FALL DETECTION AND CLASSIFICATION USING WEARABLE MOTION SENSORS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

By
Mustafa Şahin Turan
August 2017

FALL DETECTION AND CLASSIFICATION USING WEARABLE MOTION SENSORS
By Mustafa Şahin Turan
August 2017

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Billur Barshan (Advisor)

Sinan Gezici

Sinan Kalkan

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

FALL DETECTION AND CLASSIFICATION USING WEARABLE MOTION SENSORS

Mustafa Şahin Turan
M.S. in Electrical and Electronics Engineering
Advisor: Billur Barshan
August 2017

Effective fall-detection systems are vital in mitigating severe medical and economical consequences of falls to people in the fall risk groups. One class of such systems is wearable sensor based fall-detection systems. While there is a vast amount of academic work on this class of systems, the literature still lacks effective and robust algorithms and comparative evaluation of state-of-the-art algorithms on a common basis, using an extensive dataset. In this thesis, fall-detection and fall direction classification systems that use a motion sensor unit, worn at the waist of the subject, are presented. A comparison of a variety of fall-detection algorithms on an extensive dataset, comprising a total of 2880 trials, is undertaken. A novel heuristic fall-detection algorithm (**f**uzzy-**a**ugmented **do**uble **th**resholding: FADoTh) using two simple features is proposed and compared to 15 state-of-the-art heuristic fall-detection algorithms, among which it displays the highest average accuracy (98.45%), sensitivity, and F-measure values. A learner version of the same algorithm (***k***-NN classifier-**a**ugmented **t**ree: kAT) is developed and compared to eight machine learning (ML) classifiers based on the same dataset: Bayesian decision making (BDM), least squares method (LSM), k-nearest neighbor classifier (k-NN), artificial neural networks (ANN), support vector machines (SVM), decision tree classifier (DTC), random forest (RF), and adaptive boosting (AdaBoost). The kAT algorithm yields an average accuracy of 98.85% and performs on par with BDM, k-NN, ANN, SVM, DTC, RF, and AdaBoost, whereas LSM produces inferior results. Finally, the same eight ML classifiers are implemented for fall direction classification into four basic directions (forward, backward, right, and left) and evaluated on a reduced version of the same dataset consisting of only fall trials. BDM achieves perfect classification, followed by k-NN, SVM, and RF. BDM, LSM, k-NN, and ANN are modified to work in the presence of data from an unknown class and evaluated on the reduced dataset. In this robustness analysis, ANN and k-NN yield accuracies above

96.2%. The results obtained in this study are promising in developing real-world fall-detection systems.

Keywords: wearable sensors, motion sensors, fall detection, fall classification, fall-detection algorithms, heuristic (rule-based) methods, machine learning.

ÖZET

GİYİLEBİLİR HAREKET ALGILAYICILARIYLA DÜŞME SEZİMİ VE SINIFLANDIRMASI

Mustafa Şahin Turan
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Billur Barshan
Ağustos 2017

Etkin düşme algılama sistemleri, düşme risk gruplarındaki kişiler için düşmelerin ciddi tıbbi ve ekonomik sonuçlarının hafifletilmesinde yaşamsal öneme sahiptir. Bu tür sistemlerin bir çeşidi, giyilebilir algılayıcı kullanan düşme saptama sistemleridir. Konuyla ilgili çok sayıda akademik çalışma bulunmasına rağmen, literatür hala etkin ve gürbüz algoritmalara ve en güncel algoritmaların ortak ve kapsamlı bir veri kümesi üzerinde değerlendirilmesine ve karşılaştırılmasına gerek duymaktadır. Bu tez çalışmasında, kişinin belindeki bir hareket algılayıcısı ünitesini kullanan düşme saptama ve düşme yönü sınıflandırma algoritmaları sunulmuştur. Ayrıca, çeşitli düşme algılama algoritmalarının toplam 2880 veri örneğinden oluşan kapsamlı bir veri kümesi üzerinde karşılaştırması yapılmıştır. Sinyallerden elde edilen iki basit öznitelik kullanan yeni bir buluşsal düşme algılama algoritması (FADoTh) önerilmiş ve 15 güncel buluşsal düşme algılama algoritması ile karşılaştırılmıştır. FADoTh algoritması, en yüksek ortalama doğruluk (%98.45), duyarlılık ve F-ölçütü değerlerine sahiptir. Aynı algoritmanın öğrenici sürümü (kAT) geliştirilmiş ve sekiz makine öğrenme (ML) sınıflandırıcısı ile aynı veri kümesi üzerinde karşılaştırılmıştır: Bayesçi karar verme (BDM), en küçük kareler yöntemi (LSM), k-en yakın komşuluk sınıflandırıcısı (k-NN), yapay sinir ağları (ANN), destek vektör makineleri (SVM), karar ağacı sınıflandırıcısı (DTC), rastgele orman (RF) ve adaptif takviyeli sınıflandırma (AdaBoost). Bu karşılaştırmada kAT algoritması, %98.85 ortalama doğruluk ile BDM, k-NN, ANN, SVM, DTC, RF ve AdaBoost ile benzer başarım gösterirken, LSM'den daha yüksek sonuçlar üretmiştir. Son olarak, düşme eylemini dört temel yöne (öne, arkaya, sağa ve sola) sınıflandırmak için aynı sekiz ML sınıflandırıcısı uygulanmış ve veri kümesinin sadece düşme aktivitelerinden oluşan indirgenmiş bir sürümü üzerinde değerlendirilmiştir. BDM kusursuz sınıflandırma elde ederken k-NN, SVM ve RF onu izleyen başarımlar sergilemişlerdir. BDM, LSM, k-NN ve ANN, bilinmeyen bir sınıftan gelen verilerin varlığına karşın değiştirilmiş ve indirgenmiş veri seti üzerinde değerlendirilmiştir. Söz konusu gürbüzlük analizinde ANN ve k-NN, %96.2'nin üzerinde doğruluk sergilemiştir. Bu çalışmada elde edilen sonuçlar, gerçek dünyada kullanılabilecek düşme sezim sistemleri geliştirilmesinde yol gösterici olacaktır.

Anahtar sözcükler: giyilebilir algılayıcılar, hareket algılayıcıları, düşme sezimi, düşme sınıflandırması, düşme sezimi algoritmaları, buluşsal (kural tabanlı) yöntemler, makine öğrenmesi.

Acknowledgement

I would like to express my deepest gratitude to my supervisor Prof. Dr. Billur Barshan for her invaluable guidance, encouragement, and support throughout the duration of my M.S. study, especially in the preparation of this thesis.

Furthermore, my special thanks go to the members of my jury, Assoc. Prof. Dr. Sinan Gezici and Assoc. Prof. Dr. Sinan Kalkan, for accepting to read and review this thesis, and providing supportive comments, feedback, and criticism.

I owe particular thanks to Asst. Prof. Dr. Ahmet Turan Özdemir for sharing the dataset that he collected during his post-doctoral studies at Bilkent University that is used throughout this thesis and 16 anonymous subjects for having participated in the laboratory experiments in the collection of the dataset. My thanks are extended to Aras Yurtman for his precious help in the MATLAB implementation of the five of the machine learning classifiers.

I am grateful to my undergraduate professor Prof. Dr. Asif Šabanović for being more than a professor to me and inspiring me with his vision and experience.

My most heartfelt thanks are due to my wife, Yekta, for her unconditional support, love, and tolerance through the countless hardships that have befallen me in the preparation of this thesis.

My sincere appreciation is reserved for my mother Elif, my father Ümit, and my grandmother Hatice for their support and help throughout my whole life.

I am also grateful to my friends Mustafa Ütkür, Dr. Hakan Töreyin, M. Umar B. Niazi, Yalım İşleyici, and Musa Tunç Arslan for making my two years of M.S. study so enjoyable and sociable for me.

Last but not the least, I also thank TÜBİTAK for supporting me with a graduate scholarship (BİDEB-2210A) throughout my M.S. study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Definition

Today, a significant portion of elderly people live alone which is a major problem in cases of emergency. Falls are often dangerous for the elderly and might lead to serious injury or even death when medical attention is not provided rapidly. A fall is defined as an unstable event where a person unintentionally ends up on the ground or other lower level [1]. Statistically, falls constitute the most common cause of injury-related deaths for people over the age of 79 [2]. Serious medical conditions can arise due to either direct injury from the contact with the ground or the extended period of lying on the ground in cases where the elderly faller is unable to get up by herself/himself [3]. Aside from primary physical injuries that might stem from falls, secondary psychological damages that falls can cause are not to be overlooked. Fear of falling that is developed over periods of repetitive falling is shown to cause decrease in the mobility, independence, and social life among the elderly [4]. Considering that more than 66% of people who have fallen once have a tendency to fall again [5], elderly people experiencing falls are prone to suffer from its long term physical, psychological, and social consequences. This signifies the importance of attention on preventing or minimizing fall-related risks in the elderly healthcare domain. The elderly would not only benefit from efforts that improve their quality of life, such efforts would also help reduce the medical costs of fall-related injuries among the elderly, which was reported to be 30 billion USD in the U.S.A. in 2010 [6]. Although the elderly face the direst danger related to falls, disabled people, patients with visual, gait, balance, orthopedic, and neurological problems, workers, athletes, mountain climbers, and children are also in the fall risk group. Furthermore, falls may have different consequences for people in different age, gender, and profession groups: While they may cause trauma in children, athletes and workers may experience injury-related absence from work. Regardless of the nature of the faller, falling is a serious, costly, and life-threatening public health problem.

Developing reliable and effective fall-detection systems would help mitigate the consequences of falls for people, society, and even economies of countries. Therefore, it is not surprising that a considerable amount of academic and commercial work has focused on recognizing a fall event as accurately and quickly as possible. A fall-detection system should not fail to recognize fall events because missing a fall (missed detection) would mean that providing prompt medical attention will not be possible; nor should it label non-fall activities as falls (false alarm), because false alarms are disturbing and frustrating for the users. Besides, timely detection of falls is crucial since the severity of fall-related health risks generally increase with the initial response time. In addition to the primary properties above, a good fall-detection system should have other qualities for the ease of use on a daily basis.

It is important for a fall-detection system to be unobtrusive and ergonomic so that it is comfortable and non-invasive to the subject. Because these systems are supposed to monitor everyday life of the subjects for prompt detection of falls, they should not create too much inconvenience for the user. Fall-detection systems that require the subject to wear or carry around equipment suffer the disadvantage of obtrusiveness. As the number and amount of equipment to be carried on increases, the comfort of the user deteriorates.

Furthermore, a fall-detection system should also be low cost, while delivering high performance. Since the main purpose of these systems is to reduce the direct and indirect costs of falls to the patients and healthcare organizations, besides improving the quality of life in general, cost is a crucial criterion in designing a fall-detection system.

Considering that fall-detection systems are expected to monitor the user continuously, they also need to have low computational complexity and energy consumption. While computational complexity increases with the amount of data processing that has to be overtaken, energy consumption depends heavily on the sensor type as well as the computational complexity. Designing algorithms with lower computational complexity will yield faster response times, whereas low energy consumption will be favorable where power is expensive or limited.

Another concern for fall-detection systems is privacy. Patients may hesitate to use a system that intrudes into their daily lives. Privacy issues may be addressed by choosing the type of sensing accordingly or issuing protocols that prohibit monitoring of sensory data outside the scope of the fall-detection algorithm.

Finally, a fall-detection system should also be able to handle situations where there might be multiple persons, pets, or some obstacles in the environment to hinder proper data collection. Some type of sensors that can be used in fall-detection systems are inherently robust against these issues in that they record data only from the subject; however, others may need additional data processing to extract the information belonging to the subject of interest (SOI), or special configurations and constraints to ensure acquisition of useful data.

1.2 Related Work

Considering the design issues outlined above, numerous academic attempts have been made to tackle the problem of fall detection, each taking on some aspects of the extensive definition of the problem. Although the academic work on fall-detection systems is yet to mature, commercial systems already exist for public use. El-Bendary et al. [7] presented a comprehensive survey on fall-detection and prevention systems including commercial ones. After grouping commercial fall-detection and prevention systems into movement-sensing solutions and anti-wandering solutions, the authors go on to review some major products one at a time before concluding that each of them has its own advantages and disadvantages and that there is still room for more research in the area. Several commercial products are available for monitoring the elderly and automatic detection of falls [8, 9, 10] which help the elderly overcome the effects of falls to some extent. Nonetheless, when these systems were reviewed and tested extensively in [11], their real-world accuracies failed to exceed 70%. Users of commercial products also report that their high false alarm rates render them annoying, eventually leading to a reluctance to use them. Patents on fall-detection methods [12] and devices [13, 14, 15] provide a valuable insight into the academic work that has commercial value. Although patented systems are more likely to consider and tackle real-world problems with existing fall-detection systems and methods, shortcomings of existing fall-detection products indicate that there is still room for development in the area.

As the current progress in developing reliable fall-detection systems is still not adequate, it is beneficial to review the current articles in this area. Although different categorizations exist [16, 17], the literature on fall-detection systems can be separated into two categories: ambient (external) sensor based systems and wearable sensor based systems. This categorization is also employed in some reviews of existing fall-detection systems [18, 19]. Despite that hybrid systems that employ some combination of these two types of systems may exist, only the pure categories of systems as specified above will be examined in this thesis in order to highlight the general characteristics.

In the first approach, smart environments are developed by installing sensors (ambient sensors) in the environment. The most commonly used ambient sensors are cameras. Camera-based systems depend on fitting the environment that the user will live in with one or multiple cameras and processing the image data gathered by them. Commercially available standard RGB (red-green-blue) cameras can be used, though depth cameras such as Microsoft Kinect are also used in the literature [6, 20]. Upon specifying the visual sensor type, appropriate feature extraction is undertaken to obtain meaningful information on the posture of the SOI [20]. Once the necessary features are extracted, various classification algorithms can be implemented to recognize falls. Although this category of fall-detection systems are non-invasive in that they do not cause discomfort to the user, they require installation of relatively expensive equipment in the living environment of the SOI and limit the mobility of the user. While a possible solution to the limited range of this category of systems is equipping the whole living environment of the SOI with cameras, this solution brings about privacy concerns. Furthermore, the user is still not monitored when s/he is outdoors for shopping, commuting, sports, etc. Apart from the high installation cost and limited range, camera-based systems also suffer from high computational cost and energy consumption caused by image processing algorithms. Moreover, vision-based systems need a special arrangement of the living area and may have problems where the view is blocked by other people or objects, which is commonly referred to as the occlusion or shadowing problem. In cases where only conventional RGB cameras are used, some postures of the user cannot be captured because of the information loss in 2-D projection. A possible solution to this issue is to combine the 2-D projections of multiple cameras to reconstruct 3-D data about the user. This solution, however, brings about more computational complexity, more cost, and the problem of associating the data to the correct objects or users, which is called the correspondence problem. Another disadvantage of camera-based systems is the need for camera calibration. On the whole, while the capacity of camera-based applications is virtually limitless, their use in the realm of fall detection has major drawbacks.

Although other ambient sensor based systems are essentialy similar to camera-based ones in installing external sensors in the living environment of the subject, this class of systems can employ a variety of sensor types, not necessarily restricted to vision sensors. Living environment of the subject can be equipped with infrared proximity sensors [21], pressure or force sensing mats on the floor (smart floors), microphones, vibration sensors etc. to capture information-bearing signals about the state and activity of the subject. One or multiple of these sensors can be used in conjunction with vision-based sensors [22] to eliminate the drawbacks a single-sensor solution might have and improve the performance. Once the necessary features are extracted, suitable methods are used to distinguish falls from activities of daily living (ADLs). While ambient sensor based systems are unobtrusive as vision-based systems are, they are advantageous over the latter considering they are low cost [16], their processing requirements are simpler, and they do not pose as major privacy concerns as camera-based ones do. Once the necessary hardware is set up, multiple persons (e.g., an elderly couple) in need of such systems can benefit from the smart environment at the same time. In addition, problems related to forgetting to wear the sensors or wearing them improperly are eliminated in ambient sensor and camera-based systems. On the other hand, this class of systems is susceptible to other people or pets moving within the workspace; however, it is possible to overcome such limitations through implementing corresponding algorithms [21]. In spite of their advantages over camera-based systems, ambient sensor based fall-detection systems also suffer substantial difficulties.

Wearable sensor based systems consist of sensors of various types worn on the body of the user. With recent advances in the Micro Electro-Mechanical Systems (MEMS) technology, these devices have become smaller, lighter, compact, embeddable, inexpensive and wireless, while also consuming less and less power. Although most studies use tri-axial, bi-axial, or uni-axial accelerometers [23], a diversity of sensors including gyroscopes, magnetometers, pressure sensors, microphones, and even cameras can be worn on the body of the subject to capture the defining characteristics of the movements. After extracting features from the sensory data that have the adequate discriminative power, several classification tools are used. An illustration of the layout of wearable sensor based fall-detection systems is given in Figure 1.1.

Wearable sensors have virtually limitless range; the SOI can be monitored wherever s/he might go indoors and outdoors. This is due to the small, light, and compact nature of wearable sensors, which are therefore easily embeddable in portable devices, daily accessories, and clothing. In addition, the sensors directly acquire and record the data of only the SOI in 3-D, without any occlusion or

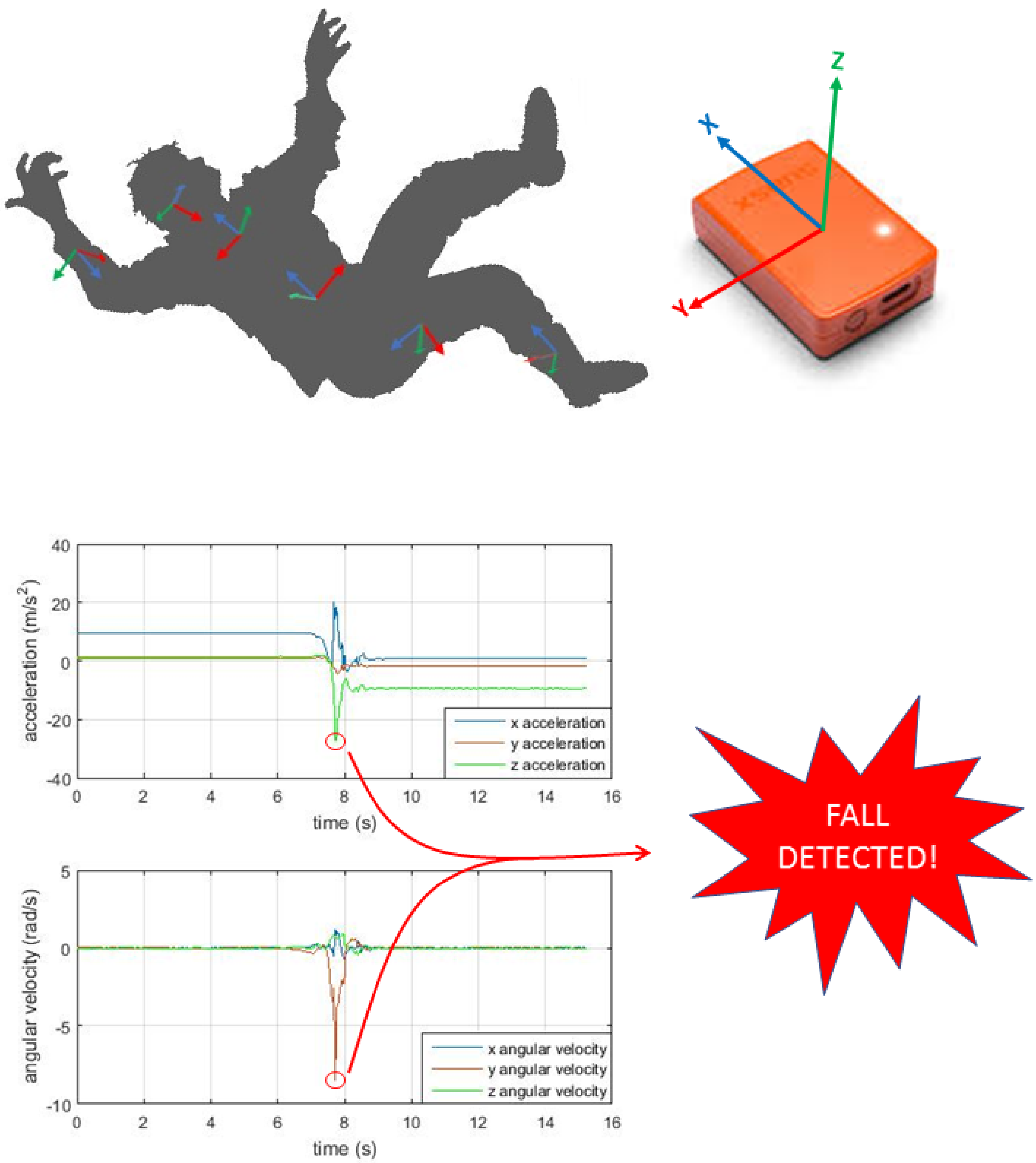

Figure 1.1: A Typical Wearable Fall-Detection System.

noise effects caused by other people, pets, or objects. Furthermore, this class of systems do not capture video or audio data of the subjects, therefore they do not pose any privacy concerns. Since the collected data are comprised of typically 1-D signals recorded from multiple axes, data processing is simpler and faster in wearable sensor based systems. However, wearable systems have been criticized because of a couple of disadvantages. First, the user may forget to wear them or may not wear them properly. In addition, since they are worn on the human body, they are battery-powered and their batteries need to be replaced or charged periodically. However, recent studies consider energy harvesting systems to overcome this problem [24, 25]. Besides, they are obtrusive to the users, that is, they can cause discomfort depending on the design. Although users might feel reluctant to carry around equipment, wearable sensor based fall-detection systems are highly preferred considering all their advantages. Since we believe that "activity can best be measured where it occurs [26]," we will focus on wearable sensor based fall-detection systems in this thesis.

Falls often occur unexpectedly while performing ADLs or while making transitions between two body postures (e.g., standing-to-sitting, lying down-to-standing). For this reason, falls and ADLs are usually considered together under the same umbrella. Activities that produce high acceleration (jumping, sitting down rapidly on a couch or bed, crouching, etc.) can be easily misclassified as falls. Therefore, fall-detection algorithms need to be sophisticated enough to correctly distinguish falls from ADLs, while at the same time be sufficiently simple to be fast and computationally efficient. Two types of approaches are used to distinguish falls from normal daily activities: one group of academic work uses heuristic (rule based) algorithms to make inferences about the captured motion, while the other group utilizes machine learning (ML) based classifiers.

Heuristic algorithms rely on defining simple rules and/or thresholds that eventually lead to recognizing fall events. Designed by hand, heuristic algorithms necessitate proper inspection of the collected sensory data from fall and non-fall scenarios. One then has to identify characteristics unique to all and only fall events to create a set of rules that lead to proper classification. Because these rules are usually simple in comparison and no training is required, heuristic algorithms are in general faster than ML classifiers. Not being restricted by a fixed set of ML tools, heuristics based fall detection is more flexible. Nonetheless, this class of algorithms may need special tuning of their parameters or thresholds according to a user or application.

Alternatively, a more recent approach is to utilize ML classifiers for fall detection. Supervised ML classifiers are mostly preferred over unsupervised ones in the fall-detection domain, since mere classification of activities is not sufficient and proper labeling is also needed. This kind of classifiers need to be trained with labeled training data beforehand. A variety of supervised ML classifiers is used in the literature, including, but not limited to, support vector machines (SVMs), artificial neural networks (ANNs), naïve Bayes (NB), Bayesian decision making (BDM), k-nearest neighbor classifier (k-NN), linear discriminant analysis (LDA), least squares method (LSM), and decision trees (DTs) [18, 20, 23, 27]. Özdemir and Barshan [28] compared different ML classifiers to show that, among k-NN, BDM, SVM, LSM, and ANN, k-NN yields the best accuracy. Although certain classifiers may give better results in specific studies, each classifier has its advantages and disadvantages, and should be selected accordingly. The selection of the parameters of the classifiers is another important issue in the implementation of ML classifiers. Because ML classifiers usually involve complex algorithms, they can be slow and computationally costly for some applications. Nevertheless, ML classifiers are shown to display better detection results in general [29].

The number of academic works on wearable sensor based fall-detection systems utilizing ML classifiers is considerably lower than those with heuristic algorithms. This is in large part due to the ease of developing novel heuristic algorithms that can resolve the binary fall detection problem, compared to developing or implementing ML classifiers. Some studies employ standard implementations of ML classifiers [30, 31, 32, 33], whereas others utilize hybrid algorithms that exploit one or multiple ML classifiers [34, 35, 36, 37].

Among the studies in the former group, Kansız et al. [30] compare the K-star, DT, and NB classifiers based on a dataset comprising 412 data instances (feature vectors) collected using smartphones in the pockets of eight subjects to conclude that the K-star classifier yields the best results with up to 88% sensitivity. Another comparison of ML classifiers using accelerometer data from the chest and the right thigh of the subjects is conducted by Yang et al. [33], where they compare the SVM, OneR, DT, and NB classifiers. They conclude that the NB classifier yields the best performance on a dataset comprising 288 data instances, with sensitivity values up to 87.5%. Similarly, Rescio et al. [38] compare four different kernel functions of the SVM classifier for detecting falls from tri-axial accelerometer data from 12 subjects. Polynomial and Gaussian radial basis function kernels yield the best performances as they display sensitivity and specificity values over 95% when using only 20% of the dataset comprising 700 data instances as the training set. Tong et al. [39] utilize a hidden Markov model-based fall prediction system for the prompt detection of falls. Their algorithm can predict falls with 100% accuracy with a minimum lead time of 200 ms, on a dataset comprising 280 data instances.

As an example of studies that offer hybrid algorithms, the article by Aguiar et al. [34] presents a fall-detection system employing a DT and a state machine. The state machine recognizes various stages of a fall using the thresholds that are determined by the DT, to eventually detect a fall or an ADL. They evaluate their algorithm on a large dataset comprising 3490 data instances that is acquired in agreement with the protocols proposed by Noury et al. [40]. Using 36 features in total, their algorithm achieves accuracy, sensitivity, and specificity values of 97.6%, 97.0%, and 98.4%, respectively. Zhang et al. [37] employ one-class SVM to extract doubtful data, before using kernel Fisher discriminant and k-NN classifiers for the detection of falls. Yuwono et al. [36] use a hybrid algorithm employing ANN and augmented radial basis function neural network classifiers together with regrouping particle swarm optimization and Gaussian distribution of clustered knowledge techniques. Their algorithm achieves sensitivity and specificity values above 97.6% and 96.8%, respectively, by combining the outputs of the two classifiers based on majority voting. Finally, Cheng and Jhan [35] combine the cascade-AdaBoost and SVM classifiers to detect falls using accelerometer data where each data sample in a sliding window is taken as a feature. The cascade-AdaBoost classifier is preferred to reduce false alarms and SVM detects falls based on the outputs of cascaded AdaBoost classifiers.

It is observed in the survey of fall-detection systems using ML classifiers that they usually involve complex data processing and ML algorithms; therefore, they are computationally more intensive than heuristic algorithms. Because of their advantages over ML classifiers, heuristic fall-detection algorithms are chosen for a more extensive review. Table 1.1 displays a summary of the literature on wearable sensors based fall detection that use heuristic algorithms.

It can be observed from the table that the most commonly used type of sensor in this approach is the accelerometer. This is reasonable because the event of fall is mostly characterized by an impact, which creates high peaks and a unique acceleration profile. Most fall types also come with a rapid change in body inclination as they result in lying down on the ground. The orientation of the body can be inferred by processing acceleration data or using gyroscopes, which measure the angular rate of the body of the subject. Moreover, magnetometers can be used to measure the orientation of the subject with respect to the Earth frame. Some studies use motion sensor units that consist of a tri-axial accelerometer, a tri-axial gyroscope, and a tri-axial magnetometer; however, acceleration data are considered to be adequate in most fall-detection studies.

One of the most suitable locations on the human body to place wearable sensors for fall detection is the waist. Most of the studies listed in Table 1.1 have selected the waist, since it is closer to the center of mass of the body of the subject. This, in turn, means that the acceleration and angular velocity readings reflect the motion of the subject better. Özdemir [41] compares different configurations of sensors, that is, their location on the body of the subject and concludes that the waist of the SOI is the best location to attach accelerometers for automatic fall detection. Results of the work by Ntanasis et al. [42], on the optimal place to fix wearable motion sensors on the human body for fall detection, also agrees with this, as they conclude that the waist and the thigh of the subject are the best places. However, some comparative studies in Table 1.1 display contrary results on the optimal place on the body to fix sensor units. Dai et al. [43] and Fang et al. [44] compare the chest, waist, and thigh of the subjects to attach their sensor units and show that their respective algorithms yield better results when the sensor units are worn on the chest, whereas Bourke et al. [45] conclude that the trunk is the best place to fix wearable sensors for fall detection. Some studies place a single sensor unit on the head and claim that rapid motions at the head are more likely to be caused by fall events [46]. Several studies consistently

Table 1.1: Wearable Sensor Based Heuristic Fall-Detection Systems.

Author	Year	No. of Subjects	No. of Sensors	Sensor Types	Sensor Placement	Features Used	No. of Activity Types	Performance (%)
Abbate et al. [48]	2011	4M	1	ACC	waist	SV, AAMV, FFI, FFAAM	17 (8 falls, 9 ADLs)	Se: 100, Sp: 100
Anania et al. [49]	2008	N/A	1	ACC	trunk	trunk inclination	9 (4 falls, 5 ADLs)	Se: 98, Sp: 100
Baek et al. [50]	2013	5	1	ACC, GYR	neck	trunk angles, SV, G_{SVM}	14 (5 falls, 9 ADLs)	Se: 81.6, Sp: 100
Bianchi et al. [47]	2010	20 (8F, 12M)	2	ACC, BAR	waist	SV, SMA, tilt angle, ΔP	16 (8 falls, 8 ADLs)	Acc: 96.9, Se: 97.5, Sp: 96.5
Boissy et al. [51]	2007	10	2	ACC	trunk	accelerations, trunk angles	15 (9 falls, 6 ADLs)	Acc: 93, FPR: 29
Bourke et al. [45]	2007	20	2	ACC	trunk, thigh	SV	16 (8 falls, 8 ADLs)	Sp: 100
Bourke et al. [52]	2008	5	1	ACC	chest	V_{vert}	10 (4 falls, 6 ADLs)	Se: 100, Sp: 100
Chao et al. [53]	2009	7M	1	ACC	chest	SV, ACP	21 (8 falls, 13 ADLs)	Se and Sp $>$ 85
Chen et al. [54]	2006	2	1	ACC	waist	SV, body inclination	N/A	N/A
Chen et al. [55]	2011	5 (1F, 4M)	1	ACC	waist	SV, body inclination	19 (13 falls, 6 ADLs)	Se$>$ 97, Sp: 100
Dai et al. [43]	2010	15 (2F, 13M)	1	ACC, GYR, MAG	chest/waist/thigh	SV, Z	N/A	FPR: 8.7–11.2, FNR: 1.0–10.0
Fang et al. [44]	2012	4	1	ACC, GYR, MAG	chest/waist/thigh	SV, Z	N/A	Se: 72.22, Sp: 73.78

Jacob et al. [56]	2011	3	3	1 ACC, 2 GYR	back	SV, G_{SVM}, A_{SVM}, $\Delta\vartheta$	N/A	Se: 100, Sp: 100
Jantaraprim et al. [57]	2010	20 (6F, 14M)	1	ACC	trunk	SV	10 (4 falls, 6 ADLs)	Se: 100, Sp: 100
Kangas et al. [58]	2007	2	1	ACC	waist	SV, Z	45 (14 falls, 31 ADLs)	Se: 100, Sp: 100
Kangas et al. [59]	2008	3 (1F, 2M)	3	ACC	wrist/waist/head	SV, Z, V	9 falls	Se: 97, Sp: 100
Kangas et al. [60]	2009	41 (24F, 17M)	1	ACC	waist	SV, Z, V	10 (6 falls, 4 ADLs)	Se: 97.5, Sp: 100
Koshmak et al. [61]	2013	7	1	ACC	waist	AM, roll, pitch	N/A	Acc: 94, Se: 90, Sp: 100
Lee and Carlisle [62]	2011	18 (6F, 12M)	2	ACC	waist	SV	12 (4 falls, 8 ADLs)	Se: 96, Sp: 82
Li et al. [63]	2009	3M	4	2 ACC, 2 GYR	chest, thigh	SV, G_{SVM}, trunk and thigh angles	N/A	Se: 91, Sp: 92
Lindemann et al. [46]	2005	1	1	ACC	head	SV, V, $SV_{x\text{-}y}$	12 (7 falls, 5 ADLs)	Se: 100
Pierleoni et al. [64]	2015	10 (2F, 8M)	3	ACC, GYR, MAG	waist	SV, body angles	18 (5 falls, 13 ADLs)	Acc: 90.37, Se: 80.74
Sorvala et al. [65]	2012	2	2	IMU, ACC	waist, ankle	SV, G_{SVM}, body inclination	10 (3 falls, 7 ADLs)	Se: 95.6, Sp: 99.6
Tolkiehn et al. [66]	2011	12 (4F, 8M)	2	ACC, BAR	waist	σ_{xyz}, σ_a, body tilt, body tilt change, pressure	25 (13 falls, 12 ADLs)	Acc: 86.97, Se: 85.24, Sp: 87.77
Wang et al. [67]	2008	5	1	ACC	head	SV, $SV_{x\text{-}y}$, V	15 (8 falls, 7 ADLs)	N/A

Wu and Xue [68]	2008	24	2	ACC, GYR	waist	V^{I}	N/A	N/A
Wu et al. [69]	2015	3	1	ACC	waist	SV, body inclination	9 (4 falls, 5 ADLs)	Se: 97.1, Sp: 98.3

Abbreviations used in Table 1.1:

SV: acceleration sum vector magnitude, Z: vertical acceleration, V: velocity, $SV_{x\text{-}y}$: SV in horizontal (x-y) plane, V_{vert}: vertical velocity, AAMV: average acceleration magnitude variation, FFI: free fall interval, FFAAM: free fall average acceleration magnitude, G_{SVM}: angular velocity sum vector magnitude, SMA: signal magnitude area, ΔP: differential pressure, $\Delta\vartheta$: total angular change, ACP: acceleration cross product, A_{SVM}: angular acceleration sum vector magnitude, AM: activity measure, V^{I}: velocity in inertial frame, σ_{xyz}: sum vector magnitude of standard deviations, σ_a: standard deviation of sum vector magnitude, Acc: accuracy, Se: sensitivity, Sp: specificity, FPR: false positive rate, FNR: false negative rate, ACC: accelerometer, GYR: gyroscope, MAG: magnetometer, BAR: barometric pressure sensor, IMU: inertial measurement unit.

agree that the arms and the legs are not suitable parts of the body to carry a fall-detection device since they are associated with higher accelerations and relatively more chaotic acceleration patterns [47].

Because the waist is considered to be one of the most suitable places to position sensors for fall detection, mobile phones are often utilized for this purpose [27, 70, 71]. Because they are very common and are already carried by most people today, a number of academic works employ them as a platform to run their algorithm on [37, 43, 72]. Besides, mobile phone-based fall-detection systems can also locate the subject when a fall is detected for a more rapid emergency response, because all recent smartphones employ GPS technology for localization [61, 62]. A small network for the user can be pre-defined on the telephone network. This network may consist of a hospital, a call center, healthcare professionals, the user's relatives and friends, etc. If the algorithm detects a fall, a warning or an alarm signal can be automatically sent to this small network to alert them. If at least one member of the network acknowledges the receipt of the alert signal, prompt medical attention can be provided. While these systems require no additional hardware to be worn on the user's body, they either require fixing the mobile phone tightly on the user's body or need to make use of special algorithms that consider small movements that the mobile phone may undergo. The work in [73] proposes a rotation-invariant activity recognition algorithm that can be exploited in mobile phone based fall-detection domain to address the mentioned problem. Similarly, mobile phones can easily be misplaced to other pockets, bags, or other locations on the user or elsewhere; therefore, solutions need to be developed to eliminate such a potential problem [74]. With the rapid advances in technology, mobile phones are getting smaller and more capable; therefore, developing comprehensive healthcare monitoring applications for them can be considered as a potent research direction.

While the most commonly used feature in the studies that are listed in Table 1.1 is total acceleration (SV), vertical acceleration (Z), body angles, total angular velocity (G_{SVM}), and speed of impact (V) are highly preferred features as well. This is only natural, since falls are mainly characterized by high impacts, sudden body posture changes, and free fall before the impact. Furthermore, several studies exploit the distinctive profile of the acceleration data of falls [48].

1.3 Current Issues and Challenges in the Area

Regarding the evaluation of the proposed systems, there is no common basis for comparison, that is, the literature lacks standards of subjects, activities, and collection procedures for creating datasets and evaluation methods [75]. Most of the existing work use their own datasets to evaluate their systems, which might have an inadequate number of subjects, fail to capture some types of falls or ADLs, or have an insufficiently small size. As is evident from Table 1.1, the number of subjects and activities vary, which signifies the need for standardized protocols in evaluating fall-detection systems. Noury et al. [40] proposed protocols for designing and conducting fall-detection experiments, while Abbate et al. [76] set out the most important criteria to consider while designing fall-detection systems. Although there is a limited number of studies that follow the proposed experimental protocols [34, 38, 55, 64, 77], the majority of the existing literature does not present comparable evaluation methods and results. This, then, brings about the need to compare the existing fall-detection methods on a common basis, using an extensive dataset. This task is tiresome since few publicly available datasets exist. For example, Ojetola et al. [78] acquired a publicly available dataset consisting of 42 subjects and four fall activities with numerous ADLs. Their dataset includes accelerometer and gyroscope readings from the chest and the thigh of the subjects.

Even when datasets are available for comparative evaluation of fall-detection systems, the performance of the algorithms on the datasets might fail to capture the real-life situations as most of the datasets used in the literature are collected from young and healthy subjects performing intentional (simulated) and scripted fall scenarios in a laboratory-like environment with protective equipment such as helmets, knee supports, and mats. Fall risk groups comprise elderly people or patients who might have impairments and abnormalities in their daily gaits, which might not necessarily match the subjects who participated in the experiments. Moreover, planned falls with protective equipment might differ from the sudden and chaotic nature of real-life falls. Using laboratory experiments in evaluating fall-detection systems thus hinders their real-world performances on fall risk groups. Although there is a limited number of studies that consider real-world falls [79] and use elderly subjects [45, 46, 57, 60, 68], these data are very scarce and are usually confined to special disease groups. Kangas et al. [80] compare real fall data collected from elderly people with those of intentional falls and conclude that the intentional fall data are similar to the real ones in their characteristics, despite that some parameters are different between experimental and real-life falls. Similarly, Bourke et al. [81] include unscripted ADL data from elderly subjects to evaluate the false alarm rate of their algorithm in a real-world setting. Their algorithm produces less than one false alarm per day.

Other issues about fall-detection systems include segmentation of the sensor signals. Some algorithms may require a time interval of sensor readings to identify some characteristics of falls such as impact after free fall and lying down after impact. In cases of real-time operation, such algorithms require proper segmentation of the sensory data. Commonly, static sliding window segmentation is employed by most of the academic works. This technique involves taking signals within a time window of pre-determined size and processing the windows by the algorithm one at a time. The consecutive windows might either overlap with each other or not. Whilst the latter option might possibly cause some falls to be missed, the former has the disadvantage of higher computational complexity. For robust detection of falls, overlapping windows are mostly employed in the literature. Automatic segmentation of the data to a variable/fixed size window around a possible fall event would prevent the detection algorithm from running on unlikely windows and thus would reduce the computational complexity considerably [28, 41].

An additional issue about the real-time operation of fall-detection systems is their run time. Recognizing fall events as quickly as possible or even predicting them before they occur is crucial in the fall prevention realm. Wu and Xue [68] developed a heuristic algorithm for a wearable sensor based fall detector that recognizes falls at least 70 ms before the impact. Their algorithm successfully detects all falls before they occur in the experiments and the authors claim that their system would facilitate development of a hip airbag to help mitigate fall-related injuries. Despite this study, most articles in the literature do not mention the detection time or discuss the real-time implementation of their systems.

1.4 Contributions and the Outline of the Thesis

Although the area of wearable fall detection has been active for a few decades, there is still a substantial need for original research to address the existing issues and challenges. This thesis tackles the lack of standardization in evaluating wearable fall-detection algorithms over an extensive dataset. Previously, Aziz et al. [29] compared five heuristic and five ML-based fall-detection algorithms using waist-mounted tri-axial accelerometers on their own dataset, which comprised a total of 210 fall and 390 non-fall activities performed by 10 participants. This thesis adds on the article by Aziz et al. [29] by proposing one heuristic and one ML-based fall-detection algorithm besides comparing them to 15 state-of-the-art heuristic algorithms and eight ML-based algorithms on a large dataset that consists of 1600 falls and 1280 non-falls performed by 16 subjects. Further, a classification of falls into four basic directions is also undertaken.

In this thesis, a number of existing fall-detection algorithms are implemented and evaluated based on the same dataset to compare their performances. A novel heuristic algorithm based on a waist-worn motion sensor unit is also presented and compared with the existing algorithms. The proposed algorithm performs better than 15 state-of-the-art heuristic fall-detection algorithms in terms of accuracy and sensitivity, where only one of the algorithms in the literature yields a higher specificity than the proposed one. In addition, a learner version of the proposed algorithm is formulated and compared to eight standard ML classifiers on the same dataset, which then performs the best. Having successfully detected falls with over 98.5% accuracy, falls are then classified with respect to their directions for more accurate emergency first response. The main contributions of this thesis are, thus, providing valuable insight to the relative performances of the state-of-the-art fall-detection algorithms over a common dataset and developing two effective fall-detection algorithms with a fall-classification module, which is sufficiently robust to recognize data from an unknown class.

The organization of this thesis is as follows: The next chapter introduces the proposed heuristic algorithm and describes the dataset used in conducting the comparison between the state-of-the-art algorithms and the proposed algorithm, along with brief descriptions of the selected algorithms and the details of the implementation. The results of the comparative evaluation is provided in the same chapter. Chapter 3 describes the learner version of the proposed algorithm in detail before presenting the comparison procedures and the results of the eight existing ML classifiers and the proposed algorithm. Following a survey of the literature on fall classification, fall direction classification is presented in Chapter 4. Finally, Chapter 5 provides a summary and concluding remarks with possible future research directions.

Chapter 2

A Novel Heuristic Fall-Detection Algorithm: FADoTh

As mentioned in the previous chapter, an effective fall-detection system should be both accurate and fast to be used for real-time detection of falls. To develop such a system, it is of utmost importance to examine the signals generated by fall and non-fall activities closely. Therefore, as a first exercise, the acceleration and angular velocity signals resulting from fall and non-fall activities over the extensive dataset that is used in this thesis are inspected. As an illustrative example, representative acceleration and angular velocity signals of walking and forward fall activities from the waist sensor are shown in parts a) and d) of Figure 2.1, respectively. In part d) of the figure, we can see a distinctive profile that most fall signals follow, characterized by an acceleration drop caused by the free-fall phase and an acceleration peak followed by the impact with the floor. It seems from these two activities that non-fall activities also have their distinct acceleration profiles: repetitive acceleration peaks and valleys in the case of walking. Although these representative signals of fall and non-fall activities are significantly distinct from one another, distinguishing falls from non-falls is not always straightforward, considering fall-like non-fall activities such as stumbling, limping, and jumping with high accelerations, and fall activities such as syncope

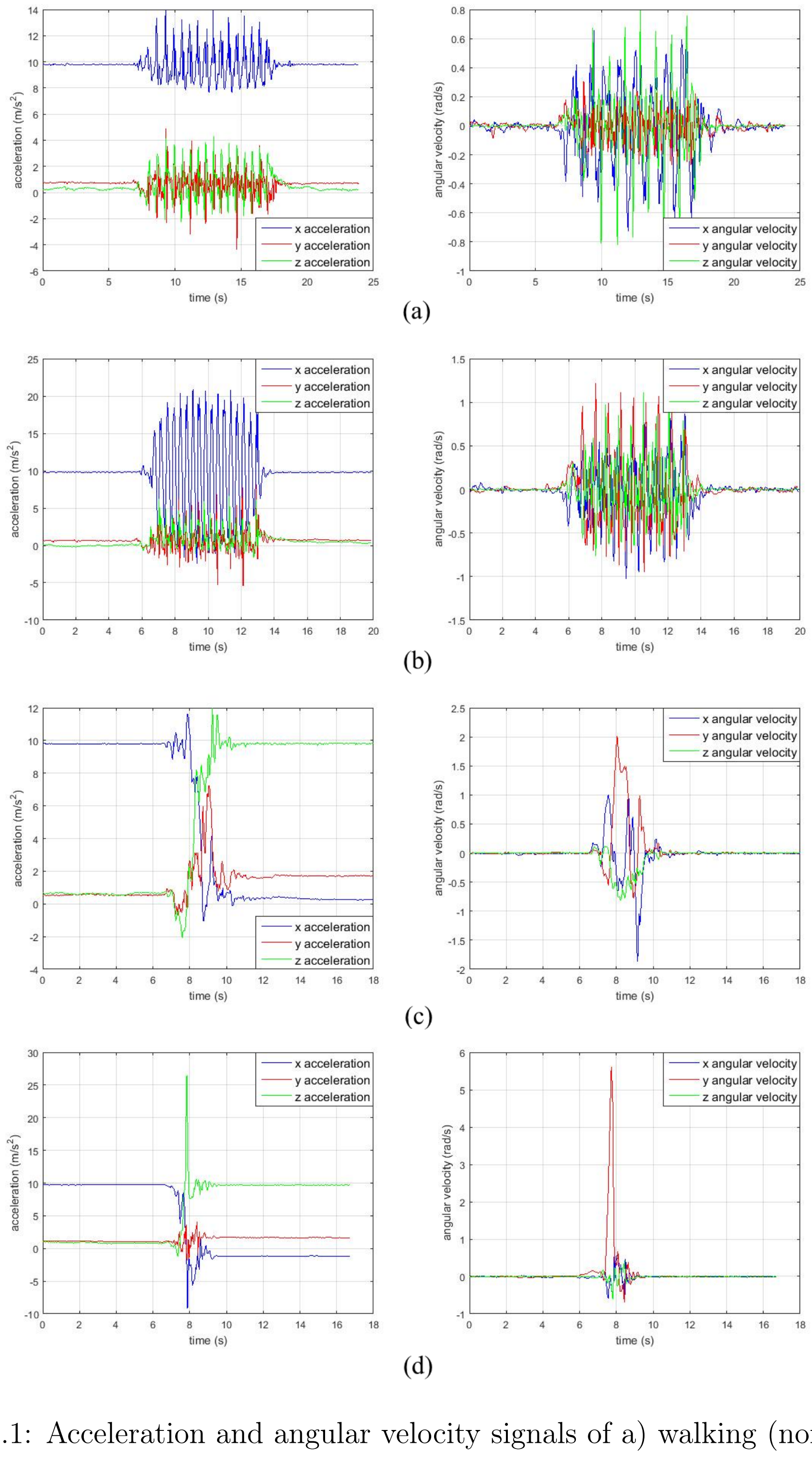

Figure 2.1: Acceleration and angular velocity signals of a) walking (non-fall), b) running (non-fall), c) standing to lying on bed (non-fall), and d) forward fall.

and falling out of a bed. Acceleration and angular velocity signals from fall-like ADLs, running and standing to lying on bed, that are provided in parts b) and c) of Figure 2.1, respectively, show that at least some ADLs can also have acceleration peaks and valleys, and high acceleration values. When a vast range of fall and non-fall activities is considered, adequately complex algorithms need to be developed in order to embrace every possible fall and non-fall scenario. Thus, the need for an extensive dataset arises. Having carefully examined the characteristics of the signals originating from fall and non-fall activities, and the differences between them, we develop a heuristic fall-detection algorithm that uses two simple features extracted from the data acquired from a waist-mounted motion sensor unit.

2.1 The Proposed Algorithm

The proposed heuristic algorithm uses total acceleration and total angular velocity data from the waist of the subject to extract two simple features named maxSVtot and maxMULT, respectively: maximum total acceleration and the maximum of the signal resulting from element-wise multiplication of total acceleration ($\mathrm{SV_{TOT}}$) and total angular velocity ($\mathrm{G_{TOT}}$) signals:

$$\mathrm{SV_{TOT}} = \sqrt{{a_x}^2 + {a_y}^2 + {a_z}^2} \tag{2.1}$$

$$\mathrm{G_{TOT}} = \sqrt{{\omega_x}^2 + {\omega_y}^2 + {\omega_z}^2} \tag{2.2}$$

where a_x, a_y, a_z, ω_x, ω_y, and ω_z represent acceleration and angular velocity signals in the x, y, and z directions, respectively. The calculation of the two features used in the proposed algorithm are as follows:

$$\mathrm{maxSVtot} = \max_{1 \leq i \leq N} \mathrm{SV_{TOT}}(i) \tag{2.3}$$

$$\text{maxMULT} = \max_{1 \le i \le N} \left(\text{SV}_{\text{TOT}}(i) \cdot \text{G}_{\text{TOT}}(i) \right) \tag{2.4}$$

where N is the length of single trial data as well as the signals SV_{TOT} and G_{TOT}.

These two features, maxSVtot and maxMULT, are selected as falls mostly cause high acceleration and rotation in the torso of the subject. Upon capturing high accelerations at the waist with the first feature, the second feature captures the data points where both acceleration and angular velocity are high at the waist of the subject. These two simple features bear enough information to yield satisfying performance without costing substantial time and computational power to compute.

The algorithm then employs double thresholding to separate and classify the extreme data, that is, it utilizes upper and lower thresholds for each feature. The data instances (feature vectors) whose first or second feature is above the upper threshold or below the lower threshold for the corresponding feature are identified as extreme data. The selection of the upper and lower thresholds, as well as the thresholds of the other algorithms, is done using cross validation, and this procedure is further explained in Section 2.3. Once the thresholds for the features are selected, any extreme data are directly assigned to one of the classes (fall or non-fall) with certainty. After this step, membership values of the non-extreme data to both of the classes are extracted using the values of their features.

Fuzzy logic is employed to calculate the membership values of the non-extreme data [82]. To this end, we assume a linear distribution of the features of non-extreme fall and non-fall instances. Because data instances with maxSVtot higher than the upper threshold are detected as falls with certainty, fall membership values of these instances are taken as 1, whereas their non-fall membership values are taken as 0. Similarly, data instances with maxSVtot lower than the lower threshold have fall membership values of 0 and non-fall membership values of 1. Thereupon, a fall membership line that takes the value of 0 at the lower

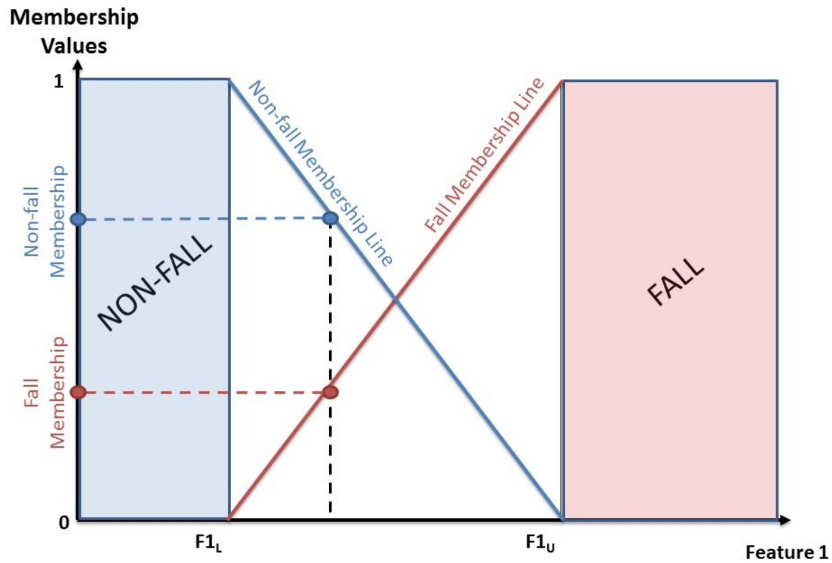

Figure 2.2: Calculation of Membership Values.

threshold and the value of 1 at the upper threshold is created, as well as a non-fall membership line with the reversed borderline values (Figure 2.2). Any non-extreme data instance takes a fall and a non-fall membership value between 0 and 1 corresponding to its feature value between the lower and upper thresholds. The same procedure is then followed for the second feature, maxMULT, to obtain the second fall and non-fall membership values of each non-extreme data instance before proceeding with the final part of the algorithm. Figure 2.2 shows the calculation of the fall and non-fall membership values for one of the two features.

Membership values of every non-extreme data instance for both features are averaged to obtain one fall membership and one non-fall membership value for that particular data instance. These membership values to the two classes are finally compared with each other and the data instance is assigned to the class with the larger average membership value. Figure 2.3 displays the flowchart of the proposed algorithm. In the figure, Feature 1 (and F1) is used instead of maxSVtot and Feature 2 (and F2) is used instead of maxMULT for compactness, while subscripts L and U indicate lower and upper thresholds, respectively. The order of the features within the figure signify their importance, that is, the feature that has more capability of distinguishing fall and non-fall activities is used before the other. The MATLAB code of the proposed algorithm is provided in

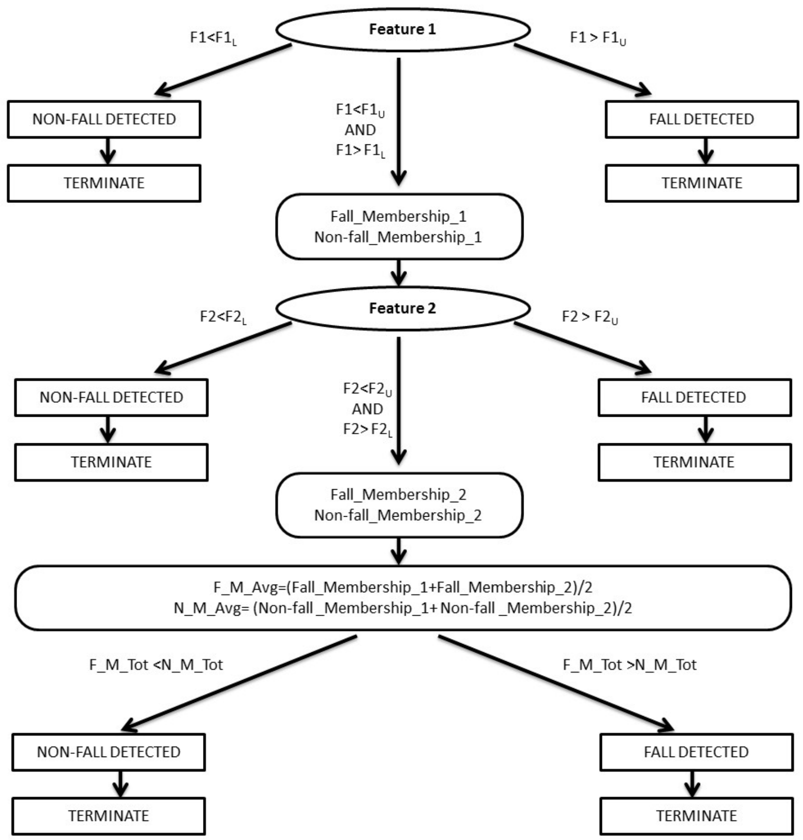

Figure 2.3: Heuristic Fall-Detection Algorithm.

Appendix A.1. Because the algorithm is a **f**uzzy-**a**ugmented **do**uble **th**resholding, we name it FADoTh.

It should be noted that, the assumption that non-extreme fall and non-fall data would have a linear distribution between the lower and upper thresholds is a strong one. Not every feature assumes such a behavior; moreover, it is not always possible to obtain such a classification of extreme values. For instance, in some features (e.g., minimum of total acceleration) both the left and right parts of the extreme data instances may belong to one of the classes and thus the proposed membership lines may not work properly. This may, then, require a different profile of membership distributions, that is, possibly of higher order, to better fit the distribution of the non-extreme data. It is, therefore, of great significance to examine the distribution of selected features thoroughly, prior to using them in the FADoTh algorithm. We have done this and have observed that the two features that are selected in this work have the desired linear distributions.

As mentioned in Chapter 1, the comparative evaluation of fall-detection algorithms on an extensive dataset that is collected according to the comprehensive protocols put forth previously in the literature [76] is highly valuable, as the current literature lacks studies that compare different algorithms. Before providing the details of the comparative evaluation of the algorithms from the literature in Section 2.3, the description of the dataset that is used throughout this thesis to evaluate all algorithms is provided in the next section.

2.2 The Dataset

The dataset that is used in this thesis was originally acquired by Özdemir and Barshan [28] in 2014 to evaluate different ML based fall-detection algorithms. It was collected by following the protocols defined by Abbate et al. [76] with the approval of Erciyes University Ethics Committee. A total of 16 young and healthy subjects performed the scripted activities with their informed written consent. The seven female subjects had an average age of 21.5, an average weight of 58.5 kg and an average height of 169.5 cm, whereas the nine male subjects averaged 24 years in age, 67.5 kg in weight and 172 cm in height. The subjects performed the activities on a soft floor mat, with protective equipment on their head, wrists, elbows, and knees in order to avoid injuries (Figure 2.4 a)-c)).

Six wireless MTw sensor units from Xsens Technologies, each consisting of three tri-axial devices (an accelerometer with a working range of ± 120 m/s^2, a gyroscope with a working range of $\pm 1200°$/s, a magnetometer with a working range of ± 1.5 Gauss) and an atmospheric pressure sensor with a working range of 300–1100 hPa were fixed to the head, waist, chest, right wrist, right thigh, and right ankle of each subject. Figure 2.4 illustrates the configuration of the sensor units on the subject's body as well as the axes of each sensor unit. Data were collected at a sampling frequency of 25 Hz and sent to a PC via a ZigBee

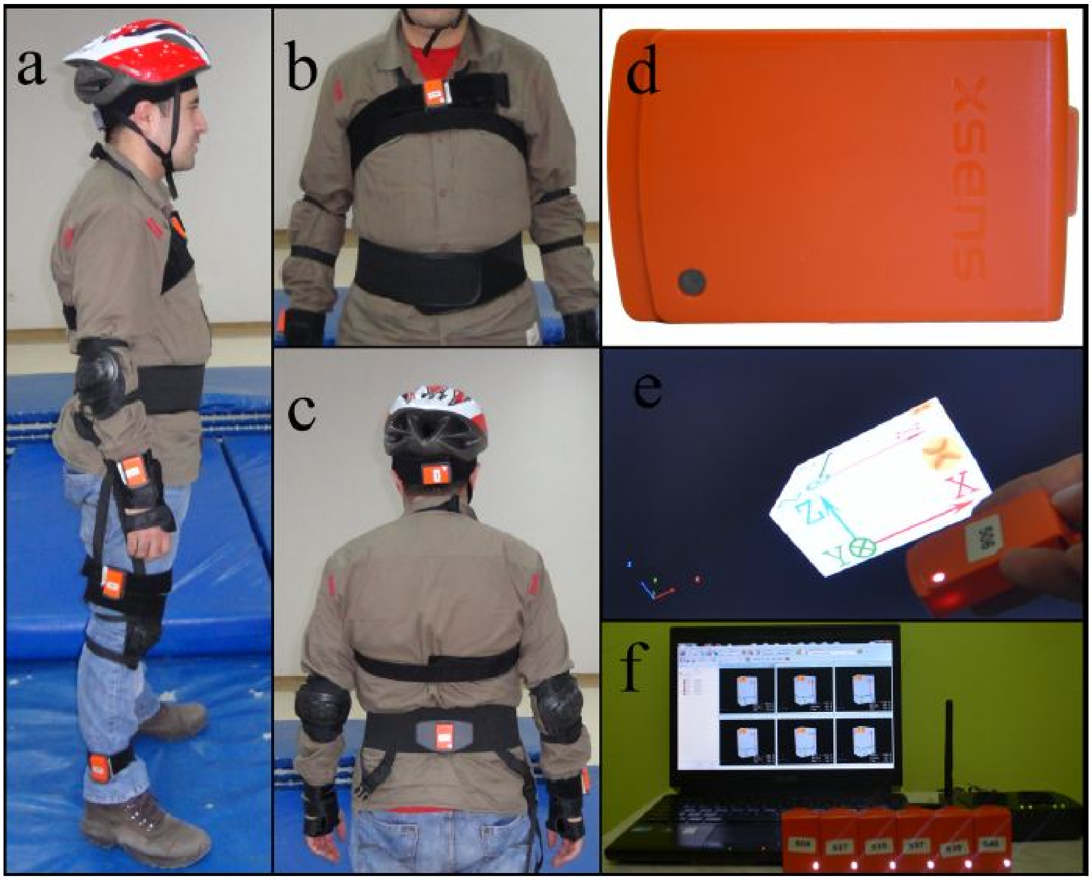

Figure 2.4: a)-c) Configuration of the sensor units on the subject's body, d) MTw sensor unit, e) axes of a sensor unit, f) connection to a PC and the interface [28].

connection for storage.

Each subject performed five different executions of 16 non-fall activities (ADLs) and 20 fall activities. A broad span of activities was selected, in agreement with the guidelines in [76], to capture most of the real-world activities so that the evaluation of developed algorithms can produce realistic outcomes. A list of these activities, with a brief explanation of each activity, is given in Table 2.1. Common ADLs were included in the dataset as well as near-fall activities in order to construct a genuine representation of real-life activities. Included fall activities also embrace a wide variety of fall types that can be encountered in real-life scenarios.

A vast dataset comprising 2880 trials was obtained: 1280 non-fall trials and 1600 fall trials. Each trial of duration 10–15 seconds was recorded and stored in a separate .txt file. All of the features are extracted over the whole duration of single trial data. In addition, three-point median filter is applied to the data before extracting the features in the proposed algorithms, in order to eliminate the high-frequency noise components of the signals.

Table 2.1: Activities in the Dataset.

No.	Description	Type
1	walking forward	non-fall
2	walking backward	non-fall
3	running	non-fall
4	squatting and then standing	non-fall
5	bending at about 90°	non-fall
6	bending to pick up an object	non-fall
7	walking with a limp	non-fall
8	stumbling with recovery	non-fall
9	ankle sprain	non-fall
10	coughing/sneezing	non-fall
11	standing to sitting on a hard surface (chair)	non-fall
12	standing to sitting on a medium surface (sofa)	non-fall
13	standing to sitting on air	non-fall
14	standing to sitting on a soft surface (bed)	non-fall
15	standing to lying on bed	non-fall
16	lying on bed to standing	non-fall
17	standing to falling forward to the floor	fall
18	standing to falling forward to the floor with arm protection	fall
19	standing to falling on knees	fall
20	standing to falling on knees and then lying down	fall
21	standing to falling forward with quick recovery	fall
22	standing to falling forward with slow recovery	fall

23	standing to falling forward, ending in right lateral position	fall
24	standing to falling forward, ending in left lateral position	fall
25	standing to falling down on the floor, ending sitting	fall
26	standing to falling backward, ending lying	fall
27	standing to falling backward, ending in right lateral position	fall
28	standing to falling backward, ending in left lateral position	fall
29	standing to falling on the right side, ending lying	fall
30	standing to falling on the right side with recovery	fall
31	standing to falling on the left side, ending lying	fall
32	standing to falling on the left side with recovery	fall
33	from lying, rolling out of bed and falling on the floor	fall
34	standing on a podium to forward fall on the floor	fall
35	syncope — standing to falling vertically	fall
36	syncope fall, slowly slipping off a wall on the side	fall

Although the dataset contains data from six different sensor units on the body of the subjects, only the data from the waist sensor unit are used throughout this study. This results from an attempt to render the proposed algorithms more feasible to embed in a hardware system, because not only the cost of a system which requires six sensor units would be considerably high, such a system would also be quite obtrusive to the user. As established in the literature survey in Chapter 1, the waist is one of the most commonly used and best performing places to capture motion signals from the subjects for fall detection, along with the chest [41, 42].

It must be emphasized that the dataset was collected from young and healthy subjects performing simulated falls in a laboratory setting with protective equipment. Although it would have been preferable to have real fall data from elderly subjects or subjects from certain disease groups, current subject profiles and fall categories were chosen with the aim of gathering an extensive dataset [28]. It has been stated in Chapter 1 that the real-life fall data or data involving elderly subjects are limited and difficult to collect because of long waiting times and fragility of the elderly. We believe this situation should not hinder the results obtained in this thesis, because it has been reported in [80] that real-life falls by older people bear similar characteristics to simulated falls.

After importing the dataset, three loops for activities, subjects, and trials are run to span every data instance from separate .txt files and extract the necessary features. Every data instance is labeled as non-fall (denoted by 0 in the MATLAB code) if the index of the activity is smaller than or equal to 16 or as fall (denoted by 1 in the MATLAB code) if the index of the activity is greater than 16. A record of the subject index for each data instance is kept to be used in subject-based cross validation, which will be explained in detail in the next section.

2.3 Comparative Study

2.3.1 State-of-the-art Algorithms

The FADoTh algorithm and 15 state-of-the-art heuristic fall-detection algorithms are evaluated based on the above-mentioned dataset and compared to each other. Before proceeding to the comparison procedure, the selected algorithms are briefly described below:

Abbate: The work by Abbate et al. [48] is based on improving the detection accuracy of a basic fall-detection system by filtering out the false alarms that are caused by three fall-like ADL types. The basic system includes an upper threshold to $\mathrm{SV}_{\mathrm{TOT}}$ with the detection of a static interval after the impact, where $\mathrm{SV}_{\mathrm{TOT}}$ does not exceed the threshold. After this basic system, AAMV feature is used to distinguish sitting/lying on a bed, sofa, or a chair that causes false alarms. Furthermore, FFI and FFAAM features are used to detect the jumping activity and eliminate the false alarms that it causes. The expansions of the acronyms

AAMV, FFI, and FFAAM are available underneath Table 1.1 in Chapter 1. Because these three features are custom created by the authors of this work and their descriptions are lengthy, it is left to the reader to resort to [48] for further details.

Anania: This algorithm by Anania et al. [49] uses an upper threshold to total angular velocity followed by an upper threshold to the trunk angle change. It also checks if these two peaks are close to each other before detecting a fall.

Baek: The algorithm by Baek et al. [50] first determines the posture of the subject by employing thresholds to roll, pitch, and yaw angles obtained by exploiting trigonometric relationships between low-pass filtered acceleration signals in x, y, and z directions and these angles. Once the lying posture is detected, upper thresholds to $\mathrm{SV_{TOT}}$ and $\mathrm{G_{TOT}}$ are employed to detect falls.

Bourke-1: This algorithm by Bourke et al. [45] uses a single upper threshold to $\mathrm{SV_{TOT}}$. Raw data are filtered with a second-order digital Butterworth low-pass filter (LPF) with a cutoff frequency of 250 Hz.

Bourke-2: This algorithm by Bourke et al. [52] uses a single lower threshold to velocity. Raw data are filtered with a second-order digital Butterworth LPF with a cutoff frequency of 15 Hz. To prevent drift in the velocity integration, a second-order digital Butterworth band-pass filter with cutoff frequencies of 0.15 Hz and 15 Hz is applied to the resulting velocity signal.

Bourke-3: This algorithm by Bourke et al. [83] employs upper thresholds to sum vector magnitudes of angular acceleration, angular velocity, and angular change signals in roll and pitch axes. Raw data are filtered with a second-order digital Butterworth LPF with a cutoff frequency of 100 Hz. Angular acceleration signal is obtained by taking the first difference of the angular velocity signal and the angular change is obtained by numerical integration of the angular velocity signal in time windows of 1.7 s.

Chen: This algorithm by Chen et al. [54] relies on detecting an impact by employing an upper threshold to $\mathrm{SV_{TOT}}$ and if the threshold is exceeded, calculating the change in the orientation of the body of the subject after the impact. If the orientation of the body after the impact is significantly different than that before the impact, a fall alarm is raised.

Jacob: Jacob et al. [56] propose an interesting algorithm which uses seven different flags that are raised when seven different features exceed the corresponding thresholds. Separate thresholds are used for $\mathrm{SV_{TOT}}$, $\mathrm{G_{TOT}}$, $\mathrm{A_{SVM}}$, total angular change, and angle changes in the x, y, and z directions. If more than four of these flags are raised, a fall is detected. Angular acceleration is obtained by numerical differentiation and the angular changes are calculated by numerical integration.

Jantaraprim: This algorithm by Jantaraprim et al. [57] employs a lower and an upper threshold to the $\mathrm{SV_{TOT}}$ signal. When $\mathrm{SV_{TOT}}$ value falls below the lower threshold before it exceeds the upper one, the time indices of these two elements of $\mathrm{SV_{TOT}}$ are inspected and if they are close to each other, a fall is detected. Raw data are filtered with a second-order digital Butterworth LPF with a cutoff frequency of 20 Hz.

Kangas-1, Kangas-2, and Kangas-3: Kangas et al. [59] have proposed three different fall-detection algorithms with low complexity. The first algorithm, Kangas-1, is based on detecting the impact of a fall and the posture of the subject after the impact. Detection of the impact is performed by applying an upper threshold to the dynamic total acceleration ($\mathrm{SV_D}$), where detection of the posture is achieved by exploiting the low-pass filtered version of the vertical acceleration ($\mathrm{Z_{LPF}}$).

The second algorithm, Kangas-2, is based on detecting the start of the fall, impact moment, and the posture. The start of the fall is detected by applying a lower threshold to $\mathrm{SV_{TOT}}$, while the detections of the remaining two events are the same as in Kangas-1.

The third algorithm, Kangas-3, is based on detecting the start of the fall, fall velocity, impact, and posture. What is different than Kangas-2 is that, the fall velocity is calculated by integrating SV_{TOT} signal from the start of the fall until the impact.

A three-point median filter is applied to the raw data before extracting any features in all three algorithms. In order to obtain SV_D, acceleration data are filtered with a second-order digital Butterworth high-pass filter with a cutoff frequency of 0.25 Hz and the sum vector magnitude of the resulting acceleration is calculated. For the calculation of posture, a second-order digital Butterworth LPF with a cutoff frequency of 0.25 Hz is used.

Lindemann: Lindemann et al. [46] use upper thresholds for the total acceleration in the horizontal plane ($SV_{TOT\text{-}HOR}$), SV_{TOT}, and total velocity before the impact. $SV_{TOT\text{-}HOR}$ is calculated by simply exploiting the configuration of the sensor unit, whereas total velocity is calculated by backward integration of SV_{TOT} over a 1.5 second window from the peak of SV_{TOT} backwards. All raw data are filtered with a LPF with a cutoff frequency of 80 Hz.

Sorvala: This algorithm by Sorvala et al. [65] uses upper thresholds to SV_{TOT} and G_{TOT} in order to detect impact. Once the impact is detected, angular change of the body is compared to a threshold and a fall alarm is raised if it exceeds the threshold. Raw acceleration data are filtered with a three-point median filter, whereas a second-order digital Butterworth LPF with a cutoff frequency of 0.25 Hz is employed to obtain angular change based on acceleration signals.

Wang: Wang et al. [67] developed an algorithm that uses an upper threshold to SV_{TOT}. If SV_{TOT} exceeds the threshold, then a fall alarm is raised. If SV_{TOT} does not exceed the threshold, $SV_{TOT\text{-}HOR}$ is compared to an upper threshold. If it exceeds the threshold, the linear velocity is calculated by backward integration and a fall alarm is raised if the velocity is greater than an upper threshold.

2.3.2 Comparison Methodology

No additional LPF is applied to the raw data other than the filters that are specified in the individual algorithms; however, 10 data samples from the beginning and the end of every trial recording are trimmed off and excluded from any processing because they bear corrupted data caused by switching the sensor units on and off.

Almost every heuristic algorithm relies on the use of thresholds and the selection of these thresholds should be done properly. Although every study that is considered here states the optimal parameter values for their algorithms, these parameters are selected according to their own dataset and do not necessarily give the best or most realistic results in another dataset. Consequently, a subject-based cross validation is employed in this study to determine the optimal parameter values and realistically evaluate the performance of each algorithm.

First, the dataset is partitioned into eight folds with respect to the subjects: seven partitions of data from one male and one female subject each and one data partition from two male subjects. This way, eight folds, each consisting of the data of a pair of subjects, are created. In a loop, each of these eight folds is kept as the test data while the remaining seven are combined to be used as the training data. For each iteration of this loop, a grid search is conducted for the parameters of each algorithm to obtain the set of parameters that gives the best classification accuracy on the training set. After the parameter set that yields the highest accuracy on the training set for that fold is obtained, the algorithm is evaluated on the test set for that fold with the obtained parameter value(s) and the corresponding confusion matrix is stored.

Fall detection is a binary decision problem that requires a decision on whether a fall has occurred or not. While testing the algorithms, we may encounter four different cases:
a) true positive (TP): a fall occurs and the algorithm detects it
b) false positive (FP): a fall does not occur but the algorithm detects a fall
c) true negative (TN): a fall does not occur and the algorithm does not detect a

		classified		
		p′	**n′**	**total:**
true	**p**	TP	FN	P
	n	FP	TN	N
	total:	P′	N′	

Figure 2.5: A 2×2 Confusion Matrix.

fall
d) false negative (FN): a fall occurs but the algorithm does not detect it

As in radar terminology, a FP can be called a false alarm and a FN can be called a missed detection. Having identified these data, a confusion matrix can be constructed as in Figure 2.5, where P represents the total number of true positives and N represents the total number of true negatives. Furthermore, P′ and N′ represent the total numbers of estimated positives and negatives, respectively.

Based on the obtained confusion matrices and the aforementioned definitions, accuracy, precision, sensitivity (recall), specificity, and F-measure values are calculated. These performance metrics are calculated as follows:

$$\text{Accuracy} = \frac{1}{2} \cdot \left(\frac{\text{TP}}{\text{TP} + \text{FN}} + \frac{\text{TN}}{\text{TN} + \text{FP}} \right) \tag{2.5}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.6}$$

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.7}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{2.8}$$

$$F\text{-measure} = 2 \cdot \frac{\text{Precision} \cdot \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}} \tag{2.9}$$

Based on the definitions given above, FP and FN ratios can be easily obtained: FP ratio $= 1 -$ Specificity and FN ratio $= 1 -$ Sensitivity.

Obviously, there is an inverse relationship between sensitivity and specificity. For example, in an algorithm that employs simple thresholding, as the threshold level is decreased, the rate of FN decreases and the sensitivity of the algorithm increases. On the other hand, FP rate increases and specificity decreases. As the threshold level is increased, the opposite happens: sensitivity decreases and specificity increases. We would like to emphasize that we have used the average of the class-based accuracies in Equation (2.5) rather than the typical definition of accuracy, given by $(\text{TP} + \text{TN})/(\text{TP} + \text{FN} + \text{TN} + \text{FP})$, since the sizes of the two classes are not exactly the same. The F-measure metric in Equation (2.9) is obtained by multiplying the harmonic mean of precision and sensitivity metrics by two and provides a combined measure of a system using the two metrics that have an inverse relationship.

While performing parameter selection, sufficiently large intervals are selected for the parameters to avoid obtaining optimal parameter values on the limits of the intervals; furthermore, the intervals of parameter sweep are updated, should such a situation be encountered.

Subject-based cross validation is preferred over m-fold cross validation in this thesis, because a more realistic evaluation of the algorithms where the data of the test subjects are not used in the determination of parameters is attempted. It is almost certain that any user of such a fall-detection system will not have contributed to this dataset, if any algorithm presented here were to be embedded in a hardware system for real-world use. Therefore, by excluding all of the data of the test subjects from the training data, we seek to avoid optimistic results that may be caused by the correlation among the data from the same subject. This selection, in turn, causes relatively high variations in the performance metrics of the eight different training and test set combinations.

2.3.3 Comparison Results

The performance metrics for each algorithm are presented in Table 2.2 where each performance metric is provided in mean±standard deviation format.

It is evident from Table 2.2 that the FADoTh algorithm has the highest average accuracy, sensitivity, and F-measure among all the algorithms considered. Only Kangas-2 has slightly higher average precision and specificity values; however, this algorithm then gives poor results in the sensitivity metric. The only algorithm that produces an average sensitivity value that is comparable to the FADoTh algorithm is Kangas-3. However, it is not within one standard deviation of the accuracy, precision, and specificity values of the FADoTh algorithm. In fact, there are only two algorithms that are within one standard deviation of the accuracy of the FADoTh algorithm: Kangas-1 and Kangas-2. Sensitivity and precision values usually have a trade-off; thus, it is beneficial to inspect the F-measure metric that combines these two metrics into one. Only two algorithms, Kangas-1 and Kangas-2, are within one standard deviation of the F-measure of the FADoTh algorithm. It can be concluded here that only Kangas-1 and Kangas-2 yield comparable results to the FADoTh algorithm, albeit not better. Although the difference between the accuracy of FADoTh and these algorithms may not seem statistically so significant, we note that subject-based cross validation is employed in this study, which generally yields lower overall accuracy values with more variations.

2.3.4 Runtime Analysis

All of the algorithms are run on MATLAB version R2015a installed on a computer with Intel® Core™ i5-3230M CPU running at 2.60 GHz, 4.00 GB RAM, and Windows 7 Home Premium 64-bit operating system while no other external application or program is running.

In order to determine the runtimes of the algorithms, 10 runs of each algorithm

Table 2.2: Results of the Comparative Study on Heuristic Algorithms.

Algorithm	Accuracy (%)	Precision (%)	Sensitivity (%)	Specificity (%)	F-measure (%)
Abbate [48]	96.52±2.07	98.13±2.49	95.37±2.59	97.66±3.15	96.70±1.84
Anania [49]	90.19±3.00	96.54±4.80	84.44±2.64	95.94±5.84	90.00±2.48
Baek [50]	96.02±1.38	97.90±1.41	94.63±3.03	97.42±1.75	96.20±1.43
Bourke-1 [45]	96.23±1.28	97.61±1.51	95.44±2.80	97.03±1.91	96.48±1.30
Bourke-2 [52]	94.95±1.16	95.05±2.44	96.31±3.18	93.59±3.23	95.61±1.11
Bourke-3 [83]	89.94±2.04	96.85±2.66	83.31±2.19	96.56±2.95	89.55±1.91
Chen [54]	95.20±1.86	98.00±2.35	92.81±2.07	97.58±2.90	95.32±1.65
Jacob [56]	90.28±2.17	94.70±3.34	86.81±3.42	93.75±4.25	90.51±1.97
Jantaraprim [57]	91.97±1.24	96.83±2.30	87.63±4.38	96.25±2.95	91.93±1.68
Kangas-1 [59]	97.80±1.12	98.41±1.97	97.63±1.33	97.97±2.56	98.00±0.88
Kangas-2 [59]	98.05±0.83	**98.94±1.16**	97.44±1.45	**98.67±1.47**	98.17±0.76
Kangas-3 [59]	97.13±1.45	96.92±2.20	98.25±2.36	96.02±2.91	97.55±1.26
Lindemann [46]	96.55±2.07	98.13±2.49	95.44±2.53	97.66±3.15	96.74±1.83
Sorvala [65]	95.02±1.29	96.45±2.52	94.56±4.44	95.47±3.34	95.39±1.50
Wang [67]	97.03±0.61	97.53±1.44	97.19±1.58	96.88±1.86	97.34±0.54
FADoTh	**98.45±1.21**	98.89±1.51	**98.31±1.03**	98.59±1.91	**98.59±1.03**

Table 2.3: Runtimes of the Heuristic Algorithms.

Algorithm	Runtime (ms)	Algorithm	Runtime (ms)
Abbate [48]	**1.242±0.033**	Jantaraprim [57]	5.357±0.026
Anania [49]	5.949±0.044	Kangas-1 [59]	5.955±0.034
Baek [50]	6.064±0.035	Kangas-2 [59]	6.105±0.022
Bourke-1 [45]	5.269±0.025	Kangas-3 [59]	6.420±0.025
Bourke-2 [52]	10.731±0.074	Lindemann [46]	1.307±0.010
Bourke-3 [83]	5.369±0.017	Sorvala [65]	6.111±0.024
Chen [54]	5.974±0.029	Wang [67]	5.344±0.009
Jacob [56]	5.533±0.014	FADoTh	5.461±0.017

on the whole dataset are conducted without cross validation. The average runtime results for the classification of a single data instance are provided in Table 2.3 in mean±standard deviation format.

Although the FADoTh algorithm ranks as the seventh fastest algorithm for classification, it is faster than every algorithm within one standard deviation from its average accuracy value. Furthermore, the only two algorithms that produced comparable performance metrics to the FADoTh algorithm, Kangas-1 and Kangas-2, are considerably slower than the FADoTh algorithm.

2.3.5 Discussion

Majority of the 15 state-of-the-art algorithms failed to produce accuracy values higher than 96% based on the comparison on the same dataset. This situation is similar for the other performance metrics although all of the algorithms compared in this thesis are reported to show high classification performance in the original datasets that they are developed and tested on. This is most likely to be caused by the relative size of the dataset used in this study. To our knowledge, none of the datasets that are used in the publications that these 15 state-of-the-art algorithms are originally proposed in are comparable in size to the dataset employed here. Not only the mere size of the dataset used here is larger than those in the included publications, the range of activities it includes spans far more of real-life scenarios as well. On the contrary, the fall-detection algorithms developed by Kangas et al. [59] perform very well on our large dataset, despite originally using a limited dataset collected from three subjects performing nine fall types for the evaluation. Their success lies in being able to successfully capture the most general defining characteristics of falls. We believe that the FADoTh algorithm can perform very well in other datasets as it exploits two major characteristics of falls: high acceleration and points of high angular velocity with high acceleration.

On the whole, the FADoTh algorithm, using only two simple features, performs better than 15 state-of-the-art heuristic fall-detection algorithms on an extensive dataset comprising a total of 2880 trials of 16 non-fall and 20 fall activities by 16 subjects, when subject-based cross validation is employed for parameter selection and performance evaluation. We believe that the FADoTh algorithm would give satisfying results in real-world applications if embedded in fall-detection hardware.

It seems possible, at this point of algorithm development, that a learner version of the same algorithm can be formulated and tested against other ML classifiers for fall detection. In the next chapter, this developed learner version is explained and compared to eight other ML classifiers based on the same dataset to assess its viability for fall detection.

Chapter 3

A Novel Fall-Detection Algorithm Based on Machine Learning: *k*AT

Two main limitations associated with FADoTh that is proposed in Chapter 2 are class assignments of the extreme data and the distribution of the non-extreme data, as mentioned previously. Focusing on these limitations, we anticipated that better detection results can be obtained with a learner version of that algorithm which would then be able to utilize training data. A learner version of the proposed heuristic algorithm is then formulated and developed, which is able to select the features to apply lower and upper thresholds to and utilize the non-extreme training data. It should be emphasized that the proposed learner, based on machine learning (ML), is a supervised learner requiring the labels of the training data.

3.1 The Algorithm

As with any ML based algorithm, the proposed learner also consists of a training and a testing phase. The procedures for each of these phases are given in the subsequent sections.

3.1.1 Training

The two features that are used in the Chapter 2 are employed in this classifier to observe that they yield an average accuracy value of 97.92% and that the performance can be improved with the addition of four more features. After the raw data are imported and filtered with the same procedure as in the heuristic algorithm, six features are extracted: maxSVtot, minSVtot, maxANGVEL, ANGVELenergy, maxABSLENGTH, and maxMULT. We believe that these features contain sufficient information to distinguish falls from ADLs because they can capture the defining characteristics of falls. Falls are usually characterized by low total acceleration in the free fall phase and high total acceleration together with high angular change of the body at the moment of impact with the ground. The selected features can capture high and low acceleration points, large angular change points, points where both acceleration and angular changes are high, and points where large changes occur in acceleration. We suggest that this feature set captures all aspects of a fall and therefore should span all fall types. Brief descriptions of these features are given below.

maxSVtot: This feature is the same as the first feature of FADoTh given in Chapter 2: it is the maximum value of the $\mathrm{SV_{TOT}}$ signal within the duration of single trial data. The calculation of this feature is given in Equation (2.3).

minSVtot: Related to the first feature, this feature is the minimum value of the $\mathrm{SV_{TOT}}$ signal within the duration of single trial data. This feature is calculated as follows:

$$\mathrm{minSVtot} = \min_{1 \leq i \leq N} \mathrm{SV_{TOT}}(i) \tag{3.1}$$

where N is the length of single trial data.

maxANGVEL: Similar to the first feature, this feature is the maximum value of the $\mathrm{G_{TOT}}$ signal within the duration of single trial data. The calculation of it is as follows:

$$\text{maxANGVEL} = \max_{1 \leq i \leq N} \text{G}_{\text{TOT}}(i) \tag{3.2}$$

where N is the length of single trial data.

ANGVELenergy: As the name suggests, this feature is the energy of the G_{TOT} signal in a window of 160 ms around the maximum peak of SV_{TOT}. With 25 Hz sampling frequency, this interval includes five samples: the sample closest to the peak of acceleration, the two samples preceding it, and the two samples following it. This feature is selected to reflect on activities where drastic rotations occur near the acceleration peak which might not necessarily be very high. The calculation of this feature is as follows:

$$\text{ANGVELenergy} = \sum_{i=k-2}^{k+2} [\text{G}_{\text{TOT}}(i)]^2 \tag{3.3}$$

where k is the index of the maximum peak of the SV_{TOT} signal.

maxABSLENGTH: This feature is selected to give three-sample data windows where rapid acceleration changes occur. First, the absolute difference of each data sample in SV_{TOT} with the previous data sample is calculated and a sequence consisting of these absolute differences for each sample is created. Then, the sum of these absolute differences in three-sample windows are observed. The maximum of these total absolute differences are then assigned as the maxABSLENGTH feature. Readers are directed to Appendix A.2 for the extraction of this feature in the MATLAB environment. This feature is calculated as follows:

$$\begin{aligned} \text{maxABSLENGTH} = \max_{1 \leq i \leq N-3} [&|\text{SV}_{\text{TOT}}(i+1) - \text{SV}_{\text{TOT}}(i)| + \\ &|\text{SV}_{\text{TOT}}(i+2) - \text{SV}_{\text{TOT}}(i+1)| + |\text{SV}_{\text{TOT}}(i+3) - \text{SV}_{\text{TOT}}(i+2)|] \end{aligned} \tag{3.4}$$

where N is the length of single trial data.

maxMULT: This feature is the same as the second feature of FADoTh given in Chapter 2: it is the maximum value of the signal resulting from the element-wise multiplication of SV_{TOT} and G_{TOT} signals. This feature is selected in order to capture the activities where both high accelerations and body rotations are observed. The calculation of this feature is given in Equation (2.4).

These six features are selected by hand and all of them are used in the algorithm. After the features are extracted and the training and test data are separated, the features are normalized using the means and standard deviations of the features of the training data. The training phase consists of taking the training data and the parameter α — which will be explained later — and then designing the classifier. The structure of this classifier will be disclosed later.

First, let us consider a typical distribution that a feature of non-fall and fall data typically assumes, displayed in Figure 3.1. In this figure, the minimum and maximum feature values of non-fall and fall data and RANGE (the distance between the lower one of the maximum values and the greater of the minimum values of the two classes) are shown. Although the assumption that the left and right parts of extreme data would belong to two different classes is still valid here, this does not cause any loss of generality in the explanation of the proposed algorithm.

As the first step of training, thresholds should be set so that extreme and non-extreme data can be discriminated. Although it would be intuitive to assign the lower threshold to non-fall$_{\text{min}}$ and the upper threshold to fall$_{\text{max}}$ at a first glance, this option would mean overfitting to the training data. To illustrate another risk in such a practice, in Figure 3.1, non-fall$_{\text{min}}$ corresponds to a data instance that is far away from the nearest non-fall data and it probably corresponds to an outlier or erroneous data. Thus, setting the lower threshold to this value would worsen the performance of the algorithm. Even in the absence of such anomalies, setting the thresholds to these extreme values would be naïve and fail to deliver the best results; hence, we use the first parameter: α.

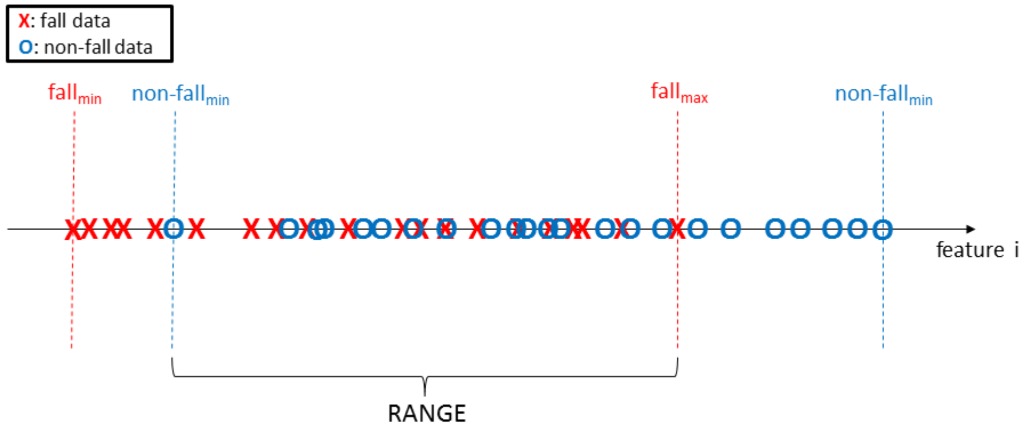

Figure 3.1: A Typical Distribution of a Feature of Non-fall and Fall Data.

This parameter is the offset ratio of the upper and lower thresholds, that is, a measure of how far away the upper and lower thresholds are from fall$_{\max}$ and non-fall$_{\min}$, respectively. Specifically, the lower and upper thresholds for any feature (Fi) are set as shown in Equations (3.5) and (3.6), respectively, using the parameter α.

$$Fi_L = \text{non-fall}_{\min} - \alpha \cdot \text{RANGE} \tag{3.5}$$

$$Fi_U = \text{fall}_{\max} + \alpha \cdot \text{RANGE} \tag{3.6}$$

As α is a coefficient of RANGE, the minimum value that it can take is -0.5. This borderline value of the parameter results in an overlap of upper and lower thresholds as the lower threshold would be equal to non-fall$_{\min}$+0.5·RANGE and the upper threshold would be equal to fall$_{\max}$−0.5·RANGE with this value of the α parameter, and RANGE=fall$_{\max}$−non-fall$_{\min}$ by definition. Extreme negative values of the parameter result in narrow non-extreme data interval and consequently large numbers of false alarms and missed detections in the classification of extreme data. Extreme positive values of this parameter, on the other hand, would result in overly conservative separation of the extreme data and accuracy would drop in the classification of the non-extreme data. In the evaluation of the proposed learner, cross validation is applied with a grid search for the α parameter ranging from -0.3 to 0.3. We observed that small positive values between 0.083 and 0.150 yield the best results.

For a single value of the α parameter, the algorithm decides on which feature to use among the six available features. To do so, lower and upper thresholds and the percentage of non-extreme data are calculated for each feature. Among the six features, the feature that yields the lowest percentage of non-extreme data is selected to be used as the first feature because it provides more extreme data that can be directly classified into classes. After the feature and the thresholds are set and the extreme data are classified, non-extreme data are stored along with their labels to be used in testing later. The process of determining the feature and thresholds to be used in the test phase, produces an element (node) of the classifier. The structure of a node is explained later.

Following the formation of the first node of the classifier, the same procedure is repeated for the non-extreme data of the first node: the data instances that do not exceed the upper threshold of the first feature and are not below the lower threshold of the first feature. This process for creating the nodes of the classifier is iterated until the size of any of the non-extreme fall or non-fall data is less than 10% of the total training data or the classifier consists of a total of six nodes. The first terminating condition is selected empirically, as excessively small values of percentage lead to overfitting and excessively large values lead to poor utilization of the thresholds. The second terminating condition is selected as the total number of features, because the learner may have to use a feature multiple times and probably overfit the training data if the training is not terminated at a node count of six.

At the end of training, a classifier structure comprising a maximum of six nodes is provided as output. Figure 3.2 illustrates the structure of a classifier that is produced by the proposed learner. Each of the nodes of the classifier structure includes the feature to be used (denoted as ftr in the figure), lower and upper thresholds (denoted as thr_l and thr_u in the figure, respectively), labels of the two parts of the extreme data (denoted as lbl_l and lbl_u in the figure),

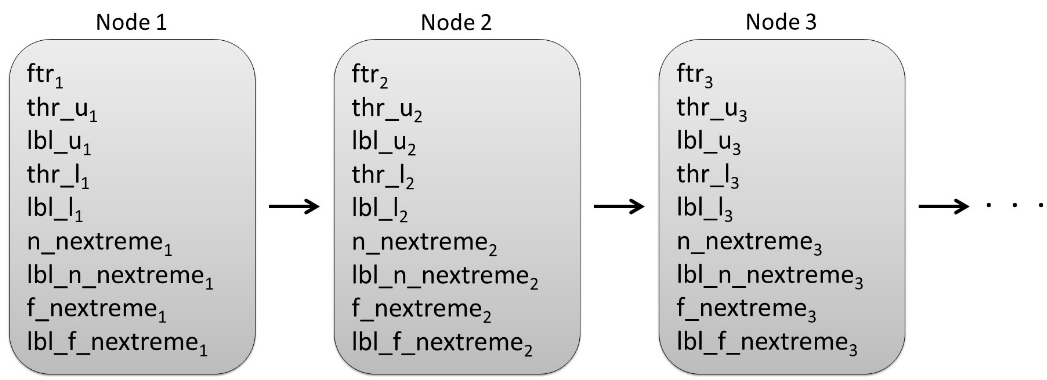

Figure 3.2: The structure of the classifier and its nodes.

and the non-extreme non-fall and fall data along with their labels (denoted as f_nextreme, lbl_f_nextreme, n_nextreme, and lbl_n_nextreme in the figure, respectively). Although the classifier in the figure is shown to have more than three nodes, there is not a minimum number of nodes in the classifier, that is, the proposed learner can even produce a classifier that has a single node. It should be noted again that this algorithm does not have the two limitations of the heuristic algorithm: it can work with features where the two parts of the extreme data belong to the same class (for instance, when both lbl_l and lbl_u are fall actions) and the distribution of the non-extreme data does not necessarily comply with a prescribed distribution profile.

3.1.2 Testing

In the testing phase, already normalized features of the test data and the labels of the feature vectors are provided as input to the testing function, as well as the classifier and the parameter k.

Similar to the heuristic algorithm, test data that exceed any of the thresholds are directly assigned to the corresponding classes and the algorithm is terminated for them. For the non-extreme data that do not exceed the thresholds of any node of the classifier, a modified version of the k-NN classifier is implemented. Contrary to the standard k-NN classifier, the proposed learner takes the k nearest neighbors in the feature space consisting of only the features that are used in the classifier structure. Throughout the development of the algorithm, we figured that the features that are used in the classifier nodes, that is, those that can separate a large percentage of the data into extreme data and classify them, perform better when k-NN is applied to the non-extreme data.

To explain in detail, the non-extreme training data of the very end node of the classifier are transferred to a feature space with reduced dimensions, which includes only the features that are used in the formation of extreme and non-extreme data and the classifier in the training phase. After this transfer is done, any test instance that does not exceed any thresholds of the classifier is compared to these non-extreme training data in this space with reduced dimensions. Euclidean distances of every non-extreme training data to this test data are calculated and the k data instances with the smallest distances are selected, hence, the name k nearest neighbors. Finally, the test instance is assigned to the class that is more frequent among these k nearest neighbors (majority voting). In the case of an even k number and equal number of nearest neighbors from both classes, the test instance is assigned to the non-fall class.

The proposed learner obviously has a tree structure due to the double thresholding it applies to the features. Furthermore, it bears elements of boosting since it utilizes a collection of "weak learners" (decision stumps); however, the weights of the training instances are equal and are not modified in the proposed learner, contrary to boosting methods.

Since k is also a parameter of the learner, a grid search is conducted for this parameter in the evaluation of the algorithm with cross validation. We considered k values ranging from 1 to 51 and observed that those between 20 and 34 provide the best results. More detail on this parameter selection process with cross validation is given in the next section. The MATLAB code of the proposed learner algorithm is given in Appendix A.2, along with the code of the parameter selection method. Algorithm 1 provides the pseudocode of the proposed learner. Because the proposed learner is a $\boldsymbol{k}$-NN classifier-**a**ugmented **t**ree, we name it

kAT.

Algorithm 1: Pseudocode of the kAT algorithm.

TRAINING PROCEDURE

Inputs: Training Set Features, Training Set Labels, α Parameter

for Each One of the Six Features **do**

  Calculate the upper and lower thresholds with the α parameter for this feature

  Separate the extreme and non-extreme data according to these upper and lower thresholds

  Calculate the percentage of the non-extreme data

end for

Select the feature with the lowest percentage of non-extreme data

Construct the node and add it to the end of the classifier

if Size of the non-extreme data is greater than 10% of the training data AND Classifier has fewer than six nodes **then**

  Call the training function for the non-extreme data and this α value

end if

TEST PROCEDURE

Inputs: Classifier, Test Set Features, Test Set Labels, k Parameter

for Each Node of the Classifier **do**

  Separate and classify the extreme data

  if This is the final node of the classifier **then**

    Calculate the distances of the non-extreme data instances to the test instances in the feature space of reduced dimension

    Select k nearest neighbors

    Conduct majority voting

    Assign the non-extreme test instances to the most frequent class among k nearest neighbors

  end if

end for

Once the classification of the whole test set is completed, a confusion matrix is obtained and the five performance metrics (accuracy, precision, sensitivity, specificity, and F-measure) are computed based on this matrix, as given in Equations (2.5)–(2.9).

3.2 Comparative Study

In this section, the evaluation of the kAT algorithm on the dataset in [28], in comparison with the performances of eight different ML classifiers on the same dataset, is presented. First, brief explanations of the selected classifiers are provided. Secondly, the methodology of the comparative study is given, followed by its results. Brief remarks on the runtime analysis of the compared algorithms are presented before providing a discussion of their overall performances.

3.2.1 ML Classifiers Selected for the Comparative Study

Eight state-of-the-art ML classifiers are selected for comparative evaluation with the kAT algorithm based on the same dataset: Bayesian decision making (BDM), least squares method (LSM), k-nearest neighbor classifier (k-NN), artificial neural networks (ANN), support vector machines (SVM), decision tree classifier (DTC), random forest (RF), and adaptive boosting (AdaBoost). Five of these classifiers have parameters which are optimized using a grid search similar to the kAT algorithm. Brief explanations of these classifiers are provided below, together with the intervals of the grid searches for the optimization of their parameters. More detail on the classifiers can be found in [84, 85].

BDM: BDM is based on fitting multi-variate Gaussian distributions to the data from each class — in our case, two classes — and obtaining the mean vectors and covariance matrices of these fitted Gaussian distributions to each class. Once these parameters are obtained, the training process is completed and the testing phase continues with calculating the *a posteriori* probabilities of each test data for both classes and assigning the test data to the class that gives the larger *a posteriori* probability. No parameters need to be optimized for BDM.

LSM: In the training phase of classification with the LSM, the mean vectors of the data from each of the two classes are calculated and stored. In testing, sums of squared distances of a test instance to the mean vectors of each class are calculated and the test instance is assigned to the class whose mean is the closest to it on the feature plane. LSM has no parameters to be optimized.

***k*-NN:** Training of the k-NN classifier comprises the storage of the training feature vectors. In the testing phase, the Euclidean distances of a test feature vector to every training feature vector are calculated and the nearest k training feature vectors are selected. The test feature vector is then assigned to the most frequently occurring class among these k training feature vectors. The parameter k is optimized through a grid search, taking integer values from 1 to 50. Optimal k values widely range from 3 to 41 among different combinations of test and training data.

ANN: ANNs are networks of units called neurons, arranged in multiple layers [86]. In this study, an ANN with only a single hidden layer of neurons is considered. The input layer (the very first layer) includes the input neurons, each of which takes the value of a feature in the feature vector; therefore, the number of neurons in the input layer is exactly the same as the number of features used. In the hidden layer (the layer in the middle), there are multiple neurons, each of which is connected to the neurons in the input layer with unique weights. Each neuron in the hidden layer then applies a non-linear activation function to the weighted sum of the input neurons using the unique weights from each of the input neurons to itself. A similar connection is then made from the hidden layer to the output layer (the third layer), which includes only a single neuron in the scope of this chapter. Once a non-linear activation function is applied to the weighted sum of the outputs of all neurons in the hidden layer, the output of the ANN is obtained. In this study, a sigmoid function in the form of $g(x) = (1 + e^{-x})^{-1}$, where x is the weighted sum of the outputs of the neurons in the previous layer, is used as the activation function in all hidden and output neurons. An output that is smaller than 0.5 corresponds to a non-fall activity, whereas an output greater than 0.5 corresponds to a fall activity. The connection weights are initialized with a uniform random distribution between 0 and 0.2, and training is performed using iterative online (stochastic) back-propagation method with a learning rate of 0.3. The algorithm is terminated when there is not a significant reduction in the average errors of all training data. While learning the weights corresponds to finding the weights of the ANN, testing is equivalent to multiplying the feature values of the test data with the obtained weights to identify its label. The number of hidden-layer neurons is optimized through a grid search where integer values from 1 to 50 are used. Optimal values for this parameter range between 8 and 38.

SVM: SVM is based on creating a hyperplane in the feature space that has the maximum margin between two classes in the training data. A kernel function can be utilized to transform the original feature space to another space, which can prove to be useful when the training data are not linearly separable in the original feature space. In this thesis, Gaussian Radial Basis Function kernel, $f_{\mathrm{GRBF}}(\vec{x},\vec{y}) = e^{-\gamma||\vec{x}-\vec{y}||^2}$ is used, where $\vec{x}$ and $\vec{y}$ are two data vectors. After the transformation to another space, an optimization is run to find the hyperplane that maximizes the margin between the data of the two classes and C is the penalty parameter of this optimization. MATLAB's built-in LIBSVM toolbox is used for the implementation of SVM. The parameters C and γ are optimized through a grid search where they both range from 10^{-5} to 10^{5} on a logarithmic scale. Optimal values of C range from 10^{-2} to 10^{2}, whereas optimal values of γ are found to be between 10^{-3} to 10^{-1}.

DTC: DTC is an ML classifier based on decision stumps: a simple structure applying a threshold to one feature, that is, comparing the value of that feature to a threshold to determine if it is higher or lower. At each node of a DTC, there is a decision stump to produce two branches which are then connected to two other nodes. Final nodes of a DTC, which do not utilize splits (a feature and a threshold value) and thus do not lead to any more nodes, are called leafs. At the leaf nodes of a DTC, the decision about the estimated class of the data instance is made. The features and the thresholds to be used in the nodes of a DTC are selected using a splitting criterion in a greedy fashion; that is, the split that yields the optimal value of a specific criterion is selected at each node without considering the optimality of the overall tree structure. In this study, Gini impurity criterion is selected as the splitting criterion, which is a measure of the frequency of incorrectly classifying a randomly chosen data instance when it is classified according to the distribution of the classes in that split. The value of this criterion is zero when the nodes created by that split are pure. Building the tree structure from the training set until all leafs are pure; that is, all training instances in one leaf belong to a specific class, results in overfitting to the training set. Therefore, different pruning techniques are employed to prevent trees having an excessive depth and number of nodes. Prepruning technique is employed to prevent DTC from overfitting, which involves rejecting to add any more nodes when building the tree when the size of a node is sufficiently small, even though the node is not pure. There are no parameters to optimize in DTC.

RF: RF is a special classifier employing bagging (bootstrap aggregating) technique: an ensemble learning technique which involves taking a large number of subsets from the training set and combining the results of the classifiers that are trained by each of these subsets to obtain the final decision. As the name suggests, RF is based on training a large number of DTCs with the randomly generated subsets of the training set and combining the results of each classifier to end up with a final decision. Because training a large number of DTCs is computationally intensive, a randomly selected subset of the features of a smaller size is used their training. Using the decisions of a large number of decision trees trained on subsets of the training set, RF eliminate the problem of overfitting that is usually pronounced with DTCs. Two parameters are optimized in the subject-based cross-validation process: the number of trees to be trained is ranged from 40 to 200 to observe that the optimal parameter values are between 80 and 140, and the number of features to be used in the training of trees is varied from 1 to 6 to obtain optimal values between 3 and 6.

AdaBoost: AdaBoost is a boosting algorithm, that is, it utilizes the whole training set to iteratively train a large number of weak learners. At the first iteration, a weak learner is trained where all instances in the training set have equal weights. Afterwards, at each iteration, the weights of the training instances that are incorrectly classified by the previous weak learner are increased to train a new weak learner with the updated weight distribution of the training set. The AdaBoost algorithm utilizes decision stumps as weak learners, and the number of weak learners to be used is a parameter that needs to be optimized. For the comparative study presented here, the number of weak learners is changed from 40 to 200 to obtain optimal parameter values between 70 and 190.

In the next section, the details of the comparative study that is undertaken are given.

3.2.2 Comparison Methodology

The same six features that are described in Section 3.1 are used for all algorithms in order to make a fair comparison. Since the number of features is not large, feature reduction is not performed in the evaluation of the eight ML classifiers as well. As with the kAT algorithm, all eight ML classifiers presented above are trained and tested with the normalized features.

Although subject-based cross validation is preferred over m-fold cross validation in the evaluation of the ML based algorithms as well, the procedure is different from that in Chapter 2 because of the nature of the ML algorithms that necessitates training. Similar to the procedure in the evaluation of the heuristic algorithms, the whole dataset is separated into eight folds in a subject-based fashion: seven folds of data from one male and one female subject and one fold of data from two male subjects. In an outer loop, each one of these eight folds is separated to be used as the test set whilst the remaining seven folds are combined to form the training set, which will be used for parameter optimization and training. In an intermediate loop, a grid mesh of parameter values is swept and specific parameter value(s) is considered in each iteration of this intermediate loop. In every iteration of the intermediate loop, training data are divided into seven sub-folds of data from the same combinations of subjects that formed it in the first place. In an inner loop, each one of the seven sub-folds is taken as the test set for the parameter selection (called sub-test set to avoid confusion) and the remaining six sub-folds are combined to obtain the training set for the parameter selection (called sub-training set to avoid confusion). In every iteration of the inner loop, the learner is trained using the designated parameter value(s) and sub-training set combination and tested using the designated parameter value(s) and sub-test set combination. After the inner loop is completed, the average of the sub-test accuracies of all seven runs are calculated. At the end of the intermediate loop, the parameter value(s) that yield the highest average sub-test accuracy is selected. At the end of one iteration of the outer loop, the learner is trained using the corresponding training set (seven of eight folds) and the selected parameter value(s) and tested with the corresponding test set (one of eight folds) and the selected parameter value(s). The confusion matrix for each of these training and test combinations obtained in every iteration of the outer loop is stored. At the very end of the subject-based cross validation with parameter optimization, eight confusion matrices are obtained. A pseudocode of this subject-based cross validation with parameter optimization is given in Algorithm 2. In addition, the MATLAB code of the kAT algorithm in Appendix A.2 also includes the code for the procedure of partitioning the data into folds and sub-folds together with subject-based cross validation with parameter optimization.

The next section presents the performances of the kAT algorithm and the eight state-of-the-art ML classifiers which are evaluated on the dataset acquired in [28] using this cross-validation method.

3.2.3 Comparison Results

Based on the eight confusion matrices stored in the outer loop of the cross-validation process, eight accuracy, precision, sensitivity, specificity, and F-measure values are calculated for each learner. Afterwards, the mean and standard deviation values of these five performance metrics are calculated and presented in Table 3.1 in mean±standard deviation format.

Algorithm 2: Pseudocode of the Cross Validation with Parameter Optimization.

```
Divide the whole dataset into eight folds with respect to the subjects
for Each Fold {Outer Loop} do
  This fold is the test set
  Remaining seven folds are combined to obtain the training set
  Best Accuracy = 0
  for Each Value of the Parameter(s) {Intermediate Loop} do
    Divide the training set into seven sub-folds with respect to the subjects
    for Each Sub-Fold {Inner Loop} do
      This sub-fold is the sub-test set
      Remaining six sub-folds are combined to obtain the sub-training set
      Train the learner with the sub-training set and the parameter value(s)
      Test the learner with the sub-test set and the parameter value(s)
      Store accuracy
    end for
    Find the average accuracy over all seven iterations of the Inner Loop
    if Average Accuracy > Best Accuracy then
      Optimal Parameter Value(s) = This Parameter Value(s)
      Best Accuracy = Average Accuracy
    end if
  end for
  Train the learner with the training set and the optimal parameter value(s)
  Test the learner with the test set and the optimal parameter value(s)
  Store the confusion matrix
end for
```

Table 3.1: Results of the Comparative Study on ML Algorithms (AB: AdaBoost, Acc: accuracy, Pr: precision, Se: sensitivity, Sp: specificity, F-m: F-measure).

classifier	Acc (%)	Pr (%)	Se (%)	Sp (%)	F-m (%)
BDM	98.23±1.21	97.76±2.06	99.12±0.88	97.11±2.75	98.43±1.05
LSM	94.34±1.29	98.17±1.81	91.56±2.19	97.81±2.19	94.72±1.21
k-NN	98.33±1.35	98.55±2.19	98.50±1.07	98.12±2.89	98.51±1.18
ANN	98.75±0.70	99.01±1.02	98.75±0.89	98.75±1.29	98.87±0.62
SVM	98.75±1.04	98.85±1.84	98.94±0.86	98.52±2.41	98.88±0.91
DTC	98.61±1.15	98.07±1.96	**99.50±0.60**	97.50±2.57	98.77±1.01
RF	98.58±1.25	98.41±1.80	99.06±0.94	97.97±2.33	98.73±1.11
AB	98.58±1.04	98.36±1.75	99.13±0.69	97.89±2.29	98.73±0.92
kAT	**98.85±0.81**	**99.14±1.19**	98.81±0.88	**98.91±1.52**	**98.97±0.72**

It is observed in Table 3.1 that the kAT algorithm provides the highest accuracy, precision, specificity, and F-measure among all ML classifiers. SVM shows an average accuracy value that is very close to that of the kAT algorithm; however it then fails to match the performance of the kAT algorithm in other metrics besides also displaying larger variations in those metrics. ANN yields more consistent results (with smaller standard deviation) than the kAT algorithm on four of the five performance metrics; nonetheless, it fails to match the performance of the kAT algorithm in any of the performance metrics. DTC produces a very good average sensitivity value of 99.50%; however it then yields average precision and specificity values that are not so high. In fact, five of the eight ML classifiers yield higher average sensitivity values than the kAT algorithm, but none can perform superior to it or other ML classifiers when all performance metrics are considered.

We would like to note that the performances of the compared classifiers are rather close to each other. In particular, seven of the eight ML classifiers are within one standard deviation of the accuracy, precision, sensitivity, and F-measure of the kAT algorithm. Moreover, ANN and SVM yield remarkably close results to the kAT algorithm. This situation is relatively different than that in Table 2.2 where only a few of 15 state-of-the-art heuristic algorithms can display results that are comparable to FADoTh. It must be noted here that the classifiers in this comparison are ML classifiers and are based on learning the data. ML classifiers have been shown to perform better than heuristic algorithms in the fall-detection realm before [29]. Therefore, the fact that the majority of the compared ML classifiers perform better than the best performing heuristic algorithm is not surprising. It should also be noted that all ML classifiers evaluated here use six features, whereas most of the heuristic algorithms in Table 2.2 use less. Regarding the closeness of the results of BDM, k-NN, ANN, SVM, DTC, RF, AdaBoost, and the kAT algorithm, these seven ML classifiers are already state-of-the-art algorithms that are capable of learning extremely complex patterns. Each of the three best performing algorithms, ANN, SVM, and the kAT algorithm, yield almost 99% accuracy on an extensive dataset, whereas the performances of the k-NN, BDM, DTC, RF, and AdaBoost classifiers are not considerably inferior to these three learners despite displaying higher variations in their results. Overall, the LSM classifier exhibits inferior performance to the remaining eight. BDM, k-NN, ANN, SVM, DTC, RF, AdaBoost, and the kAT algorithm yield comparable results to each other and their relative performances can change in an evaluation based on a different feature set and dataset.

3.2.4 Runtime Analysis of ML Classifiers

As in Chapter 2, all of the algorithms are run on MATLAB version R2015a installed on a computer with Intel® Core™ i5-3230M CPU running at 2.60 GHz, 4.00 GB RAM, and Windows 7 Home Premium 64-bit operating system while no other external application or program is running.

To determine the runtimes of the algorithms, subject-based cross validation of each algorithm is run without parameter optimization. First, the whole dataset is divided into eight folds of data based on subject pairs as before and in a loop, each one of these eight folds is used as test data while the remaining seven are combined to obtain the training data. In each iteration of this loop, the learners are trained using the corresponding training data and fixed parameter value(s) and then

Table 3.2: Runtime Results of the ML Algorithms.

classifier	Training (μs)	Testing (μs)	Total (μs)
BDM	0.94±1.57	198.70±3.86	199.64±4.01
LSM	0.24±0.26	**28.01±0.90**	**28.25±0.92**
k-NN	—	527.49±126.46	527.49±126.46
ANN	2010.40±98.99	83.97±25.28	2094.37±103.28
SVM	310.20±12.33	68.09±15.78	378.29±22.55
DTC	57.76±22.61	124.58±69.44	182.33±90.58
RF	7050.21±119.39	73.70±74.63	7123.92±176.36
AB	19807.98±707.74	13707.04±490.71	33515.02±1173.08
kAT	13.95±8.25	420.36±68.51	434.32±76.14

tested with the corresponding test data and the same parameter value(s). This process is analogous to executing only the outer loop in Algorithm 2. Both the training and testing times of the learners are recorded for each iteration. Training, testing, and total times are given in Table 3.2 in terms of mean±standard deviation for the classification of a single data instance. The total time is the sum of the training and testing times. The same abbreviations as in Table 3.1 are used.

From Table 3.2, it is observed that the kAT algorithm is the fifth fastest algorithm among all the ML classifiers with respect to the total time. While the kAT algorithm is not the fastest, it is faster than SVM, k-NN, RF, and AdaBoost, four algorithms that perform comparable to it in Section 3.2.3. It should be stressed that the fastest algorithm, LSM, did not match the performance of the kAT algorithm. SVM, on the other hand, not only produced comparable results to the kAT algorithm in the previous section, but is also faster. On the other hand, training of the kAT algorithm is considerably faster than that of SVM. Concerning the training times of the learners, k-NN does not require any learning time as the learning phase corresponds to storing the training data. Following k-NN, LSM and BDM are trained quickly since their simple training procedures make use of optimized vector operations of MATLAB. The optimization processes in the training phases of the ANN and SVM learners require longer training times. While DTC does not require particularly long runtimes to train, RF and AdaBoost classifiers that depend on constructing a large number of DTCs then need significantly long training times. Testing times of LSM and SVM are the lowest when compared to the other learners because the testing procedure corresponds to simple calculation and comparison in these two learners. Testing of the ANN algorithm is faster than that of BDM, k-NN, and the kAT algorithm because it utilizes the optimized vector multiplication functions of MATLAB. Despite requiring a substantial time to train, RF classifier is quicker than six of the compared classifiers to test, because the test procedure corresponds to a number of threshold comparisons. The k-NN classifier displays relatively slow testing times because the distances from one test instance to every other training instance are created. Despite utilizing optimized vector functions of MATLAB, this process is computationally demanding. Despite the high performance that the AdaBoost classifier displays, it takes a total of more than 33 ms to train and test single data instance. This runtime is nearly five times as much as that of the second slowest algorithm, RF. The testing time of the kAT algorithm is considerably more than those of LSM, SVM, and ANN since it involves thresholding and a modified k-NN algorithm.

We would like to refer to the obvious difference between the total runtimes of the ML classifiers in Table 3.2 and the runtimes of the heuristic algorithms in Table 2.3. The apparent speed of ML classifiers compared to the heuristic algorithms by a large margin is not normally expected; the sum of the training and testing times of a ML classifier is usually anticipated to be more than the classification time of a heuristic algorithm. However, the large difference between the runtimes in Table 2.3 and the total runtimes in Table 3.2 arise from one major difference in the evaluation of the two groups of classifiers: while the runtimes of heuristic algorithms also include preprocessing and feature extraction, training and testing times of the learners in this chapter are measured after all preprocessing, feature extraction, and feature normalization have been completed. The latter method could not have been applied in the evaluation of the heuristic algorithms since each algorithm has a different set of features and the runtime of the algorithm is affected by this set. In the evaluation of the ML classifiers, features need to be extracted beforehand for the complicated procedure of cross validation with parameter optimization to be conducted; otherwise, this procedure would take days or even weeks for the evaluation of only a single learner. Furthermore, including the time to do preprocessing and feature extraction would hinder the clarity of the results of the comparison between the training and testing times of different ML classifiers, since all learners use the same set of features and preprocessing and feature extraction take much longer time than training and testing the learners. With an average feature extraction time of 5.6 ms for a single data instance, even the fastest ML classifier, LSM, is slower than eight of the algorithms in Table 2.2, whereas the slowest ML classifier, AdaBoost, is slower than all algorithms in that table. It should be noted that the features used in this chapter are simple features that utilize optimized vector operations of MATLAB, whereas some algorithms in Table 2.2 use features such as velocity. Calculation of this feature involves integrating the acceleration data, which is a computationally intense operation.

3.2.5 Discussion

Performance of the eight ML classifiers are evaluated on the dataset in [28] using subject-based cross validation and compared to the performance of the kAT algorithm. The kAT algorithm yields the highest average accuracy, precision, specificity, and F-measure among all ML classifiers; however, the differences between the results of BDM, k-NN, ANN, SVM, DTC, RF, AdaBoost, and the kAT algorithm are not very significant. Two of the ML classifiers, ANN and SVM, perform comparably well to the kAT algorithm, with their average accuracy and F-measure values only within 0.1% of it. Moreover, SVM, ANN, and the kAT algorithm achieve better performances than FADoTh of Chapter 2. Among these two state-of-the-art ML classifiers, SVM produces high variations in its results in cross validation, whereas the average total runtime of ANN is nearly five times as much as that of the kAT algorithm. Although SVM ran faster than the kAT algorithm, the difference is not acute. BDM, k-NN, DTC, RF, and AdaBoost yield accuracy values that are within one standard deviation of the three best performing learners: ANN, SVM, and the kAT algorithm. The results of the

LSM classifier are considerably inferior to the remaining eight classifiers, despite that LSM is the fastest one among the compared classifiers. Considering the comparable performances of BDM, k-NN, ANN, SVM, DTC, RF, AdaBoost, and the kAT algorithm, any one of these learners can be chosen to be implemented under different scenarios. BDM or DTC can be selected for a shorter total runtime with acceptable performance, while ANN would be more suitable for implementations where less variations of the results are desired. The k-NN classifier can be chosen for a system where lower training times are desired and an ongoing learning from the used test data are planned. The SVM classifier can be of preference when low testing times are important. DTC and AdaBoost can then be implemented for high sensitivity values. The kAT algorithm then provides a reasonable balance between low training time, high performance, and small variations of the results.

We believe that the kAT algorithm performs comparably with BDM, k-NN, ANN, SVM, DTC, RF and AdaBoost, if performance metrics and runtimes are considered altogether. With an accuracy of 98.85% over an extensive dataset when subject-based cross validation is applied and an average total runtime of 434.32 μs, it would provide excellent results if embedded in fall-detection hardware.

Having detected falls with such high accuracies, we suggest that an effective fall-detection system should also be able to classify the direction of a fall. Motivation for developing such an algorithm, a survey of the existing literature on the topic, and the description as well as the evaluation of the proposed fall direction classification system are presented in the next chapter.

Chapter 4

Classification of Fall Direction

We have proposed two novel algorithms to detect falls with waist-mounted wearable motion sensor units: a heuristic algorithm (FADoTh algorithm of Chapter 2) and a learner version of the same algorithm (kAT algorithm of Chapter 3). The former achieves a fall-detection accuracy of 98.45% on an extensive dataset, whereas the latter achieves a slightly better accuracy on the same dataset: 98.85%. Both of these algorithms are compared to the state-of-the-art algorithms of their own type on the same dataset based on subject-based cross validation. While FADoTh performs superior to the 15 state-of-the-art heuristic fall-detection algorithms, its learner version, kAT, yields results that are comparable to eight state-of-the-art ML classifiers. With such results on fall detection, we intend to implement a fall direction classification algorithm that can estimate the direction of a fall effectively, should a fall be detected. The overall system with the fall-detection and fall direction classification modules is illustrated in Figure 4.1. Upon receiving the activity data, the system employs feature extraction and classifies the data into a fall or ADL activity employing either a heuristic algorithm or an ML classifier. If a fall is detected, fall direction classification module is activated. In this module, new features are extracted from the raw data and an algorithm is employed to classify the fall into four basic directions: forward, backward, right, and left.

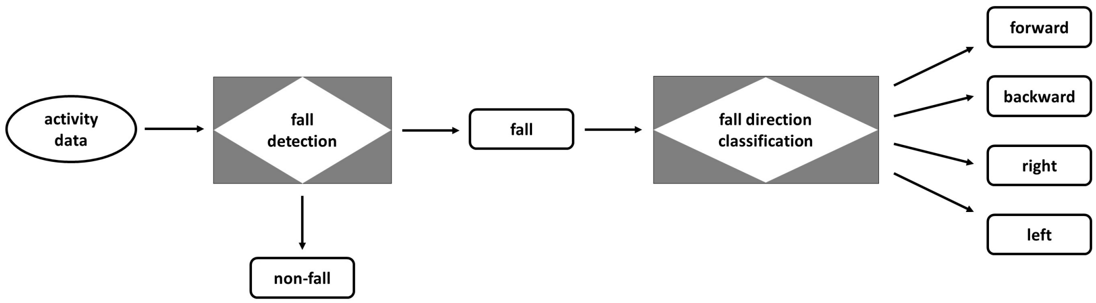

Figure 4.1: The overall system consisting of a fall-detection and a fall direction classification module.

4.1 Motivation and Related Work

In Chapter 1, we presented the need for an effective fall-detection system: such a system would not only benefit the elderly, it would also diminish the harm that people from different fall risk groups are likely to suffer from falls. Falls can cause serious and even life-threatening injuries to the elderly and special disease groups, and injuries that could tarnish the professional performance of workers and athletes. We therefore think that it is preferable if fall-detection algorithms could offer more than solely detecting falls, which is a binary decision process.

The direction of a fall can contain various cues regarding the nature of the fall. Forward falls are mostly caused by tripping, whereas lateral falls can be caused by loss of consciousness, and backward falls can be due to slipping. The direction of a fall can also be of value in an emergency scenario to assess the situation and apply appropriate treatment. That is, knowing the direction of a fall can lead the emergency response team to a more accurate diagnosis of the state of the subject. Lateral falls are more likely to cause limb and neck injuries whereas backward falls are often associated with the head coming into contact with the ground and therefore are more dangerous. A forward fall is usually more controlled than a lateral or backward fall because people are more experienced to use their arms in the forward position. A wearable fall-detection system usually already includes motion sensor units which can supply adequate information for recognizing the directions of falls. Therefore, once a fall is detected, a fall-detection system can provide invaluable fall direction information without requiring any additional hardware. Typical acceleration, angular velocity, and magnetic field signals of falls belonging to the four basic directions are provided in Figure 4.2. The two fall-detection algorithms that are proposed in this thesis already rely on these motion sensor data from the waist of the subject, which, we presume, are sufficient for effective fall direction classification as well.

Research has been conducted on classifying the type of falls in both the ambient assisted living field [87] and wearable sensors field [31, 66, 88, 89, 90, 91], albeit much limited in quantity compared to the vast literature on fall detection. Almost all of these fall classification algorithms are activated after detecting a fall, except that Choi et al. [89] integrated fall direction classification into the fall-detection algorithm. They classified the activities of the subjects into seven activity types: three ADLs and four falls towards the basic directions — forward, backward, right, and left. Using NB and a novel feature selection method, they were able to classify the activity types of 670 data instances in their dataset with 99.4% accuracy.

Although most wearable sensor based fall classification studies classify falls into four basic directions as given above, Tolkiehn et al. [66] classify falls into three directions: forward, backward, and lateral. Utilizing a sensor node comprising a tri-axial accelerometer and a barometric pressure sensor at the waist of the subject and using a heuristic algorithm employing the azimuth angle of the subject, they were able to achieve a 94.12% fall direction classification accuracy on a dataset consisting of 119 fall instances. Dinh and Struck [90] also consider these three types of falls with the addition of collapse type. Attempting to capture four types of falls, they use accelerometer data collected from the waist of the subject and convert them from Cartesian to spherical coordinates. With an algorithm that utilizes fuzzy logic and ANNs, they achieve an average fall classification sensitivity of 94% over all types of falls, on a dataset collected from five subjects and comprising a total of 100 fall instances.

Majority of the studies on wearable sensor based fall classification use motion sensor units on the waist, chest, or the back of the subject; however, Tao et al. [91] propose a fall direction classification system with force sensor resistors (FSRs) in the shoe insoles of the subjects. They attempt to classify falls into four directions

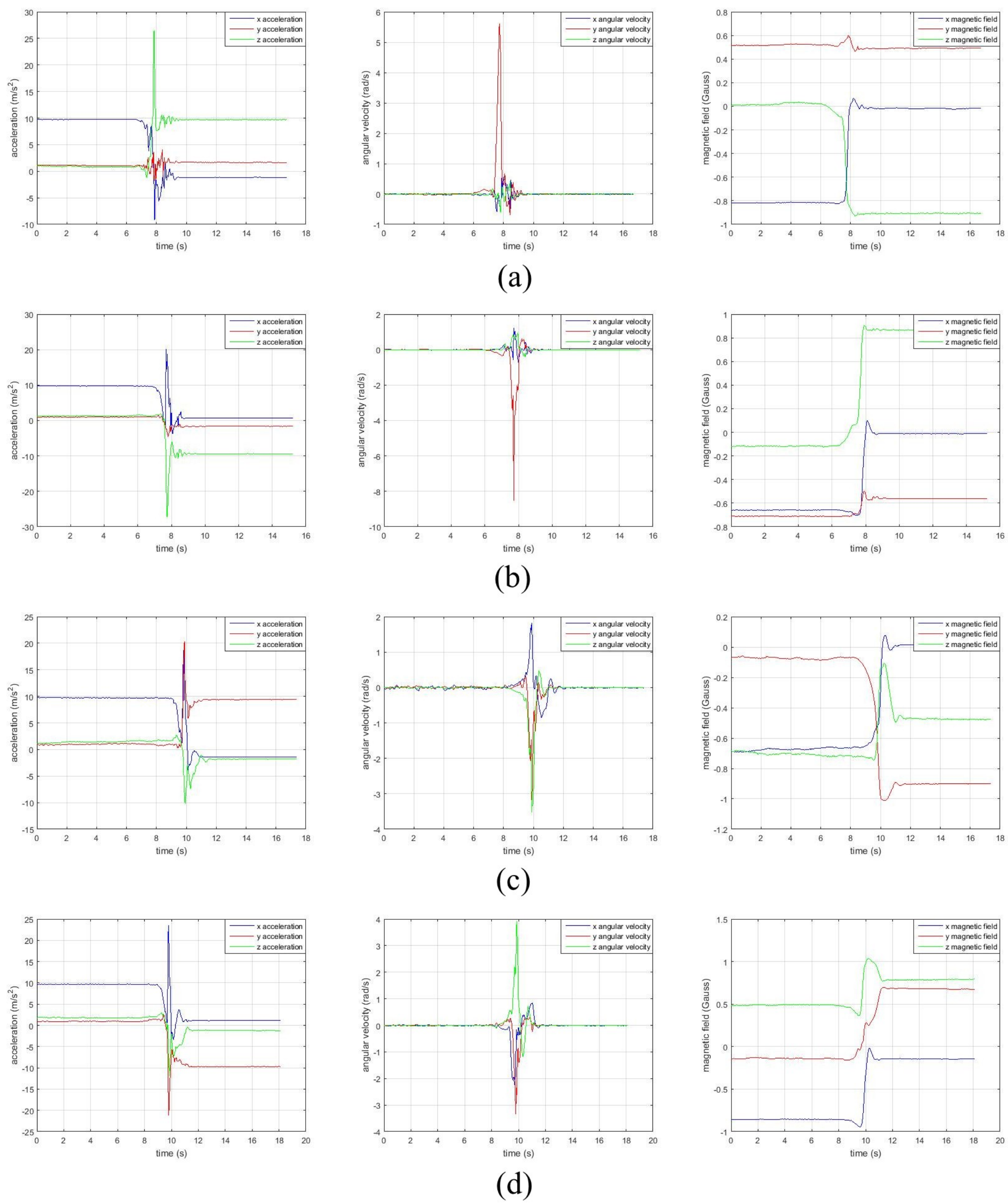

Figure 4.2: Acceleration, angular velocity, and magnetic field signals of a) forward, b) backward, c) right, and d) left falls.

as Choi et al. [89]. Using four FSRs in the insoles of both right and left shoes each, they achieve a fall direction classification of 75% on a very limited dataset of only 12 fall instances.

The study by Albert et al. [88] attempts to classify falls into the four basic directions mentioned above using the accelerometer data from a mobile phone fixed to the back of the subjects. Their dataset for fall classification consists of a total of 223 fall instances collected during laboratory experiments involving 15 subjects. They compare five different classifiers — SVM, sparse multi-nomial logistic regression (SMLR), NB, k-NN, and DT — with a large feature set of 178 features in total, without using any feature reduction technique. They evaluate these classifiers on their own dataset with two different cross-validation techniques: 10-fold and subject-based cross validation. SMLR yields the best classification performance with a classification accuracy of 99.6% with both types of cross validation.

There is need for further research in the fall classification field, especially considering the broad field of fall detection. Existing literature not only lacks in providing effective fall classification systems, but also fails to present an exhaustive evaluation process. In this thesis, eight different ML classifiers are implemented for classifying falls into four basic directions. These classifiers are then evaluated and compared to each other, on an extensive dataset comprising 1600 fall instances using a subject-based cross validation with parameter selection. Besides, four of the eight ML classifiers are modified for robustness analysis against unknown classes. To our knowledge, previous studies on fall direction classification do not tackle the problem in such depth and achieve as high performance. This study adds on the existing work in the literature by evaluating and comparing eight state-of-the-art ML classifiers on a large dataset and conducting a robustness analysis. The datasets used in previous studies are rather limited. Furthermore, this study contributes to the fall direction classification area by modifying four of the eight ML classifiers to avoid making any decisions in the presence of data from an unknown class and conducting a robustness analysis. No other study in the literature considers such an evaluation of their fall classification algorithms.

4.2 Fall Direction Classification System

In this section, the classifiers that are considered for the problem of fall classification are presented. The reduced dataset that is used for fall direction classification is then given before displaying the results of the evaluation and comparison of the classifiers. A brief discussion of the runtime analysis of the considered classifiers is given before the final remarks of the section.

4.2.1 Considered Classifiers

For fall direction classification, eight ML classifiers that are compared to kAT in Chapter 3 are implemented and compared with each other. Although it would have been preferable to use kAT also in fall direction classification, this is not done since it is a binary classifier. It is possible to convert any binary classifier to work in multi-class classification; however, it is avoided in this thesis because of the excellent performance of the eight ML classifiers.

Similar to fall-detection algorithms, raw data are filtered using a three-point median filter. Then, 27 simple features are extracted from the raw data collected from the motion sensor unit attached to the waist of the subject: minima, maxima, and mean values of the accelerometer, gyroscope, and magnetometer data in the x, y, and z directions. When the feature set consisting of these 27 (=9 axes$\times$3 features per axis) simple features is normalized with the same procedure as in Chapter 3, corresponding training and test sets are used in the eight ML classifiers that are also featured in Chapter 3: BDM, LSM, k-NN, ANN, SVM, DTC, RF, and AdaBoost.

4.2.2 The Reduced Dataset

For the evaluation of the considered classifiers in fall direction classification, the dataset that is used in Chapters 2 and 3 will be used partially. Because only fall activities are classified into four basic directions, only fall-type activities from Table 2.1 will be included in this evaluation; that is, the dataset will be reduced in size. Table 4.1 shows the fall types that are considered within the scope of this chapter.

It is observed from the table that there are four classes of falls corresponding

Table 4.1: Fall Activities in the Reduced Dataset.

No.	Description	Direction
17	standing to falling forward to the floor	forward
18	standing to falling forward to the floor with arm protection	forward
20	standing to falling on knees and then lying down	forward
21	standing to falling forward with quick recovery	forward
22	standing to falling forward with slow recovery	forward
23	standing to falling forward, ending in right lateral position	forward
24	standing to falling forward, ending in left lateral position	forward
34	standing on a podium to forward fall on the floor	forward
26	standing to falling backward, ending lying	backward
27	standing to falling backward, ending in right lateral position	backward
28	standing to falling backward, ending in left lateral position	backward
29	standing to falling on the right side, ending lying	right
30	standing to falling on the right side with recovery	right
31	standing to falling on the left side, ending lying	left
32	standing to falling on the left side with recovery	left
19	standing to falling on knees	undefined
25	standing to falling down on the floor, ending sitting	undefined
33	from lying, rolling out of bed and falling on the floor	undefined
35	syncope — standing to falling vertically	undefined
36	syncope fall, slowly slipping off a wall on the side	undefined

to the four basic directions, together with an undefined direction class. Although this table contains all the fall activities in our dataset, only 15 of the 20 different fall types are used in the evaluation of fall direction classification algorithms in this section because the remaining five class types do not have well-defined directions. Specifically, there are eight forward, three backward, two right, and two left fall types. Remaining five fall types are reserved to be used in Section 4.3.

In the reduced dataset that is used in this section, there are 15 types of falls, each of which is performed five times by each one of the 16 subjects. This sums up to a total of 1200 fall instances: 640 forward, 240 backward, 160 right, and 160 left fall instances. Note that the sizes of the data belonging to the four classes are not comparable; that is, size of the forward fall class is almost three times as large as the size of the backward fall class and even four times as large as the sizes of the right and left fall classes. This imbalance in class sizes would corrupt the performance of certain classifiers. For instance, such a distribution of class sizes is likely to cause ANN to learn the weights with a bias towards the forward fall class because there are more instances from that class and every data instance updates the weights once in each epoch in online back propagation. To avoid this problem, every data instance in backward, right, and left fall classes are copied and added to the dataset until the sizes of the classes become comparable; that is, the data from backward fall class are copied twice and added to the dataset while the data from right and left classes are copied three times and added to the dataset to make the sizes of the four classes comparable. Finally, the order of the data instances are rearranged with a random permutation.

4.2.3 Evaluation and Comparison of the Classifiers

Eight ML classifiers using 27 simple features extracted from the accelerometer, gyroscope, and magnetometer data collected from the waist of the subjects are evaluated on the reduced dataset described above. Because the considered ML classifiers are the same as those in Chapter 3, the same evaluation technique is also employed here. The subject-based cross validation with parameter optimization that is explained in Section 3.2.2 of Chapter 3 is used in the evaluation of the fall direction classification algorithms here as well.

For the parameter selection of k-NN, ANN, SVM, RF, and AdaBoost classifiers, separate grid searches for each parameter are conducted. For the k-NN classifier, integer k values ranging from 1 to 50 are used in subject-based cross validation with parameter selection and an optimum k value of 1 is obtained for each of the eight folds. The number of hidden neurons parameter for ANN classifier takes integer values from 1 to 50 and the optimal values range from 19 to 46 for the eight folds. Both the C and γ parameters of SVM are varied from 10^{-5} to 10^{5} and optimal values in intervals $[10^{-2}, 10^{1}]$ and $[10^{-3}, 10^{-1}]$ are obtained for these two parameters, respectively. For RF, the number of DTCs to be trained and the number of features to be used in their training are changed from 40 to 240 and from 1 to 27, respectively. Optimal intervals for these parameters are found to be between 120 and 200, and 1 and 10, respectively. Finally, the number of weak learners for the AdaBoost algorithm is varied from 50 to 250 to find optimal parameter values between 100 and 190.

Eight 4×4 confusion matrices that are obtained in each iteration of the outer loop of the cross-validation process are summed up and shown in Table 4.2 to reflect the overall performance of each classifier. Upon inspecting this table, one can see that forward falls are mostly confused with right and left falls by LSM, k-NN, ANN, SVM, DTC, and AdaBoost. Furthermore, LSM, ANN, RF, and AdaBoost can classify falls from the backward direction class into the left direction class. DTC further confuses falls in the backward direction with those to the right and to the left. Although misclassifications occur, no major problems are observed in Table 4.2 other than those mentioned.

After producing the confusion matrices, accuracy, precision, sensitivity, specificity, and F-measure metrics are calculated separately for each of the four classes and the averages of these metrics over all four classes are taken. Performance metrics of the eight considered ML classifiers are given in Table 4.3 in mean±standard deviation format.

Table 4.2: Confusion Matrices in Fall Direction Classification (AB: AdaBoost).

BDM:

$$\begin{array}{cc} & \text{classified} \\ \text{true} & \begin{bmatrix} 640 & 0 & 0 & 0 \\ 0 & 720 & 0 & 0 \\ 0 & 0 & 640 & 0 \\ 0 & 0 & 0 & 640 \end{bmatrix} \end{array}$$

SVM:

$$\begin{array}{cc} & \text{classified} \\ \text{true} & \begin{bmatrix} 629 & 0 & 9 & 2 \\ 0 & 720 & 0 & 0 \\ 0 & 4 & 636 & 0 \\ 0 & 0 & 0 & 640 \end{bmatrix} \end{array}$$

LSM:

$$\begin{array}{cc} & \text{classified} \\ \text{true} & \begin{bmatrix} 616 & 0 & 15 & 9 \\ 0 & 696 & 3 & 21 \\ 0 & 8 & 632 & 0 \\ 0 & 0 & 0 & 640 \end{bmatrix} \end{array}$$

DTC:

$$\begin{array}{cc} & \text{classified} \\ \text{true} & \begin{bmatrix} 621 & 5 & 1 & 13 \\ 3 & 687 & 12 & 18 \\ 4 & 16 & 620 & 0 \\ 12 & 32 & 0 & 596 \end{bmatrix} \end{array}$$

k-NN:

$$\begin{array}{cc} & \text{classified} \\ \text{true} & \begin{bmatrix} 628 & 0 & 9 & 3 \\ 0 & 720 & 0 & 0 \\ 0 & 0 & 640 & 0 \\ 0 & 0 & 0 & 640 \end{bmatrix} \end{array}$$

RF:

$$\begin{array}{cc} & \text{classified} \\ \text{true} & \begin{bmatrix} 640 & 0 & 0 & 0 \\ 0 & 717 & 0 & 3 \\ 0 & 4 & 636 & 0 \\ 0 & 0 & 0 & 640 \end{bmatrix} \end{array}$$

ANN:

$$\begin{array}{cc} & \text{classified} \\ \text{true} & \begin{bmatrix} 616 & 4 & 13 & 7 \\ 0 & 717 & 0 & 3 \\ 0 & 4 & 636 & 0 \\ 4 & 0 & 0 & 636 \end{bmatrix} \end{array}$$

AB:

$$\begin{array}{cc} & \text{classified} \\ \text{true} & \begin{bmatrix} 619 & 0 & 1 & 20 \\ 0 & 717 & 0 & 3 \\ 4 & 24 & 618 & 0 \\ 4 & 4 & 0 & 632 \end{bmatrix} \end{array}$$

Table 4.3: Results of the Comparison for Fall Direction Classification (AB: AdaBoost, Acc: accuracy, Pr: precision, Se: sensitivity, Sp: specificity, F-m: F-measure).

classifier	Acc (%)	Pr (%)	Se (%)	Sp (%)	F-m (%)
BDM	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
LSM	97.88±1.85	98.04±1.66	97.92±1.83	97.88±1.84	97.98±1.74
k-NN	99.55±1.17	99.57±1.10	99.53±1.20	99.54±1.17	99.55±1.15
ANN	98.68±2.10	98.77±1.91	98.65±2.17	98.67±2.11	98.71±2.04
SVM	99.43±1.20	99.47±1.12	99.41±1.23	99.43±1.20	99.44±1.18
DTC	95.61±3.67	96.10±3.20	95.61±3.72	95.61±3.68	95.86±3.45
RF	99.43±0.89	99.47±0.83	99.43±0.92	99.43±0.90	99.45±0.87
AB	97.73±2.52	97.94±2.37	97.67±2.61	97.72±2.53	97.80±2.48

It is evident from Table 4.3 that BDM achieves perfect classification with the given feature set when evaluated on the reduced dataset with subject-based cross validation. This is an excellent result especially considering the size of the dataset; that is, BDM classifies all fall instances to correct directions without needing any parameter optimization and any complicated features. The performances of three other classifiers are also very good. The k-NN classifier follows BDM with the average values of all performance metrics being above 99.5%. Similarly, SVM and RF yields performance metrics above 99.4%, only within 0.1% of k-NN. Although their performances are not comparable to the top three ML classifiers here, ANN and LSM produce performance metrics above 98.6% and 97.8%, respectively. This overall high performance of the ML classifiers considered show that the feature set chosen for fall classification bears adequate information for reliable and accurate direction classification.

It must be noted that the compromise between precision and sensitivity metrics is not as obvious in this table as it is in Table 3.1. However, this can be explained by the fact that these values are averaged over all classes in this table, contrary to Table 3.1 where these performance metrics are calculated for a binary classification problem using only TP, FP, TN, and FN values.

Table 4.4: Runtime Results for Fall Direction Classification (AB: AdaBoost).

classifier	Training (μs)	Testing (μs)	Total (μs)
BDM	3.05±2.58	1004.39±66.21	1007.44±65.79
LSM	0.51±0.19	**36.53±1.88**	**37.04±1.98**
k-NN	—	327.93±374.67	327.93±374.67
ANN	2605.01±84.74	115.50±14.51	2720.51±92.19
SVM	210.43±27.69	135.92±261.27	346.35±288.18
DTC	362.95±316.11	587.93±1127.23	950.89±1438.55
RF	10170.05±300.71	230.55±378.49	10400.59±671.46
AB	41504.24±4736.56	24810.02±421.73	66314.32±5063.93

4.2.4 Runtime Analysis

All eight ML classifiers are run on the same MATLAB version installed on the same computer as in Chapters 2 and 3. Training and testing times of each algorithm are recorded with the same procedure as in the previous chapter, that is, subject-based cross validation without parameter estimation is executed and training and testing times for each training and test set combination are recorded for each algorithm. Training, testing, and total times of the eight ML classifiers run for fall direction classification are given in Table 4.4 in mean±standard deviation format and for the classification of a single data instance.

It can be observed from Table 4.4 that the total runtime of the LSM classifier is significantly lower than the other algorithms. In fact, it is nearly as low as one tenth of the next fastest algorithm, k-NN. Average total runtimes of k-NN and SVM are on par with each other and significantly lower than the total runtimes of BDM, ANN, DTC, RF, and AdaBoost. Although the total runtime of ANN is over 70 times more than that of LSM, it corresponds to an average of 2.72 ms for the classification of a single data instance. We think that this runtime is feasible for real-world operation. Despite displaying a very good performance, RF displays a total runtime that is nearly as much as four times that of ANN.

Furthermore, AdaBoost requires even more time to train and test single data instance, because it uses all 27 features in each weak classifier that it trains. Considering the relatively large total runtime of DTC, the runtime results of these two algorithms that use a large number of DTCs are expected to be slow.

Compared to Table 3.2, differences occur in the total runtimes of the different classifiers. While the total runtimes of BDM, LSM, ANN, DTC, RF, and AdaBoost are higher than their runtimes in Chapter 3 in proportion to the difference in the number of features used (27 vs 6), k-NN and SVM run faster in fall direction classification. This behavior of the k-NN classifier can be explained with the considerable decline in the size of the training set that the algorithm needs to consider for each test instance, whereas, smaller number of training instances cause the optimization process of SVM to run faster.

4.2.5 Discussion

Eight different ML classifiers are implemented to classify fall actions into four basic directions. A feature set consisting of 27 simple features extracted from the accelerometer, gyroscope, and magnetometer data acquired from the waist of the subject are used in all eight ML classifiers. Classifiers are evaluated based on a reduced dataset comprising a total of 1200 data instances, with a subject-based cross validation with parameter selection.

Perfect classification is achieved using the BDM classifier, whereas SVM, k-NN, and RF achieve average performance metrics above 99.4%. Upon conducting a comparison of the runtimes of the considered classifiers, it is observed that the majority of the selected classifiers can train and test a single data instance roughly in 1 ms time. Besides, the four best performing algorithms — BDM, k-NN, SVM, and RF — achieve smaller total average runtimes than AdaBoost.

It seems that BDM, k-NN, SVM, or RF with the feature set employed here, can be used for superior fall direction classification under real-world scenarios. However, under these scenarios, one needs to consider more than the evaluation process that has been undertaken. We next tackle the problem of fall direction classification in the presence of test data from an unknown class. To our knowledge, such an investigation has not been considered previously in the fall classification realm. Four of the eight ML classifiers are modified and a robustness analysis for these modified fall direction classification algorithms is presented in the next section.

4.3 Robustness Analysis

In the previous section, an evaluation of the eight fall direction classification algorithms, where four basic directions are considered, is performed. Such an evaluation implicitly makes the assumption that all types of falls are in one of these four well-defined directions. However, falls can have no direction or unclear directions in the way they occur, for instance, syncope and falling out of a bed. A fall direction classification system that only classifies a fall instance into one of these four directions would erroneously classify such falls into one of the four directions regardless. An effective fall direction classification system should be able to assign the data instances that do not belong to any of the four basic classes to the unknown (zero or null) class. However, the standard implementations of the eight ML classifiers that are used in this thesis do not provide such a function. Therefore, in this section, the standard implementations of four of these algorithms are modified to obtain satisfactory results in the presence of test data from an unknown class. After this modification, algorithms are included in a robustness analysis where test instances from falls whose directions do not belong to any one of the four basic directions are also used. Modifications that are made on the standard implementations of BDM, LSM, k-NN, and ANN classifiers are briefly explained below.

BDM: Modifications are made both to the training and the test phases of the BDM algorithm. After calculating the mean vectors and the covariance matrices of the classes, *a posteriori* probabilities of each training instance for all classes are obtained in a loop. For each class, the minimum *a posteriori* probability among all training instances is stored to obtain a 4×1 threshold vector of minimum *a posteriori* probability values for each class. In the test phase, if the maximum *a posteriori* probability of a test instance to one of the four classes does not exceed the average of the 4×1 threshold vector over the four classes, the test instance is assigned to the unknown class.

LSM: Similar to the BDM algorithm, modifications to both the training and the test phases of the LSM algorithm are made. Upon calculating the mean vectors of the classes in the training phase, the sum of the squared distance of every training instance to the mean vector of the class that it belongs to is calculated and a 4×1 threshold vector consisting of the maximum of these distances over the whole training set is obtained. In the test phase, after the classification of each test instance, the sum of the squared distance from the test instance to the mean vector of the class that it is assigned to is compared to the minimum value of the 4×1 threshold vector. If the obtained sum of squared distance is greater than this threshold value, the test instance is assigned to the unknown class.

***k*-NN:** A distance threshold similar to the modified LSM classifier is also employed in the k-NN classifier. In the training phase, after storing the training instances, Euclidean distances from every training instance to the others are calculated. The maximum distance from one training instance to another of the same class is then stored for each class. Upon storing this 4×1 distance threshold vector, the test procedure begins. In the standard test procedure of the k-NN algorithm, after the label of the test instance is estimated as the most frequent class among its k nearest neighbors, the distance from the test instance to the nearest neighbor is calculated and compared to the average of the 4×1 distance threshold vector. If the nearest distance is greater than this distance threshold, the test instance is assigned to the unknown class. A grid search for k from 1 to 50 is conducted for parameter optimization in the subject-based cross-validation process that will be explained later, and the optimum k value of 1 is obtained for each fold.

ANN: Before explaining the modifications made to the ANN classifier, it is beneficial to stress one difference that the structure of a multi-class ANN classifier has, when compared to the binary ANN classifier that is implemented in Chapter 3. Contrary to a single output neuron in the ANN structure that is used in fall detection, the ANN used for fall direction classification has four different output neurons, which use the same sigmoid activation function and can display output values ranging from 0 to 1. When a data instance is given as input to the ANN, results obtained in each one of these four output neurons indicate the confidence levels of that data instance belonging to each one of the classes. Bearing this in mind, one can go on to make modifications to the ANN classifier. To make the ANN classifier capable of detecting unknown classes, a confidence threshold is employed. After obtaining the confidence levels of class memberships of each test data through the standard test procedure of ANN, the maximum of these confidence levels is compared to the confidence threshold. If it is less than this confidence threshold, the data instance is assigned to the unknown class. An empirical confidence threshold value of 0.85 is employed in this section. The number of hidden neurons parameter of the modified ANN classifier is optimized in the subject-based cross-validation process. This parameter is varied from 1 to 50 and the optimal parameter values are found to be in the interval of $[2, 22]$ for the eight different folds.

These four modified ML classifiers are evaluated on the reduced dataset that is presented in Section 4.2.2. In that section, only 15 of a total of 20 fall types are used and the remaining five types of falls are excluded from the evaluation of the fall classification algorithms. The remaining five fall types that are labeled as "undefined" in Table 4.1 do not have a well-defined direction unlike the other fall types belonging to one of the four basic directions. These five fall types are, therefore, used here as an unknown fall class to evaluate the robustness of the four modified fall direction classification algorithms. Excluded from any training or validation set in parameter selection, these fall types are to be used only in the test procedure of already trained classifiers with already optimized parameters. There is a total of 400 fall instances in this unknown class; therefore, the total dataset comprises 1600 fall instances.

The same subject-based cross validation with parameter optimization that is employed in Chapter 3 is also used here with a minor modification. After dividing the folds and sub-folds of data and separating the training, test, sub-training, and sub-test sets corresponding to every iteration of the three (inner, intermediate, and outer) loops, data from the unknown class are also separated into eight folds based on the same pairwise subject combinations used previously. Every fold of data from the unknown class is then added to the test set of the corresponding iteration. At the end, every *training* set in the outer loop consists of a total of 1050 (=15 fall types$\times$14 subjects$\times$5 trials) data instances whilst every *test* set in the outer loop consists of a total of 200 (=20 fall types$\times$2 subjects$\times$5 trials) data instances.

The overall confusion matrices of the four modified classifiers, that are obtained by summing up the eight 5×5 confusion matrices that are acquired in the outer loop of the cross-validation process, are provided in Table 4.5. In this table, there is an obvious trade-off between detecting the falls in the four basic directions correctly and rejecting to make a decision when a data instance from the unknown class is encountered. Modified BDM, k-NN, and ANN classifiers display a strict approach in rejecting to make a decision; that is, they classify a considerable amount of falls that belong to one of the four basic directions into the unknown class to then correctly classify the data from the unknown class with high accuracies. Modified LSM algorithm, on the other hand, displays a less strict approach in rejecting to make a decision. It shows high accuracy in classifying fall instances that belong to one of the four basic directions; however, it does not show acceptable performance in rejecting to make a decision in the presence of data from an unknown class.

After obtaining the confusion matrices for each of the eight folds, the same five performance metrics are calculated in the same way as in Section 4.2.3. The results of the robustness analysis for the four fall direction classification algorithms are provided in Table 4.6 in mean$\pm$standard deviation format.

The performance metrics of the modified fall direction classification algorithms in the presence of data from an unknown class are all above 90.8%. ANN achieves

Table 4.5: Confusion Matrices in the Robustness Analysis.

BDM:

$$\begin{array}{cc} & \text{classified} \\ \text{true} & \begin{bmatrix} 592 & 0 & 0 & 0 & 48 \\ 0 & 690 & 0 & 0 & 30 \\ 0 & 0 & 580 & 0 & 60 \\ 0 & 0 & 0 & 624 & 16 \\ 6 & 7 & 0 & 0 & 387 \end{bmatrix} \end{array}$$

k-NN:

$$\begin{array}{cc} & \text{classified} \\ \text{true} & \begin{bmatrix} 572 & 0 & 0 & 0 & 68 \\ 0 & 711 & 0 & 0 & 9 \\ 0 & 0 & 620 & 0 & 20 \\ 0 & 0 & 0 & 632 & 8 \\ 0 & 7 & 0 & 2 & 391 \end{bmatrix} \end{array}$$

LSM:

$$\begin{array}{cc} & \text{classified} \\ \text{true} & \begin{bmatrix} 615 & 0 & 0 & 1 & 24 \\ 0 & 693 & 3 & 15 & 9 \\ 0 & 8 & 616 & 0 & 16 \\ 0 & 0 & 0 & 636 & 4 \\ 11 & 95 & 0 & 28 & 268 \end{bmatrix} \end{array}$$

ANN:

$$\begin{array}{cc} & \text{classified} \\ \text{true} & \begin{bmatrix} 599 & 0 & 0 & 0 & 41 \\ 0 & 708 & 0 & 0 & 12 \\ 0 & 0 & 612 & 0 & 28 \\ 0 & 0 & 0 & 620 & 20 \\ 1 & 2 & 2 & 3 & 392 \end{bmatrix} \end{array}$$

Table 4.6: Results of the Robustness Analysis (Acc: accuracy, Pr: precision, Se: sensitivity, Sp: specificity, F-m: F-measure).

classifier	Acc (%)	Pr (%)	Se (%)	Sp (%)	F-m (%)
BDM	94.51±4.68	94.70±3.73	94.64±4.48	94.51±4.67	94.67±4.10
LSM	92.96±2.05	92.83±2.62	90.89±2.11	92.87±2.05	91.85±2.32
k-NN	96.25±2.78	95.84±2.55	96.30±2.78	96.25±2.78	96.07±2.66
ANN	**96.41±3.11**	**96.20±3.05**	**96.62±2.95**	**96.42±3.11**	**96.41±3.00**

the highest average values in all performance metrics, followed by k-NN, which is within only 0.4% of ANN in all of the performance metrics. BDM and LSM classifiers follow the performances of these two classifiers with performance values above 94.5% and 90.8%, respectively. Comparing the results in this table with those in Table 4.3, one can observe a significant decline in the overall performances of the classifiers. This decline is caused by not only the fall instances with a certain direction that are assigned to the unknown class but also the fall instances from the unknown class that are classified into one of the four directions. This decline in performances, however, is not dire considering the difficult nature of detecting data from the unknown class with simple modifications to the standard ML classifiers. It is also evident in Table 4.6 that the variations (standard deviations) in the results of the performance metrics over different combinations of training and test data are larger than the previous case.

The runtimes of the modified classifiers are also investigated. Because of the additions in the training and test phases of the classifiers, we expect their training, testing, and total times to be considerably higher than those in Table 4.4.

The same procedure as the one in Section 4.2.4 is followed to obtain the training, testing, and total times of the four modified fall direction classification algorithms. The MATLAB version and the PC that are used to run the algorithms are the same throughout this thesis; therefore, the details of these can be found in Sections 2.3.4 and 3.2.4. The same subject-based cross validation without parameter optimization as in Section 4.2.4 is employed here, albeit with a minor modification: the test data belonging to the unknown class is also divided into eight folds of data from the same two-subject combinations as used previously and each of these folds of additional test data from the unknown class is then added to the test data of the corresponding iteration of the loop. Training, testing, and total runtimes of the four modified fall direction classification algorithms are presented in Table 4.7 in mean±standard deviation format and for the classification of a single data instance.

It is observed in Table 4.7 that there are significant changes in the runtimes and especially in the training times of the fall direction classification algorithms, when compared to the runtimes in Table 4.4. The total runtime of the ANN classifier in this table is almost twice as much as that in Table 4.4, whereas the increase is almost three times in BDM, four times in LSM, and more than 14 times in k-NN. These differences were previously predicted to be caused by the fact that every

Table 4.7: Runtime Results for the Robustness Analysis.

classifier	Training (ms)	Testing (ms)	Total (ms)
BDM	1.438±0.016	1.359±0.017	2.797±0.033
LSM	**0.054±0.001**	**0.061±0.002**	**0.144±0.003**
k-NN	2.554±0.010	2.223±0.032	4.777±0.039
ANN	4.734±0.183	0.181±0.039	4.915±0.188

training instance needs to be considered in the training phase in order to obtain the thresholds. The reason for the significant increase in the runtimes of k-NN when compared to the former case is the fact that it calculates the distance from every training instance to every other training instance, meaning that it considers all combinations of pairs for the training set. On the whole, the rankings of the total runtimes of the modified fall direction classification algorithms are the inverse of the rankings of their performances in the robustness analysis. That is, the best performing algorithm, ANN, is the slowest in Table 4.7, which is followed by k-NN, BDM, and LSM, which are the second, third, and fourth best performing algorithms in the robustness analysis, respectively. Even with a total runtime of nearly 5 ms for the classification of a single data instance, ANN yields good results in the robustness analysis and it can be implemented in a real-world fall-detection and classification system.

In this section, BDM, LSM, k-NN, and ANN classifiers are modified to recognize data instances from an unknown class. Highest average classification accuracies of 96.41% and 96.25% are achieved by the ANN and k-NN classifiers, respectively. Although the total runtimes of the modified algorithms are considerably higher than their original versions, they have an essential function in the presence of test data from an unknown class. Besides, the achieved performances in the presence of test data from the unknown class are not comparable to the perfect classification obtained in the classification of falls into four basic directions; however, the results obtained in this robustness analysis bear great importance in the evaluation of the fall classification algorithms in a realistic scenario. This robustness analysis is especially valuable since none of the earlier studies reviewed in Section 4.1 capture the presence of unknown fall types. It should be noted that such an analysis is not needed in a fall-detection study where a binary decision is made, since every activity is either a fall or non-fall activity and therefore an unknown class does not exist. Classifying the direction or type of a fall, however, is a type of activity recognition; thus, this kind of investigation happens to be crucial.

Chapter 5

Summary, Conclusions, and Future Work

In this thesis, robust and effective fall detection and fall direction classification using wearable motion sensor units worn on the waist of the subject are achieved. In the first chapter, a comprehensive literature survey on fall detection with wearable sensors is presented. Afterwards, a novel heuristic and a novel machine learning algorithm are developed for fall detection and compared to 15 state-of-the-art heuristic fall-detection algorithms and eight ML classifiers, respectively, on an extensive dataset. Besides, eight ML classifiers are implemented for superior fall direction classification and four of them are then modified for robustness analysis.

In the second chapter of the thesis, a novel heuristic fall-detection algorithm is proposed: FADoTh. The FADoTh algorithm utilizes two simple features and applies upper and lower thresholds to these features to separate and classify extreme data. Membership values of the non-extreme data instances are calculated by using fuzzy logic and the non-extreme data are assigned to fall or non-fall classes accordingly. A large dataset comprising a total of 2880 data instances — 16 subjects performing 36 different activities five times each — is used for evaluating and comparing the FADoTh algorithm with 15 state-of-the-art heuristic fall-detection algorithms. Subject-based cross validation is used for parameter selection and evaluation of all algorithms, which is more challenging than other cross-validation techniques. The FADoTh algorithm performs the best among all of the heuristic algorithms in terms of accuracy, sensitivity, and F-measure. In the runtime analysis of the heuristic algorithms, the FADoTh algorithm displays faster runtimes than the algorithms that yield comparable performances, despite being the seventh fastest algorithm.

In the third chapter of the thesis, a learner version of FADoTh is presented: kAT. The kAT algorithm uses the parameter OFFSET to determine which feature to use and set upper and lower thresholds. Similar to the heuristic algorithm, extreme data are directly classified and the non-extreme data are stored for the testing phase. In the testing phase, a modified k-NN classifier is employed to classify the non-extreme data, with the second parameter of the kAT algorithm: k. The kAT algorithm is evaluated and compared to eight ML classifiers — BDM, LSM, k-NN, ANN, SVM, DTC, RF, and AdaBoost — on the same dataset with subject-based cross validation that employs parameter optimization. All of the learners are run with six features which are presumably adequate for fall detection. The kAT algorithm yields better performance than the LSM classifier and performs on par with the other ML classifiers: while the average accuracy, precision, specificity, and the F-measure of the kAT algorithm are higher than all classifiers, the margin between the remaining seven ML classifiers and the kAT algorithm is not very significant. The average total runtime of the kAT algorithm is faster than four of the seven ML classifiers that yield comparable results, although SVM, BDM, LSM, and DTC are faster than it.

In the fourth chapter of the thesis, fall direction classification is considered. The same eight ML classifiers using a simple feature set consisting of 27 features are implemented for classifying falls into four basic directions: forward, backward, right, and left. Eight ML classifiers are evaluated on a reduced version of the same dataset comprising 1600 fall instances in total, with the same subject-based cross validation using parameter optimization. BDM achieves perfect classification whereas k-NN, SVM, and RF follow with average values of all performance metrics above 99.5%, 99.4%, and 99.4%, respectively. BDM, LSM,

k-NN, and ANN are then modified to avoid making any decision in the presence of data from an unknown class. A robustness analysis is conducted where 400 test instances belonging to an unknown fall direction class are included in the test set to evaluate the performance of the modified classifiers. ANN and k-NN display satisfactory results with average accuracies of 96.41% and 96.25%, respectively. To our knowledge, the study presented here is the first one in the fall classification field that considers the presence of data from an unknown class to modify the classifiers and conduct a robustness analysis. Secondly, the size of the dataset used for the evaluation of the fall direction classification algorithms is the largest among all studies in the fall classification literature.

In the future, FADoTh and kAT algorithms can be evaluated on public datasets for an even more thorough assessment of their performances. Although the dataset used in this thesis is extensive in size, it consists of segmented data collected from laboratory experiments with young and healthy subjects. A dataset including real-world data from middle-aged or elderly subjects can be acquired and used to evaluate the proposed algorithms in the future. Different ways of segmentation can be considered and compared. Moreover, the kAT algorithm can be improved and evaluated on other classification problems, not only restricted to the fall-detection realm. Although the fall direction classification system is satisfactory in performance, it is susceptible to the misalignment of the sensor unit. This is another direction of academic work that can be carried out in the future. With the pervasiveness of wearable devices, there is an ever growing need for systems that can harvest their own energy. Therefore, energy harvesting can be considered and applied to address the problem of battery maintenance in wearable systems. Furthermore, the algorithms can be embedded in hardware for real-world use. Such an implementation would also enable the evaluation the algorithms based on real-world data collected from real fall and non-fall activities of the middle-aged or even the elderly.

Bibliography

[1] World Health Organization, "Falls," `http://www.who.int/mediacentre/factsheets/fs344/en/`, Accessed on: 20 August 2017.

[2] C. Griffiths, C. Rooney, and A. Brock, "Leading causes of death in England and Wales—how should we group causes?" *Health Statistics Quarterly*, vol. 28, no. 9, November 2005.

[3] D. Wild, U. S. L. Nayak, and B. Isaacs, "How dangerous are falls in old people at home?" *British Medical Journal*, vol. 282, no. 6260, pp. 266–268, January 1981.

[4] J. E. Walker and J. Howland, "Falls and fear of falling among elderly persons living in the community: occupational therapy interventions," *American Journal of Occupational Therapy*, vol. 45, no. 2, pp. 119–122, February 1991.

[5] L. J. Baraff, R. Della Penna, N. Williams, and A. Sanders, "Practice guideline for the ED management of falls in community-dwelling elderly persons," *Annals of Emergency Medicine*, vol. 30, no. 4, pp. 480–492, October 1997.

[6] N. Pannurat, S. Thiemjarus, and E. Nantajeewarawat, "Automatic fall monitoring: A review," *Sensors*, vol. 14, no. 7, pp. 12900–12936, July 2014.

[7] N. El-Bendary, Q. Tan, F. C. Pivot, and A. Lam, "Fall detection and prevention for the elderly: A review of trends and challenges," *International Journal of Smart Sensing Intelligent Systems*, vol. 6, no. 3, pp. 1230–1266, June 2013.

[8] SENSE4CARE, "Angel4," http://www.sense4care.com/en/products/angel4-whiteblack, Accessed on: 20 August 2017.

[9] Philips, "Automatic fall detection," https://www.lifeline.philips.com/automatic-fall-detection.html, Accessed on: 20 August 2017.

[10] Medical Guardian, "Medical alert systems," https://www.medicalguardian.com/help-me-choose/?aid=243&utm_source=TopTenReviews&utm_medium=referral, Accessed on: 20 August 2017.

[11] TopTen Reviews, "Best fall detection sensors," http://www.toptenreviews.com/health/senior-care/best-fall-detection-sensors/, Accessed on: 20 August 2017.

[12] J. R. Barfield and S. C. Welch, "Adaptive classification of fall detection for personal emergency response systems," U.S. Patent 0 061 863, 5 March 2015.

[13] M. L. Lehrman, M. D. Halleck, and E. L. Massman, "System and method for detecting motion of a body," U.S. Patent 6 703 939, 9 March 2004.

[14] T. Ganyi, "Fall detection methods and devices," U.S. Patent 9 427 177, 30 August 2016.

[15] G. Yi and D. Wan, "Fall detection method and system," U.S. Patent 9 153 114, 6 October 2015.

[16] F. Hijaz, N. Afzal, T. Ahmad, and O. Hasan, "Survey of fall detection and daily activity monitoring techniques," in *IEEE International Conference on Information and Emerging Technologies (ICIET)*, (Karachi, Pakistan), pp. 1–6, 14–16 June 2010.

[17] M. Mubashir, L. Shao, and L. Seed, "A survey on fall detection: Principles and approaches," *Neurocomputing*, vol. 100, pp. 144–152, January 2013.

[18] Y. S. Delahoz and M. A. Labrador, "Survey on fall detection and fall prevention using wearable and external sensors," *Sensors*, vol. 14, no. 10, pp. 19806–19842, October 2014.

[19] S. Chaudhuri, H. Thompson, and G. Demiris, "Fall detection devices and their use with older adults: a systematic review," *Journal of Geriatric Physical Therapy*, vol. 37, no. 4, p. 178, October 2014.

[20] R. Igual, C. Medrano, and I. Plaza, "Challenges, issues and trends in fall detection systems," *Biomedical Engineering Online*, vol. 12, no. 1, p. 66, July 2013.

[21] A. Yazar, F. Keskin, B. U. Töreyin, and A. E. Çetin, "Fall detection using single-tree complex wavelet transform," *Pattern Recognition Letters*, vol. 34, no. 15, pp. 1945–1952, November 2013.

[22] B. U. Töreyin, Y. Dedeoğlu, and A. E. Çetin, "HMM based falling person detection using both audio and video," in *International Workshop on Human-Computer Interaction*, (Las Vegas, NV, U.S.A.), pp. 211–220, 22–27 July 2005.

[23] L. Schwickert, C. Becker, U. Lindemann, C. Maréchal, A. Bourke, L. Chiari, J. Helbostad, W. Zijlstra, K. Aminian, C. Todd, S. Bandinelli, and J. Klenk, "Fall detection with body-worn sensors," *Zeitschrift für Gerontologie und Geriatrie*, vol. 46, no. 8, pp. 706–719, December 2013.

[24] S. Chamanian, S. Baghaee, H. Ulusan, Ö. Zorlu, H. Külah, and E. Uysal-Bıyıkoğlu, "Powering-up wireless sensor nodes utilizing rechargeable batteries and an electromagnetic vibration energy harvesting system," *Energies*, vol. 7, no. 10, pp. 6323–6339, October 2014.

[25] W. Y. Toh, Y. K. Tan, W. S. Koh, and L. Siek, "Autonomous wearable sensor nodes with flexible energy harvesting," *IEEE Sensors Journal*, vol. 14, no. 7, pp. 2299–2306, July 2014.

[26] N. Kern, B. Schiele, and A. Schmidt, "Multi-sensor activity context detection for wearable computing," in *1st European Symposium on Ambient Intelligence (EUSAI)*, (Veldhoven, The Netherlands), pp. 220–232, 2–3 November 2003.

[27] M. A. Habib, M. S. Mohktar, S. B. Kamaruzzaman, K. S. Lim, T. M. Pin, and F. Ibrahim, "Smartphone-based solutions for fall detection and prevention: Challenges and open issues," *Sensors*, vol. 14, no. 4, pp. 7181–7208, April 2014.

[28] A. T. Özdemir and B. Barshan, "Detecting falls with wearable sensors using machine learning techniques," *Sensors*, vol. 14, no. 6, pp. 10691–10708, June 2014.

[29] O. Aziz, M. Musngi, E. J. Park, G. Mori, and S. N. Robinovitch, "A comparison of accuracy of fall detection algorithms (threshold-based vs. machine learning) using waist-mounted tri-axial accelerometer signals from a comprehensive set of falls and non-fall trials," *Medical & Biological Engineering & Computing*, vol. 55, no. 1, pp. 45–55, January 2017.

[30] A. O. Kansız, M. A. Güvensan, and H. I. Türkmen, "Selection of time-domain features for fall detection based on supervised learning," in *Proceedings of the World Congress on Engineering and Computer Science*, (San Francisco, CA, U.S.A.), 23–25 October 2013.

[31] O. Ojetola, E. I. Gaura, and J. Brusey, "Fall detection with wearable sensors–SAFE (SmArt Fall dEtection)," in *IEEE 7th International Conference on Intelligent Environments (IE)*, (Nottingham, U.K.), pp. 318–321, 25–28 July 2011.

[32] S. Shan and T. Yuan, "A wearable pre-impact fall detector using feature selection and support vector machine," in *IEEE 10th International Conference on Signal Processing (ICSP)*, (Beijing, China), pp. 1686–1689, 24–28 October 2010.

[33] X. Yang, A. Dinh, and L. Chen, "A wearable real-time fall detector based on naïve Bayes classifier," in *IEEE 23rd Canadian Conference on Electrical and Computer Engineering (CCECE)*, (Calgary, Canada), pp. 1–4, 02–05 May 2010.

[34] B. Aguiar, T. Rocha, J. Silva, and I. Sousa, "Accelerometer-based fall detection for smartphones," in *IEEE International Symposium on Medical Measurements and Applications (MeMeA)*, (Lisbon, Portugal), pp. 1–6, 11–12 June 2014.

[35] W.-C. Cheng and D.-M. Jhan, "Triaxial accelerometer-based fall detection method using a self-constructing cascade-AdaBoost-SVM classifier," *IEEE Journal of Biomedical and Health Informatics*, vol. 17, no. 2, pp. 411–419, March 2013.

[36] M. Yuwono, B. D. Moulton, S. W. Su, B. G. Celler, and H. T. Nguyen, "Unsupervised machine-learning method for improving the performance of ambulatory fall-detection systems," *Biomedical Engineering Online*, vol. 11, no. 1, p. 9, February 2012.

[37] T. Zhang, J. Wang, P. Liu, and J. Hou, "Fall detection by embedding an accelerometer in cellphone and using KFD algorithm," *International Journal of Computer Science and Network Security*, vol. 6, no. 10, pp. 277–284, October 2006.

[38] G. Rescio, A. Leone, and P. Siciliano, "Supervised expert system for wearable MEMS accelerometer-based fall detector," *Journal of Sensors*, vol. 2013, pp. 1–11, 2013.

[39] L. Tong, Q. Song, Y. Ge, and M. Liu, "HMM-based human fall detection and prediction method using tri-axial accelerometer," *IEEE Sensors Journal*, vol. 13, no. 5, pp. 1849–1856, May 2013.

[40] N. Noury, A. Fleury, P. Rumeau, A. K. Bourke, G. ÓLaighin, V. Rialle, and J. E. Lundy, "Fall detection—principles and methods," in *29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, (Lyon, France), pp. 1663–1666, 23–26 August 2007.

[41] A. T. Özdemir, "An analysis on sensor locations of the human body for wearable fall detection devices: Principles and practice," *Sensors*, vol. 16, no. 8, pp. 1161–1185, July 2016.

[42] P. Ntanasis, E. Pippa, A. T. Özdemir, B. Barshan, V. Megalooikonomou, "Investigation of sensor placement for accurate fall detection," *Proceedings of the 6th EAI International Conference on Wireless Mobile Communication and Healthcare (MobiHealth)*, 14–16 November 2016, Milan, Italy, P. Perego, G. Andreoni, and G. Rizzo (eds.), *Lecture Notes of the Institute for Computer Sciences, Social Informatics, and Telecommunications Engineering (LNICST)*, vol.192, pp.225–232, Springer, June 2017. DOI: 10.1007/978-3-319-58877-3_30

[43] J. Dai, X. Bai, Z. Yang, Z. Shen, and D. Xuan, "PerFallD: A pervasive fall detection system using mobile phones," in *8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, (Mannheim, Germany), pp. 292–297, 29 March–2 April 2010.

[44] S.-H. Fang, Y.-C. Liang, and K.-M. Chiu, "Developing a mobile phone-based fall detection system on Android platform," in *IEEE Computing, Communications and Applications Conference (ComComAp)*, (Hong Kong, China), pp. 143–146, 11–13 January 2012.

[45] A. K. Bourke, J. V. O'Brien, and G. M. Lyons, "Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm," *Gait & Posture*, vol. 26, no. 2, pp. 194–199, July 2007.

[46] U. Lindemann, A. Hock, M. Stuber, W. Keck, and C. Becker, "Evaluation of a fall detector based on accelerometers: A pilot study," *Medical and Biological Engineering and Computing*, vol. 43, no. 5, pp. 548–551, October 2005.

[47] F. Bianchi, S. J. Redmond, M. R. Narayanan, S. Cerutti, and N. H. Lovell, "Barometric pressure and triaxial accelerometry-based falls event detection," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 18, no. 6, pp. 619–627, December 2010.

[48] S. Abbate, M. Avvenuti, G. Cola, P. Corsini, J. Light, and A. Vecchio, "Recognition of false alarms in fall detection systems," in *IEEE Consumer Communications and Networking Conference (CCNC)*, (Las Vegas, NV, U.S.A.), pp. 23–28, 9–12 January 2011.

[49] G. Anania, A. Tognetti, N. Carbonaro, M. Tesconi, F. Cutolo, G. Zupone, and D. De Rossi, “Development of a novel algorithm for human fall detection using wearable sensors,” in *7th IEEE Conference on Sensors*, (Lecce, Italy), pp. 1336–1339, 26–29 October 2008.

[50] W.-S. Baek, D.-M. Kim, F. Bashir, and J.-Y. Pyun, “Real life applicable fall detection system based on wireless body area network,” in *IEEE Consumer Communications and Networking Conference (CCNC)*, (Las Vegas, NV, U.S.A.), pp. 62–67, 11–14 January 2013.

[51] P. Boissy, S. Choquette, M. Hamel, and N. Noury, “User-based motion sensing and fuzzy logic for automated fall detection in older adults,” *Telemedicine and e-Health*, vol. 13, no. 6, pp. 683–694, January 2007.

[52] A. K. Bourke, K. J. O’Donovan, J. Nelson, and G. ÓLaighin, “Fall-detection through vertical velocity thresholding using a tri-axial accelerometer characterized using an optical motion-capture system,” in *30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, (Vancouver, Canada), pp. 2832–2835, 20–25 August 2008.

[53] P.-K. Chao, H.-L. Chan, F.-T. Tang, Y.-C. Chen, and M.-K. Wong, “A comparison of automatic fall detection by the cross-product and magnitude of tri-axial acceleration,” *Physiological Measurement*, vol. 30, no. 10, pp. 1027–1037, August 2009.

[54] J. Chen, K. Kwong, D. Chang, J. Luk, and R. Bajcsy, “Wearable sensors for reliable fall detection,” in *27th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, (Shanghai, China), pp. 3551–3554, 1–4 September 2005.

[55] D. Chen, W. Feng, Y. Zhang, X. Li, and T. Wang, “A wearable wireless fall detection system with accelerators,” in *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, (Phuket, Thailand), pp. 2259–2263, 7–11 December 2011.

[56] J. Jacob, T. Nguyen, D. Y. C. Lie, S. Zupancic, J. Bishara, A. Dentino, and R. E. Banister, “A fall detection study on the sensors placement location

and a rule-based multi-thresholds algorithm using both accelerometer and gyroscopes," in *IEEE International Conference on Fuzzy Systems (FUZZ)*, (Taipei, Taiwan), pp. 666–671, 27–30 June 2011.

[57] P. Jantaraprim, P. Phukpattaranont, C. Limsakul, and B. Wongkittisuksa, "Improving the accuracy of a fall detection algorithm using free fall characteristics," in *IEEE International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, (Chiang Mai, Thailand), pp. 501–504, 19–21 May 2010.

[58] M. Kangas, A. Konttila, I. Winblad, and T. Jämsä, "Determination of simple thresholds for accelerometry-based parameters for fall detection," in *29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, (Lyon, France), pp. 1367–1370, 23–26 August 2007.

[59] M. Kangas, A. Konttila, P. Lindgren, I. Winblad, and T. Jämsä, "Comparison of low-complexity fall detection algorithms for body attached accelerometers," *Gait & Posture*, vol. 28, no. 2, pp. 285–291, August 2008.

[60] 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, June 2009.

[61] G. A. Koshmak, M. Linden, and A. Loutfi, "Evaluation of the Android-based fall detection system with physiological data monitoring," in *35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, (Osaka, Japan), pp. 1164–1168, 3–7 July 2013.

[62] R. Y. W. Lee and A. J. Carlisle, "Detection of falls using accelerometers and mobile phone technology," *Age and Ageing*, vol. 40, no. 6, pp. 690–696, November 2011.

[63] Q. Li, J. A. Stankovic, M. A. Hanson, A. T. Barth, J. Lach, and G. Zhou, "Accurate, fast fall detection using gyroscopes and accelerometer-derived posture information," in *IEEE 6th International Workshop on Wearable and Implantable Body Sensor Networks (BSN)*, (Berkeley, CA, U.S.A.), pp. 138–143, 3–5 June 2009.

[64] P. Pierleoni, A. Belli, L. Palma, M. Pellegrini, L. Pernini, and S. Valenti, "A high reliability wearable device for elderly fall detection," *IEEE Sensors Journal*, vol. 15, no. 8, pp. 4544–4553, August 2015.

[65] A. Sorvala, E. Alasaarela, H. Sorvoja, and R. Myllylä, "A two-threshold fall detection algorithm for reducing false alarms," in *IEEE 6th International Symposium on Medical Information and Communication Technology (ISMICT)*, (La Jolla, CA, U.S.A.), pp. 1–4, 25–29 March 2012.

[66] M. Tolkiehn, L. Atallah, B. Lo, and G.-Z. Yang, "Direction sensitive fall detection using a triaxial accelerometer and a barometric pressure sensor," in *33rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, (Boston, MA, U.S.A.), pp. 369–372, 30 August–3 September 2011.

[67] C.-C. Wang, C.-Y. Chiang, P.-Y. Lin, Y.-C. Chou, I.-T. Kuo, C.-N. Huang, and C.-T. Chan, "Development of a fall detecting system for the elderly residents," in *IEEE 2nd International Conference on Bioinformatics and Biomedical Engineering (ICBBE)*, (Shanghai, China), pp. 1359–1362, 16–18 March 2008.

[68] G. Wu and S. Xue, "Portable preimpact fall detector with inertial sensors," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 16, no. 2, pp. 178–183, April 2008.

[69] F. Wu, H. Zhao, Y. Zhao, and H. Zhong, "Development of a wearable-sensor-based fall detection system," *International Journal of Telemedicine and Applications*, vol. 2015, pp. 2–12, January 2015.

[70] F. Sposaro and G. Tyson, "iFall: An Android application for fall monitoring and response," in *31st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, (Minneapolis, MN, U.S.A.), pp. 6119–6122, 2–6 September 2009.

[71] G. R. Yavuz, M. E. Koçak, G. Ergün, H. Alemdar, H. Yalçın, O. D. İncel, L. Akarun, and C. Ersoy, "A smartphone based fall detector with online

location support," in *PhoneSense: International Workshop on Sensing for App Phones*, (Zürich, Switzerland), pp. 31–35, 2 November 2010.

[72] S.-Y. Chang, C.-F. Lai, H.-C. J. Chao, J. H. Park, and Y.-M. Huang, "An environmental-adaptive fall detection system on mobile device," *Journal of Medical Systems*, vol. 35, no. 5, pp. 1299–1312, October 2011.

[73] A. Yurtman and B. Barshan, "Activity recognition invariant to sensor orientation with wearable motion sensors," *Sensors*, vol. 17, no. 8, article no: 1838, August 2017.

[74] O. Banos, M. A. Toth, M. Damas, H. Pomares, and I. Rojas, "Dealing with the effects of sensor displacement in wearable activity recognition," *Sensors*, vol. 14, no. 6, pp. 9995–10023, June 2014.

[75] N. Noury, P. Rumeau, A. K. Bourke, G. ÓLaighin, and J. E. Lundy, "A proposal for the classification and evaluation of fall detectors," *IRBM*, vol. 29, no. 6, pp. 340–349, December 2008.

[76] S. Abbate, M. Avvenuti, P. Corsini, J. Light, and A. Vecchio, "Monitoring of human movements for fall detection and activities recognition in elderly care using wireless sensor network: A survey," *Wireless Sensor Networks: Application-Centric Design*. Rijeka, Croatia: InTech, 2010.

[77] H. Rimminen, J. Lindström, M. Linnavuo, and R. Sepponen, "Detection of falls among the elderly by a floor sensor using the electric near field," *IEEE Transactions on Information Technology in Biomedicine*, vol. 14, no. 6, pp. 1475–1476, November 2010.

[78] O. Ojetola, E. Gaura, and J. Brusey, "Data set for fall events and daily activities from inertial sensors," in *Proceedings of the 6th ACM Multimedia Systems Conference*, (Portland, OR, U.S.A.), pp. 243–248, 18–20 March 2015.

[79] F. Bagalà, C. Becker, A. Cappello, L. Chiari, K. Aminian, J. M. Hausdorff, W. Zijlstra, and J. Klenk, "Evaluation of accelerometer-based fall detection algorithms on real-world falls," *PloS One*, vol. 7, no. 5, pp. 37062–37070, May 2012.

[80] M. Kangas, I. Vikman, L. Nyberg, R. Korpelainen, J. Lindblom, and T. Jämsä, "Comparison of real-life accidental falls in older people with experimental falls in middle-aged test subjects," *Gait & Posture*, vol. 35, no. 3, pp. 500–505, March 2012.

[81] A. K. Bourke, P. Van de Ven, M. Gamble, R. O'Connor, K. Murphy, E. Bogan, E. McQuade, P. Finucane, G. ÓLaighin, and J. Nelson, "Evaluation of waist-mounted tri-axial accelerometer based fall-detection algorithms during scripted and continuous unscripted activities," *Journal of Biomechanics*, vol. 43, no. 15, pp. 3051–3057, November 2010.

[82] L. A. Zadeh, *Fuzzy sets, fuzzy logic, and fuzzy systems: Selected papers by Lotfi A. Zadeh*, vol. 6. Singapore: World Scientific, 1996.

[83] A. K. Bourke and G. M. Lyons, "A threshold-based fall-detection algorithm using a bi-axial gyroscope sensor," *Medical Engineering & Physics*, vol. 30, no. 1, pp. 84–90, January 2008.

[84] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. New York, NY, U.S.A.: John Wiley & Sons, 2000.

[85] E. Alpaydın, *Introduction to Machine Learning*. Cambridge, MA, U.S.A.: MIT Press, 2nd ed., 2010.

[86] S. Haykin, *Neural Networks: A Comprehensive Foundation*. New Jersey, NJ, U.S.A.: Prentice Hall, 2nd ed., 1998.

[87] E. Principi, D. Droghini, S. Squartini, P. Olivetti, and F. Piazza, "Acoustic cues from the floor: A new approach for fall classification," *Expert Systems with Applications*, vol. 60, pp. 51–61, October 2016.

[88] M. V. Albert, K. Kording, M. Herrmann, and A. Jayaraman, "Fall classification by machine learning using mobile phones," *PloS One*, vol. 7, no. 5, pp. e36556–e36561, May 2012.

[89] Y. Choi, A. S. Ralhan, and S. Ko, "A study on machine learning algorithms for fall detection and movement classification," in *IEEE International*

Conference on Information Science and Applications (ICISA), (Jeju Island, South Korea), pp. 1–8, 26–29 April 2011.

[90] C. Dinh and M. Struck, "A new real-time fall detection approach using fuzzy logic and a neural network," in *IEEE 6th International Workshop on Wearable Micro and Nano Technologies for Personalized Health (pHealth)*, (Oslo, Norway), pp. 57–60, 24–26 June 2009.

[91] Y. Tao, H. Qian, M. Chen, X. Shi, and Y. Xu, "A real-time intelligent shoe system for fall detection," in *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, (Phuket, Thailand), pp. 2253–2258, 7–11 December 2011.

Appendix A

Code

A.1 MATLAB Code of the Proposed Heuristic Fall-Detection Algorithm (FADoTh)

```
load('data.mat');
CONF=zeros(2,2);    % Create Confusion Matrix
for sub=1:size(data,1)
 for act=1:size(data,2)
  for ts=1:size(data,3)
   datamed=medfilt1(data{sub,act,ts},3);
   % 3-pt median filter
   SV_tot=sqrt(sum(datamed(:,10:12).^2,2));
   % Total acceleration
   SV_tot_trimmed=SV_tot{sub,act,ts}(10:end-9);
   % Trimmed total acceleration
   angvel=sqrt(sum(datamed(:,13:15).^2,2));
   % Total angular velocity
   angvel_trimmed=angvel(10:end-9);
   % Trimmed total angular velocity
   maxsvtot=max(abs(SV_tot_trimmed));
```

```
% First feature: maximum of total acceleration
mult=abs(SV_tot_trimmed.*angvel_trimmed);
% Multiplication signal
maxmult=max(mult);
% Second feature: maximum of multiplication signal
  if maxsvtot<MAXSVTOT_lower
  % Extreme conditions for first feature
  label=0;
  elseif maxsvtot>MAXSVTOT_upper
  % Extreme conditions for first feature
  label=1;
  else
  first_nonfall_membership=abs(maxsvtot-MAXSVTOT_upper)/
      (MAXSVTOT_upper-MAXSVTOT_lower);
  % First membership values
  first_fall_membership=abs(maxsvtot-MAXSVTOT_lower)/
      (MAXSVTOT_upper-MAXSVTOT_lower);
    if maxmult<MAXMULT_lower
    label=0;
    elseif maxmult>MAXMULT_upper
    label=1;
    else
    second_nonfall_membership=abs(maxmult-MAXMULT_upper)/
        (MAXMULT_upper-MAXMULT_lower);
    % Second membership values
    second_fall_membership=1-abs(maxmult-MAXMULT_lower)/
        (MAXMULT_upper-MAXMULT_lower);

    nonfall_membership=(first_nonfall_membership+second_nonfall_membership)/2;
    % Total membership values
    fall_membership=(first_fall_membership+second_fall_membership)/2;
```

```
        label=double(fall_membership>nonfall_membership);
       % Labels for non-extreme data
        end
      end
    g=1+double(act>16);
    h=label+1;
    % label: 0-> non-fall , 1-> fall
    CONF(g,h)=CONF(g,h)+1;
    end
  end
end
%%% PERFORMANCE METRICS %%%
CONF
acc=CONF(1,1)/sum(CONF(1,:))/2+CONF(2,2)/sum(CONF(2,:))/2
precision=CONF(2,2)/sum(CONF(:,2))
sensitivity=CONF(2,2)/sum(CONF(2,:))
specificity=CONF(1,1)/sum(CONF(1,:))
Fmeasure=2*precision*sensitivity/(precision+sensitivity)
```

A.2 MATLAB Code of the Proposed Machine Learning Based Fall-Detection Algorithm (kAT)

```
load('data.mat');
labels=zeros(size(data,1)*size(data,2)*size(data,3),1);
features=zeros(size(data,1)*size(data,2)*size(data,3),6);
%[maxsvtot,minsvtot,maxangvel,angvelenergy,maxabslength,
    maxmult]
subject=zeros(size(data,1)*size(data,2)*size(data,3),1);
f_lbl=zeros(size(data,1)*size(data,2)*size(data,3),7);
% All features and labels combined for testing
%%% PREPROCESSING & FEATURE EXTRACTION %%%
for sub=1:size(data,1)
 for act=1:size(data,2)
  for ts=1:size(data,3)
   datamed=medfilt1(data{sub,act,ts},3);

   SV_tot=sqrt(sum(datamed(:,10:12).^2,2));
   SV_tot_shaved=SV_tot(10:end-9);
   indmax=find(SV_tot==max(SV_tot));
   angvel=sqrt(sum(datamed(:,13:14).^2,2));
   angvel_shaved=angvel(10:end-9);
   angvel_int=angvel(indmax-2:indmax+2);

   angvel_energy=angvel_int'*angvel_int;
   maxangvel=max(abs(angvel_shaved));
   maxsvtot=max(abs(SV_tot_shaved));
   minsvtot=min(abs(SV_tot_shaved));

   abs_temp=abs([0;SV_tot_shaved]-[SV_tot_shaved;0]);
   abs_length=abs_temp(2:end-1);
```

```
    cum_abs_length=[abs_length' 0 0]+[0 abs_length' 0]+[0 0 abs_length'];
    maxabslength=max(cum_abs_length);

    mult=abs(SV_tot_shaved.*angvel_shaved);
    maxmult=max(mult);

    features((act-1)*size(data,1)*size(data,3)+...
            (sub-1)*size(data,3)+ts,:)=[maxsvtot minsvtot maxangvel angvel_energy maxabslength maxmult];
    labels((act-1)*size(data,1)*size(data,3)+...
            (sub-1)*size(data,3)+ts)=double(act>16);
   % 0-> non-fall 1-> fall
    subject((act-1)*size(data,1)*size(data,3)+...
            (sub-1)*size(data,3)+ts)=sub;
    end
  end
end
perm=randperm(length(labels));
% Randomize the order of the data

features=features(perm,:);
labels=labels(perm);
subject=subject(perm);
%%% Data Separation for Subject-Based Cross Validation %%%
tr_set=cell(8,1);
tst_set=cell(8,1);
fall=cell(8,1);
nonfall=cell(8,1);
tr_set_lbl=cell(8,1);
tst_set_lbl=cell(8,1);
tr_set_sub=cell(8,1);
tst_set_sub=cell(8,1);
```

```
meann=cell(8,1);
stdd=cell(8,1);
tr_sub=cell(8,7);
tst_sub=cell(8,7);
fall_sub=cell(8,7);
nonfall_sub=cell(8,7);
tr_sub_lbl=cell(8,7);
tst_sub_lbl=cell(8,7);

sbjcts={[1 10] [2 11] [3 12] [4 13] [5 14] [6 15] [7 16] [8 9]};
% Subjects for the test data
sbsbjcts={[2 11] [3 12] [4 13] [5 14] [6 15] [7 16] [8 9];...
    [1 10] [3 12] [4 13] [5 14] [6 15] [7 16] [8 9];...
    [1 10] [2 11] [4 13] [5 14] [6 15] [7 16] [8 9];...
    [1 10] [2 11] [3 12] [5 14] [6 15] [7 16] [8 9];...
    [1 10] [2 11] [3 12] [4 13] [6 15] [7 16] [8 9];...
    [1 10] [2 11] [3 12] [4 13] [5 14] [7 16] [8 9];...
    [1 10] [2 11] [3 12] [4 13] [5 14] [6 15] [8 9];...
    [1 10] [2 11] [3 12] [4 13] [5 14] [6 15] [7 16]};
% Subjects for the sub-test data
for indx=1:8
tr_set{indx}=features(~ismember(subject,sbjcts{indx}),:);
meann{indx}=mean(tr_set{indx},1);
stdd{indx}=std(tr_set{indx},0,1);
% Find the mean and standard deviation values
tr_set{indx}=(tr_set{indx}-repmat(meann{indx},size(tr_set{indx},1),1))./repmat(stdd{indx},size(tr_set{indx},1),1)
% Normalize training data
tst_set{indx}=features(ismember(subject,sbjcts{indx}),:);
```

```
tst_set{indx}=(tst_set{indx}-repmat(meann{indx},size(tst_set{indx},1),1))./repmat(stdd{indx},size(tst_set{indx},1),1);
% Normalize test data
tr_set_lbl{indx}=labels(~ismember(subject,sbjcts{indx}));
tst_set_lbl{indx}=labels(ismember(subject,sbjcts{indx}));
% Assign labels
tr_set_sub{indx}=subject(~ismember(subject,sbjcts{indx}));
tst_set_sub{indx}=subject(ismember(subject,sbjcts{indx}));
% Subjects
fall{indx}=tr_set{indx}(tr_set_lbl{indx}==1,:);
nonfall{indx}=tr_set{indx}(tr_set_lbl{indx}==0,:);
 for sbindx=1:7
 tr_sub{indx,sbindx}=tr_set{indx}(~ismember(tr_set_sub{indx},sbsbjcts{indx,sbindx}),:);
 tst_sub{indx,sbindx}=tr_set{indx}(ismember(tr_set_sub{indx},sbsbjcts{indx,sbindx}),:);
 % Training and test sub-folds
 tr_sub_lbl{indx,sbindx}=tr_set_lbl{indx}(~ismember(tr_set_sub{indx},sbsbjcts{indx,sbindx}));
 tst_sub_lbl{indx,sbindx}=tr_set_lbl{indx}(ismember(tr_set_sub{indx},sbsbjcts{indx,sbindx}));
 % Labels of the training and test sub-folds
 fall_sub{indx,sbindx}=tr_sub{indx,sbindx}(tr_sub_lbl{indx,sbindx}==1,:);
 nonfall_sub{indx,sbindx}=tr_sub{indx,sbindx}(tr_sub_lbl{indx,sbindx}==0,:);
 % Subjects of the training and test sub-folds
 end
end
%%% PARAMETER OPTIMIZATION %%%
acc_thr=zeros(8,7);
CONF_best=cell(8,7);
```

```
acc_act=zeros(8,1);
specificity_act=zeros(8,1);
sensitivity_act=zeros(8,1);
precision_act=zeros(8,1);
Fmeasure_act=zeros(8,1);
CONF_act=cell(8,1);
K_best=zeros(8,1);
offset_best=zeros(8,1);
classifier_act=cell(8,1);
for indx=1:8
% Outer Loop
 for offset=linspace(-0.3,0.3,37)
 % Interval of the OFFSET Parameter - Intermediate Loop
  for K=1:51
  % Interval of the K parameter of the learner
  acc=zeros(1,7);
  precision=zeros(1,7);
  sensitivity=zeros(1,7);
  specificity=zeros(1,7);
  Fmeasure=zeros(1,7);
  for sbindx=1:7
  % Inner Loop
  classifier=iteration(fall_sub{indx,sbindx}, nonfall_sub{
     indx,sbindx}, tr_sub{indx,sbindx}, [tr_sub{indx,
     sbindx} tr_sub_lbl{indx,sbindx}], 1, size(tr_sub{indx
     ,sbindx},1), [], offset);
  [CONF, acc(sbindx), precision(sbindx), sensitivity(
     sbindx), specificity(sbindx), Fmeasure(sbindx)]=test(
     classifier, tst_sub{indx,sbindx}, tst_sub_lbl{indx,
     sbindx}, [tr_sub{indx,sbindx} tr_sub_lbl{indx,sbindx
     }], K);
  end
  if mean(acc)>mean(acc_thr(indx,:))
```

```
    acc_thr(indx,:)=acc;
    K_best(indx)=K;
    offset_best(indx)=offset;
    end
    end
  end
  classifier_act{indx}=iteration(fall{indx}, nonfall{indx}, tr_set{indx}, [tr_set{indx} tr_set_lbl{indx}], 1, size(tr_set{indx},1), cutoff, [], offset_best(indx));
  [CONF_act{indx}, acc_act(indx), precision_act(indx), sensitivity_act(indx), specificity_act(indx), Fmeasure_act(indx)]=test(classifier_act{indx}, tst_set{indx}, tst_set_lbl{indx}, [tr_set{indx} tr_set_lbl{indx}], K_best(indx));
end
%%% TRAINING FUNCTION %%%
function  [classifier] = iteration(fall, nonfall, features, f_lbl, num_iter, initial_size, featuresused, offset)
if size(fall,1)/initial_size >0.1 && size(nonfall,1)/initial_size >0.1
% First Terminating Condition
 threshold=inf;
 % Initial upper threshold of portion of excluded extreme instances
 for ftr=1:size(features,2)
 % Sweep through all features
 fall_max=unique(max(fall(:,ftr)));
 fall_min=unique(min(fall(:,ftr)));
 nonfall_max=unique(max(nonfall(:,ftr)));
 nonfall_min=unique(min(nonfall(:,ftr)));

 maxes=[nonfall_max fall_max];
 mins=[nonfall_min fall_min];
```

```
[maxx, maxind]=min(maxes);
% Find thresholds w/ activity types -> maxind=1 means nonfall maxind=2 means fall
[minn, minind]=max(mins);
range=max(maxes)-min(mins);
thr_max=maxx+range*offset;
thr_min=minn-range*offset;

nonextr_fall=fall(fall(:,ftr)>thr_min & fall(:,ftr)<thr_max,ftr);
nonextr_nonfall=nonfall(nonfall(:,ftr)>thr_min & nonfall(:,ftr)<thr_max,ftr);
uncut_count=(size(nonextr_fall,1)+size(nonextr_nonfall,1));
perc_nonextremes=double(uncut_count/size(features,1));

 if perc_nonextremes<threshold
 classifier.ftr=ftr;
 classifier.falldata=fall(fall(:,ftr)>thr_min & fall(:,ftr)<thr_max,ftr);
 classifier.nonfalldata=nonfall(nonfall(:,ftr)>thr_min & nonfall(:,ftr)<thr_max,ftr);
 classifier.r_thr=thr_max;
 classifier.r_label=mod(maxind,2);
% 0-> non-fall 1-> fall
 classifier.l_thr=thr_min;
 classifier.l_label=mod(minind,2);
 threshold=perc_nonextremes;
 classifier.threshold=threshold;
 classifier.interval_ind=f_lbl(:,ftr)>thr_min & f_lbl(:,ftr)<thr_max;
 end
```

```
 end
features(features(:,classifier.ftr)>classifier.r_thr,:)=[];
features(features(:,classifier.ftr)<classifier.l_thr,:)=[];
fall(fall(:,classifier.ftr)>classifier.r_thr,:)=[];
fall(fall(:,classifier.ftr)<classifier.l_thr,:)=[];
nonfall(nonfall(:,classifier.ftr)>classifier.r_thr,:)=[];
nonfall(nonfall(:,classifier.ftr)<classifier.l_thr,:)=[];
if num_iter<=6      % Second Terminating Condition
    classifier.next=iteration(fall, nonfall, features, f_lbl, num_iter+1, initial_size, featuresused, offset);
else
    classifier.next=[];
    classifier.next.ftr=0;
end
else
    classifier=[];
    classifier.ftr=0;
end
end
%%% TEST FUNCTION %%%
function [CONF, acc, precision, sensitivity, specificity, Fmeasure] = test(classifier, test_data, test_lbl, f_lbl, K)
dummy=classifier;
mid=ones(size(f_lbl,1),1);
 while dummy.ftr~=0
 % Obtaining the ultimate non-extreme training data
 mid=mid & dummy.interval_ind;
 dummy=dummy.next;
 end
```

```
CONF=zeros(2,2);
falsealarm=0;
 for inddex=1:size(test_data,1)
 feature=test_data(inddex,:);
 label=test_lbl(inddex);

 dummy=classifier;
 features_used=[];
  while dummy.ftr~=0
   if feature(dummy.ftr)>=dummy.r_thr
   % Assign the classes of the extreme data
   CONF(label, dummy.r_label+1)=CONF(label, dummy.r_label+1)+1;
   break;
   elseif feature(dummy.ftr)<=dummy.l_thr
   % Assign the classes of the extreme data
   CONF(label, dummy.l_label+1)=CONF(label, dummy.l_label+1)+1;
   break;
   else
   % Process non-extreme data
   features_used(end+1)=dummy.ftr;
    if dummy.next.ftr==0
    % Modified k-NN procedure
    distance=sqrt(sum((f_lbl(:,unique(features_used))-repmat(feature(unique(features_used)),size(f_lbl,1),1)).^2,2));
    [bb inn]=sort(distance);
    all_nn=f_lbl(inn,:);
    all_nn_mid=mid(inn);
    all_mid=all_nn(all_nn_mid,:);
    nn=all_mid(1:min([nn size(all_mid,1)]),:);

```

```
        output=double(sum(double(nn(:,end)==1))>sum(double(nn(:,end)==0)));
       CONF(label, output+1)=CONF(label, output+1)+1;
        end
     dummy=dummy.next;
      end
  end
end
%%% PERFORMANCE METRICS %%%
acc=(CONF(1,2)+CONF(2,3))/sum(sum(CONF));
precision=CONF(2,3)/sum(CONF(:,3));
sensitivity=CONF(2,3)/sum(CONF(2,:));
specificity=CONF(1,2)/sum(CONF(1,:));
Fmeasure=2*precision*sensitivity/(precision+sensitivity);
end
```