Smartphone Sensing

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

in

Computer Science

by

Emiliano Miluzzo

DARTMOUTH COLLEGE

Hanover, New Hampshire

June, 2011

Examining Committee:

(chair) Andrew T. Campbell, Ph.D.

Tanzeem Choudhury, Ph.D.

Lorenzo Torresani, Ph.D.

Matt Welsh, Ph.D.

Brian W. Pogue, Ph.D.
Dean of Graduate Studies

Abstract

The increasing popularity of smartphones with their embedded sensing capability and the availability of new application distribution channels, such as, the Apple AppStore and the Google Android Market, is giving researchers a unique opportunity to deploy mobile sensing applications at unprecedented scale and collect sensor data way beyond the boundaries of traditional small-scale research laboratory deployments. This thesis makes a number of contributions to smartphone sensing by introducing new sensing models, algorithms, applications, and systems.

First, we propose CenceMe, the first large-scale personal and social sensing application for smartphones, which allows users to share their real-time "sensing presence" (i.e., activity and context) with friends using the phone, web, and social network sites (i.e., Facebook, Myspace, Twitter). CenceMe exploits the smartphone's onboard sensors (viz. accelerometer, microphone, GPS, Bluetooth, WiFi, camera) and lightweight, efficient machine learning algorithms on the phone and backend servers to automatically infer people's activity and social context (e.g., having a conversation, in a meeting, at a party). The development, deployment, and evaluation of CenceMe opened up new problems also studied in this dissertation.

Sensing with smartphones presents several technical challenges that need to be surmounted; for example, the smartphone's sensing context (i.e., the position of the phone relative to the event being sensed varies over time) and limited computational resources present important challenges that limit the inference accuracy using phones. To address these challenges, we propose an "evolve-pool-collaborate" model that allows smartphones to automatically adapt to new environments and conduct collaborative sensing among co-located phones resulting in increased robustness and classification accuracy of smartphone sensing in the wild. We call this system, Darwin Phones.

The final contribution of this dissertation explores a new mobile sensing application called VibN, which continuously runs on smartphones allowing users to view live feeds associated with hotspots in a city; that is, what is going on at different locations, the number of people and demographics, and the context of a particular place. VibN addresses a number of critical problems to the success of smartphone sensing, such as, running continuous sensing algorithms on resource limited smartphones, resolving privacy issues, and developing a sensor data validation methodology for applications released via the app stores (i.e., validating sensor data and identifying patterns without any notion of ground truth evidence). Such a methodology is crucial to the large-scale adoption of smartphone sensing in the future.

Smartphone sensing is an emerging field that requires significant advances in mobile computing, machine learning, and systems design. It is an exciting area of research that is cross-disciplinary and likely to touch on many application areas and scientific domains moving forward. The work presented in this dissertation identifies new problems and solutions that help advance our understanding in what is now a fast-moving area of research.

Acknowledgments

First of all, I would like to thank my adviser Professor Andrew T. Campbell for his support, the time he spent to advice, guide, and teach me how to pursue ambitious ideas and transform them in the work of this dissertation. He educated me to the importance of exercising intellectual and technical curiosity as the only means to achieve academic impact. By challenging ideas and assumptions, he fostered a sense of criticism and the need to think outside of the box, necessary characteristics to be a researcher.

Professor Tanzeem Choudhury added significant value to my work with her machine learning experience. Conversations with her helped shape my ideas and took them to the level required for top-tier conference publications.

I am grateful to Professor Lorenzo Torresani for his advise on some of my research ideas, in particular for his machine learning course, which introduced me to the intricacies of this fascinating topic.

I would like to thank Nicholas Lane for his feedback and collaboration in many of my research projects.

I have had the pleasure to work with many other great people while I was at Columbia University, Dartmouth College, and Nokia Research. They are Gahng-Seop Ahn, Shane Eisenman, Chieh-Yih Wan, Kristóf Fodor, Mirco Musolesi, Ronald Peterson, Hong Lu, James Oakley, Xiao Zheng, Cory Cornelius, Ashwin Ramaswamy, Zhigang Liu, Tianyu Wang, Michela Papandrea, Andy Sarroff, Shaohan Hu.

I must express my tremendous appreciation to my dear parents and family members, who always supported me from very far away while suffering for the long distance that separates us. Thank you for your love and support.

Finally, I owe immense gratitude to my wife, for her endurance to put up with my often hectic life style during my Ph.D. I do not think I could have made it without her. Her presence and strength gave me the drive to accomplish this. Thank you for everything.

Dedication

To my wife Giulia, for her love, continuous support and encouragement.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Overview

Smartphones are becoming more and more central to our everyday lives. While early mobile phones were designed to primarily support voice communication, technological advances helped reduce the "gap" between what we consider conventional phones and computers. As this technological divide further diminished, a new paradigm is fast emerging: people are beginning to replace their personal computers with smartphones. The mobility and power afforded by smartphones allow users to interface more directly and continuously with them more than ever before. Smartphones represent the first truly ubiquitous mobile computing device. A critical component that opens up smartphones to new advances across a wide spectrum of applications domains is founded on the embedded sensors in these devices. Sensor enabled smartphones are set to become even more central to people's lives as they become intertwined with existing applications such as social networks and new emerging domains such as green applications, recreational sports, global environmental monitoring, personal and community healthcare, sensor augmented gaming, virtual reality, and smart transportation systems. As such, the global density of smartphones will provide ground breaking ways to characterize people, communities, and the places people live in as never possible before. These advances are enabled not only by embedded sensing, but by a number of other factors as well, including, increased battery capacity, communications and computational resources (CPU, RAM), and new large-scale application distribution channels – also called app stores (such as, Apple App Store, Google Android Market, Nokia Ovi Store). By mining large scale sensing data sets from applications deployed on smartphones through the app stores and using machine learning techniques to analyze the data, it is now possible to discover patterns and details about individuals and ensembles of people not possible before [1, 2, 3]. As a result, we can exploit real-time and historical sensing data from communities of people, making inferences at scale, and advancing the design of new people-centric sensing systems across many diverse application domains [4, 5, 6, 7, 8, 9].

In this dissertation, we propose a number of models, algorithms, applications, and systems that advance *smartphone sensing* or mobile phone sensing [10]. By relying on an ever expanding set of embedded smartphone sensors (e.g., accelerometer, microphone, digital compass, GPS, gyro-

scope, camera, light sensor) rather than specialized sensors [11, 12], and exploiting the ubiquity of smartphone usage, it is possible to characterize people's microcosmos, i.e., activity, context, and surroundings characteristics at very fine grained levels, both in space and time. We call *sensing presence* [13, 14] the result of inferring a personal status (e.g., walking, jogging, meeting friends), disposition (e.g., happy, sad, doing OK), habits (e.g., at the gym, coffee shop today, at work) and surroundings (e.g., noisy, music, lots of people around) using smartphone's sensors. We first highlight the challenges of developing inference and machine learning algorithms for a continuous sensing application called CenceMe [13] deployed on off-the-shelf smartphones. We inject sensing presence into popular social networking applications such as Facebook, MySpace, and Twitter to foster new levels of "connection" and implicit communication (albeit non-verbal) between friends in social networks [13, 14]. Next, we propose a novel distributed computing and collaborative framework to deal with the uncertainty of running inference on smartphones at scale in the wild [15]. We propose a way to validate inference labels collected from the wild where ground evidence is unavailable [16]. Such a validation methodology is a key step to ensuring that the data collected from the wild is reliable and thrustworthy.

1.1.1 Smartphone Sensing

Since Mark Weiser's vision over two decades ago [17] of how the world would change with the introduction of ubiquitous computing, there has been significant progress towards his vision. Context aware computing and smart wearables [18, 19, 20, 21, 22], wireless sensor networks [23, 24, 25], activity recognition using wearables [11, 26, 27, 28, 29, 30], and Human-Computer Interaction (HCI) [31, 32, 33, 34] are just examples of technologies that have spun off the ubiquitous computing idea.

The Smartphone: a Computer with Sensors. Recent technology advances and miniaturization have accelerated the convergence between mobile phones and powerful computers facilitating the development of the smartphone technology. Smartphones computation and storage capabilities are ever growing while integrating a suite of sensors (accelerometer, microphone, GPS, WiFi, digital compass, gyroscope, and, in the future, air quality and chemical sensors [35]). Although many of these sensors have been mainly introduced to drive the device user interface, by taking advantage of smartphones' computational power and sensing capabilities, and their tight coupling with users' daily lives, smartphones can become very compelling platforms to replace the custom designed sensors that researchers have previously adopted to recognize users' activities and context [11, 12]. This new approach, proposed in the CenceMe project [13, 14], where smartphone's onboard sensors data is interpreted through lightweight machine learning algorithms running on the smartphone itself, is giving rise to a new area of research called *smartphone sensing* [10]. The use of these devices for researchers and developers is also facilitated by the availability of free downloadable software development kits to program them. However, the programmability of these platforms is not the only quality that makes smartphones compelling.

The App Stores. The real game changer is the introduction of large scale application distribution systems – or app stores – (such as, Apple App Store, Google Android Market, Nokia

Ovi), which, for the first time, allow researchers to explore research ideas beyond the boundary of their laboratories, and to validate theories at a very large scale. Sensor-enabled smartphones are becoming a mainstream platform for researchers to collect information-rich data because these devices allow the characterization of human activity and context at unprecedented scale and in a pervasive manner [36, 37, 38, 39]. Smartphone sensing facilitates the growth of new classes of applications in the social [14] and utility sphere [40, 41], green and environmental monitoring domains [42, 43, 44, 35], healthcare [45, 46], augmented reality [47], smart transportation systems [48, 49, 50], and virtual worlds [51, 52, 53].

Smartphone Sensing Today and Tomorrow. This dissertation contributes to spearheading the emerging area of smartphone sensing, identifying some of the key challenges and proposing solutions in order to overcome them. Through large scale application deployments and one of the first implementations of learning algorithms on off-the-shelf smartphones with CenceMe [13, 14], we highlight some of the challenges in running mobile sensing applications on smartphones. These challenges span from smartphone programmability, to mobility, the need to preserve the phone user experience, the need to achieve classifier accuracy in the wild and to mitigate the sensing context – that is, the position of the phone carried by a user (e.g., in the pocket, in the hand, inside a backpack, on the hip, arm mounted, etc.) in relation to the event being sensed.

Early smartphones such as the Nokia N95 [54], equipped with accelerometer and GPS, lacked efficient software infrastructure to support mobile sensing applications. These limitations were reflected in inefficient application programming interfaces (APIs) to access some of the phone's components, such as the sensors. There were also limitations for implementing efficient resource management routines, i.e., turn the sensors off when not needed. More recently, iOS, Android, and MeeGo smartphone operating systems are much more supportive platforms for mobile sensing application programming. They provide a more complete set of APIs to access the low level components of the phone operating system (OS) while taking advantage of more powerful hardware (CPU and RAM). However, in spite of the OS and hardware improvements, there are still issues that limit mobile sensing applications. Smartphone battery capacity is still a bottleneck, reducing the possibility to run continuous sensing applications. Another issue is represented by unpredictable or undesirable behavior for some of the sensors. Smartphones' sensors have been mainly introduced to enhance the user experience when interacting with the devices, e.g., flipping the user interface from landscape to portrait mode with the accelerometer. For this reason Apple iOS currently shuts down the accelerometer when an application is pushed to run as background process since there is no active user interface that needs the accelerometer support. The consequence of this approach is the impossibility to rely on a continuous accelerometer data stream, which is the foundation for reliable activity inference. Other limitations for iOS, for example, derive from the limited run-time control for the developer of the localization engine, a notoriously power hungry sensor. Besides software impediments, there are other factors that need to be taken into consideration in order to make a mobile sensing application successful. These are issues related to the sensor data interpretation, inference label accuracy and validation. When a machine learning algorithm is deployed to

run in the wild without supervision, the lack of ground truth evidence calls for mechanisms that associate, for example, a validation weight to the inference labels in order to provide a certain confidence about the trustworthiness of the inferred labels. Also, the variation of the smartphone sensing context degrades the quality of the inference (this is the case for mobile sensing applications exploiting the microphone for audio sensing for example, which is negatively impacted by the phone being in a user's pocket or backpack). There is a need to design inference schemes that are reliable and boost the inference results of applications running in the wild. Although they have increasingly powerful hardware platforms, smartphones still have limitations in running computationally intensive algorithms, such as the learning phase of a machine learning technique. External, more powerful support, e.g., cloud computing, could be exploited to mitigate this problem.

Smartphone Sensing Application Requirements. When designing mobile sensing applications it is paramount to make sure the aforementioned issues are properly addressed. However, this can be done successfully only by meeting the tradeoff of three important and contrasting metrics: *inference fidelity*, *inference responsiveness*, and *resource usage*. Inference fidelity reflects the accuracy of the inferred labels. Inference responsiveness refers to the time needed to compute the inference labels, while resource usage is a parameter that quantifies the impact of the mobile sensing application on the smartphone CPU, RAM, and battery life. Recent work proposes early solutions to accommodate this tradeoff in the context of location and activity recognition [55, 56, 57, 58, 59].

Privacy and Trust. Privacy and security are very sensitive issues for continuous and pervasive sensing applications that infer user's activity, context, and surrounding conditions. Solutions to protect users' privacy for smartphone sensing applications have been proposed [60, 61, 62, 63]. There is also a need to take into account the risk of malicious downloaded applications that hack into user's data stored on the device [64].

Opportunistic Sensing. The work in this dissertation is founded on an opportunistic sensing paradigm [4, 6], where the user is not an active participant of the sensing process (i.e., actively taking a sensor reading). In this case, sensing happens automatically and continuously when the system determines that the sensing context is right for sensing. Our opportunistic sensing approach can be differentiated by a participatory sensing system, where instead the user is asked to participate in the sensing process by actively triggering or reporting the sensor readings from the mobile devices [7].

1.1.2 Problem Statement

A major research focus of the pervasive, ubiquitous computing, and sensor networking community is sensing user's activity, context, and surroundings, e.g., sensing presence [13], to realize mobile sensing applications in different domains, such as social networking, gaming, green transportation, and health care.

Lack of Scalability Using Wearables. The main challenge to the success of these applications is the lack of scalability of any of the solutions requiring the instrumentation of places with ad-hoc sensors or customized wearables to sense users' activity, context, and the environment [65, 66, 67, 68, 69, 70, 12, 27, 11, 29]. Both large monetary cost to deploy and maintain infras-

tructures and the need for incentives to motivate people to wear specialized sensors, along with the possibility of realizing only small-scale deployments, have been the main barriers to the popularity of pervasive and ubiquitous computing applications and systems at scale. The increasing popularity of smartphones and their computational capabilities, along with the array of sensors mounted on them, have become a game changer for both researchers and application developers: there is no need anymore to rely on custom infrastructures to infer users' context and activities because of the possibility to exploit the pervasiveness of smartphones while taking advantage of their computational and sensing capabilities.

Our work is one of the first approaches that identifies the smartphone as the fulcrum of the mobile computing revolution and sensing presence inference. We do so by bringing intelligence to the phone and inferring sensing presence on the move from the device's onboard sensors. While in some early work researchers [71, 72] and industry players [73] realized the centrality of smartphones to deliver context-aware applications, they either relied on body-worn sensors interfacing with mobile phones [71] or on a set of logic rules that combined the input from the phone's GPS, phone usage patterns (idle/active, battery charging, frequency of made and received calls, SMS usage), and Bluetooth proximity detection to identify the user's context [72]. For the first time, we bring intelligence to off-the-shelf smartphones to compute a user's sensing presence going beyond simple heuristic rule-based approaches and mere user-driven input. We deploy machine learning techniques on the phone itself that derive the sensing presence automatically and transparently to the user by sourcing data from the smartphone's onboard sensors, such as the accelerometer, microphone, GPS, WiFi, Bluetooth, gyroscope, and magnetometer.

Machine Learning Limitations on Smartphones. Many challenges arise from bringing machine learning to smartphones. Most of the known machine learning algorithms to date have been designed to run on powerful computers and do not adapt well to resource constrained devices such as smartphones. It is important to design mobile sensing applications for smartphones while meeting the phone user experience requirements, e.g., the ability to make and receive calls, an acceptable battery duration and the tradeoff between inference fidelity, inference responsiveness, and phone resource usage (where inference fidelity reflects the inferred labels accuracy. Inference responsiveness refers to the time needed to compute the inference labels, while resource usage is a parameter that quantifies the impact of the mobile sensing application on the smartphone CPU, RAM, and battery life). It is also necessary to identify new machine learning algorithms that are less resource demanding and suitable for the limited resources available on smartphones. When analyzing sensor data to infer a user's sensing presence it is fundamental to protect the privacy of the user. One way to meet this requirement is to operate as much as possible on the smartphone for feature extraction without exposing the raw data to the external world, and communicating only the features or the result of the inference to external entities. To reduce the impact on the battery consumption, local feature extraction and some on-the-phone data pre-processing is also required to minimize the amount of data to be sent to the servers. Running feature extraction on the phone implies the need to identify features that require low computation (compared to the complex features extracted on

server machines), yet are effective to produce accurate machine learning models. These observations, combined with the facts that battery capacity increase is slower than smartphone computation growth, and that wireless transmission is one of the main causes of battery drain, demand frameworks that rely on split techniques that distribute and tradeoff local (on the device) versus remote (on the server) computation. For this reason, a split-level computation/classification approach is needed, whereby part of the tasks – cheap in terms of computation – can run on the phone while more resource-intensive routines – such as training a learning algorithm – are offloaded to servers. Our split-level computation approach differs from techniques that completely rely on the external cloud computing support and envision only thin clients on the phone itself [74].

Mobility and Sensing Context. There are other challenges that need to be addressed in order to build reliable smartphone sensing applications. One is the sensing context problem, i.e., the position of the phone carried by a person (e.g., in the pocket, in the hand, inside a backpack, on the hip, arm mounted, etc.) in relation to the event being sensed. Mobile phones carried by people may have many different sensing contexts that limit the use of a sensor, for example: a chemical sensor or the microphone offer poor sensing quality when buried in a person's backpack. Another challenge is the classification model scalability in the wild after it has been trained in a fully supervised manner before the deployment. Regardless of the richness of the training data used to build a classification model, in order to adjust to user behavior for best performance tuning, classifiers might need to be adapted once they are deployed. In order to reduce the burden on application developers to retrain classifiers in a supervised manner and on users to provide labels, in the Darwin Phones paper [15] we propose the use of automatic classifier evolution by exploiting semi-supervised learning techniques. To reduce the impact of the sensing context on the inference accuracy we introduce classifier cooperation across different co-located smartphones. Moreover, there is a need to preserve the phone user experience. In order to achieve this goal, we let smartphones pool (or borrow) classification models from surrounding smartphones or from the backend if the classifier is already available. Thus, we reduce the impact on the smartphone resources (CPU, RAM, and battery) of a mobile sensing application.

Need for Multi-Modality Sensing. While a multi-modality sensing approach has been demonstrated to be helpful in different contexts [6, 27, 75, 76, 77, 78, 79], multi-modality sensing is often necessary in mobile sensing applications to boost the overall sensing presence inference accuracy and to compensate for the lack of effectiveness of some of the sensors (due to the sensing context problem for example). We show how this approach boosts the sensing presence inference accuracy in CenceMe [14], Darwin Phones [15], and Discovery to infer a smartphone sensing context [80].

Deployment at Scale. The application distribution system support in particular (e.g., Apple App Store, Android Market, Nokia Ovi) is a game changer for the research community, because it enables the instant deployment of applications onto millions of smartphones and gives the opportunity to collect very large data sets from the wild as never possible before. By mining rich, large-scale data sets, researchers will be able to answer novel research questions. We are given the opportunity to characterize spaces at a very fine grained level, which is generally impossible without

burdensome subject polling. Such information may be useful, for example, to help city managers understand how people exploit urban spaces, resulting in improved urban planning. Alternatively, physicians may learn the health behavior of a community and use this information for community health assessment and recommendations. Distributed sensor monitoring and inference can expand opportunities for automated personal information sharing and for people behavior characterization. However, when an application is deployed in the wild, there is no inherent method for the developer to verify whether inferences are meaningful and correct. For instance, if a mobile device reports that a user's activity is walking, we lack ground truth to verify that the inference is not the result of a false positive misclassification. While erroneous classification may be tolerated in leisure applications, it may not be acceptable for more critical applications, such as those that assess wellbeing or health. We propose a sensor inference validation methodology where we combine data from multiple dimensions (microblog, GIS, public reports databases) with different sensing modalities in order to provide improved confidence about the inferred labels collected in the wild [16].

1.2 Thesis Outline

The proposed smartphone sensing system architectures, algorithms, and applications in this dissertation are rigorously evaluated using a combination of simulation, analysis, and experimentation. Experimental research plays a key role in the work presented. We build small experimental smartphone sensing systems in the laboratory, study their behavior in the wild, and apply our findings toward the construction of larger and more scalable smartphone sensing systems and applications. By implementing smartphone sensing system architectures, algorithms, and applications on off-the-shelf smartphones and leveraging large scale application distribution channels such as the Apple App Store and Google Android Market we discover and highlight the challenges presented by realistic mobile sensing system deployments and propose solutions to address them.
An outline of our study follows.

1.2.1 CenceMe: A Mobile Sensing Application to Infer and Share Personal Sensing Presence

In Chapter 2 we present the design, implementation, evaluation, and user experiences of the CenceMe application, which represents the first system that combines the sensing presence inference using off-the-shelf, sensor-enabled mobile phones with the sharing of this information through social networking applications such as Facebook, MySpace, and Twitter. We discuss the system challenges for the development of software on one of the earliest programmable smartphones, i.e., the Nokia N95 mobile phone, and show the performance of the software on the Apple iPhone. We present the design and tradeoffs of split-level classification, whereby personal sensing presence is derived from classifiers which execute in part on the phones and in part on the backend servers to achieve scalable inference. We report performance measurements that characterize the computational requirements of the software and the energy consumption of the CenceMe phone client. We validate the system

through a user study where 22 users used CenceMe continuously over a three-week period in a campus town. From this user study we learn how the system performs in a production environment and what uses people find for a personal sensing system. Smartphones and "app stores" are enabling the instantaneous distribution of a wide variety of third-party applications to a very large number of users around the globe with the potential to collect rich, large-scale data sets. This new era represents a game changer for our research community – one which we are still analyzing and exploiting. We discuss our experiences in developing, distributing, and supporting the CenceMe deployment at scale through the Apple App Store.

1.2.2 Darwin Phones: A Distributed and Collaborative Inference Framework for Smartphone Sensing Support

In Chapter 3 we present Darwin, an enabling technology for smartphone sensing that combines collaborative sensing and classification techniques to reason about human behavior and context on mobile phones. Darwin advances smartphone sensing through the deployment of efficient but sophisticated machine learning techniques specifically designed to run directly on sensor-enabled smartphones. Darwin tackles three key sensing and inference challenges that are barriers to the mass-scale adoption of smartphone sensing applications: (i) the human-burden of training classifiers, (ii) the difficulty of performance reliability in different environments (e.g., indoor, outdoor), and (iii) the need to scale to a large number of phones without jeopardizing the "phone experience" (e.g., usability and battery lifetime). Darwin is a collaborative reasoning framework built on three concepts: classifier/model evolution, model pooling, and collaborative inference. To the best of our knowledge Darwin is the first system that applies distributed machine learning techniques and collaborative inference concepts to mobile phones. Another innovation is a technique to infer the phone sensing context, that is, the position of the phone carried by a person (e.g., in the pocket, in the hand, inside a backpack, on the hip, arm mounted, etc.) in relation to the event being sensed. We implement the Darwin system on the Nokia N97 platform and Apple iPhone. While Darwin represents a general framework applicable to a wide variety of emerging mobile sensing applications, we implement a speaker recognition application to evaluate the benefits of Darwin. We show experimental results from eight users carrying Nokia N97s and demonstrate that Darwin improves the reliability and scalability of the proof-of-concept speaker recognition application without additional burden to users.

1.2.3 VibN: A Large-Scale Mobile Sensing Application for People and Place Characterization

In Chapter 4, we discuss a large-scale mobile sensing application to characterize places and communities. The increasing popularity of smartphones, as well as the growth of their distribution channels, is giving researchers a unique opportunity: the ability to deploy mobile sensing applications at unprecedented scale and to collect data beyond the boundaries of a research lab. The

result is an invaluable source of information that, when mined, enables the analysis of personal and inter-personal behavior, as well as user interaction with spaces. We present VibN, a mobile sensing application deployed in large scale through the Apple App Store and Android Market. VibN has been built to determine the "whats going on" around the user in real time by exploiting multiple sensor feeds. The application allows its users to explore live points of interest in the city by presenting real time hotspots from sensor data. Each hotspot is characterized by a demographic breakdown of inhabitants and a list of short audio clips. The audio clips augment traditional microblogging methods by allowing users to automatically and manually provide rich audio data about their locations. VibN also allows users to browse historical points of interest and view how locations in a city evolve over time. Additionally, VibN automatically determines users' personal points of interest, which are a means for building a user's breadcrumb diary of locations where they have spent significant amounts of time. We present the design, evaluation, and results from the large scale deployment of VibN through the popular Apple App Store and Android Market. We also discuss a validation methodology for sensed data gathered in similar large scale deployments, where ground truth evidence is unavailable.

1.3 Thesis Contribution

Herein, we make several broad contributions to the smartphone sensing field, as summarized in the following.

1. The work in this dissertation contributes to spearheading the emerging area of smartphone sensing. In Chapter 2 we go beyond the need to employ smart wearables and custom-designed sensors for people's activity and context recognition [18, 19, 20, 21, 22, 11, 26, 27, 28, 29, 30] and discuss how we bring, for the first time, intelligence to off-the-shelf smartphones to infer a person's sensing presence and their microcosmos through a series of machine learning techniques running on the smartphone itself. We show how to implement lightweight, yet effective, machine learning algorithms that can operate on a smartphone by relying on the cloud computing support, according to what we call the split-level computation/classification approach, to improve the resource efficiency of a learning technique on the smartphone. We carry out one of the first large-scale smartphone sensing user studies by deploying the CenceMe application through the Apple App Store, overcoming a number of technical challenges to run a continuous sensing system on the phone and develop it to a production level system used simultaneously by thousands of users. We expose the many challenges arising from running mobile sensing applications on smartphones, ranging from the need of efficient duty-cycling sensing and inference techniques, to privacy issues, and the importance of robust and resilient machine learning classifiers to cope with sensing context and different individual user behavior. We also discuss the experiences in developing, distributing, and supporting the CenceMe deployment at scale through the Apple App Store. The CenceMe publications [13, 14] have been widely cited as seminal work in the smartphone

sensing space by researchers studying privacy and security [61, 81, 82], energy efficient sensing and inference algorithm modeling [55, 56, 58, 59, 83, 84], mobile sensing applications [40, 41, 42, 48, 50, 85, 34], inference engine toolbox [86], architectures for mobile social applications [87, 88], middleware [89], and social studies [90].

2. The CenceMe deployments [13, 14] highlight many issues playing against the feasibility of smartphone sensing applications at scale. Some of the important factors that limit the performance of a mobile sensing application are the phone sensing context, the lack of generality of classification models at the time of training, the need to preserve the phone user experience, and mobility. In Chapter 3 we present Darwin Phones [15], which introduces a collaborative reasoning framework built on three concepts: *classifier model evolution*, *model pooling*, and *collaborative inference*. With its *evolve-pool-collaborate* model, Darwin is the first collaborative framework introduced to support mobile sensing applications at scale. Automatic classification model evolution using semi-supervised techniques is introduced to tune a classification model to users with different habits and contexts than the ones captured in the data used for training the initial model. Classification model pooling meets the requirement of preserving the phone user experience and it allows to perform inference quickly during the short rendezvous time induced by mobility. Mobile phones have the opportunity, when possible, to pool classification models from co-located phones or from the backend in order to save those resources (RAM, CPU, battery) that otherwise would be heavily employed to train a classification model from scratch. Finally, mobile phones cooperate in the inference phase to achieve higher classification confidence. Darwin is based on the idea that multiple phones classifying an event and sharing their local estimate about the event can lead to better global classification results. A technique to infer the phone sensing context is proposed.

3. Mobile social applications (e.g., Loopt, Foursquare), review systems (e.g., Yelp), and microblogging services (e.g., Twitter) are gaining popularity. The goal of these applications is to instantaneously connect people and provide detailed information about places (i.e., points of interest) and people at any time. However, a drawback of these applications is that the information provided might change at a slow time scale and therefore become stale quickly. In Chapter 4 we present VibN, a new mobile sensing application that supports live points of interest of a city. VibN has been released at scale through the Apple App Store and Google Android Market. VibN allows the user to view live feeds associated with the hotspots in a city – i.e., what is going on at different locations, the number of people and demographics (i.e., sex, age, marital status) and the context of the place (e.g., if it is a club, then what kind of music is being played). The VibN project is addressing a number of problems related to capturing and distributing live points of interest, such as: running continuous sensing algorithms on resource limited smartphones, studying the interaction between mobile devices and cloud servers, resolving privacy issues, and developing a new sensor data validation methodology for applications released via app stores. This methodology is needed to validate the

inferred labels and identify patterns without any notions of ground truth evidence. Such a methodology is crucial to verifying the reliability of inferred labels collected from large scale deployments.

We believe that CenceMe, Darwin, and VibN significantly advance the understanding of smartphone sensing by proposing some early solutions to solve the challenges in this new space and by opening up new research directions.

Chapter 2

A Mobile Sensing Application to Infer and Share Personal Sensing Presence

2.1 Introduction

One of the most common text messages people send each other today is "where r u?" followed by "what u doing?". With the advent of powerful and programmable mobile phones, most of which include a variety of sensing components (e.g., accelerometers, GPS, proximity sensors, microphone, camera, etc.) there is a new way to answer these questions. In essence, mobile phones can create mobile sensor networks capable of sensing information that is important to people, namely, where are people and what are they doing?

The sensing of people is driving a new application domain that goes beyond the sensor networks community's existing focus on environmental and infrastructure monitoring, where people are now the carriers of sensing devices, and the sources and consumers of sensed events. The expanding sensing capabilities of mobile phones (e.g., Nokia N95 and Apple iPhone) combined with the recent interest by the mobile phone vendors and cellular industry in open programming environments and platforms, typified by the recent release of the Android platform [91] and the Apple iPhone SDK [92], is accelerating the development of new people-centric sensing applications and systems [5].

In this Chapter, we present the design, implementation, evaluation, and user experiences of the CenceMe application [13], a new people-centric sensing application. CenceMe exploits off-the-shelf sensor-enabled mobile phones to automatically infer people's sensing presence (e.g., dancing at a party with friends) and then shares this presence through social network portals such as Facebook. We evaluate a number of important system performance issues and present the results from a user study based on an experiment conducted over a three-week period in a campus town. The user study included 22 users consisting of undergraduates, graduates, and faculty at Dartmouth College.

We discuss results, experiences, and lessons learnt from the deployment of CenceMe on off-the-shelf mobile phones. These phones, while fairly powerful computers, present a number of limitations in supporting the demands of a continuous personal sensing application such as CenceMe. We implement CenceMe on the Nokia N95 phones. Although the N95 is a top-end device with a great

deal of computation capability, the Symbian operating system and Java Micro Edition (JME) virtual machine which runs on top of the N95 are rather limiting due to the fact that they have both been designed to use small amounts of memory and computational resources. Additional implementation challenges arise from the fact that manufacturers and operators limit the programmability of mobile phones to preserve the closed nature of their devices and operational networks. For this reason appropriate certificates purchased from a Certificate Authority are needed, yet are not sufficient for full deployment of an application such as CenceMe. We show the tradeoffs and discuss the difficulties in implementing an always-on sensing application on the Symbian/JME platform which more generally is designed to accommodate simple applications such as gaming and calendar plugins.

Contributions of our work include:

- The design, implementation, and evaluation of a fully functional personal mobile sensor system using an unmodified mobile phone platform.
- The design of lightweight classifiers, running on mobile phones, which realize a split-level classification paradigm. We show they have a limited impact on the phone's functionality.
- Measurements of the RAM, CPU, and energy performance of the classifiers and the CenceMe software suite as a whole, showing the tradeoff between the time fidelity of the data and the latency in sharing that data.
- Performance of the CenceMe software on the Apple iPhone as a way to measure the impact of a mobile sensing application on the popular iPhone device.
- A validation of the CenceMe application through a user study. This is one of the first user studies that involves a large group of people using a personal sensing application running on off-the-shelf mobile phones for a continuous period of time. The study provides useful insights into how people understand and relate to personal sensing technology. The study offers some suggestions on the further development of people-centric sensing applications.
- Discussion of our experience in developing, distributing, and supporting CenceMe for the Apple iPhone, first released when the Apple App Store opened in 2008. We had to come to terms with supporting a fairly complex real-time sensing application outside the normal controlled laboratory setting. Instead of deploying the CenceMe application to a small set of local users (e.g., 30+ users when we first deployed CenceMe on Nokia N95s in 2007) we had to deal with thousands of users distributed around the world.

In Section 2.2, we present a number of design considerations when building an always-on sensing application such as CenceMe on mobile phones. The CenceMe implementation is discussed in Section 2.3, while in Section 2.4 the phone and backend classifier algorithms are presented. In Section 2.5, we show the performance of the CenceMe classification algorithms as well as detailed power, RAM, and CPU measurements. In Section 2.6 we discuss the capability of the Apple iPhone for supporting mobile sensing applications. In Section 2.7, we present the results of our user study

and then in Section 2.8 the experience in developing, distributing and supporting CenceMe through the Apple App Store. In Section 2.11 we discuss the related work and summarize the Chapter in Section 2.12.

2.2 Design Considerations

Before describing the implementation of the CenceMe application on the phone and backend servers, we first discuss the system development challenges encountered when implementing an application such as CenceMe on the phone. These impact several aspects of the architectural design.

2.2.1 Mobile Phone Limitations

OS Limitations. Although top-end mobile phones have good computational capability, often including multiple processors, they are limited in terms of the programmability and resource usage control offered to the developer. For example, the Nokia N95 is equipped with a 330 MHz ARM processor, 220 MHz DSP, and 128 MB RAM. However, when developing a non-trivial application on mobile phones a number of challenges arise. This is due in part because mobile phones are primarily designed for handling phone calls in a robust and resilient manner. As a result, third party applications running on the phone may be denied resource requests and must be designed to allow interruption at any time so as not to disrupt regular operations of the phone. This places a heavy burden on application exception handling and recovery software. While programmers may expect exception handlers to be called rarely, in Symbian they are called often and are critical to keeping an application and the phone operational. At the same time, testing exception handlers is difficult because a voice call can interrupt application code at any point in its execution; OS induced exceptions are outside the control of the programmer.

API and Operational Limitations. Additional limitations arise from the APIs provided by the phone manufacturers. JME implements a reduced set of the Java Standard Edition APIs for use on mobile phones. Because each phone model is different even from the same manufacturer, the Symbian OS and JME must be ported to each phone which typically results in missing or malfunctioning APIs for important new or existing components, such as an accelerometer or GPS. These API limitations may not be resolved by the manufacturer because new models replace old models in quick succession. As a result, the programmer is forced to come up with creative solutions to API limitations. Examples of such API limitations and operational problems encountered with the N95 include a missing JME API to access the N95 internal accelerometer and JME audio API that exhibits a memory leak, respectively.

Security Limitations. To preserve the phone's integrity and protect the cellular network from malicious attacks, phone manufacturers and cellular network operators control access to critical components, including the APIs for access to the file system, multimedia features, Bluetooth, GPS, and communications via GPRS or WiFi, through a right management system. Properly signed keys from a Certificate Authority are needed to remove all restrictions on using these APIs.

Energy Management Limitations. An important driver for application designers on mobile phone platforms is power conservation, in particular, when radio interfaces such as Bluetooth, GPS, and GPRS are used by the application. As we show in Section 2.5, the phone's Bluetooth, GPS, and GPRS radios are responsible for draining most of the battery power when CenceMe is running. As application developers, we want to build applications that offer good fidelity and user experience without significantly altering the operational lifetime of the standard mobile phone. Therefore, designing efficient duty-cycles for the application and its use of power hungry radios such Bluetooth and GPS radio is necessary to extend the phone's battery life. In addition to the power consumed by Bluetooth and GPS, data upload from the phone via GPRS can also draw a large amount of power, particularly when the phone is far from a cell base station. A challenge is therefore to reduce the use of these radios without significantly impacting the application experience. Currently, the Symbian version of JME does not provide APIs to power cycle (i.e., toggle on and off) the Bluetooth and GPS radios to implement an efficient radio duty-cycle strategy.

The sensing and classification algorithms that run on the phone can also consume a considerable amount of energy if left unchecked. As discussed in Section 2.5, sampling the phone's microphone and running a discrete Fourier transform on the sound sample uses more power than sampling the accelerometer and classifying the accelerometer data. Given this, the only way to reduce energy at the application layer is to design a sensing duty-cycle that samples sensors less frequently and avoids the use of the radios for communications or acquisition of satellite signals for location coordinates.

2.2.2 Architectural Design Issues

In response to the observations discussed above we design the CenceMe application using split-level classification and power aware duty-cycling. We also develop the application with software portability in mind.

Split-Level Classification. The task of classifying streams of sensor data from a large number of mobile phones is computationally intensive, potentially limiting the scalability of the system. With this in mind, we propose the idea of pushing some classification to the phone and some to the backend servers. However, some classifiers require data that is only available at the server (e.g., for multiple users in the case of the social context classification discussed in Section 2.4.2). We call the output of the classification process on the phone *primitives*. When primitives arrive at the backend they are stored in a database and are ready to be retrieved for a second level of more complex classification. The classification operation on the backend returns *facts*, which are stored in a database from where they can be retrieved and published. With the split-level classification approach some of the classification can be done on the phone with the support of the backend, or under certain circumstances done entirely on the phone.

CenceMe's split-level design offers a number of important advantages: *i)* support of *customized tags*. A customized tag is any form of activity, gesture, or classified audio primitive that the user can bind to a personal meaning. For example, a customized tag could be created by a user by associating a certain movement or gesture of the phone (e.g., the phone being moved along an imaginary circle)

with a user supplied meaning or action, e.g., going to lunch. After associating the tag "lunch" with the action, the next time the user repeats the action the user's presence state "lunch" is recognized, uploaded, and shared with their social network. This technique gives the user the freedom to build her own classified state beyond a set of defaults offered by CenceMe, hence, it provides extensibility of the application; *ii)* resiliency to cellular/WiFi radio dropouts. By pushing the classification of primitives to the phone, the primitives are computed and buffered when there is no or intermittent radio coverage. Primitives are stored and uploaded in batches when the radio coverage becomes available; *iii)* minimization of the sensor data the phone sends to the backend servers improving the system efficiency by only uploading classification derived-primitives rather than higher bandwidth raw sensed data; *iv)* reduction of the energy consumed by the phone and therefore monetary cost for the data cellular connection by merging consecutive uploads of primitives; and finally *v)* negation of the need to send raw sensor data to the backend, enhancing the user's privacy and data integrity.

As discussed in Section 2.4, we design the classifiers that produce the primitives to be lightweight in order to match the capabilities of the phone.

Power Aware Duty-Cycle. To extend the battery lifetime of the phone when running the CenceMe application we apply scheduled sleep techniques to both the data upload and the sensing components. This leads to the following question: how long can the sensors, Bluetooth, GPS, and communications upload be in a sleep mode given that the larger the sleep interval the lower the classification responsiveness of the system? Typically, a real time sensing system would supply sensor data using a high rate duty-cycle. However, such an approach would conflict with energy conservation needs. Our approach is based on a duty-cycle design point that minimizes sampling while maintaining the application's responsiveness, as judged by users. This design strategy allows CenceMe to operate as near to real-time as possible; that is, some system delay is introduced before a person's sensing presence is updated on the backend servers. In the case of the current implementation the introduced delay varies according to the type of presence being inferred. The introduction of delay to improve the overall energy efficiency of the system makes good sense given the goal of CenceMe to allow buddies in social networks to casually view each other's sensing presence. For example, knowing that a buddy is in a conversation one minute after the actual conversation began seems reasonable. Other activities may allow even greater introduced latency; for example, people remain at parties for periods typically greater than five minutes or more, therefore, the delay introduced by the classifier in this case has little effect on the accuracy of the system status reports. In Section 2.5.2 we present the CenceMe system performance evaluation under varying upload and sensing duty-cycles to best understand these tradeoffs. In Section 2.7 we discuss results from the user study that indicate that even though users view their buddies status via the CenceMe portal infrequently they expect current information when viewed to be accurate and timely. This lends itself to a design that senses at an even lower duty-cycle on average but temporarily increases the sensing rate when a buddy's page is accessed. This results in bandwidth and storage capacity improvements.

Software Portability. To design for better software portability we push as much as we can to JME. We follow this design goal to maximize software re-usability given that the majority of

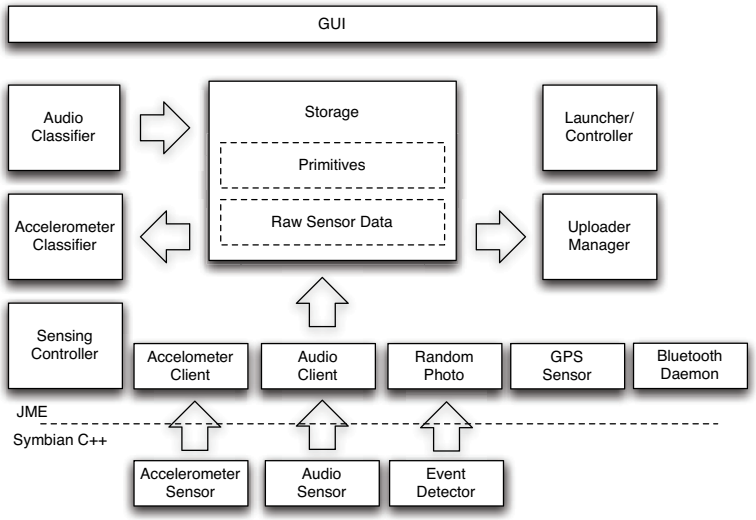


Figure 2.1: Architecture of the CenceMe phone software.

modern mobile phones use a Java virtual machine to support JME programs. However, because of the API limitations discussed earlier, a number of components need to be implemented directly using native Symbian APIs to support the necessary features offered by the phone but not available through JME.

2.3 CenceMe Implementation

In this section, we present the CenceMe implementation details. The CenceMe application and system support consists of a software suite running on Nokia N95 mobile phones and backend infrastructure hosted on server machines. The software installed on the phones performs the following operations: sensing, classification of the raw sensed data to produce primitives, presentation of people's presence directly on the phone, and the upload of the primitives to the backend servers. Primitives are the result of: *i)* the classification of sound samples from the phone's microphone using a discrete Fourier transform (DFT) technique and a machine learning algorithm to classify the nature of the sound; *ii)* the classification of on board accelerometer data to determine the activity, (e.g., sitting, standing, walking, running); *iii)* scanned Bluetooth MAC addresses in the phone's vicinity; *iv)* GPS readings; and finally, *v)* random photos, where a picture is taken randomly when a phone keypad key is pressed or a call is received. Classification algorithms that infer more complex forms of sensing presence (i.e., facts) run on backend machines, as discussed in Section 2.4.2.

2.3.1 Phone Software

Figure 2.1 shows the CenceMe software architecture for the Nokia N95 phone. The phone architecture comprises the following software components:

Symbian Servers. The accelerometer sensor, audio sensor, and event detector sensor are Symbian

C++ modules that act as daemons producing data for corresponding JME client methods. Their function is, respectively: polling the on board accelerometer sensor, sampling the phone's microphone, and detecting incoming/outgoing calls and keypad key presses. The sensed data is sent to the JME methods through a socket. Events detected by the event detector daemon are used by the random photo module at the JME level to generate random pictures, to trigger a photo upon an incoming phone call or to signal the application that it has to restart after a phone call for reliability reasons.

Bluetooth Daemon. This component resides at the JME level and is used to perform an inquiry over the Bluetooth radio to retrieve the MAC addresses of any neighboring Bluetooth nodes. The MAC addresses of the neighboring nodes are used to determine if there are CenceMe phones in the area at the time of the inquiry.

Accelerometer Client. This component is written in JME and connects through a socket to the accelerometer sensor to retrieve the accelerometer data byte stream. The byte stream is stored in local storage and retrieved by the activity classifier to compute the activity primitive, as discussed in Section 2.4.1.

Audio Client. This JME client component connects through a socket to the Symbian audio server to retrieve the audio byte stream that carries the PCM encoded representation of the sound sample. The byte stream is stored in local storage and retrieved by the audio classifier to compute the audio primitive, as discussed in Section 2.4.1.

Random Photo. This JME module is designed to trigger the capture of a photo upon detection of incoming calls or pressed keypad keys. The events are received through a socket from the event detector daemon. When the picture is taken it is stored locally until the next upload session.

GPS. The JME GPS implementation supplies a callback method that is periodically called by the Nokia GPS daemon to provide the geographical location of the phone. The GPS coordinates are stored locally and then uploaded to the backend servers.

Sensing Controller. This component is responsible for orchestrating the underlying JME sensing components. The sensing controller starts, stops, and monitors the sensor clients and the Bluetooth manager and GPS daemon to guarantee the proper operation of the system.

Local Storage. This component stores the raw sensed data records to be processed by the phone classifiers. As the classification of raw data records is performed, the data records are discarded, hence none of the sampled data persists on the phone. This is particularly important to address the integrity of the data and the privacy of the person carrying the phone since none of the raw sensed data is ever transferred to the backend. Primitives, GPS coordinates, and Bluetooth scanned MAC addresses are stored in local storage as well, waiting for an upload session to start.

Upload Manager. This component is responsible for establishing connections to the backend servers in an opportunistic way, depending on radio link availability, which can be either cellular or WiFi. It also uploads the primitives from local storage and tears down the connection after the data is transferred. Details about how the upload manager interacts with the backend are discussed in Section 4.2.2.

Figure 2.2: ClickStatus on the Nokia N95.

Privacy Settings GUI. The privacy settings GUI allows the user to enable and disable the five sensing modalities supported on the phone, (viz. audio, accelerometer, Bluetooth, random photo, and GPS). Users can control the privacy policy settings from the phone and the CenceMe portal. By doing so users determine what parts of their presence to share and who they are willing to share sensing presence with or not as the case may be.

ClickStatus. To complement the full visualization of current and historical sensing presence available via the CenceMe portal (a screenshot of the portal is shown in [93]), we developed ClickStatus, a visualization client that runs on the mobile phone. The sensing presence is rendered as both icons and text on the phone GUI, as shown in Figure 2.2. The presence rendered by ClickStatus is subject to the same privacy policies settings as when viewed using the CenceMe portal.

After a user logs in with their CenceMe credentials, they are presented with a list of their CenceMe buddies downloaded from the CenceMe server. CenceMe buddies are Facebook friends running CenceMe on their N95. While this is always done at start up, a user has the ability to refresh their buddy list at any time via a menu command option. By highlighting and selecting a buddy from buddy list, a user triggers ClickStatus to fetch via GPRS or WiFi the latest known sensing presence for the selected buddy from the CenceMe server. This presence is displayed on a separate result screen; from there a user can either exit to return to their buddy list or refresh the currently displayed buddy's presence.

WatchTasks. The purpose of WatchTasks is to restart any process that fails. WatchTasks also serves several other ancillary purposes including: *i)* launching CenceMe when the phone is turned on; *ii)* starting the CenceMe application software components in the correct order; *iii)* restarting the CenceMe midlet after a phone call is complete. This is detected when the event detector daemon exits, signaling the end of a call; *iv)* restarting all support daemons when CenceMe fails. Such action is necessary when we cannot reconnect to specific daemons under certain failure conditions; and finally *v)* restarting all the CenceMe software components at a preset interval to clear any malfunctioning threads.

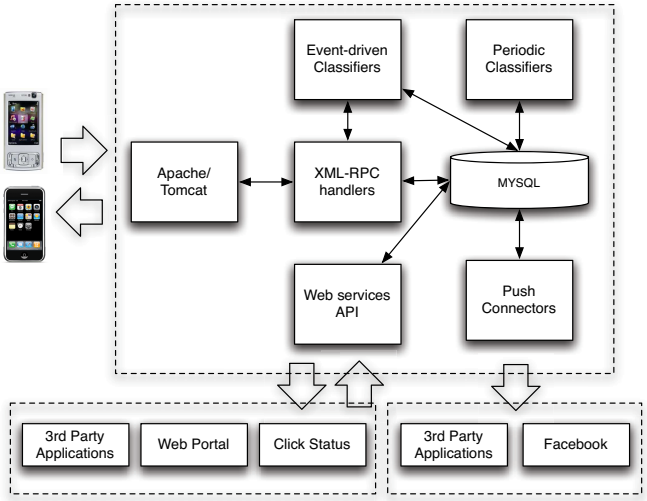


Figure 2.3: Software architecture of the CenceMe backend.

The CenceMe phone suite uses a threaded architecture where each JME component shown in Figure 2.1 is designed to be a single thread. This ensures that component failure does not compromise or block other components.

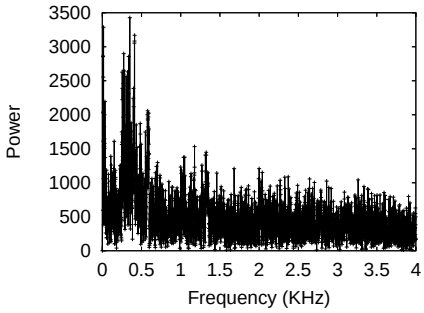
2.3.2 Backend Software

The CenceMe backend software architecture is shown in Figure 2.3. All software components are written in Java and use Apache 2.2 and Tomcat 5.5 to service primitives from phones and the application requests from the CenceMe portal, ClickStatus, and Facebook. Communications between the phone and the backend uses remote procedure calls implemented by the Apache XML-RPC library on the server. Requests are handled by Java servlets in combination with a MySQL database for storage.

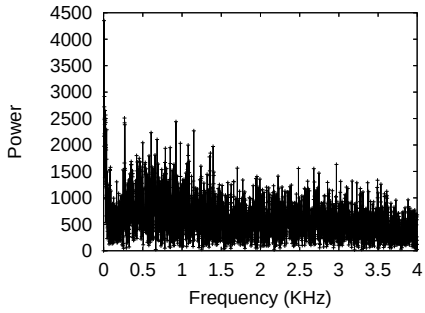
Phone $\Leftrightarrow$ Backend Communications. Data exchange between the phone and the backend is initiated by the phone at timed intervals whenever the phone has primitives to upload. Primitives are uploaded through XML-RPC requests. Once primitives are received at the backend they are inserted into the MySQL database.

Backend-to-phone communications such as in the significant places service described in Section 2.4.2 are piggybacked on both: *i)* the return message from XML-RPC requests initiated by the phone for primitive upload or periodic ping messages that the phone sends with an ad-hoc XML-RPC control message; and *ii)* the XML-RPC acknowledgment sent to the phone in response to a primitive upload.

Presence Representation and Publishing. CenceMe presence is represented through a set of icons that capture the actual presence of a person in an intuitive way. For example, if a person is driving a car they are represented by the car icon; if a person is engaged in a conversation, an icon of two people talking represents the state. CenceMe publishes presence by means of either a “pull” or “push” approach. Popular applications such as Facebook and MySpace require a push approach.



(a) DFT of a human voice sample registered by a Nokia N95 phone microphone.



(b) DFT of an audio sample from a noisy environment registered by a Nokia N95 phone microphone.

Figure 2.4: DFT of audio samples.

This allows content to be inserted via some variant of a HTTP transported markup language (e.g., FBML, XML). Other applications such as Skype, Pidgin, and iGoogle require a pull mechanism to make content available. The CenceMe backend supports pull-based data publishing by exposing a standard web service based API. This API is also used to support the data needs of CenceMe components such as ClickStatus and the CenceMe portal. Push-based publishing is supported by the PushConnector component shown in Figure 2.3. This component handles the generic operation of pushing CenceMe presence based on user preferences to a number of applications. For the Facebook implementation, three Facebook widgets are offered to expose a subset of the functionality available on the portal, namely, BuddySP, Sensor Status, and Sensor Presence. Buddy SP is a buddy list replacement widget that lists CenceMe friends for user navigation. It is the same as the standard widget that lists friends within Facebook but augments this list with a mini-sensor presence icon view. Sensor Status provides automated textual status message updates such as “Joe is at work, in a conversation, standing”. Finally, Sensor Presence provides a simplified version of the user’s current status through an iconized representation of the user’s presence.

2.4 CenceMe Classifiers

In this section, we discuss the algorithms used by the CenceMe classifiers running on the phone and the backend according to the split-level classification design discussed earlier.

2.4.1 Phone Classifiers

Audio classifier. The audio classifier retrieves the PCM sound byte stream from the phone’s local storage and outputs the audio primitive resulting from the classification. The primitive is stored back in local storage (see Figure 2.1). This audio primitive indicates whether the audio sample represents human voice and is used by backend classifiers such as the conversation classifier, as discussed in Section 2.4.2.

The audio classification on the phone involves two steps: feature extraction from the audio

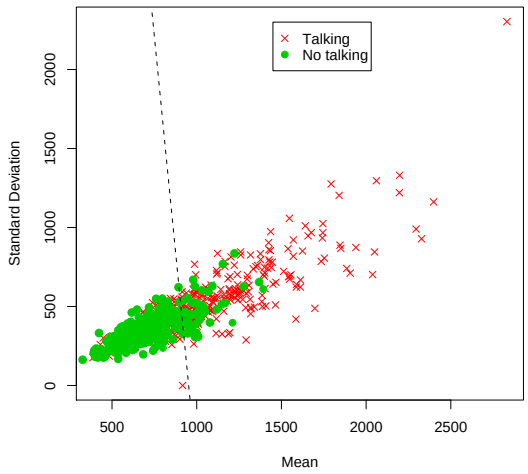


Figure 2.5: Discriminant analysis clustering. The dashed line is determined by the discriminant analysis algorithm and represents the threshold between talking and not talking.

sample and classification. The feature extraction is performed by running a 4096 bin size discrete Fourier transform (DFT) algorithm. A fast Fourier transform (FFT) algorithm is under development.

An extensive a-priori analysis of several sound samples from different people speaking indicated that Nokia N95 sound streams associated with human voice present most of their energy within a narrow portion of the 0-4 KHz spectrum. Figures 2.4(a) and 2.4(b) show the DFT output from two sound samples collected using the Nokia N95. The plots show the capture of a human voice, and the sound of an environment where there is not any active conversation on-going, respectively. It is evident that in the voice case most of the power concentrates in the portion of spectrum between $\sim$250 Hz and $\sim$600 Hz. This observation enables us to optimize the DFT algorithm to be efficient and lightweight by operating in the $\sim$250 Hz to $\sim$600 Hz frequency range. Classification follows feature extraction based on a machine learning algorithm using the supervised learning technique of discriminant analysis. As part of the training set for the learning algorithm we collected a large set of human voice samples from over twenty people, and a set of audio samples for various environmental conditions including quiet and noisy settings.

The classifier's feature vector is composed of the mean and standard deviation of the DFT power. The mean is used because the absence of talking shifts the mean lower. The standard deviation is used because the variation of the power in the spectrum under analysis is larger when talking is present, as shown in Figure 2.4. Figure 2.5 shows the clustering that results from the discriminant analysis algorithm using the mean and standard deviation of the DFT power of the sound samples collected during the training phase. The equation of the dashed line in Figure 2.5 is used by the audio classifier running on the phone to discern whether the sound samples comes from human voice or a noisy/quite environment with 22% mis-classification rate. Audio samples misclassified as voice are filtered out by a rolling window technique used by the conversation classifier that runs

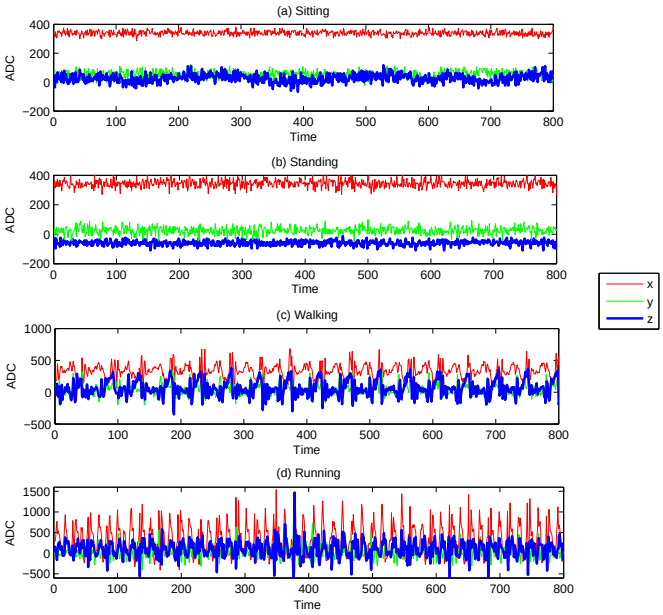


Figure 2.6: Accelerometer data collected by the N95 on board accelerometer when the person carrying the phone performs different activities: sitting, standing, walking, and running.

on the backend, as discussed in Section 2.4.2. This boosts the performance fidelity of the system for conversation recognition.

Activity classifier. The activity classifier fetches the raw accelerometer data from the phone's local storage (see Figure 2.1), and classifies this data in order to return the current activity, namely, sitting, standing, walking, and running. The activity classifier consists of two components: the preprocessor and the classifier itself.

The preprocessor fetches the raw data from the local storage component and extracts features (i.e., attributes). Given the computational and memory constraints of mobile phones, we use a simple features extraction technique which prove to be sufficiently effective, rather than more computationally demanding operations such as FFT. The preprocessor calculates the mean, standard deviation, and number of peaks of the accelerometer readings along the three axes of the accelerometer.

Figure 2.6 shows the raw N95 accelerometer readings along the three axes for sitting, standing, walking, and running for one person carrying the phone. As expected, the sitting and standing traces are flatter than when the person is walking and running. When standing, the deviation from the mean is slightly larger because typically people tend to rock a bit while standing. The peaks in the walking and running traces are a good indicator of footstep frequency. When the person runs a larger number of peaks per second is registered than when people walk. The standard deviation is larger for the running case than walking. Given these observations, we find that the mean, standard deviation, and the number of peaks per unit time are accurate feature vector components, providing high classification accuracy. Because of lack of space, we do not report similar results to those shown in Figure 2.6 for other people. However, we observe strong similarities in the behavior of

the mean, standard deviation, and the number of peaks for the accelerometer data across different individuals.

Our classification algorithm is based on a decision tree technique [94][95]. The training process of the classifier is run off-line on desktop machines because it is computationally costly. In order to maximize the positive inference of an individual's activity, prior work suggests that the best place on the body to carry a phone is the hip [26]. After interviewing the participants in our user study, we conjecture that most of people carry their phones in their pants pockets, clipped to a belt or in a bag. We collected training data from ten people that randomly placed the mobile phone inside the front and back pockets of their pants for several days. We plan to consider other usage cases in future work.

At the end of the training phase, we feed the training set to the J48 decision tree algorithm, which is part of the WEKA workbench [96]. The output of the decision tree algorithm is a small tree with depth three. Such an algorithm is lightweight and efficient. The time needed by the preprocessor and the classifier to complete the classification process is less than 1 second on average running on the Nokia N95.

2.4.2 Backend Classifiers

Backend classifiers follow the split-level classification design and generate facts based on primitives provided by the phone or facts produced by other backend classifiers. Facts represent higher level forms of classification including social context (meeting, partying, dancing), social neighborhood, significant places, and statistics over a large group of data (e.g., does a person party more than others, or, go to the gym more than others?). However, some of the classifiers (e.g., conversation and CenceMe neighborhood) will eventually be pushed down to the phone to increase the system classification responsiveness. In this case, the primitives would still be uploaded to the backend in order to make them available to other backend classifiers.

Backend classifier processing is invoked in two ways: either event triggered or periodic. An example of an event triggered classifier is the "party" classifier: it receives as input the primitives from the phone that contain the volume of an audio sample and the activity of the user and returns whether the person is at a party and dancing. Along with trigger based classifiers there is a collection of periodically executed classifiers. An example of such classifiers is the "Am I Hot" classifier that runs periodically according to the availability of data in a window of time, (i.e., day long data chunk sizes).

In what follows, we describe the backend classifiers and their implementation in more detail.

Conversation Classifier. This classifier's purpose is to determine whether a person is in a conversation or not, taking as input the audio primitives from the phone. However, given the nature of a conversation, which represents a combination of speech and silences, and the timing of sampling, the audio primitive on the phone could represent a silence during a conversation. Thus, the phone's audio primitives are not accurate enough to determine if a person is in the middle of a conversation. To address this the backend conversation classifier uses a rolling window of N phone audio primi-

tives. The current implementation uses N=5 to achieve classification responsiveness, as discussed in Section 2.5.1.

The rolling window filters out pauses during a conversation to remain latched in the conversation state. The classifier triggers the "conversation" state if two out of five audio primitives indicate voice. The "no conversation" state is returned if four out of five audio primitives indicate a "no voice". We determined experimentally that fewer samples are needed to trigger the conversation state than no conversation state. We therefore design the conversation classifier following an asymmetric strategy that quickly latches into the conversation state but moves more conservatively out of that state. We made this choice because if the conversation classifier can be used as a hint to determine if a person can be interrupted (for instance with a phone call), then we only want to drop out of conversation state when the conversation has definitely ended.

The accuracy of the conversation classifier is discussed in Section 2.5.1.

Social Context. The output of this classifier is the social context fact, which is derived from multiple primitives and facts provided by the phone and other backend classifiers, respectively. The social context of a person consists of: *i)* neighborhood conditions, which determines if there are any CenceMe buddies in a person's surrounding area or not. The classifier checks whether the Bluetooth MAC addresses scanned by the phone, and transmitted to the backend as a primitive are from devices belonging to CenceMe buddies (i.e., the system stores the Bluetooth MAC addresses of the phones when CenceMe is installed); *ii)* social status, which builds on the output of the conversation and activity classifiers, and detected neighboring CenceMe buddies to determine if a person is gathered with CenceMe buddies, talking (for example at a meeting or restaurant), alone, or at a party. For example, by combining the output of the conversation classifier, the activity primitive, and neighboring Bluetooth MAC addresses a person might be classified as sitting in conversation with CenceMe friends. Social status also includes the classification of partying and dancing. In this case a combination of sound volume and activity is used. We use a simple approach that uses an audio volume threshold to infer that a person is at a party or not. Training for this is based on a few hours of sound clips from live parties using the N95 microphone. We also take a simple approach to the classification of dancing. We determine a person is dancing if the person is in the "party" state and the activity level is close to running, given that the accelerometer data trace for running is close to dancing. Although we realize the definition of social context is somewhat simplistic and could be improved, this is a first step toward the representation of people's status and surroundings in an automated way.

Mobility Mode Detector. We employ GPS location estimates as input to a mobility mode classifier [97, 48]. This classification is currently only binary in its output, classifying the mobility pattern as being either traveling in a vehicle or not (i.e., being stationary, walking, running). We use a simple feature vector based on multiple measures of speed; that is, using multiple distance/time measurements for variable sizes of windowed GPS samples and the built-in speed estimation of the GPS device itself. The classifier is built with the JRIP rule learning algorithm, as implemented in WEKA [96], based upon manually labeled traces of GPS samples. We compensate for any inaccuracy in

GPS samples by filtering based on the quality measures (i.e., horizontal dilution of precision and satellite counts) and outlier rejection relative to the estimates of previous and subsequent GPS samples.

Location Classifier. The function of this component is to classify the location estimates of users for use by other backend classifiers. GPS samples are filtered based on quality (as discussed above) to produce a final location estimate. Classification is driven based on bindings maintained between a physical location and a tuple containing: *i)* a short textual description; *ii)* an appropriate icon representation; and *iii)* a generic class of location type (i.e., restaurant, library, etc.). Bindings are sourced from GIS databases and CenceMe users. We use the Wikimapia [98] for GIS data in our implementation. Relying solely on GIS information limits the richness of shared presence. Typically, people tend to spend a larger proportion of their time in relatively few locations. This motivates the idea of user-created bindings. CenceMe allows users to insert their own bindings via either the portal or the phone. Using the phone, users can manually bind a location when they visit it. Similarly, users can use the portal to also add, edit or delete bindings manually. CenceMe also provides the ability to learn significant places in an automated manner in contrast to the manual bindings discussed above. New bindings learned by the system are based on the mobility pattern of the user. This aspect of CenceMe directly builds on the existing work in location trace analysis referred to as significant places [99] [100]. In CenceMe we perform k-means clustering using WEKA [96] where the parameters of the clustering algorithm are determined experimentally. Once a potential significant place is discovered the next time the person enters that location the phone prompts the person's mobile phone to confirm or edit the details of the location. Default labels and icons are initially based upon the most popular nearest known existing binding defined by the user or CenceMe buddies. To reduce the burden on the users to train the classifier with their own bindings we structure the classifier to initially borrow existing bindings from their CenceMe buddies [79].

Am I Hot. Making the large volumes of data collected by CenceMe easily digestible to users is a challenge. We address this challenge using a series of simple and meaningful metrics that relate historical trends in user data to either recognizable social stereotypes or desirable behavioral patterns. These metrics are calculated on a daily basis and users view patterns in their own data and compare themselves with their buddies. The metrics include the following: *i)* nerdy, which is based on individuals with behavioral trends such as being alone (from the Bluetooth activity registered by the person's phone), spending large fractions of time in certain locations (e.g., libraries) and only infrequently engaging in conversation; *ii)* party animal, which is based on the frequency and duration with which people attend parties and also takes into account the level of social interaction; *iii)* cultured, which is largely location based, being driven by the frequency and duration of visits to locations such as theaters and museums; *iv)* healthy, which is based upon physical activities of the user (e.g., walking, jogging, cycling, going to the gym); and finally, *v)* greeny, which identifies users having low environmental impact, penalizing those who drive their cars regularly while rewarding those who regularly walk, cycle or run.

Table 2.1: CenceMe activity classifier confusion matrix

	Sitting	*Standing*	*Walking*	*Running*
Sitting	0.6818	0.2818	0.0364	0.0000
Standing	0.2096	0.7844	0.0060	0.0000
Walking	0.0025	0.0455	0.9444	0.0076
Running	0.0084	0.0700	0.1765	0.7451

Table 2.2: CenceMe conversation classifier confusion matrix

	Conversation	*Non-Conversation*
Conversation	0.8382	0.1618
Non-Conversation	0.3678	0.6322

2.5 System Performance

In this section, we present an evaluation of the CenceMe application and system support. We start by discussing the performance of the CenceMe classifiers and then present a set of detailed power, memory, and CPU benchmarks. Finally, we present the results from a detailed user study.

2.5.1 Classifiers Performance

We examine the classifiers performance based on a small-scale supervised experiments. We discuss classifier accuracy, and the impact of mobile phone placement on the body, environmental conditions, and sensing duty-cycle. The results are based on eight users who annotate their actions over a one week period at intervals of approximately 15 to 30 minutes, unless otherwise stated. Annotations act as the ground truth for comparison with classifier outputs. The ground truth data is correlated to the inference made by the CenceMe classifiers. This data is collected at different locations and by carrying the mobile phone in various positions on the body. Tables 2.1, 2.2 and 2.3 show the confusion matrices for the activity, conversation, and mobility classifiers, respectively, over a one week period. These reported values represent good approximations; the human annotations may be inaccurate or incomplete at times.

General Results

While the activity inference accuracy reported in Table 2.1 is up to 20% lower than that reported using custom hardware [27], we achieve our results using only the accelerometer on a Nokia N95 and engineering the system to be power efficient and work around the resource limitations discussed earlier. We find that our classifier has difficulty differentiating sitting and standing given the similarity in the raw accelerometer traces, as shown in Figure 2.6. We observe that variations in locale (e.g., office, restaurant) and people (e.g., body type, weight) do not significantly impact the activity classification performance.

The conversation classification accuracy reported in Table 2.2 is high, but the classifier also suffers from a relatively high rate of false positives. This is due to a combination of classifier design

Table 2.3: CenceMe mobility mode classifier confusion matrix

	Vehicle	*No Vehicle*
Vehicle	0.6824	0.3176
No Vehicle	0.0327	0.9673

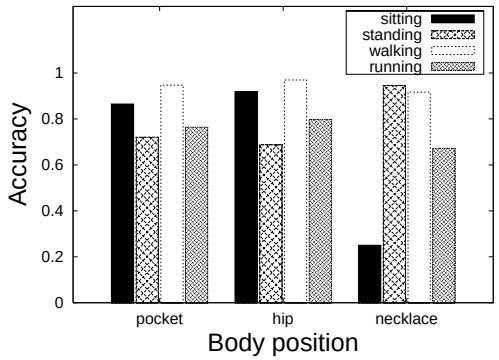


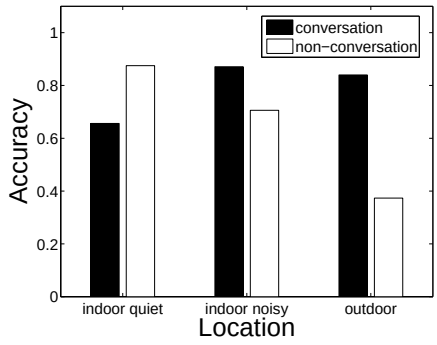
Figure 2.7: Activity classification vs. body position.

and "mis-annotation" by participants. The classifier reports conversation even if the person carrying the phone is silent but someone is talking nearby. Naturally, participants often did not account for this fact. Furthermore, due to the asymmetric state latching for the conversation classifier discussed in Section 2.4.2, the classifier remains in the conversation state for a longer time than the real conversation duration, generating false positives.

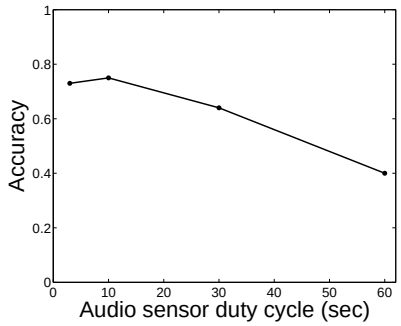
Impact of Phone Placement on the Body

While mobile phone placement on the body is a personal choice, prior work has shown body placement to affect the accuracy of activity inference [26]. We assess the impact on classification when the Nokia N95 is placed at different places on the body, namely, in a pocket, on a lanyard, and clipped to a belt. Classification accuracy derived from the ground truth annotated data is shown in Figure 2.7. The pocket and belt positions produce similar results for all classified activities, while the lanyard position yields poor accuracy when classifying sitting, and a relatively lower accuracy for running. In follow-up laboratory experiments, we find that the length of the lanyard cord and the type of lanyard we provided to participants affect the results. If the lanyard is long the phone rests frequently on the body, particularly while walking and standing, allowing for accurate classification. However, even when seated a lanyard-mounted phone may swing from side to side with incidental torso movements, causing a mis-classification as standing or walking. Furthermore, running is sometimes classified as walking because the lanyard damps the accelerometer signatures that indicate running, compared to other body positions (e.g., belt, pocket) where the phone is more rigidly affixed to the body.

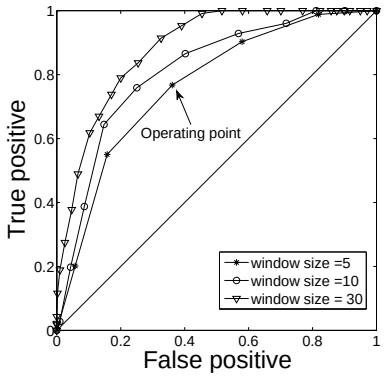
We find that conversation classification accuracy is much less sensitive to the body placement of the phone. When the phone is worn as a lanyard, conversation and no conversation are detected



(a) Conversation classifier in different locations.



(b) Conversation classifier accuracy with a variable duty-cycle.



(c) ROC curves for the conversation classifier.

Figure 2.8: Conversation classifier performance.

with 88% and 72% accuracy, respectively. The same test repeated with the phone in a pocket yields a classification accuracy of 82% for conversation and 71% for no conversation, despite the muffling effect of clothing.

Impact of Environment

We find activity classification accuracy to be independent of environment. Mobility classification is inherently not tied to a particular location but rather on transitions between locations. However, we do see an impact from the environment on conversation classification accuracy. Figure 2.8(a) shows the classification accuracy categorized by location, where the different locations are: outdoors, indoor noisy (i.e., an indoor location with background noise such as in a cafe or restaurant), and indoor quiet (i.e., with very low background noise such as at the library or office). The classifier detects conversation with more than an 85% success rate when in an indoor noisy environment. In outdoor scenarios there is an increase in false positives but the accuracy of detection of conversation, a design focus, remains high. Lower conversation detection accuracy in very quiet indoor environments occurs because the classifier is trained with the average case background noise. In a noisy environment there is an increase in power across all of the frequencies so a threshold set for

this environment in mind will be larger than if a very quiet environment is assumed. As a result, in very quiet environments fewer conversations are detected since the contribution of background noise is lower. These performance characteristics are a direct result of the audio classifier design, which attempts to reduce the use of the phone's resources.

Impact of Duty-Cycle

Applying a sleep scheduling strategy to the sensing routine is needed in order to increase the battery lifetime of the phone. Note that in Section 2.5.2 we discuss lifetime gains with a ten minute inter-sample time. However, this has a negative impact on the performance of the classifiers, particularly in detecting short-term (i.e., duration) events that occur between samples. For example, in Table 2.3, the vehicle state is only correctly detected 68% of the time. This lower accuracy is a product of shorter car journeys around town for durations less than the inter-sampling rate. This problem is aggravated by other factors such as the delay in acquiring good GPS-based positioning data. To investigate the impact of duty-cycling on conversation classification, we set up an experiment with eight users that periodically reprogrammed their phones with different duty-cycles while keeping a diary. Figure 2.8(b) shows the performance of the phone's conversation classifier as the microphone sensing duty-cycle varies. Each value represents the average of five trials. We see that there is little benefit in adopting a sleeping time smaller than 10 seconds. However, longer duty-cycles impact performance. We observe only a 40% accuracy using the conversation classification for a 60 second duty-cycle, which is the longest duty-cycle we considered experimentally.

A longer sensing duty-cycle also implies a reduction of the conversation classifier rolling window size to maintain the high responsiveness of the classifier. A smaller conversation classifier rolling window size leads to a higher mis-classification rate. This becomes apparent if we look at the Receiver Operating Characteristic (ROC) curves of the conversation classifier as shown in Figure 2.8(c). The ROC curves show the impact of the window size and threshold that triggers conversation (reflected in the curve shape) on the classifiers true positive and false positive rates. We use offline analysis to determine the output of the conversation classifier as we alter the window size and threshold value. We observe that the larger the window (i.e., N=10,30), the larger the true positives to false positives ratio becomes. In our current implementation, we adopt N=5 and an audio sensing rate of 30 seconds (our default operating point is labeled in the figure). With these parameters the worst-case conversation classification delay omitting communication delays is 1.5 minutes. On the other hand, if we used a window where N=30, which would give higher accuracy, we would get a delay of 9 minutes on average. This illustrates the trade off between sampling rate and classification speed. However, we choose to operate at a point in the design space that increases the true positive rate at the expense of being less accurate in the detection of non-conversation because the cost, from a user's perspective, of being wrong when detecting a conversation is larger than the cost of being wrong when detecting non-conversation.

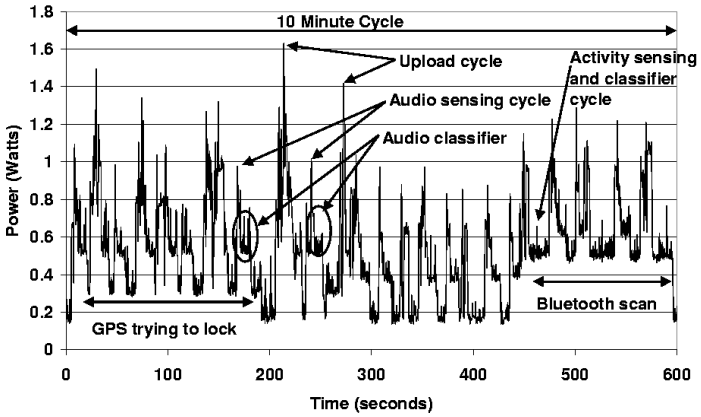


Figure 2.9: Details of the power consumption during a sampling/upload interval.

2.5.2 Power Benchmarks

Power measurements of CenceMe are made using the Nokia Energy Profiler, a standard software tool provided by Nokia specifically for measuring energy use of applications running on Nokia hardware. The profiler measures battery voltage, current, and temperature approximately every third of a second, storing the results in RAM.

Figure 2.9 shows the typical contribution of various sensors and classifiers to the overall energy budget during a ten minute sensing cycle. Bluetooth proximity detection requires a 120 second scan period to capture neighboring MAC addresses due to the cache flushing limitations of the Bluetooth API in JME. GPS location detection is inherently power hungry and takes time to acquire "a lock" when turned on. CenceMe allows 120 seconds for a lock to be acquired and then the N95 keeps the GPS activated for another 30 seconds (which is out of our control). The highest spikes shown on the plot are due to the upload of data which uses the cellular radio. The next highest spikes are due to sampling of audio data. The period of several seconds following the audio sample is where the audio classifier runs, using a relatively high amount of energy to compute a DFT. The accelerometer sampling and activity classification are fast and use little power. While this is a typical pattern of energy consumption there are other factors which can cause variations, including: distance to cell tower, environmental radio characteristics, the amount of data to upload, the number of Bluetooth neighbors, denial of resources due to the phone being in use for other purposes, network disconnections, sensor sample intervals, sample durations, upload interval, GPS lock time, and temperature.

Figure 2.10 shows the energy consumption measured with the profiler for sampling intervals ranging from 10 seconds to 60 seconds with power in Watts on the vertical axis. The second line and axis in the graph shows the latency in getting the facts to the backend as a function of the sample interval including the sample interval itself, classifier latency, and network delay. The audio classifier latency is actually a multiple of three times the values on this line since the classifier needs at least three facts from the phone in order to detect conversation and social setting. The horizontal axis shows the sampling interval for the accelerometer and audio. The proximity and GPS sensors

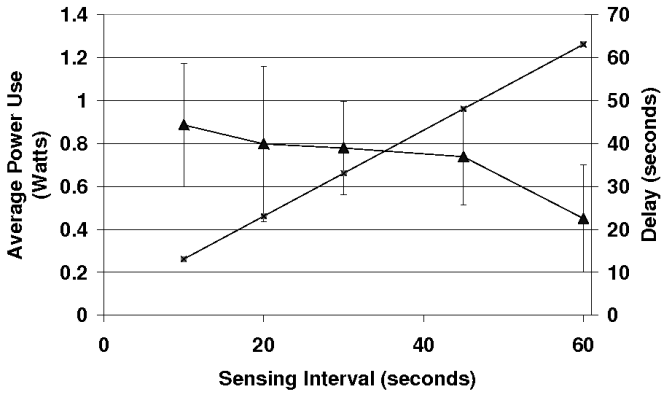


Figure 2.10: The tradeoff between energy consumption and data latency in CenceMe.

are sampled at ten times the x-axis value (e.g., a 60 second interval means Bluetooth and GPS are sampled at 600 seconds, or ten minute intervals).

The combination of the two lines show the tradeoff between energy use and data latency for any particular sampling interval. There is no optimal sampling interval since users will have different requirements at different times. For example, users may want a short sample interval when they are active, a slow interval when they are inactive, and a very slow interval when their phone is running out of energy. We are currently considering several methods of automatic adaptation of the sample rate based on sensor input and battery state, combined with a user preference selector that lets the user shift the emphasis between long battery life and greater data fidelity.

Overall battery lifetime running the entire CenceMe software suite on a fully charged N95 is measured five times by running the battery to depletion under normal use conditions while using no other applications on the phone. This results in 6.22 +/- 0.59 hours of usage. The reason for the large standard deviation is that there are many factors impacting battery life such as temperature, the number of calls and duration, the number of ClickStatus queries, range from cell towers when used, and the environmental and atmospheric conditions. Without the CenceMe software running, and the phone in a completely idle state, low power state power consumption is 0.08 +/- 0.01 Watt-Hours per hour. The CenceMe suite consumes 0.9 +/- 0.3 Watt-Hours per hour when running with no user interaction. The conversation and social setting classifier consumes 0.8 +/- 0.3 Watt-Hours per hour with all other parts of the CenceMe system idle. The activity classifier consumes 0.16 +/- 0.04 Watt-Hours per hour with all other parts of the CenceMe system idle. Any use of the phone to make calls, play videos or listen to music will reduce the runtime. While the approximately 6 hour lifetime is far below the idle lifetime of the Nokia N95, we have identified several areas where we believe we can significantly reduce power usage while also decreasing data latency, as discussed in Section 2.2.2.

Table 2.4: RAM and CPU usage for the CenceMe N95 client

	CPU	*RAM (MB)*
Phone idle	2% (+/- 0.5%)	34.08
Accel. and activity classif.	33% (+/- 3%)	34.18
Audio sampling and classif.	60% (+/- 5%)	34.59
Activity, audio, Bluetooth	60% (+/- 5%)	36.10
CenceMe	60% (+/- 5%)	36.90
CenceMe and ClickStatus	60% (+/- 5%)	39.56

2.5.3 Memory and CPU Benchmarks

We also carried out benchmark experiments to quantify the RAM and CPU usage of the CenceMe software running on the N95 using the Nokia Energy Profiler tool. For all measurements we enable the screen saver to decouple the resource occupation due to the CenceMe modules from that needed to power up the N95 LCD.

We start by measuring the amount of RAM and CPU usage when the phone is idle with none of the CenceMe components running. We then repeat the measurement when either the accelerometer sampling and activity classification or audio sampling and classification are active. Then we add each of the remaining CenceMe modules until the whole software suite is running. The results are shown in Table 2.4. As expected, audio sampling and feature vector extraction require more computation than the other components. This is in line with the power measurements result shown in Figure 2.9 where audio sampling and processing are shown to use a relatively high amount of energy. We also note that the memory foot print does not grow much as components are added. Together CenceMe and ClickStatus occupy 5.48MB of RAM.

2.6 The iPhone as a Mobile Platform for People-Centric Sensing Applications

In this Section we discuss the performance of the first generation Apple iPhone when using the on-board sensors to realize mobile sensing applications. This is one of the early studies that evaluates the capabilities of the first iPhone generation and its ability to support people-centric sensing applications. The iOS features, including its ease of use, rich UI, and efficient application distribution system through the Apple App Store makes the iPhone an appealing platform for development of new mobile applications. A natural question for our community is what are the trade-offs when implementing and deploying a sensing application using the iPhone; more specifically:

- How easy is it to program a sensing application on the iPhone?
- What are the pros and cons of the iPhone in comparison to other sensor capable mobile platforms?
- What is the energy profile when the iPhone's sensors, WiFi and cellular radio are involved in realizing the application?

- What is the processing performance of the iPhone when running signal processing algorithms such as fast fourier transform, a common tool used to interpret audio and accelerometer sensor data?

We address these questions in this Section. While the presentation of our results is limited due to space, we provide a short qualitative comparison of a number of devices used for mobile sensing including the first generation Apple iPhone, Nokia N95, and Intel Mobile Sensing Platform (MSP). The main contribution of this study is the observations and insights when running CenceMe, a representative people-centric sensing application [14], on the iPhone. Specifically, we quantitatively evaluate the first generation iPhone's computational capability, energy profile, and localization accuracy when running CenceMe. We believe this study will be useful to the growing community of iPhone developers, particularly, those interested in building people-centric sensing applications.

2.6.1 Comparison of Mobile Sensing Platforms: iPhone, N95 and MSP

In what follows, we present a short qualitative comparison of the first generation Apple iPhone, Nokia N95 mobile phone, and the Intel Mobile Sensing Platform (MSP)[11]. All these devices are actively being used in support of mobile sensing applications and systems development. The N95 is one of the top-end Nokia mobile phones equipped with an accelerometer and GPS, while the MSP is representative of the class of embedded devices used for human activity recognition research [11]. A comparison of some of the technical details of the three devices is reported in Table 2.5. As shown in Table 2.5, all the three platforms present similar computational capabilities given similar processors, and comparable storage and ROM size. The RAM on the MSP is much smaller than on the iPhone and N95, which first and foremost are designed as mobile phones, hence the need to handle multiple processes at the same time including graphics computation. The MSP short-range radio technology is flexible allowing the implementation of advanced pairing algorithms between nodes while the use of the iPhone and N95's short-range radio is limited to simple neighbors interactions. The main difference between the three devices is represented by the sensing capability; specifically, the MSP outshines both the iPhone and the N95 in terms of number of available sensors. This is not surprising, given that the MSP is an embedded purpose-built platform for activity recognition. However, even with a reduced set of on board sensors, the iPhone and N95 are powerful devices and capable of inferring human activites - for example, we have implemented the full featured CenceMe application on the N95 [14] as well as a limited version on the iPhone [101]. However, providing mobile phones with more sensing capabilities (e.g., gyroscope) would greatly enhance the humans presence classification accuracy given the broader input to the classifiers feature vectors.

2.6.2 Programmability Characteristics of the iPhone

In what follows, we analyze the programmability characteristics of the first generation iPhone. Any third-party application is handled by the iPhone OS using a sandboxing model which does not allow the application to access some of the iPhone functionality (such as WiFi APIs or iTunes) for security

Table 2.5: Mobile devices specs comparison

	iPhone 1st gen	**Nokia N95**	**Intel MSP 430**
Processor	412 MHz ARM	330 MHz ARM	416 MHz Xscale
RAM	up to 70 MB	up to 128 MB	256 KB
ROM	20 MB	up to 160 MB	32 MB
Storage	up to 8GB/16GB	min-SD card (up to 8 GB)	mini-SD card (up to 8 GB)
Sensors	3-axis accel, mic, GPS	3-axis accel, mic, GPS	3-axis accel, mic, light, barometer, temp, IR, humidity, compass
Radio	WiFi	WiFi, Bluetooth	Bluetooth, Zigbee

reasons. A simplified version of a SQL database, namely *sqlite* [102], designed to run on resource constrained environments, is also supported as a means to ease application on-the-phone storage.

By following a systematic approach, we intend to answer the following question: what are the positive and negative aspects of the iPhone as a programmable platform? Although the first generation iOS presents a rich set of features making it on the surface a good platform for the development of sensing applications, it also provides some barriers in its current stage of development. In what follow, we briefly discuss the pros and cons of the current iPhone development environment.

- **Advantages**

- *The programming language*. The iPhone is programmed in Objective-C[92]. Objective-C is a superset of the C language, with some object oriented programming features. The advantage of Objective-C, over other languages such as Symbian adopted by Nokia, is that it is a quite simple language to learn and use. The rich set of iPhone APIs and the well designed iPhone emulator running on a desktop machine, makes iPhone programmability, UI design, and code debugging an efficient process for developers.

- *APIs*. Given the extensive documentation, a developer has easy access to comprehend available APIs. They are also well engineered in order to better abstract the developer from lower level components. An example is the location engine API: the API call returns data transparently to the user regardless of the location coordinates coming from WiFi, cellular triangulation, GPS, or a combination of them. The accelerometer and microphone API are easily manageable as well.

- *Indoor Localization*. By using WiFi [103] and cellular triangulation to determine the location, the first generation iPhone localization for indoor spaces is quite accurate, as discussed in Section 2.6.3. This is an important feature, for example, for mobile social networking applications considering that people spend most of their time in indoor locations.

- *User Interface*. The iPhone experience is greatly enhanced by the Cocoa-Touch layer architecture [92] that enables a pleasant user experience. This, complemented by the powerful graphics framework, currently makes the iPhone UI one of the best presentation layers of any mobile devices.

- *Application Distribution*. Apple provides an efficient way to distribute third-party applications to the public through the App Store. Once the application has been tested and approved by Apple, the application is posted on App Store from where it can be downloaded and automatically installed on any iPhone.

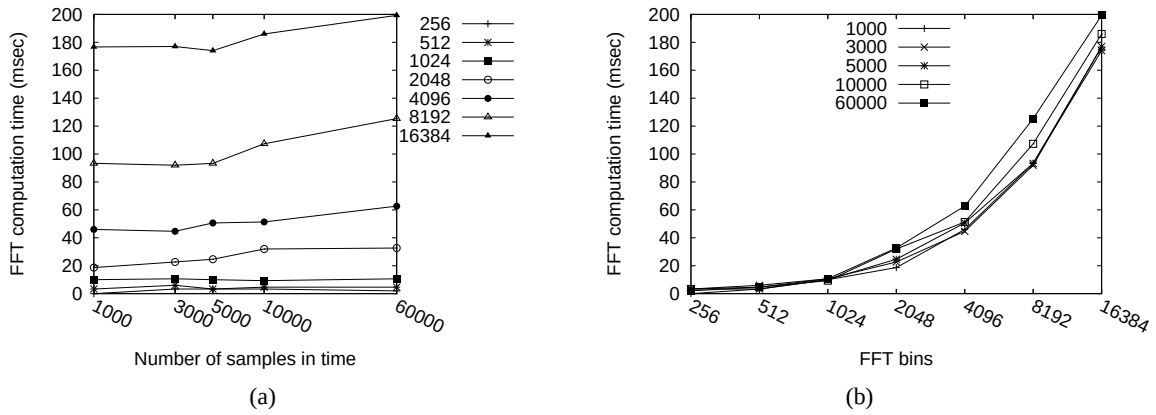


Figure 2.11: FFT computation time as a function of (a) the number samples in time while varying the FFT bin size (as shown in the legend) and (b) the FFT bin size while varying the number of samples in time (as shown in the legend).

- **Disadvantages**

- *Lack of daemon functionality*. The main drawback of the first generation iPhone is its lack of ability to run a third-party application in background mode. This is enforced by Apple for security reasons. This means that anytime the phone goes into sleep mode or the user launches another application, the currently running third-party application is terminated. Consequently, any sensing application cannot provide continuous sensor data feeds, generating just an intermittent data stream, thereby limiting the effectiveness of the sensing application. Apple's response to the lack of background capability is the *Push Notification Service* coming in the next releases of the SDK. With the Push technology, probes are sent by the Apple backend servers, which serve as relays for Push messages sent by a sender host to a receiver iPhone. As the receiver iPhone is woken up by the probe the user is asked by the iPhone OS whether to let the application run in response to the probe message or not. The latest iOS supports the application background mode, however some of the sensors, i.e., the accelerometer, are shutdown when an application is pushed to the background. This limits the accuracy of a mobile sensing application relying on a continuous stream of accelerometer data to perform activity recognition.

- *Short-range radio API limited capability*. With the first iOS generation it was not possible to have access to the Bluetooth or WiFi radio stack APIs and the only means to exchange information with neighboring iPhones is through the iPhone networking stack via the Bonjour service through WiFi. The networking capability was therefore limited and does not allow developers to build sophisticated pairing protocols. The latest iOS proposes a much more comprehensive and flexible set of networking APIs.

2.6.3 Performance Evaluation

In this section, we report initial results from a number of experiments aimed at evaluating the first generation iPhone computational capability, battery duration, and localization accuracy by using the

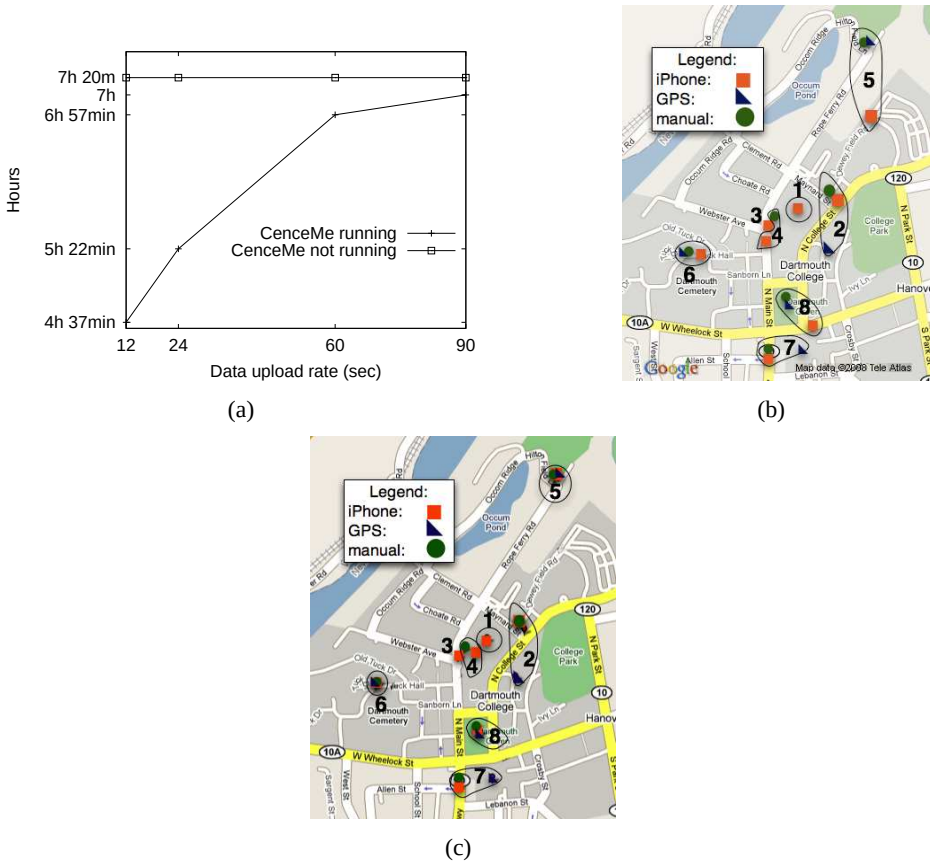


Figure 2.12: (a) Battery duration with and without CenceMe running while the iPhone screen saver is set to off; localization accuracy for eight different locations in the Dartmouth campus of (b) the old iPhone (no GPS), and (c) the iPhone 3G (with GPS).

original iPhone model, i.e., the one without GPS, and the iPhone 3G, which instead mounts a GPS.

Computational Capability. In order to evaluate the processing power of the iPhone we run a fast fourier transform (FFT) algorithm, which is part of the CenceMe software suite, and measure the iPhone computation time. The FFT computation evaluation is performed during normal CenceMe usage pattern [101]. The FFT we use is Kiss FFT [104], a well known open source high performance FFT library. We choose the FFT as a means to evaluate the iPhone under high computational load since the FFT is a common tool used in inference techniques applied to sensor data such as the accelerometer and audio data streams. As shown in Figure 2.11, the iPhone computation time up to 4096 FFT points is below 60 msec even for a large number, i.e., 60000, of sampled events in time. Many of the sensor data analysis algorithms make use of 512 - 2048 FFT points calculation which means that they could efficiently leverage the iPhone's computational power. Large data bins in time, up to 60000 samples in our experiment, could also be quite efficiently handled by the FFT on the iPhone in at most 200 msec.

Battery Lifetime. We perform some experiments to quantify the battery drain of the first generation iPhone when running CenceMe compared to the baseline power usage without CenceMe. We set the screen saver to off so that the phone never goes to standby mode.

Table 2.6: Localization accuracy for different places in the Dartmouth Campus - Legend: **C.S.** = Cellular Signal; **A** = Old iPhone accuracy (m); **B** = iPhone 3G accuracy (m); **C** = Garmin GPS accuracy (m); **D** = Old iPhone-Garmin GPS localization difference (m); **E** = iPhone 3G-Garmin GPS localization difference (m)

	Location	**WiFi**	**C.S.**	**A**	**B**	**C**	**D**	**E**
1	Computer Science Bld indoor	good	good	83	22	N/A	N/A	N/A
2	Computer Science Bld outdoor	good	good	44	17	14	29	36
3	Library outdoor	good	good	17	9	8	0	1
4	Library indoor	good	mediocre	13	5	N/A	N/A	N/A
5	Golf course	none	good	759	17	5	45	1
6	Engineering Bld	weak	weak	95	17	5	14	0
7	Main St.	none	weak	179	47	11	5	4
8	The Green	none	good	323	47	5	24	2

The battery duration for different data upload rates when CenceMe is running is compared to the duration when CenceMe is not running, as shown in Figure 2.12(a). With the phone's standby mode off and running CenceMe continuously, the battery lifetime spans between 4 hours and 37 min to 7 hours according to the upload rate. We then turn the screen saver back on and set it to 5 minutes and we run CenceMe with the following cycle: run for 5 minutes, let the screen saver go off, leave the screen saver up for 5 minutes, wake the phone up for 5 minutes and so on until the battery discharges completely. In this way, for the same upload rates, we obtain a phone usage time (meaning time available to play with CenceMe) between 4 hours 50 min and 5 hours 20 min whereas the battery overall lasts between 10 hours and almost 11 hours. This battery duration is similar to the duration obtained with iPhone usage patterns comparable to the one of our experiment running applications different from CenceMe.

This is because the prevalent battery drain is due to the large iPhone LCD screen rather than the networking activity for data transmission/reception operated by CenceMe.

Localization Accuracy. To evaluate the localization accuracy of both the first generation iPhone (without GPS) and the iPhone 3G (with GPS) we carry out the following experiment: we walk in the Dartmouth College campus with both the iPhone models and a Garmin eTrex GPS. We record the geographical coordinates from the Garmin and the iPhone devices at eight different locations. On the maps in Figure 2.12(b) and Figure 2.12(c), referring, respectively, to the old iPhone and iPhone 3G, eight clusters are shown, each indicating: the location manually tagged by the person carrying the devices, the location reported by the Garmin, and the location indicated by the iPhone. Since the pre-iPhone 3G localization engine uses WiFi [103] and cellular triangulation, whenever either the WiFi or the cellular coverage is poor the result is low localization accuracy.

This can be seen for locations associated to clusters 5 and 8 where there is poor WiFi and/or cellular coverage. In case of clusters 1 and 4, which are indoor locations where GPS performs poorly, the iPhone localization is more accurate given the high quality of the WiFi and cellular signal. Overall, since the small town of Hanover is not served by many cell towers, which would allow the iPhone localization triangulation algorithm to be more accurate, the iPhone estimates an accuracy between 13 and 759 meters, as shown in column A of Table 2.6. However, the actual distance

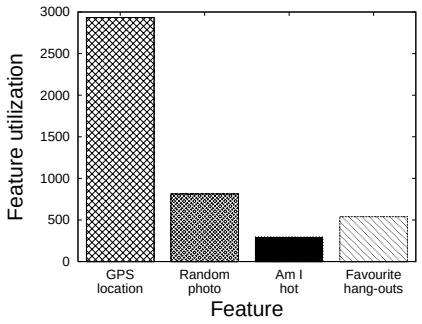
difference between the iPhone and GPS reported locations (as shown in column D of Table 2.6) is 45 m at most, indicating that the iPhone localization algorithm uses a conservative approach to estimate its accuracy. The GPS boosts the localization accuracy of the iPhone 3G, being particularly effective where there is a lack of WiFi coverage or the cellular signal is poor. This can be seen from columns B and E of Table 2.6 where, respectively, the error estimated by the iPhone 3G and the localization difference between the iPhone 3G and GPS is smaller than the old iPhone model case.

2.7 User Study

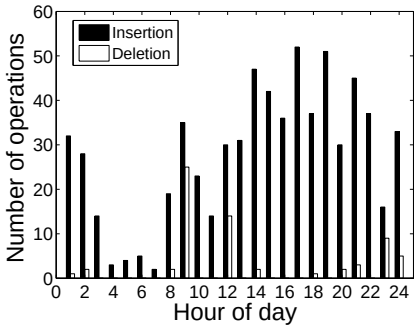
Because CenceMe is designed to be a social network we need to go beyond simple measures of system performance to best understand the utility of people-centric sensing applications such as CenceMe. Our goal is to bring CenceMe to the attention of potential users, ask them to use CenceMe and provide detailed feedback about their user experience by means of a survey. For this reason we conducted an "operational" experiment. The experiment conducted over a three week period involved 22 people. Participants were each given a Nokia N95 with the CenceMe software (including ClickStatus) and a free voice/data plan. Users had server side accounts and access to the CenceMe portal. While some of the users were friends we placed all users in the same buddy list as a means to create some community. The pool of candidates picked within the population of students and staff at our university was composed of 12 undergraduate students, 1 research assistant, 1 staff engineer, 7 graduate students, and 1 professor. The research assistant and four undergraduates have little computer science background. Sixteen participants are active Facebook users. Before discussing the detailed experience of users, we summarize some results from the user study:

- Almost all of the participants liked using CenceMe and its features. One user wrote: "*it's a new way to be part of a social network*".
- Facebook users are particularly active in terms of willingness to share detailed status and presence information with their friends.
- Privacy could be a concern but users are fine with sharing their presence status as long as they have the means to easily and efficiently control their privacy settings.
- CenceMe stimulates curiosity among users. Users want to know what other people are doing while on the move.
- CenceMe can aid people in learning their own activity patterns and social status.

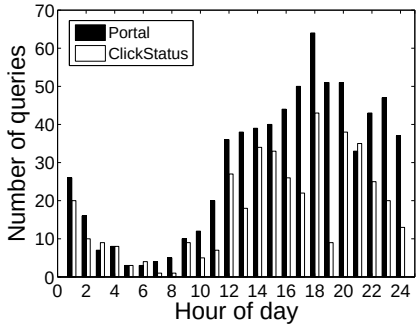
A new way to connect people. Almost all the participants find the idea of providing and viewing detailed information about people they are close to compelling, useful, and fun. In particular, location, activity/conversation, the historical log of the person's presence, random images, and social context are the features that people like the most. This pattern is confirmed in Figure 2.13(a), where the cumulative participants' feature utilization for different hours of the day derived from



(a) Utilization distribution across the different features.



(b) Comparison between the number of random pictures inserted into the database versus the number of pictures deleted.



(c) Comparison between the CenceMe portal and ClickStatus usage.

Figure 2.13: CenceMe user study statistics.

the analysis of system logs on the backend is shown. It is evident that location information which reveals where friends are is the feature most used by the participants. The random photos was also found to be of interest because it can be used as a way to tag the person's day as in a diary: "oh yeah... that chair... I was in classroom 112 at 2PM". The photos are often blurred, since they are taken outside the control of the person, but they still serve the diary tagging purpose. Some of the participants did not particularly like the fact that the system takes pictures outside their control, so they opted to turn that feature off by customizing their privacy policy on the phone.

What is the potential CenceMe demographic? We believe that people-centric sensing applications such as CenceMe could become popular among social networking application users, for whom sharing context and information is popular. For many of these users, privacy is less of a concern than for others, as shown by their interest in publicly publishing personal history in detail in blogs and on social networks. This tendency is also highlighted in Figure 2.13(b) which shows a comparison between the cumulative number of random photos inserted into the database versus the total number of photos deleted for different hours of the day. Once photos are uploaded users are given the opportunity to selectively delete them from the system.

Few participants (4 out of 22) disabled the random photo for the entire duration of the experiment and others disabled it at different times of the day to meet their privacy needs or the needs of

the people around them. In general, as shown in Figure 2.13(b), the number of non-deleted photos available for sharing is much greater than the number of deleted photos. Most participants did not mind having pictures taken at any time of the day and in random settings and then being shared with all the other participants. Many of them were excited by the idea of guessing what their friends were doing through the hint provided by random photos. Moreover, no CenceMe presence sharing restriction was applied by the participants, who allowed their sensing presence to be accessible by everyone in the group. Although some users stated that they could foresee wanting to apply a presence sharing restriction policy under certain conditions (e.g., if their parents had access), they felt comfortable with the idea of others seeing their presence most of the time.

Learn about yourself and your friends. *"CenceMe made me realize I'm lazier than I thought and encouraged me to exercise a bit more"*. This quote is taken from one participant's survey. Other users expressed similar thoughts. Users view CenceMe as an application that potentially could tell them things that might be intuitively obvious, but are often invisible in their lives due to familiarity and repetition. Some examples are lack of physical activity and spending a lot of time in front of a computer. Near-real time presence sharing and historical presence representation are ways to capture peoples' lifestyle and trends about activity, social context (am I often alone? do I party too much?), and location.

My friends always with me. The study highlights that the participants enjoyed retrieving their friends' presence on the mobile phone with ClickStatus in addition to checking the portal. The average number of times per day they checked presence was 4 ± 3 times, where 3 is the standard deviation. Figure 2.13(c) shows a comparison between the total number of times presence is accessed through the portal or via ClickStatus distributed throughout the day. Although the number of times the participants access the portal is larger than their use of ClickStatus on the N95, ClickStatus is actively used. This is clear from Figure 2.13(c), where the use of ClickStatus rises during the time of day when people are presumably most likely on the move because they are going to class (between noon and 6PM) or hanging out with friends (between 8PM and 11PM).

Overall, the user experience is positive. Because many of them enjoyed using CenceMe, they kept the CenceMe phone for a while after the end of the experiment. We are currently working on revising some of the components and improving a few architectural elements in order to reflect some of the valuable feedback from the participants. Specifically, future revisions of the CenceMe system will include:

- An improved CenceMe software module on the phone that prolongs the battery life. Our goal is to achieve a 48 hour duration without recharging the device.
- An enhanced version of the portal to provide finer grained privacy policy settings as well as an enhanced ClickStatus user interface to provide the user with more powerful ways to browse their friend's presence.
- A shorter classification time for primitives and facts because many of the participants believe that real time access to buddies' sensing presence should be one of the features of the system.

System architectural revisions are currently under consideration to meet this requirement. A burst mode for sensing may prove to be useful.

2.8 CenceMe Large-Scale Deployment

The smartphone's programmability, pervasiveness and growing computational capability, along with the application distribution systems support (i.e., vendor specific app stores) are contributing to an explosion of smartphone-centered research across many area of interest to the UbiComp community: from gaming to social networking, green applications, and mobile sensing. The application distribution system support in particular (e.g., Apple App Store, Android Market, and Nokia Ovi) is a game changer for the research community because it enables the deployment of applications to millions of smartphones in the blink of an eye and gives the opportunity to collect very large data sets from the wild as never possible before. By mining rich, large-scale data sets researchers will be able to answer novel and exciting research questions discovering, for example, the way people use and interact with their mobile phones [105, 106], with location-based social networks [107] and green apps [42].

In this Section, we report on our experience from releasing CenceMe [14] to the public through the Apple App Store. CenceMe is a social sensing application for smartphone that was first implemented on the Nokia N95 devices in 2007 and evaluated as part of a small scale study [14]. Following this, CenceMe was ported to the iPhone and released publicly in July 2008 when the App Store was first launched. Since its release on the App Store CenceMe has been used by over 9100 active users in over 95 countries. The goals of the iPhone CenceMe release are the following: i) scale our project outside the confined settings of a research lab and give the application a "global" dimension; ii) understand how a mobile sensing application works in the wild without the direct control of the researchers; iii) assess the interest of people toward a mobile sensing application, which, by exploiting the phone's sensors to transparently infer human activity and the surroundings of the user, opens up new dimensions, in terms of content and privacy.

We describe the design of the iPhone CenceMe client and backend and the challenges encountered in building a mobile sensing application to be publicly used. We discuss the "*deploy-use-refine*" approach we adopt in order to evolve the system design as users' feedback is collected. We describe the software challenges and limitations that could impact the capabilities of a global mobile sensing application. When a mobile sensing application runs unmanned in the wild there is no guarantee that the event being inferred (having sensed it using the phone onboard sensors) is correct, since there is no notion of ground truth. We propose a mechanism that boosts the fidelity of mobile sensing applications by relying on a multi-sensing modality approach to mitigate the effect of lack of ground truth data.

It is important to point out that releasing an application to the public for research purposes requires to be compliant with the ethic code, privacy, and security constraints that protect the users of the application. It is also necessary to follow the research institution or university ethics recommendations

from the IRB (Institutional Review Board) regulating user data collection and treatment.

2.9 IPhone CenceMe

CenceMe is a social sensing application which integrates with popular social networking platforms such as Facebook, MySpace, and Twitter to augment a person's context status by using the phone's onboard sensors [14]. By running machine learning algorithms on the mobile phone itself over the sensor data collected by the phone's accelerometer, microphone, bluetooth/wifi/GPS, and camera and fusion techniques on backend machines, CenceMe infers a person's activity (e.g., sitting, walking, running) and social context (e.g., dancing, with friends, at a restaurant, listening to music) in an automated way. This information is shared within the person's social circle automatically giving the user the ability to customize their privacy settings to regulate what and where to publish the sensing-based presence. In what follows, we describe the architecture of the iPhone CenceMe client (in order to meet usability, classifiers resiliency, and preserve the phone user experience in terms of battery life for example) and backend (designed to be robust against failures, bursty user access, etc).

2.9.1 Client

The iPhone CenceMe client is implemented using a combination of Objective-C and legacy ANSI C code. Objective-C is mainly used for the user interface implementation, to access the low level sensors, the internal sqlite database, and to respect the model-view-controller principle of iPhone OS. C is adopted to implement the activity recognition classifier (which relies on a decision tree algorithm), and for the audio classifier (that determines the surrounding audio level - noise, quiet - or if a person is in a conversation). The audio classifier is a support vector machine (SVM) technique using the LIBSVM C library [108].

The client is responsible for: i) operating the person's presence inference over the sensor data by locally running the inference algorithms; ii) communicating the inference labels to the backend; iii) displaying the user's and their buddies sensing presence (activity, social context, location), the privacy configurations, and various other interface views that allow, for example, a user to post short messages on their preferred social network account. The classifiers are trained offline in a supervised manner, i.e., taking large collections of labeled data for both the audio and accelerometer modalities and using that data to train the classification models which are later deployed on the phone. Although earlier versions of the iPhone OS did not support multitasking (i.e., the capability to run applications in the background) the CenceMe client is designed to properly duty-cycle the sensing, inference routines, and communication rendezvous with the backend to limit the battery drain of the phone when the application is active. Reducing the battery drain is critical to avoid rapid battery discharges when the application is used, a condition that would negatively impact the phone user experience.

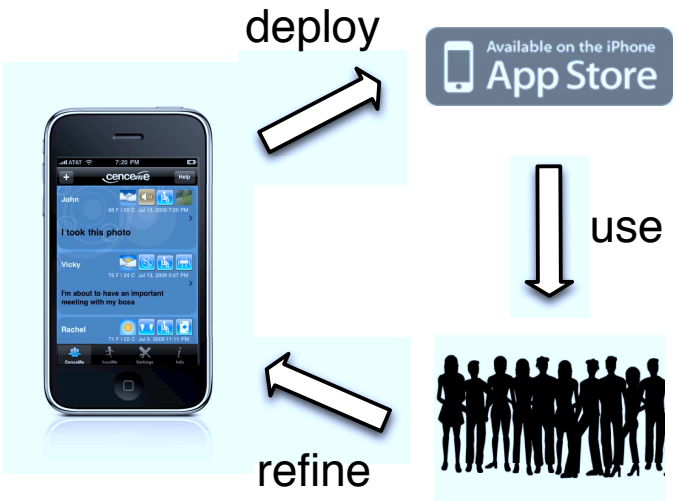


Figure 2.14: The "*deploy-use-refine*" model adopted for the CenceMe large-scale deployment.

2.9.2 Cloud

The iPhone CenceMe backend, which is implemented on the Amazon Web Service cloud infrastructure [109], is comprised by a series of different virtual machine images. Each machine is an Amazon EC2 virtual machine instance running Linux which provides a series of PHP and Python based REST web service allowing multiple machines to be composed together. Each image performs a different role in the backend infrastructure and has been designed to be initialized and composed together to offer different operating points of cost and performance. This allows us to temporarily initialize different numbers of machines of different types depending on the existing or expected user workload. It also allows us to manage the cost of running the CenceMe service so that we can provision additional machines (which incur additional costs) only when user demand requires it (for example, when a new model of the Apple iPhone is released and temporarily many users try out our application, which causes us to reconfigure our system).
The server side system is responsible for: i) storing user sensor presence information and allowing other CenceMe clients restricted access to this data; ii) publishing this sensor presence information to third party social network such as Twitter and Facebook; iii) maintaining the CenceMe social network friend link structure; iv) performing routine user registration and account maintenance tasks; and v) the collection of statistics about user behavior both on the client and backend side of the system.

2.10 Large Scale Deployment: Lessons Learnt

In this section, we discuss the experience we gained by deploying CenceMe on the App Store and having it used by thousand of users worldwide. Throughout the development and the evolution stages of iPhone CenceMe we applied a "*deploy-use-refine*" model (see Figure 2.14). According to

this strategy, after initial tests in the lab, the application is deployed on the App Store. Following this phase, users start downloading and using the application. Their feedback and user experience over time, submitted to us via a dedicated customer support email channel or the CenceMe discussion board [110], trigger the application fixing and refinement process, in order to meet users satisfaction and improve the application usability. In what follows, the lessons learnt from the iPhone CenceMe deployment are reported.

Information Disclosure. When leveraging an application distribution system such as the App Store to collect data to be used for research purposes, it is very important that the user downloading the application is fully informed about the nature of the application and the data being collected, as well as how the data is going to be handled. Full disclosure of such information is often required by universities IRB and disclaimers should be made clear in the application terms of service. Given the sensitive nature of the iPhone CenceMe data, i.e., inference labels from sensor data, our university IRB makes us add a different consent form following the terms of service page which explicitly mentions the purpose of the application and describes the nature of the data collected along with the use we are making of the data. According to IRB, this extra step is needed because people do not often read terms of service notes carefully, thus, a second dedicated disclosure form is required. Of course, by requiring a consent form through the involvement of IRB as often needed when carrying out research involving human subjects, the cycle of an application deployment becomes much longer. The IRB might take months to approve a certain research project, and even so several iterative steps may be needed in order to meet the IRB requirements. This implies long cycles before an application can be released. This extra time should be taken into consideration by researchers that want to carry out research at large scale through application distribution systems. The second implication of adding an explicit consent form in the application is that users might opt out from using the application (as we verified with some of the iPhone CenceMe users). This is because people are not yet used to downloading research applications from a commercial platform such as the App Store and they do not often understand the purpose of the research. As a consequence, the pool of users participating in the research might grow slowly.

Monetary and Time Costs. Moving research outside the lab for large scale deployments through the App Store has also monetary and time related costs. Bursty incoming user data, along with the necessity to rely on robust and reliable backend servers, most likely demand the support of cloud computing services [109]. In this way, the researchers' maintenance job is greatly alleviated since existing cloud services guarantee reliability and the ability to rapidly scale to more resources if needed. The flip side is that researchers have to be ready to sustain the subscription cost.

It is also to be taken into account the time overhead needed to adapt the application to new phone OS releases (which often carry API changes) in order to make the application able to transition through different versions of the software seamlessly. Without this support, users would not be guaranteed a smooth usage of the application which could potentially be dropped with severe impacts on the research outcome. Users might also ask questions and need to be guided through the use of the application. Researchers need to be ready to devote some of their time to customer service support.

A prompt response from an application developer gives strong feelings about the solidity of the application and the people supporting it.

Software Robustness. Software robustness and clean user interface (UI) design may be a foregone conclusion. However, the effects of poor software design (which implies little robustness of the application) and poor UI layouts should not be underestimated. People downloading an application from any distribution system expect the software to be robust, simple to use, with easy-to-navigate UI. If any of these requirements are not met, users might loose confidence in the application and not use it anymore. As researchers, we might not have the UI design skills often required to make an application attractive. It is then important to collect feedback from domain experts that can guide the researcher to a proper design of the UI. We learnt about this issue after a couple of iterations of the iPhone CenceMe client. We modified the UI design and the different navigation views by taking into account users feedback and our own experience in using the app.

By dealing with software that needs to run on mobile phones, researchers have to pay great attention to the impact the application might have on the phone performance itself. Phone manufactures often guarantee that the user experience with the phone is not degraded when third party apps are running. Hence, resources are reclaimed, namely RAM and CPU, when the phone OS assesses that there is a need for it. Researchers have to make sure that the application does not take too many CPU cycles and/or occupy too much RAM, otherwise the application might be shut down unexpectedly. This is a particularly important aspect to be considered for applications designed to run in the background. Researchers that want to deploy applications at large scale have to be ready to write code at near-production level, in order to maximize the application usability and robustness.

Although testing the application in the lab might not let you discover all the possible glitches in the code, extensive testing phases are required before submitting an application to the distribution system. This is important in order to minimize the likelihood that users will encounter problems with an application and to reduce the chances that an application is rejected during the screening process; for example in the case of the Apple App Store. It should be noted that Android Market does not screen applications making it more attractive in the case of some applications. One of the CenceMe releases did not pass the screening phase because of a debugging flag mistakenly left in the code causing the application to crash. As a result of this silly mistake the application was pushed to the back of the application screening process queue by Apple, delaying the new release of CenceMe by several weeks.

Hardware Incompatibilities. New phone models or the evolution of existing models could present hardware differences that could impact the application performance. We experienced this issue during the iPhone 2G to 3G generation transition phase, where the former mounts a different microphone than the latter. We started noticing a performance drop of the audio classifier when the same application was running on a 3G phone. The drop was caused by the fact the audio classifier for conversation detection was trained using audio samples mainly recorded with iPhones 2G. Since the frequency response of the iPhone 3G microphone is different from the 2G model, the classifier trained with 2G audio was not able to infer accurately 3G audio. For a large scale application de-

veloper it is then important to realize these differences in time to limit misbehaviors when people replace their devices.

User Incentives. In order for users to use a research application they have to have an incentive and enjoy it when the application is active on their phone. If there is no or little return to them, the application might be used rarely with scarce benefit to the research outcome. In order to engage users we include a feature in the iPhone CenceMe client named *IconMe*. IconMe allows a user to select an icon that better represents their status, mood, and surroundings and associate a 140 character message to it. Such a message is shared with the CenceMe friends and posted on the personal Facebook, MySpace, and Twitter account according to the user's preferences. We found this microblogging service an effective way to stimulate the use of iPhone CenceMe.

User Reviews. Users reviews through blog posts or on the application distribution review system itself can be brutal. And the impact of such reviews may be negative for the application reputation. In order to limit the likelihood people report something not pleasant about the application, which might discourage others from downloading it, the testing phase preceding the deployment has to be extensive as pointed out above. For example, at the very beginning of the App Store launch we thoroughly tested iPhone CenceMe on legacy iPhones, i.e., not jailbroken – which have modified software to enable deeper interaction with the platform than what possible with legacy phones. It never occurred to us that jailbroken phones might induce unexpected behaviors when running our application. Some App Store reviews from iPhone CenceMe users were reporting unexpected crashes when using the application. After some investigation, we realized that the issue was present only on jailbroken phones (the reviews were in fact from jailbroken phone users) and caused by a routine accessing the sqlite database. It is again very important to point out that across-platforms tests are needed in order to make sure the application is robust. This is a practice that is not usually required for limited lab setting experiments, but necessary for large scale deployments and is again a factor that impacts the duration of the research.

Software Limitation. Although sensors are being included in smartphones, the APIs to access them and their behavior is not entirely designed to support mobile sensing applications. An example is the accelerometer APIs. Phones operated by Android OS and the recent new iPhone OS4 support multitasking and give the possibility to run applications in the background. Activity monitoring applications require continuous streams of accelerometer data. However, since the accelerometer on modern smartphones is mainly intended to drive the user interface (e.g., move from portrait to landscape mode) the manufacturers opted not for delivering accelerometer data to the application anymore when the app is pushed to the background. Although this is a sound design choice from a UI standpoint because an application in the background does not need an active UI, it is not desirable for a continuous sensing application. Such a limitation has to be taken into consideration.

Lack of Ground Truth. As soon as a mobile sensing application is being used and inferred labels start converging to the backend, the question is: how reliable that label is? Is the event being inferred actually occurring? We do not have ground truth evidence when the application operates in the wild. One way to increase the trustworthiness of the data is to randomly prompt the user to

provide a label for the current inferred event. Even if this procedure is sparse in time, it might allow us to collect significant statistical evidence in the long run. The other option is to design and exploit novel techniques that mine peoples multimedia content, (e.g., microblog posts/tweets, video, photos, audio clips) as a means of seeking correlation between the semantic of keywords/characteristic features of the multimedia content and the actual activity. We are currently investigating both the possibilities.

2.11 Related Work

There is growing interest in the use of sensor-enabled mobile phones for people-centric sensing [7, 73, 9, 111, 4]. A number of diverse applications are emerging. In [42], the authors describe an application that determines pollution exposure indexes for people carrying mobile devices. A micro-blogging service is discussed in [112] that uses mobile devices to record multimedia content in-situ and shares this content in a real-time. In [51], we discuss the integration of the CenceMe application with Second Life [113]. The use of personal sensor streams in virtual worlds is a new and interesting area of research.

Cellphones have been used to learn about social connections [114, 3] and provide context-aware communications using location information from cellular towers and manually configured preferences in the iCAMS system [115]. The iCAMS system allows users to pick the preferred method of communication according to a person's status and location (e.g., in person, email, home/work phone). WatchMe [116] is a similar system that aims at choosing the best way to communicate with buddies. WatchMe relies on GPS trace analysis to determine whether a person is walking or driving, and uses the phone's microphone to infer talking and silent states. CenceMe differs from iCAMS and WatchMe because of the rich context it provides about a person in an automated and transparent way. In the same way CenceMe also differs from Twitter [117], an application to publish text-based status messages generated by users.

There is a large body of work on activity inference and modeling using customized sensors worn by people [118, 11, 76, 119, 19]. CenceMe differs from this work because it implements the activity inference algorithms on commercial mobile phones. As discussed in this Chapter there are a number of important design tradeoffs that need to be taken into account when implementing always-on people-centric sensing applications like CenceMe on off-the-shelf mobile phones. Systems such as SATIRE [22] also assume sensing devices with great capabilities being embedded into "smart clothing". An interactive dancing project [120] requires people to wear customized sensors mounted on shoes to track dancing activity. In [11] the authors discuss their experience building efficient classification techniques on the Intel Mobile Sensing Platform (MSP), a small form factor wearable device for embedded activity recognition. The MSP platform is quite powerful compared to many cellular devices. The CenceMe classifiers have been tailored to operate on less capable devices than the MSP while remaining effective.

Researchers have started leveraging commercial application distribution systems to carry out

global scale research [121, 122, 107, 105] while identifying challenges [123]. The authors of [107] analyze the public available data from the Brightkite mobile social network to study the behavior of users. Results from the study of a large deployment through the App Store of a game application are reported in [121], while [122] discusses the findings from a global scale study of a HCI technology deployed on the App Store. The authors of [106] and [105] present the results of medium and large scale experiments to investigate smartphone usage patterns, respectively.

2.12 Summary

In this Chapter we presented the implementation, evaluation, and user experiences of the CenceMe application, which represents one of the first applications to automatically retrieve and publish sensing presence to social networks using Nokia N95 mobile phones. We described a full system implementation of CenceMe with its performance evaluation. We discussed a number of important design decisions needed to resolve various limitations that are present when trying to deploy an always-on sensing application on a commercial mobile phone. We also presented the results from a long-lived experiment where CenceMe was used by 22 users for a three week period. We discussed the user study and lessons learnt from the deployment of the application and highlighted how we could improve the application moving forward. We discussed our experience from our large-scale CenceMe deployment through the Apple App Store. We gained a number of important insights using new application delivery channels presented by app stores and supporting a large number of users over a long period of time for what is essentially a research app and not a commercial app – a sort of Trojan horse of sorts, a research project masquerading as a phone application. Many important questions remain in this new era. How do we collect and validate our research data when we have limited control over users and lack real ground truth? How do we make sure we have a good cross section of users to validate our study? Will app stores continue to allow academic researchers to use their deliver channels to do research? If a research app becomes wildly popular how do small academic labs respond to that in terms of financing cloud infrastructure and supporting potentially 100s of thousands of users? It is clear that the new environment represents a fast way to try out new ideas in the market place driving more and more innovation. In essence, the app stores are the digital equivalent of the petri dish – we can germinate new ideas and watch them grow or fade in the real world, with real users, distributed across the world. This is a very exciting departure from how we did research before app stores.

Many challenges were discovered during the CenceMe deployments. They range from the need to design more scalable classifiers that efficiently operate in the wild, to the importance of addressing inference robustness against sensing context and mobility, and the necessity to find ways to validate inferred labels collected from the wild. In the next chapter we discuss a distributed and collaborative inference framework to improve the robustness of mobile sensing applications. In Chapter 4 we present a large-scale mobile sensing application for people and places characterization, where we show preliminary results for a technique to validate inference labels collected from the wild.

Chapter 3

A Distributed and Collaborative Inference Framework for Smartphone Sensing Support

3.1 Introduction

The continuing need to communicate has always pushed people to invent better and more efficient ways to convey messages, propagate ideas, and share personal information with friends and family. Social-networking, for example, is the fastest growing phenomenon of the Internet era where people communicate and share content with friends, family, and acquaintances. Recently, researchers started investigating new ways to augment existing channels of communication and improve information exchange between individuals using the computational and sensing resources offered by sensor-enabled mobile phones (aka smartphones). These phones already utilize sensor data to filter relevant information (e.g., location-based services) or provide better user experiences (e.g., using accelerometer data to drive smartphone sensing applications). However, information about user's behavior (e.g., having a conversation) and personal context (e.g., hanging out with friends) is often provided manually by the user. This naturally leads to the following thoughts: what if the available sensors are further exploited to automatically infer various aspects of a person's life in ways that have not been done before? What if the characterization of the person's *microcosmos* could be seen as a new form of communication? We believe that as sensor-enabled mobile phones become commonplace, they can be used at a personal-scale to enrich and support communication and collaboration, to measure and improve task performance, and to aide in the assessment of health and wellness.

There is a growing research effort in using mobile phones to infer information about people's behavior and their environment [71, 111, 124, 48, 14, 41, 40]. These systems typically rely on pre-trained models or classifiers, where the training data from events of interest are acquired in advance. It is often exceedingly hard to obtain a representative training set from which to build reliable classifiers (e.g., samples of an individual's voice in all possible environments). As a result classifiers tend to perform poorly. Furthermore, current approaches do not take advantage of increased sensing density offered, for example, by the cloud of mobile phones around us. This cloud represents an

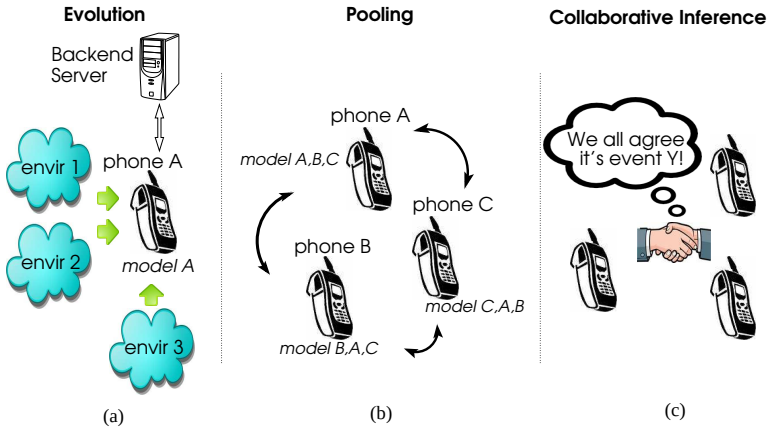


Figure 3.1: Darwin's steps: (a) evolution, (b) pooling and (c) collaborative inference. They represent Darwin's novel evolve-pool-collaborate model implemented on mobile phones.

ensemble of in situ resources that can cooperate to boost sensing performance, make sensing more robust, and achieve a common sensing task in a more efficient and scalable way.

With the rising interest in smartphone sensing applications we believe there is a need to provide mechanisms that maximize the robustness, accuracy, and scalability of these applications. In this Chapter, we present Darwin, a novel collaborative reasoning system that is self-extending and utilizes co-located mobile devices to achieve better accuracy and scalability, at lower cost to the user. As shown in Figure 3.1, Darwin combines three different computational steps to achieve its goal:

Classifier evolution is an automated approach to updating models over time such that the classifiers are robust to the variability in sensing conditions common to mobile phones (e.g., phone in the pocket, in pocket bag, out of the pocket), and settings (e.g., noisy and loud environments). A fully supervised learning method, where labeled examples from different context are provided to the system, would not be practical in this case since the phone owner would continually have to provide labels any time an event is detected that is determined to be in a different setting and context. This simply does not scale and would be unacceptable to users. While self-evolving classification models techniques have been investigated in the past [125], we show the actual deployment of such techniques on a real phone based system.

Model pooling is a novel technique which is designed to answer the following question: can we reuse models that have already been built and possibly evolved on other phones? This would increase the system scalability because there would be no need to retrain classifiers for events which already have classifiers trained to recognize them. With pooling, mobile phones exchange classification models whenever the model is available from another phone, thus, allowing mobile phones to quickly expand their classification capabilities; that is, if a given mobile phone does not have a model for a person or an event, there is no need for it to create a new classifier as it can readily obtain and use another phone's model. Note, that models can be exchanged in-situ between co-located phones or from servers over the network. In either case the basic pooling process remains the same.

Collaborative inference combines the classification results from multiple phones to achieve

better, more robust inference with higher confidence in the sensing result. After pooling, co-located mobile phones all have the same classifiers. At this point they can run the same inference algorithms in parallel and a final inference result can be obtained by combining the output from the different phones. This allows the system to be more robust to degradation in sensing quality experienced by some of the phones (e.g., a person carrying the phone in the pocket) and take advantage of improved sensing quality offered by others (e.g., the phone is out of the pocket near the event to be sensed).

We show the performance of Darwin by exploiting the audio modality of mobile phones, in particular, we show the benefit of applying Darwin to a speaker recognition application using audio sampled by the onboard microphone. We show the performance of the speaker recognition algorithm on the Nokia N97 [126] and Apple iPhone [92] in different settings and context when Darwin and the speaker recognition application are used by eight people.

The reason we select speaker recognition is not because we intend to design a new speaker recognition algorithm (there is a considerable amount of literature on this topic [127, 128, 129, 130, 131]), but to show how Darwin improves a mobile sensing application inference quality.

Darwin is founded on an opportunistic sensing paradigm [4], where the user is not an active participant in the sensing activity (i.e., actively taking a sensor reading). In this case, sensing happens automatically and continuously when the system determines that the sensing context is right for sensing. Darwin can be thought of as a sensing system running in the "background mode" of the mobile phone without any user intervention in actual sensing. The key contribution of our work is to show how Darwin can boost the inference accuracy of mobile sensing systems by applying *distributed computing and collaborative inference concepts* to these systems when devices come together opportunistically. We conjecture that Darwin applied to other smartphone sensing applications and systems that use the microphone as an audio sensor [40, 41] would also see similar performance improvements because audio sensed data is sensitive to the characteristics of the environment (e.g., noise, other people speaking, etc.) and sensor context of the phone (e.g., in or out of pocket for example). At the end of the Chapter, we also show how Darwin could be integrated with a mobile social networking application, a place discovery application, and a friend tagging application.

Today, smartphone sensing applications mostly exploit the GPS, accelerometer, digital compass, and microphone sensors for personal sensing. In the future, smartphone sensing will be societal-scale supporting a broad set of social, health and environmental applications, such as, tracking pollution, population well-being, or the spread of disease. It is also likely that more sophisticated sensors will be embedded in phones, such as, pollution and air quality sensors [35] and galvanic skin response (GSR) sensors. The sensing and inference quality of these applications is affected by many factors. Importantly, *phone sensing context*, i.e., the position of the phone on a person's body in relation to the sensed event, is challenging for these emerging applications. A phone in the pocket or bag might perform poorly when sensing air quality or audio events. Classification models are also limited by the quality of the trained data and their inability to capture different characteristics from the data in different environments. Darwin's novel **evolve-pool-collaborate** model is designed to

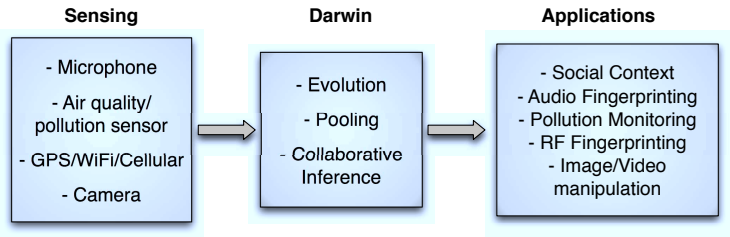


Figure 3.2: Examples of application domains Darwin can be applied to: social context (e.g., in conversation, in a meeting) and ambient audio fingerprinting using the microphone; pollution monitoring leveraging the phone's pollution sensor; radio fingerprinting for localization with GPS, WiFi and cellular triangulation; and applications exploiting the phone's camera.

provide a foundation for a broad family of existing and emerging sensors and applications, as shown in Figure 3.2. To the best of our knowledge, Darwin represents the first system implemented on a number of mobile phones that can evolve, pool, and enable cooperation providing robust, efficient, and scalable sensing.

The structure of this Chapter is as follows. Section 3.2 presents the detailed design of the Darwin system, followed in Section 3.3, by a discussion of a technique introduced to infer the phone sensing context. In Section 3.4 we discuss privacy and trust issues. Section 3.5 presents the system implementation and performance evaluation of Darwin applied to a proof-of-concept speaker recognition application. Following this, we discuss a number of other sensing applications built on Darwin and then discuss the related work in Section 3.6 and Section 3.7, respectively. Section 3.8 concludes with the chapter summary.

3.2 Darwin Design

In this section, we present the detailed design of the Darwin system including the use case speaker recognition application.

3.2.1 Design Considerations

The design of Darwin is governed by the limited computational resources on the phone to run computationally intensive machine learning algorithms and by mobility issues. In what follows, we discuss these motivating factors and the design decisions that address them.

The main goal of mobile phones is expanding way beyond just making phone calls. Compared to early mobile phones, modern smartphones are also powerful programmable platforms with, at this writing, up to 1GHz processors and 1GB of application memory [132]. While smartphones have increasing resources, running continuous sensing applications presents a number of important challenges. These range from the design of efficient duty-cycling algorithms that can maintain acceptable fidelity and time between charges, to the need to push more intelligence to the phone in terms of classification algorithms without impacting the user experience (e.g., freezing the phone,

slowing the UI, blocking calls). Machine learning algorithms that run on the phone to process sensor data should be implemented in an efficient and lightweight manner. Darwin is designed to reduce on-the-phone computation based on a split-level computation design [14], offloading some of the work to backend servers (as discussed in Section 3.2) while trading off the cost for local computation and wireless communication with backend servers.

Users carrying mobile phones also presents a number of challenges for continuous sensing applications that have to operate under real-world mobility conditions. The context of the phone is challenging to sensing. Users carry phones in many different ways. Therefore, when a phone senses an event, its context (e.g., in/out of the pocket, in/out the bag, etc.) will impact the sensing and inference capability of the phone. Another challenge that mobility creates is that the same phone may sense the same type of event under different conditions (e.g., the same person speaking in a quiet office or noisy restaurant). This leads to poor inference. A group of co-located phones running the same classification algorithm and sensing the same event in time and space could compute different inference results because of the context problem and slight environmental differences, as discussed above. In essence, each phone has a different viewpoint of the same event. These real-world issues arise because sensing takes place out in the wild – not in a controlled laboratory setting – and is governed by the uncontrolled mobility of users.

Darwin exploits mobility and addresses these challenges. It uses classifier evolution to make sure the classifier of an event on a phone is robust across different environments – works indoors and outdoors for example. Extracting features from sensed events in order to train a classifier is costly for a mobile phone in terms of computation and time. Darwin allow phones to pool classification models when co-located or from backend servers. Pooling radically reduces the classification latency because a phone can immediately start to make inferences without the need to train classifiers. Different phones running the same classifier and sensing the same event are likely sensing the event differently, as discussed above. Darwin uses collaborative inference to compensate for this difference, boosting the final inference result. Darwin exploits mobility because it is designed to be opportunistic in its use of classifier evolution, pooling, and collaborative inference.

3.2.2 Darwin Operations

In what follows, we present a high level description of how Darwin operates: **(1)** each mobile phone builds a model[1] of the event to be sensed through a seeding phase. Over time, the original model is used to recruit new data and evolve the original model (see Figure 3.1(a)). The intuition behind this step is that, by incrementally recruiting new samples, the model will gather data in different environments and be more robust to environmental variations. The phone computes the feature vector[2] locally on the phone itself and sends the features to a backend server for training. This is

[1] A classification model is represented by a mathematical expression with parameters. For example, in the case of a Gaussian classification model, the model is identified by a Gaussian function with mean and standard deviation as the parameters. Refer to Section 3.2.5 for further details.

[2] A feature vector is a vector of numerical elements representing an event. For example, in the case of activity recognition applications that use the accelerometer, two feature vector elements often used are the mean and standard deviation

because the feature vector computation is quicker and more energy efficient than the training phase of a machine learning algorithm such as a Gaussian Mixture Model (GMM), which is the technique we use (it takes approximately 2 hours to train a GMM with 15 seconds of audio from experiments on the Apple iPhone and N97). **(2)** When multiple mobile phones are co-located they exchange their models so that each phone has its own model as well as the co-located phones' models. Model pooling, as shown in Figure 3.1(b), allows phones to share their knowledge to perform a larger classification task (i.e., in the case of speaker recognition, going from recognizing the owner of the phone to recognizing all the people around in conversation). After models are pooled from neighboring mobile phones, each phone runs the classification algorithm independently. However, each phone might have a different view of the same event – i.e., different phone sensing context. For example, one of the phones might be inside the user's pocket whereas another one might be outside, or one of the phones could be closer to the sensing event than others. **(3)** Collaborative inference exploits this diversity of different phone sensing context viewpoints to increase the overall fidelity of classification accuracy, as illustrated in Figure 3.1(c).

3.2.3 Speaker Recognition Use Case

We choose speaker recognition as our proof-of-concept application because the audio modality is generally sensitive to environment and phone sensing context and we believe the findings from this application will generalize to other classification problems such as in [40, 41] or pollution for example [35] for which the phone sensing context is important. The speaker recognition application attempts to determine the identity of a speaker by analyzing the audio stream coming from a microphone. The recognition process includes the following steps:

Silence Suppression and Voicing. The system first eliminates audio chunks that contain silence or low amplitude audio and then runs a voicing routine to remove the chunks that do not contain human voice. By focusing only on chunks that are likely to contain human speech we reduce the false-alarm rate of the classification system. The silence suppression filter works on 32 ms of audio at a time and discards portion of the audio whose root mean square (RMS) value falls below a threshold $\mathscr{T}$. The threshold $\mathscr{T}$ is determined experimentally under various conditions, for example, recording voice using the mobile phone in quiet indoor environments, on a sidewalk of a busy road, and in a restaurant. The voicing is performed by training a GMM using several hours of non-voice audio captured in various conditions (e.g., quiet environments, noisy from car traffic, etc.) and discarding the audio chunks whose likelihood falls with a +/- 5% from the non-voice likelihood. This threshold is also determined experimentally and found to be accurate for many different scenarios. More advanced techniques could be considered in future work such as the voicing scheme proposed in [133].

Feature Vector. The feature vector consists of Mel Frequency Cepstral Coefficients (MFCCs) which are proven to be effective in speech audio processing [134, 135, 136]. We use coefficients 2 to 20 and skip the first coefficient because it models the DC component of the audio.

of the accelerometer readings.

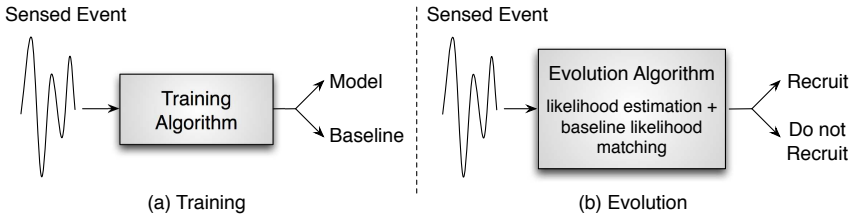


Figure 3.3: Darwin's (a) Training and (b) Evolution steps.

Speaker Model Computation. Each speaker is modeled with a mixture of 20 gaussians (hence, a 20-component GMM). The reason we use GMM is because GMM algorithms are widely used in the speaker recognition literature [137, 138]. An initial speaker model is built by collecting a short training sample – 15 seconds in our current implementation. When a user installs the application on their phone they are asked to provide a sample of voice by reading loud text displayed on the phone screen. In Darwin, the initial model evolves to capture the characteristics of the different environments where the person happens to be located.

Each speaker i's model M_i corresponds to a GMM$_i$, $\forall\ i$, $i = 1..N$ (where N is the number of speakers), and GMM$_i$ is the model trained and evolved by phone P_i for speaker i[3]. A GMM$_i$ is characterized by the tuple $M_i = <\boldsymbol{\mu_i}, \boldsymbol{\Sigma_i}, \mathbf{w_i}>$, where $\boldsymbol{\mu_i}$ is the $\mathscr{P} \times \mathscr{Q}$ multi-dimensional array containing the mean values of the gaussian distribution, $\mathscr{P}$ is the number of components and $\mathscr{Q}$ is the number of dimensions of the GMM model. $\boldsymbol{\Sigma_i}$ is the $\mathscr{Q} \times \mathscr{Q} \times \mathscr{P}$ multi-dimensional covariance matrix, whereas $\mathbf{w}_i$ is the $1 \times \mathscr{P}$ array of weights for each gaussian component. For a detailed discussion of a GMM for speaker identification see [137]. The Darwin implementation uses $\mathscr{Q} = 19$ (i.e., the number of MFCCs coefficient employed) and $\mathscr{P} = 20$. We fix the number of components to 20 because the larger the number of components, where a component represents a single gaussian within the mixture, the more accurate the model is. At the same time as the number of components increases, the computing cost for the model increases. For our implementation, we experimentally verify that 20 components provide the best tradeoff between computation cost and classification accuracy.

Speaker Inference. In Darwin, speaker inference operates in a collaborative and distributed manner. The inference is collaborative because all the mobile phones contribute to the final results by communicating the confidence level associated with certain audio with other mobile phones in the neighborhood using the short-range radio, such as, Bluetooth or WiFi. The inference is distributed in the sense that the final inference result is derived locally on each individual mobile phone without relying on any particular external entity. We show how collaborative inference boosts the performance of the speaker recognition application.

3.2.4 Classifier Evolution

Initial Training

The initial training phase is intended to be short so that the application can be used immediately by people without requiring a prolonged initialization phase. Clearly, a short training data set implies a less accurate classification model. For this reason off-the-shelf speaker recognition applications [139, 130, 140] use large number of samples typically several tens of seconds of speakers' voices in order to achieve acceptable recognition accuracy. In our system, the initial speaker model is just the starting point and the model is used to recruit new training data and evolve the model on-the-go without additional user intervention. For applications other than speaker recognition, the initial model of the event to be sensed is provided by the system during the deployment phase. The evolution algorithm is designed to be applied to different sensing modalities other than audio, i.e., air quality sensing, etc. The initial training phase consists of taking a sample or seed of the sensed data and using half of the data to build the model. The remaining half for building a baseline likelihood (BL) as shown in Figure 3.3(a).

During the initial training phase, a person is asked to talk into her phone until a voicing audio stream of 30 seconds is collected by the mobile phone. The first 15 seconds of the training set are used for training purposes and the remaining 15 seconds to set a baseline for the classifier evolution technique, as discussed in Section 3.5. The raw audio stream received from the microphone is first passed through the silence suppression and voicing filter; then, the training and baseline audio are both fed into the MFCCs extractor. The MFCCs^{train} computed from the training audio form the feature vector that is used to build a GMM of the speaker. The baseline audio is used to extract MFCCs^{base} and to determine the BL that is used to recruit new audio samples, as discussed in Section 3.2.4. The baseline is computed only for the model that the phone trains, which, in the case of speaker recognition, is the model of the phone owner voice.

Evolution

An important challenge in training classifiers is to obtain sufficient amount of labeled data for supervised training. Data labeling is tedious and expensive and real-time labeling is seldom practical since people are averse to being interrupted in order to label data. In essence, labeling data does not scale.

Therefore, machine learning algorithms running on mobile phones can not fully rely on supervised learning strategies. In contrast, a fully unsupervised learning process without any human intervention can often latch onto idiosyncrasies of the data and is not guaranteed to model the classes the user might care about. Darwin takes a different approach by using a classifier evolution algorithm on mobile phones which uses a simple semi-supervised learning strategy [125]. We show that such an approach can be effective in boosting the performance of the speaker recognition

[3]We associate one speaker to a single phone. Whenever the speaker is using n phones we would have n different GMM models, one per phone for the same speaker.

system. Darwin's classifier evolution revolves around the fact that mobile phones create different classification models for different environments if existing classification models do not fit those environments. When the speaker recognition application bootstraps, the speaker's voice is sampled in a quiet environment to build the initial model for the owner of the phone (15 seconds of voicing audio). During runtime, if the likelihood of the incoming audio stream is much lower than any of the baseline likelihoods corresponding to the different models on the phone, then a new classification model is evolved. The classifier evolution algorithm comprises the following steps: (i) obtain the likelihood for a 96 ms incoming audio chunk; (ii) compare the maximum likelihood estimator (MLE) result with BL for models already existing on the phone; (iii) recruit the new audio as part of the training data, if necessary; (iv) re-train the classification model; and finally (v) compute the new BL, which is the likelihood of the original baseline plus the newly recruited data. The classifier evolution algorithm is detailed in Algorithm 1.

Comparison with BL. If L is the number of classification models for different environments, the $Likelihood_{new}$ is compared with BL_i, $\forall\ i$, i=1..L. At the beginning, the phone contains only BL_1, which is derived from the initial training data set. If $Likelihood_{new}$ falls between a +/- 3% interval around one of the BL_i, then that BL_i is used as the reference BL for the environment where the speaker is situated. We determine the 3% threshold experimentally in different settings and this value is the one that best generalizes to different environments and with the best performance. Because the threshold is derived for a set of environments where people spend most of their time (e.g, quiet indoors, outdoor along a sidewalk of a busy street, and in a noisy restaurant) we are confident that this threshold would extend to other similar environments.

After having determined which model to use, the absolute value of the difference between $Likelihood_{new}$ and BL_i is recorded. The vector **vecDiff** holds these differences for each new audio chunk. The mean and standard deviation of the elements in **vecDiff** are then calculated.

Data Recruiting and Model Re-Training. New data is recruited for re-training if the following two conditions hold: *i*) the standard deviation of the elements in **vecDiff**, when the new difference is added, oscillates outside a +/-5% boundary around the standard deviation calculated before the new difference is added to **vecDiff**; and *ii)* the $Likelihood_{new}$ is greater or equal than (BL_i - mean{**vecDiff**}). The first condition ensures we are recruiting only voice data that differs from the current model and the second normalizes the likelihood with respect to the mean. As new data is added to the training set, the inference model i is re-computed and a new BL_i is calculated by adding the audio chunk to the *baseline audio* of model i. The recruiting stops when the likelihood stabilizes inside the +/- 5% boundary because convergence is reached.

As a product of the classifier evolution algorithm, different classification models are automatically created for different environments and locally stored on the phone. We prove the effectiveness of our algorithm by running the speaker recognition application and Darwin system in three different environments, that reflect common places people find themselves. In our experiments, three models are automatically created. In our speaker recognition application each model uses 63KB of storage space and approximately 500KB of storage for the training data for each model, which

Algorithm 1 Pseudocode for the Darwin classifier evolution algorithm running on node *i*.

NumEnvType ← number of environments for which a model is defined.
while there is a new incoming audio chunk **do**
  Compute $Likelihood_{new}$
  Compare $Likelihood_{new}$ with BL_i, $\forall$ i, i=1..L
  if $\exists$ $j \in \{1..\text{NumEnvType}\}$ s.t. $Likelihood_{new}$ is within +/-3% BL_j **then**
    {The phone has a baseline for the environment}
    $BL_{comp} \leftarrow BL_j$
    $environmentType \leftarrow j$
  else
    {The phone does not have a baseline for the environment}
    $BL_{comp} \leftarrow BL_1$
    $environmentType \leftarrow 1$
  end if
  LikelihoodDifference ← | $Likelihood_{new}$ - BL_{comp} |
  Add LikelihoodDifference to **vecDiff**
  meanDiff ← mean(**vecDiff**)
  stdDevDiff ← stddev(**vecDiff**)
  if (($stdDevDiff \leq (previousStdDev - 5\%)$ OR ($stdDevDiff \geq (previousStdDev + 5\%)$)) AND ($Likelihood_{new} \geq BL_{comp}$ - mean(**vecDiff**)) **then**
    Recruit new training data for *environmentType*
    Calculate new model for *environmentType*
    Add data to *baseline audio* of *environmentType* model
    Calculate new BL_j
  end if
end while

includes the initial training set plus the fraction of data recruited as the model evolves. The overall storage requirement for a single speaker with three different environments is ∼1.8MB. This amount of storage is reasonable for modern mobile phones because they have several gigabytes of persistent memory and up to 1GB or more of application memory. Under more limited memory constraints the number of classification models for the same speaker and for the other speakers could be set to a maximum and the models arranged according to a circular buffer policy. As the number of models in the buffer exceeds the buffer capacity, the oldest models could be removed to accommodate new models.

3.2.5 Model Pooling

Model pooling is based on the simple premise of sharing classification models that have already been built and optimized by other mobile phones. It is a simple but effective way to increase classification timeliness by minimizing the inference latency. Model pooling boosts classification scalability,

accuracy, and speed by providing a mechanism for fast exploitation of existing models by mobile phones rather than building classifiers from scratch themselves. Model pooling boosts classifiers scalability because models are not required to be trained for multiple events, but just for those events the mobile phone is most likely to be exposed to. For example, in the speaker recognition case the percentage of time that the phone owner's voice is captured, because of conversations with nearby people or phone calls, is greater than the percentage of time any other speaker is captured. In this case, it would be possible to accurately re-train and refine over time, using the evolution algorithm, only the phone owner's voice model rather than everybody else's voice model. Mobile phones can pool models – voice models in the case of speaker recognition – from other co-located phones for events that they do not have a model for. These models are readily available, usable, and require no extra training steps. Model pooling does not necessarily occur between co-located phones. Models can be pooled from backend servers too. Assume a particular phone builds a sophisticated audio inference or pollution model of an area [141, 41, 35]. Phones can geo-tag models and upload them to backend servers. From this point on other phones moving into these geo-tagged areas do not have to wait to generate their own models. They simply download models from a server if available and start making inferences.

In order to formalize model pooling, let P be a mobile phone, M_p be the model derived and optimized for a certain event by P, and M_i be the classification models individually derived and optimized by N other mobile phones P_i where $i = 1 .. N$. If K of these mobile phones are co-located with P, following the pooling phase P would have its own model M_p and models M_i, where $i \in \{1 .. K\}$. Each model M_i is stored locally by P in order to be used again without the need to pool them by phone P_i, unless the model has been evolved by P_i. In that case, node P_i would announce to its neighbors that a new model has been computed and is ready to be pooled. As a result of the model pooling algorithm, all phones receive the classification models of all the other phones at the end of the model pooling phase.

After node P has pooled a classification model $M_i = <\boldsymbol{\mu_i}, \boldsymbol{\Sigma_i}, \mathbf{w_i}>$ from node P_i, node P will replace P_i's classification model M_i only if P_i announces a new classification model M_i^{new}. Node P_i determines that it is time to announce the new classification model M_i^{new} when the evolution of the model is complete.

3.2.6 Collaborative Inference

Collaborative inference is designed to boost the classification performance by taking into account the classification results of multiple phones "observing" the same event instead of relying on only individual classification results. The idea behind collaborative inference is as follows: given that an event could be sensed by multiple, distributed but co-located mobile phones, Darwin leverages the classification results of individual phones to achieve better accuracy. The hypothesis is that we can take advantage of the multiple sensing resources around an event and *exploit their spatial distribution and context diversity*. In the case of speaker recognition, when a person is talking with a group of people, the phones of the other people would pick up the speaker's voice with different

characteristics. Some phones could be in pockets, others on a clip belt or closer to a source of event than the other phones. In this case, if the application relies only on each individual phone's classification, the speaker classification accuracy would be poor when the phone sensing context is not good (e.g., in pocket or near a noise source) in relation to the specific event. The same reasoning holds for pollution measurements using mobile phones.

Following the pooling phase, each phone P_i contains the other phones classification models M_i, where $i \in \{1 .. K\}$, and K is the number of neighbors. Each phone P_i runs the inference algorithm in parallel using the common set of models. The fact that the event is classified by multiple phones using common classification models and that these phones may be exposed to different contexts (some of which are better than others) can lead to higher accuracy when the results are combined. The collaborative inference phase breaks down into three distinct steps: *i)* local inference operated by each individual phone; *ii)* propagation of the result of the local inference to the neighboring phones; and *iii)* final inference based on the neighboring mobile phones local inference results.

Local Inference

During the local inference phase each node individually operates inference on the sensed event using its own inference model of the event and the inference models pooled from other phones. The goal is to locally derive the confidence level (or likelihood) of the inferred event in order to be able to communicate this confidence level to neighboring phones. By having the confidence level of each phone sampling the event, all the phones are able to run the collaborative inference step to compute the final inference result. In order to operate collaborative inference, the phones must be time-synchronized because they need to perform inference on the same event at the same time. We rely on the fact that mobile phones support time synchronization through the cellular infrastructure. We measure a time synchronization error between four iPhones synchronized through the cellular network of 500 ms. If the error is larger than 500 ms we use a loose synchronization approach. One of the phones (randomly picked) sends a broadcast message which, when received by all the neighbors at the same time, triggers the sampling phase. After having received this broadcast message phones are synchronized.

Following the audio signal sampling and filtering, the stream is divided in 96 ms long chunks[4]. MFCCs are extracted from each chunk. At this point, a maximum likelihood estimation algorithm is run in order to verify which of the models best fits the audio chunk. To avoid having the maximum likelihood estimator running through too many pooled models, which could potentially be costly in terms of computation for a mobile phone, the estimator only uses the models of the phones detected in the neighborhood. Neighbor detection is performed using short-range radio technology, such as, Bluetooth and WiFi.

In order to increase the classification accuracy we feed the maximum likelihood estimation result into the *windowing* block. After the windowing step the local classification result is derived.

[4]We use 96 ms for each audio chunks to make it a multiple of the MFCC binning size of 32 ms. This multiple makes the duration of the audio chunk small enough to maximize the likelihood of capturing just one speaker at a time.

If the maximum likelihood estimator returns that the $i-$th audio chunk belongs to event E, the event E is deemed to be true if and only if the following two conditions hold: (i) event E, i.e., a certain speaker speaking, is detected at least once in a window comprising the previous two audio chunks; and (ii) the classification accuracy confidence is at least 50% larger than the confidence for any other events. The reason for the first condition is to guarantee that a voice sample is included – since audio chunks are 96 ms long, if a speaker is detected at least twice in a window of time of duration 96 ms $\times$ 3, then we have better confidence that we are really capturing that speaker. A larger window size would increase the classification latency and after experimentation we determine that a window size of 3 best suits our system, which requires near-real time classification results. The reason for the second condition, having determined the 50% threshold experimentally, is to dampen the impact of false positives. The pseudo-code of the local inference algorithms is shown in Algorithm 3. The locally inferred speaker ID is associated with the classification model that best fits the audio chunk following the windowing policy. Because a phone computes the inference confidence for each of the K models as reported in Algorithm 3, the result of the local inference takes the form of the following vector $\mathbf{LI}^{j}_{1..s} = \{confidenceSpeaker_1, confidenceSpeaker_2, .. , confidenceSpeaker_k\}$, where the subscript notation $1..s$ indicates that the vector contains the inference results for speakers 1 to s and the superscript index j indicates that the vector is generated by node j. Consider for example, that three nodes (N_1, N_2, and N_3) are co-located and running the speaker recognition application. If S_1, S_2, and S_3 are the speakers associated with nodes N_1, N_2, and N_3 respectively, and assuming S_1 is actually speaking, then an output for the **LI** vectors could, for example, be: $\mathbf{LI}^{1}_{1,2,3} = \{0.65, 0.15, 0.2\}$, $\mathbf{LI}^{2}_{1,2,3} = \{0.4, 0.5, 0.1\}$, and $\mathbf{LI}^{3}_{1,2,3} = \{0.7, 0.2, 0.1\}$. The reason of N_2 expressing lower confidence about speaker 1 could be caused by the fact that mobile phone N_2 may not have a good sensing context.

Local Inference Propagation

Following the local inference step, each mobile phone i broadcasts the result of its local inference (i.e., the vector $\mathbf{LI}^{i}$) in order to give neighboring phones the ability to move to the final step, i.e., the final inference. A time stamp is associated with vectors $\mathbf{LI}^{i}$ in order to align local inference results in time. A weight is also associated with vectors $\mathbf{LI}^{i}$. The weight is an indication of the quality of the local inference. For example, assuming the mobile phone determines its context (e.g., in or out of the pocket) and is running an audio or pollution sensing application [35], then the value of the weight is small or large when the phone is in or out of the pocket, respectively (for a discussion of how to determine the sensing context see Section 3.3). Therefore, a $\mathbf{LI}^{i}$ vector with a small weight indicates that the mobile phone i is not in a suitable sensing context and has less influence on the final inference phase.

The phones broadcast their local inference results at the rate the local inference is computed. The local inference is computed right after an audio sample is collected. Consequently, if the phone polls the microphone every A number of seconds, the local inference is computed and broadcasted every A seconds as well. For example, a local inference broadcast rate of 1Hz implies receiving the

Algorithm 2 Pseudocode for the Darwin local inference algorithm running on node i using K pooled models. The number K is determined by the number of detected neighbors. The locally inferred speaker ID corresponds to the index of the model best fitted by the audio chunk following the windowing policy.

while there is a new incoming audio chunk **do**

{Go through all the classification models:}

for j = 1 to K **do**
arrayLE[j] ← likelihood estimation for audio chunk
end for

Retrieve the max likelihood from arrayLE and the associated index indexMA.
Retrieve the second larger confidence from arrayLE and the associated indexSMA.

{Windowing policy:}

if $(arrayLE[indexMA] - arrayLE[indexSMA] \geq 0.5)$ AND $((indexMA == previousMAX)$ OR $(indexMA == previousPreviousMAX))$ **then**
locallyInferredSpeaker ← indexMA
previousPreviousMAX ← previousMAX
previousMAX ← indexMA
end if

end while

local inference results, hence, computing the final inference, every second. Clearly, the higher the broadcast rate, the faster the final inference (see Section 3.2.6). We do not propose any specific local inference broadcast rate, since this depends on the application requirements. Rather, we show the cost of transmitting a single confidence level from which the total cost for each specific transmission rate is derived (see Section 3.5.4).

Final Inference

The goal of the final inference phase, which follows the local inference propagation, is to compensate event misclassification errors of each individual mobile phone achieving better classification confidence and accuracy of sensed events. Mobility and context affect the local inference result, sometimes such that an event could be misclassified. For the final inference, Darwin combines the local inference results derived by individual mobile phones that are spatially distributed in the area of the target event. The technique used to perform the final inference is to find the inferred speaker (IF) that maximizes the joint probability as in Equation 3.1.

$$IF = \underset{\mathbf{j}, j \in \{1..S\}}{\arg\max} \{Prob(s_j^1), Prob(s_j^2), .., Prob(s_j^K)\} \tag{3.1}$$

where K is the number of co-located phones that have been contributing with their local inference, S is the number of speakers, and $Prob(s_j^i)$ is the probability that speaker j is detected by phone i. This operation is facilitated by the fact that the property of independence between the

different $\mathbf{LI}^{j}_{1..S}$ vectors, $\forall j,\ j = 1..K$, is verified since each node (i.e., phone) N_i, $i = 1..K$, performs independent event sensing. Given the property of independence, we can re-write Equation 3.1 as:

$$IF = \underset{\mathbf{j}, j \in \{1..S\}}{\arg\max} \{ \prod_{i=1}^{K} \mathbf{LI}^{i}_{1}, \prod_{i=1}^{K} \mathbf{LI}^{i}_{2}, .., \prod_{i=1}^{K} \mathbf{LI}^{i}_{S} \} \tag{3.2}$$

Since each node N_i has the vectors $\mathbf{LI}^{k}_{1..S}$ after the local inference propagation phase (where k $\in$ $\{1,..,K\}$), each node N_i can compute Equation 3.2 to produce the final inference. In order to assure that Equation 3.2 is calculated using the local inference results of the same event, Equation 3.2 is computed only over the local inference results that are aligned in time. For each computation of Equation 3.2 only the local inference results which differ in time for at most δ ms are considered. In our implementation, we set δ = 150 ms, which we determine experimentally to be the best value. The larger δ, the bigger is the likelihood that the local inference results refer to different events. The smaller δ, the higher is the likelihood to capture just one event, but the closer we get to the phones time synchronization error. Because time stamps are associated with **LI** vectors (see Section 3.2.6) it is possible for the phone to determine if an event is sampled at the same time.

In order to provide a more robust system against false positives, a windowing approach is adopted where a speaker is deemed to be speaking only if they are detected for at least one more time in the past two iterations of Equations 3.2. This policy, similar to the windowing policy introduced for the local inference in Section 3.2.6, provides experimentally the best tradeoff between classification delay and classification accuracy.

3.3 Discovering the Sensing Context

Efficiently computing the low level context of the phone, that is, the position of the phone carried by a person (e.g., in the pocket, in the hand, inside a backpack, on the hip, arm mounted, etc.) in relation to the event being sensed - which we call the *phone sensing context* - is a fundamental building block for new distributed sensing applications built on mobile phones and for Darwin. These observations have grown out of our implementation of CenceMe discussed in Chapter 2.

While there has been significant research in the area of context aware applications and systems, there has been little work on developing reliable, robust, and low cost (i.e., in terms of energy efficient and computational costs) algorithms that automatically detect the phone sensing context on mobile phones. We envision a future where there are not only personal sensing applications but we see the mobile phone as enabling global sensing applications where the context of the phone in relation to the sensing event is crucially important.

The different context impacts the fidelity of a sensing application running on mobile phones. For example, the camera is of little use in the pocket but the microphone might still be good [40]. Researchers are developing new sensors for the phones that we imagine will be available over the next decade, these include CO_2 and pollution sensors [35]. If the phone is carried inside the pocket or a backpack, an application relying on CO_2 or pollutants measurements would perform very poorly

given that the phone is not exposed to open air. A better position for such sensing would be out of the pocket when the phone is exposed to a more suitable context for sensing. Similarly, if the accelerometer readings of the phone are used to infer the person's activity, the accelerometer would report different data if the phone is mounted on the arm or clipped to the belt. This is because, given the same activity, such as walking for example, arm swings would activate the accelerometer much more strongly for an arm-mounted phone than on the belt, where the phone oscillates more gently. In both cases a mechanism to infer the context of the mobile phone is needed in order to make the applications using the CO_2 or pollution sensor and the accelerometer, respectively, react appropriately. We envision a learning framework on the phone that is more sophisticated than what is implemented today. For example, when sensors report different sensor readings according to the position on the body, such as the accelerometer, the application's learning engine should switch to different classification algorithms or sensor data treatment policy in order to meet the application requirements.

Today the application sensing duty-cycle is costly because it is not driven by the phone sensing context, therefore, it is costly in terms of energy usage for sensing, computation and potentially communications if the inference is done on the backend, as in the case with split-level classification [14]. By offering system developers accurate phone sensing context prior to running classification algorithms, very low duty-cycle continuous sensing application systems are possible. In this case, the phone sensing context mechanism would refrain the application from activating a power hungry sensor if the context is unsuitable (e.g., don't activate the pollution sensor if the phone is not out of the pocket) or it may weight real-time sensor readings or inferences based on knowledge of where the phone is on the body (e.g., if the microphone is needed to measure human activity [40] and it is in the bag). In what follows we discuss Discovery [80], a framework that addresses the context problem supporting mobile phone-based sensing with improved accuracy and lower duty-cycle systems. Discovery is designed to automatically detect the phone sensing context as people move about in their everyday lives. Automatic context detection is a primary issue for smartphone sensing applications because prompting the user to provide information about the position of the mobile phone on the body is not a viable and scalable solution. Phone sensing context is an important building block toward the successful implementation of Darwin and personal, social, and public sensing applications on mobile phones. Discovery, while preliminary, provides important steps towards the goal of providing reliable phone sensing context.

3.3.1 Phone Sensing Context

Accurate, robust and low duty-cycle detection of phone sensing context is an important enabler of distributed sensing applications on phones, in particular, continuous sensing applications that sample sensors, make inferences, and communicate with the backend services in real-time.

Assume mobile phones are equipped with pollution, CO_2, or more specialized environmental sensors as we imagine [35]. Measurements from any of these sensors would most likely be impeded by the presence of clothing or fabric (e.g., phone inside the pocket or backpack) or by a short time

interval the sensors are exposed to an ideal sensing context (i.e., phone in hand or exposed to open air). Therefore, phone sensing context detection would improve the sensing system performance. We could stop the system from activating the sensors when the quality of the sensor data is likely to be poor (e.g., phone inside the pocket). This would help reduce the sensing duty-cycle improving the battery lifetime of the phone, which continuos sensing application significantly limit today (e.g., phones running CenceMe [14] were initially limited to only 6 hours of operation). We could inform the system when a suitable sensing context is triggered or detected (e.g., phone taken out of the pocket) to maximize the accuracy and robustness of the sensing application which would then take advantage of the new context for collecting as many sensor readings as possible. It is evident the importance of the phone sensing context role in driving mobile phones sensors duty-cycle lower.

Another reason to provide phone sensing context as a low level service on phones is to improve the inference fidelity of distributed sensing applications. Although previous work [27] shows that it is possible to obtain reasonably good activity classification accuracy when using training data from sensors mounted on different parts of the body, it is not clear how an activity classifier would perform when the device is a phone, not specifically mounted (but moving as a dynamic system), and operates in noisy, everyday environments that people find themselves in, rather, than under laboratory test conditions. Many questions remain. Would training data from many activities and different parts of the body make a single classification model accurate enough? To avoid excessively diluting the training data set, would it not be preferable building a classification model for each single activity and position of the mobile phone on the body and then switch models according to the detected phone sensing context? For example, a system could have a “walking” activity classification model for when the mobile phone is in the pocket, in the person’s hand, and in the backpack and use one of the models according to the detected phone sensing context. Results obtained from experimentation in [14] show, for example, that activity classification accuracy varies when the phone is carried in the pocket or in the hand. A system that used phone sensing context to drive the classification model by switching in the right technique would alleviate this problem. We believe this is of importance now that smart phones are growing in sensing and computational capability and new demands are emerging from different sectors such as healthcare. It is important to note that in the case of health care sensing applications it is fundamental to limit the classification error. Sensing context detection could drive inference model switching in order to achieve better classification accuracy.

We argue that phone sensing context detection could also be exploited by existing phone applications and services. For example, by inferring that the phone is in the pocket or bag, a caller might be informed about the reason the callee is not answering the phone call while the callee’s phone ring tone volume could be increased so the callee might pick up. One could imagine people enabling this type of additional presence provided to legacy phone service through Discovery. By using the gyroscope (which measures the angular rate change of the phone) to detect the user taking the phone out of the pocket and moving it upwards, the screen saver could be disabled and the phone’s keypad made automatically available. One could imagine many such adaptations of the UI with phone sensing context enabled. Similarly, the action of moving the phone towards the lower part of the

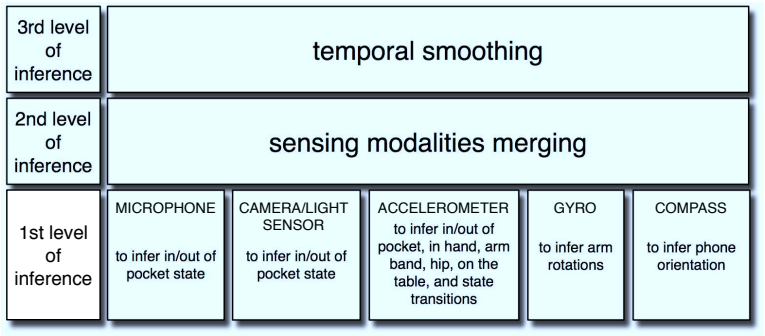


Figure 3.4: Discovery's inference steps.

body could trigger power saving mode. The camera application on the phone could be automatically started as the phone is detected in the user's hand and moved in a vertical position, which is the condition that normally precedes the action of taking a photo. One could imagine phone sensing context provided by the Discovery framework discussed in the next section being applicable to many emerging applications finding their way on to smartphones. For example, reality mining using mobile phone sensor data is starting to be explored as an enhanced form of communication and for social purposes.

3.3.2 Discovery Design

The idea behind Discovery is to use the entire suite of sensing modalities available on a mobile phone to provide enough data and features for context discovery at low cost and for increased accuracy and robustness. Many research questions arise in response to the challenges discussed above: how do we combine the input from multiple sensors, such as, accelerometer, microphone, gyroscope, camera, compass, etc., to infer the phone sensing context? What are the best learning approaches and feature selection policies in order to provide a reliable and scalable context inference system? How do we design low duty-cycling policies with acceptable accuracy when employing phone sensing context? What is the inference accuracy and energy cost tradeoff between using all the possible sensors and only a subset of them according to their availability on the mobile phone? Which sensor set is more responsive to the type of noise in the system (i.e., classification outside controlled laboratory environments)? We believe that Discovery in its totality needs to ultimately address these demanding challenges. However, our preliminary work focuses on a simple phone sensing context: is the phone in the pocket or out. This sounds like a trivial context that could be solved by a number of different sensors. We focus on the microphone - a powerful and ubiquitous sensor on every phone on the market - making Discovery suitable to potentially all phones not just the smart ones. In what follows, we outline out initial framework design.

Discovery consists of a hierarchical inferences pipeline, as illustrated in Figure 3.4:

First Level Inference - Uni-sensor inference: In this phase, the sensor data from individual sensors is used to operate a first level of inference. Features extraction is tailored to each sensor. This first inference step provides hints about the nature of the current phone sensing context, which,

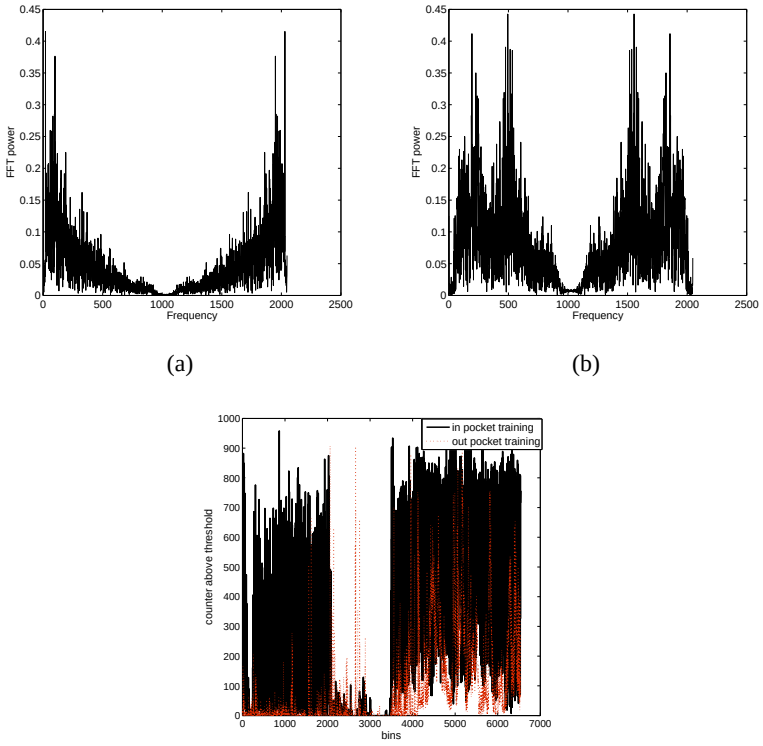


Figure 3.5: (a) FFT power of an audio clip when the phone is inside the pocket; (b) FFT power of an audio clip when the phone is outside the pocket; (c) Count of the number of times the FFT power exceeds a threshold T for both the in-pocket and out-of-pocket cases.

however, might not be conclusive. For example, the use of the camera or light sensor to infer if the phone is in or out the pocket could be misleading because a phone out of the pocket could be in a dark environment, the camera could be covered by the person's hand or by the surface where the phone is positioned. For this reason, a second level of inference built on top of the first is needed.

Second Level Inference - Multi-sensor inference: In this phase, the inference process is based on the output of the first phase. Hence, the first level of inference provides the features to the second level. At this stage, the combination of the camera/light sensor and microphone output would provide better confidence about the actual sensing context. The accelerometer as well could be used as a hint to determine if the phone is inside or outside the pocket given the different accelerometer data signatures when the phone is in a person's hand versus when it's in the pocket. Similarly, by measuring the angular rate change, the gyro could provide indications that the phone has been taken out of the pocket considering that the arm rotation would be picked up by the gyroscope.

Third Level Inference - Temporal smoothing: In this phase, temporal smoothing and Hidden Markov Model (HMM) techniques are used on the output of the second level inference. This step exploits the correlation in time of sensed events when a phone experiences a certain context.

3.3.3 Discovery Implementation

For our initial implementation of Discovery context classifiers are implemented on the Nokia 95 and Apple iPhone. The preliminary system implements a set of sophisticated inference models that include Gaussian Mixture Model (GMM) and Support Vector Machine (SVM) on the Nokia N95 and Apple iPhone with focus on a limited set of sensors and inferences; that is, we uses the microphone sensing modality to infer the phone sensing context of in the pocket and out of the pocket. We discuss our initial results in the next section. Further modalities, such as accelerometer, compass, and light sensor, are going to be used in combination with the microphone to infer a larger set of sensing context as part of our future work. The initial idea is to evaluate which learning technique (between GMM and SVM) is better suited to the problem and, at the same time, to investigate the adoption of more than one learning strategy in concert to perform the final classification. More learning strategies will be evaluated in the following phase of this work. The challenge with GMM and SVM is that the phone has not been developed to run these computationally demanding models. Part of our efforts is to implement light weight versions of these models as a way forward to do more sophisticated multi-inference classification, as called for by Discovery. In particular a 20-component GMM is adopted, where the number of components is chosen by evaluating the model over the test data set varying the number of components and picking the number of components returning the best classification accuracy.

Feature Selection. The selection of an appropriate set of features is a key step to good classification performance. At the moment, a supervised learning approach is adopted and Discovery relies on a 23-dimensional feature vector extracted from an audio clip. A richer selection of features will be evaluated as part of our future work. The current features are:

1st-19th: Mel-Frequency Cepstral Coefficients (MFCC), which have been proven to be reliable features in audio signal classification problems. For the MFCCs extraction we rely on a well-known Matlab libray [142] which is largely used by the research community. We also developed a C version of the MFFC extractor library that can run on the phone;

20th: power of the audio signal calculated over the raw audio data;

21st, 22nd: mean and standard deviation of the 2048-point FFT power in the 0-600 Hz portion of the spectrum. The reason for focusing on this portion of the spectrum can be seen from Figures 3.5(a) and 3.5(b), where the presence of a pattern between the two FFT distributions - for in pocket and out-of-pocket recording - is clear. It can be seen that such a pattern is more evident in the 0-600 Hz portion of the spectrum rather than in the whole 0-1024 Hz range;

23rd: this feature is the count of the number of times the FFT power exceeds a certain threshold T. This threshold is determined by measuring the Euclidean difference between the count of the in-pocket and out-of-pocket cases and picking the threshold that maximizes such a distance. An example of the count for both the in-pocket and out-of-pocket cases is shown in Figure 3.5(c) where it can be seen how these features can be used to discriminate between the in pocket and out of pocket cases. The x-axis of Figure 3.5(c) reports the number of bins the clip has been split in to.

Consequently, for the mixture model, a 20-component, 23-dimensional GMM is used. The

Table 3.1: Sensing context classification results using only the microphone. Explanation: when a result is reported in X/Y form, X refers to the *in pocket* case, and Y refers to the *out of pocket* case. If the column reports only one value, it refers to the average result for both *in* and *out* of pocket. Legend: **A** = GMM; **B** = SVM; **C** = GMM training indoor and evaluating indoor only; **D** = GMM training outdoor and evaluating outdoor only; **E** = SVM training indoor and evaluating indoor only; **F** = SVM training outdoor and evaluating indoor only; **G** = GMM training using only MFCC; **H** = SVM training using only MFCC.

Classification results	A	B	C	D	E	F	G	H
Accuracy	84% / 78%	80%	75% / 84%	84% / 83%	68%	81%	77% / 79%	71%
Error	16% / 22%	20%	25% / 16%	16% / 17%	32%	19%	23% / 21%	29%

SVM classifiers adopts the 23 dimensional feature vector.

Training. The training phase is performed using audio data collected with a Nokia N95 and Apple iPhone in different settings and conditions from a person going through different environments for several hours. Namely, the audio is recorded in a quiet indoor office environment and an outdoor noisy setting (along a road with cars passing by). In both scenarios the phone is carried both in the pants pocket and outside the pocket in the hand. The choice of these scenarios, i.e., indoor and along a road, is motivated by the fact that they are representative of classes of locations where most likely people spend a lot of their time while carrying their phone both inside and outside the pocket. For each configuration 14 minutes of audio are recorded at different times. Half of each clip (i.e., about 7 minutes of audio) is used to train the classifiers. The training data is finally labeled accordingly.

Prediction. For prediction, the remaining half of each audio clip not part of the training set (i.e., duration of about 7 minutes) is used. Each sample consists of a 96 msec chunk from which the 23 features are extracted. For each configuration there are about 58000 samples available for training and 58000 for evaluation.

3.3.4 Preliminary Discovery Evaluation

In what follows, preliminary results from using both the GMM and SVM classification techniques are reported. The results highlight that the audio modality is effective in detecting the in/out of pocket context with reasonable accuracy. Higher accuracy can be achieved by combining further modalities such as accelerometer and light sensor. Columns A and B in Table 3.1 show, respectively, the classification results for GMM and SVM when the training data combines both indoor and outdoor audio and the phone is carried in and out the pocket. The results are quite encouraging, since we obtain about 80% accuracy (see the accuracy values in columns A and B) adopting a non sophisticated feature set and using only one sensing modality, i.e., the microphone. We are confident that by involving more sensing modalities into the classification process, for example the accelerometer and light sensor, a more accurate selection of the feature vector, and temporal smoothing, it might be possible to achieve a much higher classification accuracy. We then train and evaluate the models for only one scenario, i.e., either indoor or outdoor. The results using GMM are in Table 3.1 column C and column D. The results for SVM are in column E and column F. In

the case of SVM trained and evaluated for the indoor scenario only (see column E) the accuracy is lower than the other cases because Libsvm (the well known SVM library implementation we adopt) is running with the default settings with the kernel optimization being disabled. From these results it is interesting to see that training the models with both indoor and outdoor data does not dilute the training data and the final classification accuracy does not drop significantly compared to the case when the models are trained for a single scenario only and evaluated for the same scenario. In fact, the accuracy in columns C, D, and F is on average close to 80% as in the case of indoor and outdoor training data set (see columns A and B). Columns G and H in Table 3.1 show, respectively, the classification results for GMM and SVM when the model is trained using only MFCCs (hence a 19-dimensional feature vector). It is evident that the addition of the 4 extra features (i.e., signal power, FFT mean, FFT stddev, and number of times a threshold is exceeded by the FFT power) boosts the classification accuracy. The improvement can be seen by comparing the results in columns G and H with the ones in columns A and B.

3.3.5 Discovery Future Work

After the initial promising results, the goal is to implement a working prototype for the Android platform as well. More sensing modalities are going to be used in combination with the audio modality. In particular, the accelerometer, magnetometer, and light sensors. Research is going to be needed in order to identify the most suitable feature vector elements that combine the characteristics of all the sensing modalities. Temporal correlation between events is also going to be taken into consideration to improve the overall accuracy. Techniques such as HMM or voting strategies will be taken into account. We will also pursue the idea of letting people customize the Discovery classifiers to accommodate their habits and needs.

3.4 Privacy and Trust

Security, privacy, and trust raise considerable challenges in the area of smartphone sensing. While we do not present solutions to these challenges, those solutions are critical to the success of the research discussed in this Chapter. Darwin incorporates a number of design decisions that are steps towards considering these challenges. First, the raw sensor data never leaves the mobile phone nor is it stored on the phone – we only store models and features computed from the raw sensor data. However, features and models themselves are sensitive data that needs to be considered appropriately and therefore protected. In the case of the speaker recognition application, the content of a conversation is never disclosed, nor is any raw audio data ever communicated between phones. The data exchanged between phones consists of classification confidence values and event models. Next, mobile phone users always have the ability to opt in or out of Darwin, hence, no model pooling and collaborative inference would take place unless the users make such a determination.

To meet privacy, security, and trust requirements Darwin phones should: i) run on trusted devices; ii) subscribe to a trusted system; and iii) run a trusted application that is either pre-installed

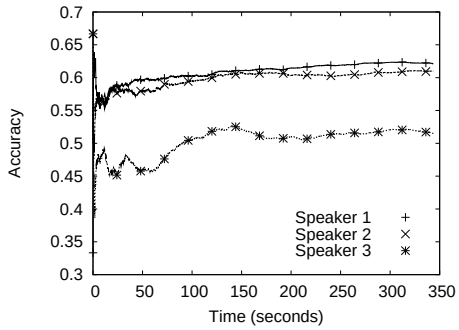


Figure 3.6: Accuracy, without evolution, for three speakers when walking along a busy road without classifier evolution and having trained the classification models for indoors only.

on the phone or downloaded from a trusted third party (e.g., Apple App Store, Nokia Ovi Store, or Android Market). Any phone-to-phone interaction (e.g., pooling and collaborative inference) should be regulated by off-the-shelf authentication and authorization mechanisms that prevent the injection of malicious code or intentionally distorted inference results from adversaries.

3.5 System Performance

In what follows, we first discuss the implementation of Darwin on the Nokia N97 and Apple iPhone and then present the detailed evaluation results of the Darwin system supporting the speaker recognition application.

3.5.1 Phone Implementation

Darwin is implemented on the Nokia N97 using C++, Kiss FFT [104] for fast fourier transform (FFT) calculations, and QT [143], which is a wrapper around C++ for the graphical user interface. On the Apple iPhone we use C++ and the FFTW fast fourier transform library [144]. The necessary algorithms, i.e., GMM training, the probability density function, and MFCC extraction are ported to the N97 and iPhone from existing Matlab code that we verified to work correctly. We plan to make this toolkit available in the future as an open source project. The availability of this toolkit on a phone is a considerable resource for building more powerful classifiers. The backend server responsible to run the model training and re-training for evolution is implemented on a Unix machine using C and standard socket programming for communications. A UDP multicast client is implemented to allow local inference results propagation whereas an ad-hoc lightweight reliable transport protocol has been built to send feature vectors to the backend for model training, to send trained models to the phones, and to exchange models during the pooling phase. Because we target heterogeneous scenarios, an audio sampling rate of 8KHz is used in order to run Darwin at the same time on the iPhone 3G, which can drive the microphone up to 48KHz sampling, the iPhone 2G and the Nokia N97, which only support 8KHz audio sampling rate.

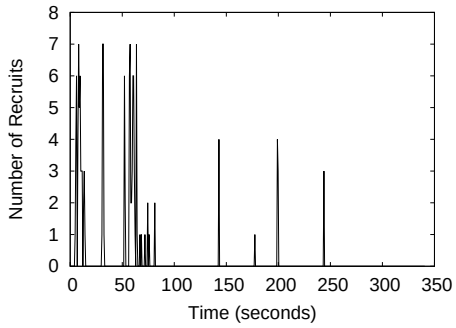


Figure 3.7: Evolution sessions count over time in the indoor scenario for speaker 8.

3.5.2 Experimental Results

We evaluate the Darwin system using a mixture of five N97 and iPhones used by eight people over a period of two weeks generating several hours of recorded audio containing speech portions. In order to evaluate the system against ground truth data, the audio data is manually labeled by extracting the voicing chunks of each of the eight speakers. The audio is recorded in different locations under differing conditions such as a quiet indoor environment, walking on a sidewalk along a busy and noisy street, and in a noisy restaurant. This provides a good basis to validate Darwin under very different operating conditions. We only present a subset of the results from our experiment due to space limitations.

The Need for Classifier Evolution

We conduct a simple experiment to show the need for classifier evolution on mobile phones. Three people walk along a sidewalk of a busy road and engage in conversation. The speaker recognition application without the Darwin components runs on each of the phones carried by the people; that is, no classifier evolution, model pooling, and collaborative inference algorithms are enabled for the experiment. The voice classifier for each person is trained in a quiet indoor environment. We quantify the performance of a classification model trained indoors when operating in a noisy outdoor setting. Figure 3.6 shows the classification accuracy [145] for mobile phones 1, 2, and 3 for speaker 1, 2, and 3, respectively. It is evident from the plot that the accuracy is quite low because the maximum speaker classification accuracy for speaker 1 and 2 is 63% and 61%, respectively, and only 52% for speaker 3. The poor performance is because the classification model trained indoor performs poorly outdoors. This highlights the challenge presented when sensing in different environments. Building audio filters capable of separating voice from other types of noise is challenging and would not likely scale given the large pool of possible sounds that may be encountered by a mobile phone on the street or in any other environments. This result motivates the need for designing efficient classifier evolution techniques that operate transparently to the person carrying the phone in order to evolve classification models according to the scenarios where people find themselves.

Let us take a look now at the evolution algorithm performance. Figure 3.7 shows the distribu-

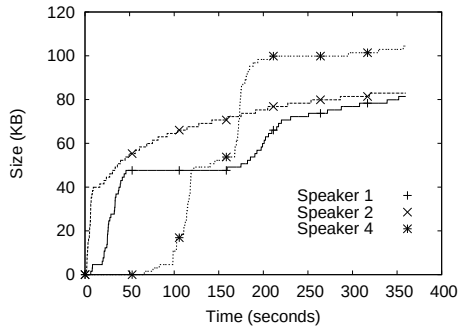


Figure 3.8: Size of the data set recruited during the evolution phase in the restaurant scenario.

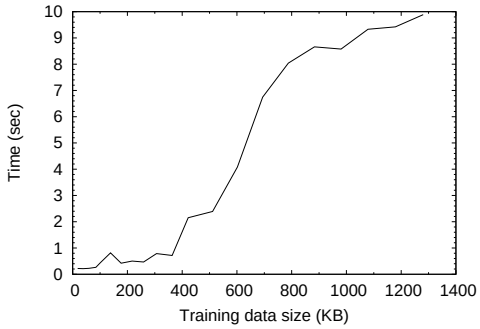


Figure 3.9: Server training computation time as a function of the training data size. The server has a 2.4GHz cpu and 4GB of RAM.

tion, for the duration of the experiment, of the number of audio chunks recruited by the classifier evolution algorithm for speaker 8 in the indoor case. Similar results are observed for other speakers not shown in the results. As expected, a larger number of chunks are recruited during the first phase, when the application is run after the initial training, than towards the end, when the model has already been refined and little or no model evolution is required.

Figure 3.10 shows the accuracy improvement as the amount of data sent from the phone to the backend for re-training grows. This result refers to a different outdoor environment than the one in Figure 3.6. This second outdoor scenario is noisier than the first one, causing the initial accuracy before evolution to be lower than the one reported in Figure 3.6. Clearly, the larger the training set capturing the characteristics of the new environment the better the performance. Figure 3.8 shows the amount of audio data recruited by the evolution algorithm for the duration of the restaurant experiment. It is interesting to see that after a few minutes of conversations the models of the three speakers diminish their training data recruitment and model evolution eventually stops (as happening for speaker 1 model). This confirms the behavior observed for Darwin in the quiet indoor environment. The only difference is in the amount of recruited data between the restaurant and quiet scenario. In fact, in the former case the amount of data recruited ranges between 78KB and 100KB, whereas in the indoor case it is between 16KB and 70KB. This is in line with the fact that less data is needed to evolve an indoor model than a “restaurant” model (i.e., a very different

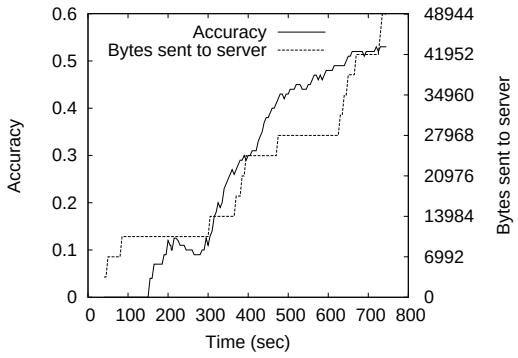


Figure 3.10: Classifier accuracy and the amount of needed training data in an outdoor scenario.

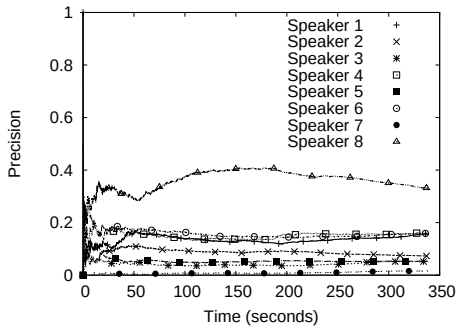


Figure 3.11: Precision for speaker 8 basic inference when speaker 8 is speaking in an indoor quiet setting.

environment from a quiet place) since initial training is performed in a quiet place. Figure 3.9 reports the server training algorithm running time as the training data set grows. The measurement refers to the outdoor case and levels off when the accuracy (shown in Figure 3.10) reaches the maximum. Right before the end of evolution the server takes about 10 seconds to train the GMM model using a 1.3MB data set.

Experimental Scenario One: Quiet Indoor Environment

We first show the performance of the speaker recognition application analyzing data collected from 5 different phones concurrently running the Darwin system in a meeting setting in an office environment where 8 people are involved in a conversation. The phones are located at different distances from people in the meeting, some on the table and some in people's pockets. The aim of the experiment is to study the impact of different phone sensing context showing that a low classification accuracy due to adverse context can be compensated by Darwin.

In Figure 3.11 we show the classification precision [145] for speaker 8 calculated by speaker 8's phone using the basic speaker recognition application when speaker 8 is talking. Figure 3.11 shows that the precision of the basic speaker recognition application is below 50%. This result indicates that speaker recognition using an individual mobile phone is challenging. The reason is that the phone could have poor sensing context for example in the pocket (as for part of the conversation of

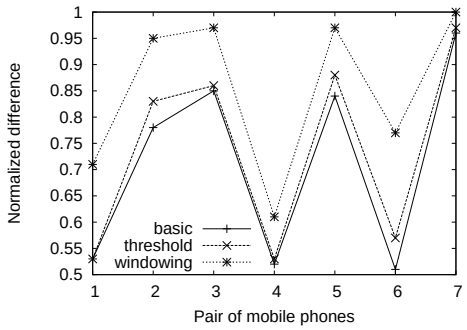


Figure 3.12: Normalized true positive-false positive difference between speaker 8 and all the other speakers when speaker 8 is speaking. The closer the normalized difference to 1, the larger is the true positives compared to false positives.

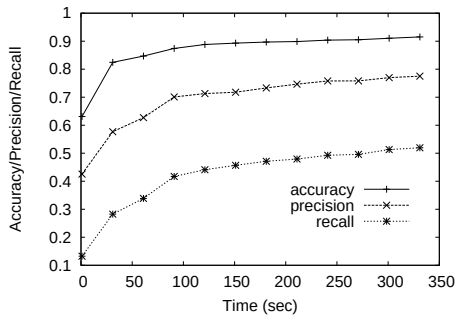


Figure 3.13: Mean recall, precision, and accuracy in an indoor quiet environment with collaborative inference for the eight speakers.

speaker 8) or affected by other factors such as noise mixed with voice.

Figure 3.12 shows the benefit of applying the 50% thresholding technique and windowing policy (described in Section 3.2.6) to the basic speaker recognition classification algorithm. The 50% thresholding technique makes sure that there is a 50% difference between the confidence of the inferred speaker and every other speaker, whereas the windowing policy reduces the effect of false positives. Figure 3.12 shows the difference between the true positive and false positives counts normalized by the true positive count for speaker 8 speaking. The closer the normalized difference to 1, the larger is the number of true positives versus the number of false positives. Figure 3.12 shows the combined positive effect of the 50% thresholding and windowing techniques, which makes the normalized difference larger compared to the basic technique. This is a first step toward the final inference, which is part of the collaborative inference phase of Darwin; however, it is not enough to achieve higher classification accuracy. In fact, when the Darwin system is activated a further performance boost is registered.

Figure 3.13 shows results for the mean recall [145], precision, and accuracy results for the eight speakers classification when Darwin is running. It is evident from the plot that Darwin boosts the classification performance of the speakers identification. The boost is due to collaborative inference, which leverages the other co-located mobile phones in order to achieve better classification

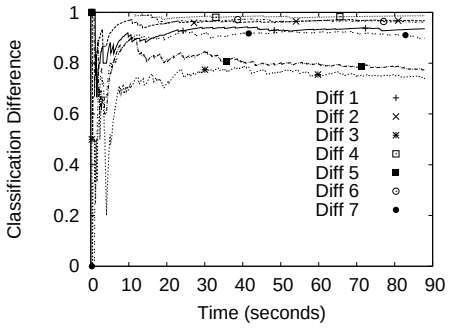


Figure 3.14: Classification difference between speaker 8 and the other speakers without Darwin.

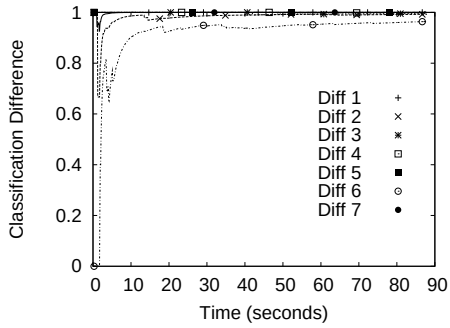


Figure 3.15: Classification difference between speaker 8 and the other speakers with Darwin.

accuracy.

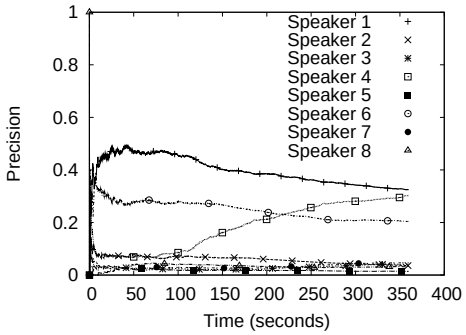
The other interesting result from Figure 3.13 is the positive impact of Darwin's classifier evolution. The accuracy, precision, and recall increase over time at the beginning of the experiment as new audio is recruited and the classification models re-trained taking into account the new audio. The performance for accuracy, precision, and recall levels off at the point where the classifier evolution algorithms does not benefit from more data, as discussed in Section 3.2.4.

The benefit of model evolution, model pooling, and collaborative inference can be also seen in the results shown in Figures 3.14 and 3.15. If we indicate with TP and FP, respectively, the true and false positives when speaker 8 is speaking, the y-axis of the plots reports the quantity (TP-FP)/TP over time. A FP count is maintained for each speaker, thus in Figures 3.14 and 3.15 eight curves are shown.

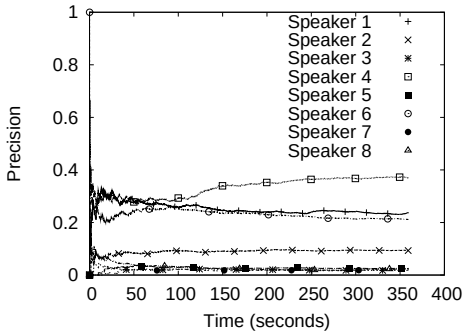
In one case Darwin is disabled (Figure 3.14), in the other case Darwin is enabled (Figure 3.15). When (TP-FP)/TP is close to 1 it means that the number of true positives dominates the number of false positives. In contrast, if (TP-FP)/TP is close to 0 we have that the number of false positives approximates the number of true positives. We can see that the difference between speaker 8's true positives and any other speakers' false positives is larger when Darwin in running (as shown in Figure 3.15) than when it is not (see Figure 3.14). This is another indication of how Darwin improves the classification result for a given speaker.

Experimental Scenario Two: Noisy Indoor Restaurant

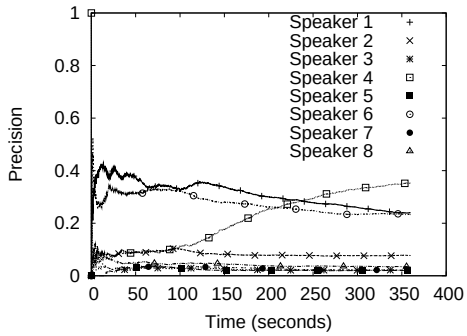
In the next experimental scenario, we evaluate Darwin when the speaker recognition application is running on five phones while five people are having a meal in a noisy restaurant. This contrasts the first scenario of a quiet indoor setting. Three out of five people are engaged in conversation. Two of the five phones are placed on the table, the other phones are in the pants pockets for the entire duration of the experiment. In Figure 3.16, we show the classification precision of speaker 4 from three of the mobile phones located around the table people are sitting at; note, we observe similar trends for the other phones. Figure 3.16(a) is the precision computed by speaker 4's phone, which is



(a) Precision for speaker 4 calculated by the speaker's mobile phone without collaborative inference.



(b) Precision for speaker 4 calculated by mobile phone 1 without collaborative inference.



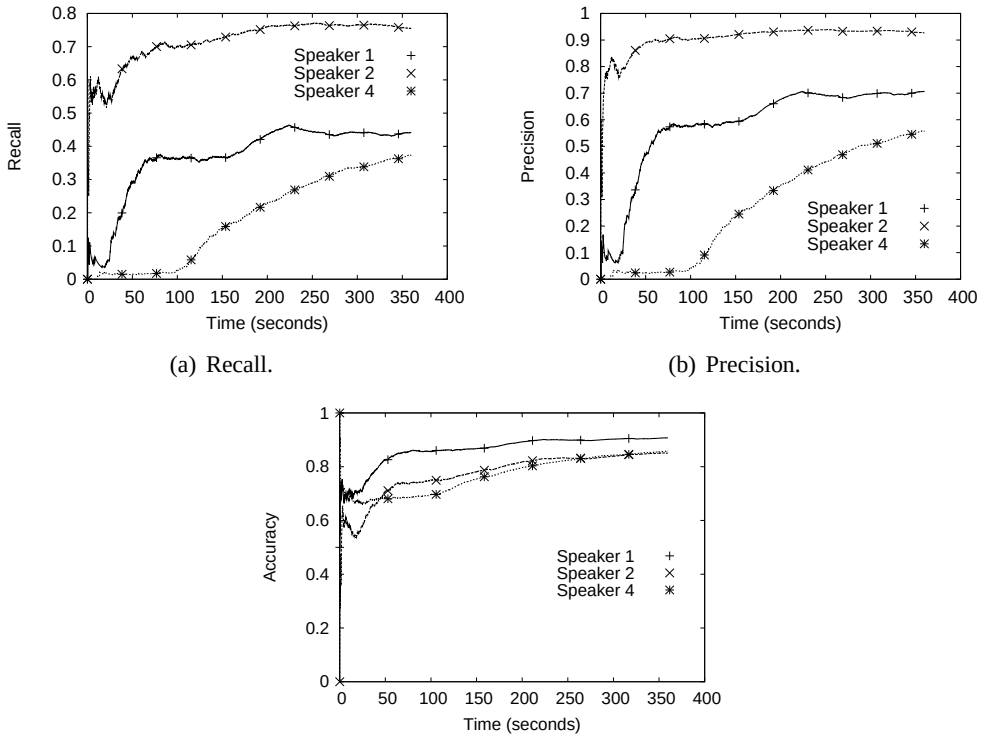
(c) Precision for speaker 4 calculated by mobile phone 2 without collaborative inference.

Figure 3.16: Precision for speaker 4 on different phones in a noisy restaurant environment without collaborative inference.

the closest phone to speaker 4 (it is carried by speaker 4) for when speaker 4 is talking. The reason we select speaker 4 for the evaluation is that speaker 4 is the closest person to a nearby table where another group of people is having a loud conversation. Here we show the benefit of the Darwin system for the phone of a speaker who is positioned in a non optimal context, i.e., close to a noise source.

Figures 3.16(b) and 3.16(c) refer to the precision calculated by two other phones, which we call phone 1 and 2, located at the opposite side of the table where speaker 4 is sitting. Figures 3.16(b) and 3.16(c) show on average higher classification precision on phones 1 and 2 when speaker 4 is talking than when the classification is performed by speaker 4's phone reported in Figure 3.16(a). This is because phones 1 and 2 are more distant from the source of noise and consequently they are able to capture higher quality speaker 4's voice audio than speaker 4's phone itself.

The important point is that Darwin boosts the classification performance of speaker 4 by leveraging other surrounding phones experiencing better sensing contexts; this is shown in Figures 3.17(a), 3.17(b), and 3.17(c) which report, respectively, the recall, precision, and accuracy of the three speakers in the restaurant experiment, including speaker 4, over time. From Figure 3.17(c) it can be seen that speaker 4's accuracy is low ($\sim$65 %) at the beginning of the experiment but starts increasing



(a) Recall.

(b) Precision.

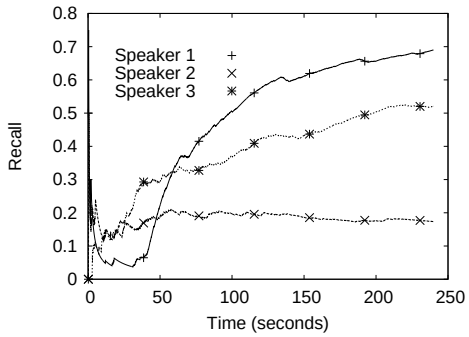
(c) Accuracy.

Figure 3.17: Recall, precision, and accuracy in a noisy restaurant with Darwin for three speakers.

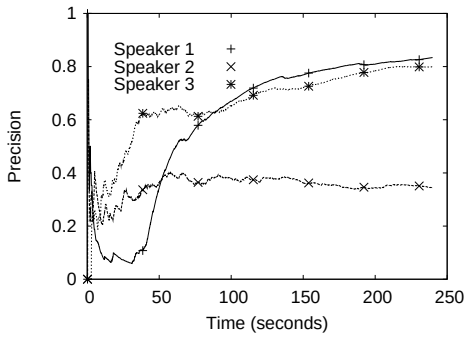
as the experiment proceeds. The classifier evolves and the surrounding mobile phones participate in collaborative inference. At the end of the experiment, speaker 4's classification accuracy reaches $\sim$80%. The fact that speaker 4's recall and precision present low values for the duration of the experiment (as shown in Figures 3.17(a), 3.17(b), respectively) confirms that speaker 4 voice is impacted most of the time by loud conversation from the next-table.

Experimental Scenario Three: Walking Outdoors Along a Sidewalk in a Town

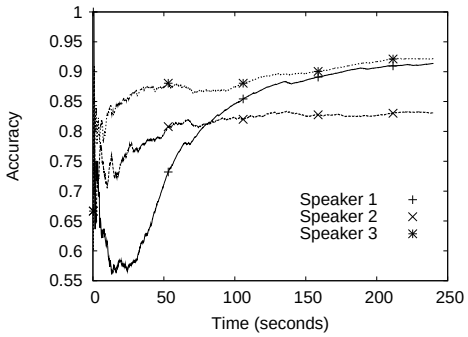
The final scenario we study is an outdoor environment where five people walk along a sidewalk and three of them are talking. This contrasts the first two scenarios. In this experiment, five people carry phones either clipped to their belt or inside their pockets. As in the restaurant scenario, the amount of audio data recruited by Darwin to evolve the classifier is larger than the indoor evolution case and ranges between 90KB and 110KB of data. The performance boost using the Darwin system can be observed in Figure 3.18 where the speaker recognition classification accuracy increases to 80-90%. The greatest improvement is observed by speaker 1 whose phone is clipped to their belt. This mobile phone is exposed to environmental conditions such as wind and passing cars making the audio data noisy and voice hard to pick up. However, we note that some of the other phones experience better sensing context and by relying on these phones Darwin boosts the final classification accuracy for speaker 1.



(a) Recall.



(b) Precision.



(c) Accuracy.

Figure 3.18: Recall, precision, and accuracy for three speakers walking on a sidewalk along a busy road.

3.5.3 Impact of the Number of Mobile Phones

In this experiment, we study how the Darwin system's performance changes as the number of mobile phone participating in model pooling and collaborative inference varies. This experiment is also conducted in the same noisy restaurant discussed in scenario two, which represents a challenging sensing environment. The experiment consists of three people speaking and five phones carried by five people positioned around the table. Some phones are placed on the table and others remain in speaker's pockets. The experimental scenario starts with only two of the five phones running the Darwin system. More nodes are subsequently added up to a maximum of five – all phones run the Darwin system and are randomly positioned around the table. The classification accuracy for each of the three speakers as a function of the number of phones running Darwin is shown in Figure 3.19. As expected, the larger the number of co-located mobile phones running Darwin, the better the inference accuracy. The performance gain using collaborative inference grows with the number of phones according to the algorithm discussed in Section 3.2.6 and in particular to Equation 3.2. While two phones do not have sufficient "spatial" diversity to experience gain from the Darwin system, the classification accuracy is boosted when the three remaining phones are added.

Speaker 3 experiences low accuracy due to the proximity of their phone to another group of people involved in a loud conversation. This is perceived as noise by the speaker identification classifier and negatively impacts speaker 3's voice classification more than speaker 1 and 2. Speaker 2 ex-

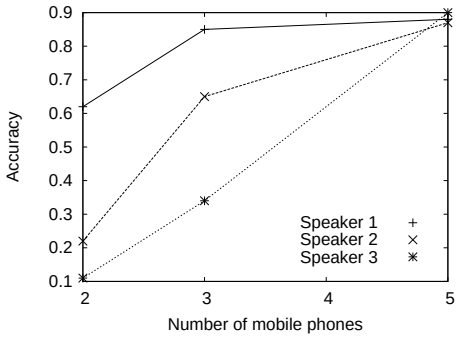


Figure 3.19: Accuracy in a noisy restaurant when an increasing number of phones participate to Darwin.

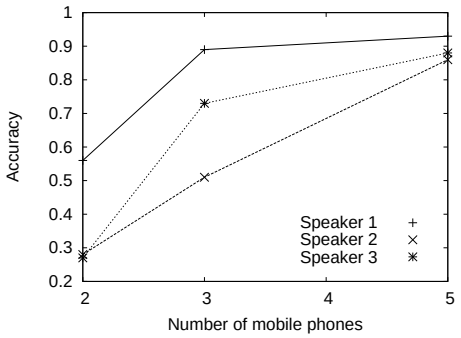


Figure 3.20: Accuracy in a quiet indoor setting when an increasing number of phones participate to Darwin.

periences low accuracy with three phones running Darwin due to speaker 2's voice characteristics. Speaker 2's classification model poorly classifies speaker 2's voice when operating individually. This could be due to the fact that the initial training audio is not recorded correctly or that the 20-component 19-dimensional GMM for speaker 2 does not properly model speaker 2's voice. In this case, a larger number of nodes is needed to perform accurate speaker 2 classification. The Darwin system compensates not only errors due to different sensing context but also for poor event classification modeling. This is possible because multiple phones co-operate to generate more accurate inference results. The confirmation that speaker 2's model is not accurate comes from the fact that speaker 2's recognition with 3 phones performs poorly in comparison with speaker 1 and 3 in a quiet setting (see Figure 3.20), which is where the classifier should perform the best given the initial indoor training stage.

We also determine that the reason for better inference accuracy with 3 phones in the restaurant experiment for speaker 3 is that the other two phones are closer to speaker 3 than they are in the quiet indoor case. This offers better audio signal quality for the collaborative inference step.

In summary, the Darwin system boosts the classification accuracy when the sensing environment or context adversely impacts quality of inference, when the individual classification model yields poor accuracy given a person's voice characteristics (as in the case of speaker 2 for our experiments), and when sensors or microphones have different hardware characteristics [138].

3.5.4 Time and Energy Measurements

When proposing a complex but powerful classification architecture such as Darwin the natural question is: how does this system impact the performance of everyday mobile phones? While we have completed a first implementation of Darwin on the Nokia N97 and Apple iPhone we recognize that there are challenges and future work to be done. In what follows, we present some time and energy measurements for Darwin running on the N97 (similar performance is observed for the iPhone). We believe that smart duty-cycling is also a future part of our work which would improve the energy results presented in this section. Averaged baseline measurement are taken before each measurement

Table 3.2: Average running time for processing 1 sec audio clip, sending, and transmitting the data.

Routine	Running Time (s)
Silence suppression	0.003
Voicing	0.565
MFCC extraction	1.4
Local inference	3.67
TX MFCC to server	0.862
RX model from server	4.7
TX model to neighbors	1.91
TX local inference	0.127
RX local inference	0.09

in order to have a baseline reading, which we subtract from each measurement. The measurements are performed using the Nokia Energy Profiler tool and repeated five times. The mean values are reported. The running time of each Darwin component for 1 second of audio sampling is reported in Table 3.2. The most computationally intensive routines are the local inference (which involves the probability density function calculation for eight speakers) and receiving the model from the server. Figure 3.21 shows the power, CPU load, and memory measurements on the N97 when running the Darwin components. It can be seen that the local inference step takes the largest amount of power (see Figure 3.21(a)) and CPU load (see Figure 3.21(b)). Substantial memory usage is measured for MFCC extraction compared to the other components (see segment (C) of Figure 3.21(c)). This suggests that the MFCC extractor implementation requires optimization. Receiving a new model from the server and broadcasting it to neighbors during pooling also causes more power drain than the other routines. However, evolution and pooling are operations that occur rarely (i.e., evolution only once for a certain environment and pooling only once to send a model to neighbors), consequently, pooling and evolution do not heavily account for resource usage and power consumption. Routines that instead occur periodically, such as audio sampling, voicing, MFCC extraction, etc., require less power each time they are active.

Finally, Figure 3.22 shows the measured battery lifetime and the inference responsiveness (defined as the inverse of inference delay) as a function of the audio sampling interval and collecting 1 second of audio each time the microphone is polled. We obtain the shortest battery lifetime ($\sim$27 hours) for a periodic sampling interval of 10 seconds (this sampling interval guarantees the highest inference responsiveness). However, if smart duty-cycling techniques are adopted [55], the phone could operate in a low sensing duty-cycle mode, e.g., with a sampling rate of 60 seconds, when Darwin is not running. As an event is detected, such as voice in case of speaker recognition, Darwin could become active in a high duty-cycle mode, e.g., using a 10 second sensor polling rate, for the duration of the event. As the event disappears the phone could go back to low duty-cycle mode and Darwin would stop working. This would guarantee high application responsiveness while maintaining several hours of battery duration. More detailed analysis of resource consumption and the development of low-energy duty-cycling for Darwin are important future work. We believe however that new duty-cycling techniques discussed in the literature for mobile phones [55, 83] could boost

the phone's battery lifetime of Darwin phones.

3.6 Demo Applications

Darwin can be used by other emerging smartphone sensing applications in addition to the speaker recognition application discussed in this Chapter. In what follows, we discuss how a number of demo applications that use different sensing modalities can be supported by Darwin. We discuss three different demo applications.

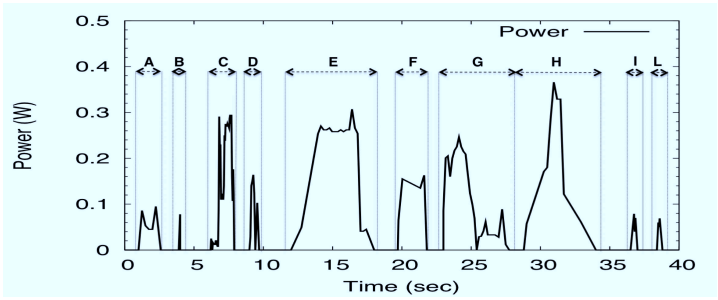
3.6.1 Virtual Square Application

Darwin could support applications in the social sphere setting, where speaker identification, for example, could be used to augment the context of a person and their buddies and make the buddy proximity detection more accurate. Similarly, it could be used for reliable detection of meetings. A pure Bluetooth or WiFi based proximity detection system might not work accurately considering the large amount of devices a Bluetooth or WiFi scan could potentially return in the area where the person is. We have developed Virtual Square, an application that exploits augmented reality to present a person's sensing status information [14] including whom the person is in proximity/chatting with at a certain moment and location. Our Virtual Square prototype is built for the Nokia N97 writing a combination of QT code, for the body of the program, and Symbian routines to access low level functionalities for the collection of magnetometer readings from the onboard magnetometer. GPS and magnetometer sensors are used to geo-tag the person's sensor context which is stored on a server and is retrievable by the buddies who subscribe to the service. Users have full control of the application privacy settings and can opt-out from disclosing who they are chatting with at any time. A screenshot of the application is reported in Figure 3.23. It is clear how Virtual Square, by simply pointing the phone as when taking a picture and moving it around, is a powerful means to see "through walls" and gather information about people and places in the very intuitive way of an augmented reality interface. The old concept of square as a place where people gather to chat and meet now becomes virtual, being enabled by the smartphones sensors which allow to characterize people's microcosmos.

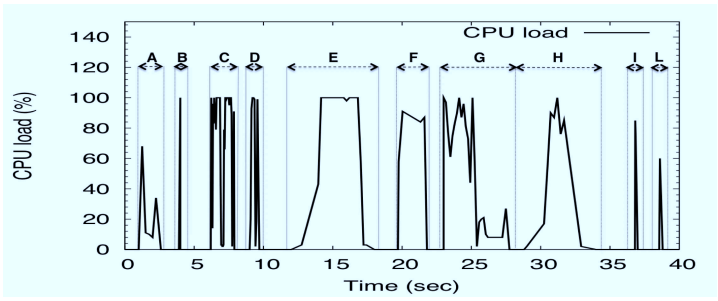
3.6.2 Place Discovery Application

Darwin could support applications using different sensing modalities; for example, place discovery applications based on radio frequency (RF) activity from WiFi access points [141]. Darwin can be integrated with such an application in the following way:

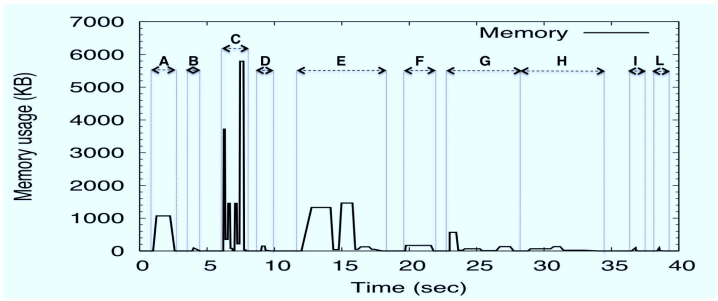
- **Initial training and evolution:** the classification model in this case is the RF signature profile that characterizes a certain place. The dynamics of the radio characteristics and the fact that access points are being added or removed in certain areas make the RF profile time varying. A RF profile could be initially built by a mobile phone and then evolved as the mobile phone visits the same



(a) Power.



(b) CPU load.



(c) Memory.

Figure 3.21: Power, CPU load, and memory usage of the N97 when running Darwin. The Darwin routines have been made run sequentially and the operations have been segmented as reported in the following labels: (A) One sec audio sampling; (B) Silence suppression; (C) MFCC extraction; (D) Voicing; (E) Local inference; (F) MFCC transmission to the server; (G) Model reception from the server; (H) Model transmission to neighbors; (I) Local inference broadcast; (L) Local inference reception.

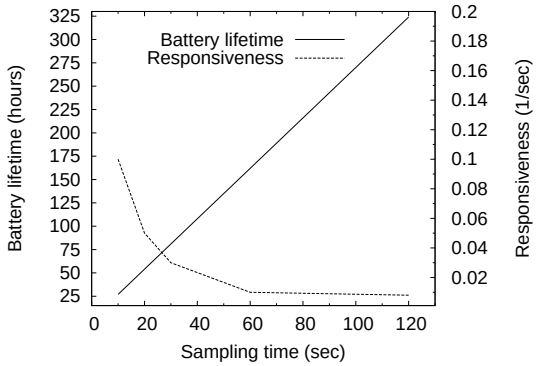


Figure 3.22: Battery lifetime Vs inference responsiveness.

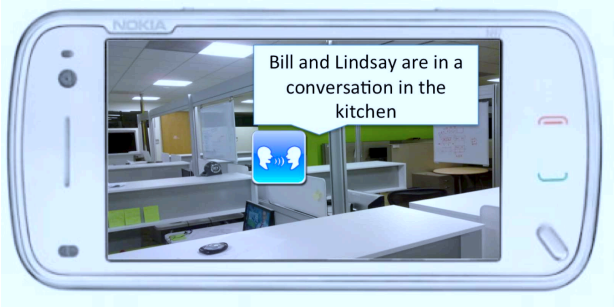


Figure 3.23: Virtual Square, an augmented reality application on the N97 with the support of Darwin.

place multiple times in the future.

- **Pooling:** when a mobile phone visits an area for which it does not have an RF profile it has two options: either build a profile, which requires time and introduces delay in inference, or, pool a profile for the area from a nearby mobile phone or backend server. Building an RF profile could take more time than the duration of the visit in the area, which means that the place might not be discovered. By pooling, the profile is immediately ready to be used in the inference phase.

- **Collaborative inference:** if multiple mobile phones are co-located in the area they can co-operate to perform a more accurate place inference. Given that sensed RF activity could be slightly different from phone to phone, collaborative inference could be used to determine what is the most likely discovered place by, for example, selecting the place that is reported with highest probability by each of the mobile phones.

3.6.3 Friend Tagging Application

The idea of this application is to exploit face recognition to tag friends on pictures. Namely, the application automatically associates a name to a person in the picture if the person is recognized. Darwin could improve the application in the following manner:

- **Initial training and evolution:** the initial training starts on each user's mobile phone. The mobile phone derives a model for the person's face through a training picture. Following this initial training seed, the face model for a person can evolve over time. For example, a person's face is often captured by the phone's camera (e.g., when using video conferencing) allowing the model to be refined under different conditions (varying light conditions, on the move, etc).

- **Pooling:** when friends get together their phones pool each other's face models. Person *A*'s phone does not have to derive a face model for person *B*. It pools it directly from person *B*'s phone.

- **Collaborative inference:** face detection can now be run in parallel to tag friends when taking group pictures. Imagine a group of co-located friends taking pictures of each other. Each picture could have different people and the lighting and angle of each shot could vary considerably. Co-located phones individually run their face classification algorithm and then exchange information to refine the final inference; for example, tagging the people that the local inferences returned with highest confidence.

3.7 Related Work

Work on applications and systems for sensing enabled mobile phones is growing in importance [42, 9, 146, 73, 124, 111, 5, 40, 41, 48, 14]. Most of the work in the literature, however, propose local sensing operations running on individual devices and do not exploit in-field mobile phones interactions. An exception to this is the work in [124], which considers context driven sampling, and calibration techniques for mobile sensor networks [147].

Sensor node co-operation is studied mainly in the context of static sensor networks where fusion [148, 149, 150] and aggregation [151, 152, 153] techniques are applied. The benefit of sensor nodes cooperation in the context of object tracking using distributed Kalman Filters is discussed in [154, 155]. In [148] the authors propose distributed energy efficient role assignment and [149] discusses signal processing techniques to reduce the amount of sensor data needed to detect an event, while [150] proposes the adoption of distributed average consensus in order to compensate sensing errors. In the CASA project [156] researchers adopt techniques for collaborative and adaptive sensing of the atmosphere using radar technologies. All these projects present techniques for static and not mobile sensor networks. To the best of our knowledge, there is little or no work addressing the issue of how to leverage context-sensitive mobile sensing devices such as mobile phones as proposed by Darwin. There is work on the context of using embedded sensors such as the Intel MSP [11] to infer people's activity. However, no interactions between these devices are taken into account in order to realize co-operative strategies such as those discussed in this Chapter.

Recently, techniques that leverage heterogeneous sensing devices in order to exploit external sensing modalities as a further input for classification algorithms or boosting application fidelity in mobile sensing scenarios are proposed in [79, 157]. Our work goes beyond the idea of borrowing sensor readings from other sensors since we propose collaborative inference techniques that combine with classifier evolution and model pooling.

Semi-supervised machine learning techniques are investigated for word sense disambiguation [158], to identify subjective nouns [159], or to classify emotional and non emotional dialogues [160]. However, no work studies semi-supervised learning techniques in the context of mobile sensing applications or frameworks.

Audio analysis for speaker identification is a well explored area in the literature [134, 136, 135, 127, 128, 129, 130, 137, 140]. Although we do not propose new speaker recognition techniques, we show how to build a lightweight speaker identification application capable of running on mobile phones.

In the literature, context awareness follows the definition that Weiser [17, 161] and others [162, 163] provided when introducing or evolving ideas and principles about ubiquitous computing. In that case, context awareness is intended as either the awareness of situations and conditions characterizing sensor devices surroundings or the behavior, activity, and status of the person carrying the sensors in order to provide smart ways to facilitate and explore interaction between machines and humans. Thus, context is seen as the collection of happenings around a monitored subject and the response of the subject to such those happenings. The work in [71, 20, 19, 21, 20] are examples of

how sensing systems are adopted to infer such a context and/or leverage context awareness. In some cases external sensors, i.e., not part of the mobile phone itself, are also needed [20, 71] in order to perform accurate context inference. The authors of [164] use the word context to mean location awareness and propose applications that efficiently build on top of it. A very large body of work focuses instead on the use of various sensing modalities such as accelerometer, magnetometer, gyroscope to infer a person's activities for different applications [28, 11, 27, 26, 14, 165, 166, 167]. The authors in [168] present an approach to help discover the position of the phone on a person's body. The work highlights two limitations: it uses simple heuristics derived from a small training data set to determine the classification rules, and it uses a single modality approach, i.e., the accelerometer. We instead rely on a systematic design using machine learning algorithms that are more scalable and robust than simple heuristics and consider a larger training data set from multiple positions on the body and different scenarios while using a multi-sensing modality approach.

3.8 Summary

In this Chapter we presented the design, implementation, and evaluation of the Darwin system that combines classifier evolution, model pooling, and collaborative inference for mobile sensing applications on phones. The classifier evolution method presented in this Chapter is an automated approach to updating models over time such that the classifiers are robust to the variability in sensing conditions and settings common to mobile phones. Mobile phones exchange classification models whenever the model is available from another phone, thus, allowing mobile phones to quickly expand their classification capabilities. Collaborative inference combines the classification results from multiple phones to achieve better inference accuracy and confidence. We implemented Darwin on the Nokia N97 and Apple iPhone in support of a proof-of-concept speaker recognition application. We also showed the integration of Darwin with some demo applications. Our results indicate that the performance boost offered by Darwin is capable of off-setting problems with sensing context and conditions and presents a framework for scaling classification on mobile devices. Future work will consider duty-cycling techniques for better energy conservation and studying simplified classification techniques, for example, building more computationally light GMMs for mobile phones without impacting performance. We believe the development of such a classification toolkit for mobile phones will enable new research on phones for human centered applications.

We argued that phone sensing context is a key system component for future distributed sensing applications on mobile phones. It should be designed to be accurate, robust, and low cost. We discussed our initial work on the Discovery framework that grew out of our work on the deployment of two continuous sensing applications implemented and deployed on Nokia and Apple phones. Our initial implementation and evaluation only focuses on a limited set of sensors/contexts, but looks promising and, as an idea, it has potential, when implemented in its full form, to become a core component of future mobile sensing systems.

In this chapter we have presented a mobile sensing distributed and collaborative inference frame-

work designed to address some of the issues discovered during the CenceMe deployments discussed in Chapter 2. In the following chapter we present a large-scale mobile sensing application aimed at characterizing people and places in space and time. We also discuss a preliminary approach towards techniques that could be used to provide inference label validation for mobile sensing applications running in the wild.

Chapter 4

A Large-Scale Mobile Sensing Application for People and Place Characterization

4.1 Introduction

The popularity of smartphones continues to increase, while the technological divide between these and more powerful computers diminishes. Accordingly, a new paradigm is becoming evident: people are replacing their personal computers with smartphones. The mobility and power afforded by smartphones allows users to interface more directly and continuously with computers than ever before. At this writing, smartphones control a 30% stake in the US market [169] and have an accelerating growth trend worldwide. As such, the global density of smartphones will provide ground breaking means to characterize people and their communities, as well as their utilization of spaces. These possibilities are facilitated by large-scale distribution systems, increasing hardware support (battery capacity, CPU, RAM), and improving sensing capabilities.

Sensor-enabled smartphones are becoming a mainstream platform for researchers to collect information-rich data because smartphones allow the characterization of human activities and context at scale [14, 40, 41, 15]. For instance, researchers have used mobile sensors to collect measurements of pollution and audio noise levels without fixed infrastructure by mounting sensors on bikes [70]. They have exploited aggregated location traces to analyze how people move in urban environments [170, 171]. And they have used contextual data to produce improved local search results [172].

We believe that continued research in smartphone sensing will allow us to characterize people, places, and communities as never before possible. As a case example, CenceMe [14] is an application that infers a person's activity and context using multiple sensors in a mobile phone. Figure 4.1 shows CenceMe data collected in Hanover, New Hampshire over a month across 20 users. Activities such as sitting, standing, walking, and running are aggregatively represented by colored markers. We can easily examine the geographic distribution of basic human activities and reason about location relationships. For instance, the red circle in Figure 4.1 marks the Computer Science department at Dartmouth College, which is mainly characterized by the "sitting" inferred state. CenceMe's in-

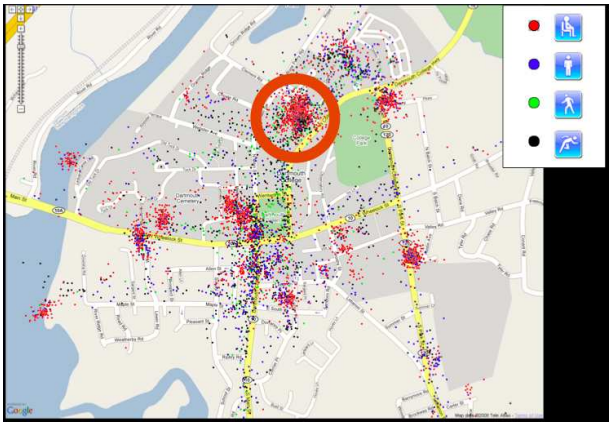


Figure 4.1: CenceMe inference labels generated over a month across 20 subjects in Hanover, New Hampshire.

ference is in accordance with the nature of a computer science department, i.e., the department is in an office building where people are mostly sitting during their work hours.

The CenceMe example helps us to understand the significance of collecting data using a continuous sensing application running on multiple smartphones. We are given the opportunity to characterize spaces at a very fine grained level, which is generally impossible without burdensome subject polling. Such information may be useful, for example, to help city managers understand how people exploit urban spaces, resulting in improved urban planning. Alternatively, physicians may learn the health behavior of a community and use this information for community health assessment and recommendations. Distributed sensor monitoring and inference can provide real-time insights, augmenting inter-person interaction, as well as interactions between people and spaces. Questions we may answer include: what music is being played at a particular club right now, how many people are at the club, and what are their demographics? Where is the quietest place in the city to read a book? How many people are jogging in the park right now, so that I won't be alone during my run today?

This Chapter presents VibN, a continuous sensing application for smartphones. The goal of VibN is to answer questions such as those posed above by collecting sensor data, executing inferences, and presenting results to users that inform them in real time about "what's going on" around them. VibN automatically provides structured visual information about the places people spend their time by displaying real-time hotspots of the city, which we call Live Points Of Interest (LPOI). We think this paradigm poses vast improvement over other models that are constrained by manual input, such as [173]. A LPOI, which is derived from a backend clustering algorithm, is represented by the demographics of its inhabitants, such as average age, ratio of men and women, and their relationship status. VibN allows its users to replay historical LPOIs, encouraging observation on how hotspots and their demographics evolve over time. In this work, we define a point of interest as any location where people spend a significant quantity of their time. Thus, places of work, living, and entertainment are points of interest.

VibN also introduces a new dimension to micro blogging through the *Vibe it!* feature, which

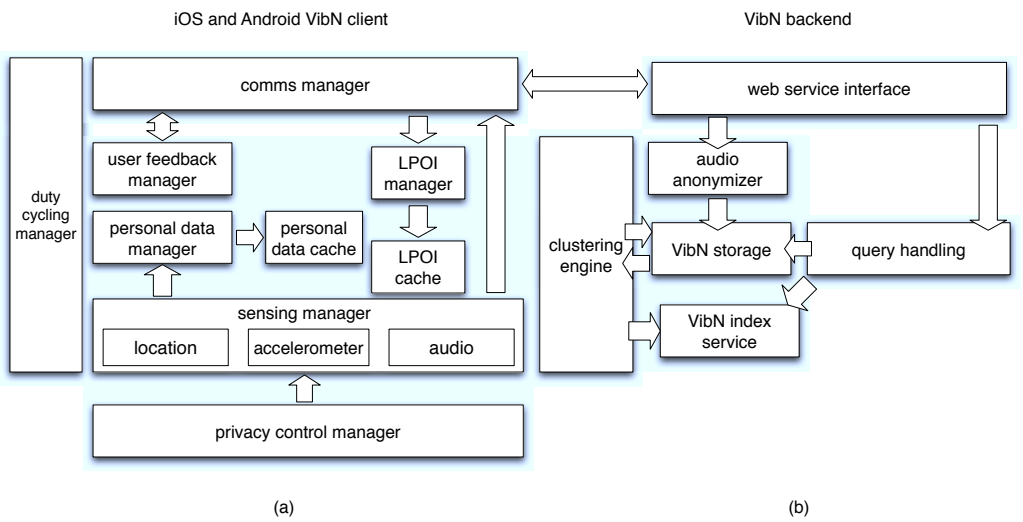


Figure 4.2: a) Architecture of the iOS and Android implementations on the phone; b) architecture of the VibN backend.

allows a user to record audio commentaries. Audio input provides richer means of building understanding about locations, activities, and events than the current short text microblogging model. However, because VibN is founded on an opportunistic sensing paradigm [4]—where the user is not an active participant in the sensing phase—it also transparently records short audio clips in the background to provide near continuous audio context. Segments of audio containing voice are filtered out from the clip to preserve the user's privacy. VibN also automatically and transparently builds a personal diary, giving the user an opportunity to track locations of significance and related audio vibes. This Chapter discusses our system design, implementation, and evaluation of VibN.

Additionally, this Chapter discusses a novel methodology for inferred label validation. Because ground truth evidence is unavailable "in the wild," we propose to exploit a multi-sensing modality approach to validating inferred labels. Validation is a key step to ensuring that the data collected in the wild by a mobile sensing application is reliable. As far as we know, our methodology has not been applied to large-scale inference application deployments. We show preliminary results demonstrating the method utility.

In summary, these are the key contributions of our work: (i) we show that VibN, by running continuously in the background on mobile phones, is able to characterize the way people and communities interact with the locations they inhabit; (ii) we present the design, implementation and evaluation of VibN, which has been deployed at large scale through the Apple App Store and the Google Android Market [174], and used by over 500 users in three weeks of operation; (iii) we present an approach for an inferred label validation methodology.

The Chapter is organized as follows. Section 4.2 presents the detailed design of the VibN application, followed in Section 4.3 by a discussion of a new method for inferred label validation. We comment of the privacy aspects of VibN in Section 4.4. Section 4.5 presents the system implementation and evaluation. We discuss related work in Section 4.6, and a summary of the Chapter in

Section 4.7.

4.2 Design

In this section, we present the design of the VibN application on the phone and of the VibN backend.

4.2.1 Phone Client

The VibN iOS architecture is shown in Figure 4.2(a). The client is modular in design and can be implemented on both the iOS and Android platforms according to the same principles of flexibility and efficiency. In the following, we describe VibN's components and design principles, followed by native differences that impact implementation.

Algorithm 3 Pseudocode for the localization engine duty-cycling manager.

```
SHORT-TIME-INTERVAL ← 5 seconds
LONG-TIME-INTERVAL ← 30 minutes
locError ← LOCALIZATION-BOUNDING-ERROR
{Application bootstrap}
previousLocation ← lastKnownLocationFromLocalDB
Line 6:
counter ← 5
while counter ≥ 1 do
    {Retrieve new location value}
    location ← newLocation
    if (location+locError ≥ previousLocation) AND (location-locError ≤ previousLocation) then
        previousLocation ← location
        upload location to server
        sleep(SHORT-TIME-INTERVAL)
    end if
    counter ← counter - 1
end while
{Next sampling scheduled after LONG-TIME-INTERVAL}
sleep(LONG-TIME-INTERVAL)
go back to Line 6
```

Sensing. We use accelerometer, audio, and localization sensor data for our application. A sensing manager activates sensors according to the directives of a duty-cycling manager. All data is sensed transparently, except for audio sensing, which can also be activated by the user. Transparent sensing occurs in the background without the user's active participation. Background audio "vibes" are introduced to periodically capture a person's context. The *Vibe it!* feature allows active participation by the user, in which they can initiate recording of a short audio clip. Every *Vibe it!* clip is geo-tagged before being uploaded to the backend server. All sensor data is handled by two components: the personal data manager, which is responsible for the personal diary points of interest; and

the communications manager, which handles bi-directional communications with the VibN server. As discussed in Section 4.2.1, the iOS and Android platforms have different methods for handling their native location and audio engines; VibN has been adapted to accommodate these differences.

Duty-Cycling Manager. The Duty-Cycling Manager orchestrates sensing-sleeping cycles in order to optimize resource usage. It is important to carefully allocate duty-cycles for mobile sensing applications in order to conserve resources, in particular battery power. We emphasize localization engine (GPS, WiFi, and cellular) regulation, the continuous use of which can dissipate a phone's battery within a few hours (refer to Section 3.3.4). VibN is not designed for continuous location tracking; its goal is to identify significant points of interest. We conjecture that people tend to spend at least 30 minutes at a time at important locations, such as the home, work, gym, restaurants, clubs, etc. We leverage this assumption and design the duty-cycling algorithm to activate the localization engine after long intervals (between 30 minutes and 1 hour) and report data to the server only if the location has remained static. In this way, we maximize the likelihood that the system captures locations that are visited for intervals longer than the sensor's sleep cycle, while ignoring places visited for short intervals.

Pseudocode for the localization duty-cycling algorithm is shown in Algorithm 3. There are two advantages to our approach: it extends the battery lifetime by applying long sleep cycles; and it promotes data pre-filtering for a server-side LPOI clustering algorithm. By scheduling localization data uploads according to visit duration, the clustering algorithm processes less data and the data more easily groups by location. Contrarily, if the application sends continuous streams of location data, the clustering algorithm would need to process data that is less structured, requiring longer convergence times and, most likely, reducing the clustering accuracy. The performance of the LPOI clustering algorithm is shown in Section 3.3.4.

The duty-cycling manager also regulates sampling intervals for background audio recording. Audio recording is randomly triggered every hour according to a coin-flipping mechanism (i.e. audio is recorded if a generated random number between 0 and 1 is greater than 0.5).

Personal Data Manager. This module manages the user's personal diary by: determining if a data point (location, or location plus audio vibe clip) is a new significant location for the user; and inserting the new data point into the personal local data cache. The personal data cache is built according to a double first-in first-out (FIFO) queueing policy. One queue contains a user's significant locations, while the other queue stores the data points for each significant place. The reason for using FIFO queues to handle personal data rather than an unbounded diary is to minimize the local storage footprint.

Since this data is only meant for personal consumption, the personal diary is not uploaded to the server and never leaves the phone. By keeping personal data local to the phone, we minimize the risk of external attackers retrieving private information. The personal data manager determines the significance of a location by analyzing the duration of a user's visit. If the visit exceeds a time threshold, then the manager flags the location as significant. We empirically use a fixed threshold of two hours, which we believe to be reasonable considering that people often visit significant locations

(a) Personal view on the iPhone.

(b) Personal details on the iPhone.

(c) Personal view on the NexusOne.

(d) Personal details on the NexusOne.

Figure 4.3: VibN personal view and personal details view on the iPhone and the NexusOne.

such as the office or home for longer periods. We realize that this policy may not generalize well to all users, since people have different living habits and styles. Future releases will give users direct control over this parameter.

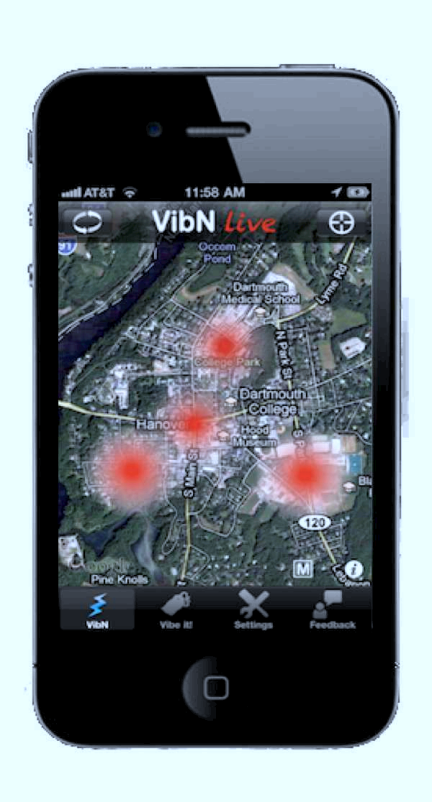
The VibN personal view is shown in Figure 4.3 for both the iOS and Android implementations. The personal view allows a user to examine their life pattern. Green blobs (Figure 4.3(a) and Figure 4.3(c)) visualize locations that the system deems significant. By tapping a green blob one can examine personal activity details, such as, the times a location was visited and the audio vibes recorded (Figure 4.3(b) and Figure 4.3(d)).

LPOI Manager. The LPOI manager maintains up-to-date live and historical points of interest on the phone and partitions them by time windows. Points of interest are refreshed in two cases: when the application is being launched; or when the application resumes from the background. Upon refreshing, the application automatically downloads points of interest co-located near the user. A bounding box is defined to be twice the span of the visible screen map's latitudinal and longitudinal scope. When the user zooms out on the map, points of interest within the new bounding box are fetched from the server.

The data associated with each point of interest is managed on the phone according to a caching mechanism. The cache is designed to strike a balance between local storage constraints and frequency of server download sessions. For each LPOI, the user can fetch up to ten audio vibes. Once downloaded, the audio vibes are stored locally and are available for future access. Besides its audio vibes, a point of interest is characterized by the demographics of its visitors. Demographic metrics include average age, average relationship status, and gender ratio. In the current implementation, demographic information is manually provided by users in the application's settings. In the future, we plan to leverage the sensors to automatically infer demographic data. For instance, voice pitch detection may be used to infer gender.

As the LPOI manager receives points of interest from the server, it partitions them according to time. A "live" bin receives points of interest derived from activity in the last hour. Historical bins are used to replay LPOI evolution over time (up to a month in the current implementation) by means of a graphical slider (see Figures 4.4(c) and 4.4(d)). The historical view allows easy identification, examination, and comparison of consistent hotspots versus transiently popular locations. Views of the live and historical points of interest, which are represented respectively by red and blue blobs, are shown for the iOS and Android implementations in Figure 4.4. By looking at a point of interest's details, we may observe how the demographics of a hotspot have changed over time. Figure 4.5 shows that there are 50% males, with a mean male age of 33, mean female age of 26, and 50% single status for a particular hotspot.

Comms Manager. The phone communicates with a server to facilitate location and audio uploads, as well as points of interest and audio downloads. We use the JSON format for data exchange, which follows a query-response pattern. The phone queries the server for information and the server responds with the requested data. Interaction with the server is driven by the sensors' duty-cycle. The phone triggers interaction only when sensor data is available, when the application



(a) Live view on the iPhone.

(b) Live view on the NexusOne.

(c) Historical view on the iPhone.

(d) Historical view on the NexusOne.

Figure 4.4: VibN live and historical views on the iPhone and the NexusOne.

Figure 4.5: One of the LPOI details on the iPhone.

is launched, or when new points of interest need to be fetched.

Privacy Control Manager. This component manages sensor access permissions according to the user's privacy settings. It supports user editing of privacy settings in the local sqlite database and grants the application access to a sensor only if the user has agreed to its use. More details on privacy management are given in Section 4.4.

User Feedback Manager. User studies, in which users are asked to report on their experience or to suggest new features, are necessary to assess the performance of a system. However, it is not always possible to collect the same quality data for large-scale deployments as for small and medium scale projects, as we have less control over compliance and it takes more time to distribute surveys over a large population. VibN's solution is the User Feedback Manager, which can dynamically survey users by presenting questions directly to the client. We are able to push down new survey questions from the server as new needs arise. Answers are uploaded to the backend providing us immediate access to important usability data.

Differences between Vibe iOS and Android

While the VibN iOS and Android implementations respect the high level architectural design guidelines of the system discussed above, these platforms present differences in some of their basic low level functions. In particular, the respective platforms handle localization, accelerometer management, and audio recording differently. These functions are dealt with separately for each platform.

Localization. The Android location engine is more flexible than the iOS counterpart. It al-

lows the programmer to individually enable localization components such as GPS, WiFi, and cellular. This makes it easier to optimize resource usage, in particularly power. Phone resources demand careful handling when designing continuous mobile sensing applications, and the individual management afforded by the Android provides increased flexibility. The iOS, however, provides less fine grained control. The programmer must specify a desired localization accuracy, which is parametrized in three levels: low, medium, high. The OS itself decides which localization components to use in order to meet the accuracy requirements. This lack of low-level control hinders thoughtful resource management by the programmer.

Accelerometer. Smartphones' sensors have been mainly introduced to enhance the user experience when interacting with the devices, e.g., flipping the user interface from landscape to portrait mode with the accelerometer. For this reason iOS currently shuts down the accelerometer when an application is pushed to run as background process since there is no active user interface that needs the accelerometer support. The consequence of this approach is the impossibility to rely on a continuous accelerometer data stream, which is the foundation for reliable activity inference. Android OS, instead, maintains the accelerometer active even when the application is sent to the background.

Audio recording. To ensure ease of portability and analysis, we opt for storing audio vibes in "wav" format. The iOS provides native routines for rendering and reading wav files. However, the Android does not. We were required to write custom Java code for building a wav file header and appending the PCM audio payload.

4.2.2 Backend

Data Collection. The VibN phone client interacts with the backend using web service interfaces supported by the Python *web.py* framework under a standard Linux distribution. The JSON format is used for data exchange. The VibN data, which we designate as "vibes," consists of the following: 1) location-only vibes; 2) audio vibes captured by the application automatically; and 3) audio vibes generated by the *Vibe it!* feature. Vibes are stored in the backend using both a binary storage service and an indexing service. The indexing service performs similarly to a distributed hash table, while the binary storage manages large binary files. The index service allows VibN to store simple metadata about media and execute queries about it. When an audio vibe is uploaded, it is indexed by the indexing service so that the vibe and its associated location can be retrieved later.

In order to preserve the privacy of users we treat automatically sampled audio vibes differently than the *Vibe it!* audio vibes. When initiating an audio recording with *Vibe it!*, a user implicitly acknowledges that data collection is taking place. However, background audio vibes are generated without user participation. For this reason, we apply an algorithm that anonymizes audio vibes automatically recorded by the phone. The algorithm removes short portions of audio from the audio stream at regular intervals so that background sounds can be identified (e.g., the sound of a car or music) but the content of conversations cannot be reconstructed. Two parameters drive the algorithm: the frequency and duration of the audio suppressions. We determine these parameters empirically with the twofold goal of preserving background sound recognition and rendering con-

versation unintelligible. This algorithm is not applied to *Vibe it!* audio vibes.

Query Handling. When fetching LPOIs, the phone client issues queries in tuple form for each LPOI. These contain a location, a bounding box representing the LPOI's region, and a time window. The time window designates the time range used for aggregating data in the historical view. There are six possible time windows, ranging from three hours to one month. By fixing window size options in advance, the clustering algorithm can asynchronously compute the points of interest for each window so that content is immediately available upon query. Given the location, the bounding box, and time window, a query is served by retrieving content from the indexing service.

Clustering Engine. The clustering algorithm runs as a background process, asynchronously to client queries, and it is based on the density-based spatial clustering (DBSCAN) technique [175]. The reason for the adoption of DBSCAN is that, by being density based, it operates in an unsupervised manner without requiring the number of clusters to be computed as input parameter like for K-Means. Clustering runs are processed on a location tile and a time window. We use location tiles of size 120 by 120 km and time periods ranging from 3 hours to as long as 1 month. Our clustering machine works on a schedule, where each entry is a specific location tile and time window combination. The output of a clustering run is a set of points of interest, stored as a record in the indexing service. This record also includes the list of audio vibes, timestamps, and demographics information associated with the point of interest.

In order to optimize the clustering process's scalability and responsiveness, computation is distributed across multiple machines; each machine operates on different sets of tiles. When computing a tile for a time range, the machine registers itself at initialization and de-registers once the task is completed. The cluster scheduling is designed to give priority to the most recent points of interest, i.e., those within the last hour, and to tiles containing high density of vibes. In this way, we guarantee that live points of interest with high upload rates are most often re-computed.

Sometimes points of interest cannot be computed because of a scarcity of vibe uploads. This condition arises, for instance, immediately after the application launch, when there is a small user base. When data is too sparce for clustering, we rely on a bootstrapping strategy. A new process is spawned that executes a request to Yelp or Bing to provide supplementary points of interest. In our current implementation, we use the Bing service, since it has a less constrained policy about the rate of queries it accepts.

Scaling. To handle scale and guarantee backend robustness, we use Amazon Elastic Cloud services. All the components reported in Figure 4.2 run on a single machine, except for the clustering process, which is distributed across multiple machines. The advantage of the elastic cloud service is that machines can be promptly instantiated or terminated based upon demand. This is a desirable feature when the user base changes over time and rapid adjustments to the backend might be needed to accommodate the application's demand.

Table 4.1: Examples of microblog posts and sensor data from the App Store CenceMe data set.

CenceMe Microblog Post	Sensor Inference
At work	Sitting
Reading at the harbor	Sitting
Out of work, going back home	Walking

4.3 Data Validation Methodology

Mobile sensing applications deployed at large scale provide a mechanism for collecting tremendous amounts of interesting and potentially useful data. However, when an application is deployed in the wild, there is no inherent method for the developer to verify whether inferences are meaningful and correct. For instance, if a mobile device reports that a person's activity is walking, we lack ground truth, or supporting evidence that the person is actually walking. Rather, an inference may be the result of a false positive misclassification. While erroneous classification may be tolerated in leisure applications, it may not be acceptable for more critical applications, such as those that assess wellbeing or health. Yet establishing ground truth can be costly; how can accuracy be verified without burdensome polling of its users? We propose a technique to boosting sensing inference accuracy and trustworthiness by mining multimedia content (such as microblog posts, public point of interest repositories, videos, photos, or audio clips) that have been posted temporally near to the inferred activity. We call this technique *sensor inference validation methodology*.

Microblog posts via Twitter, Facebook, MySpace, etc., are a popular form of communication. Such channels encapsulate valuable information about the context of a person. Text messages sent by phone may also be mined for bolstering inference. Geo-temporal information can be exploited to hint at the nature of a person's concurrent activity. Why not exploit rich coexisting data to buttress sensing inference? We may seek correlation between multimedia/textual content posted by a person and the actual activity sensed by the phone. For example, if a mobile phone's sensors report "sitting" accordant data, the phone might actually be sitting on a table while the person is moving. However, if by mining the person's last microblog message we learn that the person is "watching TV," we achieve higher confidence that the person is indeed stationary. Similarly, if a person is reported as "running" while they are microblogging about going to the gym, we can reason that the person is probably engaged in the activity of running.

An example of correlating microblog posts and inferred activity is shown in Table 4.1. The data has been culled from the CenceMe [14] data set; the microblog messages have been posted from within the CenceMe application. From this example we can see that it is possible to draw reasonable correlations between user-driven input and an inferred activity by using time-based accelerometer sensor data. Being at work or reading a book usually implies little motion. In the same example, a user is inferred to be walking when they have posted that they are returning home. In these cases we are given the opportunity to attach confidence to inferred labels by correlating sensed data with microblog posts.

Table 4.2: Inferred states, based upon combinations of sensing modalities.

Inferred State	Activity Classifier	Audio Classifier	Location Change Classifier
Street: Walking	Walking	Street Sound	Low
Street: Standing	Stationary	Street Sound	0
Street: Running	Running	Street Sound	Low
Office or home: Standing	Stationary	Quiet	0
Restaurant: Walking	Walking	Noisy Business	Low

The basis of our validation methodology is that we exploit semantic keywords and characteristic features of multimedia content, location, and sensor data. Using this approach, we can identify, flag, and discard inference labels that might lead to inaccurate conclusions; detect deliberate attempts to perturb automatic inferences; and refine inferences due to questionable sensor data.

Although the range of possible human activities is high, the set of possible inferred states is constrained to those most commonly found in daily routines, as well as the range of classifier outputs available from the sensed data. Our methodology operates on three levels to validate an inferred state. First, on labels inferred by sensor-based classifiers; second, on low level sensor feature vectors; and third on microblog text. On the first level, inferred sensor-based labels are exploited to support cross label validation. On the second level, we detect differences between a user's sensor data feature vector and a training feature vector. Finally, we look for semantic matches between microblogs and location metadata. The validator's output is a set of inferred state weightings, where a weight is associated with a confidence level for each possible inferred state.

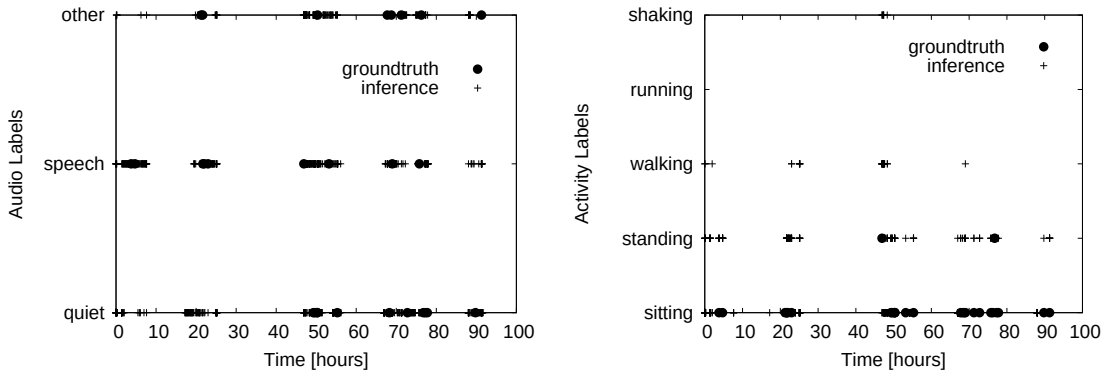
As an example, we use the CenceMe application [14] data. We have three classifiers that output labels. The activity classifier operates on the accelerometer data and returns the following labels: stationary, walking, running. An audio classifier recognizes acoustic contexts such as an indoor office or home, street sound, or noisy business. Finally, the location change classifier labels how quickly a person's geospatial location is changing. In our example, we work with a small set of inferred states, some of which are listed in the left hand column of Table 4.2. However, this model may generalized to a larger inferred state space.

A confidence-weight associated with a state is calculated using Formula 4.1:

$$ConfidWeight = \frac{\sum_{i=1}^{N} B_i + \sum_{j=1}^{Z} F_j + M}{N + Z + 1}, \tag{4.1}$$

where N is the number of label classifiers and Z is the number of sensor feature vectors. In Table 4.2, there are 3 label classifiers, listed in the header for the three right hand columns. B, F, and M are boolean, such that B_i is 1 if the i-th classifier is evaluated to true with respect to an inferred state. F_j is equal to 1 if the Euclidian distance between the sensed feature vector and the training feature vector for an inferred state is below an arbitrary threshold θ. M is equal to 1 if there exists a match between any of the keywords in a person's microblog message and the keywords extracted from the Geographic Information System (GIS) metadata or a local search on the location.

The validation methodology is designed to attribute lower confidence to an inferred state when individual classifiers are well-aligned but the underlying input space is not. For example, assume



(a) Audio inference labels and ground truth.

(b) Activity inference labels and ground truth.

Figure 4.6: Sparse ground truth and inferred labels over time. It might be possible to build models to predict inferred labels from sparse ground truth events.

that we intend to validate whether a person's activity classifier is accurate. We can collect data from an audio classifier, a location change classifier, and microblog text. The phone's activity classifier reports "running." Yet the other data streams indicate a restaurant scenario: an audio classifier indicates a noisy business; there is no significant spatial movement (as indicated by GPS), and the user has used the word "pizza" in a microblog. Each of the classifiers are within θ Euclidian distance from the training data. Given the restaurant scenario, "running" would receive a confidence of $\frac{6}{7}$, while "not running" would have a confidence of $\frac{7}{7}$=1.

4.3.1 Future Work for Data Validation Methodology

More research is needed to develop a complete and effective validation methodology solution. We are planning to introduce two techniques, designed to work in concert as part of the data validation methodology.

The first technique is about exploiting the correlation between events that occur near in time, given that human activities and behaviors tend to cluster within a certain time interval. For example, if a person is sitting at the office, it is likely that they have been sitting for some time already and they will persist in the same state for some time in the future. Similarly, if a person is jogging, the same activity can last for a certain amount of time. Our idea is to seek correlations between ground truth hints, sparsely collected from users, and the classifiers' inferred labels for past and future time intervals. To better understand this approach, in Figure 4.6 we report some results from an experiment we conducted in campus with the VibN application and 20 participants over a period of three weeks. Each user has been randomly prompted during the day (at most 10 times) to report their actual activities, context, and location. The goal is to see if we can exploit a ground truth event to guess the user's activity and context in a time interval centered around the ground truth event itself.

Figure 4.6 shows over time the ground truth events provided by one of the users and the output of the classifiers. From Figure 4.6 we can see that a given ground truth data point tends to overlap with

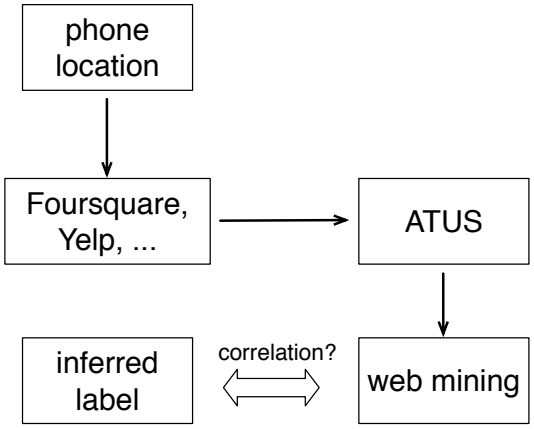
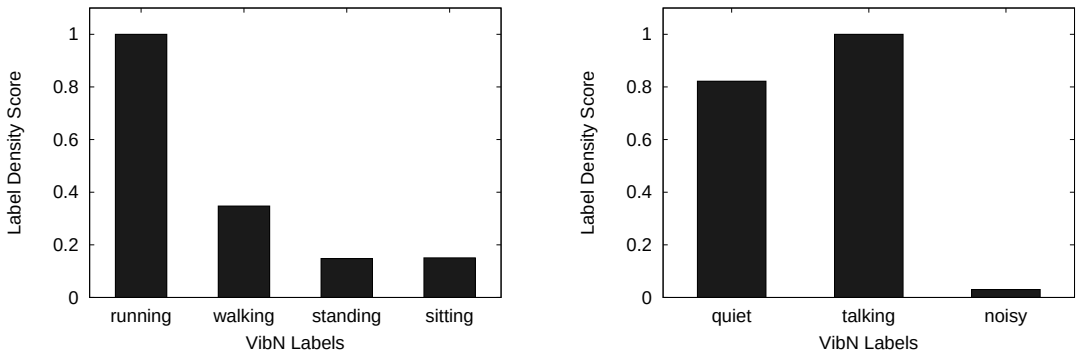


Figure 4.7: Inferred labels validation flow involving external sources of information such as public points of interest, expected behavior public database, and web mining techniques.

the classifier labels for time intervals centered around the ground truth report. This preliminary experiment provides a promising indication for the possibility to exploiting sparse ground truth labels as a way to validate inferred labels in a time interval centered around the ground truth timestamp. Thus, we hope to be able to design models that validate inferred labels for which we do not have ground truth evidence. In order to accomplish our goal, we will build on previous work that models physical activity inference using wearable sensors [176]. There is another challenge, which is to find ways to collect ground truth labels from users without possibly interrupting daily life routines. Prompting a person too often for ground truth collection would imply poor user experience, with the result of having any application designed for data validation methodology rarely used or not used at all. A successful data validation methodology should be able to collect ground truth labels while at the same time engaging a user for long periods of time. To this end, we are planning to leverage some of the Von Ahn's invisible computing principles [177].

The other approach we are planning to introduce as part of our validation methodology is taking advantage of the large corpus of information from third-party sources as a means to build stronger confidence about a certain inferred label. We plan to move beyond the preliminary approach presented in the previous section, where we take advantage of prior knowledge for a place (represented in Table 4.2) in order to verify if a certain activity is likely to happen in that place. The idea is to adopt a more general approach through which validating inferred labels without possibly the need to specify prior knowledge manually (as in Table 4.2). In order to do so, we rely on the algorithm represented by the flow in Figure 4.7. Starting from the latitude and longitude coordinates associated with an inference label, we can exploit off-the-shelf public point of interest databases to retrieve the category of the place. Examples of categories are restaurant, museum, gym, coffee store, etc. This information can be obtained exploiting publicly available programming interfaces of popular point of interest or local search providers such as Foursquare, Yelp, Google, and Microsoft Bing. After a category for a given place is retrieved, we can determine the most likely activities and behaviors that are going to be observed in that place. To retrieve the list of activities associated with a place



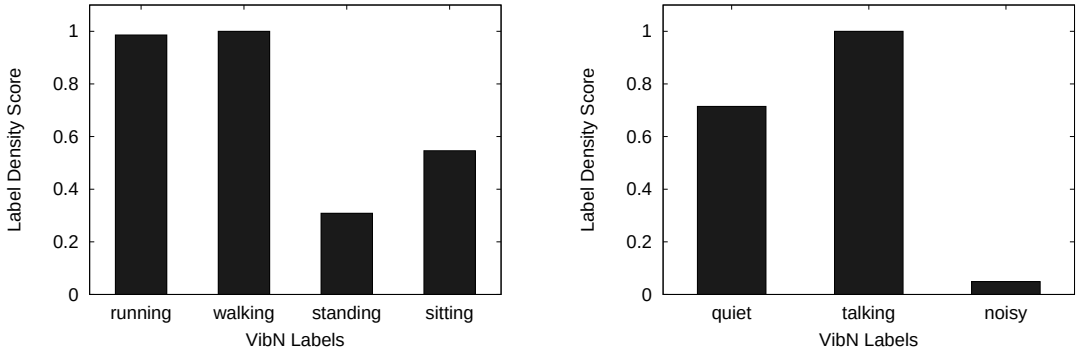
(a) VibN activity inference label distribution in the web documents for the gym category.

(b) VibN audio inference label distribution in the web documents for the gym category.

Figure 4.8: VibN inferred labels distribution from the web documents for physical activity and audio context for the gym category.

category we can rely, for example, on the American Time Use Survey (ATUS), an initiative of the US Bureau of Labor Statistics to keep track of the amount of time people spend in their own activities and where [178]. From ATUS it is possible to obtain a list of activities that people usually perform in each place category. Having retrieved the list of people's activities and behaviors for a certain place, we then exploit content-rich web documents to find correlations between our inferred labels and the content of these documents using a similar approach as in [179]. We mine the web to obtain information about how the activities retrieved from ATUS are performed and we use this information as a form of ground truth. For example, if the place category is "restaurant" and from ATUS the most likely associated activity is "eat a meal", our web query would be: "how to eat a meal". In response to this query we download 100 web pages identified by the first 100 hits. This threshold has been chosen meeting the requirement to collect as much information as possible while still being able to analyze the corpus of documents reasonably fast. We then try to see if our inference labels occur in the web documents we have downloaded.

The larger the correlation between the inference labels and the content of the web documents, the more confident we can be about the correctness of the inferred label because from prior knowledge, i.e., the ATUS activity list, we obtain that the activity or behavior represented by our inferred labels can be found in the web documents that describe that activity or behavior. Conversely, if it is not possible to find correlations between the inferred labels and the content of the web documents, a smaller confidence should be associated with an inference label. The web documents are retrieved issuing "how to" queries for each activity identified in the ATUS list for a certain place category. As the web documents have been downloaded, we need to find correlations between the inferred label words and the content of the web documents. Examples of inferred labels are "running", "walking", "sitting", or "talking". In order to maximize the chance of finding our inferred label words inside the web documents we rely on a stemming algorithm step [180] applied to our inferred label words. In this way, all the words with the same stem are reduced to a common form. For example, the word "talking", after the stemming phase can be considered equivalent to the word "talk" and "talked".



(a) VibN activity inference label distribution in the web documents for the subway category.

(b) VibN audio inference label distribution in the web documents for the subway category.

Figure 4.9: VibN inferred labels distribution from the web documents for physical activity and audio context for the subway category.

At this point, we minimize the chance of missing the word talking and all the variations of it in the web document. Thus, if we aim to validate the inferred label talking, not only do we look for the word talking in the corpus of web documents we have downloaded, but also for all the variations of the word talking derived by the stemming algorithm. In order to determine the distribution of our inferred labels in the web documents, we also look for the other inferred label words in the same corpus. Following this string matching step, we build a distribution of the most found inferred label words in the document. If, for example, talking is the most present inferred label word for a certain place category, then we conclude that there is a high chance that talking has been correctly inferred by the mobile sensing application, given the fact that prior knowledge from ATUS contemplates activities for which talking is predominant.

Figure 4.8 and Figure 4.9 show the distribution of the VibN inferred labels for two different place categories: gym and subway. These distributions have been obtained from mining 100 web documents by querying the web with "how to $<X_i>$" queries, where $<X_i>$ is one of the activities taking place in each location, and there is N of such activities in a location according to ATUS. The web document analisys is performed by looking for the stemmed VibN inferred labels into these documents. If we consider the top two most recurring labels, we can see in Figure 4.8 that running and walking, along with talking and being quiet, are the most recurring activities at the gym. For the subway, the most present activities are walking, running, talking, and being quiet as shown in Figure 4.9. We consider this as a promising initial result. In fact, we are given the opportunity to validate inferred labels from applications running in the wild looking at the presence of our inferred labels in web documents obtained from the "how to" queries. If, for example, we receive a walking or running activity label from a mobile sensing application running in the wild, and the person is located at the gym, from Figure 4.8(a) we could asses with some confidence that the label is accurate. Similarly, if we receive the talking and quiet labels, we can again associate high confidence with these labels by looking at Figure 4.8(b).

Our plan is to carry out a more systematic study of the approach presented in this section to

make sure to develop a robust and scalable data validation methodology that can be extended to a large body of activities and place categories.

4.4 Security, Privacy, and Trust

Security, privacy, and trust are important matters for mobile sensing applications. As such, VibN attempts to ensure a secure and trustworthy system with the following steps: personal diary data never leaves the phone and the user has full control over it; uploaded data is stripped of any details that could reveal a person's identity; details on live points of interest are an aggregate representation of a location without exposing any individual's information; data sent and received over the wireless link is protected by SSL encryption; users can disable the sensors at any time; background audio recordings are automatically stripped of vocal content in order to preserve conversational confidentiality.

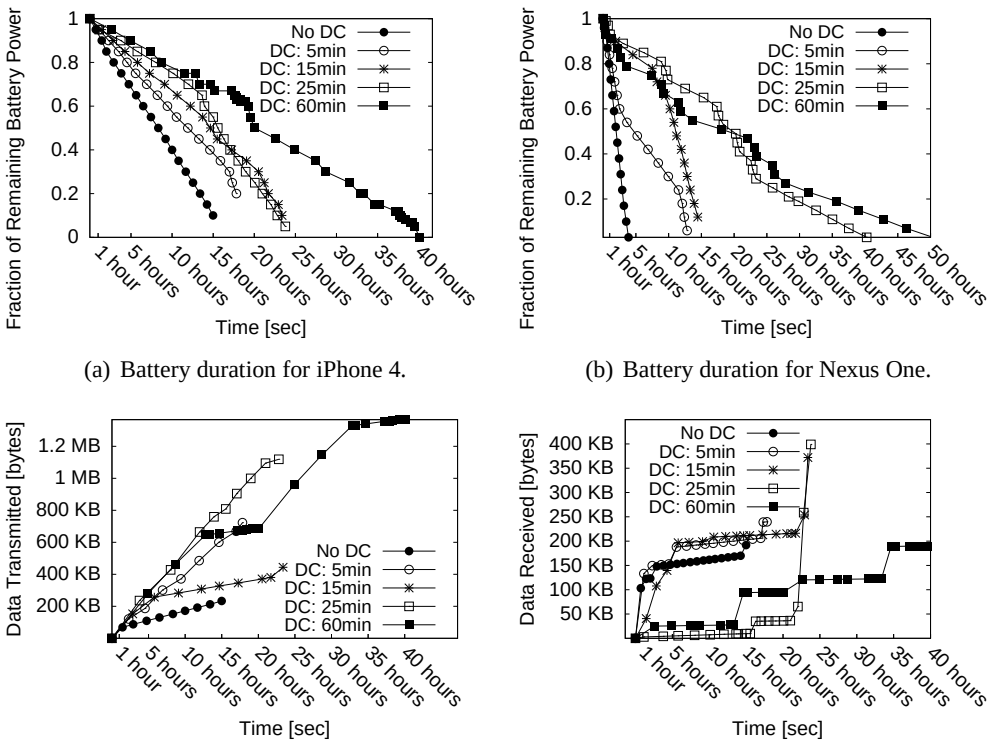
Future development of this work will implement speech-to-text translation mechanisms in order for a user to post audio vibes in a privacy-preserving manner. We also plan to implement location-driven privacy control, where the system can be told to disable sensors automatically in specific locations. For example, the home could be designated a no-sensing area, so that the system would automatically stop sensing as a person enters their home.

4.5 VibN System Evaluation

VibN is implemented on iOS and Android platforms, and it is able to run on Apple iPhone/iPod Touch devices, as well as on multiple Android phones. The iOS implementation consists of about 21,000 lines of Objective-C/C code versus 10,700 lines of Java code for the Android. The application was released to the public through the Apple App Store and Android Market on November 2010. In approximately 3 weeks, 500+ users have downloaded and used the application continuously. In this section, we present a system characterization of VibN, and a characterization derived from the large data set collected from the app store users. As far as we know, this is the first characterization of a mobile sensing application released at large scale through the app stores.

4.5.1 System Performance

In what follows, we show the performance of the iPhone and Android VibN application when running, respectively, on iPhone 4 and Nexus One devices. Since the location engine on these devices is the main cause of battery drain, we focus on the battery life as a function of the localization duty-cycle (refer to Figures 4.10(a) and 4.10(b)). From our experiments, we derive the optimal location engine duty-cycle time to be 30 minutes. After several weeks of application use, we determined this to be the interval that minimizes battery usage while collecting significant points of interest. With a 30 minute sleep cycle, the iPhone 4 battery duration is 25 hours, versus 40 hours on the Nexus One. The reason for longer battery duration on the Nexus One is that Android provides native APIs



(a) Battery duration for iPhone 4.

(b) Battery duration for Nexus One.

(c) Tx data.

(d) Rx data.

Figure 4.10: iPhone 4 and Nexus One battery duration when running VibN and amount of data received and transmitted during VibN operations.

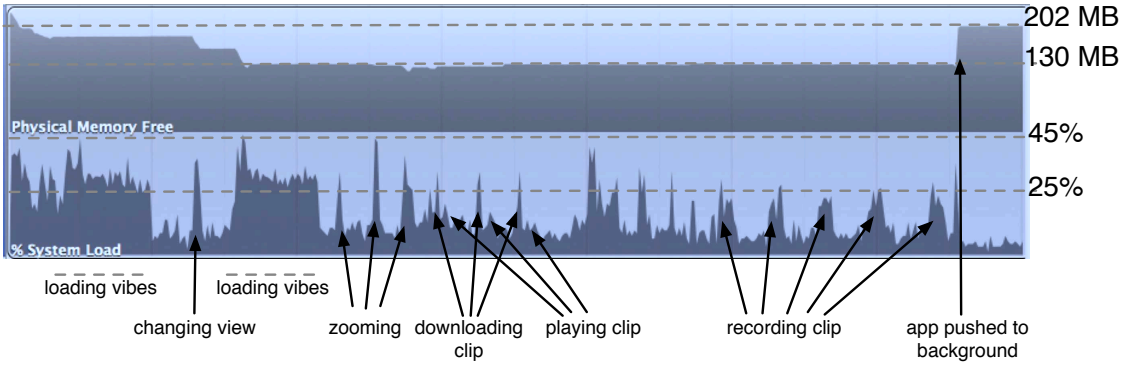


Figure 4.11: CPU usage and free memory for VibN running on iOS.

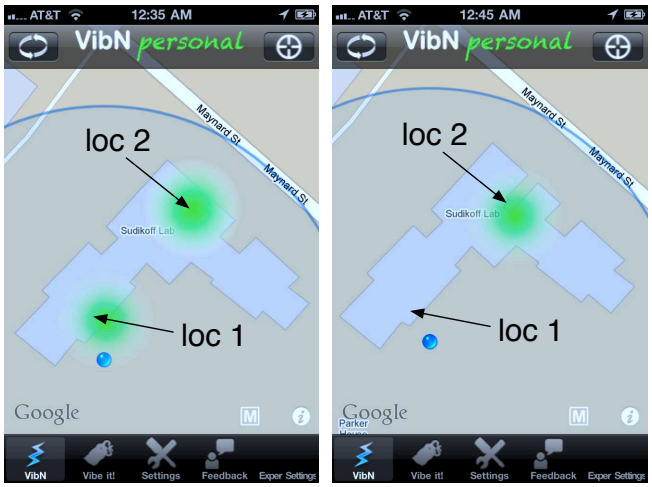


Figure 4.12: Personal points of interest for an indoor location with an iPhone 4. The dampening region radius is: a) 11m and b) 27m.

to actively regulate the localization components. This gives the developer flexibility to build more power-efficient applications. The total amount of data transmitted and received by the iPhone 4 for different localization duty-cycles is shown in Figure 4.10(c) and Figure 4.10(d) (the Nexus One reports similar numbers).

The CPU load and memory footprints for the VibN components on the iPhone 4 are reported in Figure 4.11. Memory utilization is not impacted by the sensors, except for the map view and LPOIs on the map. However, the impact on CPU load from all components is evident. Uploading and downloading data, and playing audio clips require considerable CPU cycles, and imply large battery drain. These results call for careful use of the audio modality in a mobile sensing application. Our implementation is such that memory gets released as the application is pushed to the background (see Figure 4.11). In this way, the performance of the phone is not impacted when VibN runs in the background, and other applications can be activated concurrently.

4.5.2 Personal Points of Interest

Personal points of interest are generated in two different ways: when the application runs in the background, and when a person records a *Vibe it!* audio clip. In both cases, given the localization error (which is larger indoors), we have to ensure that the system does not create different points of interest for the same physical location. In order to achieve this goal, a dampening scheme is required. VibN accepts new points of interest only if they lie outside a bounding box centered on the person's location. The bounding box must be dimensioned properly so that significant places are generated when the person moves to nearby locations and a new significant place is warranted. We evaluated the accuracy of point of interest placement in indoor locations for different dampening box sizes, ranging in radius from 11 to 60 meters. The result for two indoor locations in adjacent buildings is shown in Figure 4.12. When the dampening region has a diameter of about 30 meters

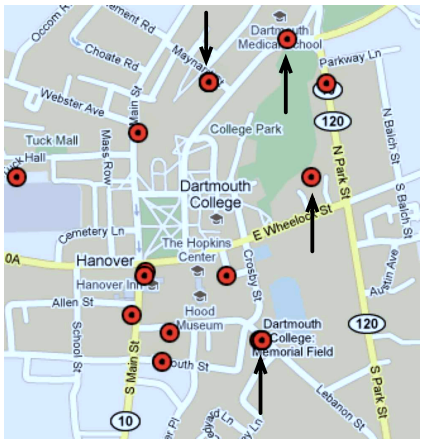


Figure 4.13: Spurious clusters caused by continuous location data upload.

and the user moves from location 2 to location 1 in an adjacent building, the significant point of interest for location 1 is not captured by VibN because it is within the dampening region (see Figure 4.12(b)). We therefore set the dampening region radius to be 11 m so that the two locations can be distinguished (as shown in Figure 4.12(a)). We found this value to be robust across multiple indoor locations in different locations. We are planning to introduce an adaptive dampening policy based on the localization error in the future.

4.5.3 Backend Clustering

In this section we discuss the performance of the clustering algorithm running in the backend to compute LPOIs. We rely on the DBSCAN clustering algorithm [175], which takes two parameters: the scope of the clustering (eps) and the minimum number of data points (k) necessary to form a cluster. The algorithm's performance as a function of several parameter values is shown in Figure 4.14. The raw vibe locations, uploaded from seven different locations, are reported in Figure 4.14(a). After several experiments, we pick k=5 and eps=0.002, which lead to better clustering accuracy while minimizing false positives. In fact, when a location is significant, several data points can be found for that place. We fix the minimum threshold to 5 data points to dampen the impact of false positives.

We also show the positive impact of the phone data pre-filtering algorithm discussed in Section 3.3.2. Continuous data uploads would generate spurious clusters, which are the result of false positives generated by sparse location data. Figure 4.13 shows the results of multiple phones continuously uploading data. Data uploads occur at intervals of less than a minute along a path which includes the locations reported in Figure 4.14(a), and stop for significant times in the same locations. The density of the vibes is larger in the original seven locations and lower along the path. It can be seen from Figure 4.13 that the sparse data along the path, which does not represent points of interest, can create spurious clusters. Only four of the original LPOIs can be identified (indicated by arrows in Figure 4.13). Hence the pre-filtering approach on the phone boosts the clustering performance.

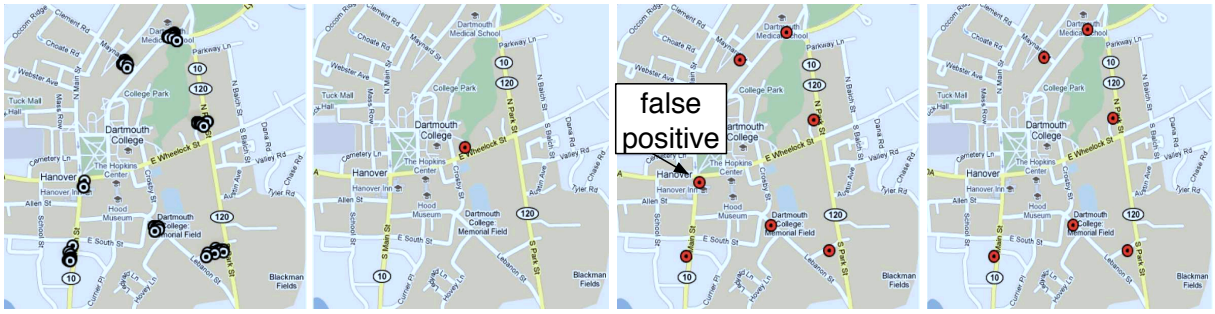


Figure 4.14: Backend clustering algorithm performance: a) raw location data from seven different places; b) result of the clustering algorithm with k=1 and eps=0.1; c) result of the clustering algorithm with k=1 and eps=0.02; d) result of the clustering algorithm with k=5 and eps=0.002

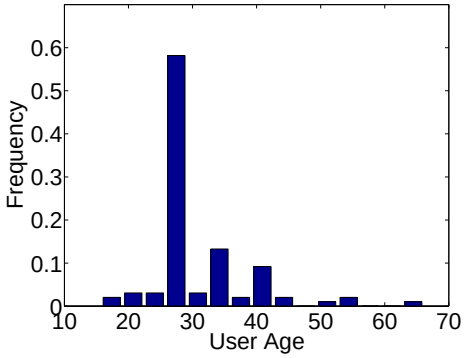


Figure 4.15: VibN users' age distribution.

4.5.4 VibN Usage Characterization

We report the application usage by analyzing data collected from the App Store and Android Market users which, as of the time of writing, is numbered at more than 500.

Demographics. The VibN users demographics characterization is shown in Figures 4.15, 4.16, and 4.17 for age, gender, and relationship status, respectively. It is interesting to observe that the average VibN users' age is below 30 (Figure 4.15).

This data supports the conjecture that the main consumers of mobile social applications from app stores are young. There is a slight bias towards female (Figure 4.16) and single (Figure 4.17) users. These measurements could be used as an indicator that mobile social network users are mainly young, single, with consistent female participation. This information could be used by developers as a hint to select the target population of their applications.

Device breakdown. The fraction of Android versus iOS users is shown in Figure 4.18. It is interesting to see that the number of Android users is larger than the number of iOS users. We believe the reason is that Android OS is supported by many different smartphone models compared to iOS, available only for Apple smartphones. With its more flexible programming platform and absence of a review process for the release of an application on the Android Market, Android becomes a very

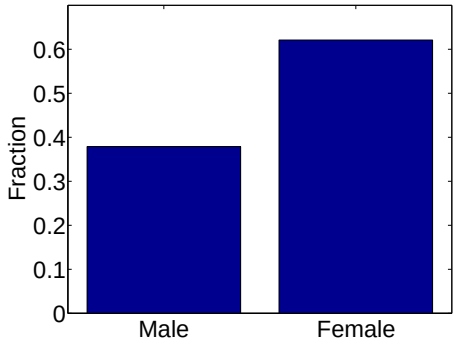


Figure 4.16: VibN users' gender distribution.

Figure 4.17: Single Vs not single breakdown distribution of VibN users.

appealing platform for researchers to quickly roll out mobile sensing applications at scale.

Usage pattern. The daily and weekly usage patterns of the VibN application are reported, respectively, in Figures 4.19 and 4.20.

It is important to identify the application usage pattern in order to design a system that is flexible enough to be responsive when necessary, for example, to handle bursts of users. In particular, by knowing when users are mostly active, we design the VibN backend in order to: instantiate more machines to accommodate high loads during day, and make the clustering algorithm more responsive during peak hours. This scheduling policy allows resource saving, while driving down the cost of renting computing power from external cloud services, e.g., Amazon cloud service.

Privacy Settings. In order to use data for research purposes, it is necessary to comply with the directives of the Institutional Review Board (IRB) university committee, which requires users to be informed if their data is going to be used for academic research. To this end, we add an informative text following the terms of service when the application is downloaded asking the user whether they

Table 4.3: Fraction of users allowing their data to be used for research purposes.

Participating	Not participating
25%	75%

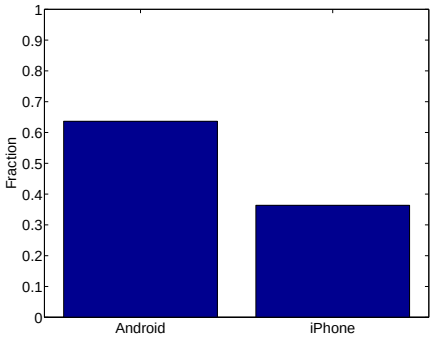


Figure 4.18: Fraction of Android Vs iOS users.

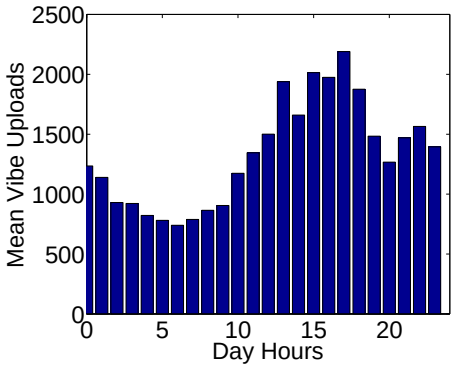


Figure 4.19: VibN daily usage pattern.

would like to participate. The breakdown of voluntary user participation versus non participation is reported in Table 4.3. These numbers point out an important lesson: it is still unusual for people to download research-oriented applications from commercial app store distribution channels. Thus, by not fully understanding the mechanisms and the risks involved, people simply opt-out from participating. Convincing people to participate to the use of research applications remains a challenge, causing the slow down of the user-base growth and of the data collection process.

4.5.5 Data Validation

In this section we present experimental results for the data validation methodology discussed in Section 4.3. We combine microblog posts, i.e., from facebook or twitter, with location data, audio, and activity inference. This helps correct localization error and detect activity misclassification errors associated with data coming from the wild. Researchers have shown that the combination of audio and video/image inference could be used to fingerprint and identify places [41].

However, defining scalable classifiers for each possible location based on a pure fingerprinting mechanism might not scale, nor produce accurate classification. To this end, we add an extra modality, which is the microblog post text, as a further way to refine the classification process. By comparing geo-tagged reviews from popular review services such as Yelp, or geo-tagged microblog

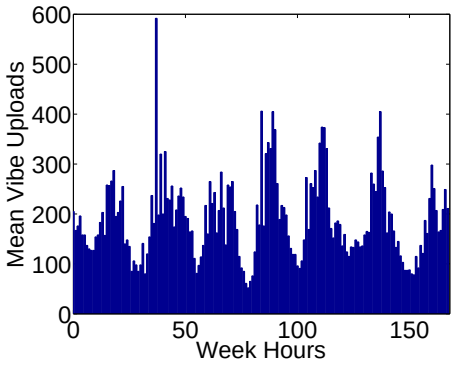


Figure 4.20: VibN weekly usage pattern.

Figure 4.21: Localization error compared to the real indoor pizza restaurant location.

posts (e.g., from twitter) with local search results or GIS databases, we retrieve the most occurring common keywords. If the found keyword set is not empty, then we have high confidence about the location where the activity is taking place. Issues might occur for nearby similar business (e.g., two pizza places few meters apart). In such a case, tie breaking techniques are probably needed in order to determine where the activity is really coming from. We reserve this to future work investigation. To evaluate the idea, we carry out experiments with people in several locations. We report the results from one of these experiments where three people go to a pizza restaurant at different times of the day. The actual location of the restaurant and the localization error from their iPhone and Nexus One are reported in Figure 4.21. The participants are asked to write a twitter message or a Yelp review when at the location. The keywords distribution extracted from the messages is reported in Figure 4.22. The keywords that are common to the microblog posts and the Yelp review or local search results are: “pizza” and “restaurant”. Since there is not any other restaurant in the proximity of the pizza restaurant, our method allows us to translate the wrong location to the real one.

As discussed in Section 4.3, we rely on Formula 4.1 to weigh classification labels. We adopt the Euclidean distance metric to measure, for a certain modality, the distance between the feature vector of newly sampled data with the training data feature vector. From the euclidean distance, we can determine whether a match exists or not by simply using a thresholding approach. If there is a match between the i-th feature vector for modality i and the feature vector expected to be seen for modality i in the same place, then the F_i parameter in Formula 4.1 is set to 1. Figure 4.23 shows the

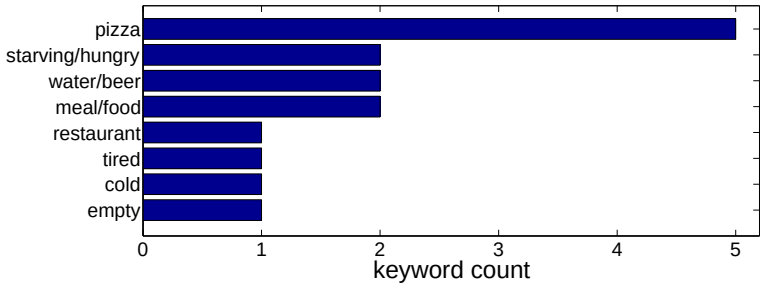


Figure 4.22: Keyword count extracted from the microblog messages posted by different people from within the restaurant.

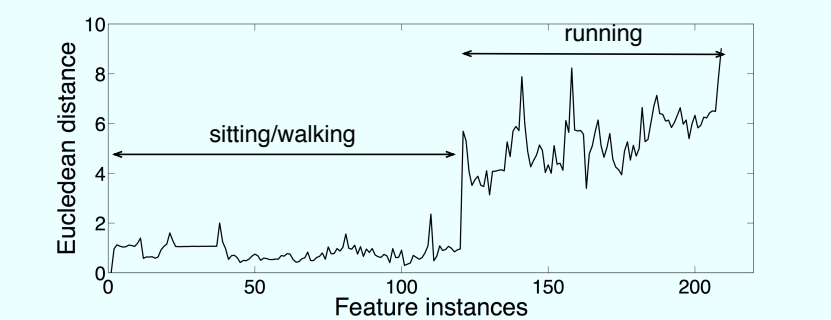


Figure 4.23: Euclidean distance between feature vectors for different activities, i.e., between sitting/walking and running.

euclidean distance between the stationary/walking activities and the running activities. It is clear that it is possible to identify thresholds discriminating between the two states. Now, assume we are in an indoor location (home, office, restaurant, etc.). If a person tries to deliberately game the system (for example by rapidly shaking the phone), then the parameter $F_{activity}$ for the physical activity is set to 0. Formula 4.1's numerator decreases, consequently assigning a smaller confidence weight to the activity performed in the particular indoor location.

4.5.6 User Feedback

The feedback channel allows us to collect valuable information about how to improve the application according to users' experience. For example, users enjoy being presented anonymous and aggregate information about LPOIs. However, one common suggestion is that VibN should allow users to connect with other users, either via text or instant messaging services. Although we have not yet implemented such a feature, we realize the importance of such a service. Adding a feature that allows users to post interesting audio vibes to each other or meet other users in LPOIs would be an interesting option to offer. Several users have expressed an interest in "friending" people they find interesting, given their significant locations. We plan to integrate social networking in future revisions of the application. Other comments, such as where to place graphical components of the

interface, have been helpful to improve the VibN design.

4.6 Related Work

Smartphones are becoming a mainstream platform for realizing mobile sensing applications and frameworks at scale [41, 181, 59, 85, 182, 43, 183, 14, 15, 48, 184]. Several techniques to optimize the usage of the phone's resources for continuous mobile sensing applications have recently been presented [55, 83, 185]. Researchers consider mobile sensors a scalable way to collect sensor data without the need of fixed infrastructure by using ad-hoc sensors on moving objects such as bikes [70], or smartphones' microphone for audio noise mapping [43]. These are examples of techniques useful to derive sensor data maps of places and cities in a scalable fashion. By analyzing mobile phones' location traces, health related parameters such as exposure to pollution can be derived [42]. Optimal fuel efficient paths can also be retrieved by combining smartphones and car sensors [186]. At the same time, there is continuous growth of applications designed to promote awareness of city events [173], or as a means of socially connecting people based on location [187]. These applications are usually user input driven. More recent applications allow association of audio clips with individual twitter accounts [188]. Overall, these applications share a similar goal, which is to meet an increasing interest in gathering real-time information about places and to more efficiently take advantage of what a city has to offer. The goal of VibN is to meet the demand for real-time rich content information by exploiting continuous sensing on smartphones. We conjecture that by being able to characterize places, people, and communities at scale using the ubiquity of smartphones, while also marrying sensor data and machine inference, we can open the door to many new dimensions. We may build novel and exciting social networking applications; and we may design green, health care, and wellbeing applications; or offer new opportunities to urban planners. These are a few examples of ways to exploit mobile sensing at scale. Researchers have already started to realize the opportunity behind using large scale application distribution systems (such as app stores) to collect data who's scope reaches beyond the boundaries of a research lab [189] . A study showing how to apply a multi-modality sensing approach to correct localization error has been shown in [41]. Sensor data validation in lab setting experiments by relying on user annotated data is presented in [26][27].

4.7 Summary

In this Chapter we presented the design, implementation, and evaluation of VibN, a continuous sensing application for smartphones. We discussed the implementation of VibN for the iOS and Android platforms and showed its performance on the Apple iPhone 4 and Google Nexus One. We presented the characterization of the application from the release of VibN to the public through app stores such as the Apple App Store and the Google Android Market. We reported the characterization of the application from the usage of over 500 users using it worldwide. We showed that VibN,

by running continuously in the background on mobile phones, is able to characterize the way people and communities interact with the locations they inhabit. We also presented a novel inferred label validation methodology to weigh the inferred states from users given the lack of ground truth support for mobile sensing applications running in the wild. This is a critical issue for mobile sensing applications. We believe this contribution is an important step forward in support of data validation and data trustworthiness assessment for large-scale mobile sensing application deployments.

Chapter 5

Conclusion

5.1 Summary

Supported by advanced sensing capabilities and increasing computational resources, smartphones will become our *virtual companions*, able to learn our lives, react, and propose solutions tailored to personal behaviors and habits.

In this thesis we have taken steps towards the realization of this virtual companion vision, by proposing applications, frameworks, and algorithmic solutions. We followed a systems oriented approach, in that we relied on live testbeds based on off-the-shelf smartphones to design, implement, and test each application, framework, and system. We also released some of our mobile sensing applications, such as CenceMe and VibN, through large-scale application distribution channels such as the Apple App Store and the Google Android Market. Thus, we had the opportunity to exercise large-scale testbeds made of 1000s of people worldwide.
The contribution of this thesis can be summarized as follows.

We presented the design, implementation, evaluation, and user experiences of the CenceMe application, which represents the first system that combines the inference of individuals' sensing presence using off-the-shelf, sensor-enabled mobile phones with sharing of this information through social networking applications such as Facebook, MySpace, and Twitter. We ported for the first time, off-the-shelf machine learning algorithms to smartphone devices showing the great potential of smartphone mobile sensing for the ubiquitous and pervasive computing landscape. This is a leap forward from previous approaches, which relied on custom-designed sensing and computing platforms. We highlighted how machine learning on smartphones comes with severe costs that need to be mitigated in order to make smartphone sensing common place. Some of the costs can be identified with the need to maintain the phone user experience, in terms of battery duration and the normal mobile phones operations – that is, making and receiving phone calls, and leaving enough computation resources for smooth user interface interaction. We showed that some of the limitations imposed by a mobile sensing application can be overcome by splitting classification and computation between the smartphone and the cloud, and by identifying features that are cheap to compute, yet effective. We showed that duty-cycling sensing and inference routines can be adopted

to tradeoff the smartphone resource usage versus inference accuracy. We reported on our experience deploying CenceMe at scale through the Apple App Store distribution system.

We presented Darwin Phones, which aims to address some of the challenges discovered during the CenceMe deployments. These challenges are: (i) the sensing context, which renders some of the sensors unsuitable for some sensing tasks; (ii) mobility, which reduces the time a mobile phone can be exposed to an event to be sensed; (iii) and limited classifier scalability, because an initially trained classifier might not be able to perform well in the wild in all possible conditions. Darwin is an enabling technology for smartphone sensing that combines collaborative sensing and classification techniques to reason about human behavior and context on mobile phones. Darwin advances smartphone sensing through the deployment of efficient but sophisticated machine learning techniques specifically designed to run directly on sensor-enabled smartphones. Darwin introduces a distributed and collaborative mobile computing evolve-pool-collaborate framework. By evolving automatically from the user, a classifier can tune its performance on-the-go with limited human intervention. Classification model pooling is a technique introduced to save computational and energy resources on the phone by pooling already available classification models from either surrounding devices or from the cloud. Moreover, by pooling classification models, the inference phase can start immediately after pooling, making the inference more responsive since there is no need to train a classifier from scratch. Finally, when multiple smartphones sense the same event, they can collaborate in the inference phase to mitigate the inference errors introduced by the sensing context and mobility.

We finally presented VibN, a large scale mobile sensing application designed to run on both Apple mobile devices and Android platforms. Researchers are given an unprecedented opportunity to move their research outside research laboratories by deploying mobile sensing applications at scale through large-scale distribution platforms such as the Apple App Store and the Google Android Market. The goal of VibN is to exploit a large-scale mobile sensing application to characterize communities and places as never possible before, providing real-time contextual information about places not available through the current technology, e.g., local search. We discussed the importance of the need for a validation methodology for inferred labels collected from sensing applications running in the wild. We showed the results from our preliminary design of the validation methodology and the path toward the future development of a complete and effective solution. Data validation is a key step towards the large-scale adoption of mobile sensing technology.

The work presented in this thesis helps spearhead a new area of research on smartphone sensing and opens up new challenges in mobile computing research.

5.2 End Note

The smartphone is to the 2010s as the Internet was to the 1990s and 2000s. This is the beginning of the smartphone era and it will be even more so in the next decade. The migration from pure vocal service driven cellular terminals towards more capable and intelligent mobile devices such as

smartphones and tablets, has changed and accelerated the way we reason about mobile computing, and consume information and data. The fast-paced smartphone evolution lends itself to the idea that in the near future people will replace their laptops with what I call *smartops*, i.e., the next generation of mobile and high-computing devices integrating the features of current smartphones with the capabilities of modern laptops.

In this scenario, mobile sensing will play an even more dominant role and we will see the proliferation of new sensing applications and systems at large scale in the areas of social networks, green applications, global environmental monitoring, personal and community healthcare, sensor augmented gaming, virtual reality, and smart transportation systems.

To be front and center in pushing this vision forward, there are several research threads I would like to pursue in order to make continuous sensing on mobile phones ready for prime time. I am particularly interested in developing a complete solution for the data validation methodology, carrying on along the path of bridging different sensing modalities and harvesting data from external sources of information (e.g., GIS databases, ATUS, Foursquare, Yelp, etc.) to provide better confidence levels about the nature of an inferred label coming from the wild.

We still have a long way to go if we want to turn our smartphones and smartops into virtual companions that are able to learn our lives, propose customized suggestions, and act as intelligent personal assistants. Research is needed to conceive smart algorithms that learn and adapt transparently to a user while reacting to their actions in order to provide suggestions that facilitate productivity, change their social experience, or improve their well-being. I am interested in working in this area to help lay the groundwork towards the successful integration of virtual companions on our smartphones and smartops.

Appendix A

Refereed Publications as a Ph.D. Candidate

My refereed publications as a Ph.D. candidate are listed below, including those that are currently under review. Work in preparation and technical reports are omitted. The published work includes ideas that are indirectly related to the central theme of this thesis, including the MetroSense architecture, an early mobile sensing application (BikeNet), a short-range radio characterization for mobile sensor networks, a calibration framework for mobile sensing systems (CaliBree), opportunistic versus participatory mobile sensing studies, and a MAC protocol for sensor networks (Funneling-MAC).

A.1 Journal Publications

1. Emiliano Miluzzo, Nicholas D. Lane, Kristóf Fodor, Ronald A. Peterson, Hong Lu, Mirco Musolesi, Shane. B. Eisenman, Xiao Zheng, and Andrew T. Campbell. A Mobile Sensing Application for Sensing Presence Inference and Sharing. Submitted to *ACM Transaction on Sensor Networks*.

2. Emiliano Miluzzo, Cory T. Cornelius, Ashwin Ramaswamy, Tanzeem Choudhury, Zhigang Liu, Andrew T. Campbell. A Distributed and Collaborative Inference Framework for Smartphone Sensing Support. Submitted to *ACM Transaction on Sensor Networks*.

3. Gahng-Seop Ahn, Emiliano Miluzzo, Andrew T. Campbell, Se Gi Hong, and Francesca Cuomo. A Localized, Sink-Oriented MAC for Mitigating the Funneling Effect in Sensor Networks, Submitted to *Computer Networks*.

4. Shane Eisenman, Emiliano Miluzzo, Nicholas Lane, Ronald Peterson, Gahng Seop Ahn, and Andrew T. Campbell. BikeNet: A Mobile Sensing System for Cyclist Experience Mapping, *ACM Transactions on Sensor Networks (TOSN)*, Vol. 6, n. 1, December 2009.

A.2 Magazine Publications

5. Nicholas D. Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, Andrew T. Campbell. A Survey of Mobile Phone Sensing, In *IEEE Communications Magazine*, pp 140–150, September 2010.

6. Andrew T. Campbell, Shane B. Eisenman, Nicholas D. Lane, Emiliano Miluzzo, Ronald A. Peterson, Hong Lu, Xiao Zheng, Mirco Musolesi, Kristoóf Fodor, and Gahng-Seop Ahn, The Rise of People-Centric Sensing, In *IEEE Internet Computing: Mesh Networking*, pp 12–21, Jul/Aug 2008.

A.3 Conference and Workshop Publications

7. Emiliano Miluzzo, Michela Papandrea, Nicholas Lane, Hong Lu, Andrew T. Campbell. Pocket, Bag, Hand, etc. - Automatically Detecting Phone Context through Discovery, In *Proc. of First International Workshop on Sensing for App Phones (PhoneSense'10)*, Zurich, Switzerland, November 2, 2010.

8. Emiliano Miluzzo, Nicholas Lane, Hong Lu, Andrew T. Campbell. Research in the App Store Era: Experiences from the CenceMe App Deployment on the iPhone, In *Proc. of The First International Workshop Research in the Large: Using App Stores, Markets, and other Wide Distribution Channels in UbiComp Research*, September 26, 2010, Copenhagen, Denmark.

9. Emiliano Miluzzo, Tianyu Wang, Andrew T. Campbell. EyePhone: Activating Mobile Phones With Your Eyes, In *Proc. of The Second ACM SIGCOMM Workshop on Networking, Systems, and Applications on Mobile Handhelds (MobiHeld'10)*, New Delhi, India, August 30, 2010.

10. Emiliano Miluzzo, Cory T. Cornelius, Ashwin Ramaswamy, Tanzeem Choudhury, Zhigang Liu, Andrew T. Campbell. Darwin Phones: The Evolution of Sensing and Inference on Mobile Phones, In *Proc. of Eighth International ACM Conference on Mobile Systems, Applications, and Services (MobiSys'10)*, San Francisco, CA, USA, June 15-18, 2010.

11. Emiliano Miluzzo, James M. H. Oakley, Hong Lu, Nicholas D. Lane, Ronald A. Peterson, Andrew T. Campbell, Evaluating the iPhone as a Mobile Platform for People-Centric Sensing Applications", In *Proc. of Intl Workshop on Urban, Community, and Social Applications of Networked Sensing Systems (UrbanSense'08)*, Raleigh, NC, USA, Nov. 4, 2008.

12. Emiliano Miluzzo, Nicholas D. Lane, Kristóf Fodor, Ronald A. Peterson, Hong Lu, Mirco Musolesi, Shane. B. Eisenman, Xiao Zheng, Andrew T. Campbell, Sensing Meets Mobile Social Networks: The Design, Implementation and Evaluation of the CenceMe Application, In *Proc. of 6th ACM Conference on Embedded Networked Sensor Systems (SenSys'08)*, Raleigh, NC, USA, Nov. 5-7, 2008.

13. Andrew T. Campbell, Shane B. Eisenman, Kristóf Fodor, Nicholas D. Lane, Hong Lu, Emiliano Miluzzo, Mirco Musolesi, Ronald A. Peterson and Xiao Zheng, Transforming the Social Networking Experience with Sensing Presence from Mobile Phones, (Demo Abstract) In *Proc. of 6th ACM Conference on Embedded Networked Sensor Systems (SenSys'08)*, Raleigh, NC, USA, Nov. 5-7, 2008.

14. Emiliano Miluzzo, Nicholas D. Lane, Andrew T. Campbell, Reza Olfati-Saber, CaliBree: a Self-Calibration System for Mobile Sensor Networks, In *Proc. of International Conference on Distributed Computing in Sensor Networks (DCOSS'08)*, Santorini Island, Greece, June 11-14, 2008.

15. Mirco Musolesi, Emiliano Miluzzo, Nicholas D. Lane, Shane B. Eisenman, Tanzeem Choudhury, Andrew T. Campbell, The Second Life of a Sensor: Integrating Real-world Experience in Virtual Worlds using Mobile Phones, In *Proc. of Fifth Workshop on Embedded Networked Sensors (HotEmNets'08)*, June 2008, Charlottesville, Virginia, USA.

16. Andrew T. Campbell, Shane B. Eisenman, Kristóf Fodor, Nicholas D. Lane, Hong Lu, Emiliano Miluzzo, Mirco Musolesi, Ronald A. Peterson and Xiao Zheng, CenceMe: Injecting Sensing Presence into Social Network Applications using Mobile Phones, (Demo Abstract) In *Proc. of Ninth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc08)*, Hong Kong, May 27-30, 2008.

17. Andrew T. Campbell, Shane B. Eisenman, Kristóf Fodor, Nicholas D. Lane, Hong Lu, Emiliano Miluzzo, Mirco Musolesi, Ronald A. Peterson, Xiao Zheng, CenceMe - Injecting Sensing Presence into Social Networking Applications using Mobile Phones, (Demo abstract) In *Proc. of Ninth Workshop on Mobile Computing Systems and Applications (HotMobile'08)*, Napa Valley, CA, USA, Feb. 25-26, 2008.

18. Nicholas D. Lane, Shane B. Eisenman, Mirco Musolesi, Emiliano Miluzzo, Andrew T. Campbell, Urban Sensing Systems: Opportunistic or Participatory?, In *Proc. of Ninth Workshop on Mobile Computing Systems and Applications (HotMobile'08)*, Napa Valley, CA, USA, Feb. 25-26, 2008.

19. Emiliano Miluzzo, Xiao Zheng, Kristóf Fodor, Andrew T. Campbell, Radio Characterization of 802.15.4 and its Impact on the Design of Mobile Sensor Networks, In *Proc. of Fifth European Conference on Wireless Sensor Networks (EWSN'08)*, Bologna, Italy, Jan. 30/31 - Feb 1, 2008.

20. Emiliano Miluzzo, Nicholas D. Lane, Shane B. Eisenman, Andrew T. Campbell, CenceMe - Injecting Sensing Presence into Social Networking Applications, (Invited paper) In *Proc. of Second European Conference on Smart Sensing and Context (EuroSSC'07)*, Lake District, UK, October 23-25, 2007.

21. Shane B. Eisenman, Emiliano Miluzzo, Nicholas D. Lane, Ronald A. Peterson, Gahng-Seop Ahn, Andrew T. Campbell, The BikeNet Mobile Sensing System for Cyclist Experience Mapping, In *Proc. of Fifth ACM Conference on Embedded Networked Sensor Systems (SenSys'07)*, Sydney, Australia, Nov. 6-9, 2007.

22. Nicholas D. Lane, Shane B. Eisenman, Emiliano Miluzzo, Mirco Musolesi, Andrew T. Campbell, Urban Sensing: Opportunistic or Participatory?, In *Proc. of First Workshop Sensing on Everyday Mobile Phones in Support of Participatory Research*, Sydney, Australia, Nov. 6, 2007.

23. Emiliano Miluzzo, Nicholas D. Lane, and Andrew T. Campbell, Virtual Sensing Range, (Poster Abstract), In *Proc. of Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys'06)*, Boulder, Colorado, USA, Nov. 1-3, 2006.

24. Gahng-Seop Ahn, Emiliano Miluzzo, Andrew T. Campbell, Se Gi Hong and Francesca Cuomo, Funneling-MAC: A Localized, Sink-Oriented MAC For Boosting Fidelity in Sensor Networks, In *Proc. of Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys'06)*, Boulder, Colorado, USA, Nov. 1-3, 2006.

25. Shane B. Eisenman, Nicholas D. Lane, Emiliano Miluzzo, Ronald A. Peterson, Gahng-Seop Ahn, and Andrew T. Campbell, Metrosense Project: People-Centric Sensing at Scale. In *Proc. of First Workshop on World-Sensor-Web (WSW'06)*, Boulder, Colorado, USA, Oct. 31, 2006.

26. Gahng-Seop Ahn, Emiliano Miluzzo, Andrew T. Campbell, A Funneling-MAC for High Performance Data Collection in Sensor Networks, (Demo Abstract). In *Proc. of Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys'06)*, Boulder, Colorado, USA, Nov. 1-3, 2006.

27. Andrew T. Campbell, Shane B. Eisenman, Nicholas D. Lane, Emiliano Miluzzo, Ronald Peterson, People-Centric Urban Sensing. (Invited Paper), In *Proc. of Second ACM/IEEE Annual International Wireless Internet Conference (WICON'06)*, Boston, USA, August 2-5, 2006.

Bibliography

[1] A.L. Barabási. Scale-free networks: A decade and beyond. *Science*, (325):412–413, 2009.

[2] D. Lazer, A. Pentland, L. Adamic, S. Aral, A.L. Barabási, D. Brewer, N. Christakis, N. Contractor, J. Fowler, M. Gutmann, T. Jebara, G. King, M. Macy, D. Roy, M. Van Alstyne. Computational social science. *Science*, (323):721–724, 2009.

[3] N. Eagle and A. Pentland. Reality Mining: Sensing Complex Social Systems. *Personal and Ubiquitous Computing*, 10(4):255–268, 2006.

[4] A.T. Campbell, S.B. Eisenman, N.D. Lane, E. Miluzzo, and R.A. Peterson. People-Centric Urban Sensing. In *Proc. of the 2nd International Workshop on Wireless Internet*, page 18. ACM, 2006.

[5] A.T. Campbell, S.B. Eisenman, N.D. Lane, E. Miluzzo, R.A. Peterson, H. Lu, X. Zheng, M. Musolesi, K. Fodor, and G.S. Ahn. The Rise of People-Centric Sensing. *IEEE Internet Computing*, pages 12–21, 2008.

[6] S.B. Eisenman. People-Centric Mobile Sensing Networks. *Ph.D. thesis*, Columbia University, 2008.

[7] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M.B. Srivastava. Participatory Sensing. In *World Sensor Web Workshop*, pages 1–5, 2006.

[8] A. Krause, E. Horvitz, A. Kansal, and F. Zhao. Toward Community Sensing. In *Proc. of the 7th International Conference on Information Processing in Sensor Networks*, pages 481–492. IEEE Computer Society, 2008.

[9] T. Abdelzaher, Y. Anokwa, et al. Mobiscopes for Human Spaces. *IEEE Pervasive Computing*, pages 20–29, 2007.

[10] N.D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A.T. Campbell. A Survey of Mobile Phone Sensing. *Communications Magazine, IEEE*, 48(9):140–150, 2010.

[11] T. Choudhury, G. Borriello, S. Consolvo, D. Haehnel, B. Harrison, B. Hemingway, J. Hightower, A. LaMarca, L. LeGrand, A. Rahimi, A. Rea, P. Klasnja, K. Koscher, J.A. Landay, J. Lester, and D. Wyatt. The Mobile Sensing Platform: an Embedded Activity Recognition System. *IEEE Pervasive Computing*, pages 32–41, 2008.

[12] T.E. Starner. Wearable Computing and Contextual Awareness. *Ph.D. thesis*, MIT Media Lab, 1999.

[13] E. Miluzzo, N. Lane, S. Eisenman, and A. Campbell. CenceMe – Injecting Sensing Presence into Social Networking Applications. In *Proc. of 2nd European Conference on Smart Sensing and Context*, pages 1–28, 2007.

[14] E. Miluzzo, N.D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S.B. Eisenman, X. Zheng, and A.T. Campbell. Sensing Meets Mobile Social Networks: the Design, Implementation and Evaluation of the CenceMe Application. In *Proc. of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys'08)*, pages 337–350. ACM, 2008.

[15] E. Miluzzo, C.T. Cornelius, A. Ramaswamy, T. Choudhury, Z. Liu, and A.T. Campbell. Darwin Phones: the Evolution of Sensing and Inference on Mobile Phones. In *Proc. of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys'10)*, pages 5–20. ACM, 2010.

[16] E. Miluzzo, M. Papandrea, N.D. Lane, A.M. Sarroff, S. Giornano, and A.T. Campbell. Tapping into the Vibe of the City using VibN, a Continuous Sensing Application for Smartphones. In *Under submission*.

[17] Mark Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3:3–11, July 1999.

[18] B. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pages 85–90. IEEE, 2008.

[19] The Context Aware Cell Phone Project. http://www.media.mit.edu/wearables/mithril/phone.html.

[20] H.W. Gellersen, A. Schmidt, and M. Beigl. Multi-Sensor Context-Awareness in Mobile Devices and Smart Artifacts. *Mobile Networks and Applications*, 7(5):341–351, 2002.

[21] J.E. Bardram and N. Nørskov. A Context-Aware Patient Safety System for the Operating Room. In *Proc. of the 10th International Conference on Ubiquitous computing*, pages 272–281. ACM New York, NY, USA, 2008.

[22] R.K. Ganti, P. Jayachandran, T.F. Abdelzaher, and J.A. Stankovic. Satire: a Software Architecture for Smart Attire. In *Proc. of the 4th International Conference on Mobile Systems, Applications and Services*, pages 110–123. ACM, 2006.

[23] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *Proc. of the 5th annual ACM/IEEE International Conference on Mobile computing and networking*, pages 263–270. ACM, 1999.

[24] J.M. Kahn, R.H. Katz, and K.S.J. Pister. Next Century Challenges: Mobile Networking for Smart Dust. In *Proc. of the 5th annual ACM/IEEE International Conference on Mobile computing and networking*, pages 271–278. ACM, 1999.

[25] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: a Scalable and Robust Communication Paradigm for Sensor Networks. In *Proc. of the 6th annual International Conference on Mobile computing and networking*, pages 56–67. ACM, 2000.

[26] L. Bao and S.S. Intille. Activity Recognition from User-Annotated Acceleration Data. *Lecture Notes in Computer Science*, pages 1–17, 2004.

[27] J. Lester, T. Choudhury, and G. Borriello. A Practical Approach to Recognizing Physical Activities. *Lecture Notes in Computer Science*, 3968:1–16, 2006.

[28] B. Harrison, S. Consolvo, and T. Choudhury. Using Multi-Modal Sensing for Human Activity Modeling in the Real World. In *Handbook of Ambient Intelligence and Smart Environments, Springer Verlag*, 2009.

[29] K. Lorincz, B. Chen, G.W. Challen, A.R. Chowdhury, S. Patel, P. Bonato, and M. Welsh. Mercury: A Wearable Sensor Network Platform for High-Fidelity Motion Analysis. In *Proc. of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 183–196. ACM, 2009.

[30] V. Shnayder, B. Chen, K. Lorincz, T.R.F. Fulford-Jones, and M. Welsh. Sensor Networks for Medical Care. In *SenSys 05: Proc. of the 3rd International Conference on Embedded networked sensor systems*, pages 314–314. Citeseer, 2005.

[31] F. Karray, M. Alemzadeh, J.A. Saleh, and M.N. Arab. Human-Computer Interaction: Overview on State of the Art. *International Journal on Smart Sensing and Intelligent Systems*, 1(1):137–159, 2008.

[32] The eye mouse project. http://www.arts.ac.uk/research/eyemouse/index.htm.

[33] M. Chau and M. Betke. Real Time Eye Tracking and Blink Detection with Usb Cameras. *Boston University Computer Science Technical Report No. 2005-12*, 2005.

[34] E. Miluzzo, T. Wang, and A.T. Campbell. EyePhone: Activating Mobile Phones with your Eyes. In *Proc. of the second ACM SIGCOMM Workshop on Networking, Systems, and Applications on Mobile Handhelds*, pages 15–20. ACM, 2010.

[35] R. Honicky, E.A. Brewer, E. Paulos, and R. White. N-Smarts: Networked Suite of Mobile Atmospheric Real-Time Sensors. In *Proc. of the second ACM SIGCOMM Workshop on Networked systems for developing regions*, pages 25–30. ACM, 2008.

[36] Research in the large workshop. http://large.mobilelifecentre.org/.

[37] E. Cuervo, P. Gilbert, B. Wu, and L. Roy Cox. CrowdLab: An Architecture for Volunteer Mobile Testbeds. In *Proc. of the 3rd International Conference on COMmunication Systems and NETworkS*, 2011.

[38] E. Miluzzo, N.D. Lane, H. Lu, and A.T. Campbell. Research in the App Store Era: Experiences from the CenceMe App Deployment on the iPhone. In *Proc. of the 1st International Workshop Research in the Large*, 2010.

[39] A. Morrison, S. Reeves, D. McMillan, and M. Chalmers. Experiences of Mass Participation in Ubicomp Research. In *Proc. of the 1st International Workshop Research in the Large*, 2010.

[40] H. Lu, W. Pan, N.D. Lane, T. Choudhury, and A.T. Campbell. SoundSense: Scalable Sound Sensing for People-Centric Applications on Mobile Phones. In *Proc. of the 7th International Conference on Mobile systems, Applications, and Services*, pages 165–178. ACM, 2009.

[41] M. Azizyan, I. Constandache, and R. Roy Choudhury. Surroundsense: Mobile Phone Localization via Ambience Fingerprinting. In *Proc. of the 15th annual International Conference on Mobile Computing and Networking*, pages 261–272. ACM, 2009.

[42] M. Mun, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, R. West, and P. Boda. PEIR, the Personal Environmental Impact Report, as a Platform for Participatory Sensing Systems Research. In *Proc. of the 7th International Conference on Mobile systems, Applications, and Services*, pages 55–68. ACM, 2009.

[43] R.K. Rana, C.T. Chou, S.S. Kanhere, N. Bulusu, and W. Hu. Ear-Phone: an End-to-End Participatory Urban Noise Mapping System. In *Proc. of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 105–116. ACM, 2010.

[44] Intel - urban atmospheres. http://www.urban-atmospheres.net/.

[45] S. Consolvo, D.W. McDonald, T. Toscos, M.Y. Chen, J. Froehlich, B. Harrison, P. Klasnja, A. LaMarca, L. LeGrand, R. Libby, et al. Activity Sensing in the Wild: a Field Trial of Ubifit Garden. In *Proc. of the twenty-sixth annual SIGCHI Conference on Human factors in Computing Systems*, pages 1797–1806. ACM, 2008.

[46] M.Z. Poh, K. Kim, A.D. Goessling, N.C. Swenson, and R.W. Picard. Heartphones: Sensor Earphones and Mobile Application for Non-obtrusive Health Monitoring. In *Wearable Computers, 2009. ISWC'09. International Symposium on*, pages 153–154. IEEE, 2009.

[47] Nokia Research Center. Mobile Mixed Reality The Vision. *Nokia Research Center Insight Series*, 2009.

[48] P. Mohan, V.N. Padmanabhan, and R. Ramjee. Nericell: Rich Monitoring of Road and Traffic Conditions Using Mobile Smartphones. In *Proc. of the 6th ACM conference on Embedded Network Sensor Systems*, pages 323–336. ACM, 2008.

[49] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson. VTrack: Accurate, Energy-Aware Road Traffic Delay Estimation Using Mobile Phones. In *Proc. of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 85–98. ACM, 2009.

[50] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava. Using Mobile Phones to Determine Transportation Modes. *ACM Transactions on Sensor Networks (TOSN)*, 6(2):1–27, 2010.

[51] M. Musolesi, E. Miluzzo, N.D. Lane, S.B. Eisenman, T. Choudhury, and A.T. Campbell. The Second Life of a Sensor: Integrating Real-World Experience in Virtual Worlds Using Mobile Phones. In *Proc. of the Fifth Workshop on Embedded Networked Sensors (HotEmNets'08)*, 2008.

[52] N.M. Boers, D. Chodos, J. Huang, P. Gburzynski, I. Nikolaidis, and E. Stroulia. The Smart Condo: Visualizing Independent Living Environments in a Virtual World. In *Pervasive Computing Technologies for Healthcare, 2009. PervasiveHealth 2009. 3rd International Conference on*, pages 1–8. IEEE, 2009.

[53] J. Lifton and J.A. Paradiso. Dual Reality: Merging the Real and Virtual. In *Proc. of First International Conference on Facets of Virtual Environments (FaVE'09)*, pages 12–28, 2009.

[54] Nokia n95. http://www.nokiausa.com/find-products/phones/nokia-n95.

[55] Y. Wang, J. Lin, M. Annavaram, Q.A. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh. A Framework of Energy Efficient Mobile Sensing for Automatic User State Recognition. In *Proc. of the 7th International Conference on Mobile Systems, Applications, and Services*, pages 179–192. ACM, 2009.

[56] M.R. Ra, J. Paek, A.B. Sharma, R. Govindan, M.H. Krieger, and M.J. Neely. Energy-Delay Tradeoffs in Smartphone Applications. In *Proc. of the 8th International Conference on Mobile Systems, Applications, and Services*, pages 255–270. ACM, 2010.

[57] K. Lin, A. Kansal, D. Lymberopoulos, and F. Zhao. Energy-Accuracy Trade-off for Continuous Mobile Device Location. In *Proc. of the 8th International Conference on Mobile Systems, Applications, and Services*, pages 285–298. ACM, 2010.

[58] Z. Zhuang, K.H. Kim, and J.P. Singh. Improving Energy Efficiency of Location Sensing on Smartphones. In *Proc. of the 8th International Conference on Mobile Systems, Applications, and Services*, pages 315–330. ACM, 2010.

[59] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell. The Jigsaw Continuous Sensing Engine for Mobile Phone Applications. In *Proc. of the 8th ACM Conference on Embedded Networked Sensor Systems*, pages 71–84. ACM, 2010.

[60] A. Kapadia, T. Henderson, J. Fielding, and D. Kotz. Virtual Walls: Protecting Digital Privacy in Pervasive Environments. *Pervasive Computing*, pages 162–179, 2007.

[61] C. Cornelius, A. Kapadia, D. Kotz, D. Peebles, M. Shin, and N. Triandopoulos. AnonySense: Privacy-Aware People-Centric Sensing. In *Proc. of the 6th International Conference on Mobile Systems, Applications, and Services*, pages 211–224. ACM, 2008.

[62] R.K. Ganti, N. Pham, Y.E. Tsai, and T.F. Abdelzaher. PoolView: Stream Privacy for Grassroots Participatory Sensing. In *Proc. of the 6th ACM Conference on Embedded network Sensor Systems*, pages 281–294. ACM, 2008.

[63] H. Ahmadi, N. Pham, R. Ganti, T. Abdelzaher, S. Nath, and J. Han. Privacy-Aware Regression Modeling of Participatory Sensing Data. In *Proc. of the 8th ACM Conference on Embedded Networked Sensor Systems*, pages 99–112. ACM, 2010.

[64] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proc. of the of OSDI 2010*, 2010.

[65] R. DeVaul, M. Sung, J. Gips, and A.S. Pentland. MIThril 2003: Applications and Architecture. In *Proc. of the 7th IEEE International Symposium on Wearable Computers*, page 4. IEEE Computer Society, 2003.

[66] O. Vinyals and G. Friedland. Towards Semantic Analysis of Conversations: a System for the Live Identification of Speakers in Meetings. In *Semantic Computing, 2008 IEEE International Conference on*, pages 426–431. IEEE, 2008.

[67] F. Asano, K. Yamamoto, J. Ogata, M. Yamada, and M. Nakamura. Detection and Separation of Speech Events in Meeting Recordings Using a Microphone Array. *EURASIP Journal on Audio, Speech, and Music Processing*, 2007(2):1, 2007.

[68] D. Lymberopoulos, A. Bamis, and A. Savvides. Extracting Spatiotemporal Human Activity Patterns in Assisted Living Using a Home Sensor Network. In *Proc. of the 1st International Conference on PErvasive Technologies Related to Assistive Environments*, pages 1–8. ACM, 2008.

[69] A. Wood, G. Virone, T. Doan, Q. Cao, L. Selavo, Y. Wu, L. Fang, Z. He, S. Lin, and J. Stankovic. ALARM-NET: Wireless Sensor Networks for Assisted-Living and Residential Monitoring. *University of Virginia Computer Science Department Technical Report*, 2006.

[70] S.B. Eisenman, E. Miluzzo, N.D. Lane, R.A. Peterson, G.S. Ahn, and A.T. Campbell. BikeNet: a Mobile Sensing System for Cyclist Experience Mapping. *ACM Transactions on Sensor Networks (TOSN)*, 6(1):1–39, 2009.

[71] D. Siewiorek, A. Smailagic, J. Furukawa, A. Krause, N. Moraveji, K. Reiger, J. Shaffer, and F.L. Wong. Sensay: a Context-Aware Mobile Phone. In *Wearable Computers, 2003. Proc. Seventh IEEE International Symposium on*, pages 248–249. IEEE, 2003.

[72] M. Raento, A. Oulasvirta, R. Petit, and H. Toivonen. ContextPhone: a Prototyping Platform for Context-Aware Mobile Applications. *IEEE Pervasive Computing*, pages 51–59, 2005.

[73] Nokia - sensorplanet. http://www.sensorplanet.org/.

[74] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: Making Smartphones Last Longer with Code Offload. In *Proc. of the 8th International Conference on Mobile Systems, Applications, and Services*, pages 49–62. ACM, 2010.

[75] B. Logan, J. Healey, M. Philipose, E.M. Tapia, and S. Intille. A Long-Term Evaluation of Sensing Modalities for Activity Recognition. In *Proc. of the 9th International Conference on Ubiquitous Computing*, pages 483–500. Springer-Verlag, 2007.

[76] J. Lester, T. Choudhury, G. Borriello, S. Consolvo, J. Landay, K. Everitt, and I. Smith. Sensing and Modeling Activities to Support Physical Fitness. In *Proc. of Ubicomp Workshop: Monitoring, Measuring, and Motivating Exercise: Ubiquitous Computing to Support Fitness), Tokyo, Japan*. Citeseer, 2005.

[77] G. Borriello, W. Brunette, J. Lester, P. Powledge, and A. Rea. An Ecosystem of Platforms to Support Sensors for Personal Fitness. 2006.

[78] B.L. Harrison, S. Consolvo, and T. Choudhury. Using Multi-Modal Sensing for Human Activity Modeling in the Real World. *Handbook of Ambient Intelligence and Smart Environments*, pages 463–478, 2010.

[79] N. Lane, H. Lu, S. Eisenman, and A. Campbell. Cooperative Techniques Supporting Sensor-Based People-Centric Inferencing. *Pervasive Computing*, pages 75–92, 2008.

[80] E. Miluzzo, M. Papandrea, N.D. Lane, H. Lu, and A.T. Campbell. Pocket, Bag, Hand, etc.- Automatically Detecting Phone Context through Discovery. In *Proc. of the First International Workshop on Sensing for App Phones (PhoneSense'10)*, 2010.

[81] A. Kapadia, D. Kotz, and N. Triandopoulos. Opportunistic Sensing: Security Challenges for the new Paradigm. In *Communication Systems and Networks and Workshops, 2009. COMSNETS'09. First International*, pages 1–10. IEEE, 2009.

[82] K.L. Huang, S.S. Kanhere, and W. Hu. Preserving Privacy in Participatory Sensing Systems. *Computer Communications*, 33(11):1266–1280, 2010.

[83] M. Musolesi, M. Piraccini, K. Fodor, A. Corradi, and A. Campbell. Supporting Energy-Efficient Uploading Strategies for Continuous Sensing Applications on Mobile Phones. *Pervasive Computing*, pages 355–372, 2010.

[84] Y. Wang, B. Krishnamachari, Q. Zhao, and M. Annavaram. Markov-Optimal Sensing Policy for User State Estimation in Mobile Devices. In *Proc. of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 268–278. ACM, 2010.

[85] K.K. Rachuri, M. Musolesi, C. Mascolo, P.J. Rentfrow, C. Longworth, and A. Aucinas. EmotionSense: a Mobile Phones Based Adaptive Platform for Experimental Social Psychology Research. In *Proc. of the 12th ACM International Conference on Ubiquitous Computing*, pages 281–290. ACM, 2010.

[86] J.E. Larsen and K. Jensen. Mobile Context Toolbox: an Extensible Context Framework for s60 Mobile Phones. In *Proc. of the 4th European Conference on Smart Sensing and Context*, pages 193–206. Springer-Verlag, 2009.

[87] J. Rana, J. Kristiansson, J. Hallberg, and K. Synnes. An Architecture for Mobile Social Networking Applications. In *2009 First International Conference on Computational Intelligence, Communication Systems and Networks*, pages 241–246. IEEE, 2009.

[88] T. Yan, M. Marzilli, R. Holmes, D. Ganesan, and M. Corner. mCrowd: a platform for mobile crowdsourcing. In *Proc. of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 347–348. ACM, 2009.

[89] A. Laurucci, S. Melzi, and M. Cesana. A Reconfigurable Middleware for Dynamic Management of Heterogeneous Applications in Multi-Gateway Mobile Sensor Networks. In *Sensor, Mesh and Ad Hoc Communications and Networks Workshops, 2009. SECON Workshops'09. 6th Annual IEEE Communications Society Conference on*, pages 1–3. IEEE.

[90] H. Wang and A. Chin. Evolution Analysis of a Mobile Social Network. *Advanced Data Mining and Applications*, pages 310–321, 2010.

[91] Android. http://www.android.com.

[92] iOS Dev Center. http://developer.apple.com/devcenter/ios/index.action.

[93] A.T. Campbell, S.B. Eisenman, K. Fodor, N.D. Lane, H. Lu, E. Miluzzo, M. Musolesi, R.A. Peterson, and X. Zheng. Transforming the Social Networking Experience with Sensing Presence from Mobile Phones. In *Proc. of the 6th ACM Conference on Embedded Network Sensor Systems*, pages 367–368. ACM, 2008.

[94] E.M. Tapia, S.S. Intille, W. Haskell, K. Larson, J. Wright, A. King, and R. Friedman. Real-Time Recognition of Physical Activities and their Intensities Using Wireless Accelerometers and a Heart Rate Monitor. 2007.

[95] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*, volume 2. 2001.

[96] Weka. http://www.cs.waikato.ac.nz/ml/weka/.

[97] D. Patterson, L. Liao, D. Fox, and H. Kautz. Inferring High-Level Behavior from Low-Level Sensors. In *UbiComp 2003: Ubiquitous Computing*, pages 73–89. Springer, 2003.

[98] Wikimapia. http://www.wikimapia.org.

[99] D. Ashbrook and T. Starner. Using GPS to Learn Significant Locations and Predict Movement Across Multiple Users. *Personal and Ubiquitous Computing*, 7(5):275–286, 2003.

[100] C. Zhou, D. Frankowski, P. Ludford, S. Shekhar, and L. Terveen. Discovering Personally Meaningful Places: An Interactive Clustering Approach. *ACM Transactions on Information Systems (TOIS)*, 25(3):12–es, 2007.

[101] Cenceme web site. http://www.cenceme.org.

[102] Sqlite. http://www.sqlite.org.

[103] Skyhook Wireless. http://www.skyhookwireless.com/.

[104] Kiss FFT Library. http://sourceforge.net/projects/kissfft.

[105] E. Oliver. The Challenges in Large-Scale Smartphone User Studies. In *Proc. of the 2nd ACM International Workshop on Hot Topics in Planet-scale Measurement*, pages 1–5. ACM, 2010.

[106] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in Smartphone Usage. In *Proc. of the 8th International Conference on Mobile systems, Applications, and Services*, pages 179–194. ACM, 2010.

[107] N. Li and G. Chen. Analysis of a Location-Based Social Network. In *2009 International Conference on Computational Science and Engineering*, pages 263–270. IEEE, 2009.

[108] LIBSVM. http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[109] Amazon Web Services (AWS). http://aws.amazon.com.

[110] CenceMe Discussion Board. http://getsatisfaction.com/cenceme.

[111] A. Kansal, M. Goraczko, and F. Zhao. Building a Sensor Network of Mobile Phones. In *Proc. of the 6th International Conference on Information Processing in Sensor Networks*, pages 547–548. ACM, 2007.

[112] S. Gaonkar, J. Li, R.R. Choudhury, L. Cox, and A. Schmidt. Micro-Blog: Sharing and Querying Content Through Mobile Phones and Social Participation. In *Proc. of the 6th International Conference on Mobile Systems, Applications, and Services*, pages 174–186. ACM, 2008.

[113] Second life. http://www.secondlife.com.

[114] N. Eagle and A.S. Pentland. Eigenbehaviors: Identifying Structure in Routine. *Behavioral Ecology and Sociobiology*, 63(7):1057–1066, 2009.

[115] Y. Nakanishi, K. Takahashi, T. Tsuji, and K. Hakozaki. iCAMS: A Mobile Communication Tool Using Location and Schedule Information. *Pervasive Computing*, pages 119–136, 2002.

[116] N. Marmasse, C. Schmandt, and D. Spectre. WatchMe: Communication and Awareness Between Members of a Closely-Knit Group. *UbiComp 2004: Ubiquitous Computing*, pages 214–231, 2004.

[117] Twitter. http://www.twitter.com.

[118] E. Welbourne, J. Lester, A. LaMarca, and G. Borriello. Mobile Context Inference Using Low-Cost Sensors. *Location-and Context-Awareness*, pages 254–263, 2005.

[119] P. Zappi, C. Lombriser, T. Stiefmeier, E. Farella, D. Roggen, L. Benini, and G. Tr ”oster. Activity Recognition from On-Body Sensors: Accuracy-Power Trade-Off by Dynamic Sensor Selection. *Wireless Sensor Networks*, pages 17–33, 2008.

[120] J. Paradiso, K.Y. Hsiao, and E. Hu. Interactive Music for Instrumented Dancing Shoes. In *Proc. of the 1999 International Computer Music Conference*, pages 453–456. Citeseer, 1999.

[121] D. McMillan, A. Morrison, O. Brown, M. Hall, and M. Chalmers. Further Into the Wild: Running Worldwide Trials of Mobile Systems. *Pervasive Computing*, pages 210–227, 2010.

[122] S. Zhai, P.O. Kristensson, P. Gong, M. Greiner, S.A. Peng, L.M. Liu, and A. Dunnigan. Shapewriter on the iPhone: from the Laboratory to the Real World. In *Proc. of the 27th International Conference extended abstracts on Human factors in Computing Systems*, pages 2667–2670. ACM, 2009.

[123] T. Henderson and F.B. Abdesslem. Scaling Measurement Experiments to Planet-Scale: Ethical, Regulatory and Cultural Considerations. In *Proc. of the 1st ACM International Workshop on Hot Topics of Planet-Scale Mobility Measurements*, page 6. ACM, 2009.

[124] R. Honicky, E.A. Brewer, J.F. Canny, and R.C. Cohen. Increasing the Precision of Mobile Sensing Systems Through Supersampling. *UrbanSense08*, page 31, 2008.

[125] X. Zhu. Semi-Supervised Learning Literature Survey. *Computer Science, University of Wisconsin-Madison*, 2007.

[126] Nokia Series. http://europe.nokia.com/nseries.

[127] E. Shriberg and A. Stolcke. The Case for Automatic Higher-Level Features in Forensic Speaker Recognition. In *Ninth Annual Conference of the International Speech Communication Association*, 2008.

[128] E. Shriberg, L. Ferrer, S. Kajarekar, A. Venkataraman, and A. Stolcke. Modeling Prosodic Feature Sequences for Speaker Recognition. *Speech Communication*, 46(3-4):455–472, 2005.

[129] I. Lapidot, H. Guterman, and A. Cohen. Unsupervised Speaker Recognition Based on Competition Between Self-Organizing Maps. *Neural Networks, IEEE Transactions on*, 13(4):877–887, 2002.

[130] G. Friedland and O. Vinyals. Live Speaker Identification in Conversations. In *Proc. of the 16th ACM International Conference on Multimedia*, pages 1017–1018. ACM, 2008.

[131] S. Zhang, S. Zhang, and B. Xu. A Robust Unsupervised Speaker Clustering of Speech Utterances. In *Natural Language Processing and Knowledge Engineering, 2005. IEEE NLP-KE'05. Proc. of 2005 IEEE International Conference on*, pages 115–120. IEEE.

[132] Nokia N900. http://maemo.nokia.com/n900/.

[133] S. Basu. A Linked-HMM Model for Robust Voicing and Speech Detection. In *Acoustics, Speech, and Signal Processing, 2003. Proc..(ICASSP'03). 2003 IEEE International Conference on*, volume 1, pages I–816. IEEE, 2003.

[134] F. Zheng, G. Zhang, and Z. Song. Comparison of Different Implementations of MFCC. *Journal of Computer Science and Technology*, 16(6):582–589, 2001.

[135] H. Ezzaidi and J. Rouat. Pitch and MFCC Dependent GMM Models for Speaker Identification Systems. In *Electrical and Computer Engineering, 2004. Canadian Conference on*, volume 1, pages 43–46. IEEE, 2004.

[136] E. Shriberg. Higher-Level Features in Speaker Recognition. *Speaker Classification I*, pages 241–259, 2007.

[137] D.A. Reynolds and R.C. Rose. Robust Text-Independent Speaker Identification Using Gaussian Mixture Speaker Models. *Speech and Audio Processing, IEEE Transactions on*, 3(1):72–83, 1995.

[138] D.A. Reynolds, T.F. Quatieri, and R.B. Dunn. Speaker Verification Using Adapted Gaussian Mixture Models. *Digital signal processing*, 10(1-3):19–41, 2000.

[139] C. Tadj, P. Dumouchel, and P. Ouellet. GMM Based Speaker Identification Using Training-Time-Dependent Number of Mixtures. In *Acoustics, Speech and Signal Processing, 1998. Proc. of the 1998 IEEE International Conference on*, volume 2, pages 761–764. IEEE, 1998.

[140] T. Kinnunen, T. Kilpeläinen, and P. Fränti. Comparison of Clustering Algorithms in Speaker Identification. In *Proc. of the IASTED International Conference Signal Processing and Communications*, pages 222–227, 2000.

[141] D.H. Kim, J. Hightower, R. Govindan, and D. Estrin. Discovering Semantically Meaningful Places from Pervasive RF-Beacons. In *Proc. of the 11th International Conference on Ubiquitous Computing*, pages 21–30. ACM, 2009.

[142] Rastamat. http://labrosa.ee.columbia.edu/matlab/rastamat.

[143] QT. http://qt.nokia.com/.

[144] FFTW. http://www.fftw.org.

[145] M. Zhu. Recall, Precision and Average Precision. *Department of Statistics and Actuarial Science, University of Waterloo, Waterloo*, 2004.

[146] Nokia Research Center. Sensing the World with Mobile Devices. *Technical Report*, 2008.

[147] E. Miluzzo, N. Lane, A. Campbell, and R. Olfati-Saber. Calibree: A self-calibration system for mobile sensor networks. *Distributed Computing in Sensor Systems*, pages 314–331, 2008.

[148] R. Kumar, M. Wolenetz, B. Agarwalla, J.S. Shin, P. Hutto, A. Paul, and U. Ramachandran. DFuse: a Framework For Distributed Data Fusion. In *Proc. of the 1st International Conference on Embedded Networked Sensor Systems*, pages 114–125. ACM, 2003.

[149] S. Patil, S.R. Das, and A. Nasipuri. Serial Data Fusion Using Space-Filling Curves in Wireless Sensor Networks. In *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON'04. 2004 First Annual IEEE Communications Society Conference on*, pages 182–190. IEEE, 2004.

[150] L. Xiao, S. Boyd, and S. Lall. A Scheme for Robust Distributed Sensor Fusion Based on Average Consensus. In *Information Processing in Sensor Networks, 2005. IPSN'05. Fourth International Symposium on*, pages 63–70. IEEE, 2005.

[151] J. Zhao, R. Govindan, and D. Estrin. Computing Aggregates for Monitoring Wireless Sensor Networks. In *Sensor Network Protocols and Applications, 2003. Proc. of the First IEEE. 2003 IEEE International Workshop on*, pages 139–148. IEEE, 2003.

[152] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and Beyond: new Aggregation Techniques for Sensor Networks. In *Proc. of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 239–249. ACM, 2004.

[153] S. Nath, P.B. Gibbons, S. Seshan, and Z.R. Anderson. Synopsis Diffusion for Robust Aggregation in Sensor Networks. In *Proc. of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 250–262. ACM, 2004.

[154] R. Olfati-Saber. Distributed Kalman Filter with Embedded Consensus Filters. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pages 8179–8184. IEEE, 2005.

[155] R. Olfati-Saber. Distributed Kalman Filtering and Sensor Fusion in Sensor Networks. *Networked Embedded Sensing and Control*, pages 157–167, 2006.

[156] The CASA Project. http://www.casa.umass.edu.

[157] S.B. Eisenman. People-Centric Mobile Sensing Networks. *Ph.D. Dissertation*, October 2008.

[158] D. Yarowsky. Unsupervised Word Sense Disambiguation Rivaling Supervised Methods. In *Proc. of the 33rd annual meeting on Association for Computational Linguistics*, pages 189–196. Association for Computational Linguistics, 1995.

[159] E. Riloff, J. Wiebe, and T. Wilson. Learning Subjective Nouns Using Extraction Pattern Bootstrapping. In *Proc. of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 25–32. Association for Computational Linguistics, 2003.

[160] B. Maeireizo, D. Litman, and R. Hwa. Co-Training for Predicting Emotions with Spoken Dialogue Data. In *Proc. of the ACL 2004 on Interactive poster and demonstration sessions*, pages 28–es. Association for Computational Linguistics, 2004.

[161] M. Weiser. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, 36(7):75–84, 1993.

[162] B. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *WMCSA*, pages 85–90. IEEE Computer Society, 2004.

[163] P. Dourish. What We Talk About When We Talk About Context. *Personal and ubiquitous computing*, 8(1):19–30, 2004.

[164] J. Lukkari, J. Korhonen, and T. Ojala. SmartRestaurant: Mobile Payments in Context-Aware Environment. In *Proc. of the 6th International Conference on Electronic Commerce*, pages 575–582. ACM, 2004.

[165] J. Parkka, M. Ermes, K. Antila, M. van Gils, A. Manttari, and H. Nieminen. Estimating Intensity of Physical Activity: a Comparison of Wearable Accelerometer and Gyro Sensors and 3 Sensor Locations. In *Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE*, pages 1511–1514. IEEE, 2007.

[166] Q. Li, J.A. Stankovic, M.A. Hanson, A.T. Barth, J. Lach, and G. Zhou. Accurate, Fast Fall Detection Using Gyroscopes and Accelerometer-Derived Posture Information. *2009 Body Sensor Networks*, pages 138–143, 2009.

[167] S.W. Lee and K. Mase. Activity and Location Recognition Using Wearable Sensors. *Pervasive Computing, IEEE*, 1(3):24–32, 2002.

[168] Y. Kawahara, H. Kurasawa, and H. Morikawa. Recognizing User Context Using Mobile Handset with Acceleration Sensors. In *Portable Information Devices, 2007. PORTABLE'07. IEEE International Conference on*, pages 1–5. IEEE, 2007.

[169] Smartphone race heats up. http://bits.blogs.nytimes.com/2010/12/02/the-u-s-smartphone-race-grows/?ref=technology.

[170] Citysense. http://www.citysense.com/.

[171] T. Horanont and R. Shibasaki. An Implementation of Mobile Sensing for Large-Scale Urban Monitoring. In *Proc. of Urbansense'08*, 2008.

[172] N.D. Lane, D. Lymberopoulos, F. Zhao, and A.T. Campbell. Hapori: Context-based Local Search for Mobile Phones using Community Behavioral Modeling and Similarity. In *Proc. of the 12th ACM International Conference on Ubiquitous Computing*, pages 109–118. ACM, 2010.

[173] Wertago. http://wertago.com/index.html.

[174] Vibn web site. http://sensorlab.cs.dartmouth.edu/vibn.

[175] Dbscan. http://en.wikipedia.org/wiki/DBSCAN.

[176] A. Subramanya, A. Raj, J. Bilmes, and D. Fox. Recognizing Activities and Spatial Context Using Wearable Sensors. In *Proc. of the Conference on Uncertainty in Artificial Intelligence*. Citeseer, 2006.

[177] L. Von Ahn. Games with a Purpose. *Computer*, 39(6):92–94, 2006.

[178] Atus. http://www.bls.gov/tus.

[179] D. Wyatt, M. Philipose, and T. Choudhury. Unsupervised Activity Recognition Using Automatically Mined Common Sense. In *Proc. of the National Conference on Artificial Intelligence*, volume 20, page 21. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[180] J.B. Lovins and MASSACHUSETTS INST OF TECH CAMBRIDGE ELECTRONIC SYSTEMS LAB. *Development of a Stemming Algorithm*. MIT Information Processing Group, Electronic Systems Laboratory, 1968.

[181] T. Das, P. Mohan, V.N. Padmanabhan, R. Ramjee, and A. Sharma. PRISM: Platform for Remote Sensing Using Smartphones. In *Proc. of the 8th International Conference on Mobile Systems, Applications, and Services*, pages 63–76. ACM, 2010.

[182] X. Bao and R. Roy Choudhury. MoVi: Mobile Phone Based Video Highlights via Collaborative Sensing. In *Proc. of the 8th International Conference on Mobile Systems, Applications, and Services*, pages 357–370. ACM, 2010.

[183] G.R. Yavuz, M.E. Kocak, G. Ergun, H. Alemdar, H. Yalcin, O.D. Incel, L. Akarun, and C. Ersoy. A Smartphone Based Fall Detector with Online Location Support. In *Proc. of PhoneSense'10*, 2010.

[184] A. Madan, M. Cebrian, D. Lazer, and A. Pentland. Social Sensing for Epidemiological Behavior Change. In *Proc. of the 12th ACM International Conference on Ubiquitous Computing*, pages 291–300. ACM, 2010.

[185] D.H. Kim, Y. Kim, D. Estrin, and M.B. Srivastava. SensLoc: Sensing Everyday Places and Paths Using Less Energy. In *Proc. of the 8th ACM Conference on Embedded Networked Sensor Systems*, pages 43–56. ACM, 2010.

[186] R.K. Ganti, N. Pham, H. Ahmadi, S. Nangia, and T.F. Abdelzaher. GreenGPS: A Participatory Sensing Fuel-Efficient Maps Application. In *Proc. of the 8th International Conference on Mobile Systems, Applications, and Services*, pages 151–164. ACM, 2010.

[187] Foursquare web site. http://foursquare.com.

[188] Tweetmic web site. http://tweetmic.com.

[189] M. Rohs, S. Kratz, R. Schleicher, A. Sahami, and A. Schmidt. WorldCupinion: Experiences with an Android App for Real-Time Opinion Sharing during World Cup Soccer Games. In *Proc. of Research In The Large Wokshop'10*, 2010.