Fall Detection using Smartwatch Sensor Data with Accessor Architecture

Anne Ngu[3], Yeahuay Wu[1], Habil Zare[3], Andrew Polican[2], Brock Yarbrough[3], and Lina Yao[4]

[1] Department of Computer Science, Temple University
{tuf02038}@temple.edu
[2] Department of Computer Science, Arizona State University
{adpolican}@gmail.com
[3] Department of Computer Science, Texas State University
{angu,zare, bcy3}@txtate.edu
[4] School of Computer Science and Engineering, University of New South Wales
{lina.yao}@unsw.edu.au

Abstract. This paper proposes using a commodity-based smartwatch paired with a smartphone for developing a fall detection IoT application which is non-invasive and privacy preserving. The majority of current fall detection applications require specially designed hardware and software which make them expensive and inaccessible to the general public. We demonstrated that by collecting accelerometer data from a smartwatch and processing those data in a paired smartphone, it is possible to reliability detect (93.8% accuracy) whether a person has encountered a fall in real-time. By wearing a smartwatch as a piece of jewelry, the well-being of a person can be monitored in real-time at anytime and anywhere as contrasted to being confined in a particular facility installed with special sensors and cameras. Using simulated fall data acquired from volunteers, we trained a fall detection model off-line that can be composed with a data collection accessor to continuously analyze accelerometer data gathered from a smartwatch to detect minor or serious fall at anytime and anywhere. The accessor-based architecture allows easy composition of the fall-detection IoT application tailored to heterogeneity of devices and variation of user's need.

1 Introduction

Internet of Things (IoT) is a domain that represents the next most exciting technological revolution since the Internet [2]. IoT will bring endless opportunities and impact every corner of our planet. In the healthcare domain, IoT promises to bring personalized health tracking and monitoring ever closer to the consumers. This phenomena is evidenced in a recent Wall Street Journal (June, 29, 2015) article entitled "Staying Connected is Crucial to Staying Healthy". IoT applications represents a new trend of softwares that involve interaction with everyday internet connected physical objects. Previous work in IoT fall detection applications required specialized hardware and software. This translated to

buying an expensive vendor-specific device and creating a new native application for each type of device, which is not scalable. Moreover, the privacy aspect of data collection is not being addressed in the sense that data is being automatically collected and transmitted to the vendor's designated server without user's permission or input. Our fall detection IoT application is created using accessor-based architecture [1] which is based on Javascript and is designed to be device agnostics.

The laborious process in creating IoT application is the design of a set of experiments to collect and label data reliability and pre-processing of the data to gain an intuition of the most significant features. For our fall detection application, we recruited six volunteers and set up a robust methodology to systematically collect and label fall data. We used the support vector machine (SVM) algorithm to train a fall detection model using mainly resultant acceleration data over a sliding window. Our model has 93.8% precision and 97.2% recall. We showed how a model trained off-line could be easily integrated with a data collection accessor running on an Android compatible phone that supports Bluetooth Low Energy (BLE) communication protocol and used to detect falls in real-time.

By wearing a smartwatch as a piece of jewelry, the well being of an elderly person can be monitored in real-time at anytime and anywhere. The application can be set up to compose with other accessors such that customized functionalities can be incorporated, e.g., further confirmation from the user, sending a text message to a trusted family member or friend, or calling 911 in the event of a fall being detected. Currently, there are no known fall-detection IoT applications that leverage data from commodity-based wearable devices to monitor the well-being of patients non-invasively, in real-time, and with privacy preserving. The later refers to the fact that personal daily activities data of the elderly person can be stored locally or archived to a secure storage of choice by the user. The main contributions of the paper are:

- An Android IoT platform that supports composition of IoT applications using the accessor design pattern that is reusable across heterogeneous devices.
- A fall detection model that leverages a commodity-based smartwatch and smartphone which provides full mobility and full control of collected data for the users.
- A methodology for collecting simulated fall data from volunteers and annotating them for model training.

The remainder of this paper is organized as follows. In Section 2, we present the current work on daily activities detection, emphasize on research work that specifically address the fall detection and also briefly discuss the need for adopting accessor framework for building this style of IoT application. In Section 3, we provide a detailed description of the accessor framework and the implementation of an Android accessor host. In Section 4, we outline the methodology we used to collect training data for fall detection. In Section 5, we discuss the generation and the evaluation of the model and finally in Section 6, we present our conclusion and future work.

2 Related Works

The World Health Organization (WHO) reported that 28%-35% of people aged 65 and above fall each year. This rate increases to 32%-42% for those over 70 years of age. Thus, a great deal of research has been conducted on fall detection and prevention. The early researches in this area were concentrated on specially built hardware that a person could wear or installed in a specific facility. The fall detection devices in general try to detect a change in body orientation from upright to lying that occurs immediately after a large negative acceleration to signal a fall. However, modern smartphones and related devices now contain more sensors than ever before. Data from those devices can be collected more easily and more accurately with the increase in the computing power of those devices. Moreover, smartwatches and smartphones are becoming more pervasive in this 21st century. There is thus a dramatic increase in the research on smartphone based fall detection and prevention in the last few years. This is highlighted in the survey paper [6]. The smartphone-based fall detection solutions in general collect accelerometer, gyroscope and camera sensor data for fall detection. Among the collected sensor data, the accelerometer is the most widely used. The collected sensor data were analyzed using two broad type of algorithms. The first is the threshold-basted algorithm which is less complex and requires less computation power. The second is the machine learning based fall detection solutions. We will review both type of work below.

A threshold-based algorithm using a trunk mounted bi-axial gyroscope sensor is described in [3]. Ten young healthy male subjects performed simulated falls and the bi-axial gyroscope signals were recorded during each simulated-fall. Each subject performed three identical sets of 8 different falls. Eight elderly persons were also recruited to perform Activity of Daily Life (ADL) that could be mistaken for falls such as sitting down, standing up, walking, getting in and out of the car, lying down and standing up from bed. The paper showed that by setting three thresholds that relate to the resultant angular velocity, angular acceleration, and change in truck angle signals, a 100% specificity was obtained. However, there was no discussion on the practicality of attaching a trunk mounted sensor on a patient for a prolonged period of time. The restriction on the mobility of the patients and the privacy issue of data storage were not discussed as well.

The use of machine learning algorithms is recently presented by John Guirry in [5] for classifying ADLs with 93.45% accuracy using SVM and 94.6% accuracy using C4.5 decision trees. These ADLs include: running, walking, going up and down stairs, sitting and standing up. Their setup include a Samsung Nexus Galaxy smartphone and the Motorola Moto Actv smartwatch. Data was collected from the accelerometer, magnetometer, gyroscope, barometer, GPS, and light sensors. They synthesized a total of 21 features from all the sensors. They did not specifically address the fall detection. Our choice to use SVM as the machine learning algorithm for our fall detection was first inspired by Guirry's work on using smartwatch paired with smartphone for ADL detection.

SVM have been used for fall detection by other scholars [9]. They used a trunk-mounted tri-axial sensor (a specialized hardware) to collect data. They

were able to achieve 99.14% accuracy with four features using only high-pass and low-pass accelerometer data. They used a 0.1 second sliding window to record minimum and maximum directional acceleration in that time period for a feature. We drew inspiration from this approach as it allowed us to have temporal data within each sampling point rather than having to choose a generalized feature for the whole duration.

Jantaraprim et. al also used SVM on fall detection for the elderly people [7]. They defined the "critical phase" for a fall sequence as a sudden drop in resultant acceleration followed by an immediate increase, and ending with an increase in the maximum acceleration value for the interval. That is, $S_{\min}$ was the value of the initial decrease in acceleration, $S_{\max}$ was the value of the corresponding increase, while $\max(A_{\text{res}})$ was the value of the maximum resultant acceleration for the phase. They obtained fall detection results with a sensitivity of 91.1% and a specificity of 99.2% using the maximum peak feature. Their SVM models were trained with both the Radial Basis Function (RBF) kernel and the Linear Kernel, which achieved the same results in their study. We adopted the same set of features as them for our fall detection model.

None of the existing work addressed the ease of composition and development of IoT application which falls under the general category of ambient data collection and analytics type. This category of applications is growing rapidly especially in the healthcare domain where personalized health tracking and monitoring has become vital to improved and affordable healthcare. We built this category of IoT applications from ground up in our earlier work for prediction of Blood Alcohol Content (BAC) using smartwatch sensor data [10]. We learnt that the creation of this category of IoT applications can be done in three phases. The first phase involves data collection, the second phase deals with pre-processing of data and training of a model, and the third phase involves creating a native application with the trained model for prediction. Out of the three phases, the second phase is application specific and there is not much opportunity for a complete automation abielt we can use existing tools such as R to streamline the pre-processing of the data and analysis of significant features. Existing machine learning algorithms available via Weka or Matlabs can be leveraged for model training. The first phase, the data collection and the third phase are almost identical across the ambient data collection and analytic IoT applications and can benefit from reuse and sharing of existing codes via wrapping them as accessors [8] which can be deployed and executed on a light-weight accessor host.

3 System Architecture

Figure 1 shows the main infrastructure used for real-time analysis of fall detection using smartwatch sensor data. It consists of a smartwatch paired with a smartphone, a cloud persistence storage, and data analysis packages such as R and Weka. Figure 2 shows the overall solution expressed as accessors. The data collection accessor is running on the smartphone that has a working accessor

Fig. 1: Infrastructure for Data Collection and Analysis

host. The data collection accessor provides an abstraction over the low-level details of data collection process such as managing the various threads for reading the sensor values at specified sampling rate from the MS Band smartwatch.

The fall detection accessor predicts "FALL" or "NOT FALL" based on a model trained offline using the collected data in phase one as discussed in Section 4. An alert accessor informs the carer in the event of a fall. In fall detection, it is critical that data can be stored locally to preserve privacy and is in close proximity to the fall detection accessor for real time prediction. However, initial data analytics/training phase usually needs to be performed on a high performance server to build an accurate model by experimenting with different machine learning algorithms. There is thus a need to transfer the collected sensor data to a cloud server securely for initial analysis and this is done by the database accessor. Our long term goal is to set up a protocol where participating users' smartphones (with consent) transmit sensor information via a REST-based web service periodically. The archived sensor data can be visualized and analyzed. The archived sensor data can be aggregated and displayed on a map to examine health and lifestyles across a region. The true positive samples can be used for re-training of the model and adapt the fall detection accessor dynamically.

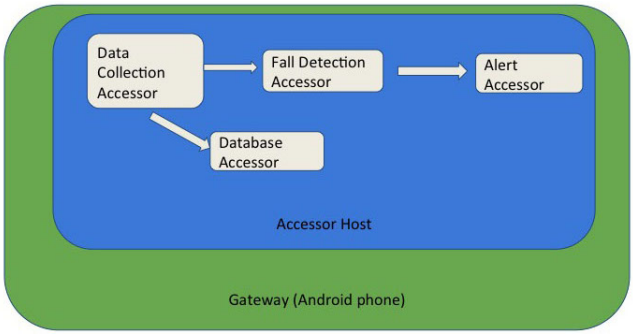

Fig. 2: Overall Solution for Fall Detection IoT Application

We could have implemented a custom native Android app for the fall detection that encompassed the functionalities provided by all the above accessors combined. This will result in a monolithic native application for data collection and prediction for each type of smartwatch device. This is equivalent to the scenario where every IoT device requires a different web browser for connection to the Internet as echoed by Zachariah et al. [11]. The accessor-based architecture provides an open, lightweight IoT development framework that serves as a bridge across a variety of IoT devices and applications.

An accessor encapsulates an IoT device such as the smartwatch. Each accessor has an interface and an implementation. The accessor interface specifies the input and output ports, the parameters and the communication protocol used for device's interaction. We implemented an Android accessor host leveraging J2V8, a Java binding for the V8 JavaScript engine. Our Android accessor host enables execution of accessors in various Android hardware and external services (Microsoft Band smartwatch) in simple script.

An accessor can run in any environment where there is a working accessor host. This is similar to the Java programming motto of "compiled once and run everywhere" paradigm. Accessors can be easily strung together in order to create an IoT application. For example, In the case of a more advanced fall detection system, the output of fall detection accessor could be routed to a text messaging accessor or a phone call accessor. The flexibility of the accessor paradigm enables various hardware and services to be interchanged with only minor adjustments.

4 Data Collection Methodology

We designed a robust methodology for collecting test data for the initial model training. Six subjects of good health were recruited. Their ages ranged from 19 to 29. We choose young healthy subjects because they are more versatile and no injuries can occur. Each subject was told to perform a pre-determined set of ADLs (Activities of daily life). Examples of ADLs that were performed include: running, walking, picking things off the ground, and throwing objects. We picked these set of activities so that it is easier to label them later. The data was collected via smartwatch through a data-collection accessor. The sensor data that we were interested in was the accelerometer. Because it was difficult to label activities for our training set based solely on the raw data we collected, we also had a stopwatch in hand to record the time stamp of each fall as it occurred.

Each ADL sequence occurred over a period ranging from 60 to 90 seconds. In some sequences, falls would be incorporated within the sequences of ADLs. Each subject was told to fall for a total of 8 times: 2 left side falls, 2 right side falls, 2 back falls, and 2 front falls. There were no restrictions placed on the subjects in regards to the number of times that they could fall within one sequence although it never exceeded two times. Subjects were also instructed to vary the intensity of the falls between each type of fall: one "soft" fall and one "hard" fall in order to capture a larger variety of falls. The hard fall was defined as a fall with faster run-up speed than the soft one. When participants in our experiment fell, their falls were padded by a twin-sized mattress in order to prevent any injuries. One issue with this method is that none of the falls collected include the "real impact" from falling directly on the ground. However, this is not critical to our labeling methodology because it is focused on identifying the "critical phase" that occurs *before* the initial impact of falling on the ground. From the raw data, we computed the values of the four features as the input to the SVM algorithm: 1) length of the acceleration vector at the time of sampling (A_{res}), 2) minimum resultant acceleration in a 750ms sliding window (S_{min}), 3) maximum resultant acceleration in a 750ms sliding window (S_{max}), and 4) the

euclidean norm of the difference between maximum and minimum acceleration in a 750ms sliding window (ΔS). More detailed descriptions for the features along with our methods for selecting them are further explained in the next section.

We then labeled each sample point as either *not falling* or *falling*. The determination of these categories was influenced by the concept of a critical phase of a fall in [7]. We defined the beginning of a fall sequence as the beginning of the critical phase and the ending of the critical phase as the occurrence of $\max(A_{res})$. This parameter, $\max(A_{res})$, is defined as the maximum resultant velocity within the critical phase. Figure 3, which is taken from [7], illustrates where the critical phase of a fall lies.

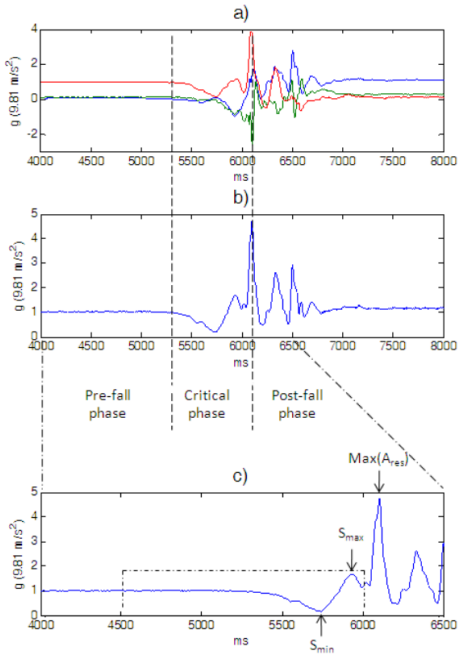

Fig. 3: "Critical phase" of a fall within an entire fall sequence, (a) Break down of the fall sequence in terms of the x, y, and z acceleration, the dotted lines denote where the critical phase occurs (b) Fall sequence in terms of resultant acceleration, once again the critical phase is within the dotted lines (c) Enlargement of the fall sequence from the beginning of the pre-fall phase up until the beginning of the post fall phase, defines where $S_{\min}$ and $S_{\max}$ are within the critical phase.

5 Training and Evaluation

Our goal is to be able to detect accurately whether someone has fallen in real time based on the motion sensed by the smartwatch that a person is wearing. We decided to use Weka, an open source Java package that covers many machine learning algorithms for training and prediction of falls. The Android accessor host supports Java binding. Both LibSVM 1.0.8 and libsvm-3.21 libraries had to be added to the accessor host in order to use the Support Vector Machine (SVM) algorithm in Weka. The total training set consisted of 1,934 samples.

We choose to use SVM because previous studies had used SVM with a tri-axial trunk-mounter sensor (a specialized hardware) with good results. We wanted to test if these results could translate to a commercial wearable device such as a smartwatch. When formulating feature selection for smartwatch, we realized that specialized trunk mounted sensors have a couple of critical advantages over wrist mounted smartwatch sensors. The main one being the ability to record data for features that relate to a subject's posture. Since the sensor is trunk-mounted, its point of data collection is from the torso of the subject, allowing the posture of the subject to be recorded. This was described in [9], where a parameter defined as Φ_z called the "posture angle" was utilized. This feature was derived by taking the angle between the vertical acceleration and the gravitational component, $|\boldsymbol{G}|$. Creating a feature like this from the smartwatch's accelerometer data is extremely difficult because there is little relation between wrist position and the overall posture. However, we noticed that by combining features from previous research in [9] and [7], it is possible to capture the balance between features that could reflect the state of the subject at the exact time of sampling, just before, and just after the sampling. This intuition captures the notion of critical phase of a fall very well. We used the principle component analysis to check the viability of this combination of features and ended up with four core features. Details of feature selection is described in the next section.

Our SVM model is trained to predict falls on a sample by sample basis, categorizing each sample taken every 250 ms as a fall or not a fall. This method does not necessarily suit the nature of the activity we are trying to detect as detecting a fall constitutes finding a pattern from a succession of points. Since this is the case, we must determine a threshold (or range) of consecutive fall predictions that would constitute a sequence of 2-5 samples as a fall. We experimented with the model in real life using actions that could be defined in two categories: (1) short term spikes in acceleration and (2) long term increase in acceleration. Actions that could be categorized as (1) are various hand and arm gestures such as waving, throwing an object, and punching. An action that would belong in category (2) would be running, which is demarcated by a sudden increase in acceleration that is maintained over a duration of time over three seconds.

We determined from experiments involving these two categories of actions that the ideal threshold for the number of consecutive predictions in one fall sequence that could be constituted as falls would be between 2 and 5. The reasoning for this is if the number of consecutive predictions is over 5, then the action would be a long term increase in acceleration such as running. In the case where it is below 2, the action would most likely be a short spike in acceleration such as arm waving. For example, if the fall sequence consisted of 8 samples and our algorithm detected only 2 of these samples, we would count all 8 samples as "correct predictions" since 2 is within our threshold. Likewise, if our algorithm only detects 6 of the 8 as falls, all the 8 samples would be counted as "wrong predictions", specifically false negatives. This occurs because 6 is over our threshold of 5. Now, if our fall sequence consisted of 8 samples and our algorithm only classified 1 of them as fall, all 8 samples would also be counted

as "wrong predictions" since 1 is below our threshold of 2. To our knowledge, no other paper on fall detection has used such a method of threshold detection based on a consecutive position prediction sample within a range.

5.1 Feature Selection

We used the Euclidean norm to measure the length (magnitude) of acceleration and velocity vectors. That is, for any vector $\boldsymbol{r}$ in $\mathbb{R}^3$, we used:

$$\|\boldsymbol{r}\|_2 = \sqrt{r_x^2 + r_y^2 + r_z^2}. \tag{1}$$

The original features that we defined were A_{res}, ΔS, $S_{\min}$, $S_{\max}$, V_{res}, and ΔV, which are defined the following.

A_{res}, resultant acceleration, is defined as the magnitude of the acceleration vector at the time of collection. Using Equation 1, we defined:

$$A_{\text{res}} = \|A\|_2. \tag{2}$$

ΔS, adapted from Liu and Cheng's paper in [9], is the magnitude of the difference between minimum and maximum acceleration in a sliding window. That is, using Equation 1, we defined:

$$\Delta S = \|S_{\max} - S_{\min}\|_2. \tag{3}$$

where $S_{\min}$ and $S_{\max}$ adapted from Jantaraprim et. al's paper [7] are defined as the minimum and maximum resultant acceleration in a sliding window of 750 milliseconds. In the original implementation by Jantaraprim et. al and Liu and Cheng's, the sliding window was designated to be 0.1 seconds.

V_{res}, the resultant velocity, is defined as the magnitude of velocity for each sample taken every 250 milliseconds, i.e.,

$$V_{res} = \|V\|_2. \tag{4}$$

Similar to ΔS, ΔV is defined as the magnitude of the difference between minimum velocity ($V_{\min}$) and maximum velocity ($V_{\max}$) for the same sliding window of 750 milliseconds, i.e.,

$$\Delta V = \|V_{\max} - V_{\min}\|_2. \tag{5}$$

We started off with six features. However, after running Principal Component Analysis (PCA), we reduced the number of features to four. We noticed that the addition of the velocity derived features: ΔV and V_{res} decreased the weights of all the features in the first two principal components to values below 0.01. With the removal of the velocity features, we found that the first two principal components contained 98% of the variance in the features. The first principal component vector, PC1, puts a lot of weight on resultant acceleration and $S_{\max}$ while the second principle component, PC2, puts a lot of weight on ΔS and resultant acceleration. In the first two components, $S_{\min}$ takes on low weights (i.e., 0.00872 and 0.1897 for PC1 and PC2 respectively).

5.2 Algorithm Testing

We tested our SVM classifier with a testing set that contained 569 samples. Since we trained it with an RBF Kernel, the weights of each of the 4 features could not be retrieved. The number of samples was chosen because it consisted of approximately 1/3 of our total data set of 1934 samples. Each sample was collected at a rate of 4Hz (every 250ms). The other 2/3 of the total data set was set aside as training data. With this data set we achieved an accuracy of 98%, specificity of 99%, a precision of 93.6%, and a recall of 80%. . The confusion matrix detailing our results is in Table 1. TP stands for True Positive, this means we predict that it is not a fall and indeed it is not a fall. FN stands for False Negative, this means we predicted that it is not a fall, but it is a fall. FP stands for False Positive, this means we predicted a fall, but it is not a fall. TN stands for True Negative, this means we predicted that there is a fall and indeed there is a fall. We also ran a lazy learning algorithm, k-Nearest Neighbors

Method	SVM		KNN		
Prediction	Not fall	Fall	Not fall	Fall	Total
Actual not fall	511 (TP)	3 (FN)	509 (TP)	5 (FN)	514
Actual fall	11 (FP)	44 (TN)	17 (FP)	38 (TN)	55
Total	522	47	526	43	559

Table 1: Confusion matrices for SVM with RBF kernel and KNN with rectangular kernel and $k = 4$. A sample is considered to be true positive (TP) if it is correctly predicted to be"not fall".

(KNN), in order to better gauge the performance of our SVM. The KNN was ran with a rectangular kernel with k = 4. The distance parameter and the kernel were adjusted to the get result with the highest sensitivity and specificity. The algorithm achieved an accuracy of 96.13%, a specificity of 99%, a precision of 88%, and a recall of 69%. The confusion matrix detailing our results is in Table 1. Most of the mislabeled samples were the same between KNN and SVM, with KNN having more false negatives.

The following table compares the performance of KNN and SVM. SVM performs better in all the classification metrics that we accounted for. The SVM model takes around six seconds to make a prediction. This is due to the fact that the conversion of accelerometer data to the four features we used in the model incurs some overhead.

6 Conclusions

A custom development of a native application for each type of IoT application running on different IoT devices is not scalable. Accessor based architecture has

Algorithm	Accuracy	Recall	Precision	Specificity
SVM	98%	80%	93.6%	99%
KNN	96.13%	69%	88%	99.2%

Table 2: Fall Detection Classification Results by Algorithm

the advantage that it is light-weight and is analogous to the Java programming environment of "compile once and run everywhere" paradigm. We demonstrated the feasibility of developing an Android accessor host (a.k.a. a virtual accessor machine) based on J2V8 engine that serves as the bridge across a variety of IoT devices and applications. This accessor host is used to prototype a real-time fall detection application that utilizes an MS Band smartwatch paired with a Google smartphone. The accessor-based fall detection application can be easily configured to run on other IoT devices such as Moto 360 smartwatch and Samsung smartphone without additional programming. This framework can be used as the foundation for developing reusable accessors for IoT applicationss. The accessor-based fall detection application can run continuously for 12 hours with 20% of battery power left intact.

We have experimented with bother eager (SVM) and lazy (KNN) machine learning techniques for fall detection, and we showed that SVM model is more reliable. While our model has an average sensitivity of 81.8%, it classifies with a high specificity and precision at values of 99% and 93.8%, respectively. In other words, our true positive rate is average while our true negative rate is high. These results show that our model has some trouble distinguishing between sudden arm gestures and actual falls. A suggestion for the future is to leverage the accelerometer sensor in the phone whenever it is possible as a trigger for the prediction. That is, fall detection should only begin when the accelerometer on the phone detects an acceleration above some threshold, assuming that the user will carry the cellphone in a certain fixed orientation (the user's pocket or a purse for women). We have developed an intuitive procedure of classifying fall data on a sequence basis rather than just a point by point basis. This procedure allowed us to extend the results of our predecessors, which used specialized sensors for activity recognition, into the realm of commercially available sensors with comparable results. An immediate future work in improving our classification is to apply dynamic Bayesian network [4], which is known to perform well for sequence data.

We acknowledge that the model is trained using data from young and healthy volunteers, which might not reflect the actual fall data from elderly people. Currently, there is no publicly available fall data of elderly people. It is impossible to collect simulated fall data from the elderly group of people because of higher likelihood of injuries. To address that, we plan to use the system to collect ADL data from elderly and verify how many of ADL activities are falsely classified as falls to fine tune our initial model. Currently, we have obtained permission to do a trial on senior citizens in a nursing home at San Marcos regarding wearing

smartwatches and carrying smartphones. We will recruit eight seniors for the trial. In particular, we want to know 1) How long seniors will wear smartwatches? 2) How much ADL activity data we can collect in a week? 3) How ADL activities affect our fall detection model ? 4) What are seniors' main concerns regarding wearing smartwatches over a long period of time? The collected activity data from our senior volunteers can be used to measure the false positive rate and gauge the practically of using resultant acceleration (A_{res}), ΔS, $S_{\max}$and $S_{\min}$ for fall prediction.

Acknowledgement

We thank the National Science Foundation (NSF) for funding the research under the Research Experiences for Undergraduates Program (CNS-1358939) and the Infrastructure grant (NSF-CRI 1305302) at Texas State University.

References

1. What are Accessors? `https://www.terraswarm.org/accessors/`
2. Internet of Things (2016), `http://https://en.wikipedia.org/wiki/Internet_of_thing`
3. Bourke, A.K., Lyons, G.M.: A threshold-based fall-detection algorithm using a bi-axial gyroscope sensor. Medical Engineering and Physics 30(1), 84–90 (2008)
4. Dagum, P., Galper, A., Horvitz, E.: Dynamic network models for forecasting. In: Proceedings of the eighth international conference on uncertainty in artificial intelligence. pp. 41–48. Morgan Kaufmann Publishers Inc. (1992)
5. Guiry, J.J., van de Ven, P., Nelson, J.: Multi-sensor fusion for enhanced contextual awareness of everyday activities with ubiquitous devices. Sensors 14(3), 5687–5701 (2014)
6. Habib, M.A., Mohktar, M.S., Kamaruzzaman, S.B., Lim, K.S., Pin, T.M., Ibrahim, F.: Smartphone-based solutions for fall detection and prevention: challenges and open issues. Sensors 14(4), 7181–7208 (2014)
7. Jantaraprim, P., Phukpattaranont, P., Limsakul, C., Wongkittisuksa, B.: Fall detection for the elderly using a support vector machine. International Journal of Soft Computing and Engineering (IJSCE) 2 (March 2012)
8. Latronico, E., Lee, E.A., Lohstroh, M., Shaver, C., Wasicek, A., Weber, M.: A vision of swarmlets. Internet Computing, IEEE 19(2), 20–28 (2015)
9. Liu, S.H., Cheng, W.C.: Fall detection with the support vector machine during scripted and continuous unscripted activities. Sensors 12(9), 12301 (2012), `http://www.mdpi.com/1424-8220/12/9/12301`
10. Mario, G., Michelle, F., Anne, N., Byron, G.: Real-time prediction of blood alcohol content using smartwatch sensor data. In: IEEE International Conference on Smart Health. Springer, Phoneix, Arizona (November 2015)
11. Zachariah, T., Klugman, M., Campbell, B., Adkins, J., Jackson, N., Dutta, P.: The internet of things has a gateway problem. In: HotMobile. ACM, Santa Fe, New Mexico, USA (February 2015)