Pervasive and Mobile Computing 38 (2017) 60–76

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

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

Advancing Android activity recognition service with Markov smoother: Practical solutions[☆]

Mingyang Zhong[a,c,*], Jiahui Wen[a,c], Peizhao Hu[b], Jadwiga Indulska[a,c]

[a] *School of ITEE, The University of Queensland, Australia*
[b] *Department of CS, Rochester Institute of Technology, USA*
[c] *NICTA, Australia*

ARTICLE INFO

Article history:
Received 15 June 2015
Received in revised form 19 May 2016
Accepted 11 September 2016
Available online 22 September 2016

Keywords:
Activity recognition
Android
Markov smoother
Signal strength

ABSTRACT

Common use of smartphones is a compelling reason for performing activity recognition with on-board sensors as it is more practical than other approaches, such as wearable sensors and augmented environments. Many solutions have been proposed by academia, but practical use is limited to experimental settings. Ad hoc solutions exist with different degrees in recognition accuracy and efficiency. To ease the development of activity recognition for the mobile application eco-system, Google released an activity recognition service on their Android platform. In this paper, we present a systematic evaluation of this activity recognition service and share the lesson learnt. Through our experiments, we identified scenarios in which the recognition accuracy was barely acceptable. We analyze the cause of the inaccuracy and propose four practical and light-weight solutions to significantly improve the recognition accuracy and efficiency. Our evaluation confirmed the improvement. As a contribution, we released the proposed solutions as open-source projects for developers who want to incorporate activity recognition into their applications.

1. Introduction

According to the latest Cisco report [1], almost half a billion (497 million) mobile devices and connections were added in 2014. Smartphones accounted for 88% (439 million) of that growth. These compact, yet powerful, multi-functional smartphones have boosted the market shift from desktop to 'thin' mobile devices. Along with their slim design, smartphones are typically equipped with various sensors that can sense location, acceleration, orientation and biometric data, and this kind of context/intelligence provided by mobile devices is able to provide context information to various applications [2].

These added capabilities enable activity recognition to be performed directly with data gathered from the on-board sensors [3], rather than depending on wearable sensors [4,5] or environment augmentation [6,7]. For example, Kwapisz et al. [8] made use of smartphone's accelerometer data to recognize five common activities (including *walking*, *jogging*). Typical activity recognition procedures include data collection, feature extraction and training of classifiers, as a result of these processes a model is produced for detecting meaningful patterns. Various machine learning techniques (e.g., Hidden Markov Model, Support Vector Machine) have been used to map patterns to the desired activities. There are also hybrid models [9,10], which combine multiple simple models to improve recognition accuracy.

[☆] This paper has been substantially extended from our CoMoRea 2015 paper (Zhong et al. 2015).
[*] Corresponding author at: School of ITEE, The University of Queensland, Australia.
E-mail address: mingyang.zhong@uq.net.au (M. Zhong).

http://dx.doi.org/10.1016/j.pmcj.2016.09.003
1574-1192/© 2016 Elsevier B.V. All rights reserved.

The accuracy of recognition depends greatly on how developers handle each step in the above processes. These skills are often not a *must have* for most mobile application developers. To ease the development of mobile applications that need the activity recognition feature, Google released its Android activity recognition (AR) services in 2013. Through the service API, developers can request for recognition results based on an interval, much like the Android location service. Initially, four types of activities were supported: *Stationary*, *On Foot*, *Cycling*, *In Vehicle* and *Unknown*. In the 2014 update, three more activities were added: *Walking*, *Running* and *Tilting*. It has to be noted that the Google AR service recognizes activities based on on-board smartphone sensors without any additional input from wearable or environmental sensors so its ability to recognize additional new activities is limited. According to the documentations,[1] the Google AR service makes use of low-power, on-board smartphone sensors to recognize user's activity with efficient energy consumption. In our attempt to study this service, we discovered inaccuracy that has a significant impact on millions of mobile applications that could be using this service to trigger certain application behaviors or adaptation.

In this paper, we present a systematic qualitative and quantitative evaluations of this AR service, with a goal to investigate its accuracy, latency and complexity. Based on other referenced sources, together with our experiments, we demonstrate scenarios in which this AR service will perform poorly. We then propose four practical solutions: (i) ARshell [11], a post-processing step which uses a Markov smoother to improve the overall accuracy of all recognition categories up to 15.2%; (ii) ARshell HMM, similarly with ARshell, a post-processing method applying Hidden Markov smoother to avoid some manual settings of ARshell, and it improves the accuracy by 1.2%; (iii) ARsignal, a lightweight cellular signal based method using Support Vector Machine (SVM) [12] that shows a strong discriminative power on classifying stationary class; and (iv) ARshell+, a hybrid approach that integrates the advantages of both ARshell and ARsignal, and it further enhances overall accuracy to 91% (21.2% improvement compared with the accuracy of the AR service). We released ARshell+ as an open-source project on GitHub as a contribution to researchers and developers who might be interested in this improvement of the Google AR service. In addition, the ARshell+ API eases the integration of AR service in research projects and mobile application development.

There are three main contributions presented in the paper:

- A systematic evaluation of the Google AR service;
- Effective and lightweight practical methods to significantly improve the AR accuracy;
- A cost analysis of our proposed solutions, regarding CPU usage, memory occupancy, and power consumption.

The remainder of this paper is organized as follows. Section 2 presents related work on activity recognition. Section 3 describes evaluations of the Google AR service. This is followed by our proposed solutions and their evaluation in Section 4. We conduct experiments to analyze the cost in terms of CPU load, memory usage and power consumption in Section 5. Section 6 concludes the paper, followed by the Appendix on how we map our approach to HMM.

2. Related work

In the literature, there exist many works on activity recognition. In this section, we will focus on work that are closely related to the scope of this paper.

In one of the pioneering work, Bao et al. [13] proposed a method to recognize physical activities with multiple sensors attached at different positions of a human subject. The participants were asked to perform daily activities. Sampling data of these activities were recorded and annotated manually. The authors then extracted features, such as *mean*, *energy* and *frequency-domain* entropy for the training processes. Through the testing of multiple classifiers, the authors found that Decision Tree performs reasonably well and achieves the highest accuracy. Since then, other similar approaches with attaching multiple sensors to human subject were proposed. For example, Quwaider and Biswas [14] proposed a method to classify specific movements of a region of the human body. Wang et al. [15] constructed common models with human knowledge to minimize labeling overhead and improve the recognition accuracy. As unsupervised and supervised approaches require estimating many parameters and the manual labeling, respectively, Palmes et al. [16] simplified the activity modeling process by mining the web to extract the most relevant objects. However, it is proposed for smart environments, and cannot be applied to activity recognition based on smartphones. Using application-specific and custom built wearable sensors may lead to high recognition accuracy [17,18], however it is obtrusive to the users and cumbersome to apply in a large scale [19].

With smartphones spreading rapidly, there is an increasing interest to perform activity recognition using on-board sensors of these smartphones. There are abundance of proposals on performing activity recognition with smartphones. In one of them, Kwapisz et al. [8] presented an activity recognition system that uses accelerometer data from 29 participants engaged in physical activities (such as *walking*, *jogging*, *climbing stairs*, *sitting* and *standing*) to recognize activities with a single device in their pocket. They segmented the sensor data into 10 s sampling windows. Features, such as *mean*, *standard deviation*, *average absolute difference*, *average resultant acceleration*, *time between peaks* and *binned distribution*, were extracted from the collected data for constructing the model. Multiple predictive models (e.g., decision tree, logistic regression, multi-layer neural networks) have been explored for activity recognition. The authors concluded that their

[1] http://developer.android.com/training/location/activity-recognition.html.

solution can recognize *walking* and *jogging* with high accuracy. Unfortunately, *walking* downstair and upstair are the most difficult activities to be distinguished due to their similar patterns in acceleration. Shoaib et al. [20] explored the role of gyroscope and magnetometer on smartphones for activity recognition. They experimented on four body positions using seven classifiers while trying to recognize six physical activities. They concluded that accelerometer and gyroscope complement each other in general, however magnetometer is not so promising due to its dependency on directions. Dernbach et al. [21] investigated the ability to recognize not only locomotion activities, but also high level activities, such as *cooking* and *cleaning*. In general, the recognition accuracy using on-board sensors alone is lower than other approaches in which specific sensors are attached to targeted areas of human body. However, activity recognition using smartphones is still a promising way to enable some levels of application adaptation and situation awareness [22].

While there are works on exploring activity recognition using smartphones, other work focus on ways to improve recognition accuracy and efficiency. For example, Hemminki et al. [23] proposed a method to eliminate the influence of gravity (vertical acceleration), while maintaining high accuracy on recognizing horizontal acceleration. They confirmed that the method improves the accuracy of recognizing modes of transportation by 20%. Maekawa et al. [24] proposed the idea of finding a model for recognition based on the similarity in user profiles, rather than constructing specific model for individual users. The idea is to minimize the complexity in data collection and training processes. Also, it encourages model reuse. On a similar idea, Cvetkovic et al. [25] proposed the idea of a general classifier for users that are alike. Then, individual differences are incorporated to be the user-specific classifier. There are also approaches focused on energy efficiency. Gordon et al. [26] proposed a predictive method to activate subset of sensors based on the likelihood of future activities. Yan et al. [27] proposed an approach to dynamically adjust the sampling rate and classify features in realtime to balance the trade off between accuracy and energy consumption. In [28], Zappi et al. proposed to balance the trade-off between accuracy and power consumption by dynamically selecting the best set of sensors that have the discriminative power to meet the desired minimum recognition accuracy. Whereas in [29], Keally et al. developed a method to choose a subset of the sensors for activity classification in order to save energy and minimize training overhead. They put a constraint so that no two selected sensors have a correlation coefficient that is greater than or equal to a pre-defined threshold, the basic idea is that sensors with *zero-decision-correlation* tend to achieve higher recognition accuracy than otherwise. The authors of [30] proposed an energy-efficient context monitoring framework, in which only a subset of the context sources are needed to be monitored. Ju et al. [31] proposed an efficient dataflow execution method for mobile context monitoring application. The observation is that multiple application running on the mobile devices may consume the same context, and then energy can be saved if the context could be processed uniformly and provided to multiple application through a middleware. In these solutions that improve energy efficiency, there are generally two ways—using a subset of sensors and adjusting the sampling rate. However, on-board sensors of smartphones are limited, such as accelerometer and gyroscope, using a subset of the sensors may not be able to achieve a reasonable accuracy. Furthermore, to guarantee the accuracy, a minimum sampling rate is required, and it also depends on the frequency of recognition updates required by the application. Thus, a basic amount of energy cost is inevitable.

Although a substantial amount of research has been carried out on activity recognition on mobile devices, there was no activity recognition service available for Android application developers before the Google AR service. This service has the potential to revolutionize the development of mobile applications that offer better user experience. The following section provides insights of the AR service regarding its accuracy, latency and complexity.

3. Evaluation of Google activity recognition service

Through the service API, developers can support activity recognition in their applications, without dealing with complex pattern analysis of sensor data. According to the documentation, the AR service is bundled together with the location services and is part of the Google Play services APK. However, our experiment results (when we remove SIM card and disable WiFi and GPS) suggest that the AR service performs activity recognition based on readings from the on-board sensors, for example, accelerometer, gyroscope and compass.

To access the AR service, a mobile application must be granted with a special permission.[2] To receive update on recognized activities, applications define a callback function and specify an interval for receiving updates of recognized activity. The intention of this update interval is to provide developers the control of freshness of measurements and power consumption. Also, the Android OS uses this interval to optimize efficiency by merging queries from different applications. When time is up, the system will trigger the callback function with the last recognized activity. In addition to the activity update logic, mobile applications define methods for *starting* and *stopping* of the service and error handling. We perform our evaluations using a demo code provided, with an additional code for recording measurements.

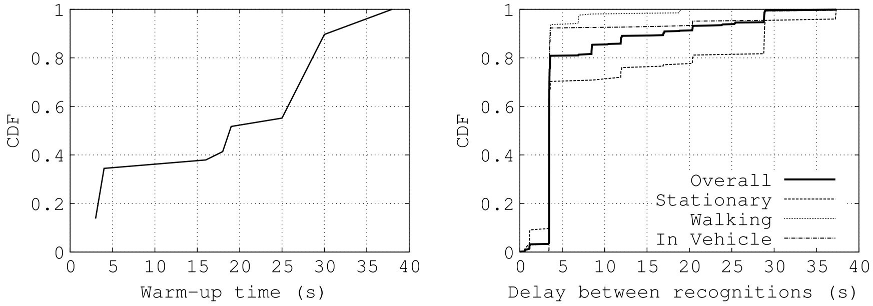
3.1. Experimental setup

All of the experiments are carried out on actual Android devices. The details of a setup are as follows:

[2] com.google.android.gms.permission.ACTIVITY_RECOGNITION.

Table 1
Definition of activity classes.

Activity class	Definition
Stationary	The device is stationary (not moving).
On Foot	The device is on a user who is walking or running.
Walking	The device is on a user who is walking (a sub-class of *On Foot*).
Running	The device is on a user who is running (a sub-class of *On Foot*).
Cycling	The device is on a bicycle.
In Vehicle	The device is in a vehicle, such as a car.
Tilting	The device angle relative to gravity changed significantly.
Unknown	Unable to detect the current activity.



(a) Delay before 1_{st} data point.

(b) Delay between consecutive data.

Fig. 1. CDF of two aspects of delay.

- Android devices: HTC Desire C with 600 MHz processor and 512 MB memory, Samsung Galaxy Nexus with 1.2 GHZ dual-core processor and 1 GB memory, and Samsung NOTE II with 1.6 GHz quad-core processor and 2 GB memory. These devices represent three distinctive classes of device capability,
- Experiment duration: the experiments were carried out by the authors over one month period, and over six thousand minutes of data have been collected using the aforementioned devices in various scenarios: *Stationary*, *Walking*, *Running*, *Cycling*, *In Vehicle* and *Tilting*, and the definition of these activities provided by Google is shown in Table 1. Note that the collected data has also been used for the comparison studies on different schemes in Section 4. All the experimental data was recorded with a human label (as the ground truth). We sampled the data outputted from the Google AR service every second, and we collected data for two hours for each activity.

3.2. *Delay*

The update *interval* parameter is designed to be a trade-off to balance between freshness of the measurement values and power consumption. Developers should choose an appropriate value according to their application requirements. To better understand the delay characteristic of the AR service, we conducted experiments to investigate the *warm-up* time, which is the delay before the first recognized event from the AR service, and the delay between two consecutive recognized events reported by the AR service.

Fig. 1 shows the Cumulative Distribution Function (CDF) of the two aspects of delay. As for the warm-up time, we conducted multiple experiments to measure the average warm-up time and standard deviation. The results are 18.3 s and 13.9 s, respectively. Furthermore, Fig. 1(a) shows that the warm-up time can be as long as 30 s or more in about 10% of our experiments, and the reason is that the AR service is not able to generate recognition result when the confidence values of all possible activities are below a threshold, detailed in Section 4.1.1. Tens of experiments have been carried out for testing the minimum waiting time from starting the service to receiving the first recognition, and the averaged value of that is 3 s.

Typically, a sampling window is used in most activity recognition, either with fixed window size or dynamic size [12], and pattern matching is applied on the sensor data with respect to this window. Because of the lack of access to the source code of the AR service, we cannot determine the exact window size. Rather, we try to measure the time delay between two consecutive recognized events to approximate the minimum interval software developers can use for receiving activity updates. We conducted experiments with the interval fixed to 0 s (notify whenever update is available), and collected data samples over an hour (roughly one thousand measurements). In our experiments, regarding the delay between receiving two consecutive updates over all activities, the absolute minimum delay was around 0.5 s. As shown in Fig. 1(b), slightly over 80% of the overall data samples show a delay less than 3.5 s. Furthermore, the minimal delays below 3.4 s are rare cases that happen with the probability less than 4%. In some extreme cases, the time between two consecutive recognitions can take as long as 37 s. To explore the impact of delay on different activities, Fig. 1(b) shows the CDF of delay of three example activities (*Stationary*, *Walking* and *In Vehicle*). We can see that *Walking* and *In Vehicle* show quicker response time than *Stationary* in

Table 2
Average delay between consecutive recognitions.

Activity	Average delay (s)
Stationary	9.7
Walking	3.9
Running	4.2
Cycling	11.3
In Vehicle	5.6
Overall	6.9

Table 3
Confusion matrix and accuracy of Google AR service.

	Classified to:								A_{aggr}
	S	W	R	F	C	V	T	U	
S	**39**%	2%	0	6%	0	21%	3%	29%	*39%*
W	0	**75**%	4%	**6**%	0	0	4%	11%	*81%*
R	0	7%	**45**%	**28**%	10%	0	4%	6%	*73%*
C	0	0	0	0	**68**%	0	22%	10%	*68%*
V	3%	0	0	1%	1%	**88**%	2%	5%	*88%*
A_{avg}									*69.8%*

Note: S—Stationary, W—Walking, R—Running, F—On Foot C—Cycling, V—In vehicle, T—Tilting, U—Unknown.

around 20% cases. Similarly to the warm-up delay, we also noticed large delay between two consecutive recognitions when the AR service is not able to generate recognition result if the confidence values of all possible activities are below the threshold, such as standing users using their smartphones or vehicles frequently stopping. We also calculated the average delay between consecutive recognitions when performing different activities as shown in Table 2. Note that there are two forms of the stationary scenarios: (i) the device is on a stationary user who is sitting or standing, and (ii) the device is on a stationary structure such as a desk. As the AR service preforms well on both, accuracy (described in Section 3.3.1) and delay for the latter case, which incurs 3.1 s of the average delay, the *Stationary* in Table 2 and Fig. 1(b) only states the results of the former case.

In another set of experiments, we investigate the relationship between update interval and accuracy of recognition. According to our results, different interval settings have no affect on the accuracy. Thus, we use 0 s as the update interval for the rest of our experiments.

3.3. Accuracy

Accuracy is a direct measure of the usefulness of the recognition algorithm. In this subsection, we evaluate the performance of the Google AR service with various scenarios. Similar to the most proposed AR solutions, after collecting enough data samples over the sampling window, the AR service proposes the most probable activity and a list of probable activities, each with a confidence value ranging from 1 to 100. That is,

$$\{(a_{most}, cv_{most}), [(a_1, cv_1), (a_2, cv_2), \ldots, (a_n, cv_n)]\}.$$

If an activity a_{most} has confidence value cv_{most} of 100, it implies absolute certainty of the activity and the list of probable activities $[(a_1, cv_1), (a_2, cv_2), \ldots, (a_n, cv_n)]$ is *null* except for *Walking* and *Running*. As these two activities are the sub-activities of *On Foot*, the *cv* of one of these two sub-activities can be 100 if the *cv* of *On Foot* is 100. We gathered results from each activity and collectively presented them in a confusion matrix, as shown in Table 3. In the confusion matrix, the first column contains the names of our labeled activities (or activity under test). The last column (indicated as A_{aggr}) is the aggregated accuracy of the AR service on correctly recognizing the activity under test. Columns between the first and last columns are the distribution of a correct and false classification for each activity. The label A_{avg} indicates the overall accuracy across all activities over all experiments.

We computed the accuracy of the AR service by comparing a_{most} with the corresponding human label activity a_{label} in a sequence of measurements. Thus, in a sequence of measurements reported by the AR service the aggregated accuracy, A_{aggr}, for an activity type $c \in C$ is calculated as

$$A_{aggr} = \frac{1}{N}\sum_{n=1}^{N}[a_{most} = a_{label}] \tag{1}$$

where $[a_{most} = a_{label}]$ equals to 1 if it is evaluated to true, 0 otherwise. The average accuracy A_{avg} across all activity types C is calculated as

$$A_{avg} = \frac{1}{C}\sum_{c=1}^{C}A_{aggr}. \tag{2}$$

3.3.1. Stationary

There are two basic forms of the stationary scenarios: (i) when a user is sitting or standing still while holding the device under test in his/her hand or having it in a pocket; and (ii) when the device under test rests on a stationary structure, such as a desk. The latter case achieves over 99% of accuracy. But it represents scenarios with no movement at all and is slightly less interesting compared to the former scenarios, which achieves 39% of accuracy. As the AR service achieves reasonable accuracy regarding the latter case, Table 3 only shows the detail of the former case. Unlike traditional on-body sensors [9,32] which are fixed on certain body positions, smartphones usually experience small motions when users interact with them, thus 61% of misclassified activities are distributed across other activities. In our experiments 32% of activities have been reported as *Tilting* and *Unknown*. As mentioned in [20], the AR service can potentially leverage the accelerometer and the gyroscope to improve the accuracy. Software developers should be careful about the meaning of being *Stationary*.

3.3.2. On Foot

In the 2014 update, Google extended the AR service to support recognition of *Walking* and *Running* as two sub-classes on *On Foot*, potentially with additional sensor inputs. Due to the different speed and motion when people walk or run, we investigate the accuracy of recognizing these two sub-activities. Data shown in the confusion table confirms our doubt. Other activities are reported even when users are running at a constant speed. When running at a lower speed, data reported from the acceleration is not as significant as for fast-running and is not as modest as for walking. Therefore, recognizing *Running* is less accurate when compared to *Walking*. Furthermore, people can perform multiple activities in a time period, and most of the activity transitions are clear and detectable, such as from *In Vehicle* to *Walking* detailed in Section 3.3.6. However, the activities performed in a time period can be a composite of *Walking* and *Running*. In this case, if the AR service is able to distinguish them, the time period will be segmented as well and different activities will be reported accordingly. Otherwise a standard fallback is applied and reports the person being *On Foot* whenever sensor readings are not distinctive and cannot distinguish between the two sub-activities. It seems ambiguous that one activity such as walking may be recognized as two classes (*Walking* and *On Foot*), but logically, both *Walking* and *Running* are sub-classes of *On Foot*, thus it is reasonable to consider *On Foot* as a correct recognition for these two sub-activities.

3.3.3. Cycling

As for *Cycling*, we noticed that around 22% of samples have been reported as *Tilting*. *Tilting* reflects that the device angle relative to gravity changed significantly (analyzed in Section 3.3.5). When cycling on uneven roads, device rotation is typically more significant compared to cycling on flat roads. Our experiment results show lower accuracy when traveling on uneven roads. In these cases, a large percentage of data has been reported as *Tilting*.

3.3.4. In Vehicle

With regard to *In Vehicle*, when the vehicle is traveling at a constant speed the recognition accuracy is around 88%. While the vehicle is traveling slowly, together with stopping patterns, the accuracy falls to 41%. From our experiments, we observe that a total of 59% of activities are misclassified in activity class rather than being in vehicle; among those, 42% of activities are reported as *Unknown*. We conjecture that the poor performance is due to the rapid changing of activities (e.g., multiple activities occurred in one sampling window). When this happens the recognition algorithm is not able to distinguish multiple activities, therefore *Unknown* is reported.

3.3.5. Tilting

According to Google's definition of *Tilting* as given in Table 1, it is not clear what kind of motions will trigger *Tilting* activity and how significant the change of angle to gravity is required. Thus, we conduct experiments to investigate the *Tilting* class. In the experiment shown in Fig. 2, we rotate the device along the X axis for an angle a of 90° and pause for $t_{pause} = 30$ s, then we rotate back. After another t_{pause} time, we repeat the rotation. The 90° rotation should be significant enough to trigger the *Tilting* output. Fig. 3(a) shows a snapshot of the time series data. From the recorded observations, the AR service seems able to recognize *Tilting* from all the *Stationary* in this scenario with an average delay 4.4 s after actual *Tilting* motions. To provide further insight into the *Tilting* class, we carry out additional experiments regarding pause time t_{pause}, rotation angle a, rotation speed and rotation axis.

Changing the t_{pause}: As in previous scenario when $t_{pause} = 30$ s the AR service is able to discern all the *Tilting* activities, extending the t_{pause} will not affect its performance. In this experiment, we shorten the t_{pause} to 15 s and 1 s. When $t_{pause} = 15$ s, around 40% of *Tilting*s are missed, as shown in Fig. 3(b). The rest are recognized as *Stationary*. When $t_{pause} = 1$ s, the rapid changing motions cause the AR service to derive activities such as *On Foot*. Only 3% of the activity recognitions are reported as *Tilting*.

Changing the a: In this experiment, we evaluate the AR service by setting the a to different values: 15°, 30°, 45°, 60° and 90°, respectively. We find that the AR service is not able to recognize *Tilting* when the a is 15° or 30°, and it detects 40% of the *Tilting*s when $a = 45°$, and it is able to detect *Tilting* with 100% of accuracy when the a is 60° or 90°. This set of experiments clarifies the meaning of the term "significantly" used in Google's definition of *Tilting*, as given in Table 1.

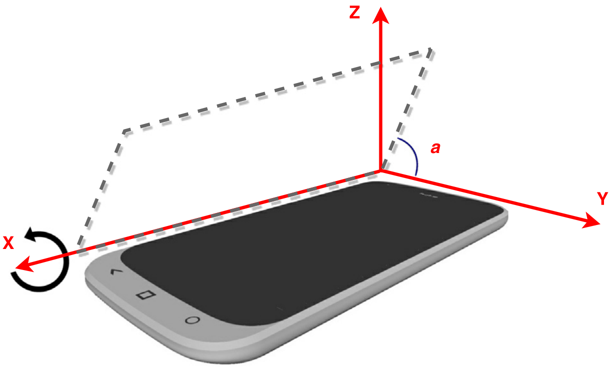
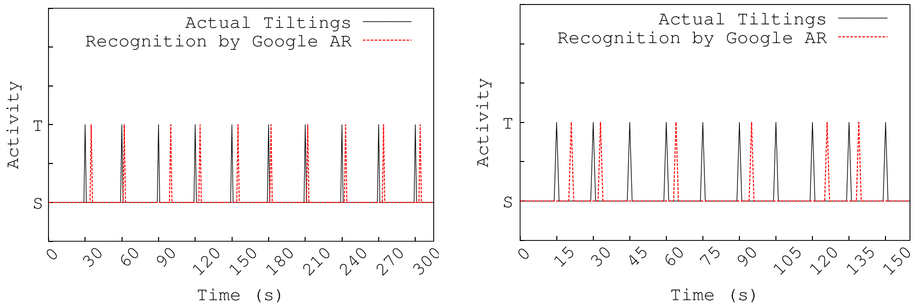


Fig. 2. Experiments for investigating *Tilting*.



(a) 30 s.

(b) 15 s.

Fig. 3. Experiments on Tilting with different pause time.

Changing the rotation speed and the rotation axis: In another set of experiments, we change the rotation speed from quick rotation to slow rotation, while in another experiment we change the rotation axis from along the X axis to rotation along the Y axis. The AR service is able to recognize *Tilting* in these experiments.

Tilting is a special activity that may occur along with other activities, as shown in Table 3. Unfortunately, we cannot use one *Tilting* output to distinguish the other 6 activities. Because all of them can contain the motion of tilting. In addition, from our experiments the angle change relative to gravity has to be greater 45° in order to trigger *Tilting*. As a result, we leverage the behavior of *Tilting* and develop heuristics in our proposed solutions to improve the recognition accuracy.

3.3.6. Transition between activities

After each interval, the AR service reports the list of probable activities, with corresponding confidence values. We conducted a detailed study on what happens when there is a transition from one activity to another. Fig. 4 shows the confidence values of each class of activities in the list of the probable activities reported by the AR service, while the bottom part of the figure presents the index of activities labeled by a human. We highlighted the most probable activity (with the highest confidence) with circles. We noticed that most misclassifications for *In Vehicle* occur during the time when a bus is stopping or in stationary (V_S, e.g., in *zone*1 between a and b). The AR service reports that the user is *Stationary*. It is logically correct. However, since the user is still inside the vehicle, it might be more appropriate to report *In Vehicle*. In *zone*2 (between b and c) in which the vehicle mostly moves with normal speed (V_N), we observed a significant increase of *Unknown* reported by the AR service, due to the frequent stopping of the vehicle. This is consistent with our results reported in Table 3. At the same time, the confidence values of *In Vehicle* typically drop below 60, which means a high entropy of the posterior distribution. This is one of the insights that we used in our improvement of the AR service to smooth the outliers when detecting uncertainties. In *zone*3 (between c and d), we observed the transition from *In Vehicle* to *Walking* after some delays. To summarize, irregular behaviors can have significant impact on the accuracy of recognition. Heuristic might be added to filter occasional misclassifications caused by irregular behaviors, therefore to prevent mis-firing of application adaptations.

4. Roadmap to the design of ARshell+

To improve the recognition accuracy, we investigate four post-processing methods to improve the recognition accuracy: (i) ARshell (Activity Recognition Shell) was initially proposed in our earlier paper [11]. It takes the outputs generated by the Google AR service and applies a Markov smoother to derive new results. The accuracy of ARshell depends on the selection of the activity transition threshold. (ii) To avoid specifying this parameter, we explore the use of Hidden Markov in ARshell,

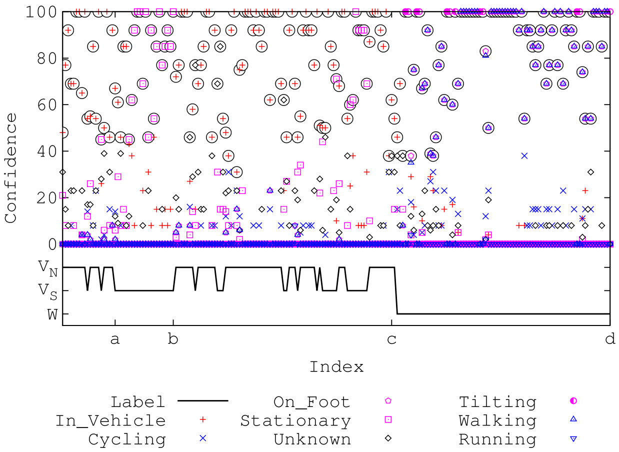


Fig. 4. Experiment on activity transition from *In Vehicle* to *Walking*.

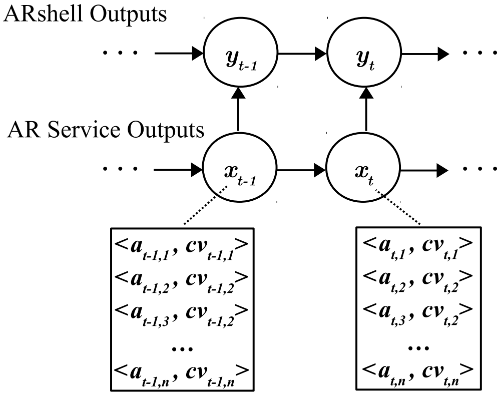


Fig. 5. Modeling of activity transition in ARshell.

therefore, naming it ARshell HMM; (iii) ARsignal is a low-cost scheme to recognize activity based on cellular signal strength; and (iv) ARshell+ merges ARshell and ARsignal to improve the recognition accuracy across all categories.

4.1. ARshell

As the Google AR service generates recognition results, ARshell compares these recognized activities (as the probable activities list sorted by their confidence values) to the previous activity generated from ARshell. We apply a Markov smoother to emphasize the temporal relationship embedded in the human activities. That is, current activity is more likely to continue into the next time window than transiting to a new one, unless new observations strongly suggest (with high confidence value) a different class of activity. This temporal characteristic of human behavior has been justified for smoothing time slice sequences in previous work. For example, in [33] the authors use the characteristic of transition probability when constructing the dynamic Bayesian network for activity recognition.

To explain the algorithm of ARshell, we model the problems concerning transition between activities in Fig. 5. As shown in the figure, we introduce the following notations:

- $y_{t-1}, y_t, \ldots \in Y$ is a list of activities that are generated by ARshell with different timestamps.
- $x_{t-1}, x_t, \ldots \in X$ is a list of activity datasets that are reported by the AR service with different timestamps. Each x is a list of probable activities with their confidence values, which is modeled as a tuples list $[(a_{i,j}, cv_{i,j})]$, where $(a_{i,j}, cv_{i,j})$ is a probable activity and its corresponding confidence value. Tuples are sorted according to the descending order of cv. i is the timestamp, where j indicates the order in the probable activities list. For example, the AR result x_t contains the tuples list of $[(a_{t,1}, cv_{t,1}), (a_{t,2}, cv_{t,2}), (a_{t,3}, cv_{t,3}), \ldots, (a_{t,n}, cv_{t,n})]$. It should be noted that $cv_{t,1} \geq cv_{t,2}$, and $(a_{t,1}, cv_{t,1})$ is always the *most probable* activity in the AR service definition. Each $x_{t-1}, x_t, \ldots \in X$ is an input to ARshell for post-processing to determine the corresponding $y_{t-1}, y_t, \ldots \in Y$.

There are four activity transitions that need to be addressed by ARshell: (i) *from Tilting or Unknown to a specific activity*; that is, $x_{t-1} \rightarrow x_t$ where x_{t-1} is reported as *Tilting* or *Unknown* and x_t is reported as one of the activities; in this case, the

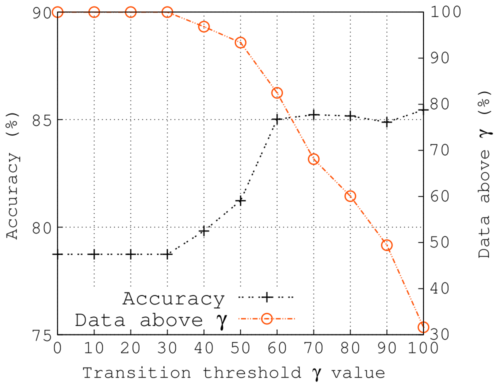


Fig. 6. Impact of threshold γ on accuracy and data size.

most probable activity $(a_{t,1}, cv_{t,1})$ is proposed for y_t, where $cv_{t,1}$ is the maximum confidence value; (ii) *from a specific activity to Tilting or Unknown*; that is, in the same transition $x_{t-1} \rightarrow x_t$ where $y_{t-1} = x_{t-1}$ is reported as a specific activity and x_t is *Tilting* or *Unknown*; in this case, we assume the last historical activity as the current activity $y_t = y_{t-1}$, with a self-transition probability ($\theta = 1 - \varepsilon$, where ε is a sufficiently small number [15]); (iii) *self-transition: a specific activity continues into next timestamp*; that is, in transition $x_{t-1} \rightarrow x_t$, if $x_{t-1} = x_t$, then $y_t = x_t$; and (iv) *activity switches from one to another*.

Whenever there is a transition like case (iv), we apply a Markov smoother to the AR service outputs. From our previous experiments, we identify that most misclassifications (also considered as noises) have confidence values below 60, which we use as a threshold γ to determine the transition of activity. As a result, when a transition occurs at timestamp t for x_t, we check the confidence value $cv_{t,1}$ of the most probable activity $(a_{t,1}, cv_{t,1}) \in x_t$. If $cv_{t,1} > \gamma$, $a_{t,1}$ is used as the output of ARshell ($y_t = a_{t,1}$). Otherwise, the Markov smoother backtracks to previous AR service outputs, and finds $\exists a_{t-1,k} \in x_{t-1}$ and $a_{t,1} = a_{t-1,k}$. We compute the $cv_{sum} = \sum(cv_{t,1}, cv_{t-1,k})$. If $cv_{sum} > \gamma$, then we say activity transition occurs and the new activity should be $a_{t,1}$. Therefore, we have $y_t = a_{t,1}$. Otherwise, we say the confidence value of two consecutive recognitions is not significant enough to determine an activity transition. In this case, ARshell will output $y_t = y_{t-1}$.

4.1.1. Finding an appropriate threshold γ

To determine an appropriate γ value, we conduct additional experiments to investigate the impact on recognition accuracy with different γ values. For all $\gamma = (0, \ldots, 100)$, we generate a subset of data that has confidence value (as reported by the AR service) above the selected threshold γ. In other words, the threshold is used as a filter to remove data with a lower confidence value. We compute the accuracy (reported activity matching the human label) for each dataset. Fig. 6 shows the results of this study. The general observation is that as we increase the γ value, there will be less data we will consider for recognition. This data is used by ARshell to determine the output of activity; it has significant impact on the ARshell performance. For example, if a new data (an activity update) is ignored because of its confidence value is below γ, then ARshell will assume previous activity continues. Therefore, we need to determine a γ value at which we maintain good recognition accuracy and allow large percentage of data to pass through to the recognition stage. As shown in the figure, the accuracy increases and becomes steady when $\gamma \geq 60$. As an example, when we compare $\gamma \geq 60$ and $\gamma \geq 70$, the difference in accuracy is not significant, while the percentage of data produced by the AR service is reduced by around 15%. Hence, we use $\gamma \geq 60$ as our threshold.

When we correlate the two lines, we make another observation. The percentage of data generated by the AR service stays at 100% for $\gamma \leq 30$. This indicates that the Google AR service only generates recognition value when the confidence value is at least 30.

4.1.2. Accuracy of ARshell

We evaluated ARshell on the three representative classes of Android devices as described in Section 3. The recognition accuracy is presented in Table 4. We observed a significant improvement regarding the accuracy for all the activities, except *Stationary*. In the scenarios for evaluating *Stationary*, we sit down or stand still with the testing devices held in our hands or in our pockets. The results of the AR service not only shows no temporal pattern, but also occasional readings from the accelerometer become outliers that cause misclassifications. It is very difficult to correctly classify these random patterns. Furthermore, we noticed that 35% of outputs were misclassified as *Cycling* in this *Stationary* scenario, and the reason for this kind of misclassification is that ARshell backtracks to previous AR service outputs and finds that the cv_{sum} of *Cycling* is greater than the γ, and then outputs *Cycling*. In contrast, if we put the devices on a desk, which implies absolute stationary, we see that the accuracy increases to close 100%. This is consistent with our observation in the previous experiments. With regard to all the other activities, our method is able to leverage the temporal information to smooth the outliers and achieve higher accuracy. For example, as shown in Table 3, when cycling there is a significant percentage of misclassifications (around 32%) that have been reported as *Tilting* or *Unknown*. By incorporating the temporal constraints, ARshell achieves a significant

Table 4
Confusion matrix and accuracy of ARshell.

	Classified to:								A_{aggr}	A_{diff}
	S	W	R	F	C	V	T	U		
S	**42**%	0	11%	0	35%	0	0	12%	*42%*	*↑3%*
W	0	**96**%	0	**0**	0	0	0	4%	*96%*	*↑15%*
R	0	0	**52**%	**46**%	0	0	0	2%	*98%*	*↑25%*
C	0	0	0	0	**92**%	0	6%	2%	*92%*	*↑24%*
V	2%	0	0	0	0	**98**%	0	0	*98%*	*↑10%*
A_{avg}									*85%*	

Note: S—Stationary, W—Walking, R—Running, F—On Foot C—Cycling, V—In vehicle, T—Tilting, U—Unknown.

Table 5
Confusion matrix and accuracy of ARshell HMM.

	Classified to:								A_{aggr}	A_{diff}
	S	W	R	F	C	V	T	U		
S	**22**%	0	0	0	0	47%	3%	28%	*22%*	*↓17%*
W	0	**82**%	0	**0**	0	0	4%	14%	*82%*	*↑1%*
R	0	0	**36**%	**58**%	0	0	3%	3%	*94%*	*↑21%*
C	0	0	0	0	**72**%	0	19%	9%	*72%*	*↑4%*
V	0	0	0	1%	0	**86**%	2%	11%	*86%*	*↓2%*
A_{avg}									*71%*	

Note: S—Stationary, W—Walking, R—Running, F—On Foot C—Cycling, V—In vehicle, T—Tilting, U—Unknown.

improvement and reduces the total misclassification for the given example to 8%, as shown in Table 4. The overall average accuracy has been improved from 65% to 85%. The column A_{diff} shows the improvement difference of each activity over the Google AR service. As shown in the table, the accuracy of all activity classes have been improved.

4.1.3. Applying HMM to ARshell

To avoid specifying the parameter γ in ARshell, we explore the use of Hidden Markov Model (HMM) to smooth the results reported by the Google AR service. We call this proposal ARshell HMM. We construct the HMM based on the posterior predictive distribution of recognition results given by the AR service. In this section, we only show the inference process of ARshell HMM that obtains the most likely sequence of predictions with the Viterbi algorithm and we discuss in the Appendix how our proposed solution is mapped to HMM.

We borrow the modeling notations from ARshell. In addition, for time t we maintain a vector to store the accumulative probability $\psi_{t-1,i}$ of time $t-1$, and the corresponding activity $a_{t-1,i}$; e.g., $[(a_{t-1,1}, \psi_{t-1,1}), \ldots, (a_{t-1,N}, \psi_{t-1,N})]$, N corresponds to the number of the supported activities. At time t, we can use dynamic programming to compute $\psi_{t,i}$:

$$\psi_{t,i} = max(\psi_{t-1,j}) * \theta_{a_j->a_i} * cv_{t,i} \quad \textit{for } i, j = 1, \ldots, N \tag{3}$$

where at time $t = 0$, $\psi_{t-1,i}$ is set to $cv_{0,i}$. The function $max(\psi_{t-1,j})$ for $j = 1, \ldots, N$ returns the highest accumulative probability from time $t-1$ [34]. $\theta_{a_j->a_i}$ is the transition probability from activity a_j to a_i. In our implementation, $\theta_{a_j->a_i} = \varepsilon$ if $a_j \neq a_i$ and $a_j, a_i \neq$ *Tilting* or *Unknown*, otherwise $\theta_{a_j->a_i} = 1 - \varepsilon$.

Finally, the prediction y_t (w.r.t. the output of ARshell HMM, as shown in Fig. 5) at each time t is the activity $a_{t,i}$ with the corresponding maximal $\upsilon_{t,i}$.

$$y_t = a_{t,i} \quad s.t.: \ arg\ max_i(\psi_{t,i}) \ \forall(a_{t,i}, \psi_{t,i}), \ i = 1, \ldots, N. \tag{4}$$

Table 5 shows the results of ARshell HMM. The accuracy improvement is not as significant as ARshell. The accuracies of *Stationary* and *In Vehicle* experience a decrease of 17% and 2% respectively. Taking *Stationary* as an example, some instances of *Stationary* have been misclassified as *In Vehicle*, *Unknown* by the Google AR service, thus without domain knowledge, such as the γ in ARshell, ARshell HMM rectifies *Stationary* as *In Vehicle*, *Unknown*, which results in low precision of recognizing such activity. In contrast to ARshell, which uses a transition threshold and characteristic to filter invalid activity transition, especially transitions caused by instantaneous recognition value with high confidence value. In other words, the HMM approach is based on the statistical results, but ignoring the heuristic that we incorporated in ARshell. For example, ARshell assumes an activity continues if the AR service provides *Unknown*, while ARshell HMM simply returns *Unknown*.

4.2. ARsignal

While developing ARshell and its HMM version, we notice significant improvement in recognition accuracy for all supported activities, except *Stationary*. Here we are interested in *relative Stationary* scenarios because (i) recognizing *absolute Stationary* is very straightforward (in fact the Google AR service achieves over 99% accuracy), (ii) *relative Stationary* is very common, but challenging to be correctly recognized; e.g., playing with your phone when waiting for a bus.

Table 6
Comparison of accuracy of different classifiers.

	SVM	ANN	Decision Tree	Naive Bayes
Precision	0.851	0.826	0.796	0.819
Recall	0.854	0.828	0.79	0.788
F-Measure	0.843	0.824	0.779	0.794

Table 7
Feature formulation.

Feature	Formulation
$\bar{s}$	$\frac{1}{N}\sum_{i=1}^{N} s_j$
var	$\frac{1}{N}\sum_{i=1}^{N}(s_i - \bar{s})^2$
σ	$\sqrt{var}$
$\lvert D\rvert$	$\frac{1}{N}\sum_{i=1}^{N} \lvert s_i - \bar{s}\rvert$
mcr	$\frac{1}{N-2}\sum_{i=2}^{N-1} I\{(s_{i-1} - \bar{s}) * (s_{i+1} - \bar{s}) < 0\}$

Table 8
Confusion matrix and accuracy of ARsignal.

	Classified to:			A_{aggr}
	Stationary	On Foot	In Vehicle	
Stationary	**83**%	7%	10%	*83%*
On Foot	8%	**50**%	42%	*50%*
In Vehicle	0	2%	**98**%	*98%*
A_{avg}				*77%*

As a result, we explore low-cost and light-weight methods that allow us to distinguish the *relative Stationary* from other activity classes. From our previous work [35] on study the correlation between bandwidth to cellular signal strength, we observed that the absolute signal deviation for *Stationary* (of both types) is significantly different from other activity classes.

To construct a suitable model for recognizing *Stationary*, we fed our sampling data to WEKA [36] and compared four typical machine learning techniques (i.e., SVM, ANN, Decision Tree, and Naive Bayesian). The results are shown in Table 6. Based on these results, we decided to develop our recognition algorithm using SVM. In our Android implementation, we use the *libSVM* library [37] to construct the model and recognition algorithm.

In the training phase, we divide our training data (collected every second) into 15 s sampling windows; that is, the sample data $S = (s_1, \ldots, s_{15})$. In the literature, sampling windows for activity recognition based on cellular signal strength are usually set to 15 s [38]. As demonstrated in our previous work [35], cellular signals can change dramatically due to many factors. Our experiments confirm 15 s is reasonable sampling window length for this type of activity recognition. The conventional features depicted in Table 7, such as mean $\bar{s}$, variance var, standard deviation σ, mean of absolute deviation $|D|$, and mean cross rate mcr that is the rate of signal crossovers the mean signal, make up our feature vector $V_{feature} = \{\bar{s}, var, \sigma, |D|, mcr\}$.

To construct the model, we find the hyperplane $\mathbf{w} \cdot x_i + b = 0$ that maximizes the margin between all $V_{feature}$ of different classes by optimizing the Quadratic Programming problem:

$$\begin{aligned} &arg \min_{\mathbf{w},\xi,b} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i \right\} \\ &s.t.: \; y_i(\mathbf{w}x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \end{aligned}$$

where x_i is the $V_{feature}$, $\mathbf{w}$ and b adjusts the orientation and the offset of hyperplane. The C parameter controls over-fitting and tolerance on the degree of false classification ξ_i for each sample.

4.2.1. Accuracy of ARsignal

We conduct experiments to evaluate the recognition accuracy for different activity classes. From the experiment results, we found that ARsignal is only able to recognize limited activity classes that include *Stationary*, *On Foot* and *In Vehicle* with accuracy above 50%, thus a smaller confusion matrix is created for ARsignal, compared with ARshell. Table 8 shows the results of these three activity classes. As also demonstrated in the literature [38], using cellular signal alone fails to distinguish all the supported activities that we discussed. However, it will be a good complementary solution to improve the recognition accuracy for detecting *Stationary*.

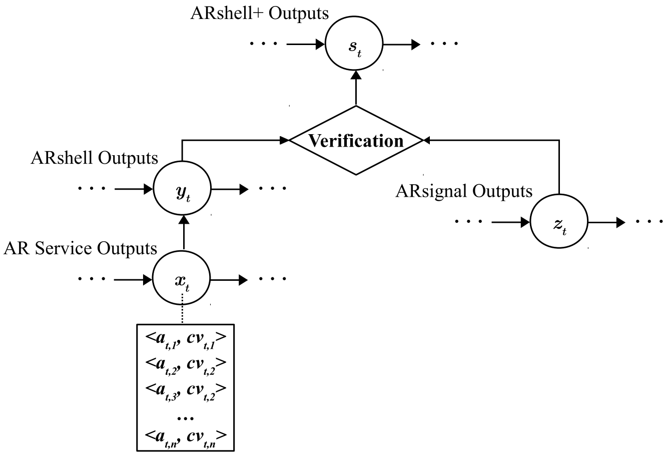


Fig. 7. Modeling of activity transition in ARshell+.

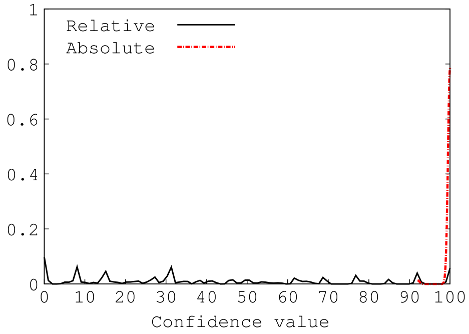


Fig. 8. Confidence value distribution of two types of *Stationary*.

4.3. ARshell+

In this subsection, we propose ARshell+, which combines ARshell and ARsignal to improve the recognition accuracy of the AR service.

Fig. 7 shows the processes of ARshell+. When the Google AR service reports a recognition result x_t (that is a list of probable activities) for time t, ARshell applies the Markov model to smooth the result and generates y_t. ARsignal will maintain 15 s of cellular signal data in a moving sampling window, and it generates recognition result z_t based on this data. A verification process checks if $z_t \neq Stationary$, then $s_t = y_t$. Otherwise, we first find the $\langle a_{t,i}, cv_{t,i} \rangle \in x_t$, such that $y_t = a_{t,i}$. If the corresponding $cv_{t,i} \geq \gamma$, then the confidence of this prediction is quite high. As a result, we set $s_t = y_t$. Otherwise, $s_t = Stationary$ if $cv_{t,i} < \gamma$.

Regarding *Stationary*, the Google AR service will not distinguish between absolute and relative stationary. We investigate this type of activity further by analyzing the distribution of confidence value in a set of experiments. Fig. 8 shows the distribution of confidence value for each *Stationary* type. The results highlight that the confidence value for relative *Stationary* is roughly a uniform distribution, while for absolute *Stationary* almost all confidence values are concentrated at 100, except three outliners (at $cv = 92$ and $cv = 98$). We conjecture that these outliners are caused by small movement of the table during the experiments. We use the observation that absolute *Stationary* typically has confidence value of 100 with very high probability. We incorporate this heuristic in ARshell+ to distinguish the two sub-classes of stationary: (i) absolute *Stationary*—the device has been put on a stationary structure, (ii) relative *Stationary*—the device is on a user who is sitting or standing. We conducted a separate set of experiments to confirm the accuracy of this contribution. From our results, we observe the recognition accuracy of 92.8% for relative stationary and 98.4% for absolute stationary. This process only applies to the result when the AR service reports *Stationary*, and it is optional for software developers. In addition, ARshell+ is able to fill up the delay between two consecutive recognitions of the Google AR service with opportunistic predictions according to the domain knowledge applied to the Markov smoother in ARshell, and provides continuous recognition results.

Table 9 shows the accuracy of ARshell+. A_{diff} column shows the improvement over the Google AR service. The accuracy of recognizing *Stationary* is not as high as using ARsignal, due to the misclassifications (with high confidence value) by the Google AR service. However, it should note that ARshell+ achieve 34% improvement over the Google AR service.

In addition to recognition accuracy, we investigate the response time when an activity transition happens. This time indicates how fast can ARshell+ recognize a change of the activity. As discussed in Section 4.1, the response for the first three

Table 9
Confusion matrix and accuracy of ARshell+.

	Classified to:								A_{aggr}	A_{diff}
	S	W	R	F	C	V	T	U		
S	**73**%	0	2%	4	9%	7%	0	5%	*73%*	↑*34%*
W	0	**96**%	0	**0**	0	0	0	4%	*96%*	↑*15%*
R	0	0	**52**%	**46**%	0	0	0	2%	*98%*	↑*25%*
C	0	0	0	0	**92**%	0	6%	2%	*92%*	↑*24%*
V	2%	0	0	0	0	**98**%	0	0	*98%*	↑*10%*
A_{avg}									*91%*	

Note: S—Stationary, W—Walking, R—Running, F—On Foot C—Cycling, V—In vehicle, T—Tilting, U—Unknown.

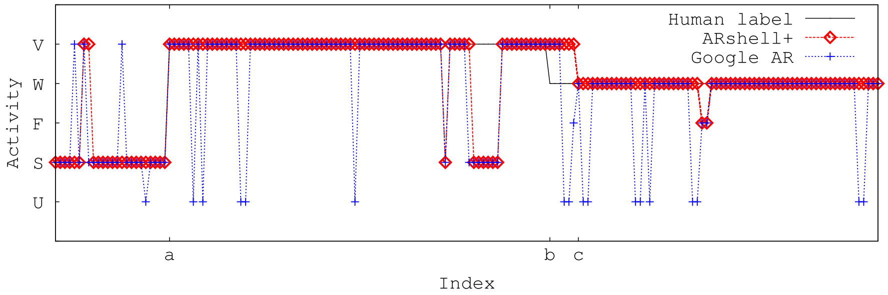


Fig. 9. Example of activity transition.

types of activity transition should be immediate. The last type of transition can be analyzed in two scenarios: (i) the most probable activity reported by the AR service is with a confidence value over γ, then the response should be immediate, and (ii) the AR service reports correct recognitions with the confidence value under γ, and the response time is one delay before receiving another update (around 3.5 s with the probability over 80%, as discussed in Section 3.2). Fig. 9 shows these two examples: (i) at *Index a*, from *Stationary* to *In Vehicle*, and (ii) at *Index b*, from *In Vehicle* to *Walking*. The region between *Index b-c* indicates an absolute delay of six updates. During this period, the Google AR service was not able to correctly recognize the sudden change, therefore ARshell suffers the delay too. However, ARshell+ reflects the change once the Google AR service generates a correct recognition.

5. Cost analysis: CPU, memory and power consumption

The evaluation results of the proposed solutions confirm the improvement in recognition accuracy. In this section, we evaluate the efficiency of the proposed solutions, by measuring the cost (i.e., CPU load, memory, and power consumption) of running the these solutions. To get a more accurate measurements, we use the ADB (Android Debug Bridge[3]) to extract information about the CPU load and memory.

5.1. *CPU usage*

The CPU load is an important efficiency indication of an algorithm. To measure the CPU usage, we stop all unnecessary background processes and mobile applications. Then, we measure the CPU usage of each solution during their run. Fig. 10 shows the CDF of CPU usage of different solutions while they are running. As shown in the figure, the CPU load of all proposed solutions is almost negligible. Among all the corresponding experiments, more than 94% of tests confirm that the proposed solutions do not increase CPU load. Less than 1% of tests show an increase of 1% in CPU usage.

5.2. *Memory occupancy*

Memory occupancy is another important indication of efficiency. To evaluate the memory usage of the proposed solutions, we evaluate two important indicators *RSS* and *VSS* measured in number of *pages*,[4] which is defined as a block of memory addresses. RSS (Resident Set Size) is the portion of memory occupied by a process that is held in the physical memory, while VSS (Virtual Set Size) is the virtual memory occupied by a process in total, including all types of memory (such as RAM, swapped out).

[3] http://developer.android.com/tools/help/adb.html.

[4] http://processors.wiki.ti.com/index.php/Android_Memory_Analysis.

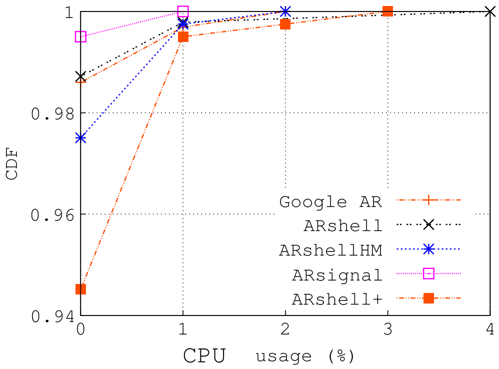
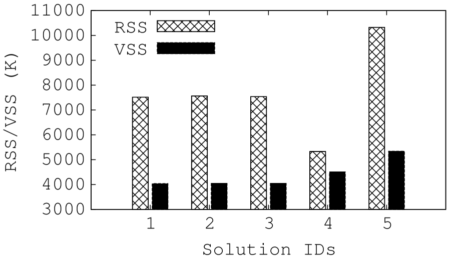


Fig. 10. CDF of CPU usage of different solutions.



(a) RSS/VSS increment.

Solution (ID)	*RSS/VSS*
Google AR (1)	7516/4040
ARshell (2)	7564/4044
ARshell HMM (3)	7540/4044
ARsignal (4)	5328/4508
ARshell+ (5)	10324/5340

(b) Details of the increment.

Fig. 11. Analysis of RSS/VSS increment (K).

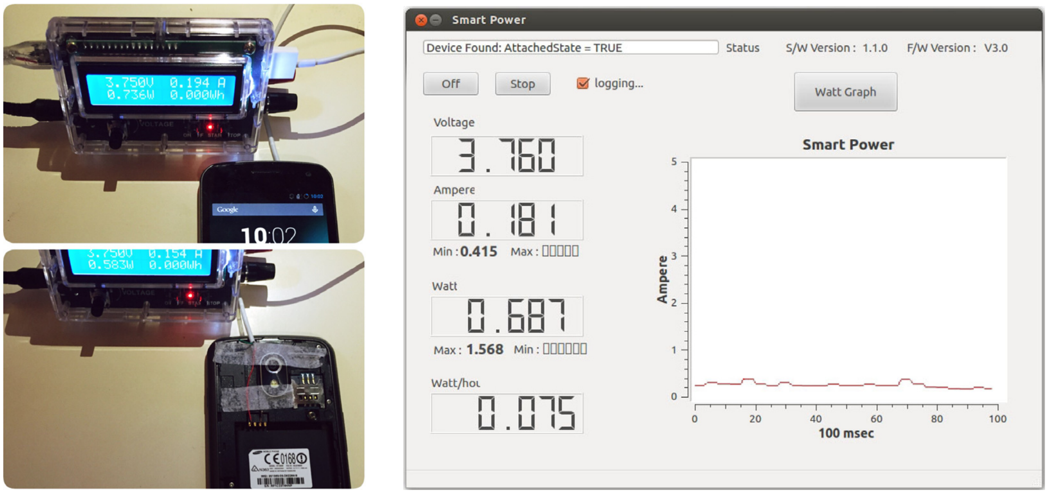
We conduct multiple experiments and measure the results for a detailed analysis. Results are averaged across multiple experiment runs. We measure our benchmark (RSS = 39 640 K and VSS = 505 176 K, respectively) with the bare system, without unnecessary applications running. Then, we measure the memory usage of the proposed solutions. Fig. 11 show the increase of each solution on top of the benchmark. From these figures, we can see that when the Google AR service is running, the RSS and VSS have increased to 7516 K and 4040 K, respectively. Because both ARshell and its HMM version apply improvement based on the Google AR service, we observe minor increase in memory usage. Interestingly, ARsignal uses less memory since its algorithm only require signal strength values that are already collected by the phones. No additional sensor data is collected for this solution. Finally, ARshell+ shows an increase in memory usage of 2760 K for RSS and 1296 K for VSS compared with ARshell. This increase is due the fact that ARshell+ combines ARsignal and ARshell. Various constructs defined in ARsignal requires memory to maintain the state information. However, the increase in memory is still reasonable. These increases represent approximately 6.9% and 0.25% over the benchmark respectively.

5.3. *Power consumption*

To measure the power consumption, we remove the battery from the phone and connect the positive and negative electrodes directly to a power meter,[5] as shown in Fig. 12(a). An application, as shown in Fig. 12(b) that comes with the power meter is used to record the power consumption of all proposed solutions. Similar approach was used in [39] in which authors use the National Instruments PCI-MIO-16E-4 sampling board to measure the voltage across the phone battery and the voltage drop across our measurement resistor.

To ensure that the evaluation only measures the power consumption related to the Google AR service and our approaches, we first check the power consumption of the bare system, without any unnecessary applications and services running. The results show that the bare system consumes 0.648 W. Then, we measure the power consumption of the Google AR service running on the bare system, and we found that it consumes 0.744 W. Since our proposed solutions are based on the Google AR service, we use its performance as benchmark for comparing the proposed solutions. Table 10 shows the averaged difference in power consumption over the Google AR service. In general, the proposed solutions slightly increase the power consumption, except ARsignal. We conjecture that ARsignal makes use of information (cellular signal strength) already collected by the phone, rather than gathering data from the on-board sensors. Similar argument as in the memory usage section, ARshell+ combines ARsignal on top of ARshell. Therefore, it increases the power consumption by 0.12 W on top of ARshell. We argue that this increase during the recognition run is still acceptable.

[5] https://odroid.com/dokuwiki/doku.php?id=en:odroidsmartpower.



Device Found: AttachedState = TRUE

Status S/W Version : 1.1.0 F/W Version : V3.0

☑ logging...

Voltage 3.760

Ampere 0.181

Min : 0.415 Max : □□□□□

Watt 0.687

Max : 1.568 Min : □□□□□□

Watt/hou 0.075

(a) Smartphone and power meter.

(b) PC side application.

Fig. 12. Experimental setup of battery consumption.

Table 10
Averaged difference in battery consumption (Watt).

	Increment
ARshell	+0.12
ARshell HMM	+0.14
ARsignal	−0.24
ARshell+	+0.24

5.4. *Prototype and API of ARshell+*

A proof of concept prototype has been developed on Android devices. In the current implementation, ARshell+ runs as a background daemon, and it tries to improve every output generated by the Google AR service. As ARshell+ is based on the AR service, it inherits the warm-up time. ARshell+ provides two mechanisms for delivery of the activity recognition results: (i) a pull-based mechanism that replies with the latest activity when requested, supporting request up to every second, and (ii) a notification-based mechanism that updates the recognition result per interval; notifications are sent when an activity change is detected or per interval depending on how ARshell+ is configured.

We also streamline the API, so that software developers can easily integrate ARshell+ into their existing projects. With our ARshell+ API, only few lines of code are required for adding/linking the activity recognition functionalities. ARshell+ also supports backward compatibility with the AR service should developers need access to the original AR data. Both ARshell and ARshell+ are available to the public as open-source projects on GitHub.[6]

6. Conclusion

In this paper, we shared the lesson learnt from the systematic analysis of the Android Activity Recognition service. Based on our findings, we proposed and evaluated four different practical solutions for smartphones that significantly enhance the activity recognition accuracy with the overall average improvement of 21%. We evaluated these proposed solutions regarding CPU usage, memory occupancy, and power consumption. Compared with API of the Google AR Service, the proposed ARshell+ offers a streamline, easier to use, API for mobile app developers to incorporate activity recognition into their existing projects or applications. It supports both pulling and notification based mechanisms for receiving recognition results. We prototyped ARshell+ and released it as an open source project for the mobile app development community.

Acknowledgments

NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

[6] https://github.com/myzhong/ARshell.

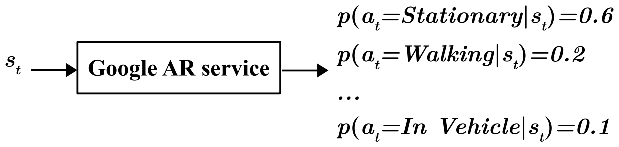

s_t
Google AR service
$p(a_t{=}Stationary|s_t){=}0.6$
$p(a_t{=}Walking|s_t){=}0.2$
...
$p(a_t{=}In\ \ Vehicle|s_t){=}0.1$

Fig. 13. Example of posterior distribution given by Google AR service.

Appendix. Mapping ARshell HMM to HMM

In the Hidden Markov model, the variables include hidden states and observations. As shown in Fig. 13, HMM models the joint distribution of those variables by making Markov assumptions that current latent state a_t (for $t = 0, \ldots, T$) only depends on previous latent state a_{t-1}, and current observation s_t only depends on current latent state a_t. The former dependency can be described by transition probability $p(a_t|a_{t-1})$ while the later can be captured by emission probability $p(s_t|a_t)$, and then joint distribution can be formulated as follows:

$$p(\boldsymbol{s}, \boldsymbol{a}) = p(a_0)p(s_0|x_0)\prod_{t=1}^{T} p(a_t|a_{t-1})p(s_t|a_t). \tag{5}$$

In our case, the observations s are the readings produced by the on-board sensors at different time points, while latent states a are the undergoing activities that we want to recognize. Note that the dependencies among the latent states can be used to characterize the temporal relationships among human activities. Specifically, assigning large values to self-transition probabilities encourages the current activity to be continued with high probability.

Central to HMM are the parameters (i.e. transition and emission probabilities) that can be used to infer the latent activities given a sequence of current observations. The transition probability is similar to that of Markov Model in ARshell (described in Section 4.1). However, as for the emission probability, this is a non-trivial work, since we do not have access to the observations (sensor readings) encapsulated in Google AR service that conceals the underlying logic of how the sensor readings are processed. Fortunately, by treating the Google AR service as a black box, the emission probability can be approximated with the predictions given by the Google AR service, formulated as follows:

$$p(s_t|a_t) = \frac{p(a_t|s_t)p(s_t)}{p(a_t)} \propto p(a_t|s_t) \tag{6}$$

where prior knowledge $p(a_t)$ is identical for different activities, because we balance the training data over all the activity classes. The variable s_t is the observation at time t, and $p(s_t)$ is a constant when calculating its evidence against different classes. Therefore, the emission probability is proportional to the posterior probability given by the Google AR service (shown in Fig. 13), and the joint distribution can be re-formulated as follows:

$$p(\boldsymbol{s}, \boldsymbol{a}) \propto p(a_1)p(a_0|s_0)\prod_{t=1}^{T} p(a_t|a_{t-1})p(a_t|s_t). \tag{7}$$

Finally, inferring the hidden states is equivalent to finding the sequences that maximize the joint probability depicted in Eq. (7), which is performed by the Viterbi algorithm as specified in Section 4.1.3.

References

[1] Cisco, Cisco visual networking index: Global mobile data traffic forecast update 2014–2019 white paper.
[2] B. Guo, D. Zhang, Z. Yu, Y. Liang, Z. Wang, X. Zhou, From the Internet of things to embedded intelligence, World Wide Web 16 (4) (2013) 399–420.
[3] L. Chen, J. Hoey, C.D. Nugent, D.J. Cook, Z. Yu, Sensor-based activity recognition, IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. 42 (6) (2012) 790–808.
[4] B. Logan, J. Healey, M. Philipose, E. Munguia, S. Intille, A long-term evaluation of sensing modalities for activity recognition, in: Proc. of Ubiquitous Computing, Innsbruck, Austria, 2007.
[5] T. Nguyen, D. Phung, S. Gupta, S. Venkatesh, Extraction of latent patterns and contexts from social honest signals using hierarchical dirichlet processes, in: Proc. of IEEE International Conference on Pervasive Computing and Communications, San Diego, CA, USA, 2013, pp. 47–55.
[6] M. Stikic, T. Huynh, K. Van Laerhoven, B. Schiele, Adl recognition based on the combination of rfid and accelerometer sensing, in: Proc. of the second IEEE International Conference on Pervasive Computing Technologies for Healthcare, Tampere, Finland, 2008, pp. 258–263.
[7] S. Lee, D. Ahn, S. Lee, R. Ha, H. Cha, Personalized energy auditor: Estimating personal electricity usage, in: Proc. of IEEE International Conference on Pervasive Computing and Communications, Budapest, Hungary, 2014, pp. 44–49.
[8] J.R. Kwapisz, G.M. Weiss, S.A. Moore, Activity recognition using cell phone accelerometers, ACM SigKDD Explor. Newslett. 12 (2) (2011) 74–82.
[9] T. Maekawa, Y. Yanagisawa, Y. Kishino, K. Ishiguro, K. Kamei, Y. Sakurai, T. Okadome, Object-based activity recognition with heterogeneous sensors on wrist, in: Pervasive Computing, Springer, 2010, pp. 246–264.
[10] J. Lester, T. Choudhury, N. Kern, G. Borriello, B. Hannaford, A hybrid discriminative/generative approach for modeling human activities, in: Proc. of International Joint Conference on Artificial Intelligence, Vol. 5, Edinburgh, Scotland, UK, 2005, pp. 766–772.
[11] M. Zhong, J. Wen, P. Hu, J. Indulska, Advancing android activity recognition service with markov smoother, in: Proc. of the 11th Workshop on Context and Activity Modeling and Recognition, PerCom Workshop, St. Louis, Missouri, USA, 2015.
[12] N.C. Krishnan, D.J. Cook, Activity recognition on streaming sensor data, Pervasive Mob. Comput. 10 (2014) 138–154.
[13] L. Bao, S.S. Intille, Activity recognition from user-annotated acceleration data, in: Pervasive Computing, Springer, 2004, pp. 1–17.

[14] M. Quwaider, S. Biswas, Body posture identification using hidden markov model with a wearable sensor network, in: Proceedings of the ICST 3rd International Conference on Body Area Networks, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, p. 19.
[15] S. Wang, W. Pentney, A.-M. Popescu, T. Choudhury, M. Philipose, Common sense based joint training of human activity recognizers, in: Proc. of International Joint Conference on Artificial Intelligence, Vol. 7, Hyderabad, India, 2007, pp. 2237–2242.
[16] P. Palmes, H.K. Pung, T. Gu, W. Xue, S. Chen, Object relevance weight pattern mining for activity recognition and segmentation, Pervasive Mob. Comput. 6 (1) (2010) 43–57.
[17] T. Choudhury, S. Consolvo, B. Harrison, J. Hightower, A. LaMarca, L. LeGrand, A. Rahimi, A. Rea, G. Bordello, B. Hemingway, et al., The mobile sensing platform: An embedded activity recognition system, IEEE Pervasive Comput. 7 (2) (2008) 32–41.
[18] L. Wang, T. Gu, X. Tao, J. Lu, A hierarchical approach to real-time activity recognition in body sensor networks, Pervasive Mob. Comput. 8 (1) (2012) 115–130.
[19] A. Pantelopoulos, N.G. Bourbakis, A survey on wearable sensor-based systems for health monitoring and prognosis, IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. 40 (1) (2010) 1–12.
[20] M. Shoaib, H. Scholten, P.J. Havinga, Towards physical activity recognition using smartphone sensors, in: Proc. of IEEE UIC/ATC 2013, Vietri sul Mare, Italy, 2013, pp. 80–87.
[21] S. Dernbach, B. Das, N.C. Krishnan, B.L. Thomas, D.J. Cook, Simple and complex activity recognition through smart phones, in: Proc. of the 8th International Conference onIntelligent Environments, Guanajuato, Mexico, 2012, pp. 214–221.
[22] N.D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, A.T. Campbell, A survey of mobile phone sensing, IEEE Commun. Mag. 48 (9) (2010) 140–150.
[23] S. Hemminki, P. Nurmi, S. Tarkoma, Accelerometer-based transportation mode detection on smartphones, in: Proc. of the 11th ACM Conference on Embedded Networked Sensor Systems, Rome, Italy, 2013, p. 13.
[24] T. Maekawa, S. Watanabe, Unsupervised activity recognition with user's physical characteristics data, in: Proc. of IEEE International Symposium on Wearable Computing, San Francisco, CA, USA, 2011, pp. 89–96.
[25] B. Cvetkovic, B. Kaluza, M. Luštrek, M. Gams, Semi-supervised learning for adaptation of human activity recognition classifier to the user, in: Proc. of Workshop on Space, Time and Ambient Intelligence, IJCAI, Bellevue Washington, USA, 2011, pp. 24–29.
[26] D. Gordon, J. Czerny, T. Miyaki, M. Beigl, Energy-efficient activity recognition using prediction, in: Proc. of IEEE International Symposium on Wearable Computing, Osaka, Japan, 2012, pp. 29–36.
[27] Z. Yan, V. Subbaraju, D. Chakraborty, A. Misra, K. Aberer, Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach, in: Proc. of IEEE International Symposium on Wearable Computing, Osaka, Japan, 2012, pp. 17–24.
[28] P. Zappi, C. Lombriser, T. Stiefmeier, E. Farella, D. Roggen, L. Benini, G. Tröster, Activity recognition from on-body sensors: accuracy-power trade-off by dynamic sensor selection, in: Proc. of the 5th European Conference on Wireless Sensor Networks, Pittsburgh, PA, 2008, pp. 17–33.
[29] M. Keally, G. Zhou, G. Xing, J. Wu, A. Pyles, Pbn: towards practical activity recognition using smartphone-based body sensor networks, in: Proc. of ACM Conference on Embedded Networked Sensor Systems, Seattle, WA, 2011, pp. 246–259.
[30] S. Kang, J. Lee, H. Jang, H. Lee, Y. Lee, S. Park, T. Park, J. Song, Seemon: scalable and energy-efficient context monitoring framework for sensor-rich mobile environments, in: Proceedings of the 6th ACM SIGMOBILE International Conference on Mobile Systems, Applications, and Services, Breckenridge, Colorado, 2008, pp. 267–280.
[31] Y. Ju, C. Min, Y. Lee, J. Yu, J. Song, An efficient dataflow execution method for mobile context monitoring applications, in: Proc. of IEEE International Conference on Pervasive Computing and Communications, Lugano, Switzerland, 2012, pp. 116–121.
[32] T. Maekawa, Y. Kishino, Y. Sakurai, T. Suyama, Activity recognition with hand-worn magnetic sensors, Pers. Ubiquitous Comput. 17 (6) (2013) 1085–1094.
[33] D. Wyatt, M. Philipose, T. Choudhury, Unsupervised activity recognition using automatically mined common sense, in: Proc. of National Conference of the American Association for Artificial Intelligence, Vol. 5, Pittsburgh, Pennsylvania, 2005, pp. 21–27.
[34] L. Rabiner, A tutorial on hidden markov models and selected applications in speech recognition, Proc. IEEE 77 (2) (1989) 257–286.
[35] M. Zhong, P. Hu, J. Indulska, Revisited: Bandwidth estimation methods for mobile networks, in: Proc. of the 8th IEEE Workshop on Autonomic and Opportunistic Communications, WoWMoM Workshop, Sydney, Australia, 2014.
[36] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The weka data mining software: an update, ACM SIGKDD Explor. Newslett. 11 (1) (2009) 10–18.
[37] C.-C. Chang, C.-J. Lin, Libsvm: a library for support vector machines, ACM Trans. Intell. Syst. Technol. (TIST) 2 (3) (2011) 27.
[38] I. Anderson, H. Muller, Practical activity recognition using gsm data.
[39] A. Rice, S. Hay, Decomposing power measurements for mobile devices, in: Proc. of IEEE International Conference on Pervasive Computing and Communications, Mannheim, Germany, 2010, pp. 70–78.