Pervasive and Mobile Computing 58 (2019) 101026

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

Beyond position-awareness—Extending a self-adaptive fall detection system☆

Christian Krupitzer [a,*], Timo Sztyler [c], Janick Edinger [b], Martin Breitbach [b], Heiner Stuckenschmidt [c], Christian Becker [b]

[a] Software Engineering Group, University of Würzburg, Würzburg, Germany
[b] Chair of Information Systems II, University of Mannheim, Mannheim, Germany
[c] Data and Web Science Group, University of Mannheim, Mannheim, Germany

ARTICLE INFO

Article history:
Available online 29 May 2019

Keywords:
Pervasive computing
Ambient assisted living
Fall detection

ABSTRACT

Ambient Assisted Living using mobile device sensors is an active area of research in pervasive computing. Multiple approaches have shown that wearable sensors perform very well and distinguish falls reliably from *Activities of Daily Living*. However, these systems are tested in a controlled environment and are optimized for a given set of sensor types, sensor positions, and subjects. We propose a self-adaptive pervasive fall detection approach that is robust to the heterogeneity of real life situations. Using the data of four publicly available datasets, we show that our system is not only robust regarding the different dimensions of heterogeneity, but also adapts autonomously to spontaneous changes in the sensor's position at runtime. In this paper, we extend our self-adaptive fall detection system with (i) additional algorithms for fall detection, (ii) an approach for cross-positional sensor fusion, (iii) a fall detection approach that relies on outlier detection, and (iv) a smart fall alert. Additionally, we present implementation and evaluation of these extensions.

1. Introduction

Fall detection in the domain of Ambient Assisted Living is an active field of research in pervasive computing [1–5]. Indeed, falls are a major reason for serious injuries of elderly people. Especially the absence of help and the remaining on the ground lead to difficult-to-treat long-term effects. The loss of self-confidence and the change in behavior to prevent falls can cause a physical as well as a psychological decline in health which in turn results in a premature death [6]. Inertial sensors which are embedded in smart devices allow to recognize critical falls in an unobtrusive way. Existing work already provides evidence concerning the feasibility and reliability of such approaches (e.g., [4,7,8]). However, a problem of many existing approaches is that they rely on simplifying assumptions, e.g., that the device is always attached to the same sensor position or that a single algorithm works smoothly in all situations [9]. These assumptions limit their applicability in real world scenarios. In particular, several researchers investigated and presented fall detection approaches

☆ A preliminary version of this work appeared in the Proceedings of the 16th IEEE International Conference on Pervasive Computing (PerCom 2018).

* Corresponding author.
E-mail addresses: christian.krupitzer@uni-wuerzburg.de (C. Krupitzer), timo@informatik.uni-mannheim.de (T. Sztyler), janick.edinger@uni-mannheim.de (J. Edinger), martin.breitbach@uni-mannheim.de (M. Breitbach), heiner@informatik.uni-mannheim.de (H. Stuckenschmidt), christian.becker@uni-mannheim.de (C. Becker).

https://doi.org/10.1016/j.pmcj.2019.05.007
1574-1192/

but most of them are optimized to a specific dataset or restricted to a specific setup [10–12], e.g., at a specific position such as wrist-worn or hip-attached.

This paper is an extension of our work presented at the PerCom 2018 [13]. There, we presented a framework that automatically applies the most suitable fall detection algorithm. In this context, we used the current sensor position and physical characteristics of the subject for selecting the best performing setup. Using a leave-one-subject-out approach [14], we trained the algorithms with personalized models. We used the FESAS framework [15] to build a self-adaptive system, i.e., it exchanges the required algorithms and modules on demand to ensure a high reliability. In this context, we also relied on the self-improving layer [16], an architectural extension for online learning of new system configurations. In several experiments, we showed that the performance of a fall detection system relies on the position of the sensor and that it is beneficial to adapt the classifier for fall detection and its model depending on the position of a wearable.

For this paper, we present additional case studies that go beyond position-awareness, derived from the future work mentioned in our original PerCom publication [13], including new algorithms, cross-positional sensor fusion, outlier detection, and a smart fall alert. This paper is structured as follows: In Section 2, we discuss existing fall detection systems. Section 3 describes our system architecture, i.e., the self-adaptive system. In Section 4, we present the results of our self-adaptive fall detection system. Based on the future work identified in [13], Section 5 introduces several case studies including (i) new fall detection algorithms, (ii) cross-positional sensor fusion-based fall detection, (iii) an outlier detection approach for fall detection, and (iv) a smart fall alert. Finally, Section 6 concludes the paper with a cross-case discussion and future work.

2. Existing fall detection systems

Mubashir et al. [2] proposed a tripartite classification of fall detection systems in (i) wearables, (ii) ambient-based, and (iii) vision-based. Wearables include different types of devices with sensors that are attached to a human's body. Ambient-based approaches integrate different sensors into the environment for detecting falls using vibration, video, and audio signals. Vision-based fall detection systems combine various algorithms to detect falls in video streams. Compared to vision-based and ambient-based approaches, wearables for fall detection have several advantages [2]: they are relatively cheap, mobile, less intrusive, and beneficial from a privacy perspective. Due to this flexibility, we focus on the class of wearable sensors. In the following, we present an overview over the positions of wearables as well as algorithms for fall detection that are present in literature.

2.1. Positions of wearables for fall detection

A wearable fall detection system depends on sensor measurements to distinguish falls and Activities of Daily Living (ADL) [17]. The most common sensors used in fall detection systems are accelerometer, gyroscope, magnetometer, and inclinometer [3,18]. According to [3], the majority of wearables for fall detection are attached to the chest, waist, or thigh. Alternatively, other wearable fall detection systems use placements at the forehead, ear, neck, shoulder, back, wrist, ankle, or foot. Extremities such as arms are involved in nearly every movement and a device worn at the wrist is therefore very active. In contrast, a waist-worn sensor stays steady most of the time. Doughty et al. [19] showed that the determined motion data based on waist- or chest-worn devices are quite similar, while knee- or thigh-worn sensors result in lower signals making it harder to distinguish between ADLs and fall events. Consequently, most wearable fall detectors are attached to the torso [3,7].

2.2. Fall detection algorithms

Approaches to distinguish between falls and ADLs [1] are based on (i) thresholds or (ii) machine learning. Threshold-based approaches for fall detection are comparatively simple algorithms that recognize a fall whenever input values exceed predefined thresholds. Detailed reviews on algorithms for threshold-based fall detection can be found in [3,4], and [20]. In contrast, machine learning approaches based on pattern recognition are more sophisticated compared to threshold-based approaches [1]. In literature, different machine learning procedures can be found for fall detection, including Support Vector Machine (SVM), Artificial Neural Network (ANN), k-NN, Decision Trees, Naive Bayes, Hidden Markov Model, or Fuzzy Frequent Pattern Mining.

Both, threshold-based and machine learning approaches for fall detection, suffer from overfitting to a (sometimes rather small) dataset as comparisons of datasets in [10] and [11] have shown. Hence, they might achieve almost perfect accuracy for their test data but might fail for other test subjects. Additionally, as we focus on wearables, the user might change the position of the device, which influences the choice of an algorithm as the motion patterns are changed.

In [13], we proposed a self-adaptive system that is able to adapt the algorithm for fall detection at runtime. This includes determining the current sensor position based on the user's movement pattern. Whereas the fact that a sensor's position is an important design consideration for a fall detection algorithm is acknowledged in research, to the best of our knowledge, there is no approach that adapts the fall detection algorithm to the sensor's position. However, the study in [13] is neither a comparison of algorithms for fall detection, nor claims to integrate all available algorithms for fall detection. Rather, it offers a flexible framework for adapting the fall detection procedure. This paper extends the fall detection system with (i) new algorithms, (ii) cross-positional sensor fusion, (iii) an anomaly detection based approach to fall detection, and (iv) a smart fall alert. For comparisons of algorithms (cf. [7,10–12,21–24]) or overviews of existing algorithms (cf. [1,3,4,12]) the reader is referred to the literature.

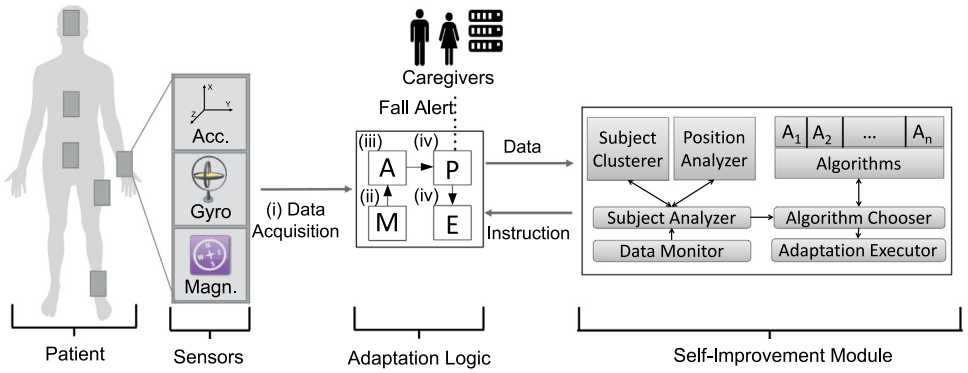

Fig. 1. Design of the self-adaptive fall detection system as MAPE elements representing (ii) data preprocessing, (iii) fall detection, and (iv) fall alert.

3. Self-adaptive fall detection

We designed our system as a self-adaptive system, so that it is able to adapt the fall detection algorithm to the current position of the wearable. Next, Section 3.1 explains the concepts of self-adaptive software systems. After, we present the system design for (i) our self-adaptive fall detection system (cf. Section 3.2) as well as (ii) the self-improvement module that detects position changes and adapts the fall detection algorithm accordingly (cf. Section 3.3). Fig. 1 shows the design of our system for self-adaptive fall detection.

3.1. Self-adaptive systems in a Nutshell

Self-adaptive systems enable adaptation of software at runtime as reaction to changes in their resources or in their environment [25]. Therefore, such systems integrate an adaptation logic that controls the managed resources. Within the adaptation logic, the MAPE control cycle **M**onitors the managed resources as well as their environment through sensors, **A**nalyzes whether the current system performance could be increased through adaptation, **P**lans the change of system parameters, the system structure, or context-adaptation, and **E**xecutes the adaptation on the managed resources [26]. Mapping to fall detection, the MAPE components capture the data (M), analyze whether a fall happened (A), and plan the reaction (P), e.g., informing caregivers in case of a fall (E). Adaptations can be changes of parameters or the system's structure [25], i.e., switching the sensor for data capturing or adjusting the algorithm for fall detection. To decide whether the current state requires adaptations, self-adaptive systems use models, rules, goals, or utility functions in the MAPE cycle [25]. However, uncertainty at runtime can lead to incomplete goals, rules, or models as well as non-optimized utility functions. Self-improvement is *the adjustment of the adaptation logic to handle former unknown circumstances or changes in the environment or the managed resources* [16]. This adjustment of the adaptation logic addresses the aforementioned issues. The self-improvement module in our fall detection system detects a position change of the wearable device and adapts the fall detection algorithm accordingly. In the following, we explain the adaptation logic and the self-improvement module of our fall detection system in greater detail.

3.2. Fall detection system

Nearly every fall detection system uses acceleration-based information as these signals are the most reliable information that can be used to detect a fall. Some authors add other sensor types such as a gyroscope to complement the acceleration data. The position of the sensors influences the patterns of this data. Reliability, usability, and acceptability are all strongly influenced by the device placement on the patient's body [24]. Different works evaluated that a device placement close to the body's center of gravity provides the most reliable sensor measurements [3,7]. As a smartphone offers the required sensors and can be worn near to the body's center, many authors propose a smartphone-based solution to fall detection. Our system tolerates smartphone-based solutions as well as dedicated wearables.

The literature specifies a sequence of actions for fall detection [2,27]: (i) Data acquisition, (ii) Data preprocessing, (iii) Fall detection, and (iv) Fall alert. Our system supports all of these actions. In the following, we will explain them in more detail. These activities are highlighted in Fig. 1. The adaptation logic collects the raw data from the sensors (cf. (i) data acquisition). Within the adaptation logic's monitor, the data is aggregated into windows and features are extracted (cf. (ii) data preprocessing). As next step in the adaptation logic, the analyzer is triggered. Using the aggregated features and specified algorithm, the analyzer divides ADLs from falls (cf. (iii) fall detection). In case of a detected fall, the planner decides corresponding actions – e.g., an acoustic signal or a notification to caregivers – and the executor triggers these actions (cf. (iv) fall alert). A smartphone can serve as an intermediary for informing caregivers and running the adaptation logic, independent from whether dedicated wearables or the smartphone itself perform the sensor data

collection. Whenever an alert message is transmitted from the wearable fall detector to the intermediary, the intermediary is responsible for further activities, such as informing caregivers or accessing remote healthcare services. This part is not covered in this paper.

3.3. Adding self-improvement

As shown in Fig. 1, we add a module for self-improvement to the system. There are two main purposes for the self-improvement module. First, the position of wearables for fall detection might change over time. This changes the pattern of the movement data. As a result, it might be possible, that the fall detection algorithm's performance is decreased as these algorithms are optimized for a specific data pattern. The self-improvement module's *Subject Analyzer* is responsible for detecting the current sensor position for a subject. Therefore, a subject's physiological characteristics – which can influence her movement patterns – are taken into account. Second, after detecting the current sensor position, the most suitable algorithm is chosen and its parameters are optimized. This is the responsibility of the self-improvement module's planner—the *Algorithm Chooser*. The Algorithm Chooser integrates rules that specify the best algorithm depending on the sensor position and the characteristics of the person and returns an algorithm that improves the system performance for the new position of the device.

Due to the disparate characteristics of a fall compared to ADLs, we expect that most of the fall detection algorithms recognize true positives reliable, hence, the improvement is to reduce false positives which can annoy users and decrease their satisfaction and trust in the device. In this context, the executor triggers the change of a fall detection algorithm.

4. A position-aware self-adaptive fall detection system

We used the FESAS framework for implementing the adaptation logic as well as the self-improvement module. FESAS offers support for developers of self-adaptive systems [15]. In this section, we present the implementation of our self-adaptive fall detection system. Additionally, this section subsumes the results of the evaluation of the self-adaptive system. Subsequently, in Section 5 we present several case studies with possible extensions to the system.

4.1. Implementation

The adaptation logic's approach is reusable for other fall detection devices. It only relies on a stream of raw acceleration sensor data of the x-, y-, and *z*-axis. We defined interfaces for the interaction between adaptation logic and the sensors as well as the self-improvement module, so that other systems might be customized to be plugged into our approach for adaptation of the fall detection algorithm. As fall detection algorithms, we implemented (i) two threshold-based algorithms from [12] and (ii) machine learning algorithms based on SVM, k-NN, Random Forest, and J48 decision trees using the WEKA machine learning framework.

The algorithms rely on time windows with feature vectors rather than the unprocessed raw data. The essential idea behind generating windows from a time-dependent data stream is to compute feature vectors that represent the performed activity in a more general way. We use windows which overlap by half and have a length of one second and consider the most common time- and frequency-based features (cf. [13]). Time-based feature values are transformed into frequency-based ones by applying Discrete Fourier transform. Additionally, we separate the acceleration and gravitational force with a low-pass filter to derive the gravity vector. The implementation of the window manager and the feature extraction is based on [9].

The implementation of the self-improvement module is based on recent work from [16]. As described in the previous section, the self-improvement module detects the current sensor position and adapts the fall detection algorithm accordingly. Therefore, the adaptation logic regularly sends the generated windows to the self-improvement module's monitor.

The analyzer is designed for modularity: It can incorporate different analyzing modules for different reasoning purposes that might be triggered simultaneously or sequentially. We implemented one exemplary sub-module: the Subject Clusterer and the Position Analyzer. First, the *Subject Clusterer* assigns the subjects to clusters. To achieve this, we used the data of the subjects to build a clustering model based on the subjects' age and Body Mass Index (BMI) using X-Means.[1] Second, the *Position Analyzer* detects the current sensor position. We treat position detection as a multi-class classification problem with the target classes chest, waist, and thigh. In this context, we trained a subject-independent classifier, hence, we considered all available training data except the target subject to train a classification model that derives the sensor's position. For avoiding oscillations due to a single wrong classification, a position change is only considered after having detected the new position twice. If the Position Analyzer detects that the sensor position was changed, it triggers the rule-based Algorithm Chooser, which chooses the most suitable algorithm and its configuration for the predicted position.

[1] In [13], the results of a preliminary experiment of the group-based clustering approach performs not as expected as the characteristics of the group of users was too homogeneous. Accordingly, we do not further discuss the subject clustering here.

Table 1
Datasets used for the evaluation.

Dataset	Device type	Position	Frequency	Subjects
MMSys [22]	SensorTag	Chest, Thigh	100 Hz	32
UMA [28]	SensorTag Smartphone	Chest, Waist Thigh	20 Hz 200 Hz	9
UniMiB SHAR [11]	Smartphone	Thigh	50 Hz	30
SisFall [12]	Self-built	Waist	200 Hz	23

4.2. Evaluation

We evaluated our approach across multiple datasets, subjects, sensor positions, and device types to demonstrate that our self-adaptive fall detection system can deal with heterogeneity. Therefore, we conducted a series of experiments, using publicly available data to make our results comparable and reproducible. The outline for the experiments is as follows:

- Experiment 1: We tested how well different fall detection algorithms perform for different datasets and evaluated whether the results can be generalized.
- Experiment 2: We tested whether the performance of the algorithms depends on the sensor position on the subjects' body.
- Experiment 3: To test the applicability under real world conditions, we created routines for subjects including changes of the sensor position and ran our system on unmodified raw data.

In the following, we provide aggregated results[2] for each experiment. Table 1 shows the datasets that we will briefly describe in the following.

MMSys: The dataset consists of accelerometer and gyroscope data collected from 42 subjects who performed two protocols: (i) four types of falls and several ADLs and (ii) ascending and descending a staircase. 32 subjects performed both protocols, 10 only followed the second protocol. The data were labeled manually as falls and ADLs.

UMA: The dataset holds accelerometer, gyroscope, and magnetometer data from 19 subject. These subjects repeated 8 types of ADLs and three types of falls. The data is incomplete for 10 subjects. Fall onsets and offsets in this dataset have to be identified manually as a fall trace consists of multiple seconds of data before the fall, the fall itself, and post-fall phase. The fall itself is not extracted.

UniMiB SHAR: For this dataset, acceleration data from 30 subjects were collected. The subjects performed 9 types of ADLs as well as 8 types of falls.

SisFall: The dataset consists of acceleration and gyroscope data from 38 subjects of whom 23 performed both, 10 ADLs and 3 falls. Similar to the UMA dataset, fall data trails contain pre- and post-fall data.

Experiment 1—Cross-Datasets Fall Detection: In the first experiment, we evaluated the performance of multiple machine learning algorithms for fall detection. We focused on a *leave-one-subject-out* approach where we, first, investigated each dataset independently. This means that we only trained the models on data of $n - 1$ subjects of one dataset and tested the model on the remaining subject. Subsequently, we also used data from the other datasets as training set to see how well the models perform across datasets. For both settings, we trained a single classifier for each subject. Overall, the Random Forest classifier performed best. The results show that existing approaches are optimized for particular datasets but fail in classifying falls of another dataset. Additionally, we performed the comparison of algorithms across datasets. The performance of the classifier decreases when data from multiple datasets is used for training.

Experiment 2—Position-Aware Fall Detection: In Experiment 1, we neglected the fact that the algorithms were trained and tested on data of different sensor positions at the same time. However, users can wear the sensors at different positions on their body. Therefore, we decided to perform an evaluation with position-aware classifiers, i.e., we trained a model for each position only with data from that position. With position-aware fall detection we focus on building a classification model for a single sensor position by training only on labeled acceleration data of this specific position. In doing so, we reduce the amount of heterogeneity in the training data and, thus, are able to train more specialized models. The results show that optimizing classifiers for a position improves the fall detection rate.

Experiment 3—Self-Adaptive Fall Detection: In addition to the previous experiments that focused on individual aspects of fall detection algorithms, we evaluated the performance of our overall self-adaptive fall detection system. Therefore, we used as input the raw data of a subset of subjects from the UMA [28] and MMSys [22] datasets and performed the whole fall detection procedure, that is, (i) generate windows and compute feature vectors at runtime, (ii) recognize the sensor position based on these features,[3] (iii) select the best classifier for this position, and (iv) classify the windows into *Falls* and *ADLs*. For each subject, we manually merged data streams from different device positions to

[2] A detailed discussion of the results can be found in [13]. Individual results for each experiment and the preprocessed data are publicly available: https://sensor.informatik.uni-mannheim.de#results2018hips.

[3] For details of the position recognizer, the interested reader is referred to [13].

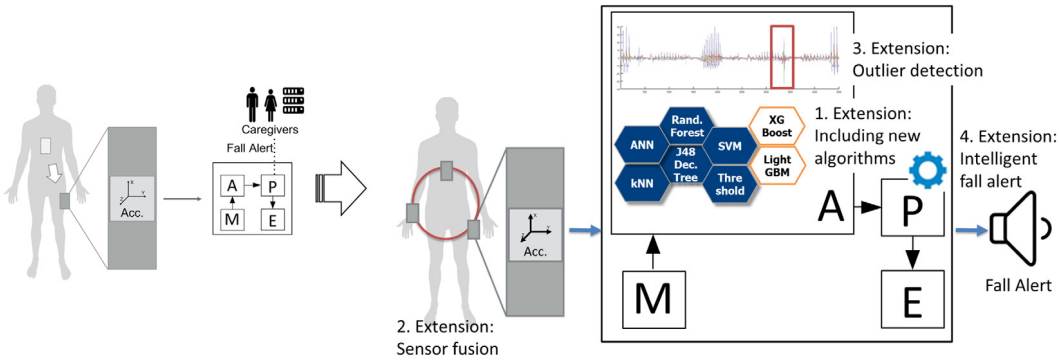

Fig. 2. Left: Original self-adaptive fall detection system from [13]. Right: The case studies presented in this paper.

simulate changes of the sensor position at runtime, e.g., when the subject moves the smartphone from the front chest pocket to the trousers pocket. As baseline, we applied all fall detection algorithms for each subject without using the self-improvement module, i.e., the fall detection algorithm did not recognize the sensor position and, thus, did not change the classifier for detecting falls. For comparison, we ran the overall process of our system, including the position detection and classifier selection with the Random Forest algorithm. By adapting to the current position, we were able to improve the fall detection.

5. Beyond position-awareness

The previous section subsumed the contributions and results of [13]. In that paper, we focused on the implementation of a self-improvement module for adapting the fall detection algorithm based on the position of the device. This mainly targets the analyzer function of the fall detection device, as this functionality is responsible for analyzing the monitored acceleration data.

In the following, we present case studies that address the future work identified in [13]. Fig. 2 depicts how these case studies extend our original system. First, Section 5.1 presents additional algorithms that extend the set of fall detection algorithms mentioned in Section 4.1. We implement XGBoost [29] and LightGBM [30]. To the best of our knowledge, both have not been applied to fall detection so far. However, due to their good performance in other similar use cases, it seems to be promising to evaluate them for fall detection.

Second, Section 5.2 describes an approach for cross-positional sensor fusion. Different positions for devices influence the performance of the fall detection [13]. Further, it might be possible that the different characteristics of falls – e.g., falls with vertical and horizontal movements simultaneously versus falls with mainly vertical movement – might additionally influence the performance of devices at different positions. Accordingly, we evaluate an approach that aggregates the streams of acceleration data from different positions.

Third, in accordance with recent literature that proposes one-class classification to fall detection (e.g., [21,31,32]), we evaluate in Section 5.3 an approach to outlier detection based on one-class classification. The algorithm is trained on data of ADLs only and detects falls as outliers. The advantage of this approach is that it omits the requirement to have fall data to train the algorithm. This is beneficial since high-quality, real-world training data of falls is difficult to collect.

Lastly, we present a smart fall alert in Section 5.4. Whereas the previous three approaches influence the analyzing functionality, the smart fall alert tries to decrease the amount of wrong fall alerts. As the machine learning approaches work window-based, one single window identified as fall triggers an alert in the basic implementation. In the previous results, especially ADLs such as running with high acceleration peaks trigger wrong fall alerts, i.e., false positives. The smart fall alert does not directly trigger an alert if the analyzer recognizes the fall, but shortly observes the situation and then decides about the necessity for a fall alert.

5.1. New algorithms

An adaptive fall detection system is only as good as the available algorithms. For that reason, we compare the set of algorithms from [13] with XGBoost [29] and LightGBM [30] where the motivation is two-part. First, these algorithms have not been used for fall detection but outperform the Random Forest in several machine learning applications. Indeed, these algorithms are also tree-based ensemble methods but differ significantly from the Random Forest. Simply put, the Random Forest relies on bagging which decreases the variance of the prediction where XGBoost and LightGBM are using boosting which should reduce the bias. This leads to the second point. It is not clear whether bagging or boosting is preferable in respect of fall detection or if it depends on the scenario. We want to compare these algorithms to clarify the difference in learning and to discuss the results in detail. Subsequently, we will outline our new results and compare them with the Random Forest, mainly focusing on the cross-dataset performance but also on the different on-body positions.

Table 2
F_2-Measure for baseline measurement across datasets for XGBoost and LightGBM in comparison to Random Forest.

Case		RandomForest	XGBoost	LightGBM
Fall	MMSys	0.80	0.67	0.84
	UMA	0.76	0.68	0.80
	UniMiB SHAR	0.65	0.59	0.74
	SisFall	0.74	0.64	0.80
ADL	MMSys	0.96	0.94	0.90
	UMA	0.94	0.92	0.87
	UniMiB SHAR	0.89	0.87	0.81
	SisFall	0.85	0.89	0.82
avg.		0.83	0.78	0.83

Implementation

Noise, variance, and bias are the main factors of classification errors, i.e., that the prediction does not fit the ground truth. Ensemble methods try to reduce these factors by combining several classification models into one predictive model where bagging and boosting are common strategies for how to build and combine classifiers for reducing the variance or bias, respectively. Random Forest, XGBoost, and LightGBM are such ensemble methods as they consist of several decision trees. Random Forest builds bagged trees while XGBoost and LightGBM build boosted trees. In case of bagging, the trees are built in parallel and independently (i.e. they are uncorrelated). In case of boosting, classifiers need to be built in sequence as each classification model should learn from the errors of the preceding model aiming to minimize a loss (or cost) function. In this context, training samples are also used to measure the performance of an individual predictor where misclassified samples gain weight and correct classified samples lose weight. This information is taken into account while the next tree is built mainly focusing on samples that were previously misclassified. Hence, the next tree always tries to recover the loss.

However, even if both XGBoost and LightGBM are using boosted trees, they also differ significantly especially in how the trees are created. More precisely, XGBoost uses a histogram-based algorithm for making a split decision where for each feature all values are split into discrete bins to determine the best split. In contrast, LightGBM uses a gradient-based one-side sampling strategy[4] which filters samples based on the gradient. Thus, at each node all instances having a large gradient are kept where random sampling is performed for choosing instances with a small gradient. The idea is that training samples with small gradients already have a smaller training error. For comparison, the Random Forest only considers a randomly chosen subset of features at each node for making a split decision. In each case, Information Gain or Gini Index is considered for measuring a split quality.

Beside the splitting strategy, XGBoost and LightGBM also differ in respect of the growing strategy, i.e., XGBoost uses a level-wise while LightGBM uses a leaf-wise growth strategy.[5] The advantage of a level-wise strategy is to keep the tree balanced where the leaf-wise strategy can produce very deep branches which in turn makes it more prone to overfitting. However, the advantage of the leaf-wise strategy is to be more flexible where the result of a leaf-wise strategy can be the same as of a level-wise strategy but not vice versa. In this context, the leaf-wise strategy chooses always the node which reduces the loss the most. Overall, there is no best setting and the most suitable strategy depends on the domain and scenario.

Evaluation

For the evaluation, we repeated the cross-datasets (Experiment 1) and position-aware experiments (Experiment 2) where we tested how these algorithms perform for different datasets and whether the device position has an influence on the performance.

Table 2 shows the performance of XGBoost and LightGBM compared to Random Forest in recognizing a fall across different datasets. It points out that for all datasets LightGBM has a higher F_2-score[6] in recognizing a fall than Random Forest. However, this goes hand in hand with a lower F_2-score in recognizing ADLs where overall LightGBM and Random Forest perform equal. Thus, while LightGBM is able to recognize more correct falls than Random Forest, also more ADLs are wrongly classified as a fall. This is a trade-off between precision and recall where we believe that, in our scenario, recall is more important because a false alarm can be just confirmed as such. In contrast, a missed fall can lead to difficult to treat long-term effects.

The results of XGBoost show that the F_2 score of recognizing ADLs is comparable to the performance of Random Forest but the number of recognized falls is significantly lower than for Random Forest and LightGBM. We think that this is evidence that a leaf-wise growth strategy is most suitable for building tree-based classifier for fall detection.

4 LightGBM also supports the histogram based algorithm but the gradient-based one-side sampling strategy is provided by LightGBM exclusively.

5 In recent implementations, XGBoost also supports the leaf-wise growth strategy.

6 In this paper, we use the F_2-score as optimizing for precision would not be very sufficient if at the same time the recall value is getting worse, i.e., we miss falls. Hence, we state that recall is more important than precision and use the F2 measure. We acknowledge that this might reduce

Table 3
F_2-Measure for position-aware measurements for XGBoost and LightGBM in comparison to Random Forest.

Case		RandomForest	XGBoost	LightGBM
Fall	Chest	0.82	0.73	0.82
	Thigh	0.74	0.62	0.75
	Waist	0.75	0.67	0.75
ADL	Chest	0.96	0.95	0.93
	Thigh	0.92	0.90	0.87
	Waist	0.93	0.91	0.89

Splitting the data by position and creating position-specific classifiers shows a different picture. As shown in Table 3, Random Forest and LightGBM have almost the same performance in recognizing falls while Random Forest makes less misclassification in respect of ADLs. It seems that in a position-independent scenario, the LightGBM is able to distinguish the data in a better way, i.e., boosted trees seem preferable when the device position is unknown while bagging seems to be more appropriated when data is low-dimensional. Besides, these results also confirm once again that the chest seems to be the best on-body position for fall-detection. Overall, we believe these results illustrate yet again that an adaptive system is required to recognize falls reliably.

From a high-level perspective, these results also allow to draw a conclusion regarding the bias–variance trade-off. As a reminder, high variance causes that the model learns the noise in the data while high bias usually results in missing important relations between features and target classes. Since we have used fall detection datasets from various authors, a high bias could be already expected beforehand. Considering our results, the Random Forest is one of the best performing algorithms which indicates also a high variance.

We assume that this can be attributed to the fact that falls can happen in several ways. Besides, as the performance between Random Forest, XGBoost and LightGBM does not differ as considerably as it is the case in other domains, it might be worth to verify also the performance of other variants of already considered classification techniques. This includes *x*NN (X-Nearest Neighbor Graph) but also a combination of an Autoencoder (i.e. transforming the raw features into embeddings) with a Logistic Regression might be a promising approach for addressing the identified problems.

5.2. Sensor fusion

Sensor fusion allows to process data from two sensors attached to different body positions at the same time. In pervasive computing environments, computing devices surround us anytime and everywhere. Already today, many people use wearables such as smartwatches or smartglasses in addition to their smartphone. Moreover, stationary sensors in cameras or floor mats complement the range of sensors available for fall detection in certain situations. This consistent availability enables to combine the data from various sensors at the same time.

The main motivation for applying sensor fusion is to improve the accuracy of the distinction between falls and ADLs. As shown in [13], different device positions influence the performance of the fall detection algorithms notably. Therefore, aggregating data from different positions may help to detect certain movement patterns which would remain unrecognized when using data from one sensor only. In addition to accuracy improvements, integrating multiple sensors enhances availability. For instance, if the smartphone shuts down due to low battery, smartwatch or camera data may still suffice to perform fall detection successfully. On the downside, sensor fusion may increase the costs for fall detection [33]. Assuming that the sensors are not already installed for other purposes, users face costs for buying additional sensors. Further, algorithms must process a higher amount of data at the same time. This complexity leads to computation costs.

The sensor fusion approaches in literature differ notably in terms of sensor types, number of sensors, and algorithms. In [33], Koshmak, Loutfi, and Linden provide an overview of sensor fusion approaches for fall detection. However, they do not consider approaches which use sensors of the same type only. Fall detection systems which, e.g., rely on accelerometer data from different body positions are excluded. Bianchi et al. combine a barometric pressure sensor with the well-known accelerometer-based fall detection [34]. The barometric pressure sensor measures the altitude of the device and is used to improve the rate of false alarms. Li et al. use two devices, one attached to the chest and one attached to the thigh [35]. Both devices contain a gyroscope and an accelerometer for a three-step fall detection process based on posture recognition. Ojetola et al. detect falls with C4.5 decision trees [36]. The features used in the classification base on values from accelerometers and gyroscopes attached at chest and thigh.

comparability with other works but we discuss in [13] that the comparison is limited as works in the field use different data collection protocols and labeling approaches.

Table 4
Evaluation of the sensor fusion approach.

Case		RF baseline	RF chest	RF thigh	RF sensor fusion
Fall	Precision	0.772	0.802	0.762	0.815
	Recall	0.688	0.635	0.763	0.813
	F_2-measure	0.703	0.663	0.763	0.813
ADL	Precision	0.929	0.919	0.945	0.957
	Recall	0.953	0.964	0.945	0.957
	F_2-measure	0.948	0.955	0.945	0.957

Implementation

Similar to Li et al. [35] as well as Ojetola et al. [36], we integrate sensor fusion from two sensors attached to different body positions at the same time. In [13], we showed that the Random Forest classifier performed best across datasets. Here, we adjust the Random Forest to not only consider features from one accelerometer but from two accelerometers simultaneously. We choose to use two sensors since using more significantly decreases recall values [28]. The features itself are identical for each sensor which results in twice as many features for the sensor fusion approach in comparison to the Random Forest used in [13].

Evaluation

To show the effectiveness of sensor fusion, we compare the performance of a Random Forest classifier that relies on features generated from only one sensor to a Random Forest classifier that combines features from two sensors at different positions. Sensor fusion is only applicable on simultaneously recorded data from several sensors. From the four datasets used for the evaluation in the previous chapter (cf. Table 1), only the MMSys [22] and the UMA [28] dataset contain data from different sensor positions. This evaluation uses the MMSys dataset. Contrary to the UMA dataset, the MMSys dataset provides synchronized measurements from accelerometers placed at chest and thigh. Moreover, it has a higher number of subjects.

Table 4 shows the results of the evaluation. The Random Forest used for the baseline measurement is trained and tested across positions. Sensor fusion leads to significant improvements compared to this algorithm. Especially for fall windows, recall as well as precision increase notably (by 0.125 and 0.043, respectively). This also holds true for ADL windows. As the previous chapter shows, algorithms optimized for a certain position outperform algorithms trained across positions. Table 4 further compares the sensor fusion approach to these position-aware algorithms. Similar to the baseline, sensor fusion improves recall and precision in this case as well. Thus, sensor fusion is a helpful extension to improve the accuracy of fall detection while even decreasing the number of false alarms.

5.3. Outlier detection

Most often, fall detection algorithms implement machine learning based approaches. Such approaches need data for training purposes. In the case of fall detection, this requires training subjects to perform falls as falls are rare in daily life. However, this can be a threat for validity. On the one hand, these falls are simulated, which might result in different patterns compared to real falls. On the other hand, fall detection systems often target elderly people as users; whereas the simulated falls are often performed by young people due to the risk for injuries for the elderly. Consequently, the falls for training might have a different pattern than the falls observable in reality.

Due to this difficulty to collect proper fall data for training, several approaches from prior research use training data from ADLs only. Kahn and Hoey [31] reviewed fall detection techniques and classified according the availability of sufficient training data. In case that no fall data is available – or it should be avoided to use such data for the aforementioned issues – they propose outlier/anomaly detection techniques or one-class classification. Whereas Medrano et al. [21] do not experience benefits of outlier detection, Micucci et al. [32] successfully applied one-class classification based on k-NN and SVM and achieved similar results as two-class classification with k-NN and SVM that used training data. Another approach by Zhou et al. [37] used transitions between ADLs to train one-class SVM for anomaly detection of falls. As different class of approaches, Khan et al. [38] use Hidden Markov Models to detect falls as anomalies. In line with this rather new research stream, in the following, we propose an approach for fall detection based on anomaly detection.

Implementation

Following the approach of Micucci et al. [32] that achieved results similar to supervised learning without the need to train the classifiers with fall data, we implemented one-class classification variants of our SVM implementations. Micucci et al. [32] used four different variants of the algorithm classes: a concatenation of raw data, magnitude, acceleration features, and local temporal patterns. They tested their approaches on the tFall dataset from [21]. For better comparability with our results, we use the same window manager as in the previous experiments and three implementations of SVM outlier detection. First, all features from our algorithm from Section 4.1 are used in the outlier SVM detection classifier. Second, two implementations with the features sets proposed in [32] are implemented: (i) a

Table 5
F_2-Measures of the outlier approaches (Mag = Magnitude; Feat = Features from [32]; All = SVM outlier with same features as 2-class SVM from [13]) and the 2-class SVM and Random Forest (RF) as comparison for cross-dataset and within each dataset.

Case		Each dataset					Cross-dataset				
		SVM	RF	Mag	Feat	All	SVM	RF	Mag	Feat	All
Fall	MMSys	0.55	0.80	0.43	0.58	0.58	0.30	0.80	0.44	0.62	0.61
	UMA	0.43	0.77	0.20	0.35	0.42	0.44	0.76	0.15	0.51	0.43
	SHAR	0.34	0.64	0.19	0.54	0.44	0.41	0.65	0.23	0.52	0.50
	Sis	0.50	0.75	0.27	0.49	0.48	0.33	0.74	0.27	0.55	0.51
ADL	MMSys	0.94	0.96	0.56	0.56	0.55	0.89	0.96	0.57	0.62	0.60
	UMA	0.84	0.93	0.59	0.55	0.56	0.87	0.94	0.40	0.71	0.58
	SHAR	0.76	0.90	0.52	0.36	0.20	0.84	0.89	0.59	0.36	0.24
	Sis	0.93	0.92	0.56	0.49	0.44	0.83	0.91	0.52	0.56	0.47

combination of energy, mean values, standard deviation, and correlation coefficients and (ii) the magnitude values.[7] We implemented all three algorithms using the outlier detection of the WEKA machine learning framework's LibSVM classifier.

Those algorithms can be seen as an extension to the available set of algorithms from Section 4.1. For training of these algorithms, we used the same data as in our previous experiments. However, the training data is limited to windows labeled as ADLs. This avoids the need to have fall data for training of the algorithms.

Evaluation

We performed several experiments using outlier detection or one-class classification, respectively, for fall detection. In line with our other experiments, we performed baseline measurements across datasets, baseline measurements within the datasets as well as a position-aware experiment using a leave-one-subject-out approach, i.e., using one specific subject as test set and all other relevant ones as training set. For all measurements, we limited the number of windows to 500 per subject. Of these 500 windows, 400 belong to ADLs and up to 100 to falls.

In contrast to our previous results, the F_2-Measures decrease with a specialization of the data, i.e., using data only from the some dataset or only the same position of the wearable for training. Table 5 shows the values for the cross-dataset baseline and each dataset separated, i.e., using only data form the same dataset for training. This can be explained with the fact that outlier detection works better with a larger dataset and different facets of data. On the other hand, this is another advantage: the algorithm needs to have less information on the data. Further, needing more data is not violating the rationale for outlier detection as it should not reduce the amount of data collected for learning in general but eliminate the need of having fall data for training. The assumption of only using data of ADLs is not violated with the cross-dataset approach. In the following, we discuss the cross-dataset baseline results.

From the three outlier detection approaches, the one with the feature combination from [32] works best. As one can see for the cross-dataset analysis, the F_2-Measures for fall detection of all three outlier approaches are above the original SVM implementation. This is interesting, especially as the training does not need to include fall data. On the other hand, the performance does not achieve the performance of the Random Forest approach, which worked best. Furthermore, the F_2-Measures for ADL detection is worst compared to the SVM and Random Forest baseline. More specific, the results show a high precision for ADL detection and a high recall for fall detection, however, a low recall for ADLs,[8] i.e., misclassified ADLs as falls. This results in having false alarms, which decreases usability. The results are in line with research (cf. [32]). There the authors report good results and high applicability of outlier recognition for fall detection. However, the authors focus on the mentioned high precision for ADL detection and a high recall for fall detection, hence, neglecting the fact of fall alerts through ADLs misclassified as falls. Still, outlier detection shows potential for fall detection.

In this paper, we focus on one-class classification for outlier detection based on SVM. To avoid the high number of ADLs that are misclassified as falls, it might be possible to further optimize the parameters of the classifier, which has not been done yet to avoid over-fitting to the datasets. Additionally, other authors propose to use k-NN for the purpose of fall detection [32]. Lastly, the WEKA machine learning framework also offers specific outlier detection which might be tested for fall detection.

5.4. Smart alert

So far, we have looked at the performance of our fall detection systems on a window level. The results showed that not all fall windows could be detected. However, having a closer look, we can find that detecting every single window during a fall is not crucial for detecting a fall. Instead, it is sufficient when at least one window during the fall is detected. Thus, to evaluate the performance of the system in a real life setting it is important to analyze the results for a whole fall

[7] As our window approach should be in line with the window approach used in the other experiments, the magnitude is calculated with the mean of the x, y, and z dimensions instead of calculating a magnitude for each value of the data trace as done in [32].

[8] For detailed results, visit our web site: https://sensor.informatik.uni-mannheim.de/#results2018beyond.

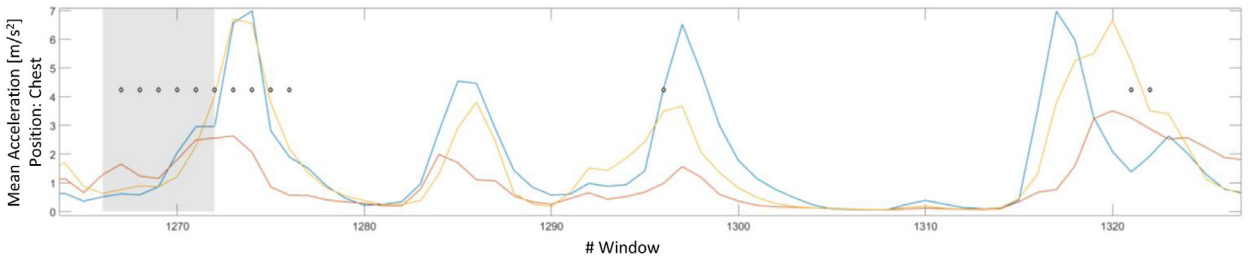

Fig. 3. Sample mean acceleration data (x,y,z) from a chest sensor in the MMSys dataset (Subject 32). The highlighted area shows an actual fall interval as labeled in the original dataset. The markers identify the windows that were labeled as fall windows by the classifier. While the 10 detected fall windows on the left are correct and belong to an actual fall incident, the single ones on the right show false alarms. The peaks in the mean acceleration data after the fall come from ADLs not further specified in the MMSys dataset. For classification, we used the same features as in [13].

incident. Therefore, we analyzed the MMSys dataset because in this dataset falls and ADLs are interleaved unlike in the other datasets where falls and ADLs are recorded separately. In total, the dataset contained 448 falls from which all were detected by our Random Forest classifier. While this shows that the algorithm can detect falls reliably, there are 1216 false positives, i.e., *false alarms*. The smart alert approach uses the different phases of a fall to reduce this number.

A fall can be described as a sequence of several phases [39]. Typically there are four phases, (i) start of fall, (ii) critical phase, (iii) impact, and (iv) post fall. The first phase describes the start of the fall, which can either be caused by tripping, being pushed, slipping or getting unconscious. The critical phase describes the short period of time when the body is moving rapidly towards the ground with an increased vertical velocity. During that phase, there is a temporary period of free fall in which the mass is being accelerated with gravitational acceleration. At the end of this phase, the person's body is rapidly slowed down by hitting the ground. This impact is also described as an acceleration in the opposite direction until the velocity is zero. The post fall phase is characterized by an absence of movement such as the patient is motionless remaining in the post fall body posture.

Each phase can be used to perform or support fall detection mechanisms. To reduce the number of false alarms, here, we use information from the critical phase and the post fall phase.

For the critical phase, we argue that we can identify false alarms that are caused by single misclassified windows. Fig. 3 shows a small trace of an MMSys subject's data. The lines show the mean acceleration data for each window. The highlighted interval indicates an actual fall. The black markers show the windows that the Random Forest algorithm has classified as *'Fall'*. Whereas the 10 markers on the left show correctly classified *Fall* windows, the markers on the right are incorrectly classified as *Fall*. By visually inspecting the data, we found that this is a common pattern. Actual fall incidents that were labeled as falls in the original data consist of multiple fall windows. Besides, there are multiple single windows that are misclassified as *Fall* and would cause a false alarm. Therefore, we do not initiate an alarm based on falls with less than n detected fall windows. Here, we determine the threshold n empirically.

The second option to reduce the number of false alarms is to use data from the post fall phase. This phase is characterized by the absence of movement as severe falls mostly lead to the inability to move. These falls are the most critical ones as the subject might not be able to call for help. Previous systems have made use of the post fall phase to detect severe falls. Mathie et al. [40] propose a system based on accelerometer data to predict movement, posture, and energy expenditure. They use acceleration and posture information to predict falls as high acceleration can indicate a fall and some postures are more likely in the post fall phase. Lee et al. [41] use cameras at the ceiling to observe movement within a room. They detect the absence of movement as well as typical post-fall postures. In their evaluation they show that they have a small number of false alarms while detecting most falls. The system is limited to indoor use-cases and requires a significant setup effort. Zhang et al. [42] detect falls using an accelerometer in a mobile phone. After detecting a high acceleration the time is measured when the sensor is motionless with an acceleration near gravity. When this time exceeds a threshold, the subject is considered motionless and the fall to be critical. Kangas et al. [43] also use acceleration data to retrieve the intensity of the impact as well as the post fall posture.

Implementation

To reduce the number of false alarms, we evaluate the characteristics of a potential fall before triggering an alarm. We eliminated fall predictions with less than n detected fall windows by applying the following steps. First, we grouped the detected fall windows into a detected fall incident. We therefore considered windows to belong to the same fall incident when the difference between them was not larger than 5 windows. Second, we checked the number of detected fall windows in each detected fall incident and excluded falls with less than n detected values where the threshold n ranges from 0 (the baseline) to 10. The excluded falls were considered false alarms.

For the post fall analysis, we used a threshold-based approach where we use the absent of movement as an indicator for a severe fall incident. As the available datasets do not distinguish between the four phases of a fall, we were not able to apply machine learning to set this threshold nor to evaluate the performance. Further, after some falls in the data the

Table 6
Evaluation of the false alarm filtering. The improvement shows how many false alarms could be avoided when detected falls with less than *n* detected windows were ignored.

n	Falls	Falls detected	Falls missed	False alarms	Improvement
0	448	448	0	1216	—
1	448	447	1	556	54.28%
2	448	447	1	359	70.48%
3	448	447	1	262	78.45%
4	448	445	3	191	84.29%
5	448	436	12	138	88.65%
6	448	426	22	78	93.59%
7	448	374	74	49	95.97%
8	448	315	133	38	96.88%
9	448	253	195	26	97.86%
10	448	218	230	26	97.86%

subjects remained on the ground for some time whereas in other falls the got up immediately after the fall. This also made it impossible to learn the characteristics of the post fall phase. However, the results in the literature have shown the effectiveness of this approach [42,43] and thus it should be considered in our smart fall alert approach.

Evaluation

The results of the false alarm filtering show that the benefit of this approach is large (see Table 6). When we exclude predicted fall incidents with only one predicted fall window, more than 50% of all false alarms can be eliminated at the cost of only one missed fall. With a threshold *n* of 3 which means that predicted falls with up to 3 windows are considered false alarms, almost 80% of all false alarms are recognized as such. At the cost of 2 more missed falls, more than 70 further false alarms could be detected. An *n* of more than 4 would further decrease the number of false alarms but would also lead to more missed false which should be avoided. Thus, the optimal window size is 4 as the number of false alarms gets reduced by 84.29% while there are just 3 more missed falls.

We further expect the number of false alarms to decline when we use information from the post fall phase. However, as the available datasets do not provide a consistent behavior after each fall. After some falls, the subjects were instructed to remain lying on the ground. After others, they were told to immediately start crouching or sitting up. Thus, we could not perform a quantitative analysis of this mechanism.

6. Conclusion

We compared the performance of different fall detection algorithms on publicly available datasets. Across all datasets, Random Forest performed best. We found that heterogeneous labeling approaches for the data captured in controlled environments reduce the performances of non-customized fall detection algorithms on these datasets. Furthermore, the algorithms are often optimized for a sensor position. Therefore, we additionally performed a position-aware comparison of the fall detection algorithms on the datasets. The results indicate that knowing the position of the sensor and adjusting the algorithm accordingly is superior to a static algorithm. Additionally, we showed that our self-adaptive fall detection system tackles these issues by determining the current sensor position and adapting the fall detection algorithm accordingly.

6.1. Summary

In this paper, we focus on further improving our self-adaptive fall detection system. First, we added new algorithms based on XGBoost [29] and LightGBM [30]. These algorithms showed very good performance in use cases related to fall detection but to the best of our knowledge they have not been used for fall detection so far. LightGBM has a higher F_2-score in recognizing a fall than our Random Forest which performed best in our basic experiments. While LightGBM is able to recognize more correct falls than Random Forest, also more ADLs are wrongly classified as a fall. However, we

believe that in our scenario recall of falls is more important because a false alarm can be just confirmed as such as shown in Section 5.4. In contrast, a missed fall can be harmful for the user and lead to life-threatening situations. XGBoost shows comparable performance to the Random Forest when detecting ADLs but the number of recognized falls is considerably lower than in case of Random Forest and LightGBM. For position-aware fall detection, Random Forest and LightGBM have almost the same performance in recognizing falls while Random Forest makes less misclassification in respect of ADLs. It seems that in a position-independent scenario the LightGBM is able to distinguish the data in a better way.

Second, we integrated an approach for sensor fusion that allows to use data from different sensors simultaneously. This leads to higher accuracy of the detection and improves reliability. We implemented sensor fusion by enabling the Random Forest classifier to use features generated by two sensors attached to different positions at the same time. The evaluation shows that sensor fusion outperforms similar algorithms that use data from one sensor only by increasing the performance in recognizing falls while even reducing the number of wrongly classified ADLs.

Third, in line with research (e.g., [21,32]), we performed one-class classification or outlier detection, respectively, for fall detection. This way, the classifier is trained on data of ADLs only. This avoids the need to have fall data, which is often only simulated as it is hard to capture real-life fall data. We implemented one-class classification based on LibSVM's one-class classification and tested three sets of features. The results indicate that for cross-dataset experiments the performance is higher than for the two-class SVM classifier. However, the performance decreases for more specialized data, e.g., using data only from the same dataset for training and testing or including position-awareness. Especially a high amount of misclassified ADLs are critical. However, again, this can be addressed with the smart fall alert.

Fourth, we introduced a smart fall alert that does not trigger an alarm on the basis of single windows but that evaluates the characteristics of an entire fall incident. Here, we use two mechanisms: (i) introducing a threshold of windows that need to be detected to be interpreted as a fall and (ii) use of the characteristics of the post fall phase as the acceleration right after a fall when the person is lying on the ground typically is minimal.

6.2. Future work

For future work, it might be beneficial to apply our improvements of the self-adaptive fall detection system in a cross-case fashion. Especially the combination of the first three case studies for analyzing whether a fall happened with the smart fall alert that plan a reaction to that analysis might be promising to benefit from the high precision of some approaches for fall detection while keeping the negative aspects of misclassified ADLs low.

Further potential arises from modifying the window manager. In this paper, we used the window manager from [13] to guarantee the comparability of the results. However, other authors propose different approaches. Event-based window manager (e.g., [36]) try to identify the peaks of a typical fall pattern in the data to use these peaks for analyzes. Other authors propose to use discrete window managers that concatenate the raw data to support the identification of patterns (e.g., [32]). The modularity of our system enables to experiment with other window managers.

So far, we rely on data of acceleration sensors, only. In this paper, we did a fusion of data of acceleration sensors from various positions. However, most devices offer additional types of sensors that are commonly used for fall detection, e.g., gyroscope or magnetometer. For future work, it might be interesting to do sensor fusion by integrating other sensor types. The flexibility of our system model supports this, however, it might result in implementing a new set of features in the window manager and implementing new algorithms for fall detection.

In the preliminary experiments with subject clustering [13], we achieved similar performance as in the baseline tests. However, the results are influenced by the fact that we could only use the age, size, weight, and gender of the subjects. On the one hand, we might miss important information, e.g., we were not able to consider the fitness level of our subjects as this information was not available. Related work suggest to use this factor [9]. On the other hand, we focused on a two-step approach where subjects are first clustered by age and subsequently by their BMI value. This resulted in a group of elderly consisting of five people only, thus, this group was not further clustered. We plan to test other clustering approaches as a leave-one-subject-out approach does not scale in respect of a larger number of people.

Conflicts of interest

The authors declares no conflicts of interest.

Acknowledgements

Martin Breitbach is supported by the German Research Foundation (DFG). The original publication at PerCom 2018 was supported by the DFG Travel Grant and the IEEE CS TCPP & TCCC Travel Grant.

References

[1] R. Igual, C. Medrano, I. Plaza, Challenges, issues and trends in fall detection systems, BioMed. Eng. OnLine 12 (1) (2013) 66.
[2] M. Mubashir, L. Shao, L. Seed, A survey on fall detection: Principles and approaches, Neurocomputing 100 (2013) 144–152.
[3] N. Pannurat, S. Thiemjarus, E. Nantajeewarawat, Automatic fall monitoring: a review, Sensors 14 (7) (2014) 12900–12936.
[4] E. Casilari, R. Luque, M.-J. Morón, Analysis of android device-based solutions for fall detection, Sensors 15 (8) (2015) 17827–17894.

[5] C.-Y. Hsieh, C.-N. Huang, K.-C. Liu, W.-C. Chu, C.-T. Chan, A machine learning approach to fall detection algorithm using wearable sensor, in: Advanced Materials for Science and Engineering (ICAMSE), International Conference on, IEEE, 2016, pp. 707–710.
[6] C. Griffiths, C. Rooney, A. Brock, Leading causes of death in england and Wales–how should we group causes?, Health stat. quart. / Off. Natl. Stat. 28 (9) (2005) 6–17.
[7] H. Gjoreski, M. Luštrek, M. Gams, Accelerometer placement for posture recognition and fall detection, in: 2011 Seventh International Conference on Intelligent Environments, IEEE, 2011, pp. 47–54.
[8] B. Aguiar, T. Rocha, J. Silva, I. Sousa, Accelerometer-based fall detection for smartphones, in: 2014 IEEE International Symposium on Medical Measurements and Applications (MeMeA), IEEE, 2014, pp. 1–6.
[9] T. Sztyler, H. Stuckenschmidt, On-body localization of wearable devices: An investigation of position-aware activity recognition, in: Pervasive Computing and Communications (PerCom), 2016 IEEE International Conference on, IEEE, 2016, pp. 1–9.
[10] R. Igual, C. Medrano, I. Plaza, A comparison of public datasets for acceleration-based fall detection, Med. Eng. Phys. 37 (9) (2015) 870–878.
[11] D. Micucci, M. Mobilio, P. Napoletano, Unimib SHAR: a new dataset for human activity recognition using acceleration data from smartphones, CoRR abs/1611.07688 (2016).
[12] A. Sucerquia, J.D. López, J.F. Vargas-Bonilla, Sisfall: A fall and movement dataset, Sensors 17 (1) (2017).
[13] C. Krupitzer, T. Sztyler, J. Edinger, M. Breitbach, H. Stuckenschmidt, C. Becker, Hips do lie! a position-aware mobile fall detection system, in: 2018 IEEE International Conference on Pervasive Computing and Communications (PerCom), 2018, pp. 1–10.
[14] T. Sztyler, H. Stuckenschmidt, W. Petrich, Position-aware activity recognition with wearable devices, Pervasive Mob. Comput. 38 (2017) 281–295.
[15] C. Krupitzer, F.M. Roth, C. Becker, M. Weckesser, M. Lochau, A. Schurr, Fesas ide: An integrated development environment for autonomic computing, in: 2016 IEEE International Conference on Autonomic Computing (ICAC), IEEE, 2016, pp. 15–24.
[16] C. Krupitzer, J. Otto, F.M. Roth, A. Frömmgen, C. Becker, Adding self-improvement to an autonomic traffic management system, in: Autonomic Computing (ICAC), 2017 IEEE International Conference on, IEEE, 2017, pp. 209–214.
[17] N. El-Bendary, Q. Tan, F.C. Pivot, A. Lam, Fall detection and prevention for the elderly: A review of trends and challenges, Int. J. Smart Sensing Intell. Syst. 6 (2013) 1230–1266.
[18] M. Shoaib, S. Bosch, O.D. Incel, H. Scholten, P.J.M. Havinga, Fusion of smartphone motion sensors for physical activity recognition, Sensors 14 (6) (2014) 10146–10176.
[19] K. Doughty, R. Lewis, A. McIntosh, The design of a practical and reliable fall detector for community and institutional telecare, J. Telemed. Telecare (2000) 150–154.
[20] H.A. Dau, F.D. Salim, A. Song, L. Hedin, M. Hamilton, Phone based fall detection by genetic programming, in: Proc. MUM, ACM, 2014, pp. 256–257.
[21] C. Medrano, R. Igual, I. Plaza, M. Castro, Detecting falls as novelties in acceleration patterns acquired with smartphones, PLoS One 9 (4) (2014) 1–9.
[22] O. Ojetola, E. Gaura, J. Brusey, Data set for fall events and daily activities from inertial sensors, in: Proceedings of the 6th ACM Multimedia Systems Conference, ACM, 2015, pp. 243–248.
[23] F. Bagala, C. Becker, A. Cappello, L. Chiari, K. Aminian, J.M. Hausdorff, W. Zijlstra, J. Klenk, Evaluation of accelerometer-based fall detection algorithms on real-world falls, PLoS One 7 (5) (2012) e37062.
[24] M. Kangas, A. Konttila, P. Lindgren, I. Winblad, T. Jämsä, Comparison of low-complexity fall detection algorithms for body attached accelerometers, Gait Posture 28 (2008) 285–291.
[25] C. Krupitzer, F.M. Roth, S. VanSyckel, G. Schiele, C. Becker, A survey on engineering approaches for self-adaptive systems, Pervas. Mob. Comput. J. 17 (Part B) (2015) 184–206.
[26] J.O. Kephart, D.M. Chess, The vision of autonomic computing, IEEE Comput. 36 (1) (2003) 41–50.
[27] X. Yu, Approaches and principles of fall detection for elderly and patient, in: HealthCom 2008-10th International Conference on E-Health Networking, Applications and Services, 2008, pp. 42–47.
[28] E. Casilari, J.A. Santoyo-Ramón, J.M. Cano-Garcí a, Analysis of a smartphone-based architecture with multiple mobility sensors for fall detection, PLoS One 11 (12) (2016) 1–17.
[29] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 785–794.
[30] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.-Y. Liu, Lightgbm: A highly efficient gradient boosting decision tree, in: Advances in Neural Information Processing Systems, 2017, pp. 3146–3154.
[31] S. Khan, J. Hoey, Review of fall detection techniques: A data availability perspective, Med. Eng. Phys. 39 (2017) 12–22.
[32] D. Micucci, M. Mobilio, P. Napoletano, F. Tisato, Falls as anomalies? an experimental evaluation using smartphone accelerometer data, J. Ambient Intell. Hum. Comput. 8 (1) (2017) 87–99.
[33] G. Koshmak, A. Loutfi, M. Linden, Challenges and issues in multisensor fusion approach for fall detection: Review paper, J. Sensors 2016 (2016) 1–12.
[34] F. Bianchi, S.J. Redmond, M.R. Narayanan, S. Cerutti, N.H. Lovell, Barometric pressure and triaxial accelerometry-based falls event detection, IEEE Trans. Neural Syst. Rehabil. Eng. 18 (6) (2010) 619–627.
[35] Q. Li, J.A. Stankovic, M.A. Hanson, J. Lach, G. Zhou, Accurate, fast fall detection using gyroscopes and accelerometer-derived posture information, in: Wearable and Implantable Body Sensor Networks, 2009. BSN 2009. Sixth International Workshop on, 2009, pp. 138–143.
[36] O. Ojetola, E.I. Gaura, J. Brusey, Fall detection with wearable sensors–safe (smart fall detection), in: 2011 Seventh International Conference on Intelligent Environments, IEEE, 2011, pp. 318–321.
[37] M. Zhou, S. Wang, Y. Chen, Z. Chen, Z. Zhao, An activity transition based fall detection model on mobile devices, in: Human Centric Technology and Service in Smart Space, Springer, 2012, pp. 1–8.
[38] S.S. Khan, M.E. Karg, D. Kulić, J. Hoey, X-factor hmms for detecting falls in the absence of fall-specific training data, in: Ambient Assisted Living and Daily Activities, Springer, 2014, pp. 1–9.
[39] N. Noury, P. Rumeau, A. Bourke, G. ÓLaighin, J. Lundy, A proposal for the classification and evaluation of fall detectors, Irbm 29 (6) (2008) 340–349.
[40] M. Mathie, J. Basilakis, B. Celler, A system for monitoring posture and physical activity using accelerometers, in: Engineering in Medicine and Biology Society, 2001. Proceedings of the 23rd Annual International Conference of the IEEE, 4, Ieee, 2001, pp. 3654–3657.
[41] T. Lee, A. Mihailidis, An intelligent emergency response system: preliminary development and testing of automated fall detection, J. Telemed. Telecare 11 (4) (2005) 194–198.
[42] T. Zhang, J. Wang, P. Liu, J. Hou, Fall detection by embedding an accelerometer in cellphone and using kfd algorithm, Int. J. Comput. Sci. Netw. Secur. 6 (10) (2006) 277–284.
[43] M. Kangas, A. Konttila, P. Lindgren, I. Winblad, T. Jämsä, Comparison of low-complexity fall detection algorithms for body attached accelerometers, Gait posture 28 (2) (2008) 285–291.