trained classifier

fclose all;
clear
no_of_class = 2;

load('extracted_20_per_test_ADLFall_ADXL345__dataset.mat',
'ADLFall_20_per_test_ADXL345_table')

for k=14:1:14
    for l=1:1:10
        newStr = join(["Experiment 5/Exp_5_Ex_",k,"_",l,".mat"],'');
        newStr1 = join(["Experiment
5/Exp_5_Ex_",k,"_",l,"_stats_20_per.mat"],'');

        Load_Mat = convertStringsToChars(newStr);
        Saved_Mat = convertStringsToChars(newStr1);

        if exist(Load_Mat, 'file')
            load(Load_Mat);
            tic
            Predict_Cat =
trainedClassifier.predictFcn(ADLFall_20_per_test_ADXL345_table);
            Actual_Cat =
categorical(ADLFall_20_per_test_ADXL345_table.Cat_B1);

            %confusion matrix struct
            [cm, order] = confusionmat(Actual_Cat,Predict_Cat)
            elapsedTime = toc;
            [m,n] = size(cm);

            accuracy = 0;
            macro_av_recall = 0;
            macro_av_precision = 0;
            macro_av_specificity = 0;
            macro_av_F1_score = 0;

            for no = 1:1:no_of_class

                %Initialize variables
                TN(no,:) = 0;
                TP(no,:) = cm(no,no);
                total_obs(no,:) = 0;
                FN(no,:) = 0;
                FP(no,:) = 0;

                recall(no,:) = 0;
                precision(no,:) = 0;
                specificity(no,:) = 0;
                F1_score(no,:) = 0;

                for i=1:1:m
                    for j=1:1:n
```

```
                    total_obs(no,:) = total_obs(no,:) + cm(i,j);
                    if i == no
                        FN(no,:) = FN(no,:) + cm(i,j);
                    end
                    if j == no
                        FP(no,:) = FP(no,:) + cm(i,j);
                    end
                end
            end

            FP(no,:) = FP(no,:) - TP(no,:);
            FN(no,:) = FN(no,:) - TP(no,:);
            TN(no,:) = total_obs(no,:) - cm(no,no) - FP(no,:) -
FN(no,:);


            recall(no,:) = TP(no,:)/(TP(no,:) + FN(no,:));
            precision(no,:) = TP(no,:)/(TP(no,:) + FP(no,:));
            specificity(no,:) = TN(no,:)/(TN(no,:) + FP(no,:));

            recall(isnan(recall))=0;
            precision(isnan(precision))=0;
            specificity(isnan(specificity))=0;

            F1_score(no,:) = (2 * recall(no,:) * precision(no,:))
/ (recall(no,:) + precision(no,:));

            F1_score(isnan(F1_score))=0;

            recall(isnan(recall))=0;
            precision(isnan(precision))=0;

            accuracy = accuracy + cm(no,no);
            macro_av_recall = macro_av_recall + recall(no,:);
            macro_av_precision = macro_av_precision +
precision(no,:);
            macro_av_specificity = macro_av_specificity +
specificity(no,:);
            macro_av_F1_score = macro_av_F1_score +
F1_score(no,:);
        end

        accuracy = accuracy/total_obs(1);
        macro_av_recall = macro_av_recall/no_of_class;
        macro_av_precision = macro_av_precision/no_of_class;
        macro_av_specificity = macro_av_specificity/no_of_class;
        macro_av_F1_score = macro_av_F1_score/no_of_class;


        save (Saved_Mat, 'accuracy',
'Predict_Cat','Actual_Cat','trainedClassifier','validationAccuracy','e
lapsedTime',...
            'cm', 'order', 'm', 'n', 'TP', 'TN', 'FP',...
            'FN', 'recall', 'precision', 'specificity',
'total_obs', 'F1_score',...
            'macro_av_recall', 'macro_av_precision',
'macro_av_specificity', 'macro_av_F1_score');
    end
end
```

```
end

%%

%For Exporting

clear

C_End(:,1) = {...
    "Fine Tree";
    "Medium Tree";
    "Coarse Tree";
    "Linear Discriminant";
    "Quadratic Discriminant";
    "Gaussian Naïve Bayes";
    "Kernel Naïve Bayes";
    "Linear SVM";
    "Quadratic SVM";
    "Cubic SVM";
    "Fine Gaussian SVM";
    "Medium Gaussian SVM";
    "Coarse Gaussian SVM";
    "Fine KNN";
    "Medium KNN";
    "Coarse KNN";
    "Cosine KNN";
    "Cubic KNN";
    "Weighted KNN";
    "Boosted Trees (AdaBoost)";
    "Bagged Trees";
    "Subspace Discriminant";
    "Subspace KNN";
    "RUSBoosted Trees"};


for meh = 1:1:24

    array_av = zeros(10,6);
    total_recall = 0;
    total_precision = 0;
    total_specificity = 0;
    total_F1_score = 0;
    total_accuracy = 0;
    total_elapsedTime = 0;

    for no = 1:1:10

        newStr = join(["Experiment
5/Exp_5_Ex_",meh,"_",no,".mat"],'');
        Load_Mat = convertStringsToChars(newStr);

        newStr1 = join(["Experiment
5/Exp_5_Ex_",meh,"_",no,"_stats_20_per.mat"],'');
        Load_Mat1 = convertStringsToChars(newStr1);

        if exist(Load_Mat1, 'file')

            load(Load_Mat1);
```

```
            load(Load_Mat, 'validationAccuracy');

            array_av(no,1) = macro_av_recall*100;
            array_av(no,2) = macro_av_precision*100;
            array_av(no,3) = macro_av_specificity*100;
            array_av(no,4) = macro_av_F1_score;
            array_av(no,5) = accuracy*100;
            array_av(no,6) = elapsedTime;

            total_recall = total_recall + array_av(no,1);
            total_precision = total_precision + array_av(no,2);
            total_specificity = total_specificity + array_av(no,3);
            total_F1_score = total_F1_score + array_av(no,4);
            total_accuracy = total_accuracy + array_av(no,5);
            total_elapsedTime = total_elapsedTime + array_av(no,6);
        end
    end

    std_array = std(array_av);

    total_recall = total_recall/10;
    total_precision = total_precision/10;
    total_specificity = total_specificity/10;
    total_F1_score = total_F1_score/10;
    total_accuracy = total_accuracy/10;
    total_elapsedTime = total_elapsedTime/10;

    C_End{meh,2} =
join([round(total_recall,2),"±",round(std_array(1),2)],'');
    C_End{meh,3} =
join([round(total_precision,2),"±",round(std_array(2),2)],'');
    C_End{meh,4} =
join([round(total_specificity,3),"±",round(std_array(3),3)],'');
    C_End{meh,5} =
join([round(total_F1_score,3),"±",round(std_array(4),3)],'');
    C_End{meh,6} =
join([round(total_accuracy,2),"±",round(std_array(5),2)],'');
    C_End{meh,7} =
join([round(total_elapsedTime,3),"±",round(std_array(6),3)],'');
end
```

6.5 The following shows an instance of the code for the stacked ensemble classifier system model

```
fclose all;
clear

load('extracted_80_per_train_ADLFall_ADXL345__dataset.mat',
'ADLFall_80_per_train_ADXL345_table')
trainingData = ADLFall_80_per_train_ADXL345_table;
load('extracted_20_per_test_ADLFall_ADXL345__dataset.mat',
'ADLFall_20_per_test_ADXL345_table')
testingData = ADLFall_20_per_test_ADXL345_table;


% load('extracted_80_per_ADLFall_MMA8451Q_dataset.mat',
'ADLFall_80_per_MMA8451Q_table')
% trainingData = ADLFall_80_per_MMA8451Q_table;
% load('extracted_20_per_ADLFall_MMA8451Q_dataset.mat',
'ADLFall_20_per_MMA8451Q_table')
% testingData = ADLFall_20_per_MMA8451Q_table;


tic
inputTable = trainingData;
predictorNames = {'Mean_x', 'Mean_y', 'Mean_z', 'Max_x', 'Max_y',
'Max_z', 'Min_x', 'Min_y', 'Min_z', 'std_x', 'std_y', 'std_z',
'AvgAcc', 'cos_AvgAcc_x', 'cos_AvgAcc_y', 'cos_AvgAcc_z'};
predictors = inputTable(:, predictorNames);
response = inputTable.Cat_B1;

% isCategoricalPredictor = [false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false, false];

% SVM with Gaussian kernel
rng('default') % For reproducibility
% mdls{1} =
fitcsvm(predictors,response,'KernelFunction','gaussian', ...
%     'Standardize',true,'KernelScale','auto');

mdls{1} = fitcsvm(...
    predictors, ...
    response, ...
    'KernelFunction', 'gaussian', ...
    'PolynomialOrder', [], ...
    'KernelScale', 4, ...
    'BoxConstraint', 1, ...
    'Standardize', true, ...
    'ClassNames', categorical({'ADL'; 'Fall'}));

% SVM with polynomial kernel
rng('default')
mdls{2} =
fitcsvm(predictors,response,'KernelFunction','polynomial','PolynomialO
rder', 3, ...
    'Standardize',true,'KernelScale','auto','BoxConstraint', 1);

rng('default')
```

```
mdls{3} =
fitcsvm(predictors,response,'KernelFunction','polynomial','PolynomialO
rder', 2, ...
    'Standardize',true,'KernelScale','auto','BoxConstraint', 1);

%KNN
rng('default')
mdls{4} = fitcknn(...
    predictors, ...
    response, ...
    'Distance', 'Euclidean', ...
    'Exponent', [], ...
    'NumNeighbors', 1, ...
    'DistanceWeight', 'Equal', ...
    'Standardize', true);


% % Decision tree
rng('default')

mdls{5} = fitctree(...
    predictors, ...
    response, ...
    'SplitCriterion', 'gdi', ...
    'MaxNumSplits', 100, ...
    'Surrogate', 'off', ...
    'ClassNames', categorical({'ADL'; 'Fall'}));




% mdls{5} = fitctree(trainingData,'Cat_B1');
%
% % Naive Bayes
rng('default')
mdls{6} = fitcnb(trainingData,'Cat_B1');
%
% % Ensemble of decision trees
% rng('default')
% mdls{5} = fitcensemble(trainingData,'Cat_B1');

%%


rng('default') % For reproducibility
N = numel(mdls);
Scores = zeros(size(trainingData,1),N);
cv = cvpartition(trainingData.Cat_B1,"KFold",5);
for ii = 1:N
    m = crossval(mdls{ii},'cvpartition',cv);
    [~,s] = kfoldPredict(m);
    Scores(:,ii) = s(:,m.ClassNames=='Fall');
end


%% Ensembled Version

rng('default') % For reproducibility
```

```
% t = templateKNN('NumNeighbors',5,'Standardize',1);
t = templateTree();
stckdMdl =
fitcensemble(Scores,trainingData.Cat_B1,'Method','Bag','Learners',t);

%%

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
svmPredictFcn = @(x) predict(stckdMdl, x);
trainedClassifier.predictFcn = @(x)
svmPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.RequiredVariables = {'AvgAcc', 'Max_x', 'Max_y',
'Max_z', 'Mean_x', 'Mean_y', 'Mean_z', 'Min_x', 'Min_y', 'Min_z',
'cos_AvgAcc_x', 'cos_AvgAcc_y', 'cos_AvgAcc_z', 'std_x', 'std_y',
'std_z'};
trainedClassifier.ClassificationEnsemble = stckdMdl;
trainedClassifier.About = 'This struct is a trained model exported
from Classification Learner R2019b.';
trainedClassifier.HowToPredict = sprintf('To make predictions on a new
table, T, use: \n  yfit = c.predictFcn(T) \nreplacing ''c'' with the
name of the variable that is this struct, e.g. ''trainedModel''. \n
\nThe table, T, must contain the variables returned by: \n
c.RequiredVariables \nVariable formats (e.g. matrix/vector, datatype)
must match the original training data. \nAdditional variables are
ignored. \n \nFor more information, see <a
href="matlab:helpview(fullfile(docroot, ''stats'', ''stats.map''),
''appclassification_exportmodeltoworkspace'')">How to predict using an
exported model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'Mean_x', 'Mean_y', 'Mean_z', 'Max_x', 'Max_y',
'Max_z', 'Min_x', 'Min_y', 'Min_z', 'std_x', 'std_y', 'std_z',
'AvgAcc', 'cos_AvgAcc_x', 'cos_AvgAcc_y', 'cos_AvgAcc_z'};
predictors = inputTable(:, predictorNames);
response = inputTable.Cat_B1;
isCategoricalPredictor = [false, false, false, false, false, false,
false, false, false, false, false, false, false, false, false, false];

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationEnsemble,
'KFold', 10);

% Compute validation predictions
[validationPredictions, validationScores] =
kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun',
'ClassifError');
```

```
Predict_Cat = validationPredictions;
Actual_Cat = categorical(ADLFall_80_per_train_ADXL345_table.Cat_B1);
%
% Actual_Cat = categorical(ADLFall_80_per_MMA8451Q_table.Cat_B1);


elapsedTime = toc;
%%

newStr = join(["ENSEM.mat"],'');
Saved_Mat = convertStringsToChars(newStr);

save(Saved_Mat,'Predict_Cat','Actual_Cat','trainedClassifier','validat
ionAccuracy','elapsedTime');
% clearvars -except trainingData ADLFall_80_per_train_ADXL345_table


%%
%First, iterate over the base models to the compute predicted labels,
scores, and loss values.
label = [];
score = zeros(size(testingData,1),N);
mdlLoss = zeros(1,numel(mdls));
for i = 1:N
    [lbl,s] = predict(mdls{i},testingData);
    label = [label,lbl];
    score(:,i) = s(:,m.ClassNames=='Fall');
     %mdlLoss(i) = mdls{i}.loss(testingData);
end

%Attach the predictions from the stacked ensemble to label and
mdlLoss.
[lbl,s] = predict(stckdMdl,score);
label = [label,lbl];
% mdlLoss(end+1) = stckdMdl.loss(score,testingData.Cat_B1);


%Concatenate the score of the stacked ensemble to the scores of the
base models.
score = [score,s(:,1)];



names = {'SVM-Gaussian','SVM-Cubic','SVM-Quadratic','KNN','Decision
tree','Naive Bayes','Stacked'};
% array2table(mdlLoss,'VariableNames',names)


%%


figure(1)
c = cell(N+1,1);
for i = 1:numel(c)
    subplot(2,4,i)
    c{i} = confusionchart(testingData.Cat_B1,label(:,i));
    title(names{i})
end
```

```
%%

% cc = confusionchart(testingData.Cat_B1,label(:,3));
```

7 REFERENCES

[1] United Nations, “World Population Ageing 2020 Highlights,” New York, 2020.

[2] World Health Organization (WHO), “Ageing and health,” *World Health Organization (WHO)*, 2021. https://www.who.int/news-room/fact-sheets/detail/ageing-and-health (accessed Jun. 14, 2022).

[3] World Health Organization, “World Report on Disability,” Geneva, Switzerland , 2011.

[4] World Health Organization (WHO), “WHO Global Report on Falls Prevention in Older Age,” Geneva, 2007.

[5] World Health Organization (WHO), “Falls,” *World Health Organization (WHO)*, 2021. https://www.who.int/news-room/fact-sheets/detail/falls (accessed May 14, 2022).

[6] M. Luštrek, H. Gjoreski, S. Kozina, B. Cvetkoviü, V. Mirchevska, and M. Gams, “Detecting Falls with Location Sensors and Accelerometers,” in *Twenty-Third Innovative Applications of Artificial Intelligence Conference*, 2011, pp. 1662–1667.

[7] N. Pannurat, S. Thiemjarus, and E. Nantajeewarawat, “Automatic Fall Monitoring: A Review,” *Sensors*, vol. 14, no. 7, pp. 12900–12936, Jul. 2014, doi: 10.3390/s140712900.

[8] National Institute on Aging, “Why Population Aging Matters: A Global Perspective,” USA, 2007.

[9] C. Tsirmpas, A. Rompas, O. Fokou, and D. Koutsouris, “An indoor navigation system for visually impaired and elderly people based on Radio Frequency Identification (RFID),” *Inf. Sci. (Ny).*, vol. 320, pp. 288–305, Nov. 2015, doi: 10.1016/J.INS.2014.08.011.

[10] E. Prassler, J. Scholz, and M. Strobel, “MAid: mobility assistance for elderly and disabled people,” in *IECON ’98. Proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society (Cat. No.98CH36200)*, 1998, vol. 4, pp. 2493–2498, doi: 10.1109/IECON.1998.724118.

[11] S. Rastogi and J. Singh, “A systematic review on machine learning for fall detection system,” *Comput. Intell.*, vol. 37, no. 2, pp. 951–974, May 2021, doi: 10.1111/COIN.12441.

[12] S. Tao, M. Kudo, and H. Nonaka, “Privacy-Preserved Behavior Analysis and Fall Detection by an Infrared Ceiling Sensor Network,” *Sensors*, vol. 12, no. 12, pp. 16920–16936, Dec. 2012, doi: 10.3390/S121216920.

[13] T. De Quadros, A. E. Lazzaretti, and F. K. Schneider, “A Movement Decomposition and Machine Learning-Based Fall Detection System Using Wrist Wearable Device,” *IEEE Sens. J.*, vol. 18, no. 12, pp. 5082–5089, Jun. 2018, doi: 10.1109/JSEN.2018.2829815.

[14] C.-Y. Hsieh, W.-T. Shi, H.-Y. Huang, K.-C. Liu, S. J. Hsu, and C.-T. Chan, “Machine learning-based fall characteristics monitoring system for strategic plan of falls prevention,” in *2018 IEEE International Conference on Applied System*

Invention (ICASI), Apr. 2018, pp. 818–821, doi: 10.1109/ICASI.2018.8394388.

[15] N. Zerrouki, F. Harrou, A. Houacine, and Y. Sun, "Fall detection using supervised machine learning algorithms: A comparative study," in *2016 8th International Conference on Modelling, Identification and Control (ICMIC)*, Nov. 2016, pp. 665–670, doi: 10.1109/ICMIC.2016.7804195.

[16] R. Igual, C. Medrano, and I. Plaza, "Challenges, issues and trends in fall detection systems," *Biomed. Eng. Online*, vol. 12, no. 1, p. 66, 2013, doi: 10.1186/1475-925X-12-66.

[17] S.-W. Yang and S.-K. Lin, "Fall detection for multiple pedestrians using depth image processing technique," *Comput. Methods Programs Biomed.*, vol. 114, no. 2, pp. 172–182, Apr. 2014, doi: 10.1016/J.CMPB.2014.02.001.

[18] R. Planinc and M. Kampel, "Introducing the use of depth data for fall detection," *Pers. Ubiquitous Comput.*, vol. 17, no. 6, pp. 1063–1072, Aug. 2013, doi: 10.1007/s00779-012-0552-z.

[19] S. Saurav, R. Saini, and S. Singh, "A dual-stream fused neural network for fall detection in multi-camera and 360^{o} videos," *Neural Comput. Appl.*, vol. 34, no. 2, pp. 1455–1482, Jan. 2022, doi: 10.1007/S00521-021-06495-5/TABLES/12.

[20] C. Khraief, F. Benzarti, and H. Amiri, "Elderly fall detection based on multi-stream deep convolutional networks," *Multimed. Tools Appl.*, vol. 79, no. 27–28, pp. 19537–19560, Jul. 2020, doi: 10.1007/S11042-020-08812-X/TABLES/7.

[21] P. Van Thanh *et al.*, "Development of a Real-Time, Simple and High-Accuracy Fall Detection System for Elderly Using 3-DOF Accelerometers," *Arab. J. Sci. Eng.*, vol. 44, no. 4, pp. 3329–3342, Apr. 2018, doi: 10.1007/s13369-018-3496-4.

[22] C. Zhang, Y. Tian, and E. Capezuti, "Privacy Preserving Automatic Fall Detection for Elderly Using RGBD Cameras," in *International Conference on Computers for Handicapped Persons*, 2012, pp. 625–633, doi: 10.1007/978-3-642-31522-0_95.

[23] Y. Lee, H. Yeh, K.-H. Kim, and O. Choi, "A real-time fall detection system based on the acceleration sensor of smartphone," *Int. J. Eng. Bus. Manag.*, vol. 10, pp. 1–8, Jan. 2018, doi: 10.1177/1847979017750669.

[24] Y. S. Su and S. H. Twu, "A Real Time Fall Detection System Using Tri-Axial Accelerometer and Clinometer Based on Smart Phones," *IFMBE Proc.*, vol. 74, pp. 129–137, 2020, doi: 10.1007/978-3-030-30636-6_19/FIGURES/7.

[25] S. Das, L. Green, B. Perez, and M. Murphy, "Detecting User Activities using the Accelerometer on Android Smartphones," 2010.

[26] P. Kostopoulos, A. I. Kyritsis, M. Deriaz, and D. Konstantas, "F2D: A Location Aware Fall Detection System Tested with Real Data from Daily Life of Elderly People," *Procedia Comput. Sci.*, vol. 98, pp. 212–219, Jan. 2016, doi: 10.1016/J.PROCS.2016.09.035.

[27] M. Gjoreski, H. Gjoreski, M. Luštrek, and M. Gams, "How Accurately Can Your Wrist Device Recognize Daily Activities and Detect Falls?," *Sensors*, vol. 16, no. 6, p. 800, Jun. 2016, doi: 10.3390/s16060800.

[28] G.-J. Horng and K.-H. Chen, "The Smart Fall Detection Mechanism for Healthcare

Under Free-Living Conditions," *Wirel. Pers. Commun.*, vol. 118, no. 1, pp. 715–753, May 2021, doi: 10.1007/S11277-020-08040-4/FIGURES/27.

[29] B. Kaudki and A. Surve, "Human Fall Detection Using RFID Technology," in *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Oct. 2018, doi: 10.1109/ICCCNT.2018.8494022.

[30] X. Tao, T. B. Shaik, N. Higgins, R. Gururajan, and X. Zhou, "Remote Patient Monitoring Using Radio Frequency Identification (RFID) Technology and Machine Learning for Early Detection of Suicidal Behaviour in Mental Health Facilities," *Sensors*, vol. 21, no. 3, p. 776, Jan. 2021, doi: 10.3390/S21030776.

[31] A. Leone, G. Rescio, and P. Siciliano, "Fall risk evaluation by surface electromyography technology," in *2017 International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, Feb. 2017, vol. 2018-January, pp. 1092–1095, doi: 10.1109/ICE.2017.8280003.

[32] G. Pérolle, P. Fraisse, M. Mavros, and I. Etxeberria, "Automatic Fall Detection and Activity Monitoring for Elderly," 2007.

[33] C. Y. Hsieh, C. N. Huang, K. C. Liu, W. C. Chu, and C. T. Chan, "A machine learning approach to fall detection algorithm using wearable sensor," in *Proceedings of the IEEE International Conference on Advanced Materials for Science and Engineering: Innovation, Science and Engineering, IEEE-ICAMSE 2016*, Feb. 2017, pp. 707–710, doi: 10.1109/ICAMSE.2016.7840209.

[34] C. Dinh and M. Struck, "A new real-time fall detection approach using fuzzy logic and a neural network," in *Proceedings of the 6th International Workshop on Wearable, Micro, and Nano Technologies for Personalized Health*, Jun. 2009, pp. 57–60, doi: 10.1109/PHEALTH.2009.5754822.

[35] G. Rescio, A. Leone, and P. Siciliano, "Support Vector Machine for tri-axial accelerometer-based fall detector," in *5th IEEE International Workshop on Advances in Sensors and Interfaces IWASI*, Jun. 2013, pp. 25–30, doi: 10.1109/IWASI.2013.6576096.

[36] J. He, C. Hu, and X. Wang, "A Smart Device Enabled System for Autonomous Fall Detection and Alert," *Int. J. Distrib. Sens. Networks*, vol. 12, no. 2, Feb. 2016, doi: 10.1155/2016/2308183.

[37] P. Mostarac, R. Malaric, M. Jurcevic, H. Hegedus, A. Lay-Ekuakille, and P. Vergallo, "System for monitoring and fall detection of patients using mobile 3-axis accelerometers sensors," in *2011 IEEE International Symposium on Medical Measurements and Applications*, May 2011, pp. 456–459, doi: 10.1109/MeMeA.2011.5966724.

[38] H. Gjoreski, M. Lustrek, and M. Gams, "Accelerometer Placement for Posture Recognition and Fall Detection," in *2011 Seventh International Conference on Intelligent Environments*, Jul. 2011, pp. 47–54, doi: 10.1109/IE.2011.11.

[39] T. Nguyen Gia *et al.*, "Energy efficient wearable sensor node for IoT-based fall detection systems," *Microprocess. Microsyst.*, vol. 56, pp. 34–46, Feb. 2018, doi: 10.1016/J.MICPRO.2017.10.014.

[40] F. A. S. Ferreira De Sousa, C. Escriba, E. G. Avina Bravo, V. Brossa, J. Y. Fourniols, and C. Rossi, “Wearable Pre-Impact Fall Detection System Based on 3D Accelerometer and Subject’s Height,” *IEEE Sens. J.*, vol. 22, no. 2, pp. 1738–1745, Jan. 2022, doi: 10.1109/JSEN.2021.3131037.

[41] M. Kangas, I. Vikman, J. Wiklander, P. Lindgren, L. Nyberg, and T. Jämsä, “Sensitivity and specificity of fall detection in people aged 40 years and over,” *Gait Posture*, vol. 29, no. 4, pp. 571–574, 2009, doi: https://doi.org/10.1016/j.gaitpost.2008.12.008.

[42] P. Pierleoni, A. Belli, L. Palma, M. Pellegrini, L. Pernini, and S. Valenti, “A High Reliability Wearable Device for Elderly Fall Detection,” *IEEE Sens. J.*, vol. 15, no. 8, pp. 4544–4553, Aug. 2015, doi: 10.1109/JSEN.2015.2423562.

[43] M. Saleh and R. Le Bouquin Jeannès, “An Efficient Machine Learning-Based Fall Detection Algorithm using Local Binary Features,” in *European Signal Processing Conference*, Nov. 2018, vol. 2018-Septe, pp. 667–671, doi: 10.23919/EUSIPCO.2018.8553340.

[44] A. R. Bagasta, Z. Rustam, J. Pandelaki, and W. A. Nugroho, “Comparison of Cubic SVM with Gaussian SVM: Classification of Infarction for detecting Ischemic Stroke,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 546, no. 5, p. 052016, Jun. 2019, doi: 10.1088/1757-899X/546/5/052016.

[45] T. Thielmann, “Navigation Becomes Travel Scouting: The Augmented Spaces of Car Navigation Systems,” in *Handbook of Research on Urban Informatics*, Siegen, Germany: IGI Global, 2009, pp. 230–243.

[46] R. Solea, A. Filipescu, A. Filipescu, E. Minca, and S. Filipescu, “Wheelchair control and navigation based on kinematic model and iris movement,” in *2015 IEEE 7th International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, Jul. 2015, pp. 78–83, doi: 10.1109/ICCIS.2015.7274600.

[47] A. Marco, R. Casas, J. Falco, H. Gracia, J. I. Artigas, and A. Roy, “Location-based services for elderly and disabled people,” *Comput. Commun.*, vol. 31, no. 6, pp. 1055–1066, Apr. 2008, doi: 10.1016/J.COMCOM.2007.12.031.

[48] J. C. Castillo, D. Carneiro, J. Serrano-Cuerda, P. Novais, A. Fernández-Caballero, and J. Neves, “A multi-modal approach for activity classification and fall detection,” *Int. J. Syst. Sci.*, vol. 45, no. 4, pp. 810–824, Apr. 2014, doi: 10.1080/00207721.2013.784372.

[49] Tunstall Healthcare, “Fall Detector Pendants and Fall Detection Devices,” *Tunstall Healthcare*. https://www.tunstallhealthcare.com.au/fall-detector-pendants (accessed May 02, 2022).

[50] Apple Inc., “Use fall detection with Apple Watch,” *Apple Inc.*, 2022. https://support.apple.com/en-au/HT208944 (accessed May 14, 2022).

[51] Apple Inc., “Apple Watch SE,” *Apple Inc.*, 2022. https://www.apple.com/au/apple-watch-se/ (accessed May 14, 2022).

[52] Live Life Alarms, “LiveLife Mobile Alarm with Fall Detect, Hands Free & GPS,” *Live Life Alarms*, 2022. https://livelifealarms.com.au/product/order-4GX-mobile-

alarm/ (accessed May 02, 2022).

[53] National Health Australia, "4G NHA Life Alarm - GPS & Fall Detector," *National Health Australia*, 2020. https://nationalhealth.com.au/product/medicalalarm4g/ (accessed May 02, 2022).

[54] National Health Australia, "Personal 3G Mobile Life Alarm with GPS and Fall Detector," *National Health Australia*, 2020. https://nationalhealth.com.au/product/personal-mobile-life-alarm-with-gps-and-fall-detector-black/ (accessed May 02, 2022).

[55] mCareWatch, "Introducing the mCareWatch," *mCareWatch*, 2022. https://mcarewatch.com.au/mcarewatch/ (accessed May 02, 2022).

[56] F. Liang, Z. Zhang, X. Li, and Z. Tong, "Lower Limb Action Recognition with Motion Data of a Human Joint," *Arab. J. Sci. Eng.*, vol. 41, no. 12, pp. 5111–5121, Dec. 2016, doi: 10.1007/s13369-016-2207-2.

[57] O. Ojetola, E. I. Gaura, and J. Brusey, "Fall Detection with Wearable Sensors--Safe (Smart Fall Detection)," in *2011 Seventh International Conference on Intelligent Environments*, Jul. 2011, pp. 318–321, doi: 10.1109/IE.2011.38.

[58] J. Wang, R. Chen, X. Sun, M. F. H. She, and Y. Wu, "Recognizing Human Daily Activities From Accelerometer Signal," *Procedia Eng.*, vol. 15, pp. 1780–1786, Jan. 2011, doi: 10.1016/J.PROENG.2011.08.331.

[59] T. Zhang, J. Wang, L. Xu, and P. Liu, "Using Wearable Sensor and NMF Algorithm to Realize Ambulatory Fall Detection," in *ICNC: International Conference on Natural Computation*, 2006, pp. 488–491, doi: 10.1007/11881223_60.

[60] Q. Li and J. A. Stankovic, "Grammar-Based, Posture- and Context-Cognitive Detection for Falls with Different Activity Levels," *Wirel. Heal. '11*, 2011.

[61] DATAFLAIR, "Kernel Functions-Introduction to SVM Kernel & Examples - DataFlair," *DATAFLAIR*, Nov. 16, 2018. https://data-flair.training/blogs/svm-kernel-functions/ (accessed Jun. 05, 2020).

[62] W. Koehrsen, "An Implementation and Explanation of the Random Forest in Python," *Towards Data Science*, Aug. 31, 2018. https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76 (accessed Jun. 05, 2020).

[63] L. Hardesty, "Explained: Neural networks | MIT News," *MIT News*, 2017. http://news.mit.edu/2017/explained-neural-networks-deep-learning-0414 (accessed Jun. 05, 2020).

[64] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, p. e00938, 2018, doi: https://doi.org/10.1016/j.heliyon.2018.e00938.

[65] A. Zheng and M. Tran, "8.2 Planes and Hyperplanes," New York, 2017.

[66] R. Nisbet, G. Miner, and K. Yale, "Chapter 8 - Advanced Algorithms for Data Mining," in *Handbook of Statistical Analysis and Data Mining Applications*

(Second Edition), 2nd ed., R. Nisbet, G. Miner, and K. B. T.-H. of S. A. and D. M. A. (Second E. Yale, Eds. Boston: Academic Press, 2018, pp. 149–167.

[67] M. K. Bhowmik, B. K. De, D. Bhattacharjee, D. K. Basu, and M. Nasipuri, "Multisensor fusion of visual and thermal images for human face identification using different SVM kernels," in *2012 IEEE Long Island Systems, Applications and Technology Conference, LISAT 2012*, 2012, doi: 10.1109/LISAT.2012.6223195.

[68] R. Berwick, "An Idiot's guide to Support vector machines (SVMs)," Boston, 2003.

[69] R. E. Neapolitan and X. Jiang, "Chapter 10 - Bankruptcy Prediction," in *Probabilistic Methods for Financial and Marketing Informatics*, R. E. Neapolitan and X. Jiang, Eds. Morgan Kaufmann, 2007, pp. 357–370.

[70] "1.9. Naive Bayes — scikit-learn 0.23.1 documentation," *scikit-learn: machine learning in Python*, 2019. https://scikit-learn.org/stable/modules/naive_bayes.html (accessed Jun. 04, 2020).

[71] R. Gandhi, "Naive Bayes Classifier," *Towards Data Science*, May 06, 2018. https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c (accessed Jun. 04, 2020).

[72] C. Reinders, H. Ackermann, M. Y. Yang, and B. Rosenhahn, "Chapter 4 - Learning Convolutional Neural Networks for Object Detection with Very Little Training Data," in *Multimodal Scene Understanding Algorithms, Applications and Deep Learning*, M. Y. Yang, B. Rosenhahn, and V. B. T.-M. S. U. Murino, Eds. Academic Press, 2019, pp. 65–100.

[73] L. Tan, "Chapter 17 - Code Comment Analysis for Improving Software Quality," in *The Art and Science of Analyzing Software Data*, C. Bird, T. Menzies, and T. B. T.-T. A. and S. of A. S. D. Zimmermann, Eds. Boston: Morgan Kaufmann, 2015, pp. 493–517.

[74] W. Mao and F.-Y. Wang, "Chapter 8 - Cultural Modeling for Behavior Analysis and Prediction," in *New Advances in Intelligence and Security Informatics*, W. Mao and F.-Y. B. T.-N. A. in I. and S. I. Wang, Eds. Boston: Academic Press, 2012, pp. 91–102.

[75] S. Ronaghan, "The Mathematics of Decision Trees, Random Forest and Feature Importance in Scikit-learn and Spark," *Towards Data Science*, May 12, 2018. https://towardsdatascience.com/the-mathematics-of-decision-trees-random-forest-and-feature-importance-in-scikit-learn-and-spark-f2861df67e3 (accessed Jun. 04, 2020).

[76] S. Panchal, "k Nearest Neighbor Classifier (kNN)-Machine Learning Algorithms," *Medium*, Mar. 09, 2018. https://medium.com/@equipintelligence/k-nearest-neighbor-classifier-knn-machine-learning-algorithms-ed62feb86582 (accessed Jun. 04, 2020).

[77] M. Schott, "K-Nearest Neighbors (KNN) Algorithm for Machine Learning," *Capital One Tech*, Apr. 22, 2019. https://medium.com/capital-one-tech/k-nearest-neighbors-knn-algorithm-for-machine-learning-e883219c8f26 (accessed Jun. 04, 2020).

[78] D. Jurafsky and J. H. Martin, “Hidden Markov Models,” in *Speech and Language Processing*, 3rd ed., Stanford: Stanford University Press, 2019.

[79] D. Lim, C. Park, N. H. Kim, S.-H. Kim, and Y. S. Yu, “Fall-Detection Algorithm Using 3-Axis Acceleration: Combination with Simple Threshold and Hidden Markov Model,” *J. Appl. Math.*, vol. 2014, 2014, doi: 10.1155/2014/896030.

[80] O. C. Ibe, “14 - Hidden Markov Models,” in *Markov Processes for Stochastic Modeling (Second Edition)*, 2nd ed., O. C. Ibe, Ed. Elsevier, 2013, pp. 417–451.

[81] Project Management Institute, “Methodology,” *Project Management Institute*, 2019. https://www.pmi.org/learning/featured-topics/methodology (accessed Apr. 03, 2019).

[82] D. Muslihat, “7 Popular Project Management Methodologies And What They’re Best Suited For,” *Zenkit*, 2018. https://zenkit.com/en/blog/7-popular-project-management-methodologies-and-what-theyre-best-suited-for/ (accessed Apr. 03, 2019).

[83] A. Sucerquia, J. D. López, and J. F. Vargas-Bonilla, “SISFALL DATASET,” *SISTEMIC*, 2016. http://sistemic.udea.edu.co/en/research/projects/english-falls/ (accessed Apr. 01, 2020).

[84] G. Vavoulas, M. Pediaditis, C. Chatzaki, E. G. Spanakis, and M. Tsiknakis, “The MobiFall Dataset: Fall Detection and Classification with a Smartphone,” *Int. J. Monit. Surveill. Technol. Res.*, vol. 2, no. 1, pp. 44–56, Oct. 2014, doi: 10.4018/ijmstr.2014010103.

[85] Amazon Web Services Inc., “Amazon Machine Learning Developer Guide,” 2020.

[86] H. Cho and S. M. Yoon, “Applying singular value decomposition on accelerometer data for 1D convolutional neural network based fall detection,” *Electron. Lett.*, vol. 55, no. 6, pp. 320–322, Mar. 2019, doi: 10.1049/EL.2018.6117.

[87] D. K. Cahoolessur and B. Rajkumarsingh, “Fall Detection System using XGBoost and IoT,” *R&D J. South African Inst. Mech. Eng.*, vol. 36, pp. 8–18, 2020, doi: 10.17159/2309-8988/2020/v36a2.

[88] A. Sucerquia, J. D. López, and J. F. Vargas-Bonilla, “Real-Life/Real-Time Elderly Fall Detection with a Triaxial Accelerometer,” *Sensors*, vol. 18, no. 4, p. 1101, Apr. 2018, doi: 10.3390/S18041101.

[89] S. D. Jadhav and H. P. Channe, “Comparative Study of K-NN, Naive Bayes and Decision Tree Classification Techniques,” *Int. J. Sci. Res.*, vol. 5, no. 1, pp. 1842–1845, Jan. 2016, doi: 10.21275/V5I1.NOV153131.

[90] P. Vallabh, R. Malekian, N. Ye, and D. C. Bogatinoska, “Fall detection using machine learning algorithms,” in *2016 24th International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2016*, Dec. 2016, pp. 1–9, doi: 10.1109/SOFTCOM.2016.7772142.

[91] X. Chen, H. Xue, M. Kim, C. Wang, and H. Y. Youn, “Detection of Falls with Smartphone Using Machine Learning Technique,” in *Proceedings - 2019 8th International Congress on Advanced Applied Informatics, IIAI-AAI 2019*, Jul. 2019, pp. 611–616, doi: 10.1109/IIAI-AAI.2019.00129.

[92] A. Sucerquia, J. D. Lopez, and F. Vargas, “Two-threshold energy based fall detection using a triaxial accelerometer,” in *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, Oct. 2016, vol. 2016-October, pp. 3101–3104, doi: 10.1109/EMBC.2016.7591385.

[93] M. P. Akhter, J. Zheng, F. Afzal, H. Lin, S. Riaz, and A. Mehmood, “Supervised ensemble learning methods towards automatically filtering Urdu fake news within social media,” *PeerJ Comput. Sci.*, vol. 7, pp. 1–24, Mar. 2021, doi: 10.7717/PEERJ-CS.425/SUPP-1.

[94] M. Villaverde, D. Aledo, D. Pérez, and F. Moreno, “On the adaptability of ensemble methods for distributed classification systems: A comparative analysis:,” *Int. J. Distrib. Sens. Networks*, vol. 15, no. 7, Jul. 2019, doi: 10.1177/1550147719865505.

[95] Nokia Corporation, “Nokia 1.4,” *Nokia Corporation*, 2022. https://www.nokia.com/phones/en_us/nokia-1-4?sku=F20BTX1362022 (accessed May 14, 2022).

[96] Apple Inc., “Buy iPhone 11,” *Apple Inc.*, 2022. https://www.apple.com/au/shop/buy-iphone/iphone-11 (accessed May 14, 2022).

[97] DLR - Institute of Communications and Navigation, “Human Activity Recognition with Inertial Sensors,” 2010. https://www.dlr.de/kn/en/desktopdefault.aspx/tabid-8500/14564_read-36508/ (accessed May 29, 2021).

[98] E. Casilari and J. A. Santoyo-Ramón, “UMAFall: Fall Detection Dataset (Universidad de Malaga),” 2018. https://doi.org/10.6084/m9.figshare.4214283.v7 (accessed May 29, 2021).

[99] S. Gasparrini *et al.*, “Proposal and Experimental Evaluation of Fall Detection Solution Based on Wearable and Depth Data Fusion,” *Adv. Intell. Syst. Comput.*, vol. 399, pp. 99–108, 2015, doi: 10.1007/978-3-319-25733-4_11.