Improving Smartwatch Based Fall Detection and Using the Accessor Design Pattern

Chris Carpenter[1], Walker Stipe[2], and Dr. Anne Ngu[1]

[1] College of Science and Engineering, Texas State University [2] Mike Cottrell College of Business, University of North Georgia

Motivations

- Falls in the elderly cause ~26,000 deaths and $34 billion in medical costs annually.
- Existing fall detection systems use expensive custom made devices.
- Can we use a light-weight accessor programming model for fall detection so that it can run with minimal battery power and accurately predict falls with only a smartwatch?

Previous Research

Our research begins where last summer's REU program left off. A Microsoft Band 2 smartwatch and an Android smartphone were used to create a fall detection application.

Problems with Performance: 44.7% True Positive
- Data frequency was too low
- Data labeling was labor intensive
- Machine learning algorithm (svm) was inaccurate

Problems with Accessor Host:
- Does not run concurrently or repeatedly.

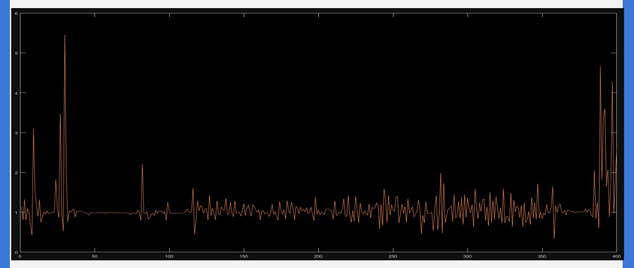
Procedure

Data Collection
- We devised a semi automated procedure to collect simulated falls from subjects ranging from 19-26 in age and all in good physical condition by modifying the data collection app to include a button that was pressed and held while a fall was being performed and released otherwise thereby labeling the data as "Fall" or "NotFall" activities.
- Participants were asked to perform at least 10 falls in each direction, (front, back, left, right) as well as other activities of daily life such as walking, sitting, jogging, waving, doing the dishes, and playing the piano.

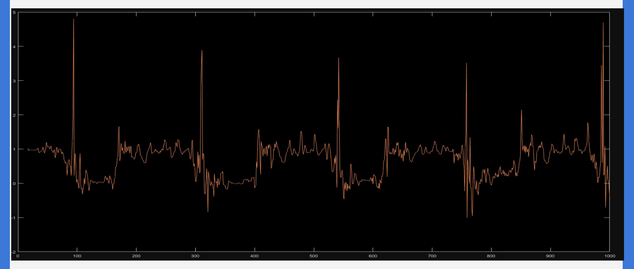
Sampling Frequency
- Previous data was sampled at 250ms which could miss critical fall data
- We used a 32ms sampling frequency as this is granular enough to detect falls without incurring unnecessary computation

Streamlining of Data Collection and App Execution
- Wrote an R script to clean all data we collected into the format desired (resultant, cvfast, smax, smin, outcome)
- Rewrote most of the data processing portion of the pure java app to work with anything we presented it with

Previous Data: 250ms Sampling Frequency



New Data: 32ms Sampling Frequency



Feature Extraction
- Resultant Acceleration: $\sqrt{(A_x)^2+(A_y)^2+(A_z)^2}$
- CvFast: $\sqrt{(A_{x\_max}-A_{x\_min})^2+(A_{y\_max}-A_{y\_min})^2+(A_{z\_max}-A_{z\_min})^2}$
- Smax and Smin: maximum and minimum resultant acceleration in the sliding window.

Features were calculated using a 750ms sliding window with a 66% overlap. All data is calculated for every instance of the sliding window, but CvFast, Smin, and Smax are calculated over the whole rectangular window, while the resultant acceleration is only calculated on a per instance basis.

Machine Learning Algorithm Choice

We utilized WEKA's Naive Bayes machine learning algorithm because it is computationally very efficient and has also shown to be superior for this application by other research as well.

ML Algorithms applied to Fall Detection

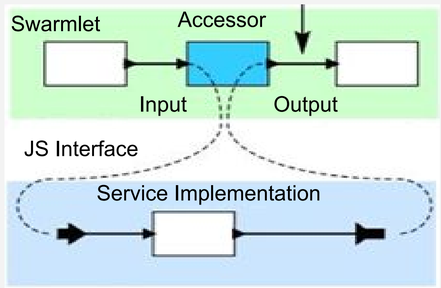
Algorithm	Correctly Classified Instances (%)	Incorrectly Classified Instances (out of 486)	Time to build model (millisecond)
SVM	64.40	173	2190
OneR	74.69	123	0
C4.5(J48)	81.07	92	550
Neural Network	84.57	75	920
Naive Bayes	**88.06**	**58**	**0**

(Yang, Dinh, Chen, A Wearable Real-time Fall Detector Based on Naive Bayes Classifier)
*A grid search was not performed for SVM

Accessors

An accessor is an actor that wraps a device or service in an actor interface. Actors are simply software components that contain streaming input and output ports.

Accessors are run on an accessor (swarmlet) host. An accessor host is a service running on the client platform that hosts applications built as a composition of accessors that stream data to each other.



Results: Real World Test

Previous TP: 44.7%

SVM Acc: 62.9%
SVM TP: 50%
SVM TN: 80%

	Predicted NotFall	Predicted Fall
Actual NotFall	24	6
Actual Fall	20	20

Naive Bayes Acc: 84.29%
Naive Bayes TP: 80%
Naive Bayes TN: 90%

	Predicted NotFall	Predicted Fall
Actual NotFall	27	3
Actual Fall	8	32

Our results are very comparable to other researchers' results. Also of note is that we utilized only the smartwatch's accelerometer data while others typically use both the smartwatch and smartphone's accelerometer data. Gjoreski, Vermeulen, and Maglogiannisa achieved TP rates of 70%, 75%, and 92.8% respectively.

Our results are for using a purely java program without utilizing the accessor design pattern. We believe that we could have the same results using the accessor design pattern, but with a reduced latency between predictions.

Accessor Design

There are several accessor hosts already available from the TerraSwarm research center, but none enable efficient design in an Android environment. We began building a true J2V8 accessor host for Android, but ran out of time. We attempted to follow the Accessor Specification v1.0 on the Accessor wiki (on the icyphy.org/accessors/wiki website) but were unable to get full functionality implemented which would have built on last year's project as they simply used the host file provided for purely javascript accessors. We went the J2V8 accessor host route because we can pass data back and forth between a java and javascript layer and we can only interact with the Microsoft Band 2 and weka through java.

Future Research

New Features
- Gravity, linear acceleration
- Velocity, Jerk

Recurrent Windowing

Include Smartphone's Accelerometer

Computational Cost Reduction
- Trigger function, (Run prediction only when feature X crosses threshold Y)

Fully Implement J2V8 Accessor Host
- This will provide a foundation for future mobile apps to be built with ease. As the accessor library expands, the ability of the apps built with this pattern also expands

Conclusions

- Using commercial smartwatch sensors can yield an acceptable fall detection system

- The Accessor Design Pattern rivals native Android apps in both functionality and development time

We thank the National Science Foundation for funding the research under the Research Experiences for Undergraduates site programs (CNS-1358939, CNS-1659807) at Texas State University to perform this piece of work and the infrastructure provided by a NSF-CRI 1305302 award. We also acknowledge Po-Teng Tseng and Vimal Moorthy Krishnamoorthy for their assistence in this project.