Lessons Learned from the Development of Mobile Applications for Fall Detection

Ítalo Linhares, Rossana M. C. Andrade, Evilasio C. Júnior, Pedro Almir Oliveira, Breno Oliveira and Paulo Aguilar

Group of Computer Networks,
Software Engineering and Systems (GREat)
Federal University of Ceará
Fortaleza, Ceará, Brazil
Email: {italoaraujo, evilasiojunior, pedromartins, brenooliveira, pauloaguilar}@great.ufc.br, rossana@ufc.br

***Abstract*—Falls are the leading cause of older adults injuries and solutions are needed to address this issue. One way to meet this is by developing applications that use sensors embedded in devices like smartphones and smartwatches. This paper presents our experience in developing such applications and the lessons learned during their development and evolution. First, we developed an application called fAlert to identify a fall using data from a smartphone's accelerometer. However, the usage of this kind of mobile device for detecting falls is not natural, because it needs to be positioned at the level of the user's chest. Then, we developed a new app called WatchAlert, which runs in smartwatches. In that case, we also created an algorithm that uses two sensors, accelerometer and gyroscope, and later evolved it to use only the accelerometer with better results. Moreover, we use first a fall detection threshold algorithm in this solution. Next, we expanded this strategy to use threshold and machine learning algorithms, which were evaluated considering the accuracy, false negative, and time criteria as well as their features. We believe that this study can support the development of new systems and devices for detecting falls. As future work, it would be interesting to assess the related energy cost of the fall detection approaches studied.**

Keywords–Falls; Lessons; IoT Health; IS for healthcare.

I. INTRODUCTION

According to the World Health Organization (WHO), falls are the leading cause of injuries in older adults. About 28-35% of the people aged 65 years and over fall every year, and this number increases when the person's age is over 70 years old, achieving 32-42% [1]. Also, falls cause 40% of all injury deaths [1]. These facts represent alarming data, and an effort is needed to decrease this number. One way to meet this need is preventing or quickly identifying older adults falls.

There are many ways to detect falls, such as with sensors or computer vision. In case of sensors, they can be embedded in smartphones, smartwatches, and other devices like wearables to monitor the vital signs and the activities performed by the user [2]–[5]. Considering computer vision, we can use image processing to identify if a person has fallen or if he/she is performing another activity [6].

Although we can use computer vision, many users may feel uncomfortable because their activities can be recorded to be processed. Still, users may not trust data storage and the permissions of those who will access the images. Then, we perceive an impact on the usage of these systems related to reliability issues, and we should use a way less invasive to users, which can be using the sensors previously cited.

Based on this, we can find in the literature efforts to detect a fall using sensors embedded in wearables devices such as smartphones and smartwatches as in [2] [7]–[10] or mixing data of wearables with other devices as in [11]. Examples of sensors present in these devices that can help detect a fall are the accelerometer and the gyroscope to check the user movement, and the magnetometer that can aid in the identification of the fall location.

Thus, we developed two applications and three thresholds-based algorithms to detect a fall. The first solution developed is the fAlert application [4], which runs in smartphones and uses accelerometer and magnetometer sensors. As the device should be situated in the user's chest, its usage is not natural to the older adult. So, we sought to evolve the solution using smartwatches.

This solution is the WatchAlert application [5], which, in the first version, was developed with two sensors, accelerometer and gyroscope, to detect a fall. We continued looking for improvements. Then, we observed that we could evolve the algorithm to use only one sensor without causing a significant reduction in its accuracy, which indicates how capable the algorithm is to identify a real fall.

Considering our experience in the development of such fall detection solutions, including the use of thresholds and machine learning techniques, and service-oriented architecture, we present the evolution of these algorithms, along with the motivations to do this and the lessons learned in this process. We consider as the main contribution of this paper is to compile and discuss the top six lessons learned while developing fall detection solutions to facilitate future research. These lessons could guide the developers of the detection fall solutions to avoid the same problems that we faced. Then, this contribution can improve the Software Engineer area to support the e-health solutions.

This paper is divided as follows: Section II discusses related work. Section III shows the fAlert and WatchAlert applications in addition to the fall detection algorithm versions. Lessons learned are discussed in Section IV. Finally, Section V brings conclusions and future directions.

II. RELATED WORK

As mentioned in the previous section, there are several works focused on fall detection [12]. The significant public health problem generated by falls, especially in the elderly population, can justify this large number of papers. However, it is tough to find papers with lessons learned about the

 ISBN: 978-1-61208-817-4

development of fall detection solutions. In general, each paper focuses on presenting, in a particular way, its fall detection approach and the results obtained with this model. Moreover, the absence of standardization regarding sensors, algorithms, and evaluation methods makes a comparison between these works a hard task.

In our research, we find just a recent paper with lessons learned related to fall detection systems [13], but there are other works presenting fall detection applications similar to those created throughout this research [14]–[17].

The work in [13] presents challenges concerning the creation of a multimodal database for fall detection data. The data acquisition system created obtains data from infrared sensors, wearable sensors, cameras, and an ECG (electrocardiogram) monitor. The authors state that the lessons learned are fundamental to database consolidation.

Different approaches use pre-processing data from the accelerometer to select the best features for the fall detection systems [15] [17]. These studies work with threshold-based algorithms, but also use other strategies to achieve better results. In [15], the proposed system, besides the accelerometer, utilize gyroscope and magnetometer in a secondary threshold verification to achieve more accuracy. [17] uses optimization algorithms (genetic algorithms, and simulated annealing) to set thresholds.

Some researches advocate the use of machine learning algorithms to detect falls. The system proposed in [14] presents a fall detection solution that uses an accelerometer and Supported Vector Machines (SVM). The authors propose a data pre-processing, combining features to find a better combination for the SVM method. Moreover, the study confronts the machine learning algorithm proposed with a threshold-based solution, and, according to the authors, when the parameters of falling and non-falling are very close, the results of the SVM method are better than the threshold-based algorithm.

However, instead of using either approach, some authors prefer to combine both thresholds and machine learning algorithms in a fall detection solution. The work in [16] proposes a design model to create a healthcare monitoring system based on wearables. This model combines both threshold and machine learning algorithms (decision tree, SVM, and k-Nearest Neighbor) to detect a fall.

The study in [13] addresses the difficulties related to building a fall database, which is important for fall detection applications but does not address the process of developing these applications. The papers [14]–[17] present different approaches to create fall detection systems. Each approach contributes to understanding the challenges and possible solutions in the design and modeling processes of these systems. However, there are other challenges during the development process that the researches do not show.

In contrast, we believe that our work stands out for presenting lessons learned, focusing on good practices for the development of fall detection applications. In this article, we discuss issues related to the design, modeling, and development processes. Besides, we expose lessons learned for fall detection strategies based on thresholds and in machine-learning techniques.

These lessons learned come from the experience of several years in the study and development of fall detection applications. They can assist the process of developing new solutions and patterns by indicating important steps to be followed and difficulties common to the process of creating this type of application.

III. Evolution of Applications

A. fAlert

We show the first solution that we developed to detect a fall. As said in the previous section, we propose an algorithm to analyze the user acceleration with smartphone support. The behavior was analyzed with an algorithm whose input is the data collected from the accelerometer and the magnetometer.

As we saw previously, the accelerometer data allows the calculation of acceleration and the comparison with the acceleration of gravity. The collision with the floor can also be calculated with the acceleration, but for the verification to be correct in the fAlert algorithm, the time window is 1.5 seconds.

In sequence, the algorithm checks if the device is in a position of 45 degrees about the floor. If the answer to this verification is positive, the algorithm detects a fall, and it asks the user if he/she is well. He/she has 25 seconds to answer, and in negative case, fAlert sends a message to a caregiver.

We evaluated this algorithm considering only one daily activity, the user laying, and some fall scenarios. We executed the experiment 30 times with four combinations of the thresholds, changing the value of detection of a collision with the floor. As values for the thresholds, we have the threshold for free fall equal to 0.2 times of the gravity acceleration (0.2G), and the threshold to determine the collision equal to (1.5G). This setting resulted in 93% of detected falls, and 67% of the daily activities recognized.

B. WatchAlert

We developed the WatchAlert application to run in a smartwatch, collecting data from the accelerometer and the gyroscope in the first version and using only the accelerometer in the second version.

1) Architectural Aspects: Although we know the device constraints, we do not know what the impact in the application is. When we started the development of WatchAlert, we decided that the algorithm (collecting and processing) should run entirely in the smartwatches. When we started the software testing phase of this first version of the algorithm, the application would break when it collected and processed the data simultaneously.

A solution adopted by us was the code division to run the collect task in the smartwatch and processing in the smartphone connected with the smartwatch. With this division, in the flow shown in Figure 1, the only activity performed in the smartwatch is “Collect the data”.

With this problem solved, we can perform the tests in the WatchAlert application, and we also verify that the collection was not continuous, and it does not allow the need to define services to run in the smartwatch to allow the collect.

2) First Version of the Algorithm: We defined an algorithm based on [2], and we used the data from the accelerometer and gyroscope, combining them to detect a fall. To do this, the steps performed in the algorithm verify initially if the user suffers a free fall from the analysis of the three axes of the accelerometer according to (1). If the values obtained with the calculation is lower than 5,3936, the first signal of a fall is detected.

 ISBN: 978-1-61208-817-4

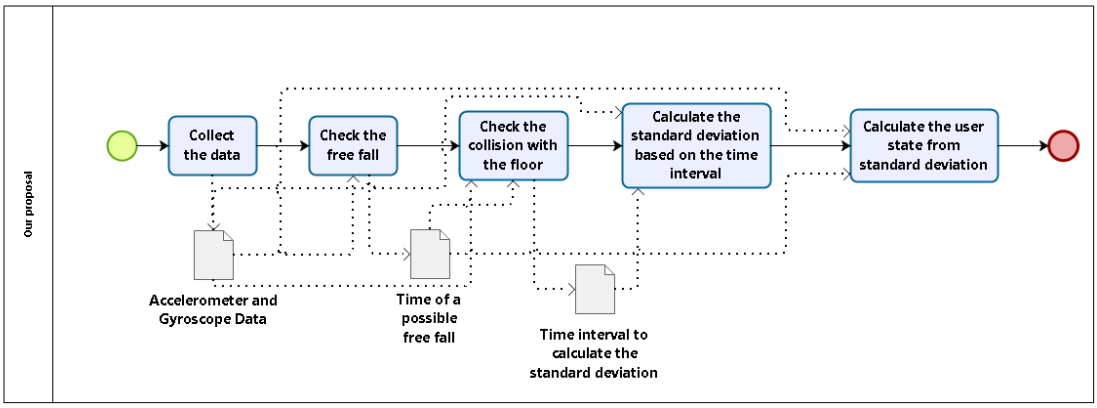

Figure 1. Flow of the second version of WatchAlert algorithm.

$$VA(t_i) = \sqrt{{A_x}^2(t_i) + {A_y}^2(t_i) + {A_z}^2(t_i)} \quad (1)$$

In sequence, the algorithm checks the user impact on the floor looking for another evidence, also using (1), and the result should be higher than 23,5359. This procedure is verified considering the interval of 0.4 seconds after the free fall. The third step of the algorithm, different from the previous steps, uses the data from the gyroscope to determine the movement performed by the user's arm. The gyroscope data contains three axes like the accelerometer, but the difference is that it calculates the angle of rotation. If the user moves the arm faster than 250°/s in the interval between the free fall and the impact, we have another sign of a fall. Equation (2) allows us to know the angle of movement.

$$GS(t_i) = \sqrt{{G_y}^2(t_i) + {G_z}^2(t_i)} \quad (2)$$

Following the steps, we return to use the accelerometer data to continue finding fall evidence. At this moment, the algorithm calculates the standard deviation from the free fall until the collision with the floor to verify if the resultant value is similar to the standard deviation of a fall previously defined.

The last step that allows us to diagnose a fall is the user's state after a possible fall. We should consider three scenarios: i) user is active; ii) user is inactive; and iii) user is partially active. In the first situation, we have the user performing a movement as clapping or jumping, and the accelerometer behavior is like a new impact. If the user is inactive, he/she falls and is immobile, needing fast support. Moreover, the last one, the user maintains a movement, but differently from the active state, he/she makes less abrupt movements.

To calculate this, we compare the result of the sum of the acceleration determined by (3) for 1.5 seconds, starting in the moment of the impact verified in the second step. If the result is greater than 1274,86, the algorithm identifies a fall.

$$SA_i = \sum_{j=1}^{N} [|A_x(j)| + |A_y(j)| + |A_z(j)|] \quad (3)$$

3) Second Version of the Algorithm: We observed the algorithm described in Section III-B2 and performed an improvement to use only the accelerometer data from smartwatches. In comparison with the previous version of the algorithm, we made two main changes: i) remove the gyroscope step; and ii) changes in the equation of the last step to identify the fall.

The first change is motivated by the evolution of our algorithm. About the second main difference, we performed it based on the analysis during tests of the first version because raw data can be too imprecise. To improve this, we use the standard deviation considering the same time interval, 1.5 seconds after the possible collision with the floor.

As a consequence of all these improvements, we changed thresholds used to determine a fall. We performed an experiment to detect these new values with data collected (data available in: https://bit.ly/31wNiiQ) using the LG Urbane smartwatch and performed with five persons, whose profile is described in Table I, and we found: i) in the first step, was 6.864655. ; ii) in the second step, was 29.41995; iii) in the third step, was 9.80665; and iv) we change to 4.903325 in the fourth step.

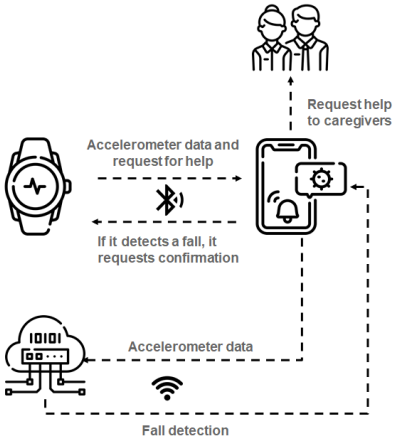

Figure 2. Overview of the third version of WatchAlert.

C. Machine Learning Algorithm

Both algorithms previously presented use thresholds-based fall identification strategy. This strategy verifies the combination of a set of measurements that, when exceeding certain thresholds, can characterize a possible fall event. Another

TABLE I. PROFILE OF PARTICIPANTS DURING DATA COLLECTION.

	P1	P2	P3	P4	P5
Gender	M	M	M	M	F
Age	28	30	23	20	20
Height	1.78m	1.72m	1.73m	1.83m	1.64m
Weight	70Kg	90kg	90Kg	82Kg	55Kg

strategy that can be used to detect falls is the usage of supervised machine learning algorithms. We have used this new approach to create the third version of WatchAlert. Figure 2 shows an overview of this version.

We evaluate the machine learning algorithms and obtained more accuracy than threshold algorithms. However, it is important to highlight that the usage of machine learning requires a higher processing power than the strategy with thresholds. This processing power can be unfeasible for some smartphones, and we used the cloud to execute this algorithm. Considering the message exchange, total data processing time, and latency, we achieved a low response time to trigger the alarm, using a simple WiFi internet connection and 4G. We believe that response time is within an acceptable margin. Below, we detail the selection and evaluation of the features and the machine learning algorithm.

It is worth mentioning that in this third version of WatchAlert, if the application is offline, it not stop working. In this case, WatchAlert behaves as in its second version, using the threshold approach that is less expensive for the mobile device and also allows to identify possible falls with slightly less accuracy than the Machine Learning approach.

1) Feature Selection: One of the first activities to create a new machine learning model is to select the features that can be used to characterize the relevant events. These features are values based on treatments or calculations made on the raw data, which we obtained from the smartwatch accelerometer.

TABLE II. FEATURES EQUATIONS.

Measure	Equation
Mean	$(V_1 + V_2 + V_3 + ...V_{n-1} + V_n)/n$
Minimum (MIN)	$Min(V_1, V_2, V_3, ...V_{n-1}, V_n)$
Maximum (MAX)	$Max(V_1, V_2, V_3, ...V_{n-1}, V_n)$
Standard Deviation (STD)	$\sqrt{\frac{\sum_{i=1}^{n}(V_i - Mean)^2}{n}}$
RMS	$\sqrt{\frac{1}{n}\left(V_1^2 + V_2^2 + V_3^2 + ...V_{n-1}^2 + V_n^2\right)}$
Kurtosis	$\frac{\sum_{i=1}^{n}(V_1 - Mean)^4}{\left(\sum_{i=1}^{n}(V_1 - Mean)^2\right)^2} - 3$

As shown before, the (1) is used to combine the three axes values of the accelerometer. Based on resulting value $VA(t_i)$ for each measurements set during a given period, we calculate the following features: mean, maximum, minimum, standard deviation, RMS, and Kurtosis. Table II presents the equations referring to the calculation each one of these features. In this table, V_i is a value of (1) for measurement i, and n is the measurement number.

These six features are used to represent the classes of events that we want to detect (fall and no-fall). The choice of these features was made based on the literature, since they are widely used by several other surveys of fall detection solutions, such as [18]. We emphasize that the evaluation process of these features is essential to choose only the features that collaborate with the result. Below, we present the machine learning algorithms examined and the feature analysis.

2) Algorithms and Feature Evaluation: First, we performed a correlation analysis among features to remove those that were directly correlated. We understand that this correlation depends on the data, but keep this behaviour among the features can reduce the performance of the machine learning classifiers. Thus, first it was applied the Shapiro-Wilk normality test [19]. At the 0.05 significance level, the data was not significantly drawn from a normally distributed population. Because our data have just ordinal fields and it is not normally distributed, we decided to use the Spearman correlation [20]. Table III presents the results obtained using the OriginPro Data Analysis and Graphing Software v9.1 [21]. Considering these results, we noticed a high correction between RMS and STD (0.76955), RMS and MEAN (0.88088), RMS, and MAX (0.70221), MAX and STD (0.87859). Hence, we decided to remove the Standard Deviation (STD) and Mean (MEAN) features of this evaluation, keeping the Maximum (MAX) because it is very relevant to the previously presented threshold algorithm.

TABLE III. SPEARMAN CORRELATION.

	STD	MEAN	MAX	MIN	RMS	Kurtosis
STD	1	0.4575	0.87859	-0.74941	0.76955	-0.31087
MEAN	0.4575	1	0.4587	-0.00775	0.88088	-0.33049
MAX	0.87859	0.4587	1	-0.63796	0.70221	0.0596
MIN	-0.74941	-0.00775	-0.63796	1	-0.3261	0.07504
RMS	0.76955	0.88088	0.70221	-0.3261	1	-0.40499
Kurtosis	-0.31087	-0.33049	0.0596	0.07504	-0.40499	1

After feature selection, it was possible to evaluate the performance of two algorithms: Support Vector Machines (SVM) [22], and Random Forest [23]. We selected these algorithms because some previous tests showed that they present better results when compared with other algorithms such as MLP, J48, Random Tree, and Naive Bayes. Furthermore, as many works in the literature have already analyzed many classifiers for the fall detection problem [24], we have decided to deepen this analysis considering only these two algorithms. Regarding SVM, we used three different kernels: Linear, Polynomial, and Gaussian.

These algorithms - Random Forest and SVM - were executed considering classifiers with attribute selection and with attribute selection combined to cost-sensitive learning [25]. For the Random Forest attribute selection, we used Information Gain. SVM attribute selection was performed with SVM Attribute Evaluation [26]. The cost matrix $[[0, 5], [1, 0]]$

 ISBN: 978-1-61208-817-4

used in the Cost-Sensitive Classifiers penalized five times more the false negatives, *i.e.*, cases in which a fall occurred and the classifier labeled no-fall. Considering the negative impact caused by falls, failure to detect a fall is extremely dangerous.

Regarding datasets, we used an original dataset with 359 no-fall instances and 67 fall instances. We understand that this number of instances represents a small dataset, but it is important to highlight the difficulty of getting large and real data related to elderly falls. Thus, we argue that this dataset can provide evidence about the feasibility of our approach. So, considering feature correction analysis, we have created a similar dataset, but without standard deviation (STD) and mean (MEAN) features. Finally, due to the class unbalancing, we have created a third version of the dataset applying the Synthetic Minority Oversampling TEchnique (SMOTE) [27]. This last dataset has 359 no-fall instances and 268 fall instances. All experiments were performed on the WEKA data mining software tool [28], and each classifier was executed 30 times.

Table IV presents the results considering accuracy, precision, recall, true positive rate (TPR), false-negative rate (FNR), false-positive rate (FPR), true-negative rate (TNR), elapsed time training (in seconds), and elapsed time testing (in seconds). We were seeking for a classifier with high accuracy, high recall, and low false-positive rate, because it is crucial to classify data correctly (effect analyzed by accuracy), avoiding the non-identification of a true fall (effect analyzed by recall), and reducing the number of false fall events alarms (effect analyzed by false-positive rate). The Equations (4), (5) and (6) present how these metrics are calculated [29].

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{4}$$

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

The best result was obtained using a Random Forest algorithm with Information Gain attribute selection, and considering the original dataset with SMOTE. We got 95.13% of accuracy, 95.13% of recall and 5.20% of false-positive rate. With these results, it is important to highlight that still remains a gap for future studies regarding the improvement of accuracy and recall, and the reduction of false-positive rate. In a real context, with a lot of no-fall events, 5.20% of fall alerts in regular situations can reduce users' adherence to this kind of monitoring. As in this work, we used only the default algorithm parameters, these results can be improved considering a fine-tuning of the parameters.

To conclude our analysis, we also checked the average time spent by each classifier in training and testing. We observed that the Random Forest time for both training and testing activities was not the best. However, the values did not differ much from the best results. In this case, the Random Forest with Information Gain Attribute Selection takes 0.17 seconds for training and 0.00175 seconds for testing. Because of this evaluation, we decided to use a Random Forest classifier in the WatchAlert third version.

IV. Lessons Learned

We describe in this section the lessons learned during the development of the applications fAlert and Watchalert in three versions of algorithms and the features used in them. During the development of the applications, we tested the algorithms in two ways: i) offline, which is when we implement the algorithm in a device not used to collect and detect a fall; ii) online, which is the use of the algorithm in the real device to monitor the user.

The first way, offline mode, is adequate when we want to know the number of falls and daily activities correctly identified based on the data that we collected using smartwatches. However, we can know this information, but we need to know how the algorithm runs in a real device and the consequences of its use. Furthermore, with our data analysis and considering both ways, we learned six lessons described hereafter. These lessons are described with a generic problem that we face, followed by the experienced situation and the lesson learned during our work.

LL01: Use Flow-based Programming

- **Problem:** Considering the sending of batch data, one problem occurs when the free-fall occurs in a batch, and the impact with the floor and the user state are in another. Then, the fall detection behavior can be compromised and the emergence service or caregivers cannot be contacted. How to deal with this situation?
- **Experience:** The problem was experienced during the development of the first version of WatchAlert. The accelerometer data from smartwatch was collected and sent in batch to smartphone, and several times the fall detection results was a daily activity, generating the false-negative response. In our studies to detect the cause of the problem, we identified that the batch data compromised the results, because the free fall was in different batch of the floor impact or the user state. Then, we search for the best way to correct the problem, and the solution found was the Flow-based Programming.
- **Lesson:** Considering our experience, we suggest the developers of the fall detection solutions use flow-based programming because it allows the sequenced data to avoid the improvement of the false-negative rate. This strategy is especially important in these situations because the fall detection solution deal with real-time data and needs of the precise and correct result to avoid the severe sequelae with the user.

LL02: Use Service-Oriented Architecture (SOA)

- **Problem:** As each device perform several actions, for instance, the smartwatch collects the data and interact with the user to receive the answer of the real status or the smartphone, which gets the data, process it and sends the message to caregivers or emergence service, it is important to deal with this. How can the developers of the fall detection solutions maintain several functionalities without one impact in another?
- **Experience:** In the first version of WatchAlert, we tried to run the code in sequence, according to the fall detection steps. However, there is a problem when the smartwatch continues collecting the data and wait for the smartphone response; then, the fall detection

TABLE IV. TABLE WITH ACCURACY AND FALSE-NEGATIVE RATE OF EVALUATED CLASSIFIERS.

Dataset	Classifier	Classifier Detail	Accuracy	Precision	Recall	TPR	FNR	FPR	TNR	Time Training (s)	Time Testing (s)
Original	Attribute Selected Classifier	RandomForest (InfoGain)	93.85	93.94	93.85	93.85	6.15	25.09	74.91	0.11	0.00093
		SVM (SVMAttributeEval) - Linear Kernel	93.46	93.70	93.46	93.46	6.54	20.24	79.76	0.20	0.00076
		SVM (SVMAttributeEval) - RBF Kernel	91.27	91.21	91.27	91.27	8.73	41.42	58.58	0.06	0.00155
		SVM (SVMAttributeEval) - Polynomial Kernel	87.21	90.33	87.21	87.21	12.79	23.06	76.94	53.49	0.00064
	Cost Sensitive Classifier	AttributeSelected (InfoGain): RandomForest	93.43	93.73	93.43	93.43	6.57	19.09	80.91	0.10	0.00078
		AttributeSelected (SVMAttributeEval): SVM - Linear Kernel	88.83	91.34	88.83	88.83	11.17	15.76	84.24	0.41	0.00063
		SVM - Linear Kernel	88.64	91.28	88.64	88.64	11.36	15.65	84.35	0.43	0.00203
		SVM - Polynomial Kernel	83.10	88.71	83.10	83.10	16.90	20.75	79.25	53.59	0.00141
		SVM - RBF Kernel	87.77	88.58	87.77	87.77	12.23	30.90	69.10	0.02	0.00173
Original - (STD, MEAN)	Attribute Selected Classifier	RandomForest (InfoGain)	93.62	93.69	93.62	93.62	6.38	25.30	74.70	0.10	0.00093
		SVM (SVMAttributeEval) - Linear Kernel	93.60	93.86	93.60	93.60	6.40	19.96	80.04	0.21	0.00096
		SVM (SVMAttributeEval) - RBF Kernel	89.86	90.05	89.86	89.86	10.14	49.71	50.29	0.05	0.00184
		SVM (SVMAttributeEval) - Polynomial Kernel	80.23	86.47	80.23	80.23	19.77	30.09	69.91	53.51	0.00047
	Cost Sensitive Classifier	AttributeSelected (InfoGain): RandomForest	92.49	93.05	92.49	92.49	7.51	19.03	80.97	0.10	0.00047
		AttributeSelected (SVMAttributeEval): SVM - Linear Kernel	89.06	91.55	89.06	89.06	10.94	15.57	84.43	0.34	0.00143
		SVM - Linear Kernel	88.92	91.47	88.92	88.92	11.08	15.48	84.52	0.42	0.00126
		SVM - Polynomial Kernel	64.50	82.20	64.50	64.50	35.50	29.73	70.27	53.43	0.00124
		SVM - RBF Kernel	87.28	87.51	87.28	87.28	12.72	36.57	63.43	0.02	0.00156
Original + SMOTE	Attribute Selected Classifier	RandomForest (InfoGain)	95.13	95.27	95.13	95.13	4.87	5.20	94.80	0.17	0.00175
		SVM (SVMAttributeEval) - Linear Kernel	88.47	88.72	88.47	88.47	11.53	12.35	87.65	0.31	0.00110
		SVM (SVMAttributeEval) - RBF Kernel	92.09	92.33	92.09	92.09	7.91	8.19	91.81	0.06	0.00277
		SVM (SVMAttributeEval) - Polynomial Kernel	88.49	88.98	88.49	88.49	11.51	11.81	88.19	36.29	0.00189
	Cost Sensitive Classifier	AttributeSelected (InfoGain): RandomForest	93.11	93.65	93.11	93.11	6.89	5.97	94.03	0.15	0.00064
		AttributeSelected (SVMAttributeEval): SVM - Linear Kernel	76.48	81.66	76.48	76.48	23.52	19.45	80.55	0.33	0.00139
		SVM - Linear Kernel	76.40	81.54	76.40	76.40	23.60	19.56	80.44	0.29	0.00295
		SVM - Polynomial Kernel	87.52	88.60	87.52	87.52	12.48	11.45	88.55	35.51	0.00109
		SVM - RBF Kernel	89.81	91.26	89.81	89.81	10.19	8.43	91.57	0.03	0.00281

was impacted, leading to a false-negative situation. To solve this, we tried to put services in the smartwatch, even with hardware constraints, and it works. The services were to collect data, communicate with the user, and connect with the smartphone. We perform the same with a smartphone to avoid any problem like a smartwatch, and the WatchAlert works well.

- **Lesson:** The lesson learned was that we should develop an application using Service-Oriented Architecture [30], dividing each functionality in service. For instance, we can run the data collection and check the user's health in the smartwatch. This avoids the increase of the false-negative rates, turning the application more accurate.

LL03: Divide the code in different devices

- **Problem:** Wearables devices have constraint hardware, which allows the use of a few services and algorithms. Then, these devices cannot collect and process the data and call the user's caregivers. How we deal with this situation without to impact on the result?
- **Experience:** We note a problem with functionalities and the accuracy of the algorithms when we use only the smartwatch. As we deal with an IoT solution, we consider dividing the algorithms in different devices of the way that the smartwatch collects the data, sends them to the smartphone, which goal is to process it, and returns the result to the smartwatch. Furthermore, we also divide the code between the smartphone and Cloud to improve the detection and use of Machine Learning algorithms. Then, the problem was solved, and we do not suffer any impact with the time of fall detection.
- **Lesson:** The lesson learned with the experienced situation was the division of the code between different devices according to each hardware configuration. This strategy is possible due to the IoT solutions, which are composed of several devices and sensors with different configurations. Then, with this lesson, the developers of fall detection applications can better use the devices and do not suffer any impact on the accuracy of the algorithm neither in any other important service of the application.

LL04: Use offline methods of fall detection integrated with ML algorithms

- **Problem:** Health applications are critical, and the information should always be available. However, if the application uses only algorithms in the cloud, the solution can have a problem when the connectivity with the Internet and the service is inaccessible. Then, how can we assure the high availability of the service and a self-adaptive detection model?
- **Experience:** We tried to use a cloud-based solution to execute our ML approach, but this strategy is affected by network problems. Considering the criticality of this type of application, it is also essential to use an offline method to ensure service availability even with network problems. This way, we can guarantee a seamless service that evolves using new data even without internet connectivity. As smartwatches are frequently associated with a smartphone, the connection is Bluetooth, and it continues even with a problem with an Internet connection, which allows us to maintain the service.

 ISBN: 978-1-61208-817-4

- **Lesson:** If you want to use ML algorithms for fall detection, it should be in a cloud, and it is crucial to have an alternative method, until threshold algorithm, that does not depend on Internet access. Thus, we suggest putting at least another algorithm in the smartphone to run in the situations of Internet problems. Therefore, it is possible to ensure high service availability even that the application lost the internet connection.

LL05: Analysis and Selection of the best features for the ML algorithm

- **Problem:** When using machine learning for fall detection, we can use several features and not all features contribute to improve the result. Moreover, the greater the number of features may consequently make the algorithm processing slower. Then, how can we select better features for the machine learning algorithm?
- **Experience:** We use a set of features as an input of ML algorithms as root mean square, mean, max, and min values. However, we note that the results could be improved. In the first step to detect the best improvement, we analyzed the correlation between features used, which showed us the need to change the features selected and demonstrated in the Section about ML algorithms. Consequently, when we execute the ML algorithms, the results were improved, and the processing time was shorter.
- **Lesson:** For best results with the ML algorithms, the features used must be selected using specific techniques for this analysis considering the application scenario. We recommend using correlation analysis and attribute analysis algorithms, such as Information Gain and SVM Attribute Evaluation. In the fall detection scenario, we suggest using the maximum, minimum values, RMS, and kurtosis, when use only accelerometer data.

LL06: Choose the most suitable ML algorithm

- **Problem:** Many times, we see works that use ML algorithms for fall detection but do not explain the chosen. However, some factors impact the algorithm results, e.g., the feature type, the amount of data, and how this data is organized. Then, what algorithm should we choose?
- **Experience:** Firstly, we choose the ML algorithms for fall detection apps based on literature, but we decide to test other ML algorithms when we change the features. To our surprise, the results were different from the literature knowledge and the tree-based algorithms, like Random Forest, which is not most used as the best algorithm for detection fall scenario considering the accelerometer data that we have. Then, for each situation, several algorithms should be tested.
- **Lesson:** Choose the ML algorithm for your fall detection applications based on the data available. In the fall detection scenario, we suggest at least evaluate variations of SVM algorithm and tree-based algorithms when work with only the accelerometer data.

V. CONCLUSION

This paper discusses issues related to the development of mobile fall detection applications. We presented two applications and three algorithms developed by the GREat research group. Throughout this process, we have built up a good experience on this topic. Thus, it was possible to describe the six main lessons learned about fall detection by smartphones and smartwatches.

In short, we observed that the usage of flow-based programming and Service-Oriented Architecture (SOA) could assist in the development of fall detection applications due to a significant amount of data that needs to be processed and to keep the data capture service independent of handling them. Also, due to the smartwatches hardware restrictions, it is crucial to delegate to the smartphone the processing of data. Finally, for cases that involve cloud-based machine learning approaches, it is essential to have an alternative method that does not depend on Internet access.

In the scenario of fall detection using accelerometer data, we observe that maximum and minimum values, RMS and kurtosis are the most important features. In sequence, on the contrary to the literature, we identified that the Random Forest is better than SVM considering the machine learning algorithms found in the literature.

Thus, we believe that by following these guidelines, it is possible to create a robust, seamless, and easy-to-maintain fall detection service using sensors present on smartphones or smartwatches. As future work, we intend to evaluate the battery and latency, and comparing the thresholds and the machine learning algorithms.

ACKNOWLEDGMENT

The authors would like to thank CNPQ for the Productivity Scholarship of Rossana Maria de Castro Andrade DT-2 (N^o 315543 / 2018-3) and Coordination of Improvement of Higher Level Personnel - Brazil (CAPES) that provided to the Evilasio Costa Junior a Ph.D. scholarship.

REFERENCES

[1] World Health Organization, "Global report on falls prevention in older age," https://www.who.int/ageing/publications/Falls_prevention7March.pdf, 2007, [retrieved: October, 2017].

[2] S.-L. Hsieh, C.-C. Chen, S.-H. Wu, and T.-W. Yue, "A wrist-worn fall detection system using accelerometers and gyroscopes," in Networking, Sensing and Control (ICNSC), 2014 IEEE 11th International Conference on. IEEE, 2014, pp. 518–523.

[3] P. Kostopoulos, T. Nunes, K. Salvi, M. Deriaz, and J. Torrent, "F2d: A fall detection system tested with real data from daily life of elderly people," in 2015 17th International Conference on E-health Networking, Application Services (HealthCom), Oct 2015, pp. 397–403.

[4] L. S. Piva, A. B. Ferreira, R. B. Braga, and R. Andrade, "falert: An android system for monitoring falls in people with special care," in Workshop on Tools and Applications of the 20th Brazilian Symposium on Multimedia and Web Systems, 2014.

[5] R. L. Almeida, A. A. Macedo, Í. L. de Araújo, P. A. Aguilar, and R. M. Andrade, "Watchalert: An evolution of the falert app for detecting falls on smartwatches," in Extended Proceedings of the XXII Brazilian Symposium on Multimedia and Web Systems. SBC, 2016, pp. 124–127.

[6] R. Cucchiara, A. Prati, and R. Vezzani, "A multi-camera vision system for fall detection and alarm generation," Expert Systems, vol. 24, no. 5, 2007, pp. 334–345.

[7] I. L. de Araújo, L. Dourado, L. Fernandes, R. M. d. C. Andrade, and P. A. C. Aguilar, "An algorithm for fall detection using data from smartwatch," in 2018 13th Annual Conference on System of Systems Engineering (SoSE), June 2018, pp. 124–131.

[8] A. Ramachandran, A. R., P. Pahwa, and A. K. R., "Machine learning-based techniques for fall detection in geriatric healthcare systems," in 2018 9th International Conference on Information Technology in Medicine and Education (ITME), Oct 2018, pp. 232–237.

 ISBN: 978-1-61208-817-4

[9] D. Ajerla, S. Mahfuz, F. Zulkernine, and R. C. Pryss, "A real-time patient monitoring framework for fall detection," Wireless Communications and Mobile Computing, vol. 2019, jan 2019, p. 13.

[10] F. Hussain, F. Hussain, M. Ehatisham-ul Haq, and M. A. Azam, "Activity-aware fall detection and recognition based on wearable sensors," IEEE Sensors Journal, vol. 19, no. 12, 2019, pp. 4528–4536.

[11] R. Jansi and R. Amutha, "Detection of fall for the elderly in an indoor environment using a tri-axial accelerometer and kinect depth data," Multidimensional Systems and Signal Processing, 2020, pp. 1–19.

[12] N. Noury et al., "Fall detection-principles and methods," in 2007 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE, 2007, pp. 1663–1666.

[13] C. J. Peñafort-Asturiano, N. Santiago, J. P. Núñez-Martínez, H. Ponce, and L. Martínez-Villaseñor, "Challenges in data acquisition systems: lessons learned from fall detection to nanosensors," in 2018 Nanotechnology for Instrumentation and Measurement (NANOfIM). IEEE, 2018, pp. 1–8.

[14] S.-H. Liu and W.-C. Cheng, "Fall detection with the support vector machine during scripted and continuous unscripted activities," Sensors, vol. 12, no. 9, 2012, pp. 12 301–12 316.

[15] B. Andò, S. Baglio, C. O. Lombardo, and V. Marletta, "A multisensor data-fusion approach for adl and fall classification," IEEE Transactions on Instrumentation and Measurement, vol. 65, no. 9, 2016, pp. 1960–1967.

[16] D. Dziak, B. Jachimczyk, and W. Kulesza, "Iot-based information system for healthcare application: design methodology approach," Applied Sciences, vol. 7, no. 6, 2017, p. 596.

[17] S. Khojasteh, J. Villar, C. Chira, V. González, and E. De La Cal, "Improving fall detection using an on-wrist wearable accelerometer," Sensors, vol. 18, no. 5, 2018, p. 1350.

[18] N. Pannurat, S. Thiemjarus, and E. Nantajeewarawat, "Automatic fall monitoring: a review," Sensors, vol. 14, no. 7, 2014, pp. 12 900–12 936.

[19] A. Ghasemi and S. Zahediasl, "Normality tests for statistical analysis: a guide for non-statisticians," International journal of endocrinology and metabolism, vol. 10, no. 2, 2012, p. 486.

[20] C. Wissler, "The spearman correlation formula," Science, vol. 22, no. 558, 1905, pp. 309–311.

[21] OriginLab, "Origin: Data analysis and graphing software," https://www.originlab.com/Origin, October 2020, (Accessed on 10/23/2020).

[22] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," IEEE Intelligent Systems and their applications, vol. 13, no. 4, 1998, pp. 18–28.

[23] L. Breiman, "Random forests," Machine learning, vol. 45, no. 1, 2001, pp. 5–32.

[24] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" The Journal of Machine Learning Research, vol. 15, no. 1, 2014, pp. 3133–3181.

[25] N. Thai-Nghe, Z. Gantner, and L. Schmidt-Thieme, "Cost-sensitive learning methods for imbalanced data," in The 2010 International joint conference on neural networks (IJCNN). IEEE, 2010, pp. 1–8.

[26] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," Machine Learning, vol. 46, 2002, pp. 389–422.

[27] N. V. C. et. al., "Synthetic minority over-sampling technique," Journal of Artificial Intelligence Research, vol. 16, 2002, pp. 321–357.

[28] M. Hall et al., "The weka data mining software: an update," ACM SIGKDD explorations newsletter, vol. 11, no. 1, 2009, pp. 10–18.

[29] C.-Y. Hsieh, K.-C. Liu, C.-N. Huang, W.-C. Chu, and C.-T. Chan, "Novel hierarchical fall detection algorithm using a multiphase fall model," Sensors, vol. 17, no. 2, 2017, p. 307.

[30] N. M. Josuttis, SOA in practice: the art of distributed system design. " O'Reilly Media, Inc.", 2007.

 ISBN: 978-1-61208-817-4