PBN: Towards Practical Activity Recognition Using Smartphone-Based Body Sensor Networks*

Matthew Keally
Dept. of Computer Science
College of William and Mary
Williamsburg, VA 23187, USA
makeal@cs.wm.edu

Gang Zhou
Dept. of Computer Science
College of William and Mary
Williamsburg, VA 23187, USA
gzhou@cs.wm.edu

Guoliang Xing
Dept. of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824, USA
glxing@msu.edu

Jianxin Wu
School of Computer Engineering
Nanyang Technological University
Singapore 639798
jxwu@ntu.edu.sg

Andrew Pyles
Dept. of Computer Science
College of William and Mary
Williamsburg, VA 23187, USA
ajpyles@cs.wm.edu

Abstract

The vast array of small wireless sensors is a boon to body sensor network applications, especially in the context awareness and activity recognition arena. However, most activity recognition deployments and applications are challenged to provide personal control and practical functionality for everyday use. We argue that activity recognition for mobile devices must meet several goals in order to provide a practical solution: user friendly hardware and software, accurate and efficient classification, and reduced reliance on ground truth. To meet these challenges, we present PBN: Practical Body Networking. Through the unification of TinyOS motes and Android smartphones, we combine the sensing power of on-body wireless sensors with the additional sensing power, computational resources, and user-friendly interface of an Android smartphone. We provide an accurate and efficient classification approach through the use of ensemble learning. We explore the properties of different sensors and sensor data to further improve classification efficiency and reduce reliance on user annotated ground truth. We evaluate our PBN system with multiple subjects over a two week period and demonstrate that the system is easy to use, accurate, and appropriate for mobile devices.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Body Sensor Networks, Mobile Phones, Motes, Machine Learning, Sensing, Activity Recognition

1 Introduction

The low cost and wide availability of wireless sensors makes body sensor networks (BSNs) an increasingly attractive solution for a wide range of applications such as personal health care monitoring [3], physical fitness assessment [1], and context awareness [19] [4]. In all of these applications, and especially in the context awareness and activity recognition domain, the need exists to provide personal control and direct application feedback to the user. For example, a sensing application for activity recognition allows user configuration of the number and placement of sensor nodes as well as runtime notification as to the current activity. Complete control over a body sensor network allows a user to tailor the application to his or her needs as well as exercise discretion over the use and dissemination of activity inferences.

For context aware and activity recognition applications, the sensing capability of multiple on-body sensor nodes is difficult to capture through the use of smartphones alone [17]. However, the portability, computational power, and interface of a mobile phone can provide the user personal control and feedback over the sensing application. We envision a practical solution to activity recognition for mobile devices which is entirely portable, under direct control of the user, computationally lightweight, and accurate.

In this paper, we focus on four major challenges for providing practical activity recognition with mobile devices. First, the hardware and software platform must be user-friendly. The hardware must be portable and easily configurable, while the software must provide an intuitive control interface with adequate system feedback. Second, classi-

*This work is supported in part by NSF grants ECCS-0901437, CNS-0916994, and CNS-0954039.

SenSys'11, November 1–4, 2011, Seattle, WA, USA.

fication must be performed accurately, handling both easy and difficult to classify activities, environmental changes, and noisy data. Accurate classification must also allow context switching between activities without extensive parameter tuning and different, complex classifiers for each activity. Third, since mobile hardware is often constrained in terms of computation power and energy, classification must be performed efficiently and redundant sensing resources must be timely turned off. Lastly, the system must have a reduced reliance on ground truth, for regular collection and labeling of sensor data can be invasive to the end user.

While there is significant existing work in the mobile activity recognition domain, most approaches do not offer practical solutions that address the above challenges. Some approaches [9] [15] [28] provide multiple on-body sensor nodes but do not provide a portable interface, such as a mobile phone, for the end user to control sampling, configure hardware, or receive real time feedback. Other approaches [18] [19] rely on computationally expensive learning algorithms and a back end server. Still more works [16] [17] rely on specific sensing models for each sensor modality, making sensor collaboration difficult. Lastly, other works [27] [23] do not provide online training for adaptation to *body sensor network (BSN) dynamics*. Compared to static sensor networks, body sensor network dynamics include the changing geographical location of the user, user biomechanics and variable sensor orientation, as well as background noise.

Towards addressing these challenges of activity recognition, we propose PBN: Practical Body Networking. PBN consolidates the flexibility and sensing capability of TinyOS-based motes with the additional sensing power, computational resources, and user-friendly interface of an Android smartphone. Our solution can also be extended beyond TinyOS motes to combine Android with a wide variety of USB devices and wireless sensors. Through the use of ensemble learning, which automates parameter tuning for BSN dynamics, PBN provides a capable, yet lightweight activity recognition solution that does not require a backend server.

With PBN, we provide an online training solution which detects when retraining is needed by analyzing the information divergence between training and runtime data distributions and integrating this analysis with the ensemble classifier. In this way, PBN determines when retraining is needed without the need to request ground truth from the user. Furthermore, we investigate the properties of sensors and sensor data to identify sensors which are accurate and have diverse classification results. From this analysis, we are able to prevent the ensemble classifier from needlessly consuming computational overhead by using redundant sensors in the online training process. Our main contributions are:

- We combine the sensing capability of on-body TinyOS-based motes with the sensors, computational power, portability, and user interface of an Android smartphone.
- An activity recognition approach appropriate for low-power wireless motes and mobile phones that does not rely on a backend server. Our approach handles BSN dynamics without sophisticated parameter tuning and also accurately classifies difficult tasks.
- We provide retraining detection without requesting ground truth from the user, reducing the invasiveness of the system.
- We reduce online training costs by detecting redundant sensors and excluding them from the ensemble classifier.
- With two weeks of data from two subjects, we demonstrate that we can detect even the most difficult activities with nearly 90% accuracy. We identify 40% of sensors as redundant, excluding them from online training.

The rest of this paper is organized as follows: Section 2 presents related work, Section 3 provides an overview of our PBN system design, and Section 4 describes the ensemble classifier. We describe our retraining detection method in Section 5, provide details on how to collect new training data in Section 6, and present a sensor selection method in Section 7. In Section 8, we evaluate our PBN platform and present conclusions and future work in Section 9.

2 Related Work

Many methods perform classification with multiple on-body sensor nodes but have no mobile, on-body aggregator for sensor control and activity recognition feedback. Some works [9] [15] [28] [22] use multiple on-body sensor motes to detect user activities, body posture, or medical conditions, but such motes are only used to collect and store data with analysis performed offline. Such approaches limit user mobility due to periodic communication with a fixed base station and also lack real time analysis and feedback to the user.

Other approaches use mobile phones for sensing and classification, but require the use of backend servers or offline analysis to train or update classification models. The authors of [18] provide model training with a short amount of initial data, but model updating is performed using a backend server. In [19], model training and classification is split between lightweight classifiers on a mobile phone and more powerful classifiers on a server. The authors of [21] present a sensor mote with an SD card attachment for interface with a mobile phone but do not provide an application which uses such a device. Mobile phone like hardware is used in [7] to perform pothole detection, but data is analyzed offline.

Some works use a limited set of sensor modalities or use a separate classifier for each sensing modality, making classification difficult for deployments with a large number of heterogeneous sensors. Some works [2] [12] focus extensively on energy aware sensing models specifically for localization or location tracking. In [16], while the authors provide an adaptive classification approach, they only make use of a mobile phone microphone to recognize user context and activities, thus eliminating a wide range of user activities that are not sound dependent. The authors of [17] provide a component-based approach to mobile phone based classification for different sensors and different applications, but each sensor component requires a separate classifier implementation. One approach uses both motes and mobile phones to perform fitness measurement [6], but simple sensing models are used that will not work for more general activity recognition applications. Activity recognition and energy expenditure is calculated in [1] using an accelerometer-specific sensing model for on-body wireless nodes.

Lastly, several activity recognition methods do not provide online training to account for environmental dynamics or poor initial training data. The authors of [14] use AdaBoost to perform activity recognition but use custom hardware with very high sampling rates. In [20], the authors also use AdaBoost for activity recognition with mobile phones, but focus mainly on ground truth labeling inconsistencies and do not provide a practical system for long term use. The authors of [27] focus on duty cycling mobile phone sensors to save energy, but provide a rigid rule-based recognition model that must be defined before runtime. A speaker and emotion recognition system using mobile phones is presented in [23] which implements adaptive sampling rates but the classification models used are trained offline.

3 System Overview and Architecture

In this section, we first present the application requirements. We then present our PBN hardware and application, which unifies TinyOS and Android. Next, we describe our PBN architecture and finally describe our PBN experimental setup which we refer to for the remainder of the paper.

3.1 Application Requirements

Our PBN system design is motivated by the requirements of a practical body networking application for activity recognition. Data from multiple on-body sensors is reported to a mobile aggregator which makes classification decisions in real time. The system must be able to accurately and efficiently classify typical daily activities, postures, and environmental contexts, which we present in Table 1. Despite these categories being common for many individuals, previous work [14] has identified some of them, such as cleaning, as especially difficult to classify.

Table 1: PBN Classification Categories.

Environment	Indoors, Outdoors
Posture	Cycling, Lying Down, Sitting, Standing, Walking
Activity	Cleaning, Cycling, Driving, Eating, Meeting, Reading, Walking, Watching TV, Working

From the table, we break down our target classifications into three categories: Environment, Posture, and Activity. With the Environment and Posture categories, we can provide insight into the physical state of the user for personal health and physical fitness monitoring applications. We also wish to measure typical activities in which a user engages, such as watching TV, driving, meeting with colleagues, and cleaning. Here, we consider cycling and walking to be both postures and activities. Such activity recognition is quite useful for participatory sensing and social networking applications since it eliminates the need for a user to manually update his or her activities online. The requirements to provide such a practical activity recognition system are:

- **User-friendly.** Different on-body commercial off-the-shelf (COTS) sensor nodes must seamlessly interact with a mobile phone aggregator for simple user configuration and operation. The hardware must be portable and easy to wear, while the software must provide an intuitive interface for adding, removing, and configuring different sensors geared to detect the user's intended activities. Labeling training data should also be a simple and non-invasive task, facilitated by the mobile phone.
- **Accurate classification.** The system must accurately handle both easy and difficult to detect activities as well as noisy data and environmental dynamics. The system must also account for the changing geographic location of the user as well as the variable orientation of the individual on-body sensors.
- **Efficient classification.** A body sensor network for activity recognition may often use low power hardware, therefore, the classification algorithm should be computationally efficient in addressing the BSN dynamics of geographic location, user biomechanics, and environmental noise. The system must also be energy efficient: by quantifying the contribution of each sensor towards accurately classifying activities, sensors with minimal contribution can be powered down. Furthermore, the system must avoid extensive parameter tuning as well as avoid unique, complex sensing models for each activity.
- **Less reliance on ground truth.** Activity recognition systems are often deployed with minimal labeled training data, thus the need exists to train a system in an online manner, requesting ground truth labels only when absolutely necessary. A reduced need for ground truth reduces the burden on the user to label training data.

3.2 PBN Hardware and Application

To achieve accurate and efficient activity recognition for mobile phones and on-body sensors, we provide an extensive hardware and software support system, which we describe in this section. In Section 3.2.1, we describe the implementation of USB host mode in the Android kernel to allow communication between a base station mote and an Android phone. In Section 3.2.2, we describe our Android application for configuration and activity recognition.

3.2.1 Unification of Android and TinyOS

Our PBN system consists of Crossbow IRIS on-body sensor motes and a TelosB base station connected to an Android HTC G1 smartphone via USB. While previous work has connected TinyOS motes and mobile phones, such efforts either use energy demanding Bluetooth [6], do not provide mobile phone support for TinyOS packets [24], or use special hardware [29]. Instead, we provide a seamless and efficient integration of TinyOS motes and Android smartphones. Our solution can be easily extended beyond the research-based TinyOS devices to work with a wide variety of commercial and more ergonomic USB and wireless sensors. Our challenges with such integration lie in 4 aspects: TinyOS sensing support, Android OS kernel support, Android hardware support, and TinyOS Java library support.

TinyOS Sensing Support. Illustrated in Figure 3, we implement a sensing application in TinyOS 2.x for Crossbow IRIS motes with attached MTS310 sensorboards. The sensor node application allows for runtime configuration of active sensors, sampling rates, and local aggregation meth-

ods. We develop a communication scheme to pass control and data packets through a TelosB base station connected to an Android smartphone.

Android OS Kernel Support. To prepare the phone to support external USB devices, a kernel EHCI or host controller driver is required. However, the USB host controller hardware documentation of the Google Developer phone is not publicly available. We incorporate the suggestions from [5] and modify the Freescale MPC5121 driver to work with the Qualcomm MSM7201A chipset on the Android phone. With these modifications, the host controller is able to recognize USB devices with a caveat: enabling the HOST controller disables the USB client mode for PC SD card access.

Hardware Support. The EHCI controller of the phone does not provide any power to external devices, such as the sensor motes in our case. To solve this limitation, we build a special USB cable that includes an external 5V battery pack with a peak load of 0.5A. The positive and ground cables are cut on the phone side such that only the USB device, not the phone, receives power. This also has the added benefit of not placing an extra load on the phone battery.

TinyOS Support on Android. Two challenges exist in providing Android TinyOS communication support: systems implementation and TinyOS software modifications. 1) Each mote has an FTDI USB serial chip that can be used to communicate with the host. The Linux FTDI driver creates a serial port device in the /dev directory. Android includes a minimal device manager that creates devices with insufficient privileges. This device manager is modified so that correct privileges are established. 2) TinyOS uses a binary serialization to communicate bidirectionally between the mote and host with the help of a C++ JNI interface. We modify the TinyOS JNI interface and associated Java libraries to compile and function on the Android platform. With such modifications, Android applications can send and receive TinyOS packets using the same Java interfaces available on a PC.

3.2.2 Android App

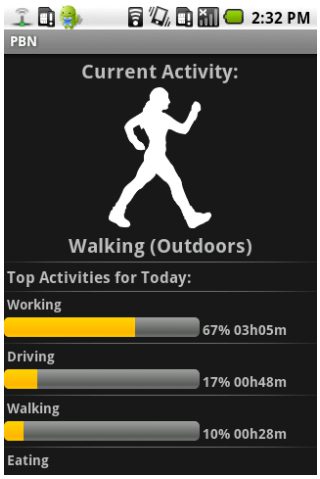


Figure 1: PBN activity status view.

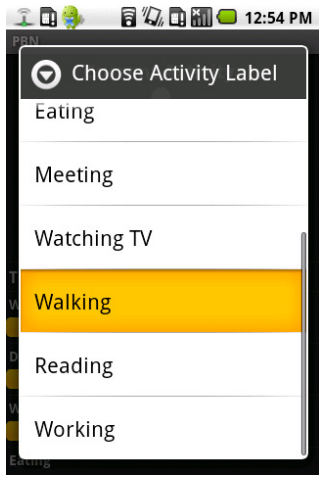


Figure 2: Ground truth logging.

To provide a user-friendly front end for PBN, we implement an Android app to allow for easy configuration, runtime deployment, ground truth labeling, and data storage and upload for offline analysis. The GUI component allows for user control of both phone and remote wireless sensors. The GUI also receives feedback as to the current activity and whether or not classifier retraining is needed, and also provides ground truth labels to the classifier.

Sensor Configuration. Our PBN Android app provides an interface to allow for easy configuration of both phone sensors as well as remote TinyOS sensors. Using our software interface, a user can easily add or remove TinyOS nodes as well as phone and TinyOS-based sensors. The user is also able to adjust sampling rates as well as local data aggregation intervals to reduce the number of radio transmissions. A user's sensor and sampling configuration can be stored on the phone in an XML file so that configuration must only be performed once. Users of different phones can also exchange saved sensor configurations.

Runtime Control and Feedback. With a created sensor configuration, a PBN user is able to quickly start and stop data sampling and activity recognition. As depicted in Figure 1, the current activity is displayed during runtime along with a configurable histogram of the most recent activities. When our classifier determines that accuracy is dropping and retraining with more labeled ground truth is needed, the PBN app prompts the user to input the current activity, illustrated in Figure 2. During retraining, an indicator and button on the current activity screen appears, allowing the user to log any changes in the ground truth. Labeled training data can be stored locally for later retraining or uploaded to a server for sharing and offline analysis.

3.3 PBN Architecture

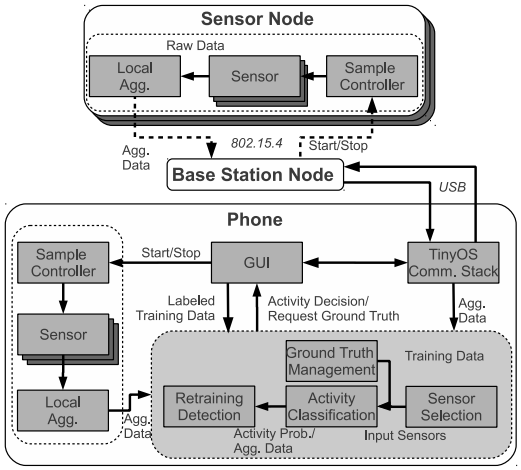


Figure 3: PBN System Architecture.

Illustrated in Figure 3, our Practical Body Networking (PBN) system resides solely on TinyOS-based motes and on an Android smartphone with no reliance on a backend server. Multiple TinyOS-based motes, each containing one or more sensors of different modalities, are attached on-body. Each mote communicates wirelessly (dotted lines) with a TinyOS mote base station, which is connected via USB (solid lines) to an Android smartphone.

Phone and mote sensors (white dotted areas) sample data at a user configured rate and aggregate data from each sensor over a larger aggregation interval. Aggregated data for each sensor is returned at each aggregation interval. For each TinyOS mote, aggregated data is returned wirelessly in a single packet. We provide a reliable communication scheme between the phone and motes and determine through our 2 week experiment that for most activities, packet loss rates are below 10%, with 99% of packets received after one retransmission, even when using the lowest transmission power.

Aggregated data from phone and mote sensors is fed into the PBN classification system (gray dotted area in Figure 3) at each aggregation interval to make a classification decision. The classifier, AdaBoost, is initially trained with the user labeling a short two minute period of each pre-defined activity (Ground Truth Management) but training can be updated online through Retraining Detection. During initial training and retraining, the Sensor Selection module reduces the training overhead incurred by AdaBoost by choosing only the most capable sensors for performing accurate classification. We now describe the core of our PBN platform:

Activity Classification. We use AdaBoost [8], an ensemble learning algorithm, as our activity recognition classifier which resides entirely on a smartphone. AdaBoost combines an ensemble of weak classifiers together to form a single, more robust classifier. With this approach, we are able to train weak classifiers for each sensor in our deployment and combine them together to recognize activities. AdaBoost is able to maximize training accuracy by selecting only the most capable sensors for use during runtime. We improve upon AdaBoost by providing online training and retraining detection as well as improve computational overhead through sensor selection.

Retraining Detection. Since initial training data may not be sufficient to account for body sensor network dynamics, AdaBoost retraining is needed in order to ensure high accuracy throughout the deployment lifetime. We investigate the discriminative power of individual sensors and from our analysis we are able to detect when retraining is needed during runtime without the use of labeled ground truth. For each sensor, we use Kullback-Leibler divergence [13] to determine when runtime data is sufficiently different from training data and use a consensus-based approach to initiate retraining when enough sensors indicate that retraining is needed.

Ground Truth Management. When retraining is needed, we investigate how much new data to collect and label in order to ensure BSN dynamics are captured, yet minimize the intrusiveness of the user manually annotating his or her current activities. We also determine how to maintain a balance of training data for each activity to ensure AdaBoost trains properly and provides maximum runtime accuracy.

Sensor Selection. During training, AdaBoost trains multiple weak classifiers for every sensor in the deployment, even if many sensors are never chosen by AdaBoost when training is complete. Through analysis of the theory behind ensemble learning, we identify both helpful and redundant sensors through the use of the Pearson correlation coefficient [25]. Based on our analysis, we provide a method to give AdaBoost only the sensors that provide a significant contribution towards maximizing accuracy, thus reducing online training overhead.

3.4 Experimental Setup

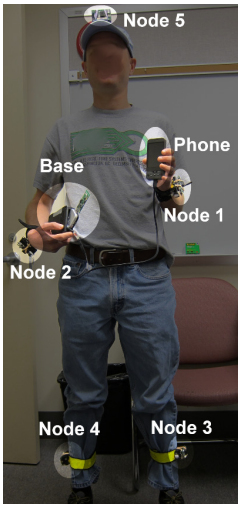


Figure 4: Subject 1.

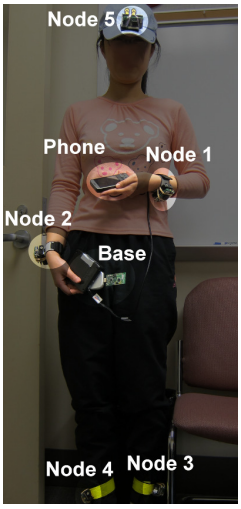


Figure 5: Subject 2.

For the remainder of the paper, we will refer to our activity recognition experiment, in which we collected two weeks of sensor data using two subjects, depicted in Figures 4 and 5. Each subject wore five Crossbow IRIS motes wirelessly linked to a TelosB base station and Android HTC G1 smartphone. The mote and sensor configuration for our experiment is summarized in Table 2. On the phone, which we attach to the waist, we make use of the 3-axis accelerometer as well as velocity from WiFi and GPS, with GPS active only when PBN detects the user is outdoors. On the mote, we use an MTS310 sensorboard with the following sensors: 2-axis accelerometer, microphone, light, and temperature. In addition to the sensors on the mote, the base station also collects RSSI information from each received packet, which has been previously demonstrated [22] to provide insight into body posture. During initial and online training, all sensors are active, while only sensors selected during training remain active during the remaining sampling periods.

Table 2: PBN Deployment Configuration.

Node	ID	Location	Sensors
Phone	0	R. Waist	3-Axis Acc., GPS/WiFi (velocity)
IRIS	1	L. Wrist	2-Axis Acc., Mic., Light, Temp.
IRIS	2	R. Wrist	2-Axis Acc., Mic., Light, Temp.
IRIS	3	L. Ankle	2-Axis Acc., Mic., Light, Temp.
IRIS	4	R. Ankle	2-Axis Acc., Mic., Light, Temp..
IRIS	5	Head	2-Axis Acc., Mic., Light, Temp.

For the microphones and accelerometers, raw ADC values are sampled at 20ms intervals to ensure quick body movements can be captured, with light and temperature ADC readings sampled at 1s intervals, and GPS/WiFi sampled every 10s. To reduce communication overhead, data for each sensor is aggregated locally on each node at 10s intervals,

which is well within the time granularity of the activities we classify. To reduce the complexity of local aggregation, data from each accelerometer axis is treated as a separate sensor. During local aggregation, light and temperature sensor readings are averaged since these sensor readings remain relatively stable for each activity. Except for GPS/WiFi, all other sensors compute the difference between the highest and lowest readings for each aggregation interval, for the change in readings indicate body movement or sound.

During the two week period, both subjects recorded all activity ground truth in order to evaluate the accuracy of training data (training accuracy) and runtime accuracy. We recorded ground truth for 3 classification categories, illustrated in Table 1: Environment, Posture, and Activity.

4 AdaBoost Activity Recognition

The core of our activity recognition approach uses ensemble learning, specifically AdaBoost.M2 [8], which we expand and improve upon in subsequent sections. In this section, we explain how we adapt AdaBoost to run on a phone for use with a body sensor network. AdaBoost is lightweight enough for mobile phones, yet previous work [14] [20], while relying on offline processing with no feedback or user control, has demonstrated AdaBoost to be accurate for classification applications. Furthermore, without user parameter tuning, our implementation of AdaBoost is able to choose the right sensors needed to maximize training accuracy for all activities. Other classifiers commonly used for activity recognition use a combination of complex, specialized classifiers per sensor modality [17] [19] or per activity [16] which require extensive parameter tuning. Other techniques use single classifiers which are computationally demanding for mobile phones, such as GMMs [18] or HMMs [9].

Using AdaBoost, we incrementally build an ensemble of computationally inexpensive weak classifiers, each of which is trained from the labeled training observations of a single sensor. Weak classifiers need only to make classification decisions that are slightly correlated with the ground truth; their capabilities are combined to form a single accurate classifier. The completed ensemble may contain multiple weak classifiers for the same sensor; some sensors may not have trained classifiers in the ensemble at all. AdaBoost incrementally creates such sensor-based weak classifiers by emphasizing the training observations misclassified by previous classifiers, thus ensuring that training accuracy is maximized.

Using Algorithm 1, we describe AdaBoost training. We define a set of activities $A = \{a_1, \ldots, a_a\}$, sensors $S = \{s_1, \ldots, s_m\}$, and observation vectors O_j for each sensor $s_j \in S$, where each sensor has n training observations. The training output is an ensemble of weak classifiers $H = \{h_1, \ldots, h_T\}$, where $h_t \in H$ represents the weak classifier chosen in the t^{th} iteration. We initialize a set of equal weights D_1 for each training observation, where during the training process, greater weights for an observation represent greater classification difficulty.

During each iteration t, we train a weak classifier $h_{t,j}$ for each sensor $s_j \in S$ using observations O_j and weights D_t. We then compute the weighted classifier error $\varepsilon_{t,j}$ for each trained sensor classifier, adding only the sensor classifier to H which has the lowest weighted error. Before the next iteration, the observation weights D_t are updated based on the current weights and the misclassifications made by the selected classifier.

Algorithm 1 AdaBoost Training

Input: Max iterations T, training obs. vector O_j for each sensor $s_j \in S$, obs. ground truth labels
Output: Set of weak classifiers H

```
Initialize observation weights D_1 to 1/n for all obs.
for t = 1 to T do
    for sensor s_j ∈ S do
        Train weak classifier h_{t,j} using obs. O_j, weights D_t
        Get weighted error ε_{t,j} for h_{t,j} using labels [8]
    end for
    Add the h_{t,j} with least error ε_t to H by choosing h_{t,j}
    with least error ε_t
    Set D_{t+1} using D_t, misclassifications made by h_t [8]
end for
```

Given an observation o, each weak classifier returns a probability vector $[0,1]^a$ with each scalar representing the probability that the current activity is a_i. To train a weak classifier $h_{t,j}$ for each sensor $s_j \in S$, we use a naive Bayes model where training observations O_j are placed into one of 10 discrete bins. The bin interval is based on the minimum and maximum possible readings for each sensor. Each binned training observation from O_j is assigned its respective weight from D_t and ground truth label in the set of activities A. A new observation is classified by placing it into a bin, with the generated probability output vector corresponding to the weights of training observations present for each activity in the assigned bin. With a weak classifier chosen for each iteration, Equation 1 defines the output of the AdaBoost classifier for each new observation o during runtime:

$$h(o) = \text{argmax}_{a_i \in A} \sum_{t=1}^{T} \left(\log \frac{1-\varepsilon_t}{\varepsilon_t} \right) h_t(o, a_i) \qquad (1)$$

In Equation 1, the activity probability vector for each weak classifier h_t is weighted by the inverse of its error ε_t. Thus, the weak classifiers with the lowest training error have the most weight in making classification decisions. To put it another way, AdaBoost chooses the sensors with weak classifiers that minimize weighted training error, achieving maximum training accuracy for all activities.

In Figure 6, we depict AdaBoost training and runtime performance for $T = 1$ to 300 AdaBoost iterations using data from Subject 1 and ground truth from the Activity classification category. For this paper, we use 300 iterations to achieve maximum accuracy. We also show in Figure 6 that AdaBoost does not choose all of the 34 sensors in our experiment, for even with 300 iterations it only chooses 18. During runtime, any sensor not selected by AdaBoost will be disabled to save energy and communication costs.

Lastly, in Figure 7, we demonstrate that AdaBoost can perform very accurately with a sizable amount of training data: maximum training and runtime accuracy is achieved with roughly 50 observations per activity for Subject 1.

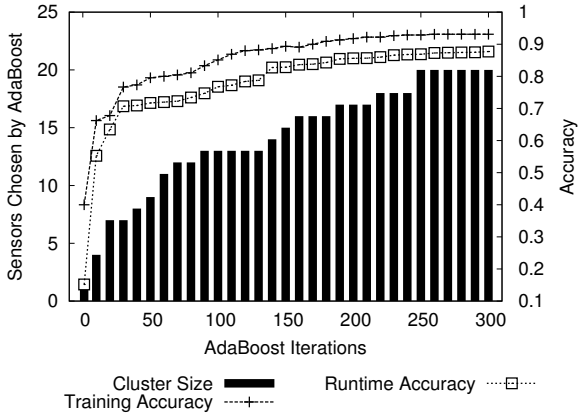


Figure 6: Training and runtime accuracy for 1-300 AdaBoost iterations.

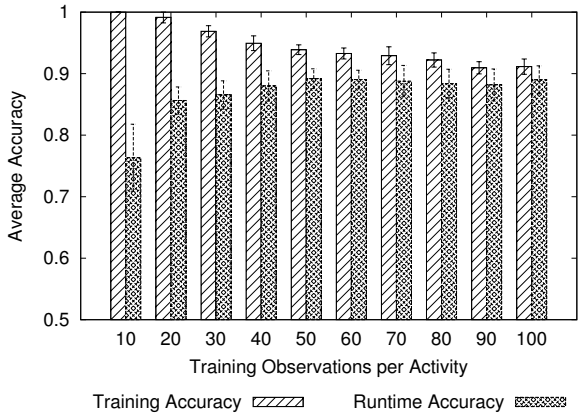


Figure 7: Average accuracy and standard deviation for 10-100 random training observations per activity.

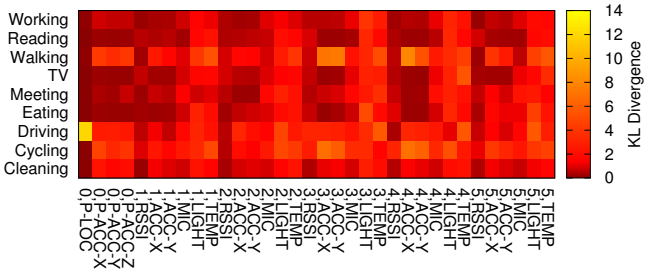


Figure 8: K-L divergence for each sensor and each activity.

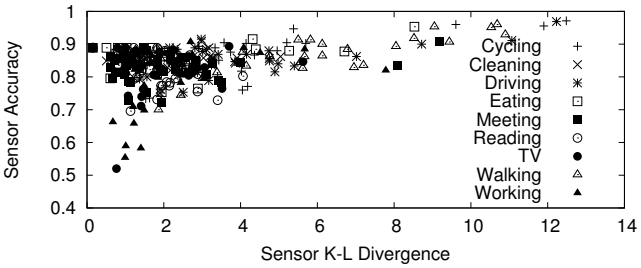


Figure 9: Sensor K-L divergence and training accuracy.

However, through retraining in Section 5 we can train AdaBoost with a very small set of initial training data and update AdaBoost during runtime.

5 Retraining Detection

In this section, we propose an online training approach to achieve high accuracy with a limited initial training data set. This approach can also be used to retrain when an existing data set is not accurate enough. First, we investigate how to quantify the discriminative power of each sensor to detect when runtime data is significantly different than training data. Second, we use the insight we gain from the discrimination analysis to detect when AdaBoost retraining is needed during runtime without retrieval of ground truth information.

5.1 Sensor Data Discrimination Analysis

We investigate how to quantify the discriminative power of each sensor and predict if it is accurate, employing the use of Kullback-Leibler divergence [13]. K-L divergence measures the expected amount of information required to transform samples from a distribution P into a second distribution Q. Using trace data from Subject 1, we demonstrate the use of K-L divergence per activity to identify which sensors are best able to distinguish between activities. We also show a clear relationship between K-L divergence per activity and training accuracy. We conclude that K-L divergence can be used to detect when retraining is needed without regular ground truth requests to compute classifier accuracy: sensors need only to compare training data to current runtime data.

In Figure 8, we analyze the ability of each sensor to discriminate between different activities; we calculate K-L divergence as "one against the rest," for one data distribution is calculated for the target activity and another distribution is calculated for all other activities. For all analyses using K-L divergence, we discretize continuous-valued sensor data into 100 bins. The figure shows that some sensors, such as those on the hands (nodes 1 and 2) have poor ability to distinguish between any activity. Conversely, sensors on the feet (nodes 3 and 4) are especially good at detecting activities that involve motion, such as walking and cycling. The GPS/WiFi velocity (0-LOC) has the highest K-L divergence of all for detecting driving, with a value of nearly 14.

In Figure 9, we compare K-L divergence per sensor and activity to individual sensor training accuracy using Nearest Centroid [11]. Nearest Centroid is a variant of k-means clustering and unlike the AdaBoost weak classifiers, it is an inexpensive classifier that does not require a weight for each training observation. The figure shows a clear relationship between K-L divergence and runtime accuracy, indicating that for a given activity, sensors with high K-L divergence are much more likely to have high accuracy compared with sensors with low K-L divergence. As with Figure 8, activities involving motion, such as cycling, walking, and driving are most easily distinguished and have the highest accuracy. From Figures 8 and 9, we can conclude that K-L divergence will allow sensors to tell activities apart and can even be used to tell if training data and runtime data for the same activity are sufficiently different that retraining is needed.

5.2 Consensus-based Detection

During runtime, retraining detection is performed in two steps, using the insight gained in Section 5.1. First, at each aggregation interval, each active sensor independently determines if retraining is needed. Second, for some aggregation interval, if enough sensors determine that retraining is

needed, PBN prompts the user to record ground truth for a short period. During this period, all sensors are woken up and sample data which is then labeled by the user. When the retraining period completes, a new AdaBoost model is trained using both the old and new training data.

Step 1. Individual Sensor Retraining Detection. In Section 5.1, we demonstrate in that K-L divergence is a powerful tool for determining sensor classification capability. Here, we use the discriminative power of K-L divergence to determine when retraining is needed for each sensor. For a sensor to determine retraining is needed, two conditions must hold: 1) The runtime data distribution for the current activity must be significantly different from the training data distribution, and 2) The runtime data distribution must have at least as many observations as the training data distribution.

During training, each sensor chosen by AdaBoost computes K-L divergence for each activity using its labeled sensor training data. Training K-L divergence per activity is used as a baseline to compare against during runtime to determine when retraining is needed. For each activity $a_i \in A$, training sensor data distribution T_i for activity a_i, and training sensor data distribution T_o for all activities $a_j \in A \setminus \{a_i\}$, each sensor computes the K-L divergence between T_i and T_o, $D_{KL}(T_i, T_o)$. Then, during runtime, for each new observation, AdaBoost classifies the observation as activity $a_i \in A$, and each active sensor adds its data to a runtime distribution R_i for activity a_i. An active sensor s_j determines retraining is needed when the runtime-training K-L divergence is greater than the training K-L divergence for the current activity a_i: $D_{KL}(R_i, T_i) > D_{KL}(T_i, T_o)$.

During runtime, to ensure a fair comparison between training and runtime K-L divergence, each sensor does not determine if retraining is needed until the runtime distribution R_i has as at least as many observations as the training data distribution T_i. We determine through evaluation that collecting fewer runtime observations per activity yields similar accuracy but more retraining instances, which imparts a greater burden on the user. Since Figure 7 demonstrates that 100 observations per activity is more than sufficient to achieve maximum runtime accuracy (this is also true for Subject 2), we limit training data to 100 observations per activity to reduce delay before the training observations and runtime observations are compared.

Step 2. Consensus Detection: Combining Individual Sensor Decisions. During each runtime interval, PBN checks to see how many sensors indicate retraining is necessary. If a weighted number of sensors surpasses a threshold, new ground truth is collected and a new AdaBoost classifier is trained. We describe how this consensus threshold for retraining is obtained.

First, upon completion of AdaBoost training, we can determine the AdaBoost weight of the sensor classifiers used to make correct classification decisions. Through an extension of Equation 1, in Equation 2, we can output the weight of the correct weak classifiers given an observation o and correct AdaBoost decision a_i, $w(o, a_i)$:

$$w(o, a_i) = \sum_{t=1}^{T} \left(\log \frac{1-\varepsilon_t}{\varepsilon_t} \right) h_t(o, a_i) \tag{2}$$

Next, using training data, a trained AdaBoost classifier, and Equation 2, we determine the average weight w_{avg} for all correct training classification decisions (o, a_i). To do this, we compute a weight w_j for each active sensor s_j, indicating how important its decisions are relative to other sensors. Depicted in Equation 3, w_j is computed as the sum of the inverse error rates for each weak classifier belonging to sensor s_j (multiple classifiers for s_j may be iteratively trained and chosen by AdaBoost). Function $f(j)$ maps sensor s_j to each AdaBoost iteration t where AdaBoost chooses the weak classifier trained by s_j.

$$w_j = \sum_{\forall t \in f(j)} \log \left(\frac{1-\varepsilon_t}{\varepsilon_t} \right) \tag{3}$$

At each runtime interval, we sum the weights w_j for each sensor s_j that indicates retraining is necessary. If the sum of the sensor weights is greater than w_{avg}, PBN notifies the user to collect ground truth and retrain a new AdaBoost classifier.

6 Ground Truth Management

We address two issues regarding labeling and addition of new sensor data to the existing training dataset during retraining. First, we address how much new data to add to the training set. Second, we show how to maintain a balance between training data set sizes for each activity to ensure AdaBoost retrains properly and maintains high runtime accuracy.

When PBN decides retraining is required, it will prompt the user to log ground truth for a window of N aggregation intervals. Through evaluation, we find that recording ground truth retroactively for the data that triggered the retraining results in no change in accuracy since any such significant change in sensor data is persistent. During the ground truth logging period, a user only notes the current activity at the start of the logging period and any activity changes throughout the remainder of the logging period. In the evaluation, we determine that with $N = 30$ (5 minutes), the retraining ground truth collection window is short enough not to be intrusive to the user but long enough to capture changes in PBN dynamics. When the ground truth labeling period is complete, the new labeled sensor data is added to the existing training set and a new AdaBoost model is trained.

Since the core of AdaBoost training relies on creating a weight distribution for all training observations based on classification difficulty, each activity must be given nearly equal amounts of training data (within an order of magnitude) for AdaBoost to train properly [10]. Without a balance in training observations across all activities, AdaBoost will focus on training activities with more training observations, creating poor runtime accuracy for activities with fewer training observations. However, if we are too restrictive in enforcing such a balance, very few new training observations will be added to the training dataset during retraining, resulting in poor adaptation to the new data. In Equation 4, we ensure that each activity $a_i \in A$ has no more than δ times the average number of training observations per activity, where O is the set of training observations.

$$\frac{|O_i| - \frac{1}{|A|} \sum_{\forall a_k \in A} |O_k|}{\frac{1}{|A|} \sum_{\forall a_k \in A} |O_k|} \leq \delta \tag{4}$$

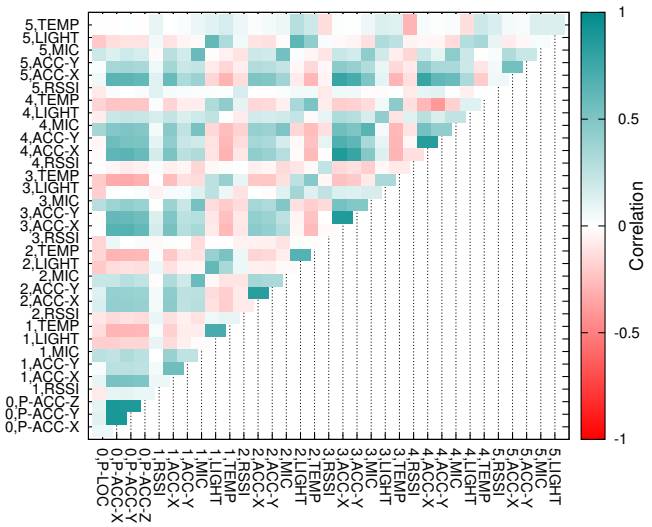


Figure 10: Raw data correlation for Subject 1.

The limit imposed in Equation 4 allows AdaBoost to place equal emphasis on training weak classifiers for each activity during retraining. If, during retraining, an activity exceeds this limit, data is removed until the number of observations is under the limit. We remove observations at random to reach the activity limit since we do not know how representative each observation is of its labeled activity; some observations may contain more noise than others.

7 Sensor Selection for Efficient Classification

While AdaBoost, through training, provides a measure of sensor selection by weighting the most accurate sensors, this approach can be computationally demanding: at each AdaBoost training iteration, a weak classifier is trained and evaluated for each sensor. In this section, we focus on identifying redundant sensors and excluding them from AdaBoost training to reduce the number of weak classifiers trained by AdaBoost. To achieve this goal, we first investigate why ensemble learning algorithms are successful: weak classifiers must be relatively accurate and different weak classifiers must have diverse prediction results [30]. Second, by identifying sensors that satisfy these properties, we provide a method to detect redundant sensors during runtime and exclude them from input to AdaBoost during online retraining. Our method adapts to BSN dynamics during runtime to ensure only helpful sensors are used by AdaBoost.

7.1 Identifying Sensing Redundancies

With ensemble learning and AdaBoost, each weak classifier in the ensemble must be slightly correlated with the true classification to achieve high runtime accuracy. Since we use data from a single sensor to train each weak classifier, it follows that sensors chosen by AdaBoost will be similarly correlated both in terms of raw data and in classification decisions made by each weak classifier. While the Pearson correlation coefficient [25] is also used in previous works [26] [31] to identify packet loss relationships between different wireless radio links; here we use correlation to identify sensing relationships between different sensors to identify both sensors that are helpful as well as redundant.

Figure 10 depicts the correlation between each sensor pair using the raw sensor data collected from Subject 1. It is noted that several correlation patterns exist: accelerometers are strongly correlated as are light and temperature sensors. We can use this information to find sensors with redundant data and remove them from the AdaBoost training process to save computational overhead as well as energy consumption.

We next illustrate that not only do correlation patterns exist between raw sensor data but also between sensor and sensor cluster classifier decisions. In Figure 11, with data from Subject 1, we show that when we add a new sensor to an existing sensor cluster, the greatest accuracy increase is achieved when the sensor and cluster have uncorrelated classification decisions. The figure shows the decision correlation and accuracy change for nearly 7,000 randomly generated clusters from size 1 through 19 using data from Subject 1, with individual sensor and sensor cluster classifiers trained using Nearest Centroid [11]. To compute the decision correlation for a classifier, each correct decision is recorded as 1 and each incorrect decision is recorded as 0. From the figure, it is clear that choosing sensors with a decision correlation close to 0 can help select sensors that will make the most contribution towards an accurate sensor cluster.

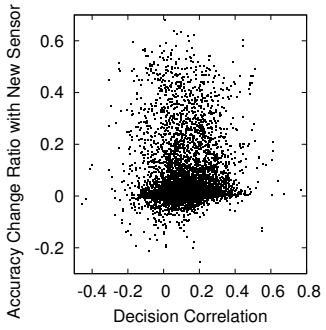


Figure 11: Decision correlation between individual sensors and sensor clusters.

7.2 Adaptive Runtime Sensor Selection

We now describe how to reduce the number of sensors given as input to AdaBoost during online retraining, reducing retraining overhead while achieving high runtime accuracy. We perform sensor selection using raw sensor data correlation, also extending the concept to sensor and sensor cluster classifier decision correlation.

Sensor selection consists of two components: Threshold Adjustment and Selection. During Threshold Adjustment, using a trained AdaBoost classifier, a threshold α is computed which discriminates between sensors chosen by AdaBoost and unused sensors. During Selection, a previously computed α value is used to select a set of sensors for input to AdaBoost during retraining. Threshold Adjustment is performed during initial training while Selection is performed during subsequent retrainings. To ensure the selection threshold stays current with BSN dynamics, we update the threshold α periodically during runtime retraining using Threshold Adjustment and apply smoothing using a moving window of previous α values.

Threshold Adjustment. During initial training, we initialize a sensor selection threshold α and update it during runtime retraining to adjust to any changes in user geographical location, biomechanics, or environmental noise. To de-

Algorithm 2 Raw Correlation Threshold for Sensor Selection

Input: Set of sensors S selected by AdaBoost, training observations for all sensors O, multiplier n
Output: Sensor selection threshold α

$R = \emptyset$ // *set of correlation coefficients*
for all combinations of sensors s_i and s_j in S **do**
  Compute correlation coefficient $r = |r_{O_i,O_j}|$
  $R = R \cup \{r\}$
end for
// *compute threshold as avg + (n * std. dev.) of R*
$\alpha = \mu_R + n\sigma_R$

termine the threshold α, we first train AdaBoost with all sensors as input and determine S, the set of sensors selected by AdaBoost. Using Algorithm 2, we determine the correlation coefficient r of raw sensor training data for each combination of sensor pairs in S. With each correlation coefficient r stored in set R for all sensors selected by AdaBoost, we determine the mean correlation coefficient μ_R and standard deviation σ_R. We then set the threshold α to be n times the standard deviation above the mean correlation. We determine through empirical evaluation that $n = 2$ is sufficient to include nearly all sensors selected by AdaBoost but exclude unused sensors.

Selection. During retraining, when the threshold is not being updated, we choose a set of sensors S^* from the set of all sensors S using the previously computed threshold α. The selected set S^* is given as input to AdaBoost to reduce the overhead incurred by training on the entire set S. In Algorithm 3, we ensure that no two sensors in S^* have a correlation coefficient that is greater than or equal to the threshold α, since we demonstrate previously in Section 7.1 that correlation closest to 0 yields the most accurate sensor clusters. To enforce the threshold, we maintain a set E, which contains sensors that have a correlation coefficient above the threshold for some sensor already in S^*. For each sensor pair in S, we determine the correlation coefficient r and add pairs to S^* where $r < \alpha$. When $r \geq \alpha$ for some sensor pair, we add the less accurate sensor to E as determined by Nearest Centroid [11] and the other to S^* as long as it is not already in E.

Pair DC. We also propose two other correlation metrics with which to select sensors: pair decision correlation (Pair DC) and cluster decision correlation (Cluster DC). With these two metrics, we select sensors based on classification decisions made by individual sensors and sensor clusters. For a sensor or cluster classifier trained using Nearest Centroid, we can convert each training data decision into an integer value as in Section 7.1 to use in computing decision correlation. Sensor selection using sensor pair decision correlation is identical to raw data correlation, except for the use of individual sensor classifier decisions rather than raw data.

Cluster DC. For sensor cluster decision correlation, we modify Algorithms 2 and 3. Instead of iterating through all sensor pair combinations, we first find an individual sensor classifier with the highest accuracy using Nearest Centroid and add it to a trained sensor cluster C^*. We then incrementally add sensors to C^* as long as the sensor has a decision correlation with the cluster that is below the threshold. The final cluster is then used by AdaBoost for training.

Algorithm 3 Sensor Selection Using Raw Correlation

Input: Set of all sensors S, training observations for all sensors O, threshold α
Output: Selected sensors S^* to give as input to AdaBoost

$S^* = \emptyset$
$E = \emptyset$ // *set of sensors we exclude*
for all combinations of sensors s_i and s_j in S **do**
  Compute correlation coefficient $r = |r_{O_i,O_j}|$
  if $r < \alpha$ **then**
    if $s_i \notin E$ **then** $S^* = S^* \cup \{s_i\}$
    if $s_j \notin E$ **then** $S^* = S^* \cup \{s_j\}$
  else if $r \geq \alpha$ and $\text{acc}(s_i) > \text{acc}(s_j)$ **then**
    // *use accuracy to decide which to add to* S^*
    if $s_i \notin E$ **then** $S^* = S^* \cup \{s_i\}$
    $E = E \cup \{s_j\}$; $S^* = S^* \setminus \{s_j\}$
  else
    if $s_j \notin E$ **then** $S^* = S^* \cup \{s_j\}$
    $E = E \cup \{s_i\}$; $S^* = S^* \setminus \{s_i\}$
  end if
end for

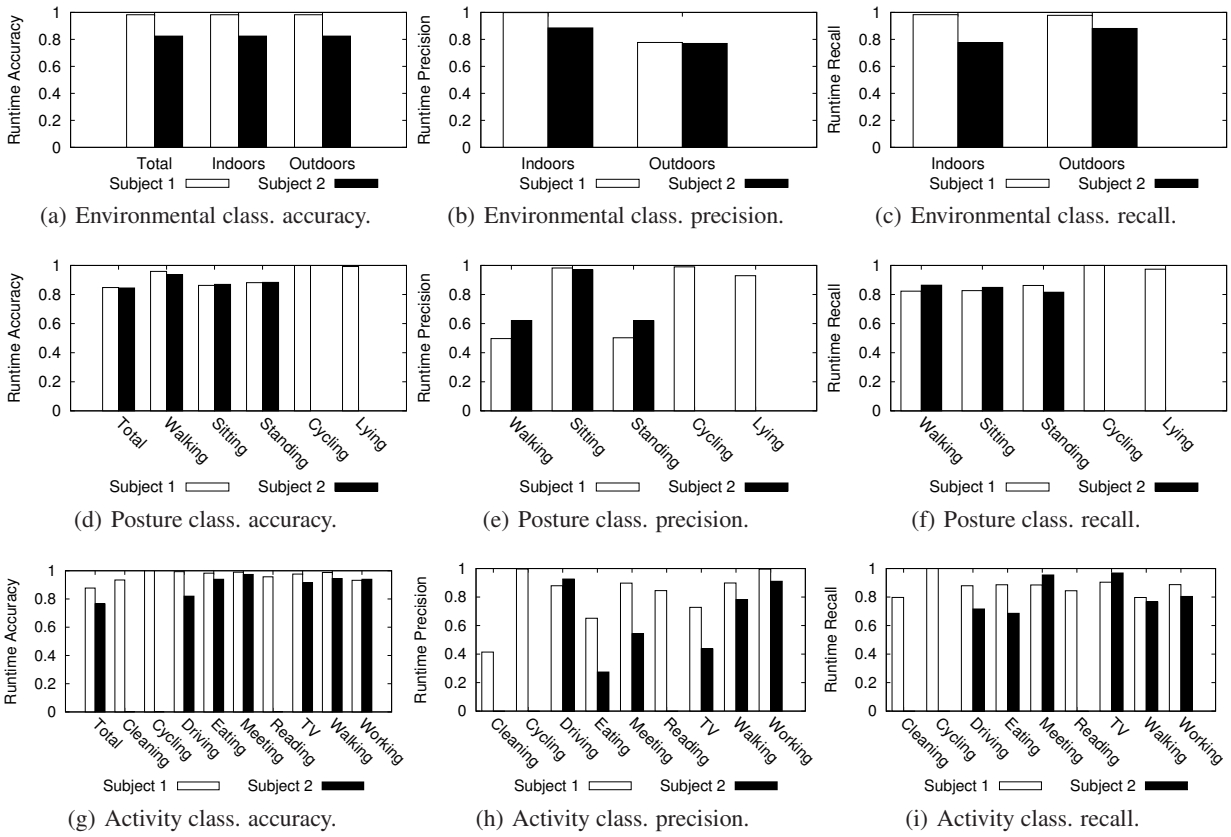
8 Evaluation

We evaluate our PBN activity recognition system using the configuration and data collection methods described in Section 3.4, using two weeks of activity recognition data and two subjects, one of whom is not an author. While these two subjects have substantially different routines and compose very different evaluation scenarios, we leave a more broadly focused evaluation with more subjects to future work. We first evaluate classification performance with a good initial training dataset in Section 8.1 and show that PBN can achieve good accuracy for even difficult to classify activities. In Section 8.2, we evaluate online training with a limited initial training dataset and compare our PBN retraining detection method to a periodic retraining approach, illustrating the benefits of PBN retraining detection. We then show the effectiveness of our sensor selection approach in Section 8.3 and evaluate our PBN application performance in terms of mobile phone hardware constraints in Section 8.4.

8.1 Classification Performance

In Figure 12, we first compare performance for both subjects in the three classification categories depicted in Table 1: Environment, Posture, and Activity, showing that PBN is able to achieve high runtime accuracy with a good initial training dataset (100 observations per activity) along with no online training or sensor selection. Subject 2 does not perform the cycling, lying down, cleaning, or reading categories, hence there is no histogram bar. In addition to total accuracy, we plot accuracy, precision (true positive/(true positive + false positive)), and recall (true positive/(true positive + false negative) for each activity.

Each classification category in Figure 12 has similar performance characteristics per subject: Subject 1 has total accuracies of 98%, 85%, and 90% for the Environmental, Pos-



(a) Environmental class. accuracy. (b) Environmental class. precision. (c) Environmental class. recall.

(d) Posture class. accuracy. (e) Posture class. precision. (f) Posture class. recall.

(g) Activity class. accuracy. (h) Activity class. precision. (i) Activity class. recall.

Figure 12: Runtime performance comparison for multiple subjects and multiple classification category sets. Subject 2 does not engage in the cycling, lying down, cleaning, and reading categories, so the corresponding bars are not presented.

ture, and Activity categories, while Subject 2 has respective total accuracies of 81%, 82%, and 76%. Interestingly, Subject 1 performs significantly better than Subject 2 since Subject 2 often performs each activity less cleanly, for example, working with different lights on in a room or eating in different locations, sometimes with friends. Lastly, more complex activities, which involve a combination of different physical movements as well as external sounds and light, perform reasonably well in comparison with more easily classified activities. Subject 1 has over 90% accuracy for complex activities like cleaning, eating, meeting, reading, and watching TV. Subject 2 also has over 80% accuracy for complex activities like eating, meeting, and watching TV.

We present a classification timeline for Subject 1 using the Activity category in Figure 13, comparing PBN classifications with ground truth. Here, it is apparent that activities involving movement, such as cycling, walking, and driving, are classified with few errors. More misclassifications are seen for more complex activities, such as working confused with meeting, cleaning, and watching TV, however, as illustrated in Figure 12, overall accuracy for each of these complex activities is near or above 90%.

In Figure 14, we show the normalized AdaBoost weights for each sensor for Subject 1 and the Activity classification category as calculated in Equation 3. In addition to the weights for the total of all classifications, we also compute the normalized weight of correct decisions made by each sensor for each activity using runtime data. This figure indicates that AdaBoost is able to select the right sensors to maximize training and runtime accuracy and exclude 16 unhelpful sensors (shown in black). The figure also shows heavy reliance on the light and temperature sensors to distinguish between indoor and outdoor activities as well as reliance on the accelerometers to detect activities involving motion, such as walking, driving, and cycling.

8.2 Online Training

In this section, we demonstrate that we can use a small training dataset, perform online training through retraining detection, and achieve similar accuracy as if we had a larger initial training data set. In this section, we focus on the Activity classification category and also initialize each runtime configuration with 10 random training data samples for each activity. To limit AdaBoost training overhead, we enforce a maximum of 100 training observations per activity since Figure 7 demonstrates that this number is more than enough to achieve maximum runtime accuracy. Since we are using random initial training data, we compute the average and standard deviation for 10 runs of each configuration.

Ground Truth Management. First, we investigate the retraining window size described in Section 6 for collecting new ground truth in Figure 15. From the figure, larger window sizes mean significantly less retraining and computational overhead but also less runtime accuracy. We use a smaller window size of 30 new data elements per retraining

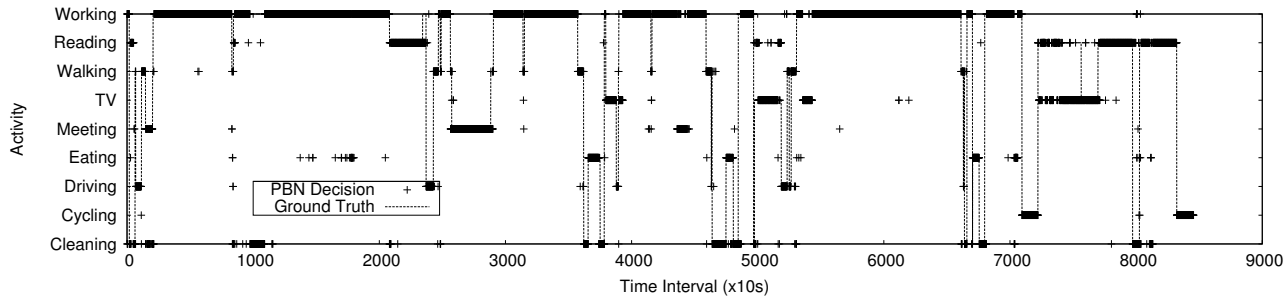


Figure 13: PBN decision and ground truth timeline for Subject 1.

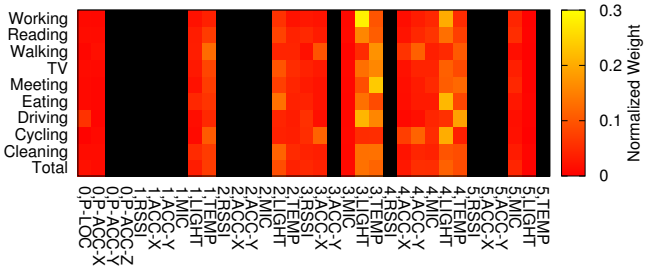


Figure 14: AdaBoost sensor weights per activity.

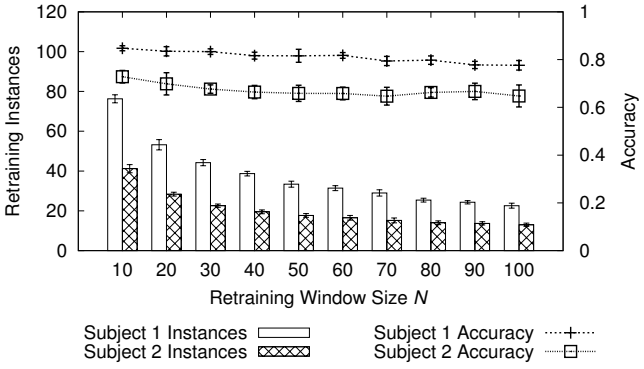


Figure 15: PBN retraining instances and runtime accuracy for new data window sizes N = 10-100.

to balance accuracy and computational overhead. We argue that roughly 20-40 retraining instances per subject over a two week period as per Figure 15 is an inconsequential burden, since the subject must only interact with the phone once per instance to input his or her current activity.

Next, we show the ideal training data balance restriction δ, from Equation 4, in Figure 16. As δ approaches 2.0, runtime accuracy increases and the number of retraining instances decreases. We choose a $\delta = 2.0$ to achieve high accuracy and fewer retraining instances, as larger δ values do not improve accuracy or reduce the number of retraining instances. Furthermore, since we enforce a maximum number of training observations per activity, larger δ values will have no effect on the balance of training data among activities.

Comparison with Periodic Retraining. In Figure 17, we compare the best retraining performance with our retraining detection and ground truth management methods to a naive retraining approach: periodic retraining. We implement periodic retraining for periods of 100 to 500 intervals, with each retraining adding 30 new training data elements to the training data set. We also choose $\delta = 2.0$ for training data balance among all activities. The figure demonstrates that PBN with retraining detection achieves high accuracy for both subjects while periodic retraining either has twice as many retraining instances for the same accuracy or especially in the case of Subject 2, lower accuracy for fewer retraining instances.

We also show that PBN with retraining detection achieves similar accuracy as periodic retraining but incurs fewer retraining instances. In Figure 18, we present a timeline of runtime accuracy and retraining instances for the first 2500 classification intervals comparing PBN to periodic retraining every 100 classification intervals. Both approaches have very volatile initial accuracy, but eventually converge by 2000 intervals. The figures also demonstrate that by reducing the number of retraining instances, PBN retraining detection consequently reduces the amount of ground truth logging by the user.

8.3 Sensor Selection

We now evaluate our sensor selection approach in Section 7 and demonstrate that we can exclude 30-40% of all sensors from AdaBoost training, yet achieve similar runtime accuracy as using no sensor selection. Removing this many sensors from training is a significant savings in computational overhead since these sensors no longer have classifiers trained during each of 300 AdaBoost training iterations.

Using 10 random observations per activity as initial training data and using online training, we compare each of the correlation metrics to determine which is the best in terms of accuracy and sensors excluded. In Figure 19, we depict the percentage of total sensors excluded from AdaBoost (SS Only), the percentage of total sensors excluded by sensor selection and AdaBoost training (SS + AdaBoost), and average runtime accuracy for each sensor selection method.

Figure 19 indicates that both sensor pair raw correlation and decision correlation exclude 30-40% of all sensors from training. Both of these methods also have nearly the same accuracy as without sensor selection, indicating that the right sensors are excluded using these methods. While sensor cluster sensor selection excludes 50% of sensors from training for Subject 1, accuracy is also worse. The figure also shows that in combination with AdaBoost training, each sensor selection method excludes more than 10% points more sensors than with no sensor selection, resulting in additional

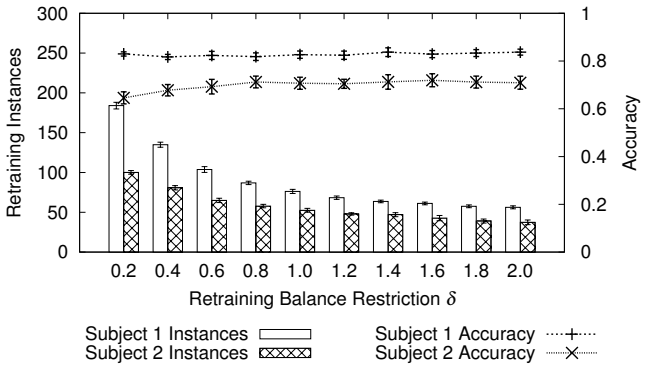


Figure 16: PBN retraining instances and runtime accuracy for activity training data balance restriction δ.

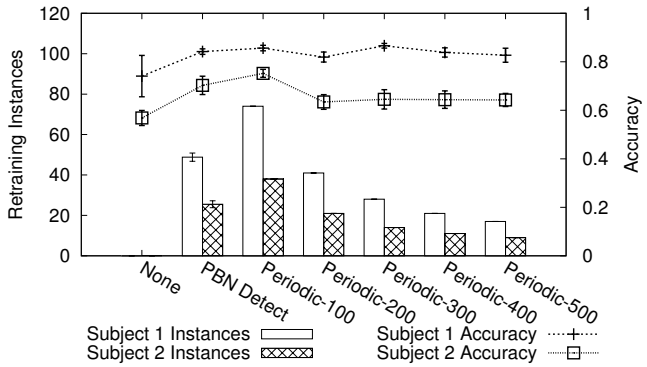


Figure 17: Retraining instances and runtime accuracy without retraining, retraining detection, and periodic retraining.

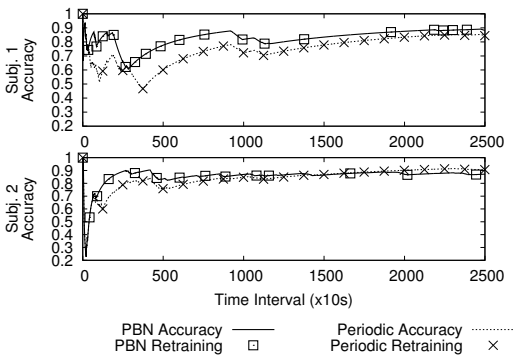


Figure 18: Runtime timeline with accuracy and retraining instances.

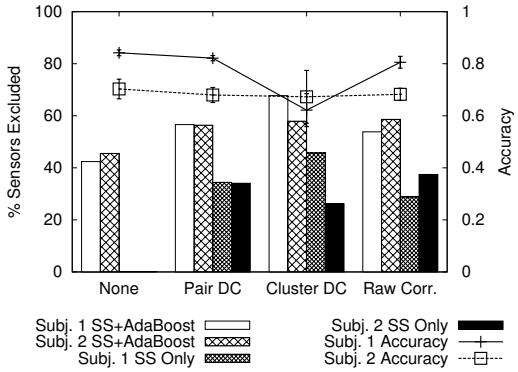


Figure 19: Sensor selection comparison with online training.

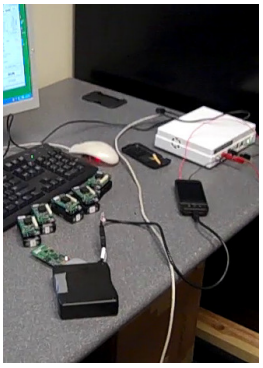


Figure 20: Power Consumption Evaluation.

energy savings since unused nodes are powered down. We suggest that using raw correlation is the best approach to sensor selection, as it requires the least computational overhead of the three metrics, yet excludes a significant number of sensors and maintains high runtime accuracy.

8.4 Application Performance

Table 3: PBN CPU, memory, and power benchmarks.

Mode	CPU	Memory	Power
Idle (No PBN)	<1%	4.30MB	360.59mW
Sampling (WiFi)	19%	8.16MB	517.74mW
Sampling (GPS)	21%	8.47MB	711.74mW
Sampling (WiFi) + Train	100%	9.48MB	601.02mW
Sampling (WiFi) + Classify	21%	9.45MB	513.57mW

During our experiments, the wireless motes had battery life measured in days (4 or 5 days of 8 hour sessions), while the Android HTC G1 phone was unable to last more than 8 hours without recharging. Here, we focus on evaluating the phone performance, for its battery lifetime is much shorter. In Table 3, we measure the CPU, memory, and power consumption of our PBN application to demonstrate that it is practical for mobile phones. We run each configuration for 5 minutes and show the average for system idle, sampling only with GPS or WiFi for localization, sampling with AdaBoost training, and sampling with AdaBoost classification.

Since retraining occurs infrequently, during the vast majority of system deployment (Sampling + Classify), PBN incurs roughly 20% CPU use with memory overhead under 10MB. Most of this overhead is due to the TinyOS Java libraries sending and receiving packets to and from sensor motes; we leave it to future work to further optimize these libraries for mobile devices.

When retraining does occur, it takes between 1 and 10 minutes on the HTC G1, depending on training data size. With newer hardware (HTC Nexus One), retraining time is halved under the same conditions. Since PBN retraining is run as a background process, it can be preempted and has little impact on performance of other applications, such as checking email or making a phone call.

Depicted in Figure 20, we evaluate PBN power consumption with the display off using a power meter from Monsoon Technologies. In Table 3, we demonstrate that during sampling and classification, PBN consumes roughly 150mW in addition to system idle. This consumption is about 1/3 of the additional 450mW consumed by the display when it is active. GPS-based localization consumes an additional 200mW, however GPS is enabled only when WiFi localization is not possible. An additional 90mW is consumed during online training, however, as previously mentioned, these periods are short lived and infrequent.

9 Conclusion and Future Work

In this paper, we present PBN, a significant effort towards a practical solution for daily activity recognition. PBN provides a user-friendly experience for the wearer as well as strong classification performance. Through integration of Android and TinyOS, we provide a software and hardware support system which couples the sensing power of on-body wireless sensors with an easy to use mobile phone application. Our approach is computationally appropriate for mobile phones and wireless motes and also chooses the sensors that maximize accuracy. We improve AdaBoost through online training and enhanced sensor selection, analyzing the properties of sensors and sensor data to identify helpful and redundant sensors as well as indicators for when retraining is needed. We show in our evaluation that PBN can perform accurately for classifying a wide array of activities and postures even when using a limited training dataset coupled with online training. In future work, we intend to provide an extensive usability study with a diverse array of subjects as well as improve energy use on the phone.

10 References

[1] F. Albinali, S. Intille, W. Haskell, and M. Rosenberger. Using Wearable Activity Type Detection to Improve Physical Activity Energy Expenditure Estimation. In *UbiComp '10*, pages 311–320. ACM, 2010.

[2] M. Azizyan, I. Constandache, and R. Choudhury. SurroundSense: Mobile Phone Localization via Ambience Fingerprinting. In *MobiCom '09*, pages 261–272. ACM, 2009.

[3] O. Chipara, C. Lu, T. Bailey, and G.-C. Roman. Reliable Clinical Monitoring using Wireless Sensor Networks: Experience in a Step-down Hospital Unit. In *SenSys '10*, pages 155–168. ACM, 2010.

[4] J. Cranshaw, E. Toch, J. Hong, A. Kittur, and N. Saleh. Bridging the Gap Between Physical Location and Online Social Networks. In *UbiComp '10*, pages 119–128. ACM, 2010.

[5] A. de Quincey. HTC Hero (MSM7201) USB Host Mode, 2010. http://adq.livejournal.com/95689.html.

[6] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell. The BikeNet Mobile Sensing System for Cyclist Experience Mapping. In *SenSys '07*, pages 87–101. ACM, 2007.

[7] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madded, and H. Balakrishnan. The Pothole Patrol: Using a Mobile Sensor Network for Road Surface Monitoring. In *MobiSys '08*, pages 29–39. ACM, 2008.

[8] Y. Freund and R. Schapire. A Decision-Theoretic Generalization of On-line Learning and an Application to Boosting. *JCSS*, 37(3):297–336, 1997.

[9] R. K. Ganti, P. Jayachandran, T. Abdelzaher, and J. Stankovic. SATIRE: A Software Architecture for Smart AtTIRE. In *MobiSys '06*, pages 110–123. ACM, 2006.

[10] H. He and E. Garcia. Learning from Imbalanced Data. *IEEE TKDE*, 21(9):1263–1284, 2009.

[11] M. Keally, G. Zhou, G. Xing, and J. Wu. Exploiting Sensing Diversity for Confident Sensing in Wireless Sensor Networks. In *INFOCOM '11*, pages 1719–1727. IEEE, 2011.

[12] D. Kim, Y. Kim, D. Estrin, and M. Srivastava. SensLoc: Sensing Everyday Places and Paths using Less Energy. In *SenSys '10*, pages 43–56. ACM, 2010.

[13] S. Kullback and R. Leibler. On Information and Sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[14] J. Lester, T. Choudhury, N. Kern, G. Borriello, and B. Hannaford. A Hybrid Discriminative/Generative Approach for Modeling Activities. In *IJCAI*, pages 766–772, 2005.

[15] K. Lorincz, B. Chen, G. Challen, A. Chowdhury, S. Patel, P. Bonato, and M. Welsh. Mercury: A Wearable Sensor Network Platform for High-Fidelity Motion Analysis. In *SenSys '09*, pages 183–196. ACM, 2009.

[16] H. Lu, W. Pan, N. Lane, T. Choudhury, and A. Campbell. SoundSense: Scalable Sound Sensing for People-Centric Sensing Applications on Mobile Phones. In *MobiSys '09*, pages 165–178. ACM, 2009.

[17] H. Lu, J. Yang, Z. Liu, N. Lane, T. Choudhury, and A. Campbell. The Jigsaw Continuous Sensing Engine for Mobile Phone Applications. In *SenSys '10*, pages 71–84. ACM, 2010.

[18] E. Miluzzo, C. Cornelius, A. Ramaswamy, T. Choudhury, Z. Liu, and A. Campbell. Darwin Phones: The Evolution of Sensing and Inference on Mobile Phones. In *MobiSys '10*, pages 5–20. ACM, 2010.

[19] E. Miluzzo, N. Lane, K. Fodor, R. Peterson, H. Lu, and M. Musolesi. Sensing Meets Mobile Social Networks: The Design, Implementation and Evaluation of the CenceMe Application. In *SenSys '08*, pages 337–350. ACM, 2008.

[20] D. Peebles, T. Choudhury, H. Lu, N. Lane, and A. Campbell. Community-Guided Learning: Exploiting Mobile Sensor User to Model Human Behavior. In *AAAI*, 2010.

[21] T. Pering, P. Zhang, R. Chaudhri, Y. Anokwa, and R. Want. The PSI Board: Realizing a Phone-Centric Body Sensor Network. In *BSN*, pages 26–28, 2007.

[22] M. Quwaider and S. Biswas. Body Posture Identification using Hidden Markov Model with a Wearable Sensor Network. In *Bodynets '08*, pages 19:1–19:8. ICST, 2008.

[23] K. Rachuri, M. Musolesi, C. Mascolo, P. Rentfrow, C. Longworth, and A. Acuinas. EmotionSense: A Mobile Phones based Adaptive Platform for Experimental Social Psychology Research. In *UbiComp '10*, pages 281–290. ACM, 2010.

[24] Z. Ren, G. Zhou, A. Pyles, M. Keally, W. Mao, and H. Wang. BodyT2: Throughput and Time Delay Performace Assurance for Heterogeneous BSNs. In *INFOCOM '11*, pages 2750 – 2758. IEEE, 2011.

[25] J. Rodgers and W. Nicewander. Thirteen Ways to Look at the Correlation Coefficient. *The American Statistician*, 42(1):59–66, 1988.

[26] K. Srinivasan, M. Jain, J. Choi, T. Azim, E. Kim, P. Levis, and B. Krishnamachari. The k Factor: Inferring Protocol Performance Using Inter-Link Reception Correlation. In *MobiCom '10*, pages 317–328. ACM, 2010.

[27] Y. Wang, J. Lin, M. Annavaram, Q. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh. A Framework of Energy Efficient Mobile Sensing for Automatic User State Recognition. In *MobiSys '09*, pages 179–192. ACM, 2009.

[28] P. Zappi, C. Lombriser, T. Steifmeier, E. Farella, D. Roggen, L. Benini, and G. Troster. Activity Recognition from On-Body Sensors: Accuracy-Power Trade-Off by Dynamic Sensor Selection. In *EWSN '08*, pages 17–33, 2008.

[29] R. Zhou, Y. Xiong, G. Xing, L. Sun, and J. Ma. ZiFi: Wireless LAN Discovery via ZigBee Interference Signatures. In *MobiCom '10*, pages 49–60. ACM, 2010.

[30] Z.-H. Zhou, J. Wu, and W. Tang. Ensembling Neural Networks: Many Could Be Better Than All. *Artificial Intelligence*, 137:239–263, 2002.

[31] T. Zhu, Z. Zhong, T. He, and Z. Zhang. Exploring Link Correlation for Efficient Flooding in Wireless Sensor Networks. In *NSDI '10*. USENIX, 2010.