DESIGNING EFFICIENT AND ACCURATE BEHAVIOR-AWARE MOBILE SYSTEMS

A Dissertation Presented

by

ABHINAV PARATE

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2014

School of Computer Science

DESIGNING EFFICIENT AND ACCURATE BEHAVIOR-AWARE MOBILE SYSTEMS

A Dissertation Presented

by

ABHINAV PARATE

Approved as to style and content by:

_____

Deepak Ganesan, Chair

_____

Prashant Shenoy, Member

_____

Benjamin M. Marlin, Member

_____

Evangelos Kalogerakis, Member

_____

Christopher Salthouse, Member

_____

Lori A. Clarke, Chair
School of Computer Science

||*Om*||

I dedicate this thesis to my parents,
Rekha and Shrawan Parate,
for their love, endless support and encouragement.

And to my wife,
Sujata,
for her love, sacrifices, patience and support
that made this thesis possible.

ACKNOWLEDMENTS

This journey towards Ph.D. has been a long one and this would not have been possible without the support of all those who accompanied me at various stages. I would like to express my immense gratitude to all those who extended their support when I faced the cold winters, and to all those whose company brought joy, cheer and the beautiful springs in this journey.

First and foremost, I would like to thank my advisor, Deepak Ganesan. His perseverance, curiosity and willingness to dive into the unknown and exciting fields are some of the traits that helped me explore research problems in the territory which neither of us had experience with. His advices, guidance and support encouraged me to evolve into an ambitious researcher from a struggling graduate student. He taught me to embrace the uncertainty in research with optimism. I thank him for teaching me the valuable lessons about the lifecycle of performing research. Thanks to many hours of discussions with him, I learnt how to identify an interesting research problem, crystallize the novel ideas and formulate robust experiments. His aim for perfection helped me improve my research process and polish my skill-set.

I would like to express my gratitude to the esteemed members of my thesis committee - Prashant Shenoy, Benjamin Marlin, Christopher Salthouse and Evangelos Kalogerakis. My special thanks to Prashant who helped me at various stages of this journey and without his support and trust in my capabilities, I would not have reached here. I would like to thank Benjamin Marlin with whom I collaborated on various projects. His keen insights had been extremely helpful in addressing various challenges in my research. His advices helped me broaden my vision as I learned to incorporate machine learning in my research in a practical manner. I thank Christopher Salthouse for his valuable feedback that has been extremely

helpful in improving this thesis. His research work is truly inspiring and motivating. I thank Evangelos Kalogerakis whose advice has been very helpful in developing a working gesture recognition system for phones. Thanks to him, the research is now on its way to explore new territories.

I thank fellow graduate students Meng-Chieh Chiu and Chaniel Chadowitz whose contributions in my research projects were of immense help. I would like to thank David Chu (Microsoft Research) and Matthias Böhmer (DFKI Germany) with whom I collaborated on a research project that resulted in a truly successful mobile application in the Android market. Matthias' advices and ideas on HCI continue to be of help when I design new Android applications. I thank the team from Yale School of Medicine- Gustavo Angarita, Edward Gaiser, Robert Mallison. I had a great learning experience while working with this medical team when we conducted user studies with human subjects; the experience that I was able to use later in my own research.

I would like to thank my lab-mates and colleagues at UMass who were on the same boat as me and it was great to have their company. Jeremy Gummeson, Pengyu Zhang, Rahul Singh, Tingxin Yan, Upendra Sharma, Akshat Kumar, Aditya Mishra, Anand Seetharam, Annamalai Natarajan, Moaj Musthag, Vijay Pasikanti, Siddharth Gupta - brainstorming, having lunches together, discussing and laughing with you all made this journey a lot easier.

During these years of Ph.D. journey, I made a few wonderful friends. Himanshu Agrawal and Pranshu Sharma - thanks for all the fun-filled cooking sessions and the disastrous skiing sessions we had. Yash Sanghai and Anil Jain, thanks for all the wonderful memories in Pufton Village and for the camping experiences in the wild. Varun Srinivasan, an unexpected friend who happens to be full of excitement, and has always been there to help in any situation.

I would also like to thank the staff of School of Computer Science, especially Leeanne Leclerc for her help in tracking graduate school requirements, and Karren Sacco for all her

administrative help in handling non-research problems that I faced while conducting user studies required for my research.

I cannot express enough gratitude towards my family. My parents, Rekha and Shrawan Parate, and my sister Sweety Parate, thanks for supporting me and my journey in a far away country. I thank my parents for the immeasurable love and encouragement I received from them during the tough times. The sweet phone conversations with my sister are among the sweetest memories I made here. My acknowledgements can never be complete without expressing my heartfelt appreciation for the person who completes me, my better half, Sujata. Sujata accompanied me with her love and unlimited patience when I was pre-occupied with my paper deadlines. It was her support that helped me defend my dissertation proposal less than 2 weeks before our wedding ceremony. Without her support, I would never have been able to accomplish this dissertation.

ABSTRACT

DESIGNING EFFICIENT AND ACCURATE BEHAVIOR-AWARE MOBILE SYSTEMS

SEPTEMBER 2014

ABHINAV PARATE

B.Tech., INDIAN INSTITUTE OF TECHNOLOGY KANPUR

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Deepak Ganesan

The proliferation of sensors on smartphones, tablets and wearables has led to a plethora of behavior classification algorithms designed to sense various aspects of individual user's behavior such as daily habits, activity, physiology, mobility, sleep, emotional and social contexts. This ability to sense and understand behaviors of mobile users will drive the next generation of mobile applications providing services based on the users' behavioral patterns. In this thesis, we investigate ways in which we can enhance and utilize the understanding of user behaviors in such applications. In particular, we focus on identifying the key challenges in the following three aspects of behavior-aware applications: detection, understanding, and prediction of user behaviors; and present systems and techniques developed to address these challenges.

In the first part of this thesis, we demonstrate the utility of wristbands equipped with inertial sensors in real-time detection of health-related behaviors such as smoking and eat-

ing. Our approach detects these behaviors in a passive manner without any explicit user interaction and does not require use of any cumbersome device. Our results show that we can detect smoking with 95% accuracy, 91% precision and 81% recall in the natural environment.

In the second part of the thesis, we design a context-query engine for sensing multiple user contexts continuously, accurately and efficiently on mobile devices; the key necessity for understanding and analyzing behaviors. Our context-query engine performs information fusion of contexts for an individual user to enable optimizations like i) energy-efficient sensing, and ii) accurate context inference while minimizing the privacy risks associated with the leakage of sensitive contexts to third party applications. We show that our context-query engine can improve accuracy of a context classifier by up to 42% and reduce the number of classifiers required to observe the user state by 33%.

Finally, in the third part of the thesis, we demonstrate the utility of predicting app usage behavior, in improving the freshness of mobile apps such as *Facebook* that present users with the latest content fetched from remote servers. We present an app prediction algorithm that utilizes user contexts to predict the app a user is likely to use and pre-fetches the data over the network for the predicted app. We show that our proposed algorithm delivers application content to the user that is on an average fresh within 3 minutes. An implementation of our algorithm is available as a widget in the Google Play Store that shows shortcuts for the predicted apps; and has now been downloaded and installed on more than 50,000 devices.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Definition (User Context). *The term **user context** is used in the area of mobile computing to refer to the state of the user of a mobile device and the environment surrounding it.*

We have entered a new age of mobile computing where context awareness on mobile devices will play a big role in providing services targeting the behavior of mobile users. A new class of mobile applications will provide behavior-aware services such as serendipitous recommender systems that recommend activities based on an individual's behavior, health applications that intervene and provide well-timed feedback to encourage users to adopt a healthy lifestyle, applications that show information and contents of interest at appropriate times based on the user behavioral patterns, and mobile operating systems that optimize their resource consumption by predicting workloads based on the user behavioral patterns. Google Now [5], an example of such a behavior-aware application, utilizes user contexts and the mobility behavior of the user, to show driving directions to the likely places of interest to the user at opportune moments.

There are many benefits to reap from **behavior-aware applications**, but the development of this class of applications is not free of challenges. The most fundamental challenge comes from the resource-constrained nature of the mobile devices on which these applications sense user contexts for behavior-awareness. The challenges posed by the limited battery life and the limited computational power available on the mobile devices are only a few of the many challenges that needs to be addressed for behavior awareness. This thesis identifies the *key challenges* in behavior-awareness, and presents new *methods* and *systems* developed to address these challenges.

1.1 Behavior-Aware Applications

Behavior-aware applications may have a variety of goals; we can broadly categorize these applications based on the following three goals:

1. **detecting** a user behavior,
2. **understanding** a user behavior, and
3. **predicting** a user behavior.

In this section, we elaborate these three aspects of behavior-aware applications, namely **behavior detection**, **behavior understanding** and **behavior prediction**. We will see that each of these three aspects of behavior-awareness, relies on the **context awareness** or the ability to sense user contexts on mobile devices.

1.1.1 Behavior Detection

Being able to detect a user behavior is an important requirement for the behavior-aware applications. A variety of machine learning techniques exist in the literature that detects user behaviors on mobile devices using the inbuilt sensors (e.g. accelerometer, gyroscope), or using wearable accessories such as fitness monitors (Fitbit [3], UP by Jawbone [8]), heart monitors (Zephyr BioHarness [192]), and others that communicate with mobile devices via bluetooth. These techniques extract behavior information from the continuous stream of data sampled from the sensors.

Recent advances in the context-awareness research have shown that mobile devices and wearable sensors can be used to detect contexts capturing various aspects of an individual user's behavior. These behavioral contexts include physical activities (e.g. walking, driving, running, talking) [106, 151], emotional or affective contexts (e.g. stress, mood) [104, 142], social contexts (e.g. user is with friends, is with colleagues or is in a party) [31, 169], sleep behavior [82, 83], addictive behavior (e.g. smoking, cocaine) [12, 126] and so on.

Behavior detection has been most popular in applications that detect specific user behaviors such as physical activities or the sleep behavior; and present users with their daily diary containing a timeline of the observed behaviors throughout the day, or presenting a summary of the duration for each type of detected behavior e.g. time spent walking, running, standing, and sitting during the day. For example, SleepBot [167] and Moves [120] are popular daily behavior logging applications available on Android operating system that logs user's sleeping behavior and physical activities respectively. Many users utilize such applications to monitor their behaviors, and use the observations to improve their lifestyle.

1.1.2 Behavior Understanding

The main advantage of context-awareness on mobile devices is that it presents us with a rich source of information about an individual user and its surroundings. We can sense various behavioral contexts as well as the contexts that reflect the environment surrounding the user. By logging all these user contexts, we can analyze the logged data to improve our understanding of user behaviors. We can learn how behavioral contexts relate with each other in space and time, how certain behaviors are influenced by the environmental factors, and what are the causes or the factors that lead a user to a specific behavior. A variety of data analysis and mining techniques such as clustering, association rule mining, temporal rule mining and linear regression can be applied over the logged contextual data for analyzing and understanding user behaviors. This contextual data opens up many interesting opportunities such as in a health domain where mobile health applications can identify as well as monitor factors leading an individual to health conditions like stress, smoking or drug-use. Applications that analyze user behaviors may implement their own behavior detection algorithms or may rely on third-party context-sensing APIs [13, 40, 80, 106] to capture various user contexts. An example of such an application is BeWell+ [91] that analyzes sleeping behavior, physical activities and social interaction behavior throughout the

day to compute the overall well-being score for an individual user. BeWell+ application implements its own algorithms for detecting various behavior contexts.

1.1.3 Behavior Prediction

The third aspect of the behavior-awareness is the ability to predict when certain behaviors will occur. This prediction capability will drive the next generation of smart applications such as smart recommender systems that recommend movies and places to visit based on the predicted user behavior, mobile operating systems that optimize resource consumption by predicting a workload based on user's device usage behaviors, applications that show targeted advertisements and contents of interest to the users, health applications for real-time interventions to prevent a user from engaging in a harmful behavior such as smoking, and so on. The most well-known example of a behavior-predicting application is Google Now [5], a proactive and intelligent personal assistant system developed for Android mobile devices. Google Now system observes user's mobility behavior to predict the user's next likely location and presents a card to the user containing information about traffic conditions to this location.

In order to realize the many benefits of behavior predictions, it is important that the applications predict the behaviors of interest with high accuracy. Since not all the users have the same behavioral patterns, *personalization* i.e. adapting the prediction mechanism for each individual user is important for the accurate behavior prediction. Moreover, user contexts play an important role in accurate behavior prediction as the user behaviors are often influenced by a variety of user contexts. For instance, an individual user's smoking behavior depends on the user's social contexts. We might observe that some individual user smokes a cigarette when the user is either *alone* or when the user is *with specific set of friends*, but never smokes when the user is *with family members*. Consequentially, behavior prediction mechanisms may benefit from the *behavior understanding* where the contexts identified as factors leading to a target behavior, can be used in prediction.

1.2 Challenges

In this section, we give a high-level overview of the challenges that must be addressed in order to build the categories of behavior-aware applications identified in the previous section.

1.2.1 Challenges in Behavior Detection

Behavior detection is often done using classification algorithms that operate over a stream of sensor data to detect if a user is engaged in a certain behavior. Developing a behavior classification mechanism that achieves high accuracy requires a lot of efforts, starting from selecting the sensing modalities appropriate for the target behavior to be detected, to developing a classification algorithm to execute on resource-constrained mobile devices. First, the process of collecting labeled data in the real-world settings, required for evaluating and/or training a classification mechanism, is a time-consuming process requiring hours of manual efforts and data cleaning. Next, noise present in the raw sensor data presents another challenge and requires developing techniques to remove this noise from the data. But most importantly, the biggest challenge is identifying the discriminative set of features that can distinguish the behavior of interest from many confounding similar behaviors observed in the real world settings. The final challenge in the behavior classification is posed by the resource-constrained nature of the mobile device that requires the classification mechanism to have a low system overhead. Being able to efficiently execute the classification mechanism on mobile devices, is crucial due to the limited computing resources and limited energy available on these devices.

The approaches used in addressing the above-mentioned challenges are closely tied to the behavior being studied and the sensor modalities being used for detection. For instance, the cigarette smoking behavior can be detected using various sensing modalities such as a wearable chest-band that measures the expansion of the chest during respiratory cycles, or a wearable wristband equipped with inertial sensors (accelerometer and gyroscope) that

tracks wrist movements in a 3D space. Due to the dissimilar nature of these sensing modalities, the noise characteristics observed in its signals differ significantly. Hence, different noise reduction techniques are required to remove the noise from the sensor signals. Similarly, the set of discriminative features differs with the choice of sensing modalities. Moreover, the list of activities that confound with the cigarette smoking behavior for these sensing modalities differ as well. Physical activities like running that result in deep inhalations and exhalations, conversation and stress are the confounding activities when using a chest-band as the sensing modality; whereas with a wristband as the sensing modality, the list of confounders include eating, drinking, and other hand-to-mouth gestures.

The difference in approaches may also arise due to the dissimilar nature of the behaviors being studied while using the same sensing modality. For example, a wristband equipped with accelerometer can be used to detect walking as well as smoking behavior. While walking detection utilizes the features that capture periodicity in signals (e.g. zero crossing rate), whereas smoking detection requires spatio-temporal features such as the vertical displacement and the average speed of the wrist during the smoking gestures. Thus, achieving the goal of high accuracy in behavior detection requires approaches that are specific to the domain of the behavior being detected.

1.2.2 Challenges in Behavior Understanding

The key requirement to understand or to analyze a specific behavior of interest is the ability to sense a variety of user contexts continuously and simultaneously. This requirement is not just limited to the learning stage, and must be met during the application/monitoring stage as well. During the monitoring stage, multiple contexts identified as factors (or causes) during the learning stage need to be sensed continuously in order to take appropriate actions when indicated by the observed user state. The continuous sensing of user contexts faces several challenges in a resource-constrained mobile environment. First, the mobile devices lack the computational power required to continuously execute all the

classification algorithms that extract high-level context information from the sensor data. Some of the classification mechanisms that operate over high-frequency data such as audio, can consume significant computation resources on the mobile device making it infeasible to run over an extended period of time. Moreover, the power required to continuously sample sensor data and to execute computations can quickly deplete the battery life of the mobile device and thus, degrading the mobile user experience. Another important challenge here is that many of the classification mechanisms are not always accurate. Using inaccurate classified outputs to learn the causes can lead to learning incorrect relations and causes. As a consequence, the applications that use the learnt relations and causes to take certain actions useful to the user, may take actions that are either incorrect or are executed at inappropriate times.

1.2.3 Challenges in Behavior Prediction

Similar to behavior classification mechanisms, the predictions mechanisms for a user behavior are required to achieve high accuracy. It is because an incorrect prediction or failure to predict a certain behavior, can have significant costs associated with it. However, the problem of behavior prediction becomes more challenging due to the fact that an individual user's pattern of behavior can change over time. As a result, the prediction mechanism is required to learn and adapt quickly to the changing behavioral patterns over time. Another important challenge comes from the fact that many applications that utilize these predictions are time-critical in nature. In other words, it is not sufficient to predict "what" behavior will happen next but "when" that behavior will happen is also an important information for this class of applications. For instance, in the health domain, predicting "when" a user is likely to smoke can be critical for real-time interventions to prevent the user from smoking.

This is worthwhile to note that addressing the above-mentioned challenges requires approaches that depend on the type of behavior being predicted. This is because the user

contexts useful in behavior prediction and the temporal behavioral patterns differ for various types of behaviors. For example, predicting the smoking behavior is very different from predicting the user behavior that the user is going to check the email application on the phone.

1.3 Problems

The space of challenges in designing robust, efficient, and accurate behavior-aware applications is vast, and the specifics depend on the application being studied. For example, the challenges in detecting behavior depends on the detection goal (smoking, drug use, etc) and the choice of sensor modality (accelerometer, microphone, etc). Similarly, the challenges in other aspects of behavior-awareness depend on the problem. In this section, we instantiate these challenges in three applications that we address in this thesis, one for each of the three areas outlined in the previous section.

1.3.1 Detecting Gesture-based Behaviors Using Wrist-worn Devices

In a recent market trend, we are increasingly seeing new wearable wrist-worn devices such as Fitbit wristband [3], UP wristband by Jawbone [8], Samsung Galaxy Gear [154], that come equipped with inertial sensors. The focus of these devices has been on recognizing few standard activities like *walking, driving, biking, sitting, running* and *standing*. However, inertial sensors available on these devices can be used to recognize behaviors beyond the list of standard activities. In particular, inertial sensors specifically 9-axis inertial measurement unit (IMU), can capture rich information about the natural hand gestures performed by a user in a day-to-day life. The natural hand gestures are useful in recognizing health behaviors such as smoking, eating, drinking and various types of physical exercises. These behaviors provide valuable information pertaining to the users' well-being and can play an important role in health applications such as smoking cessation, monitoring overeating or binge-eating and monitoring fitness regime. In contrast to the physical activities

like walking and biking, these behaviors are relatively uncommon and happens only a few times a day and sometimes for a short duration of few minutes. Thus, it is important that the algorithm that detects these behaviors should have high accuracy; the algorithm should detect most of the instances of these behaviors (high recall) and should have only few false positives (high precision). This identifies our first problem that we address in this thesis "How can we leverage wrist-worn devices equipped with inertial sensors to accurately sense behaviors based on wrist movements?"

The biggest challenge in behavior sensing using wrist-worn devices is that a typical user performs hundreds of hand-gestures in a day and many of these gestures have arm movement patterns similar to the behaviors we need to distinguish. Moreover, gestures for eating, smoking and drinking are all hand-to-mouth gestures and may have similar repetitive patterns. Another important challenge comes from our aim to design a passive gesture tracking system that should detect gestures without any explicit information given by the user regarding the type of activity being performed. This is in contrast to typical gesture recognition systems such as Nintendo-Wii where a user signals the beginning of a gesture.

1.3.2 Understanding Behavior in a Resource-constrained Mobile Environment

A lot of research in the recent years has focused on an independent analysis of a single behavioral context such as physical activity, user's emotional state, sleeping behavior or user's social interactions. We argue that instead of looking at the various contexts independently, we can learn spatio-temporal correlations that exist across these contexts; and leverage these correlations to i) gain insights about an individual user, and ii) enable system optimizations such as reduced energy consumption and improved accuracy in context-sensing. These optimizations have benefits particularly for the *behavior understanding* and *analysis* applications that require simultaneous and continuous sensing of multiple user contexts. In addition to behavior analysis, the ability to continuously sense multiple contexts *accurately*

and *efficiently* is of utmost importance for supporting contextualized services on resource-constrained mobile devices. We address the challenges in achieving accurate continuous context-sensing on mobile devices in the second problem of this thesis "How can we improve continuous context sensing performance?" Our goal is to improve the two performance metrics, namely, *energy-efficiency* and *accuracy* of the context-inference algorithms (or classifiers) used in observing user contexts. The intuition behind this problem is that the user contexts capture various aspects of an individual user's habits, behaviors, surroundings and physiology all of which are inter-linked and hence, user contexts are correlated across space and time. Consequentially, we can improve *energy-efficiency* in context-sensing by sensing only fewer contexts and inferring contexts that are implied by the correlations. At the same time, *accuracy* of the individual context-inference algorithms/classifiers can be improved by correcting the outputs of the context-inference algorithms if the outputs are inconsistent with the correlations. For example, if a context-inference algorithm falsely detects that the user is "driving" while the user is in "office", we can correct the output of the algorithm to "not driving".

While there are benefits to reap, there are number of challenges that we must address. The first challenge is how to model the correlations across contexts over time. The context-relationship model we choose should be able to take into account that the output of the context-inference algorithms (or classifiers) are not always accurate and have an uncertainty associated with it. The second challenge is how to learn the context-relationship model for an individual user. Learning a unique model for each individual user can be beneficial because the interaction between user contexts is personalized in nature. The third and a practical challenge is minimizing privacy risks for a user who may be concerned about leakage of a sensitive user context to a third party application by leveraging the context-relationship model. The final challenge is that the system overhead in combining and inferring contexts from a relationship model should not be high. We should be able to infer contexts and ensure privacy in real-time on a mobile device.

1.3.3 Predicting App Usage Behavior for Improved Experience with Mobile Apps

The third problem we address in this thesis is "How can we predict user's app usage behavior and utilize the predictions to improve the user experience with mobile apps?" In particular, we focus on a class of mobile apps that shows useful dynamic content to the user, obtained from the web. Here, our goal is to improve the freshness of the dynamic content in the mobile apps such that user finds up-to-date content whenever an app is launched. To achieve this goal, an app prediction algorithm that utilizes user contexts can be very useful. It is because user contexts often provide a helpful clue to predict what app will be used by a user in a given context. These clues can be used to refresh the content for the predicted app prior to its use by the user. However, one of the requirements from the prediction algorithm is that the algorithm should be light-weight, that is, the algorithm should be able to execute and train itself on off-the-shelf phones. This is because prediction algorithms require long temporal history of user data to train an accurate predictor; and this data, consisting of user contexts, can be privacy-sensitive in nature. As a result, users might prefer an algorithm that trains itself on the device and not on the remote server in the cloud. Moreover, an algorithm that has a low system overhead during its execution stage, will have a low impact on the overall performance of the phone. However, the low system overhead requirement means that the prediction algorithm cannot rely on user contexts that are expensive to obtain, for instance, location from a GPS sensor.

To address this problem, we first need to answer what user contexts are useful in predicting the app likely to be used. Apart from this, the main challenge in addressing this problem is that an individual user's app usage behavior changes over time as new apps are installed on the device. Thus, an algorithm that predicts the app that is likely to be used next, should be able to adapt quickly to the changing user behavior. The second challenge is the time-critical nature of the problem i.e. the algorithm should be able to predict "when" an app is likely to be used as the freshness of the dynamic content depends on the well-timed update of the content. Finally, the algorithm should judiciously use the net-

work bandwidth in refreshing the predicted applications. It is because network bandwidth is a precious resource and costs money to the user. Wasteful or too frequent refreshes can quickly consume the network bandwidth permitted by the user's cellular data plan.

1.4 Thesis Statement

Our thesis is that we can

- sense behavioral contexts like smoking and eating with high recall and precision, using wrist-worn devices equipped with inertial sensors.
- support behavior understanding and analysis applications by leveraging correlations across user contexts to reduce energy-requirements in continuous context-sensing and to improve the accuracy of context-inference algorithms.
- improve user experience with mobile apps by using contexts to predict app usage behavior and to decide when to refresh an app's content so that the user is presented with the freshest content.

Table 1.1. Thesis overview

Question	Answer	Chapters
How can we leverage wrist-worn devices equipped with inertial sensors to accurately sense behaviors?	RisQ: A mobile solution for hand-gesture based behavior recognition.	Chapter 3
How to make continuous context-inference energy-efficient and accurate on mobile devices?	CQue: A context-query engine exploiting context correlations.	Chapter 4
How to predict user's app usage behavior to improve user experience with mobile applications?	PREPP: An app prediction algorithm for refreshing apps.	Chapter 5

1.5 Thesis Overview

In this thesis, we address three key research questions. We answer one question in each part of the thesis and describe the systems we have developed as a solution. Table 1.1 gives an overview of this thesis summarizing the questions and the systems we developed in response. Next, we give an overview of the systems described in this thesis.

1.5.1 RisQ: Gesture Recognition with Inertial Sensors

In the first part of this thesis, we focus on building a passive gesture recognition system for mobile devices that can recognize hand-to-mouth gestures like smoking and eating using wrist-worn devices. We employ wristbands equipped with inertial sensors to detect hand-to-mouth gestures. The ability to detect hand-to-mouth gestures can be useful in measuring the contextual factors that influence harmful behaviors like smoking or binge-eating. This can further enable the use of mobile devices as a platform of choice for intervention-based strategies to prevent a user from relapsing to the harmful behavior.

In Chapter 3, we design *RisQ*, a mobile solution that leverages a wristband containing a 9-axis inertial measurements unit to capture person's arm movements, and a data processing pipeline to accurately detect the gestures like smoking and to measure the duration of the smoking sessions in real-time. *RisQ* explores the use of arm-trajectory computed using data from the inertial measurements unit to extract a set of features that can distinguish between hand-to-mouth gestures like smoking, eating and drinking. Also, we present a method that leverages multiple inertial sensors placed on a person's body together with 3D animation of a person's arm to aid in labeled data collection in the real-world settings. This approach reduces the number of self-reports a user is required to provide as labels for the various behaviors. The *RisQ* data processing pipeline along with the data collection method enables ubiquitous and non-intrusive data collection and gesture recognition on the mobile devices.

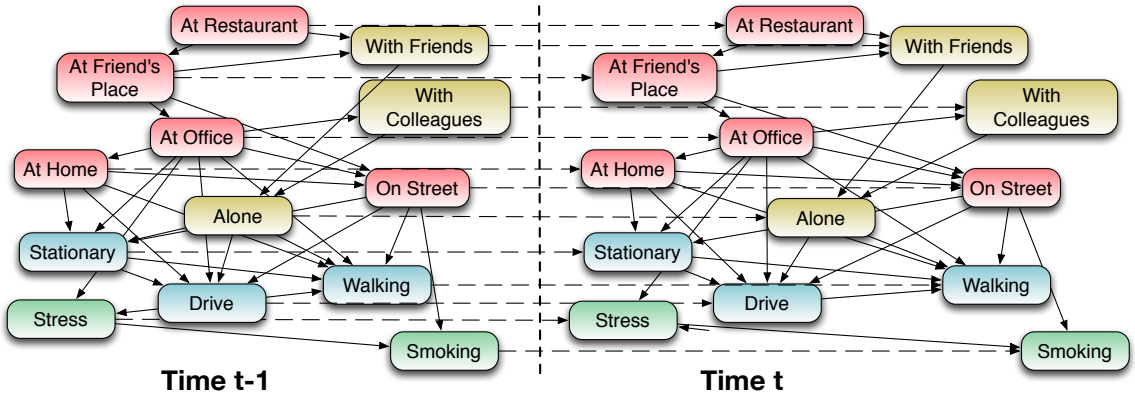


Figure 1.1. An example of Dynamic Bayesian Network showing correlations among contexts. The static correlations are represented by edges between contexts at the same timestamp whereas temporal correlations are represented by dashed edges between contexts across two timestamps.

1.5.2 CQue: Context Query Engine

In the second part of the thesis we argue that there is a need for an inference layer over the context-inference algorithms (or classifiers) that fuses the outputs of several context-inference algorithms to learn deeper insights into an individual's habitual patterns and associated correlations between contexts, thereby enabling new system optimizations and opportunities.

In Chapter 4, we design *CQue*, a dynamic bayesian network based layer that operates over classifiers for individual contexts, observes relationships across these outputs across time, and identifies opportunities for improving energy-efficiency and accuracy by taking advantage of these relations. In addition, such a layer provides insights into privacy leakage that might occur when seemingly innocuous contexts revealed to different mobile applications may be combined to reveal more information than originally intended. The main advantage of *CQue* is that the decision of "what" sensors to use and "when" to use is made by the *CQue* context query engine while ensuring optimal performance and minimizing the leakage of privacy sensitive context.

Figure 1.1 shows an example of dynamic bayesian network representing correlations among social, location, physical activity and health contexts. It models static correlations among contexts such as "at home" context implies that user is likely to be "stationary". It also models temporal consistency of contexts. An example of a temporal consistency is: a user who is "stationary" now is likely to be in the same state in the near future.

1.5.3 PREPP: Predictive Practical Prefetch

In the third part of the thesis, we focus on improving the user experience by increasing the responsiveness of the mobile applications via faster launch and initialization of the applications with the content fetched prior to its use by the user, based on the app usage behavior predictions.

In Chapter 5, we present *PREPP*, a system that predicts the application likely to be used by a user, prelaunches the predicted application and prefetches the application data over the network to present up-to-date content to the user. *PREPP* has an *app prediction model* that uses the *last set of apps* used by the user and the *time of day* as the contextual features useful in predicting the next likely app to be used. In addition, *PREPP* has a *temporal model* that estimates the appropriate time when the predicted app is likely to be used. Together, these two models enable *PREPP* to prefetch the application data over the network in a cost-aware manner such that a user is delivered the significantly fresh content while keeping the network bandwidth cost bounded. This increases the responsiveness of the mobile apps as the apps are already initialized and are ready with the up-to-date data.

1.6 Thesis Contributions

We now summarize the main contributions of this thesis.

1.6.1 Research Contributions

- First, we develop *RisQ* a data processing pipeline for gesture-based behavior recognition on mobile devices using a wristband equipped with inertial sensors. We identify

a set of discriminative features that can classify hand-to-mouth gestures like smoking and eating. We present a probabilistic model that analyzes sequences of hand-to-mouth gestures and infers which gestures are part of a smoking session. We demonstrate that *RisQ* can be used for accurate smoking behavior detection in ubiquitous settings.

- Next, we study the problem of accurate and energy-efficient context-sensing while reducing the privacy risks. Our contribution is *CQue*, a context-query engine that provides mobile applications with a query interface to obtain more accurate context classification while remaining agnostic of what classifiers and sensors are used and when these are used, and provides user with a way to specify what context they wish to keep private, and only allow information that has low privacy leakage to be revealed. We show that fusion of individual context classifier outputs by *CQue* can improve accuracy and reduce the energy requirements in continuous sensing of multiple user contexts.

- Finally, we design *PREPP* that improves the responsiveness of the mobile applications by predicting the application use, prelaunching and prefetching the application data over the network. We develop an app prediction model and a temporal model that has a low training overhead, adapts to changing app usage patterns and provides app usage predictions with high accuracy without using power-hungry contexts.

1.6.2 System Contributions

In addition to the research contributions, we developed various libraries and tools that are now available in public domain.

- **mStream Data Processing Library** This Java library implements data processing pipelines for various sensor modalities like ECG, accelerometer, electrodermal activity and inertial measurements unit. This library can be used to execute on Android devices as well as on web-servers.

- **mCrowdViz Data Visualization and Data Processing Tool** mCrowdViz is a web-based interactive application, enabling users to visualize and interact with the sensor data. This application integrates mStream data processing library to enable sensor data processing on the web-server and to visualize the output of the various stages in the data processing pipeline.

- **AppKicker Widget** AppKicker [16] is an Android widget available in Google Play Store. This widget implements the app prediction algorithm developed in *PREPP* to show the dynamic list of predicted applications as shortcuts to a user. This has several other prediction algorithms to choose from like Least Recently Used, Most Recently Used, and so on.

- **Gesture Monitor App** This is an Android application for data collection from a variety of external sensors including Invensense inertial measurements unit, Zephyr Bioharness chest band for ECG signals and Affectiva Q-sensor for electrodermal activity.

- **APPM API** This library provides an implementation of app prediction algorithm developed in *PREPP*. This can be used to predict next-likely event and compute likelihood of an event to occur within a some time-interval.

1.7 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 gives background and related work in the area of context-aware mobile systems and performance optimizations. In Chapter 3, we present a gesture recognition algorithm to detect smoking gestures and sessions in real-time on mobile devices using a wearable inertial sensor. In Chapter 4, we present *CQue* a dynamic bayesian network model for information fusion across context classifiers to improve accuracy and reduce the energy consumption in context sensing. Chapter 5 presents an app prediction model and a temporal model for prefetching application data

and prelaunching mobile applications to improve the responsiveness of the applications. We conclude with Chapter 6 which presents the summary of the thesis and discusses some future research directions.

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter presents a high level survey of literature related to the area of context-aware mobile computing to set the stage for our contributions. More detailed related work sections are also provided in the remaining chapters.

In this chapter, we first look at the classification mechanisms available to sense a variety of user contexts on phone. Next, we give an overview of the works relevant in the area of modeling correlations across user contexts. Lastly, we present a survey of techniques used for performance optimizations in context-sensing and network data access.

2.1 Context-aware Mobile Systems

In this section, we present state-of-the-art context-aware mobile systems with focus on the context inference mechanisms that can be used to extract high-level user contexts from the sensor data streams. The set of user contexts studied in the literature can be divided into following four broad categories with some overlap: i) physical activities, ii) health and behavior contexts, iii) ambient environment, and iv) semantic locations. In the following, we look at the related work for user contexts in each category.

2.1.1 Physical Activities

This category of user contexts deals with recognizing a phone user's activity i.e. 'what a user is doing?'. Activity recognition is often done using inertial sensors such as accelerometer and gyroscope and has been used to detect a wide array of human activities such as walking, standing, sitting, driving, running and so on [21, 36, 151, 188]. Apart

from the inertial sensors, sensors such as electrocardiograph sensor, heart-rate monitor, microphone and proximity sensors can be used in activity recognition. Lu et al [106] used microphone to detect a wider range of activities like *typing, brushing teeth, washing* and *clapping*. Rahman et al [150] showed use of respiratory signals collected from a wearable band worn around a user's chest to detect whether the user is speaking, listening or quiet. Lester et al [95, 96] and Annavaram et al [15] used multimodal sensing to recognize activities such as sitting, jogging, etc. to measure physical fitness. Annavaram et al [15] showed that using additional sensors such as heart-rate monitor is beneficial in discriminating similar activities such as *sitting* and *lying down* which are otherwise difficult to discriminate using a single accelerometer. Lester et al [95, 96] showed that a wearable device containing multi-modal sensors placed on a single body location, can be used for robust activity recognition. They reported high accuracy for activity recognition independent of the wearable device placement. Quwaider et al [146] used relative proximity between strategically placed sensors on a body to monitor body postures and physical activities. They showed that proximity sensors are better than accelerometer in detecting and discriminating sedentary activities like sitting and standing. Logan et al [102] studied activity recognition in the context of instrumented smart homes using infra-red motion detectors to detect activities like *using a computer, watching a TV* and *dish-washing*. Ganti et al [67] presented a wearable personal monitoring service embedded in user garments containing accelerometer and GPS sensors. They used Hidden Markov models for recognizing activities such as *walking with an umbrella* and *eating with a fork and knife*. Lymberopoulos et al [107] focused on extracting spatio-temporal physical activity patterns in an instrumented smart-home to create an spatio-temporal activity model for a person.

While activity recognition has been studied widely, there are significant challenges in achieving high accuracy across users. A part of the challenge comes from the fact that there exist variations across users performing the same set of activities. To address this challenge, Lane et al [92] introduced community similarity networks that incorporates

inter-person similarity measure to identify groups of similar users, and then crowd-source sensor data from these groups to personalize the activity classifiers for the users in the same group. Zhao et al [194] and Stikic et al [171] explored semi-supervised approach with active learning to improve the activity classification performance whereas Lester et al [97] used multi-modal sensing (accelerometer, audio, light, pressure, temperature, humidity) to improve the classification accuracy by capturing temporal regularities and smoothness of activities using Hidden-Markov models.

2.1.2 Health and Behavior Contexts

With the increasing availability of on-body wearable sensors that communicate with smartphones, sensing health and behavior contexts has gained a lot of attention from the research community. For example, wearable chest-bands that monitor physiological signals such as heart rate, ECG signals and breathing waveforms have been used to detect smoking behavior [12, 103], infer psychological stress level experienced by a user [142], detect cocaine use among addicts [126] and to observe sleep behavior [82]. Audio and visual cues have been shown to be useful in detecting behaviors such as emotions, mood, stress and well-being [75, 83, 93, 104, 147, 158]. Sensors that measure physiological signals such as skin conductance, skin temperature and heart rate are found to be more suitable for recognizing emotions or human affects. Various prototypes have been built that use these signals [2, 140] for various purposes such as recognizing stress [155], predicting user's response and rating for a movie [166] and detecting convulsive seizures [143]. Lane et al [91] developed 'BeWell' mobile application that captures three health behaviors, namely sleep, physical activity and social interaction on smartphones, and provides feedback to the users for their general well-being.

2.1.3 Ambient Environment

Context-awareness research is not limited to sensing a user's personal contexts but also includes sensing the user's surrounding environment and other people around the user. The

ability to sense the surrounding environment provides us with a greater understanding of how a user interacts with the surrounding environment. This has great applications in health and recommendation systems where the user behavior is influenced by the surroundings. User's social contexts such as *with friends, surrounded by people in a restaurant, in a party, in a meeting with colleagues* fall in this category of ambient environment contexts. A variety of approaches detect the ambient environment based on the signatures found in the sound signals [105, 109, 110, 139, 169]. These works demonstrate that the environmental noise provides a rich source of contextual information can be used for recognizing a wide range of *auditory scenes* such as *in a bus, in a car, in a subway train, at church, at a bar, in a lecture, bathroom, in a meeting, car turning signal, listening music, running water faucet* and so on. In addition to these contexts, there have been efforts to monitor potholes in the city using phone's accelerometer and GPS [7, 58, 118], to monitor road traffic conditions [88, 118] and braking detection [118]. PEIR [124] is a participatory sensing system that uses location data sampled from the smartphones to estimate the environmental impact and exposure. This system integrates external sources of information such as weather and traffic to calculate environmental impact and exposure. Typical ambient context-aware systems such as CenceMe [31] and EmotionSense [149] leverage a wide range of sensing modalities such as bluetooth, ambient light sensor, compass, microphone, accelerometer and location to detect a particular ambient context. METIS context-sensing platform [148] utilizes sensors embedded in the external infrastructure in addition to the sensors available on phones. The METIS system dynamically decides whether the sensing should be offloaded to the infrastructure or should be executed on the phone while considering the power consumption, accuracy and mobility of the user. On the other hand, CenceMe [31] and Common-Sense [56] achieve reduction in power consumption via participatory sensing where the users of the system share their sensed data with others.

2.1.4 Semantic Locations

The ability to sense a phone's geographic location using GPS on phone has resulted in a large number of apps that provide location-based services such as finding places of interests in the vicinity (e.g. Urban Spoon [9], Yelp [190]), getting driving directions to the place of interest (e.g. Google Maps [4]), sharing current location address with friends (e.g. Foursquare [65], Facebook [59]) and so on. In addition to GPS-based global localization, there have been efforts towards indoor localization where the goal is to find the location of a user carrying a phone within a building. The indoor localization techniques can be based on using FM [35], bluetooth [22, 42], sound and ambient light [18], WiFi [161, 162] or a combination of various sensors [183].

More recent research efforts have been towards converting raw location coordinates to semantic locations that are more meaningful to the user. These semantic locations are crucial for supporting easier location search and recommendation services that recommend semantic places such as near-by restaurants and movie theaters. Reverse geocoding is a popular technique where raw coordinates obtained using GPS sensor are converted into a human-readable address or name of the place. Reverse geocoding is used by social networking applications such as Facebook [59] and Foursquare [65] that allows users to share their semantic locations/places by checking-in via mobile applications. Ye et al [189] utilized the check-in behavior of the users to automatically annotate all the recorded places with category tags or semantic annotations relevant to those places. Other approaches [33, 101, 119] used traces of GPS data obtained from a large number of approaches to cluster GPS data and extract semantic locations from it in an unsupervised manner. SurroundSense [18] and CenceMe [31] extracted semantic locations using a variety of sensing modalities including sound, ambient light, WiFi, GPS, bluetooth and so on.

2.2 Relationship Modeling

In this section, we look at the related work in the area of modeling correlations across random variables. The ability to model correlations is particularly useful in context-awareness research as correlations across user contexts can help us gain insights into an individual user behavior and provide an understanding of how this user interacts with the surrounding environment. Our work on modeling context-relationships, builds upon the most relevant works in the area of sensor data networks and probabilistic databases. In the following, we give an high level overview of these works.

2.2.1 Graphical Models for Sensor Data Acquisition

There exists a substantial amount of work that leverages spatial and temporal models of correlations between distributed sensor sources to optimize sampling and communication in a sensor network [38, 52, 53, 98, 116]. For example, BBQ [53] uses a Dynamic Bayesian Network [50, 66] that is a special class of graphical model, to select minimum number of sensor nodes for data acquisition such that it can answer range queries within query-desired confidence bounds. Meliou et al [116] extended this work to sensor network routing where a query can be answered with desired confidence bounds while traversing the minimum number of nodes in a network spanning tree. Graphical models are also used in [52], which explored the problem of answering range queries while minimizing the energy cost of sampling sensors, and in [38], where sensors use models to reduce the communication costs by transmitting samples to a base station only when the ground truth is significantly different from the prediction made by the model.

Dynamic Bayesian Networks Dynamic bayesian networks (DBNs) [50] have found significant applications in modelling of stochastic temporal processes and in active and dynamic information fusion. In [68], the authors used DBN learned from experts for detecting human speaker from audio and visual cues. Choudhury et al [37] extended this work by proposing an algorithm that boosts the process of learning the DBN structure and

its parameters to improve the classification accuracy. Zhang et al [193] used DBN in the multi-sensor fusion problem to actively select the most relevant set of sensors for decision making. Forbes et al [64] looked at the problem of driving an autonomous vehicle requiring dynamic decision making where the DBNs were used to maintain the current belief state about the vehicle's world. The ability of the DBN to model temporal consistency had been used in forecasting [47], for example, sleep apnea [46]. In [23], authors used DBN for temporal modeling of emotional speech and to recognize the emotion. Nodelman et al [129] addressed the limitation of DBN that it cannot be used to model processes that evolve at different time granularities and present a continuous-time variation of DBNs. Most of these works rely on the ability of the DBN to maintain the current belief state of the world and use it in decision-making.

2.2.2 Graphical Models in Probabilistic Databases

A second related area is probabilistic database systems, which provide a way to store data with the associated uncertainty and allow querying over this uncertain data. An instance of a probabilistic database gives the probability distribution on all the certain database instances called *possible worlds* [48]. Prior work in this domain proposed to allow each tuple in the database to have an associated probability of belonging to the database [25, 48], and to extend this approach to model correlations across attributes and relations [160, 182]. The probabilistic models used in [25, 48] are based on restrictive assumptions that cannot be used to model complex relations as observed in user contextual information. PrDB [160] used *Bayesian Networks* as a compact way of representing the joint probability distribution of random variables and can be used to combine uncertain data from independent sources to compute posterior probabilities and give better estimates on uncertainty. BayesStore [182] used a similar approach, and defined query operators in probabilistic settings as well as algorithms for Select-From-Where queries. Gupta et al [72] used an undirected graphical model called Conditional Random Fields [90] in probabilistic

database to store and query the results of information extraction from the text documents. However, these systems were not designed for processing time-series of observations and do not model the temporal relationships. In general, the query processing systems over databases or streams have an inherent assumption that the source of information is fixed and reliable i.e. data is always available at all timestamps for all the relations or streams. As opposed to this, in context-sensing on phones, the number of streams providing data can change because the number of context classifiers undergoing execution may change due to energy constraints or sensor unavailability. Unlike work in probabilistic databases, context-sensing environment on phone also needs to control the components in the context processing pipeline, and decides when to switch on classifiers, and sensors, to ensure that queries are answered with required accuracy and delay while minimizing energy consumption. In contrast, work on probabilistic databases largely assumes that the sources of data cannot be controlled.

2.3 Performance Optimization

In this section, we look at the related work in the area of performance optimization. We focus on two lines of related works: i) optimization of context sensing performance, and ii) optimization of network access performance on mobile devices.

2.3.1 Optimizing Context Sensing Performance

In popular mobile operating systems like Android and iOS, the support for context-sensing had been limited to providing an interface [14] to the application developers that allows the mobile applications to sample data from various sensors available on the phone. The task of sensing and inferring contexts from the sensor data is left to the application developers. This approach has an adverse impact on the performance of mobile operating systems as context-sensing consumes precious resources such as CPU cycles, network bandwidth and energy on phones. The recent trend has been towards building context sens-

ing engines [40, 62, 106, 185] that provide context query interface to the app developers resulting in a clear separation between app development and context-sensing. This separation has the advantage that limited resources available on the phones can be managed more efficiently by the context-sensing engine while maintaining the balance between energy, latency and accuracy of context-sensing [40]. The energy efficiency in context-sensing is achieved by sharing the computational pipeline across contexts [40, 62, 106] and by offloading some of the computations to the cloud at opportune moments [40]. Acquisitional Context Engine (ACE) proposed by Nath et al [127] supports continuous context-aware applications by exploiting correlations across contexts to reduce energy consumption in sensing. ACE exploits context correlations to use already observed contexts to infer contexts that are correlated with the observed contexts, without requiring any additional sensor data.

Chu et al [39] argued for the integration of context-awareness in the mobile operating system itself as an OS can more effectively manage resources required for context-sensing. Moreover, this has an additional benefit of enhanced privacy as mobile apps are no longer required to sample raw sensor data and all the sensor data processing is delegated to the operating system. In recent developments, popular mobile operating systems like Android and iOS have now integrated activity recognition in the OS and provide an API to the app developers to receive phone users' activity updates. Both Android's activity recognition API [13] and iOS' Core Motion API [80] support detection of following activities: *in vehicle, on bicycle, on foot, still* and *unknown*. In addition, Android OS now provides logical sensors for step detection and counting steps during walking [14].

Apart from the works that optimize resource usage at a high-level of context-sensing pipelines, optimizations at the physical sensor level have been proposed [24, 63, 78, 136] to increase the execution time of the wearable physiological sensors that operate on small wearable-batteries. A common optimization used to increase battery life is to perform duty cycling [136] where a sensor regularly alternates between on and off states. Compressive

sensing [152, 174, 175] had been used to save energy in sampling sensor data by recovering a sparse signal from under-sampled measurements. Hua et al [78] proposed using "just-in-time" sampling to extend the sensor's battery life where the sensor is turned on at the most informative moments, based on the predictive mechanisms.

2.3.2 Optimizing Network Access Performance

The network connectivity (WiFi, 3G, LTE) on mobile devices have made these devices the indispensable accessories in our daily life with access to real-time dynamic content. However, increasing reliance on network connectivity for instant information comes at a cost of increased network latency and increased energy consumption in phones. *Prefetching* is a popular technique that has been used to address these challenges where some network content of interest to the user, is predicted and fetched over the network prior to its use by the user. Thus, reducing the latency observed by the user in accessing a network resource.

Prefetch is not a new concept and has a long history in systems research. Prefetching has been used widely in areas such as databases [49, 131, 168, 180] and filesystems [32, 71, 89, 94, 111, 121, 137]. Early works in databases and file systems research used prefetching to reduce the latency in accessing pages from the hard disk by loading the pages into memory before the pages were accessed. These approaches [61, 114, 115, 168] exploited the sequential nature of page reads by the file systems and the databases, to prefetch sequential pages into memory. As a consequence, these approaches have limited benefits for the class of applications where page reads are not sequential. To address this limitation, two distinct lines of research works were proposed in the literature. First, there are approaches that proposed *informed prefetching* where the applications themselves provide prefetching hints to the operating system in order to achieve efficient prefetching [32, 137, 121]. On the other hand, another line of works focused on algorithms to discover file access patterns and use these patterns to predict and prefetch pages in memory [71, 89, 94, 111]. Kistler et al [85] used prefetching in the context of distributed file sys-

tems to improve latency in accessing files and to maintain availability in case of network disruptions.

In a more relevant research domain of web-browsing on desktops, prefetching and caching have been in use for a long time in desktop web-browsers to reduce the web-page load times [10, 84, 112, 125, 130]. Modern web browsers such as Mozilla Firefox [122] support the use of explicit prefetching hints provided in the webpages to prefetch and cache the content [123]. However, the desktop based approaches are not directly applicable to the resource-constrained mobile execution environment as the resource constraints of network bandwidth and energy are not applicable to the desktop browsers. Moreover, user behavior on a desktop is considerably distinct from the mobile user behavior where the web access happens not only through the browser but also through a variety of mobile apps. This mobile user behavior is also determined by the location of the user as observed by Yan et al [186].

Prediction by Partial Match (PPM) [41, 117] is a Markov-model based prediction technique where Markov models of various orders are used to predict the next likely event to occur. A number of PPM-based approaches have been shown to give good results for prefetching and caching in the desktop browser environment [20, 45, 54, 60, 125]. Similar to other approaches for desktop browsing, these approaches do not take into account the resource limitations of the mobile environment but more importantly, these approaches do not take freshness of the prefetched-content into account. In other words, these approaches assume that the cached content does not go stale. This assumption does not hold true in the mobile environment where users seek the latest information and the information loses its value as the time passes.

In the area of mobile computing, a lot of network data prefetch-based strategies have been proposed in the literature to tackle the resource limitations and the network-latency issues [19, 51, 77, 108, 123, 145]. Higgins et al [77] argued for *informed prefetch* with support in mobile OS enabling third-party applications to provide explicit hints to exe-

cute prefetch. Lymberopoulos et al [108] proposed exploiting the spatiotemporal signatures associated with mobile web-browsing to predict and prefetch dynamic web-content, to achieve low web-access time and lower energy consumption. Qian et al [145] profiled third party applications' (e.g. Pandora music streaming app) and analyzed its data prefetch pattern to optimize the energy consumption in network data prefetch. Balasubramanian et al [19] observed that a lot of energy is wasted in 3G tail-time and developed a protocol to batch network prefetches across applications to amortize the tail energy costs. Deng et al [51] showed that energy consumption can be reduced by 66% in a 3G network by learning the traffic patterns to predict network activity, and make mobile device's radio state transition between active and idle state based on the predictions.

Timeliness of prefetch in mobile systems has been addressed in [17, 173, 177]. These are subscription-based approaches where a client/user subscribes to the web content of interest and the client gets updated in a timely manner while optimizing energy consumption and network usage incurred in the updates. Armstrong et al [17] proposed use of an edge proxy server that polls for the updates in contents of interest and pushes the updates to the client in batches to save energy. Similar approach is used in Android OS where the content-generating servers for mobile apps, push updates and notifications via Google Cloud Messaging Service [69]. The main benefit of this approach is that a mobile client is required to maintain a single persistent network connection with the Cloud messaging service instead of maintaining one connection per mobile app. Recently, iOS mobile operating system introduced background app refresh [79] where the operating system schedules an app to prefetch data at appropriate times. iOS determines the appropriate time to refresh an app by learning user's app usage patterns based on the location and the time of day contexts. However, to reduce the impact on battery life, these refreshes are scheduled only when the iOS device is connected to WiFi, plugged into a power source or when it is being actively used. Auto-update system proposed by Vartiainen et al [177] runs a low-priority daemon service on the client that manages periodic fetching of the contents of interest to

the user. The decision to fetch content is made based on the high-level profile selected by the user where each profile is associated with policies on content-fetching.

CHAPTER 3

RISQ: RECOGNIZING NATURAL HAND GESTURES WITH INERTIAL SENSORS ON A WRISTBAND

The ubiquity of mobile phones and the increasing availability of wearable sensors have sparked the interest of the research community to use mobile devices as a platform for health monitoring applications. In this chapter, we present *RisQ*, a mobile solution that leverages a wristband containing a 9-axis inertial measurement unit to capture changes in the orientation of a person's arm, and a machine learning pipeline that processes this data to accurately detect hand gesture-based behaviors, namely, smoking and eating in real-time. Our key innovations are four-fold: a) an arm trajectory-based method that extracts candidate hand-to-mouth gestures, b) a set of trajectory-based features to distinguish smoking gestures from confounding gestures including eating and drinking, c) a probabilistic model that analyzes sequences of hand-to-mouth gestures and infers which gestures are part of individual smoking sessions, and d) a method that leverages multiple IMUs placed on a person's body together with 3D animation of a person's arm to reduce burden of self-reports for labeled data collection. Our experiments show that our gesture recognition algorithm can detect smoking gestures with high accuracy (95.7%), precision (91%) and recall (81%). We also report a user study that demonstrates that we can accurately detect the number of smoking sessions with very few false positives over the period of a day, and that we can reliably extract the beginning and the end of smoking session periods.

3.1 Introduction

Tobacco use remains the single largest preventable cause of death and disease in the United States and worldwide. According to CDC estimates, cigarette smoking kills more

than 440,000 Americans each year, either through cancer, heart disease, stroke, or lung diseases. It also increases the chances of other serious illnesses, such as diabetes [1]. In addition, smoking-related illness in the United States costs $96 billion in medical costs and $97 billion in lost productivity each year. The numbers are more alarming worldwide, where tobacco use is increasing rapidly in low- and middle-income countries. In fact, it is estimated that there are about a billion smokers worldwide, with more than 80% in the low and middle-income countries. Of these, about 6 million die through smoking-related causes each year.

At the heart of addressing this scourge is early detection and timely treatment. Several smoking cessation programs have been developed that show that intervening at opportune moments can help a person quit smoking. In theory, continuous sensing using the mobile phone and wearables has the potential to enable such informed and timely interventions both by observing the evolution of a patient's smoking pattern over time, as well as measuring contextual factors that influence smoking (environment, social interactions, stress, etc). However, the first step towards such methods is the ability to detect smoking events in real-world settings, a goal that has proven elusive.

We argue that there is a dire need for a simple and reliable smoking detector that has high sensitivity and specificity, and is easy to wear on a day-to-day basis. Existing methods fall short on one or more of these axes. A sensitive detector for smoking is a tomography meter (e.g. CReSS monitor [43]). However, the user needs to attach the tomography device to the cigarette prior to smoking. This is both burdensome and can change the underlying smoking behavior. A recent alternative is to use the deep respiration cycles associated with inhalation and exhalation of smoke, which can be detected by a respiration chest band (mPuff [12] and Sazonov et al [103]). Unfortunately chest bands are cumbersome to wear for long periods and therefore have limited appeal beyond clinical trials. Other wearables that have been previously proposed include RF-based proximity sensors worn on the collar and wrist to detect when the hand is in the vicinity of the mouth, but this is not

robust to confounders [156]. One promising approach is to embed a sensor in a cigarette lighter which detects whenever the lighter is lit [157]. Unfortunately this approach does not provide information about individual puffs and its duration, which is particularly useful for determining the degree of nicotine intake across subjects [74, 76].

Our work relies on a wristband embedded with a single, low-power 9-axis inertial measurement unit (IMU) that fuses information from an accelerometer, gyroscope, and compass to provide 3D orientation of the wrist. IMUs are easy to integrate with wrist-worn wearables, many of which already have accelerometers embedded for calorie or activity tracking. In the rest of this section, we present the background of inertial measurement units and next describe the various challenges in robustly recognizing smoking gestures from the orientation data obtained from an IMU.

3.1.1 Background

Inertial Measurement Unit: The Inertial Measurement Unit (IMU) is an electronic device consisting of 3-axis accelerometer, 3-axis gyroscope and 3-axis magnetometer. IMU has an on-board processor that fuses the output of these three sensors to compute the orientation of the device in a 3D world (absolute) coordinate system defined by y-axis pointing towards the magnetic north, z-axis pointing perpendicular to the ground in the direction opposite to the earth's gravity and x-axis pointing to the geographical east. We use the Invensense MPU-9150 IMU for our experimental study. This sensor outputs the 3D orientation of a device in the form of a *quaternion*.

Quaternions: Quaternions are convenient mathematical entities for representing orientations and rotations of objects in three dimensions. Quaternions have found a great applicability in robotics, computer graphics, and computer vision communities due to the ease in calculations for performing rotations of objects in 3D space.

Formally, a *quaternion* is defined using a scalar component q_s and a 3D vector (q_x, q_y, q_z). We write a quaternion $\mathbf{q}$ as follows:

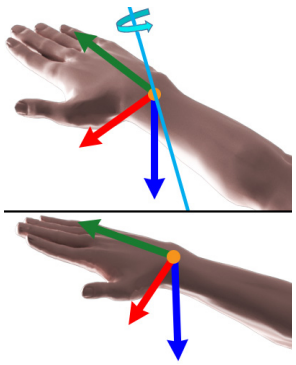


Figure 3.1. Top: 3D arm and its initial reference frame. A quaternion rotates the arm about a rotation axis (shown in cyan). **Bottom:** rotated arm and its updated reference frame.

$$\mathbf{q} = q_s + q_x\hat{i} + q_y\hat{j} + q_z\hat{k}$$

where $\hat{i}, \hat{j}$ and $\hat{k}$ are imaginary basis elements, each of which squares to -1. A quaternion $\mathbf{q}$ is said to be a *unit quaternion* if its magnitude $|\mathbf{q}|$ given by $\sqrt{(q_s^2 + q_x^2 + q_y^2 + q_z^2)}$ is equal to one. 3D rotations can be compactly represented with unit quaternions. Given a 3D rotation defined through a unit vector $\langle x, y, z\rangle$ representing the axis of rotation, and an angle θ representing the amount of rotation about the axis, the corresponding quaternion representing this rotation is defined as:

$$\mathbf{q} = \cos\tfrac{\theta}{2} + x\sin\tfrac{\theta}{2}\hat{i} + y\sin\tfrac{\theta}{2}\hat{j} + z\sin\tfrac{\theta}{2}\hat{k}$$

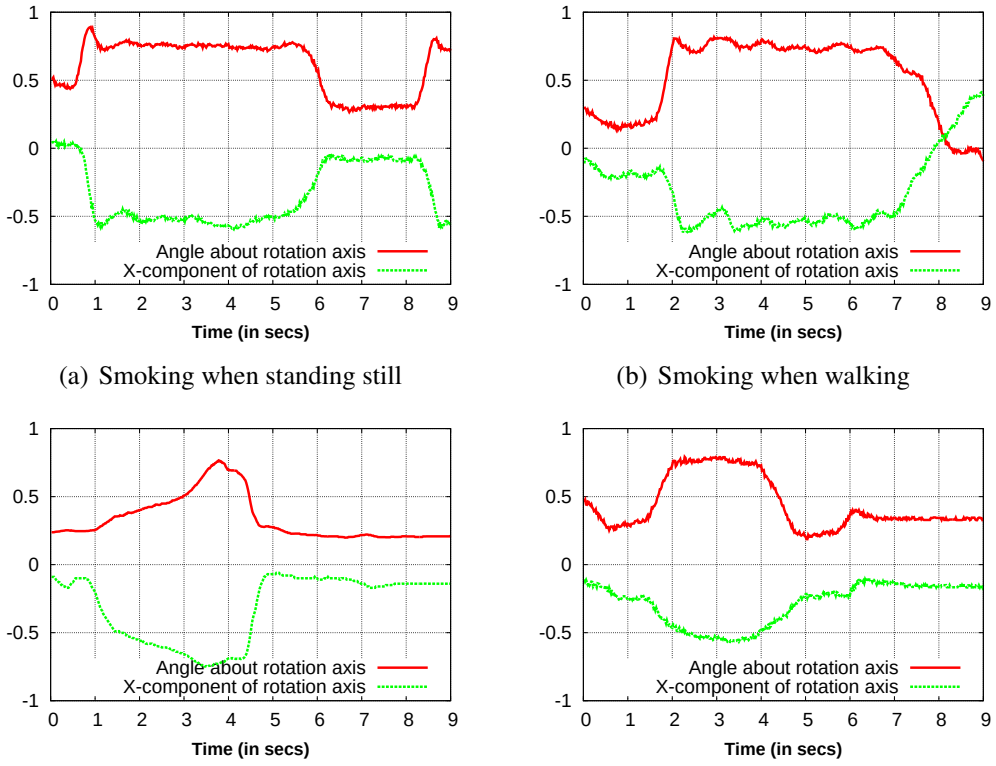
It can be shown that a point $\mathbf{p}$ in 3D space with coordinates $\{p_x, p_y, p_z\}$ can be rotated using a quaternion with the following formula:

$$\mathbf{p}' = \mathbf{q}\cdot\mathbf{p}\cdot\mathbf{q}^{-1}$$

where $\mathbf{p}$ is written as $p_x\hat{i} + p_y\hat{j} + p_z\hat{k}$, and $\mathbf{q}^{-1}$ is the so-called conjugate quaternion of $\mathbf{q}$, which is expressed as:

$$\mathbf{q}^{-1} = \cos\tfrac{\theta}{2} - x\sin\tfrac{\theta}{2}\hat{i} - y\sin\tfrac{\theta}{2}\hat{j} - z\sin\tfrac{\theta}{2}\hat{k}$$

It is straightforward to see that a quaternion not only represents a 3D rotation, but can also represent the 3D orientation of an object. Given an initial orientation of a 3D object defined by its initial frame of reference, the quaternion q rotates the object yielding a new orientation (see Figure 3.1).



(a) Smoking when standing still

(b) Smoking when walking

(c) Eating with a spoon

(d) Drinking from a cup

Figure 3.2. Illustration of IMU signals for various hand-to-mouth gestures. (a) & (b) Smoking gestures are characterized by a quick change in orientation when taking a cigarette towards the mouth and a long dwell time while taking a "smoking puff". (c) In contrast, eating gestures are characterized by a slow change when taking a spoonful of food towards the mouth and a very short dwell time. (d) The orientation pattern of drinking gestures (from a cup) lies in between the other two gestures with a relatively moderate rate of orientation change and a relatively lower dwell time compared to smoking gestures.

Figure 3.2 shows time series of the first two components of the quaternion data for various hand-to-mouth gestures, namely smoking, eating and drinking. In this figure, we can observe distinct characteristics that can be useful in distinguishing these gestures of differ-

ent types. However, there are many challenges in robustly recognizing smoking gestures from quaternion data as described next.

3.1.2 Challenges

The central challenge that we face is that we need to detect and recognize a smoking gesture from a plethora of other gestures that a user performs each day. While recognition of intentional gestures from inertial sensors is commonplace in gaming devices (e.g. Nintendo Wii), there are several challenges in achieving our goal in a natural setting.

Signal variation due to orientation changes: The first challenge is that the signal from the inertial sensor varies significantly depending on the user's body orientation. Thus, if the user changes his/her body orientation while smoking, the orientation output of the sensor will change. An example is shown in Figure 3.3, where a 180° change in the user's body orientation completely changes the signal. However, we still need to detect the beginning and end of a smoking gesture despite the unknown user's body orientation. Note that this problem is not present in gesture recognition systems designed for user interaction, such as the Nintendo Wii. This is because a user typically faces a fixed direction during the interaction. In addition, the user can consciously adjust the gesture so that it is recognized by the system.

Detecting smoking gestures: A second challenge is that a smoking gesture needs to be detected without any explicit information given by the user regarding the beginning and end of a smoking session. Devices such as the Nintendo Wii address this problem by having the user press a button, thereby explicitly providing information about the beginning and end of the gesture. Other gesture recognition systems such as the Bite-Counter [55] use a similar method: they assume that the user presses a button prior to an eating session. In contrast, we wish to avoid any user interaction altogether, and instead design a passive gesture tracking system. Thus, we aim at designing methods that distinguish a smoking

gesture from a huge number of different gestures that a user can perform using his/her hands.

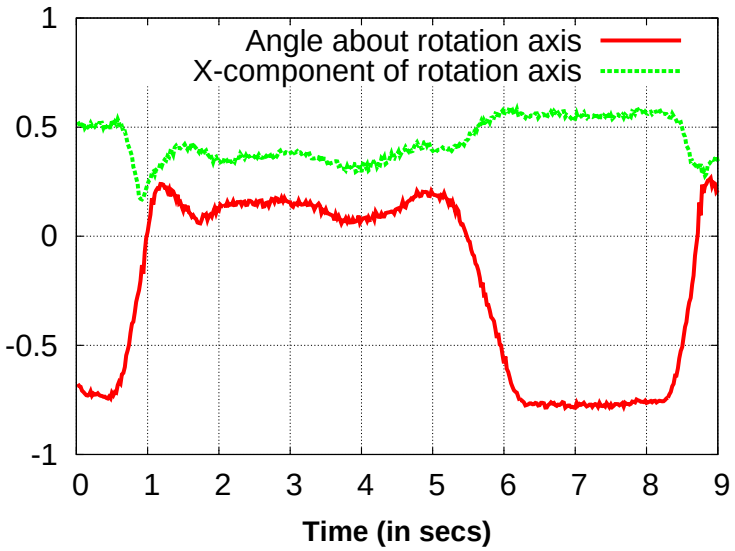


Figure 3.3. IMU signals for the smoking gesture in Figure 3.2(a) when the user changes facing direction by 180°. The characteristic patterns observed in IMU signals change with the user's orientation.

Concurrent activity while smoking: A third challenge is that the user might smoke while performing a concurrent activity. Concurrent activities can modify the characteristic patterns of smoking gestures and further complicate our problem. For example, Figure 3.2 shows how a smoking gesture looks when the user is stationary (Figure 3.2a) against when the user is walking (Figure 3.2b). We can see that smoking gestures have a few bumps when the user walks, in addition to the distortions at the beginning and end of the gesture where the user brings the cigarette down and continues walking by swinging his/her hand. Similarly, the user might be conversing with a friend while smoking, or might perform other activities while sitting or standing. All such concurrent activities also complicate the smoking gesture recognition problem.

Confounding gestures: A fourth challenge is that many other gestures have arm movement patterns similar to smoking. For example, a user can perform several isolated gestures

each day that involve raising the hand towards the mouth and bringing it back down similarly to smoking gestures. In particular, eating and drinking activities not only have similar hand-to-mouth gestures, but also involve repeated sequences of these gestures, as in smoking. Figure 3.2 shows an example of an eating gesture. Visually, there are differences in the pattern of eating compared to smoking. However, designing an algorithm that robustly distinguishes smoking from other confounding gestures across different users is particularly challenging.

Labeling smoking puffs: In order to recognize smoking gestures robustly, we design a method based on *supervised* classification. However, to train such a classifier, we need training data in the form of fine-grained labels for the beginning and end of each gesture. Many existing approaches for obtaining such labeled data are not feasible or have drawbacks: a) requesting self-reports from the users is impractical, since users cannot possibly label each smoking puff manually while smoking: such an approach would force the users to change their gestures from smoking to interacting with a device to create these self-reports, b) video capture of the smoking session for post-facto annotation is restrictive, since it requires the user to always face the camera, and c) having an observer annotate each smoking puff is cumbersome and not scalable. Thus, one particular challenge is how to enable fine-grained labeled data collection while limiting the burden on participants.

3.1.3 Contributions

The key innovation of our work is the ability to recognize sequences of hand gestures that correspond to smoking sessions "in the wild". This involves a sensing pipeline with several important aspects: a) we detect candidate hand-to-mouth gestures by continuously tracking and segmenting the 3D trajectory of a user's hand, b) we extract discriminative trajectory-based features that can distinguish a smoking gesture from a variety of other confounding gestures like eating and drinking, c) we design a probabilistic model based on a random forest classifier and a Conditional Random Field that analyze sequences of hand-

to-mouth gestures based on their extracted features, and accurately outputs the beginning and end of smoking sessions. Finally, (d) in order to train the classifier and Conditional Random Field, we also propose a simple method to gather training labeled data "in the wild": we ask subjects to wear two IMUs, one on the elbow and one on the wrist, which allows us to build 3D animations of arm movements that can be easily labeled by a third party without compromising the subject's privacy. The only limited burden on the subject is to specify coarse time windows where smoking occurred to verify that we detect gestures within the correct periods of time. Each module in this whole pipeline addresses challenges specific to our problem domain, and we present several new ideas and algorithms.

Our results show that we can detect smoking gestures with 95.7% accuracy and 91% precision. In addition, we can detect smoking session time boundaries reliably: the error in the estimated duration of a smoking session is less than a minute. Finally, we demonstrate with a user study that we can accurately detect the number of users' smoking sessions in the period of a day with only few false positives (less than two in our study). In all, we think that *RisQ* is a very promising approach for use in smoking cessation and intervention.

3.2 Related Work

In this section, we describe the three areas of related work — gesture recognition in computer vision, gesture recognition and motion capture using wearable inertial sensors, and smoking detection using special-purpose devices.

Computer Vision: This line of research deal with the problem of gesture detection and recognition using a camera. The camera is used to track a person and the motion of the arm. This gives the coordinates of various points on the body that can be used as features for gesture recognition. Yang et al. [187] used Hidden Markov Models (HMM) for segmenting and identifying the set of gestures for human-robot interaction. Elmezain et al. [57] proposed a method to use Conditional Random Fields to separate gestures from non-gestures without needing to train for non-gestures. Wang et al. [184] combined HMMs with CRFs

in a Hidden Conditional Random Field that can be used as a single gesture class detector or a multi-way gesture classifier. All these methods employ similar machine learning formulations, however, recognizing arms accurately in images or video is prone to errors due to occlusions, clothing, illumination changes and other errors in pose tracking from 2D images. These computer vision methods also assume that the user is always expected to turn towards the camera. Yin et al. [191] use HMMs for gesture recognition but it requires a use of special glove to enable the tracking of the arm in a cluttered background.

Wearable sensors: The advances in wearable sensors technology has generated a lot of interest in gesture recognition and other novel uses of these sensors. Vlasic et al. [181] proposed a full body motion capture system using inertial sensors. Such systems are useful to capture a person's full body motion, but require several such sensors, acoustic sensors, careful calibration and initial body pose estimates. Agrawal et al. [11] proposed *PhonePoint Pen* to enable users to use a mobile phone as a pen to write in the air using an accelerometer in the phone. This work identifies six basic strokes in the English characters that are essentially straight lines and two half-circles. These strokes are identified using correlation with the ideal strokes. *u*Wave [100] uses a dynamic time warping (DTW) technique to recognize gestures from the accelerometer present in Nintendo Wii. DTW is used to match a temporal sequence with a given template, when the number of templates is small. Chen et al. [34] use a 6-axis inertial sensor to track interactive gestures and rely on *push-to-gesture* mechanism to spot the gesture segment. All these approaches detect "control" gestures that have little variations across users. Dong et al. [55] proposed Bite Counter to detect food bites in an eating session using a gyroscope. This approach requires a user to press a button to indicate the start and the end of an eating session. We compared this with our approach and found it to be less useful for smoking detection. Varkey et al. [176] train two SVMs: first to detect a high-level activity in a window of inertial sensor data and another to detect micro-activities within a window. They evaluated their approach for smoking detection using a synthetic dataset provided by non-smokers who imitated the smoking gestures. Their approach for

recognizing puffs relies on detecting two consecutive micro-activites 'arm moving up' and 'arm moving down'. They showed that the cumulative misclassification for micro-activity detection within smoking sessions is around 20%. This approach can lead to a significant error in estimating the number of puffs and the puff duration and may perform worse if the data from confounding gestures is included. This work has not been evaluated for data in the wild and it is unclear if this approach can be used to detect smoking session boundaries reliably.

Smoking Detection: A sensitive detector for smoking is a tomography meter (e.g. CReSS monitor [43]). However, the user needs to attach the tomography device to the cigarette prior to smoking. This is both burdensome and can change the underlying smoking behavior. A recent alternative is to use the deep respiration cycles associated with inhalation and exhalation of smoke, which can be detected by a respiration chest band (mPuff [12] and Sazonov et al [103]). Unfortunately chest bands are cumbersome to wear for long periods and therefore have limited appeal beyond clinical trials. Other wearables that have been previously proposed include RF-based proximity sensors worn on the collar and wrist to detect when the hand is in the vicinity of the mouth, but this is not robust to confounders [156]. One promising approach is to embed a sensor in a cigarette lighter which detects whenever the lighter is lit [157]. This does not provide information about individual puffs, which is particularly useful for determining the degree of nicotine intake across subjects [74].

3.3 System Overview

In this section, we provide an overview of the *RisQ* computational pipeline that recognizes smoking gestures and sessions. We also overview the training data collection methodology we used to obtain fine-grained labeled data for the purpose of training our supervised classification method.

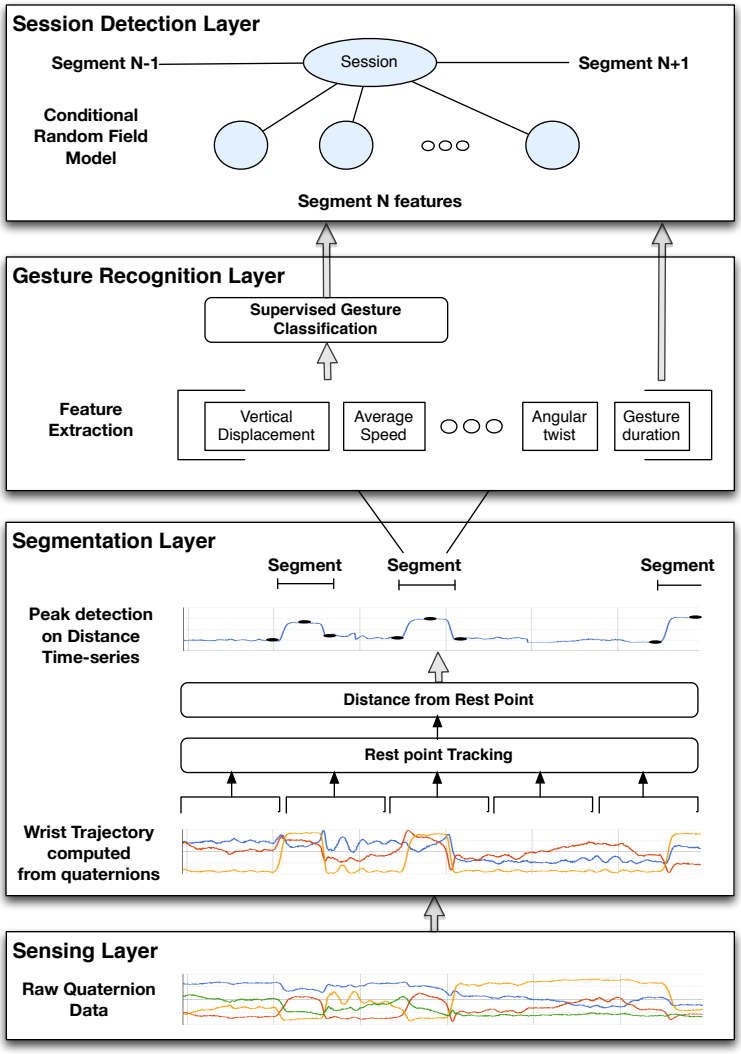


Figure 3.4. *RisQ*: Quaternion data processing pipeline

Computational pipeline: Figure 3.4 gives an overview of the computational pipeline that we use for detecting smoking gestures and sessions. At the lowest layer of the pipeline is the extraction of quaternion data from the single wrist-worn 9-axis IMU.

The second layer in the pipeline is the segmentation layer that extracts segments containing candidate gestures from the raw sensor data and filters out extraneous data. The intuition behind the segmentation process is that, while performing a hand gesture, humans start from "a rest position" in which the arm is relaxed, then move their arm, and finally

the arm falls back to another, possibly different rest position. Thus, the gestures tend to lie in segments between these resting positions. The segmentation layer accomplishes the segment extraction by computing the spatio-temporal trajectory taken by the wrist using quaternion data and tracking rest positions. Based on the computed trajectory, at each time step it computes the distance of the wrist from rest positions, and identifies segments using a peak-valley detection algorithm. The output of this layer are the extracted segments containing candidate hand gestures.

In the third layer of our pipeline, our method computes a feature vector for each segment, consisting of features that can discriminate hand-to-mouth gestures corresponding to smoking from a large number of other hand-to-mouth gesture candidates. We then use a supervised classifier, called Random Forests [29, 44], that outputs the probability for the type of gesture ("smoking", "eating" or "other" gesture) based on the extracted features for each segment individually.

The top-most layer in the processing pipeline is a session detection layer that identifies whole sessions ("smoking", "eating", or "other") based on the Random Forest classifier outputs and the segment features. The key intuition behind this layer is that each smoking session involves a continuous sequence of smoking gestures, and is unlikely to contain gestures from other activities. We use a probabilistic model based on a Conditional Random Field (CRF) [90] that captures probabilistic dependencies between the type of gestures contained in consecutive segments. In particular, it has the advantage of filtering out spurious classification outputs of the lower layer and robustly detecting the beginning and end of smoking sessions.

While the focus of this chapter is on recognizing smoking gestures and sessions, our approach can be tailored to other instances of repetitive gestures. In §3.6.3, we use eating sessions to train our method, and we find that it performs considerably better than state-of-art eating detectors that use inertial sensors.

Labeled data collection: We use a novel data collection methodology to obtain fine-grained gesture labeled data from participants for training our method. Our data collection framework consists of two tools: (i) a logging application and (ii) a 3D visualization and labeling tool. The logging application collects data from on-body IMU sensors placed on the user's wrist and upper arm. Since our goal is to reduce the burden of labeling for the user and avoid interfering with his gestures, our logging application provides a simple user-interface to log events on his mobile phone. This helps us identify time windows that require fine-grained labeling of the smoking or eating gestures. The 3D visualization and labeling tool produces 3D hand animations from the logged sensor data and provides an interface for fine-grained labeling of the gestures, such as "smoking a puff" or "taking a bite of food". The type of gestures can be easily labeled by other users in these animated sequences. Unlike video recording, our visualization tool allows users to see and label gestures from any viewing angle they prefer.

3.4 Gesture Detection and Recognition

In this section, we describe how we detect and label hand-to-mouth gestures when the user wears the single IMU on his wristband. The first step in our data processing pipeline is to partition the quaternion time-series obtained from the IMU into segments such that each segment contains a gesture (§3.4.1). The second step is to extract some features that can be used to discriminate different types of gestures (§3.4.2). The last step is to recognize the detected gestures and extract smoking sessions (§3.4.3).

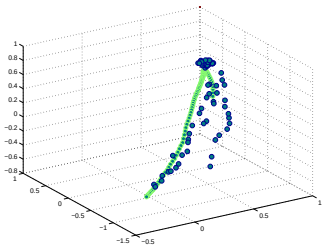
3.4.1 Gesture Detection

In order to detect a gesture, it is essential that we segment the time series of quaternions into segments, such that each segment contains a complete gesture. The problem of correctly segmenting sequences of quaternions presents a number of challenges. First, the segmentation should extract segments representing the entire gesture duration. Otherwise

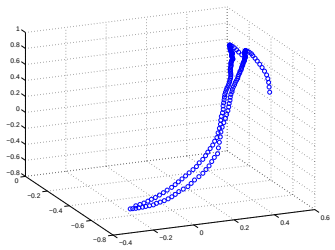
characteristic features that are useful in classifying a gesture will be lost. Moreover, the extracted segments should not be much longer than the gesture, otherwise the computed features related to the gesture will be inaccurate. Another fundamental challenge is that the characteristic patterns observed in a time series of quaternions are dependent on the orientation of the person's body performing the gesture. For example, the pattern observed for a gesture when the person is lying on a bed is very different from the pattern observed when he is standing. The problem gets further complicated due to the large number of variations in the gestures observed for the same activity, within and across users, making the variability of characteristic patterns extremely high. An additional challenge comes from the huge number of all possible gestures and possibly confounding gestures performed by humans throughout their daily life.

The intuition behind our segmentation approach is the observation that the humans tend to perform a gesture starting from an initial rest body pose, and then ending the gesture in another, possibly different, rest pose. This observation holds true for the hand-to-mouth gestures we intend to identify. For example, a person may start a smoking gesture by having his hand lie on an armrest of a chair, then move his hands towards his mouth, then back to the armrest of the chair or the top of a table. Alternatively, the arm might end up being straight, hanging from the body. In general, the beginning and end rest poses are not necessarily stationary and are not necessarily identical. In the following sections, we describe our segmentation approach based on continuous tracking of the rest positions of the fore-arm and computing the spatio-temporal trajectory of the wrist in 3D space.

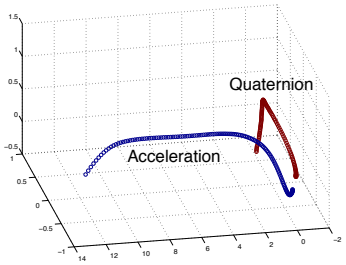
Relative trajectory computation: Our first step towards identifying hand-to-mouth gestures is computing the spatio-temporal trajectory of the hand from the stream of quaternion data. The main challenge here is that trajectory is affected by body motion and orientation. For example, a smoking gesture when a person is walking takes a much longer path in 3D space compared to the same gesture performed when the person stands at one place. Sim-



(a) Trajectory smoothing using curve-fitting



(b) Trajectories for two consecutive smoking puff gestures



(c) Quaternion and Acceleration-based trajectories for the same gesture when the user is walking

Figure 3.5. Figure (a) shows a noisy trajectory for a smoking puff gesture and the smooth trajectory (green curve) obtained after curve-fitting. Figure (b) shows trajectories for two consecutive puffs while standing still. These two trajectories are well-aligned for the isomorphic gestures. Figure (c) shows the trajectory for a smoking puff gesture when the user is walking, computed from acceleration data (blue curve) and quaternion data (brown curve). The acceleration-based trajectory is significantly elongated and distorted when the user is walking, while holding the cigarette in his mouth. The quaternion-based trajectory instead is computed relatively to the elbow and is free of such motion artifacts.

ilarly, a smoking gesture could be affected by any changes to the pose of the user's body while smoking.

The key insight in our approach is that we care less about the absolute trajectory of the hand in 3D space. We care more about the trajectory of the hand relative to a body joint, such as the shoulder joint. Such a relative trajectory would not depend on the orientation of the person's body, and every smoking gesture should result in roughly isomorphic trajectories with similar shape characteristics.

While such a relative trajectory would be easy to compute if the individual also had an IMU close to the shoulder, the fact that we only have a single wrist-worn IMU complicates

the problem. To address this, we make a simplifying assumption that the position of the elbow remains stationary (the shoulder joint does not rotate) while the gesture is being performed. Specifically we compute the trajectory of the wrist relative to a stationary elbow, assuming that the wrist is in a fixed, unit distance away from it. We find that the above assumptions have little influence in our trajectory computation, since hand-to-mouth gestures primarily involve rotating the elbow and not moving the upper arm.

More formally, given a time sequence of quaternions ($\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$), we can compute the trajectory taken by the wrist by computing the position of the wrist for each time step t ($t = 1...n$). The position of the wrist $\mathbf{w}$ at time t in the elbow's frame of reference $\mathcal{F}$ can be computed as follows:

$$\mathbf{w} = \mathbf{q}_t \cdot \mathbf{w}_0 \cdot \mathbf{q}_t^{-1} \tag{3.1}$$

where $\mathbf{w}_0 = 0\hat{i} + 1\hat{j} + 0\hat{k}$ (i.e., has coordinates $\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$) represents the position of the wrist in the device's local coordinates based on the assumption that the length of the forearm of person has unit length. This assumption also works in our favor as the computed trajectories are independent of the particular variations of the arm length across the population.

Due to this approximating nature of the computed trajectory and the continuous shifts in the elbow position, the wrist trajectory does not always appear smooth. Thus, we smooth the trajectory by fitting a curve using cubic B-splines. Figure 3.5(a) shows a sample trajectory for a smoking puff gesture before and after the smoothing operation.

IMU vs Accelerometer: One question that is worth asking at this point is whether we really need a 9-axis IMU that provides quaternion data (by fusing information from the accelerometer, gyroscope, and compass) to compute trajectories, or whether we can do the same with a single accelerometer. Figure 3.5(b) shows two smoothed sample trajectories taken for two consecutive cigarette puffs while standing still. These trajectories are computed relative to the elbow and align well as the gestures are isomorphic. In Figure 3.5(c), we show the accelerometer-based and quaternion-based trajectories for the smoking

gesture when the user is walking. We see that the accelerometer-based trajectory in this case appears longer. This is because it is relative to the starting point in world coordinates. On the other hand, the trajectory computed using quaternions is relative to the elbow has similar characteristics to trajectories observed when standing still.

Segment Extraction: Our segmentation procedure is based on the observation that humans tend to keep their hands in rest positions and any gesture starts from one rest position and terminates at another rest position. The key challenge is determining these rest positions in a continuous stream of wrist movement. The rest positions need not be the same across gestures (e.g. rest position while sitting and smoking would be different from standing and smoking). Even the beginning rest position for a single smoking puff might be different from the final rest position, since there can be many positions where the human arm is "relaxed".

Our segment extraction method involves three stages:

- First, our segment extraction method continuously tracks the rest position from which a current hand gesture started. To detect the rest position, the trajectory is traced according to short sliding time windows (e.g. 10 seconds). Within each window, positions where the wrist velocity is very low are computed. If one or more candidates exist, then we compute the centroid of these low velocity points as the "rest point". Thus, we can now annotate each point in each trajectory trace with the most recent point corresponding to the rest position for which the hand gesture started.
- Second, we compute the spatial distance of each point in the trajectory from the most recent rest point. For any hand-to-mouth gesture, we expect that the distance from the rest point should increase rapidly, plateau for a short period, and then return to the next rest point.
- Third, we use a peak detector to detect peaks and troughs in the distance time series obtained from the previous stage, and look for a tell-tale trough-peak-trough pattern that is typical of a hand-to-mouth gesture. To filter false positives from either

tiny hand movements or very long gestures, we use a relatively loose threshold for distance separation between trough and peak, as well as the expected duration of a typical smoking puff. At this point, the segment extraction pipeline provides the trough-peak-trough segment to the next level of the classification pipeline.

3.4.2 Segment Feature Extraction

Table 3.1. Feature set extracted for a segment containing a gesture. The table describes each feature type and gives the count of features used for each type.

Feature Set		
Duration Features		
Duration	4	Total gesture duration, and duration for the ascending, descending, and the stationary stage.
Velocity Features		
Speed	6	Mean, max and variance of the wrist speed. We compute these for the ascending and the descending stages.
Displacement Features		
distZ	2	Vertical displacement of the wrist during the ascending and descending stage.
distXY	2	Horizontal displacement of the wrist during the ascending and descending stage. This is computed on a plane parallel to the ground.
dist	2	Net displacement between the rest position and the peak position. This is computed for the ascending and the descending stage.
Angle Features		
roll velocity	4	Median and maximum angular velocity about the axis of the arm. We compute 2 features each for the ascending and the descending stage.
roll	2	Net angular change about the axis of the arm during the ascending and the descending stage.
pitch	12	Angle between the arm and the direction of the gravity (pitch angle). We compute the pitch angle at the peak during the ascending and descending stage. Also, we obtain 9 decile features from the pitch distribution for the ascending stage. We compute the median pitch angle for the descending stage.

In order to recognize the type of gesture in each segment, we need to compute features that can discriminate different types of gestures. The feature extraction procedure is based on the observation that a hand-to-mouth gesture can be divided into three stages: i) an "ascending" stage that corresponds to the trough to peak movement when the hand goes

towards the mouth, ii) a "stationary" stage that corresponds to the period where the hand stays near the peak (close to mouth), and iii) a "descending" stage where the hand goes from peak to trough i.e. back to the rest position. These three sub-segments are useful for extracting features as we describe below.

Table 3.1 shows the set of features computed for each segment. The following paragraphs describe the features we compute from the smoothed spatio-temporal trajectory and the quaternion information:

- **Duration features.** These features measure the time duration of the segment containing the gesture and the durations of the three stages within the segment: ascending, descending and stationary.
- **Velocity features.** Second, we use the computed positions of the wrist at each time step to compute the instantaneous velocity of the wrist for the ascending and the descending stages. We compute the average speed, the maximum speed and the variance in speed for both ascending and the descending stages giving us six features.
- **Displacement features.** These features measure the displacement of the wrist during the ascending and the descending stages of the wrist motion. We measure the vertical displacement, displacement on the plane parallel to the ground and the net displacement at the end of each stage as features.
- **Angle features.** From the orientation information, we can compute the *roll* component of the rotation about the axis of the arm. This is computed by converting the quaternions into Tait-Bryan angles. We call the rate of change in the angle of the rotation along the axis of the arm "roll velocity". We compute the median and the maximum of instantaneous roll velocities and use them as features. Also, we compute the net roll angle as the feature. We extract these 3 features for the ascending and the descending stage. Next, we compute "pitch" that measures the angle between the vector along the arm and the direction of the gravity. We use pitch angle at the peak point of the ascending and the descending stages as features. Next, using the

distribution of pitch angles observed at each time step for the ascending stage, we compute 9 values that give the $10^{th}, 20^{th}, ..., 100^{th}$ percentile values as the features. For the descending stage, we use median pitch as a feature.

3.4.3 Gesture Recognition

After extracting segments that contain individual gestures, the next step in our pipeline is to recognize each gesture contained in each segment. The input to this step is a set of segments with their features and the output is a set of labels representing the type of gesture (e.g. "smoking", "eating", or "other" depending on available labels).

Labeling segments poses a number of challenges. First, there is no simple mapping between the duration, velocity, displacement and angular features and the labels. For example, if the duration of the gesture is more than a particular threshold, we cannot infer that the gesture is a "smoking puff" with total certainty. It might be the case that the user instead drinks slowly from a cup. Therefore, we have to also consider other features: for example, smoking is likely to also be strongly correlated with certain ranges of arm twisting angles, wrist velocities, horizontal and vertical displacements and so on. Manually constructing a mathematical formula that detects all smoking gestures based on features is very hard. Noise in the trajectory features make things worse, thus we need to adopt a probabilistic method that outputs the probability of assigning each label to each segment. In the following paragraph, we describe a probabilistic classifier for recognizing each individual gesture. Unfortunately, classifying each gesture independently from all the others in a sequence of segments does not seem to work very well. For example, a sequence of gestures might be labeled as $\{smoking, smoking, eating, smoking, smoking\}$. However, it is very unlikely that the user interrupts and intermixes gestures in a small period of time. Such noisy labelings can easily occur due to feature noise and large deviation of wrist trajectories from the ones in our training data. Thus, we employ a probabilistic model that takes as input the probabilistic output of the classifier for each individual segment, and

additionally examines how likely is for successive segments to have the same or different label. Based on this probabilistic model, we estimate the most likely joint labeling of all segments together. We describe this probabilistic model later in this section.

Individual classification of gestures: To build a mapping from segment features to segment labels, we observe that the type of gestures is likely to be correlated with certain value ranges of segment features extracted in the previous section. For this reason, we use a classifier that has a form of a decision tree. A decision tree contains a set of nodes and at each node, the value of an individual feature is compared to a threshold. Then depending on whether the feature of the segment has a smaller or larger value than the threshold, the classifier examines the left or right child node, which in turn examines the value of another feature. When a leaf node is reached, the decision tree outputs a probability based on how many segments in our training data had the same set of splits while traversing the tree.

The thresholds and feature choices at each node of the tree are automatically constructed from the training data during an offline stage. The best choice of features and thresholds are selected to maximize the confidence of predictions [44].

Unfortunately, fitting a single decision tree in the training data results in poor performance. The reason is that a single set of splits yields predictions that are suited only for segments whose features are very similar to the ones of the training segments. A popular alternative is to build an ensemble of decision trees: each decision tree is fitted to small different random subsets of the training data. This leads to decorrelating the individual tree predictions and, in turn, results in improved generalization and robustness [44]. This ensemble of decision trees is called random forest classifier. The output of the random classifier is the probability of assigning a label to each segment, computed as the average of the probabilities outputted by the individual decision trees.

Joint classification of gestures: As we explained in the beginning of the section, classifying each segment independently from all the others in a sequence of segments yields noisy label predictions. Instead, we classify all segments jointly by using the output of the

random forest and also considering that consecutive segments in a sequence are more likely to have the same rather than different label.

Given a sequence of $k = 1...K$ segments, we introduce a random variable C_k for each segment representing the type of gesture contained in the segment. We express the joint probability of assigning a sequence of labels to all the random variables $\mathbf{C} = \{C_1, C_2, ..., C_K\}$ of the segments given their features $\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_K\}$ as follows:

$$P(\mathbf{C}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{k} \phi(C_k, \mathbf{x}_k) \prod_{k,k+1} \psi(C_k, C_{k+1})$$

The unary factors $\phi(C_k, X_k)$ assess the consistency of each individual segment to each label based on the random forest classifier. The pairwise factors $\psi(C_k, C_{k+1})$ assess the consistency of each pair of consecutive segments to their labels. The denominator $Z(\mathbf{x})$ serves as a normalization constant for each input sequence to ensure that the sum of probabilities over all possible label assignments is equal to 1. The type of this model is known as Conditional Random Field [90] in the literature of machine learning.

In our implementation, the unary factors have the following form:

$$\phi(C_k = l, \mathbf{x}_k) = \exp(\theta_l f_l(\mathbf{x}_k) + \theta_{l_0})$$

where $f_l(\mathbf{x}_k)$ are the probabilistic outputs of the random forest classifier for each label $l = \{smoking, eating, other\}$. The parameters $\{\theta_l, \theta_{l,0}\}$ re-scale the probabilities of the random forest classifier, and are learned from the training data. The $\exp$ function ensures that the final result will lie in the interval $[0, 1]$ (i.e., will also be a probability) after applying the normalization constant, as typically done in logistic regression and log-linear CRF models [172].

The pairwise factors evaluate how likely is to assign pairs of labels in consecutive segments and are expressed as:

$$\psi(C_k = l, C_{k+1} = l') = \exp(\theta_{l,l'})$$

The parameters $\theta_{l,l'}$ are also learned from the training data. The learned parameters turn out to favor the same label for consecutive segments, as expected.

Parameter learning: It is extremely hard to hand-tune the parameters $\boldsymbol{\theta} = \{\theta_l, \theta_{l,0}, \theta_{l,l'}\}$ in the above probabilistic model. Instead, the model parameters are learned such that they maximize the likelihood of the training sequences. Specifically, the parameters are estimated to maximize the following objective function [86]:

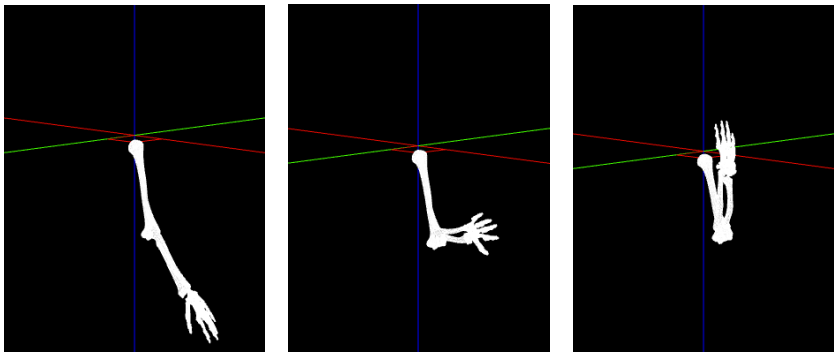
$$L(\{\lambda_k\}) = \sum_{n=1}^{N} \log(P(\mathbf{C}[n] \mid \mathbf{x}[n])) - \mu||\boldsymbol{\theta}||^2$$

where the first term expresses the log likelihood of the training sequences given their features. The second term is a regularization term that penalizes large values in the parameters. Such large values would favor over-fitting of the parameters to the training dataset, and are less likely to generalize to new data. The parameter μ is selected through cross-validation.

The objective function is maximized using the L-BFGS method [99] which is a popular optimization method for solving unconstrained optimization problems. The objective function is convex function, thus, a unique maximum exists [86].

Inference: Given a sequence of segments, we estimate the most likely joint assignment of labels to the segments based on the above model. From the assigned labels, we can simply extract the different gesture sessions by tracking when the label changes in the sequence of segments.

Inference is performed with the technique known as max-product belief propagation [86], which yields the most likely sequence labeling in this type of model. The complexity of the evaluating the unary and pairwise terms, as well as executing belief propagation is linear in the number of segments, thus the labelings can be predicted very efficiently.



(a) (b) (c)

Figure 3.6. Smoking gesture visualization using sensor data obtained from the two IMUs placed on the upper arm and the wrist.

3.5 Labeled Data Collection

In this section, we describe the framework used to capture ground truth data in uncontrolled environment settings with a low labeling overhead for the user, and the dataset that we obtained through this approach. The main idea is to have the subject wear two inertial sensors, one at the wrist, and one at the elbow, and segment the data post-facto through a 3D visualization tool. We note that the use of two sensors is used only for gathering training data. During actual usage of *RisQ* (i.e., during testing), the user always wear a single IMU on a wristband, which is more practical, user-friendly, and less costly.

3.5.1 Labeling Framework

Sensors: We use Invensense MPU-9150 IMU to obtain orientation data sampled at 50Hz. We fit a user with a wristband and an armband, each containing an IMU device and use a mobile app to log the sensor data on the phone. Our goal is to reconstruct the hand gestures performed by a user using the logged data.

Constructing hand gesture: A human arm consists of an upper arm and a forearm. The upper arm is connected to the shoulder, and is connected with the forearm at the elbow. The forearm extends from the elbow to the wrist. For constructing the hand motion, we assume that the orientation is uniform for the entire forearm and the entire upper arm.

This assumption is simplistic as the orientation of the forearm close to the elbow can be different from the orientation of the wrist but this assumption enables us to reconstruct a fairly accurate hand motion with fewer sensors.

In order to reconstruct the hand motion, we use a digital model of a 3D human skeleton in a polygon mesh format (Figure 3.6). The 3D model consists of skeletal limbs that that can be oriented independently by applying the orientation information in the form of quaternions to the shoulder and elbow joints of the skeleton. The orientations given by the quaternions are applied to the skeleton hierarchically as in skeleton-based animation methods commonly used in computer graphics applications. Given that we log the sensor data at 50Hz, we render the 3D model at 50 frames per seconds where we draw a new pose of the 3D model every frame to enable real-time playback. Rendering is implemented in Processing 2, a popular library used in computer graphics.

Labeling: Using the 3D animation and visualization method described above, we can now mark the start and the end of a gesture like a cigarette puff. However, manually visualizing and marking the gesture for hours of data can be a time-consuming exercise, and in the absence of any additional information, it is difficult to distinguish smoking or eating from any confounding gestures that may look similar. To address this, we ask the participating users in the data collection to mark the beginning and the end of the smoking and eating periods using a mobile app. For fine-grained gesture labeling, we limited our efforts to the marked periods.

3.5.2 Gesture Dataset

Our dataset consists of IMU data collected for over 32 hours from 15 volunteers for over 20 smoking sessions, 15 eating sessions, 6 drinking sessions, and a variety of other gestures. Users typically wear sensors for several hours, therefore, our dataset contains many potential confounding gestures. Of this data, we had to discard a few smoking and eating sessions due to interruptions and extreme noise in the data. This left us with 28 hours

of data, containing 17 smoking sessions and 10 eating sessions. Overall these sessions include 369 smoking puffs and 252 food bites. The data included users performing several concurrent activities including smoking while standing alone, smoking in a group while having a conversation, smoking using a hand-rolled cigarette and smoking while walking around.

We used our 3D visualization and labeling tool to hand-label the start and end of each smoking puff within the smoking periods reported by the participants. There are a few instances when defining a precise start or a precise end of a gesture is not possible. Since our focus is on detecting a smoking puff, any interval that begins at least 1 second prior to the puff and ends after at least 1 second of the puff is valid for training our algorithm.

3.6 Evaluation

Our evaluation has three parts. We start with evaluating how well we detect individual smoking gestures (taking smoking "puffs") and whole smoking sessions. Then, we compare our results against prior work that detects wrist gestures and smoking behavior. Finally, we include a user study for which we implemented the full computational pipeline on a mobile phone, and demonstrate the effectiveness of *CQue* for recognizing users' smoking sessions in real-time.

3.6.1 Recognizing Smoking Sessions

The ultimate goal of our work is to detect smoking sessions accurately so that we can track: a) how many times a user smoked each day, b) the length of each smoking session, and c) how many puffs they took during each session.

Smoking session detection: First, we look at how many smoking sessions were identified correctly by *RisQ*. We ran a leave-one-out cross-validation experiment where we leave a single smoking session for testing, and use rest of the data for training. We repeat this

procedure for each smoking session in our dataset described in §3.5. At each time, we evaluate the output of max-product belief propagation in our CRF.

RisQ detected 15 complete sessions out of the 17 sessions in our dataset— it missed 2 smoking sessions for two users who had gestures largely different from other users.

RisQ is also able determine the duration of smoking sessions. Table 3.2 shows the results of this experiment. The average ground-truth session duration in this set up was 326.2 seconds. We observe an average error of 65.7 seconds in our session duration estimate for the 17 smoking sessions in our leave-one-out cross-validation experiment. We note that when we exclude the two undetected sessions, the average error is reduced to 26.22 seconds. The errors are caused due to the approximate estimates of the beginning and end time-stamps of each session.

Table 3.2. Errors in session duration estimates using our CRF.

Statistic	**Avg ± Std. Dev.**
Duration of smoking sessions	326.21±19.65 s
Error in estimation	65.7±30.6 s

3.6.2 Recognizing Smoking Gestures

We examine now how well we can recognize individual hand-to-mouth smoking gestures that correspond to each smoking puff in a session. Since the total number of smoking gestures is sufficiently large in our dataset (369 "smoking puffs", 252 "food bites" and 4976 other gestures), we perform a 10-fold cross-validation experiment here: we split the gestures into 10 groups (folds), and of the 10 folds, a single fold is retained for testing, and the remaining 9 are used for training. Then we repeat for each test fold.

First, we evaluate how well individual smoking gestures can be detected by the Random Forest (RF) classifier which examines each segment independently of the others. Then we evaluate the performance of the CRF that infers the type of gesture for each segment by considering the probabilistic dependencies of the segment labels in a sequence.

Table 3.3. Performance metrics for smoking puff detection obtained using 10-fold cross-validation evaluation with i) Random Forests classifier (RF), ii) Conditional Random Fields (CRF) with Random Forests. CRF reduces 75.83% of the number of false positives generated by RF, thus improving precision.

	Performance Metrics			
	Accuracy	Recall	Precision	False-positive rate
Random Forests	93.00%	0.85	0.72	0.023
CRF	95.74%	0.81	0.91	0.005

Gesture detection performance: Table 3.3 shows the performance metrics for 10-fold cross-validation for smoking puff gesture detection using i) the RF classifier , and ii) CRF. We see that the RF classifier gives 93% accuracy in gesture recognition, and a moderate precision value of 0.72 in detecting smoking gestures. The observed precision value is only moderate because a small fraction of 5,228 non-smoking gestures in our dataset gets falsely classified as smoking gestures. The CRF significantly improves the performance by reducing the number of false positives by 75.83%. This results in a much higher precision value of 0.91 in detecting smoking gestures, and overall 95.74% accuracy in overall gesture recognition. The CRF only causes a slight decrease in recall from 0.85 to 0.81.

Optimizing performance: We now examine how we can optimize the precision and recall for smoking gesture detection by tuning our method. While training the RF classifier, we use a cost function that adds a penalty for missing a training smoking gesture. When this penalty is increased, it increases recall for the smoking gestures, although it also increases the number of false positives. Figure 3.7 shows the change in precision and recall obtained using RF versus CRF as this penalty increases. We can make two observations here: first, a false positive rate as small as 0.05 results in a significant drop in precision (0.6) of the RF classifier. This is because of the large number of non-smoking hand gestures. Second, the CRF cleans up the mis-classifications of the RF classifier to improve precision. Figure 3.7 shows that this improvement can be as high as 0.53. However, this improvement comes at a cost of drop in recall, as CRF smooths out some of the true gestures. We can see that the best performance is achieved for a false positive rate of 0.023.

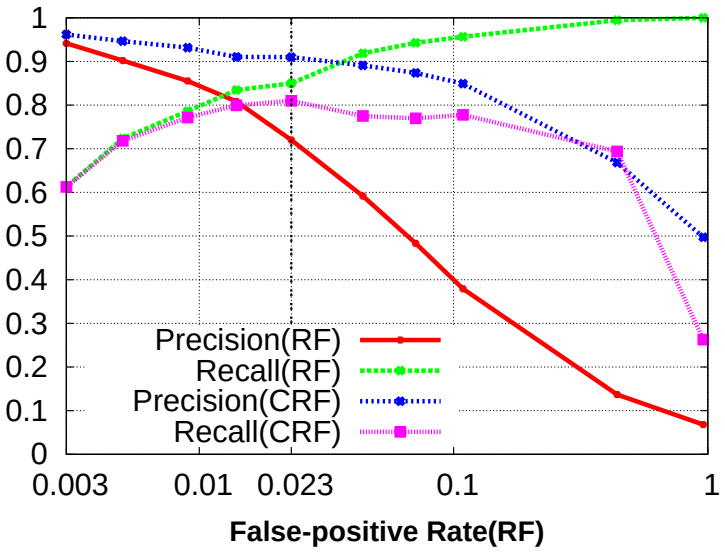
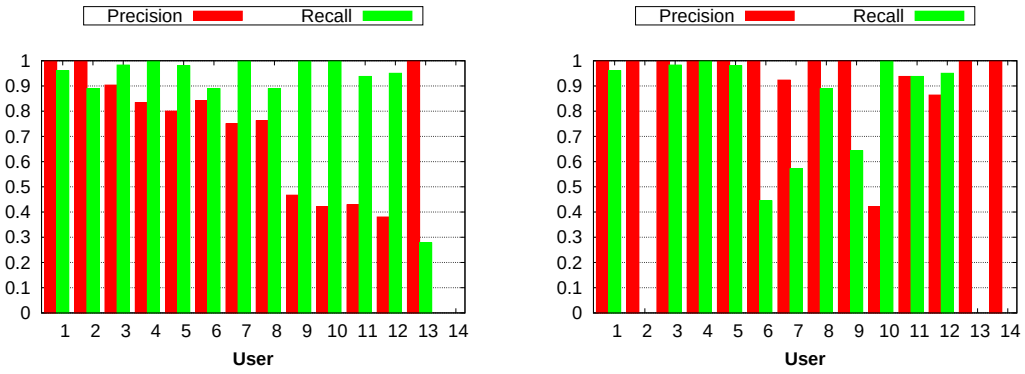


Figure 3.7. Precision & Recall versus False Positive rate, while adjusting the cost function of the Random Forest (RF) classifier during training. A small increase in false positive rate reduces precision of the RF classifier but the CRF improves the precision dramatically.

Generalization across subjects: We now examine whether our method can successfully detect gestures for a user by using training data only from other users. In other words, we examine how well our method *generalizes* to new users' data not included during training. For this experiment, we test the smoking gesture detection for each of 14 smokers, given training data from the other 13 in our dataset. Figure 3.8 shows precision and recall in this "leave-one-user-out" scenario using the Random Forest classifier and the CRF. The precision and recall of our method are high on average which indicates that our method is able to generalize to new users. In particular, the CRF improves precision, but it may occasionally filter out correctly detected puffs. This happens if the user's gestures and their temporal distribution is largely different from what is observed in the training. This suggests the it is better to acquire training data from a variety of users.

3.6.3 Comparison #1: Bite Counter

Our first comparison examines a different IMU-based gesture recognition technique, and the benefits of using our computational pipeline.



(a) Random Forests (b) CRF

Figure 3.8. "Leave-one-user-out" evaluation results. The CRF improves precision over the RF classifier. Occasionally the CRF smooths out puffs correctly detected by the RF classifier resulting in a slightly lower recall.

Table 3.4. Eating gesture detection using Bite-counter [55] for cases: i) within eating sessions when it is known that user is having a meal, and ii) when session information is not available.

	Eating Sessions		All data	
	Recall	Precision	Recall	Precision
Bite-Counter	0.60	0.57	0.65	0.03
RF	0.92	0.78	0.69	0.64
CRF	N/A	N/A	0.64	0.78

Bite-Counter [55] is a state-of-art method to detect eating bites while having a meal using angular velocity from a gyroscope worn on the wrist. The intuition is that the angular velocity about the axis along the wrist increases above an empirically observed threshold (T_1) while taking a bite and drops below another threshold (T_2) after taking the bite. Also, there is a minimum time-lapsed (T_3) between these two threshold-crossing events while eating food, and a threshold on the time interval between two consecutive bites (T_4). Thus, this algorithm requires searching for the values of these four parameters: $\{T_1, T_2, T_3, T_4\}$ that have the best score. The score is given by $\frac{4}{7}\times$ precision$+\frac{3}{7}\times$recall. One caveat is that the Bite-Counter algorithm knows when an eating session is in progress since the user presses a button, whereas *RisQ* requires no such information.

Table 3.5. Smoking gesture detection using Bite-counter [55] for cases: i) within smoking sessions when it is known that user is smoking, and ii) when session information is not available.

	In-Sessions		All data	
	Recall	Precision	Recall	Precision
Bite-Counter	0.60	0.71	0.73	0.05
RF	0.97	0.95	0.85	0.72
CRF	N/A	N/A	0.81	0.91

Eating gesture detection: For a fair comparison, we first look exclusively at eating sessions in our dataset, and train *RisQ* to detect eating gestures using the same feature set and approach that we used for smoking. For this purpose, we labeled the bite gestures in 10 of the eating sessions in our dataset. Table 3.4 shows the results for two cases — one where we examine eating gesture detection within known eating sessions (case for which Bite-Counter is optimized), and one where we look across the entire data that includes all types of gestures. For evaluation limited to eating sessions, we only use the Random Forest classifier, and excluded CRF as we do not need to detect sessions. Even when only looking within the eating sessions, we see that *RisQ* has substantially higher precision and recall (21% improvement in precision and 32% improvement in recall). We suspect that our benefits are primarily due to the use of more robust trajectory-based features rather than just the wrist rotation, and the use of the Random Forest classifier as opposed to simple thresholds. As might be expected, when the eating sessions are not known a priori, the performance of Bite-Counter drops substantially. Bite-Counter still has good recall, but the precision is extremely low (0.03) i.e., the algorithm is generating a huge number of mis-classifications since it is picking up a substantial fraction of cases where the wrist is rotated, and detecting them as a bite.

We must note here that *RisQ* was not designed to detect eating behavior in the first place. Thus, it is possible that with more careful exploration of features, the detection of eating gestures can improve. However, the fact that it performs better even without such tuning demonstrates the broader potential of our approach for gestures other than smoking.

Smoking gesture detection: Just as *RisQ* can be re-trained to detect eating gestures, Bite-Counter can be re-trained to detect smoking gestures by looking for wrist rotations that are representative of smoking. We use the methodology in [55] to learn appropriate parameters for Bite-Counter if it were used for detection smoking gestures. Table 3.5 shows the comparative results. The results clearly show that *RisQ* perform much better than Bite-counter.

3.6.4 Comparison #2: mPuff

We now turn to an alternate approach for detecting smoking gestures using a respiration chest band: mPuff [12]. Although the results are on different datasets, mPuff reports precision of 91%, true positive rate of 83% and false positive rate of 8%. Our results are similar for precision, and better for other metrics.

To directly compare these methods, we collected several hours of data from the Zephyr Bioharness Chestband for respiration data and the wrist-worn IMUs. Our dataset was captured in the wild, and has several smoking sessions in addition to other activities such as drinking, driving, walking, working on a computer, and so on.

We used the approach suggested in mPuff, which suggests a list of features and the use of an SVM classifier for training the parameters. However, there was substantial noise in the breathing dataset due to other concurrent activities (walking, driving, etc), as a result of which many breathing waveforms had to be discarded prior to training the classifier. The results of training using the clean breathing waveforms were unsuccessful, and we got low precision/recall numbers using mPuff. The results using *RisQ* instead were similar to those shown earlier.

While further exploration is needed for a quantitative comparison, we highlight that dealing with noise and confounders is a challenge no matter what sensor modality is used for detecting behaviors such as smoking. The fact that *RisQ* works well despite the confounders is one of its major strengths.

Figure 3.9. Smoking monitor mobile app and an IMU-equipped wristband.

3.6.5 User Study in the Wild

Our final study looks at the entire computational pipeline learnt using the dataset described in §3.5, and executed in real time for new users.

Android-based Implementation: Our Android-based implementation for the user study comprises of a wearable wristband with an IMU and bluetooth that continuously streams quaternions at a rate of 50 Hz to an Android phone. The *RisQ* computational pipeline for real-time smoking session detection is executed on the phone. Our application generates notifications when it detects that a smoking session is in progress and prompts the user to confirm if the detection is correct. Apart from smoking detection, our application also acts as a logging app to record data from the IMU worn by a user and to record ground truth events like the beginning and the end of a smoking session.

The complete UI of the mobile application and an IMU equipped wristband are shown in Figure 3.9. The main interface of the app has a set of UI buttons that can be used by a

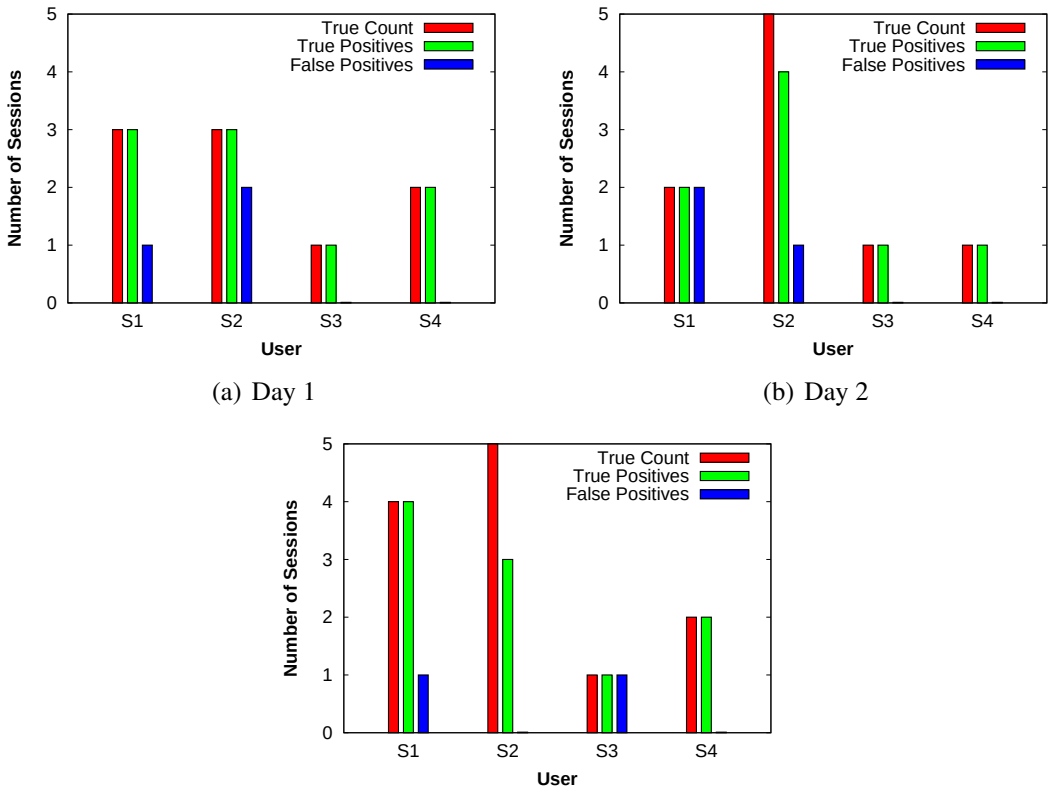
user to report smoking events as well as confounding events like drinking and eating. A user can use this to report the session boundaries using *before, during* and *after the session* events. In addition, the UI has a tab to view the events previously reported by the user. A user can use this interface to correct the reported event times or to delete a report if there was an error in submitting reports.

The complete gesture recognition pipeline has been implemented in Java. This pipeline detects smoking gestures every 20 seconds in the the buffered sensor data. The classification is done using a Random Forest classification model, learnt using the Weka data mining software [73]. The classification outputs are then buffered for a minute and used in forming a CRF model to detect a smoking session in progress. The inference using CRF is done every minute, thus there is an average latency of 30 seconds in detecting the start of a smoking session.

User study: We recruited 4 users for this study. The users wore the IMU-fitted wristband for four hours per day for three days. We asked these users to always carry the phone with our mobile application installed. To record the ground truth information, we ask users to report the rough start and the end time for their smoking sessions using our app. Also, our app prompted these users whenever it detected smoking and the users could respond by confirming yes or no. Our goal was to check how well our algorithm would work if it were integrated into a real-world intervention algorithm that a therapist might use, such as a phone call or text message.

Table 3.6. Summary of user study results obtained for 4 subjects for 3 days. Table shows average recall and precision in smoking session detection using *RisQ* for 4 subjects.

User	Number of Sessions	Performance Metrics	
		Recall	Precision
S1	9	1.00	0.69
S2	13	0.77	0.77
S3	3	1.00	0.75
S4	5	1.00	1.00
All	**30**	**0.90**	**0.77**



(a) Day 1

(b) Day 2

(c) Day 3

Figure 3.10. User study results for 4 subjects for 3 days. Figures show the following information for each subject and each day: i) ground-truth count of the smoking sessions, ii) number of sessions detected correctly i.e. true positives, and iii) number of falsely detected sessions i.e. false positives. The results show that we rarely missed a session and had only a few false detections (0-2) per day.

How well can *RisQ* detect smoking sessions in real-time?: Figure 3.10 shows the actual number of sessions, number of sessions detected by the mobile app correctly and the number of sessions falsely detected for the four users. We see that the number of sessions varied between 1 to 5 in a day for the period that the mobile app was running. Of all the 30 reported smoking sessions, we missed only 3 sessions. The number of false session detections was low and less than or equal to 2 per day. Table 3.6 gives the overall summary of results for the 4 subjects. We observed an average recall and precision to be 0.90 and 0.77 respectively. In our informal exit interviews with the users, they reported no discomfort from the small number of false alarms that was raised by *RisQ*. In summary, the

user study provides strong evidence of the effectiveness of our technique in recognizing smoking sessions across users in real-world scenarios.

Table 3.7. System overhead measured on a Galaxy Nexus phone

Statistic	**Value**
Time for segmentation	92.34±6.85 ms (52-134ms)
Time for feature extraction	79.88±5.30 ms (9-227ms)
Time for CRF Inference	5.89±2.50 ms (1-23ms)
Memory	12-20 MB
Binary Size	1.7 MB

***RisQ* Implementation benchmarks:** We finalized our evaluation with a brief benchmark of our real-time implementation of *RisQ*. We first demonstrate that our mobile app has a low system overhead. Table 3.7 shows the summary of the execution overhead of the smoking detection pipeline measured on Samsung Galaxy Nexus device running Android 4.3 OS. The execution of the three stages in the pipeline: segment extraction, feature vector extraction and session detection using the CRF take an average time of 92.34 ms, 79.88 ms and 6 ms respectively. These overheads are incurred every 20 seconds for segmentation and feature extraction and once in a minute for the CRF inference. The memory requirement to execute our app is modest and varies between 12-20 MB. Users running our application for the user study did not find our app to impact the performance of their phones in their daily usage.

Table 3.8. Specifications for Invensense MPU-9150 IMU

Operating Voltage	2.4-3.6 V
Operating Current (accel,gyro,compass)	4.25 mA
Streaming Data rate	1150 bytes/s

Power Consumption: Table 3.8 gives the performance metrics for the operation of Invensense MPU-9150 IMU that we used in our user study. This device comes with a 110mAh battery that provides up to 4 hours of bluetooth streaming of quaternion data sampled at 50Hz. The limited execution time of the device is due to the bluetooth commu-

nication overhead that requires an operating current of 50 mA when active. In comparison, the operating currents required for execution of accelerometer, compass (sampled at 8Hz) and gyroscope sensors are 140 μA, 350 μA and 3.6 mA respectively. The execution time can be increased to 12 hours by using a larger battery that fits the form factor of a wristband. This can be increased further by reducing the sampling rate and by switching off the compass. The performance of our approach is not impacted by the lack of compass as the compass is primarily used to correct the drift accumulated around the yaw-axis. Since we extract orientation-independent features for short duration gestures, the impact of the small drift on the features is negligible.

3.7 Conclusion

Mobile phones can play a major role in detecting and preventing harmful user behavior, such as smoking. In this chapter, we tackle the problem of recognizing smoking behavior using a wristband equipped with a single 9-axis inertial sensor and a mobile phone application. Our results demonstrate that we can accurately detect smoking gestures in the wild. While we focus on smoking gesture recognition in this paper, we also have preliminary experiments that indicate that our pipeline can be used for detecting other behaviors such as eating. This could be also useful for preventing harmful eating patterns that can lead to obesity. In the future, we plan to improve our trajectory extraction procedure, enrich our method with more trajectory features, and investigate more complex probabilistic models for recognizing different types of users' gestures and behavioral patterns.

CHAPTER 4

CQUE: LEVERAGING GRAPHICAL MODELS TO IMPROVE ACCURACY AND REDUCE PRIVACY RISKS OF MOBILE SENSING

The proliferation of sensors on mobile phones and wearables has led to a plethora of context classifiers designed to sense the individual's context. We argue that a key missing piece in mobile inference is a layer that fuses the outputs of several classifiers to learn deeper insights into an individual's habitual patterns and associated correlations between contexts, thereby enabling new systems optimizations and opportunities. In this chapter, we design *CQue*, a dynamic bayesian network that operates over classifiers for individual contexts, observes relations across these outputs across time, and identifies opportunities for improving energy-efficiency and accuracy by taking advantage of relations. In addition, such a layer provides insights into privacy leakage that might occur when seemingly innocuous user context revealed to different applications on a phone may be combined to reveal more information than originally intended. In terms of system architecture, our key contribution is a clean separation between the detection layer and the fusion layer, enabling classifiers to solely focus on detecting the context, and leverage temporal smoothing and fusion mechanisms to further boost performance by just connecting to our higher-level inference engine. To applications and users, *CQue* provides a query interface, allowing a) applications to obtain more accurate context results while remaining agnostic of what classifiers/sensors are used and when, and b) users to specify what contexts they wish to keep private, and only allow information that has low leakage with the private context to be revealed. We implemented *CQue* in Android, and our results show that *CQue* can i) improve activity classification accuracy up to 42%, ii) reduce energy consumption in classifying

social, location and activity contexts with high accuracy(>90%) by reducing the number of required classifiers by at least 33%, and iii) effectively detect and suppress contexts that reveal private information.

4.1 Introduction

The past decade has seen unprecedented growth in sensor-rich mobile phones and wearable accessories such as fitness monitors, sleep monitors, heart monitors and others. With the proliferation of such devices, there has been significant emphasis on techniques to draw higher-level inferences from continuous sensor data as we move around in our day-to-day lives. Research has shown that several aspects of our behavior can be inferred including physical activity, sleep behavior, social context, movement patterns, emotional or affective context, and mental disorders, with varying degrees of accuracy.

The growing landscape of high-level inferences presents an interesting opportunity: *can we combine these inferences to obtain deeper insights into individual behavior?* Intuition suggests that the outputs of individual inference algorithms must be correlated across space and time; after all, they sense various dimensions of an individual's habits, behaviors and physiology all of which are inter-linked. As a simple example, take the case of an individual's mobility patterns and how much of it can be inferred without using any location sensor such as GPS, cell tower or WiFi. Location is often correlated to social context — the fact that phones of colleagues are in proximity (detected via bluetooth) can indicate that the likely location is the workplace whereas the fact that a family member's phone is nearby means that one is likely at home. Similarly, location relates to activity context — a user may be more sedentary at work than at other times during the day, therefore one could infer that the most likely location is the workplace by observing the level of activity. More broadly, almost all inferences are related in one way or another — studies have shown that sleep (or lack of it) affects our mood and productivity, social interactions influ-

ence our emotional context, addictive behavior is correlated to specific locations and social interactions, and so on

The existence of a rich interaction graph between inferences presents an opportunity and a challenge. On one hand, a model of the relations across the different inferences can be leveraged to gain insights into individual behavior, thereby enabling new systems optimization opportunities. For example, a high level inference framework might observe relations between semantic location and activity levels, and learn that an individual has mostly sedentary behavior at the workplace. This provides two opportunities to optimize inference of the location context, "work": a) energy-efficiency can be improved by relying primarily on activity inference rather than expensive GPS, and b) accuracy can be improved when there is significant error (e.g. indoor settings) by fusing it with activity levels.

On the other hand, the ability to leverage relations across inferences comes with a steep price tag — loss of privacy. A mobile application that purports to merely be using the accelerometer to detect activity levels may indeed be inferring your location. More disturbing is the possibility that a single application developer may have several applications, one that monitors sensors for activity level, and perhaps another that uses bluetooth, which may be leveraged in conjunction to reveal much more than from the seemingly innocuous individual applications. Compounding this issue is the fact that we do not have tools that allow us to reason about how much of privacy exposure occurs from revealing seemingly innocuous sensor data if the adversary were to have a good model of the relations across contexts.

In this chapter, we present a mobile "big data" inference framework, *CQue*, that takes as input streams of inferences from diverse on-body and smartphone-based sensors, and provides a unified way for exploiting and exploring relations across these inferences. *CQue* can be used in several ways: a) an inference algorithm can leverage *CQue* to improve accuracy and/or energy-efficiency, while remaining agnostic of how this is achieved, b) a user can issue "what if" queries that explore the extent of privacy leakage of sensitive

information that might be possible if certain sensor data were revealed to one or more applications, and c) a user can specify privacy policies to provide a first order protection against privacy leakage of sensitive information to the untrusted applications.

In terms of system architecture, the key benefit of *CQue* is that it separates *detection* from *fusion*. Existing classifiers are largely designed in a stovepipe manner to address a specific context sensing goal as best as possible. While they start with detection of the specific activity (e.g. conversation, walking, etc), real-world vagaries often result in spurious state transitions due to several confounding factors. To address these issues, classifiers often rely on other sensor sources that can identify confounders and reduce errors. *CQue* provides a clean separation between the detection layer and the fusion layer — in this way, a classifier can be designed to just focus on detecting the phenomena of interest, and leverage fusion mechanisms to further boost performance by just hooking into *CQue*.

To users and applications, *CQue* offers a simple context querying interface with support for several types of queries. A "context query" can request different contexts, while optionally specifying constraints such as confidence requirements and delay bounds. For example, a query might request sedentary activity context with 90% certainty and a maximum notification delay of two minutes. *CQue* uses the query constraints to reason about how to duty-cycle inference algorithms, and how much temporal history to leverage to improve accuracy and confidence. A "what if" query provides a measure of potential leakage of sensitive information if applications were allowed access to specific sensor sources. For example, a query can request the expected leakage of "home" location context if an application had access to accelerometer and bluetooth data. Internally, *CQue* would execute such a query by providing a measure of the correlation between the outputs of inference algorithms that operate on accelerometer and bluetooth data, and the specific location context that the user does not wish to reveal. Finally, through privacy policies, a user can specify a context as private that should not be revealed, and suppress the non-private contexts in real-time that can be used to infer private context using correlations among these contexts.

A user can specify this policy specific to an application or a group of applications that can potentially collude.

Our results show that:

- *CQue* can answer context queries with high confidence by performing fusion of information from multiple context-inference algorithms and can improve accuracy up to 42% over standard context-inference algorithms.
- When energy is limited, *CQue* can lower execution costs for multiple context queries by exploiting context relations to run fewer inference algorithms. If energy is plentiful, *CQue* can decide what context algorithms in addition to the query set should be executed to improve accuracy.
- *CQue* is effective in assessing privacy risks and provides privacy while ensuring high utility to the applications.

The rest of the chapter is organized as follows. §4.2 describes the prior work done in the related research areas. In §4.3, we provide an high-level overview of the *CQue* query engine along with the description of how a context query can be specified in our framework. In §4.4.1, we describe the relationship model used by *CQue*. Rest of §4.4 describes the components of *CQue* query engine in detail that are responsible for optimizing multiple context queries and addressing privacy. We describe the experimental evaluation of *CQue* in §4.6 respectively. We discuss possible extensions of this work in §4.7. Finally, the conclusions are provided in §4.8.

4.2 Related Work

In this section, we describe three areas of related work — model-driven sensor data acquisition, context inference for mobile phones, and privacy in temporal data.

Graphical models for sensor data acquisition: There has been substantial work on leveraging spatial and temporal models of correlations between distributed sensor sources

to optimize sampling and communication in a sensor network [38, 52, 53, 98, 116]. For example, BBQ [53] uses a Dynamic Bayesian Network to select the minimum number of sensor nodes for data acquisition such that it can answer range queries within query-desired confidence bounds. Meliou et al [116] extended this work to sensor network routing where a query can be answered with desired confidence bounds while traversing the minimum number of nodes in a network spanning tree. Graphical models are also used in [52], which explored the problem of answering range queries while minimizing the energy cost of sampling sensors, and in [38], where sensors use models to reduce the communication costs by transmitting samples to a base station only when the ground truth is significantly different from the prediction made by the model. The similarities with prior efforts are only in that they leverage DBNs — we use DBNs in a novel application context which is for real-time inference on a mobile phone to enable energy-accuracy-delay tradeoffs and protect privacy.

Context-Sensing: Our work is closest to ACE [127] that proposed a context sensing engine that exploits relationships among contexts to infer an energy-expensive context from a cheaper context. The central difference between ACE and *CQue* is that the former uses Association Rule Mining to learn rules among contexts (e.g. *Driving* $\Rightarrow \neg$*At Home*), but such mining approaches cannot take a context classifier's uncertainty into account. The ability to take uncertainty into account is critical when using classifiers that operate on raw sensor data — for example, if a classifier may detect *Driving* with a low confidence, say 0.7, GPS may detect *At Home* with an error radius of 100m, and so on. In these cases, the rules would assume perfect context information whereas *CQue* would use the uncertainty in inferring the relations. Thus, a probabilistic approach is strictly superior to a rule based one.

Also related to *CQue* are efforts to develop a context-sensing engine for phones that can be used by applications to request contexts [62, 40, 106]. The Jigsaw context sensing engine [106] comprises of a set of sensing pipelines for accelerometer, microphone and GPS. More

recently, Kobe [40] proposed a context querying engine for mobile phones that can be used to plugin different classifiers, and that balances energy consumption, latency and accuracy of the classifiers by offloading computation to the cloud. Unlike these approaches, *CQue* can leverage probabilistic relations across contexts.

CQue is complementary to prior work on context sensing engines which have largely explored optimizations of individual classifiers (e.g. [21, 97, 106, 151]). *CQue* allows individual classifiers to be easily integrated without worrying about how to leverage other contexts to improve performance and efficiency.

Privacy in temporal data: There has been some work in addressing privacy where the adversary is aware of the temporal correlations [70, 135, 144]. MaskIT [70] presents a privacy-preserving mechanism to filter a stream of contexts that can be used to answer context queries requested by the phone applications. It presents two privacy checks to decide when a user context can be released while providing privacy guarantees against an adversary knowing temporal correlations modeled as a Hidden Markov model(HMM) of user contexts.

Our goal in this work is not to to prove privacy guarantees or design new privacy metrics, rather our argument is that the use of DBN enables considerably more flexible privacy policies in comparison with an HMM-based approach. When MaskIT identifies that a privacy breach might occur, it suppresses all contexts at that time without considering which subset of them might contribute to the breach. In contrast, *CQue* enables more fine-grained reasoning of which specific context(s) lead to a privacy breach such that only that information can be suppressed.

4.3 CQue Overview

The *CQue* framework has been designed with the goal of providing an easy-to-use abstraction to application developers and end users, both for exploiting correlations across inference to improve energy-efficiency and accuracy, as well as to use these correlations

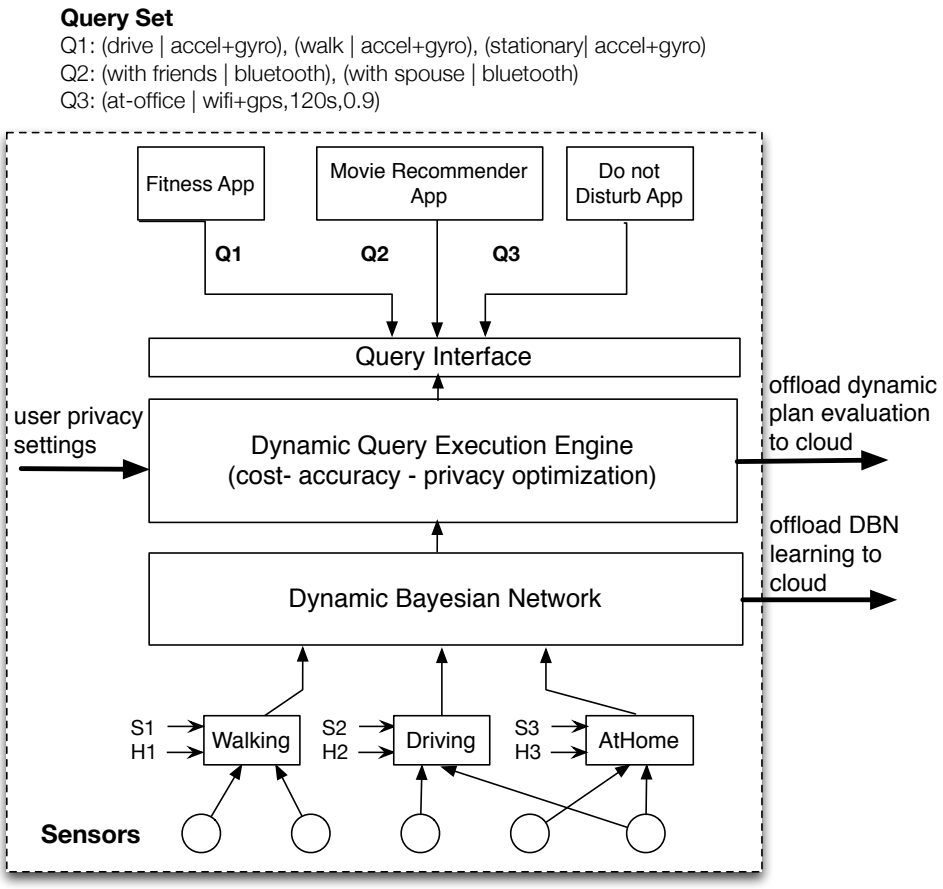


Figure 4.1. High-level overview of the framework describing the workflow

to ascertain privacy leakage or contextual insights. This goal is accomplished using a high level query language and an underlying execution environment running as a service on the phone. In the following, we provide an high-level overview of the *CQue* framework.

Context Query Interface At the top level of *CQue* are applications that issue context queries using a high-level declarative query language. *CQue* provides a simple query language for the applications to request contexts. Consider the following three queries:

```
Q1. (drive | accel+gyro+gps, 240s, 0.8)
Q2. (home? | accel+gyro)
Q3. (home? | with friends)
```

The first query requests the context *drive*, and includes three other optional fields. The keywords *accel*, *gyro* and *gps* suggest that any combination of classifiers that use those sensors may be used in answering the query (to ensure that the app only uses sensors for which it has approval from the user). These keywords could also refer to other inferences, if the application needs more explicit control over other contexts that should be fused. Finally, the query can also replace this with a '*' which would let *CQue* automatically choose sensors and classifiers to optimize the system. The value '240s' in the query is the *delay tolerance* and denotes that the application can tolerate the maximum delay of 240 seconds for each record in the query response. The query processor uses this delay window to improve the confidence in the output context value. The query can also provide a value for *confidence threshold* in the range $[0.0 - 1.0]$. In the above example, the query processor will output context as soon as it reaches confidence of '0.8', provided it is within the delay tolerance period. If confidence values associated with the context output are lower than the query-desired *confidence threshold*, the context is reported as 'unknown'.

The next two queries marked by '?' are privacy queries and provide a measure of *information gain* in queried context "home" if any inference algorithm operating over accelerometer and gyroscope sensors were used (Q2) or if *with friends* inference algorithm was used (Q3). In turn, this measure provides a *privacy score*, an indicator of privacy leakage. Note that in providing a *privacy score*, *CQue* is limited to using inference algorithms that are available in its library.

CQue Architecture The central tenet of *CQue* is an architectural separation between the fusion mechanisms that inference algorithms use to improve performance. Our design employs a cleaner, layered architecture where any classifier can simply hook into *CQue*, and leverage temporal consistency and context fusion mechanisms to improve performance. *CQue* models temporal consistency with a time-series of previous observations, and uses active learning mechanisms to automatically determine when user input is needed to improve this model. *CQue* also automatically determines the relationships between a classifier

output and all other observations from other classifiers/sensors that it has access to, and determines when and how to use additional observations to improve accuracy. In short, it can make the process of designing a new inference algorithm a lot easier for a designer.

In contrast to the stovepipe approach, our design separates these components into a cleaner, layered architecture where any classifier can simply hook into *CQue*, and leverage temporal consistency and context fusion mechanisms to improve performance. *CQue* models temporal consistency with a time-series of previous observations, and uses active learning mechanisms to automatically determine when user input is needed to improve this model. *CQue* also automatically determines the relationships between a classifier output and all other observations from other classifiers/sensors that it has access to, and determines when and how to use additional observations to improve accuracy. In short, it can make the process of designing a new inference algorithm a lot easier for a designer.

Execution Environment The execution environment of *CQue* is shown in Figure 4.1 and consists of following core components: (i) a graphical model, namely, Dynamic Bayesian Network, and (ii) a query processor. The *Dynamic Bayesian Network* (DBN) is the at the core of *CQue* and plays an important role in making various decisions during query processing. The DBN provides a model of the relationships across multiple contexts for the phone user and keeps track of the time-series of observations for various contexts. It then uses context relationships to boost confidence in observations made by individual classifiers or to correct them. The second component is a *query processor* which is responsible for interaction with applications including receiving context queries and generating the responses to the queries. In addition, depending on the current state of the DBN and the time-series of observations, the query processor needs to determine which context-inference algorithms should be executed such that it can provide best answers for the queries within the specified constraints. These decisions are made by the query processor at every time step during execution. As shown in Figure 4.1, the result of these decisions are the signals $S_1, S_2, ..., S_n$ indicating which context-algorithms be executed. Finally, upon observing the output of se-

lected context-inference algorithms, *query processor* enforces user-specified privacy policies (defined in §4.4.2) by deciding a maximal set of queried contexts whose values can be released to the applications without violating the privacy policies.

In addition to the query execution, the query processor is responsible for sampling human-provided context values during the learning phase of the DBN. Such human input can be minimized through active learning to only obtain input at appropriate times to improve the structure of the DBN. These human inputs shown as signals $H_1, H_2, ..., H_n$ in Figure 4.1 are provided to the context-inference algorithms that can use it to retrain themselves to personalize for the phone user.

4.4 The Execution Environment

The execution environment of *CQue* is responsible for i) modeling and learning the relationships across contexts, ii) multi-query optimization such that uncertainty in queried contexts is minimized while operating within the budget and delay constraints of a query, and iii) executing privacy queries and enforcing privacy policies. In the following, we describe each component of the execution environment in detail.

4.4.1 Inference Framework

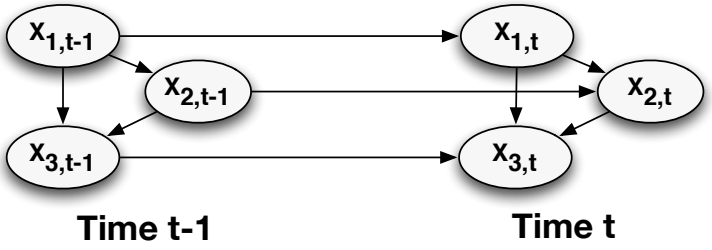


Figure 4.2. An example of 2-step Dynamic Bayesian Network. The nodes in the graph model the set of random variables $\mathbf{X} = \{X_1, X_2, X_3\}$.

The core learning framework in *CQue* is a Dynamic Bayesian network (DBN), which is a class of Bayesian networks that can represent a time-series of random variables to model

temporal consistency. A DBN can describe both temporal and static relationships among random variables at different time instances. This model is represented as a directed acyclic graph consisting of nodes corresponding to each random variable at each time instance. The static dependencies across random variables at time instance t are represented by edges connecting nodes corresponding to these random variables. The temporal dependencies are represented by transition edges connecting nodes at time instance $t-1$ with nodes at time instance t. Each node in this graph has a conditional probability table (CPT) describing dependencies on its parent nodes. Figure 4.2 shows an example of 2-step DBN for two time slices which can be unrolled to accommodate a time-series of any length by duplicating the time-slices and transition edges (We note that a Hidden Markov Model is a simplest Dynamic Bayesian Network that models and relates states of a single random variable over adjacent time steps.). We use notation $\mathbf{X}$ to denote the set of random variables. We use $\mathbf{X_t}$ to denote set of nodes representing random variables $\mathbf{X}$ at time instance t and use $\mathbf{X_{1...t}}$ as short notation to describe time series of nodes $\mathbf{X_1}, ..., \mathbf{X_t}$. With this notation, the joint probability distribution for a time series of nodes is given by

$$Pr(\mathbf{X_{1...t}}) = Pr(\mathbf{X_1}) \prod_{t=2}^{t} Pr(\mathbf{X_t}|\mathbf{X_{t-1}})$$

In the above equation, probability $Pr(\mathbf{X_t}|\mathbf{X_{t-1}})$ is computed using the DBN.

DBN Model for CQue In *CQue*, we have three types of DBN nodes — sensors, classifier-provided context, and real-world context. The goal of the DBN is to model relationships across real-world contexts while taking into account the uncertainty associated with the classifier-provided contexts. A key challenge in constructing a DBN model is ensuring computational tractability on a mobile phone since the execution time for probabilistic inference is exponential in the number of variables in a chain in DBN. Thus, to optimize execution cost, we need to reduce the computational overhead incurred by adding classifiers and sensor nodes to the DBN.

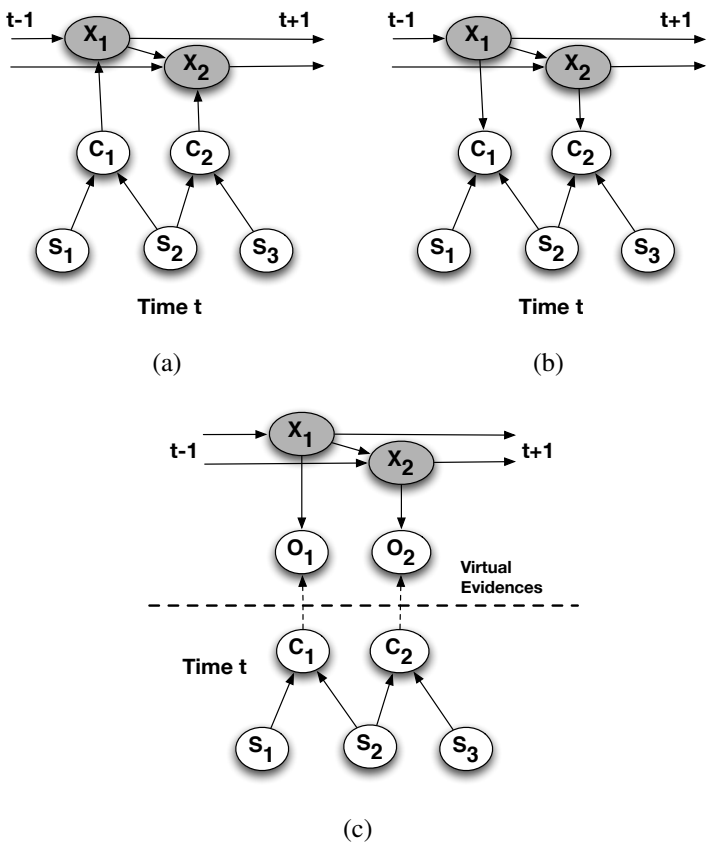


Figure 4.3. Various models for including sensors and classifiers in a time-slice of DBN for real world contexts a) A simple model for 2 contexts where sensor nodes $\mathbf{S}$ determine classifier state $\mathbf{C}$ and the classifier generates probability distribution for the states of real world $\mathbf{X}$. b) A slight variation of earlier model with arrows between classifier nodes $\mathbf{C}$ and real world context nodes $\mathbf{X}$ reversed. In this model, the classifier state is determined by both sensors and the real world. c) A model separating DBN from the layer of sensors and classifiers. In this model, the classifiers provide virtual evidences to the DBN.

To illustrate the tradeoffs, we consider two possible DBN models for *CQue*. Figure 4.3(a) shows one model where the set of sensor nodes **S** at the lowest level determine the state of the classifiers **C**, which in turn generate the probability distribution of the values for current state of the real world **X**. The distribution of **X** is then given by conditional probability $Pr(\mathbf{X}|\mathbf{C} = \mathbf{c})$ where c is the classifier state. This model is efficient when classifiers (**C**) are observed i.e. switched on, since the contexts in real world (**X**) are conditionally independent of the sensors (**S**) given the state of the classifiers **C**. In this case, computing the probability for **X** does not require iteration over possible states of **S**; in other words, sensor

nodes can be ignored. However, if some classifier node C_i is not observed then the probabilistic inference of corresponding real-world node X_i may require evaluating probability over a chain of classifier and sensor variables connecting C_i to some observed classifier variable C_j or some observed sensor variable S_k. Thus, our first model is computationally *cheap* if classifiers are observed, and *expensive* when they are switched off.

An alternative model is shown in figure 4.3(b). This model assumes that the current state of the classifier nodes **C** is affected not only by the sensors but also by the current state of the real world. The advantage of this model is that when a classifier, C_i is turned off, contexts in real world (**X**) are conditionally independent of sensors that are solely connected to C_i, hence these sensor nodes can be ignored in the inference. However, we need to learn joint probability distributions of **X**, **C** and **S** for each individual, which will require significantly more labels from the individual. Hence, this model is *cheap* computationally but *expensive* in terms of human labels required for training.

To address the limitations of these models, we construct an approximation of the above schemes where classifiers and sensors are separated from the DBN into a different layer. Figure 4.3(c) shows an example where classifiers provide the probability of a context through observation nodes called virtual evidences, **O**, to the DBN. The uncertainty in evidence is accounted in these observation nodes using Pearl's method [138]. In this method, for some context X_i and corresponding observation node O_i, if $L(x)$ gives the likelihood of classifier stating that $X_i = x$ if X_i is actually in state x and if $\overline{L(x)}$ gives the likelihood of a classifier stating that $X_i = x$ if X_i is not in state x, then the conditional probabilities for observation node O_i given the current state for real world context X_i satisfies:

$$Pr(O_i = x|X_i = x) : Pr(O_i = x|X_i \neq x) = L(x) : \overline{L(x)}$$

With these conditional probabilities and given a series of virtual evidences $\mathbf{O_{1...t}}$, we can compute the joint probabilities for time series of real world contexts $\mathbf{X_{1...t}}$ as follows:

$$Pr(\mathbf{X_{1...t}}, \mathbf{O_{1...t}}) = Pr(\mathbf{X_1}|\mathbf{O_1}) \prod_{t=2}^{t} Pr(\mathbf{X_t}|\mathbf{X_{t-1}}) Pr(\mathbf{O_t}|\mathbf{X_t})$$

This model has several benefits: a) it is computationally cheap both when classifiers are turned on or off since it avoids the complexity of modeling sensor and classifier variables in the DBN itself, b) it requires no additional human labels for modeling these variables while training the DBN, and c) it can easily be extended to handle multiple classifier implementations for the same context without changing the DBN structure. The main benefit of this model is that the classifiers can be treated as black boxes, and we do not need to assume knowledge of what sensors they utilize, what sampling rates they use, how they duty-cycle sensors to save energy, and what features are extracted in the classifiers.

DBN usage in CQue The output of the DBN is used in several ways by *CQue*.

1. **Filtering/Smoothing** Most of the contexts that we observe in the real world have some temporal consistency i.e they last for some time period. Such temporal relationships can be used to correct intermittent misclassifications made by context-inference algorithms. Since a DBN models temporal consistency with the time-series of previous observations, a DBN can correct the current output and reduce the probability of a context taking an incorrect value. Apart from the temporal consistency, a misclassification can be corrected using relationships with other contexts if the observations are available for them.

2. **HindSight** The previous case described using history to correct output at the current time. Similarly, a DBN can be used to improve the confidences in historical observations, which perhaps had low confidence. Future observations with high confidence can be used to improve the confidence of previous observations. This capability can be exploited when queries specify a higher delay tolerance threshold.

3. **Value of Information** The DBN can also be used to assess the value of a context-inference algorithm. *Value of information* (VOI) is defined as the expected gain in certainty for the random variables in DBN if an additional observation is made. This

is useful for the query processor, which can decide what context observation can provide highest utility.

4.4.2 Query Processor

The query processor is responsible for i) multi-query optimization i.e. achieving the confidence requirements of context queries while operating within query constraints, ii) execution of privacy leakage queries, and iii) enforcement of user-defined privacy policies. Multi-query optimization in *CQue* raises following challenges: a) how to choose the inference algorithms to execute given the budget?, and b) how to handle different delay tolerance requirements of queries? In addition, privacy queries and policies require us to answer: a) how to execute privacy queries?, and how to enforce privacy policies? We address these question in the rest of this section.

Which inferences to execute given a budget?

The *CQue* query processor needs to execute the set of inference algorithms that provides maximum value of information (VOI) for the queried contexts within the budget constraints. The budget is typically in the form of energy since inference algorithms consume energy, but in cases where the user is interrupted to provide labels, the budget may be the number of interruptions allowed per day.

Due to the relationships across contexts, the set providing maximum VOI may be larger than the query set as some contexts that are not part of the query set may be useful to improve the inference accuracy for contexts that are in the query set. Similarly, this set can be smaller than the query set if some of the queried contexts are strongly implied by other contexts in the query. In addition, the set of inferences that provides the maximum VOI can change over time, depending on the user's current context as well as dynamics in the user's behavior patterns. In this section, we describe an adaptive approach to deciding the set of inferences that need to be executed to satisfy query demands while remaining within the budget.

The problem of selecting the optimal set of context-algorithms to execute given a budget and the set of queries can be formalized as follows. Let Q be the set of context queries, let E_{t-1} be the set of context classifiers undergoing execution currently and v be the corresponding classifier outputs. Let $\mathcal{F}(Q, E, E_{t-1}, v)$ be the utility function giving the VOI of the contexts, E, for the query set where E is the subset of observation nodes $\mathbf{O}$ in the DBN. Let $C(E)$ give the cost of executing algorithms for contexts in E. If $\mathcal{B}$ is the maximum budget, then our goal can be defined as identifying a set E_t i.e. the set of context classifiers to be executed next such that:

$$E_t = \underset{E \subseteq \mathbf{O_t} : C(E) \leq \mathcal{B}}{\arg\max} \mathcal{F}(Q, E, E_{t-1}, v) \tag{4.1}$$

Since our goal is to have high certainty for query set Q, we consider the information gain for query set Q (also called VOI) as the utility function which is given by:

$$\mathcal{F}(Q, E, E_{t-1}, v) = H(Q|E_{t-1} = v) - H(Q|E, E_{t-1} = v) \tag{4.2}$$

where $H(.)$ is the entropy function. We must note that computing optimal-set of classifiers to be executed at the next iteration is an **NP**-hard problem[87]. We solve this optimization problem using a greedy approach as described in Algorithm 1. The results by Nemhauser et al. [128] and Krause et al. [87] have shown that a greedy algorithm provides a solution within the constant factor $(1 - \frac{1}{e})$ of the optimal solution for a general bayesian network.

Our extension of optimization problem to the DBN does not change the results as long as we select nodes E_t from a set S such that the nodes corresponding to S in DBN are independent to each other given the query set Q. In a DBN, we select E_t from a set of observation nodes O_t where each observation node is connected to the DBN only through a parent hidden node. Hence, any path between any two observation nodes is always blocked by one of the parent hidden nodes. Thus, any two observation nodes in DBN are **d**-separated [153] and hence, independent of each other.

We note that Algorithm 1 gives near-optimal solution when each classifier is assigned a unit cost. In a specific scenario, there may be several considerations in determining the cost, which depends on the cost of sensing, processing, or obtaining human input. If the resulting costs are non-uniform in nature, we refer the readers to an algorithm as described in [87].

Algorithm 1 Compute Optimal Set

Input: Budget k; query set Q; set $\mathbf{O_t}$ from DBN;
utility function $\mathcal{F}$; previous observed set E_{t-1} and corresponding set of classifier outputs v; Cost function C.
Output: Set of observations $E \subseteq \mathbf{O_t}$
Let $E = \phi$
for $i = 1$ to k **do**
  $o^* = \arg\max_{o \in \mathbf{O_t} \setminus E} \mathcal{F}(Q, E \cup \{o\}, E_{t-1}, v)$
  $E = E \cup \{o^*\}$
end for
return E

How to handle delay requirements of queries?

In *CQue*, we unroll the DBN such that it can accommodate a series of observations of length W. At each time-step, we slide the DBN window over observations such that the oldest observations are dropped and the latest observations are added to the DBN. As a result of this sliding window, any context observation resides in DBN for exactly W time-steps. Hence, we can compute the probability for queried contexts W times, corresponding to having $0, 1, 2, ..., W-1$ observations from the future. This allows our framework to answer a query with a delay upto $W-1$ time-steps. Our framework can use this delay period in three cases: i) if the sequence of historical observations have low confidence resulting in low confidence for the latest observation, ii) if the sequence of historical observations have fluctuating values for the contexts indicating low confidence in the output of context inference algorithms, and iii) if the latest observation is different from the historical observations. In the first two cases, we wait for the future observations and hope that these observations are consistent and have high confidence. If this happens, we boost the

probability of queried context in hindsight. In the third case where the latest observation is different from the historical observations, it can happen either because of intermittent misclassification by the classifier or because the context value has actually changed. Any future observation can help in distinguishing these cases and improve the certainty in the context.

How to execute privacy queries?

Unlike queries that request current state of an individual, *privacy queries* look for aggregate information regarding the mutual information shared between a context and sensor data or other contexts available in *CQue*. In our framework, we provide two types of privacy queries: i) user specifies a query context, Q and wants to know information revealed about the queried context from a specified list of sensors (S) and contexts (C); and ii) user specifies a query context Q and wants to know ranked list of contexts along with a *privacy score* that indicates how much information is revealed by a ranked context about the queried context. Thus, these queries help user understand how much information can be revealed about certain context from other contexts. Together, these two query types enable users to make a decision regarding their own privacy.

For the first query type, given a user specified sensor list S we identify context classifiers C^S that can execute given sensors S. In addition to S, user can provide a set of contexts C. Now, we want to have an uncertainty measure in queried context Q if contexts in $C' = C \cup C^S$ are observed. We use a normalized variant of mutual information based on information theory that gives a distance metric between Q and C' and is given as follows :

$$D(Q|C') = 1 - \frac{H(Q) - H(Q|C')}{H(Q)}$$

This metric takes value 1 if Q is independent of C' and takes value 0 if Q is fully determined by C'. We use this measure as a *privacy score* where a higher score indicates less information leakage.

For the second query type, a user is interested in identifying ordered list of contexts that reveal most information about queried context Q. For each context c supported in *CQue*, we use *privacy score* $D(Q|c)$ as described above to measure information leaked about queried context Q. We rank contexts based on this metric.

How to enforce privacy policies?

As described in §4.2, one of the benefits of *CQue* is that the DBN provides in-depth and real-time information about correlations between different context outputs in contrast with HMMs and other techniques that have been leveraged in prior work. *Our focus is not on identifying the best privacy exposure policy or proving its privacy properties; rather it is to demonstrate that CQue can be used to develop such methods.*

To demonstrate these benefits, *CQue* supports a privacy policy that can suppress context values that leads to change in confidence or the probabilistic belief about a private context greater than threshold δ. Such a policy can provide greater privacy control than a simpler policy that just blocks a specific private context from being revealed. In *CQue*, a user can define these policies specific to a potential adversarial entity where this entity can be an application or a group of applications or all the applications on the user's phone. A policy for a group of applications can be useful to protect against information collusion among these applications in a group.

In *CQue*, we assume a strong adversary having access to the user's DBN. Now, we enforce user policy by using the DBN to calculate the confidence or the posterior probability for the private context using the output for the set of contexts, requested by the adversarial entity, as observations in DBN. Also, we compute prior probability for the private context without using any observations in the DBN. If the difference between posterior probability and the prior probability is greater than δ, then we suppress the value of the context that causes the maximum change in probabilities. If the change in probabilities is still high with the remaining context outputs, we repeat the process of removing context value that causes

maximum change until we reach the threshold limit δ. We release the remaining contexts to the adversarial entity.

4.4.3 Learning the Graphical Model

Now, we describe the challenges involved in learning the DBN model for the phone user. There are two challenges in learning the DBN: a) how to learn structure without executing all sensors and context inference algorithms continuously on the phone since this consumes significant energy, and b) how to minimize interruptions of the phone user to obtain ground truth labels for learning.

Context relationship hints: The learning process can degrade user experience due to the energy cost of running several context inference algorithms concurrently on the phone. While it is possible to randomly sample a few contexts at a time, this can slow down the learning process as the random approach may not always sample related contexts together. To perform efficient sampling and facilitate faster learning of personalized relationships, we maintain, for each context, a list of contexts to be sampled together that may possibly have soft relationships. The soft relationships may or may not hold for individual users and it does not necessarily result in edges in DBN connecting softly-related contexts. As it may not be possible to envision all the soft relationships, we still perform random sampling and bias the sampling for contexts that seem to be related.

While the use of soft relationships can address the efficient sampling problem, we can speed up the learning process and parameter estimation in DBN by the use of *hints* that can be provided by the context-inference algorithm developers. A context developer can include hints for each context that identify positive causes and negative causes for it. As an example, *driving* is a direct positive cause for user's location being *on street* whereas user's location being *at home* is a negative cause for *on street*. These hints can be directly utilized in setting the parameters for relationships that are true for a majority of users.

Minimizing human-provided labels: One of the challenges in learning the dynamic bayesian network is that we do not have access to ground truth for contexts. While ground truth can be obtained by having the user provide ground truth for an initial training period, we wish to minimize interruptions and hence use this option sparingly. Instead, we can sample the contexts inferred by the algorithms along with the confidence value associated with the output. The inferred context values provide us with the partial observations of the underlying Markovian process. We use the Structural EM algorithm described in [66] to learn the structure of the DBN from the partial observations. In this algorithm, the structure of the DBN is improved iteratively until the MDL score of the structure B given a training data set D converges. Let us suppose a structure B consisting of n random variables $X_1, ..., X_n$. We use the notation x_i and Π_{x_i} to denote an assignment for X_i and its parent set $\mathbf{Pa}(X_i)$ respectively. Also, $N(x_i, \Pi_{x_i})$ denotes number of instances in the training data D where $X_i = x_i$ and $\mathbf{Pa}(X_i) = \Pi_{x_i}$. Let N be the total number of instances in training data. Then, the MDL score for a DBN structure B given data set D is given as:

$$\sum_i \Big\{ \sum_{x_i, \Pi_{x_i}} E[N(x_i, \Pi_{x_i})|D] \log \theta(x_i, \Pi_{x_i}) - \tfrac{\log N}{2} \#(X_i, \mathbf{Pa}(X_i) \Big\}$$

where $\theta(x_i, \Pi_{x_i}) = \frac{E[N(x_i)|D]}{E[N(x_i, \Pi_{x_i})|D]}$ and $\#(X_i, \mathbf{Pa}(X_i)$ is the number of parameters needed to represent $Pr(X_i|\mathbf{Pa}(X_i))$. We need to take expected counts as we have access to only partial observations. We simplify the learning process by first learning the static network of the DBN followed by learning the transition edges of the network. This process is suboptimal in nature but it has been shown in practice to yield parameters estimates close to optimal.

While learning the DBN without any human input is ideal, this can lead to errors particularly if the training and test context distributions are very different. Thus, there is a need for at least some corrective human input. We randomly sample human input for a small fraction of contexts in-order to correct DBN parameters, and assign these human-provided contexts higher weight over the instances obtained using context-inference algorithms.

4.4.4 Classifier Personalization

So far, our discussion has assumed that the underlying context classifiers are black-boxes *i.e.* they do not expose their internal behavior or allow changes. Thus, the DBN is limited to the context output and uncertainty provided by the classifiers. A natural question is whether we can do better if we were able provide *feedback* from the DBN to the classifier in-order to improve its performance further.

In *CQue*, any human input provided to the DBN can not only be used to correct DBN parameters, but also be used to personalize the classifiers. A classifier personalized for an individual user is beneficial as it provides higher classification accuracy and better uncertainty estimates. The use of personalized classifiers also benefits the DBN — with personalized classifiers, the correct DBN parameters can be learnt using fewer future human input resulting in shorter training period. *CQue* provides an API to the developers of context classifiers to receive human-provided labels that can be leveraged for personalization.

4.5 Implementation

We now describe the implementation of *CQue* on Android smartphones running Android OS version 2.2 or higher.

Inference Engine: The *CQue* inference engine is designed to run in real-time on a mobile phone to avoid any delays incurred in accessing the cloud. The query processor maintains a model of a DBN that is personalized to the phone user. This model is stored as a file in an XMLBIF format which is an interchange format that is recognized by most bayesian inference softwares. In our implementation, we use the well known variable-elimination algorithm's implementation for probabilistic inference provided by the Java Bayes package [6]. One downside is that accurate inference using variable-elimination can take exponential time in the number of variables in the bayesian network. While this was not an issue for the number of contexts in our current implementation, this may be a bottleneck as the number of contexts increase. Future implementations of *CQue* will use fast approximate

inference algorithms such as importance sampling and MCMC simulation to optimize inference performance on the phone [153].

Query Plan: As discussed in §4.4.2, *CQue* uses a dynamic query plan where the decision regarding what contexts to execute given a certain energy budget is periodically re-visited to capture dynamics in the incoming context stream. Such dynamic adaptation can be expensive to perform on the phone since it depends on the number of contexts in the DBN. So, for infrequent plan evaluation or if the size of the DBN is small, our implementation performs such planning on the phone, but otherwise offloads the computation to the cloud.

DBN Learning: While the execution engine can execute on a mobile phone, learning the DBN is more computationally intensive and requires cloud support. We implemented the DBN learning algorithm using partial observations by modifying the WEKA package [73]. *CQue* offloads the process of learning the DBN to the cloud by sending the appropriate context instances to the server at the end of each day. The DBN is learnt at the end of every day in the cloud and is sent back to the *CQue* framework in XMLBIF format. Over time, the frequency of updating the DBN reduces as its structure and parameters stabilize. Also, to facilitate faster learning of DBN parameters, we maintain a configuration file that provides initial parameter estimates for the DBN, based on average numbers from a general population. The configuration file has entries for each context containing a list of positive and negative causes for it along with the conditional probability. For example, the negative cause list for the context *at office* looks like: *at home,0.99; at store,0.85; at restaurant,0.93; driving,1.0*.

Query Processor: We implemented the context engine as a background service running on android phones. This service is responsible for activating context-inference algorithms and appropriate sensors, and providing the context values generated by the algorithms to the query processor. Communication between the query processor and the context engine uses Android IPC. Our implementation currently supports semantic location contexts, social contexts and activity contexts: a) **Activity contexts**: *walking*, *driving* and *stationary*, b)

Social contexts: *with friends*, *with colleagues* and *alone*, and c) **Location contexts:** *at restaurant*, *at home*, *at office*, *at store* and *on street*.

Our implementation uses decision-tree classifiers for activity recognition. We used well-known set of features extracted from the 3-axis accelerometer and gyroscope readings. The following features were computed for accelerometer readings along each of the three axis: mean, standard deviation, mean-crossing rate, energy given as normalized sum of squared discrete FFT component, peak frequency in FFT and peak energy. In addition, we computed correlation between each pair of axes. The gyroscope readings were used to compute mean and standard deviation of angular velocity around each of the three axis. For classifying social and location contexts, we used a user-provided mapping from bluetooth devices to the social context and a mapping from WiFi access points to the corresponding semantic location (either user provided or using publicly available databases). Unlike activity contexts, the classified social and location contexts have higher confidence associated with them. While we restricted ourselves to the above contexts due to the nature of our datasets, our architecture is general is designed to accommodate other context-engines that may be added by practitioners.

User Interface: The *CQue* framework provides an interface to the user to specify an energy budget and preferences for interruption (during training) and privacy. The energy budget can be specified as a fraction of a full battery that may be used for running *CQue* and a battery threshold below which *CQue* should be stopped. For interruption preferences, a user can specify the total interrupts permitted per day while training the DBN. For privacy preferences, a user can select private contexts, select a level of protection against adversary from low, medium and high where these values map to values for threshold for change in adversarial confidence $\delta = 0.4$, $\delta = 0.25$ and $\delta = 0.05$ respectively. Additionally, user can select the applications for which these preferences apply.

4.6 Experimental Results

In this section, we describe the set of experiments performed to evaluate our framework. We first describe the data sets and the evaluation metrics used for the experimental evaluation. Next, we present our analysis of privacy for various contexts.We then look at the energy-accuracy tradeoffs and demonstrate the benefits of using DBN over context-classifiers using a set of experiments, and conclude with an evaluation of how various parameters such as delay, and interruption budget impact results.

4.6.1 Data Sets and Evaluation Metrics

4.6.1.1 Datasets

In order to conduct our experiments, we used following two datasets:

Reality Mining Dataset: This data set contains data collected continuously for 100 students and staff at MIT over academic year 2004-2005. The duration of user traces varied from 30 days to 269 days. In our evaluation, we used data from 37 users who had data for at least 10 weeks.

Activity Dataset: While the reality mining dataset has rich multi-sensor data, it relies on self-reports for activity classification, hence it does not allow us to understand how accuracy of activity classifiers can be improved through the use of *CQue*. To address this, we collected a trace of user contexts for two weeks from seven users. Each of the seven users provided about 50 context labels per day and contributed raw data from the following sensors: accelerometer, gyroscope, compass, bluetooth, WiFi and GPS. Since there could be labeling error, we corrected activity context information using GPS information and manual correction at the end of each day. All users providing data were graduate students. We used data from four users to train the activity classifiers. For testing, we used these classifiers to classify activities for the remaining three users. For each user in the test set, we divided their data into two weeks — in the first week, a personalized user-specific DBN is trained using the outputs of the classifier algorithms, and a few human labels obtained

from the ground-truth set (depending on the interruption limits, and label selection scheme), and the data in the second week is used to evaluate *CQue*.

4.6.1.2 Set of Contexts

In both the datasets, we have a set of contexts that includes location contexts like *at home*, *at office*, *on street* and three activity contexts, namely, *walking, driving* and *stationary*. For social contexts, we use the groups of bluetooth devices in the close proximity of a user as the social context. In Activity dataset, we have these groups explicitly labeled as *friends, colleagues, roommates, spouse* whereas in Reality mining dataset, we do not have explicit labels but we identify these groups as *group-1, group-2* and so on. For each context, the corresponding classifier outputs *true* or *false* value along with the confidence value associated with the classification output. We use these values as input in DBN.

4.6.1.3 Evaluation metrics

We use three performance metrics in our evaluation: 1) **Accuracy**, which is computed as the fraction of these contexts that are correctly classified, 2) **Confidence**, or the probability of the most likely value for the context provided by the DBN, and 3) **F-Measure** which gives the harmonic mean of *recall* and *precision* (higher score is better). **Recall** for a context is computed as the ratio of number of times the context was correctly *classified* as *true* to the total number of times the context was actually *true* in the data. **Precision** for a context is given as the ratio of number of times the context was correctly classified as *true* to the total number of times the context was classified as *true*.

4.6.2 Cost-Accuracy Tradeoffs using DBN

One of the benefits of a DBN is that it allows a tradeoff cost for accuracy — by understanding the relations across different contexts, an inference algorithm can be turned off to reduce overhead. While the "cost" can be different depending on the sensing and communication needs of the inference algorithm, or the burden of user input, we use a sim-

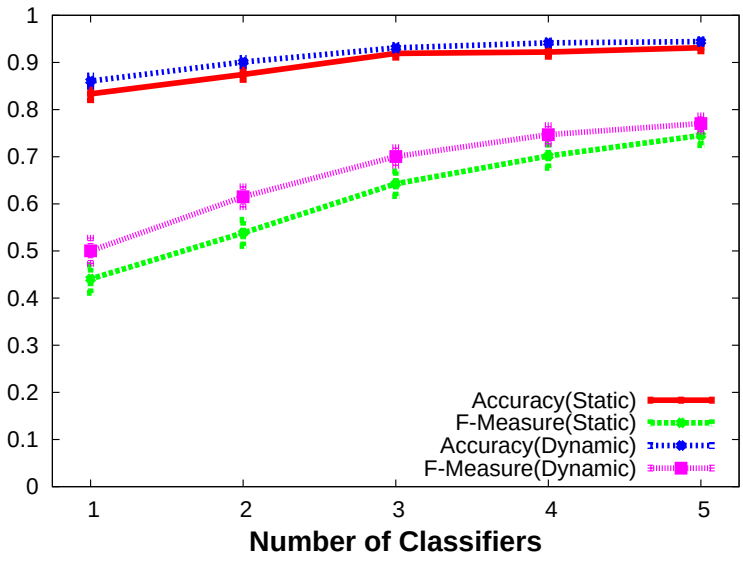


Figure 4.4. Effect of varying number of executing context classifiers when number of queries (≥ 6) is higher than the number of executing contexts. The figure shows aggregate accuracy and F-measure over 37 users for static and dynamic plan execution.

plistic model where we assume that all classifiers have equal cost. This model provides an intuitive understanding of the cost-accuracy tradeoffs.

Classifiers $<$ Queries We first look at the case where the system runs fewer classifiers than the number of queries and exploits the context-relationships to answer context queries for which there are no observations coming from the classifier. This case demonstrates that the DBN can be used to answer the context queries in expectation which is not possible otherwise using classifiers alone. We evaluate this model using the Reality Mining dataset.

In this experiment, we consider all the contexts that we support to be in the query set. The number of queried contexts varied between 6 to 11 for the 37 users in the Reality Mining dataset. We then vary the budget such that it can execute between 1 to 5 classifiers. Based on the budget, the set of classifiers are chosen that can provide maximum information gain as described in §4.4.2.

Figure 4.4 shows the accuracy and F-measure averaged over 37 users as the number of classifiers increases. We see that both these metrics improve as we increase the number

of classifiers, and even with two classifiers executing, we get fairly good accuracy and F-measure. Thus, significant benefits can be obtained with only a small number of classifiers by leveraging a DBN model of the relations across contexts.

It might seem surprising that very high accuracy can be achieved even with a single classifier. In fact, this is because most contexts have biased distribution (e.g. the stationary state is true $(> 90\%)$ of the time), hence high accuracy can be achieved by just using the model in expectation with no classifiers executing! But this gives poor recall, precision and consequently the F-measure is low, and more context input is required to improve these metrics. Figure 4.4 shows that DBN achieves 75% F-measure and 94% accuracy using only 4 classifiers resulting in at least 33% cost reduction.

Figure 4.4 also compares the use of a dynamic plan where the query plan is re-evaluated every 20 minutes vs a static plan where the query plan is evaluated once. The dynamic plan provides better accuracy and F-measure than static plan across the board. This is because the dynamic plan selects appropriate set of classifiers using value of information provided by the observed classifier outputs in real-time whereas static plan selects the set of classifiers without considering the real-time classifier outputs.

Classifiers $>$ Queries We now look at the benefits of using a DBN when the number of classifers that execute are greater than the number of queries. We use the Activity dataset in this study, since we have raw data for activity inferences, which are typically the contexts with highest inaccuracy. We look at the performance for the three activity contexts (walking, driving and stationary), which have highest uncertainty. We compare two mechanisms: a) using just the activity recognition classifiers, and b) using all the context classifiers with DBN. We show results for three representative users in the dataset.

Figure 4.5(a) shows that accuracy is worst when we use only classifiers alone, and the use of DBN improves improve accuracy significantly. The improvement in accuracy by using DBN for users U1 and U2 is above 24% for both *stationary* and *driving* contexts. Improvements are small for *walking* activity (2%) since DBN neither observed any context

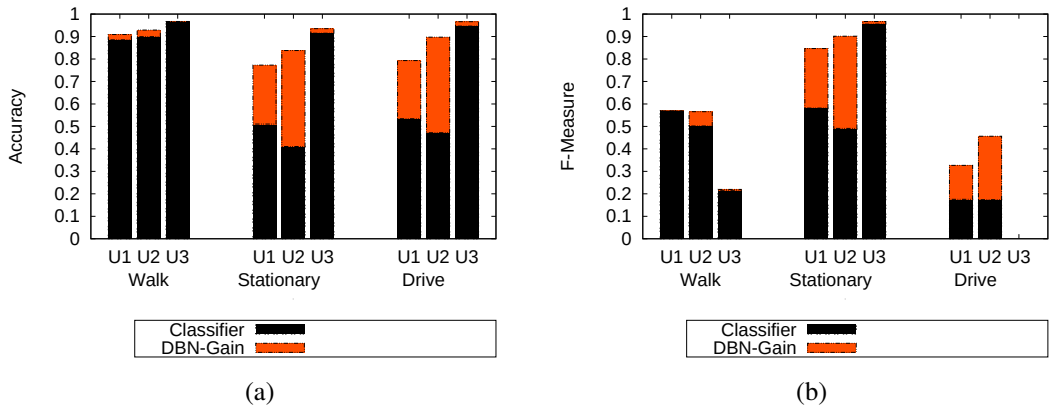


Figure 4.5. Accuracy and F-Measure for various activity classes for Classifier only and gain provided by Classifier+DBN mechanism on data from 3 users. Results show that using the DBN is beneficial in most cases, except in the case of rarely occurring contexts.

strongly related with *walking* nor the temporal consistency. This was because *walking* was a rare context and rarely lasted for longer than a few minutes.

Figure 4.5 (b) shows F-measure for various classes. We see that this metric generally improves since the main role of the DBN is to correct the outputs provided by the classifiers and hence, improve recall and precision. Since the DBN performs corrections using the temporal consistency, the recall value for a certain context may drop if the context is rarely seen and lasts for a very short duration. In our traces, we observed that the *walking* activity was infrequent. In such a case, if the classifier does not generate output with high confidence the DBN can smooth out the walking context resulting in reduced recall but higher precision. As a result, F-measure does not see significant gain. In contrast, the *driving* context though rarely seen as compared to *stationary* lasts longer and hence, sees improvement in recall and hence, improves F-measure.

4.6.3 Evaluating Privacy

In this section, we evaluate two aspects of how *CQue* can be useful in dealing with privacy breaches due to correlations across contexts.

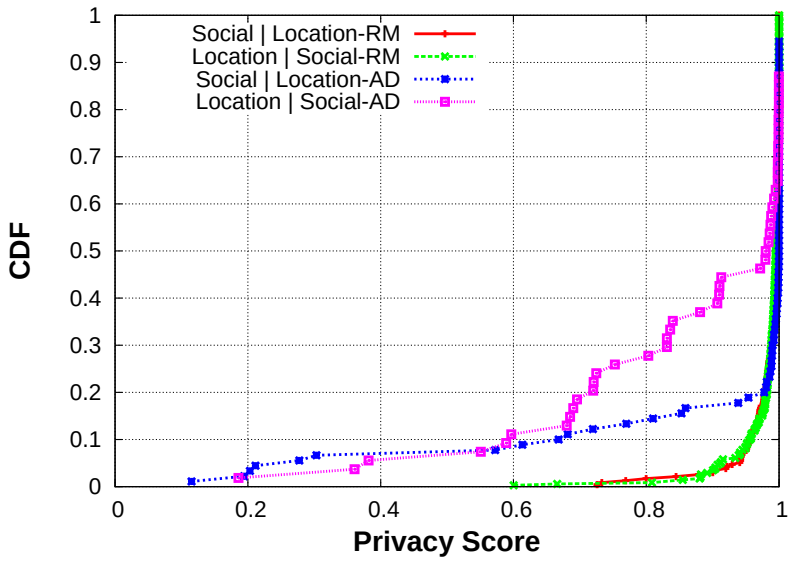


Figure 4.6. Cumulative distribution of privacy scores for cross-context pairs in i)Reality Mining dataset (RM), and ii)Activity dataset(AD). From the distribution of privacy scores, we can conclude that there is significant cross-context privacy leakage in Activity dataset whereas no such leakage is observed in Reality Mining dataset. This emphasizes the importance of personalized DBN to evaluate privacy for a user.

Location-Social Relations: In our first experiment, we look at a two sensor scenario and understand the correlations between these contexts across different datasets. Recall that location is obtained through GPS and social interactions through bluetooth. We use *CQue* to understand how a privacy breach can occur for a *privacy-sensitive* context through indirect observations. Here, we assume a *strong adversary* that has access to the personalized DBN for a user. Potentially, this is possible if various apps on a phone that observe different contexts decide to collude and combine their data to generate a complete DBN.

Given such an adversary, a *privacy-sensitive* context can be inferred by observing one or more other contexts. Using our *privacy score*, we can rank contexts in the decreasing order of its capability to infer *privacy-sensitive* context. In this experiment, we look two scenarios: a) all cases where we use one location context as a private context (p) and a social context o as observation, and b) the reverse scenario where a social context are private and location is observed. We evaluate *privacy score* $D(p|o)$ for each pair $\langle p, o \rangle$ of contexts.

Table 4.1. Cross-context privacy leakage in location and social contexts for i)Reality Mining dataset(RM), and ii)Activity dataset(AD). For activity dataset, we see significant increase in accuracy for all the location contexts when all the social contexts are observed and vice versa. For reality mining dataset, accuracies do not change with cross-context observations.

(a) Privacy leakage in All Location Contexts

Metric	Observation	
	None	All social contexts
Accuracy (RM)	73.9±3.0	74.81±2.73
Accuracy (AD)	61.11±5.55	88.9±6.26

(b) Privacy leakage in All Social Contexts

Metric	Observation	
	None	All location contexts
Accuracy (RM)	73.86±2.9	74.18±2.82
Accuracy (AD)	64.01±15.29	87.3±5.94

Figure 4.6 gives distribution of privacy scores for these scenarios for the Reality Mining and Activity Datasets.

The results are interesting — we see that there is considerably higher correlation across location and social context in the case of the activity dataset than in the reality mining data. This is also reflected in Table 4.1, which shows the accuracy of inferring location/social contexts only based on the prior distribution of these contexts v.s. observing the other context. The results show that there is only a small change in the case of the RM dataset, but the accuracy increases by more than 25% in the case of the AD dataset. In other words, an adversary would be able to infer an individual's location with substantial accuracy if they only had access to the bluetooth information on the AD dataset. Our explanation for these results is that bluetooth usage is far more prevalent in recent times, therefore the correlations have increased. Overall, our results show that *CQue* can be used to provide intuition about the correlations across contexts, thereby enabling more informed decision about what to expose.

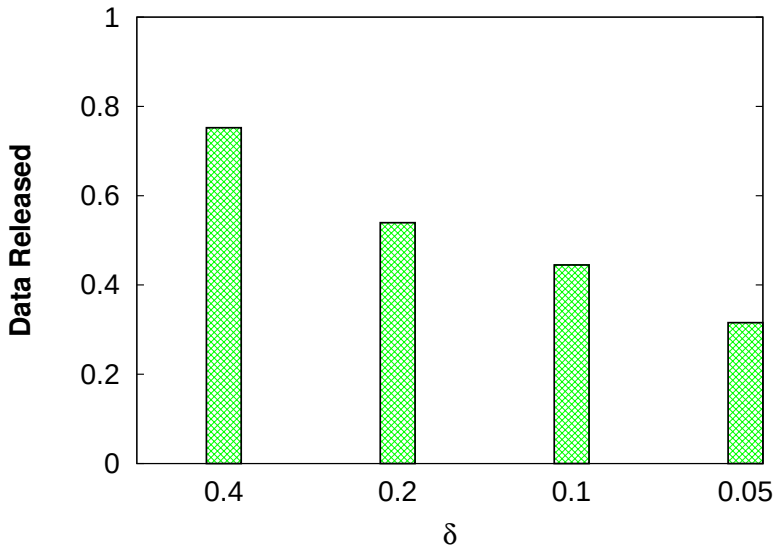


Figure 4.7. Fraction of data that can be released to prevent change in adversarial confidence greater than δ for a private context *at home*.

Suppression Policy for Privacy: In our second experiment, we look at how *CQue* can be used to implement a real-time suppression policy for protecting privacy. Our policy is intended to be representative and illustrate how the DBN may be used, and we make no formal claims regarding its privacy properties. In our experiment, the user can define a policy that permits releasing a maximal set of context observations such that the change in adversarial confidence for private context upon observing this set is less than threshold δ. As a result, some of the non-private contexts can be suppressed in real-time to control the change in adversarial confidence. Thus, it results in lower utility for the applications seeking contexts. Figure 4.7 gives an example showing the fraction of context data that can be released as a result of this suppression for various values of δ where we chose *at home* as the private context and rest of the contexts as queries. A smaller value of δ results in much tighter privacy control but releases too few contexts.

Table 4.2 shows the fraction of data suppressed in context of each category: *social, location,* and *activity* for value of threshold $\delta \in \{0.1, 0.4\}$. In this case, we choose *at friend's place* as a private context available in Activity dataset. We can see that *CQue*

suppresses in an intelligent manner where it suppresses highly correlated social context more frequently than the less correlated location or activity contexts. If we were to use a mechanism like MaskIT [70], it would suppress every context at the same level. Since *CQue* releases more contexts, it results in higher utility for the applications and the users who use these applications.

In conjunction, these experiments demonstrate the potential use of *CQue* both for understanding privacy implications of releasing a context, as well as to implement privacy policies that leverage relations across contexts.

Table 4.2. Table showing fraction of data suppressed in contexts of each category to prevent change in adversarial confidence greater than δ for a private context *at friend's place*.

Threshold δ	Social Context	Location Context	Activity Context
0.4	27.67%	13.99%	17.52%
0.1	26.81%	16.01%	17.35%

4.6.4 DBN with Personalized Classifiers

While all our previous results have assumed classifiers to be black-box code that cannot be modified, we now look at the case where we can personalize the classifiers using the human input obtained by the DBN during its learning phase. This feature leverages the fact that contexts like user activity can see significant benefit with personalization as there are differences across individuals.

In our experiment, we personalize the classifiers for activity contexts in AD dataset by training it with additional user-specific labels obtained from the user's data for training DBN i.e. the first week of user's trace. Figure 4.8 gives ROC curves for each activity class for both personalized and non-personalized classifiers. The ROC curve shows the performance of the classification mechanism when the discrimination threshold is varied. A classification mechanism with a larger area under the ROC curve indicates that it is a better mechanism. In figure 4.8, it can be seen that the personalized classifier tends to have

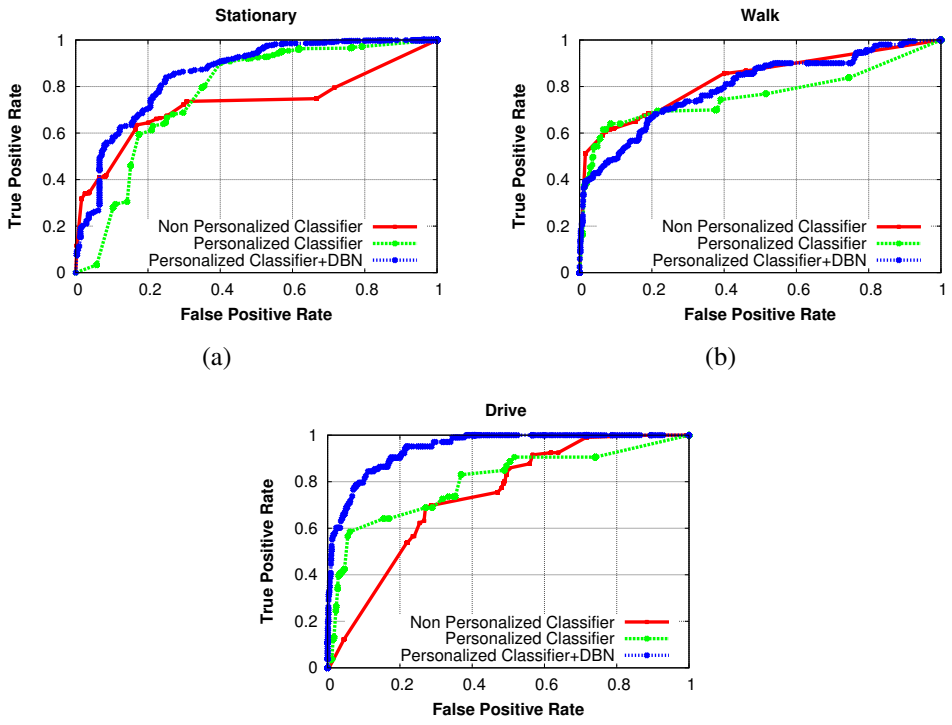


Figure 4.8. ROC curve for each activity class for i) non-personalized classifier, ii) personalized classifier, and iii) DBN using personalized classifier.

larger area under the ROC curve compared to the non-personalized classifiers. Moreover, the area under ROC curve for DBN is always larger than the area for either of the classifier-based schemes. Except for *walking* which has a small operating region where the classifier performs better than DBN, the DBN clearly dominates the classifiers over all operating regions in the ROC. This shows that classifier personalization together with a DBN can provide the best method to improve accuracy.

4.6.5 Impact of Delay

The delay tolerance period provided by queries provides a tuning knob that can influence results provided by the query engine. In this experimental setup, we study the impact of increasing value of this parameter on activity contexts in AD dataset. In general, whenever the DBN observes a change in context value output by the classifiers, the confidence

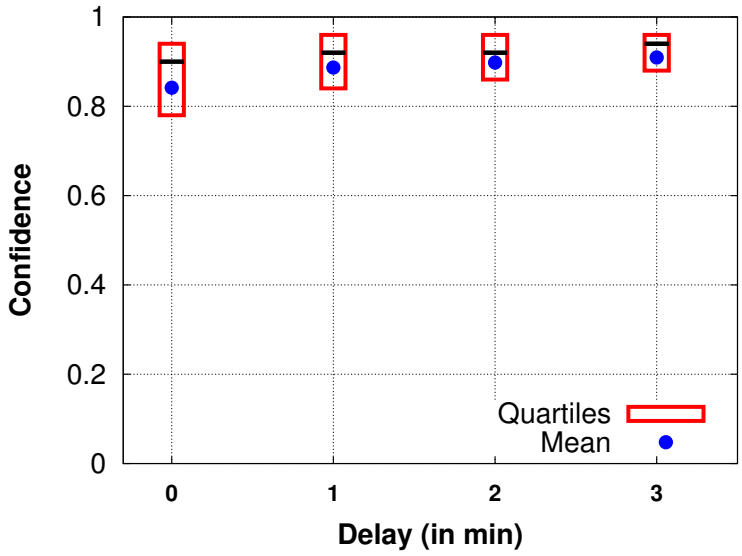


Figure 4.9. Effect of delay on confidence distribution, as we increase delay-tolerance for queries from zero to three minutes. We consider all the query results that had conf< 0.98 when DBN was used with 0 delay in response (35% of the results), and show how summary statistics for this group improves as we increase the delay-tolerance for the queries from 0 to 3. The columns in the graph show the range of confidence between the 1^{st} quartile and the 3^{rd} quartile, the black bars give the 2^{nd} quartile or the median confidence whereas the blue circular dots give the mean confidence observed.

of the DBN may drop. This confidence improves with more observations obtained in the future. Thus, the main role of using a delay-tolerance period is to improve confidence in the DBN output. Figure 4.9 shows the distribution of confidence when delay-tolerance is increased. Apart from boosting confidence, the use of a delay-tolerance window reduces the intermittent misclassifications by the classifiers. We observe that a delay-tolerance period of three minutes identifies 44 additional misclassifications.

4.6.6 Implementation Benchmarks

We benchmark our implementation on Samsung Galaxy Nexus phone running Android OS version 4.1.2. This device has 700MB RAM and 1.2 GHz dual-core processor. In this evaluation, we look at the following two metrics: i) *inference time* i.e. the time required to make a probabilistic inference about a context given a set of evidences from the classifiers,

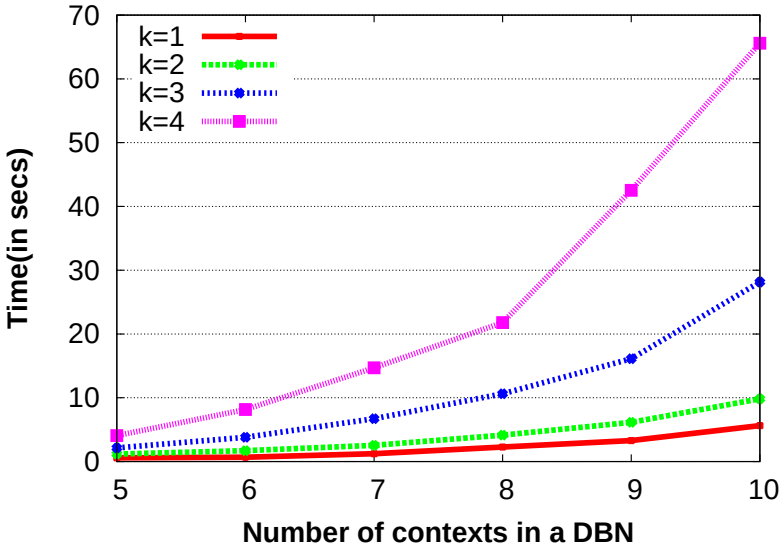


Figure 4.10. Time to generate a dynamic plan selecting best k classifiers where k varies from 1 to 4 and the query set consists of all the contexts available in a DBN. The figure shows the variation in time as the number of contexts in a DBN, N is varied from 5 to10. The plan evaluation time increases linearly in k and exponentially in N.

and ii) *plan evaluation time* i.e. the time required to generate a dynamic plan for a given set of queries. All experiments are done using the Reality Mining dataset.

Inference Time: Since the computational complexity of an accurate inference algorithm to infer a context is exponential in number of parent nodes for the context in DBN, the actual required time for probabilistic inference varies from context to context and from user to user due to variations in the personalized DBN model structure. Thus, we measure *inference time* using traces from multiple users with varying size of the DBN model. In this experiment, we select one context as a query and use rest of the contexts as observations. We repeat this experiment for all the available contexts for each user. The user traces consisted of 5-11 contexts. Upon evaluation on Galaxy Nexus phone, we observed an average *inference time* across all the contexts and the users to be 9.31 ± 0.71ms. Also, the fastest context had an average *inference time* of 2.13ms for a user trace consisting of 7 contexts whereas the slowest context had an average *inference time* of 21ms for a user with DBN model consisting of 11 contexts.

Plan Evaluation time: We evaluate *plan evaluation time* on Galaxy Nexus phone for multiple user traces consisting of 5-11 contexts. In this evaluation, we set all the contexts as queries and measure time to generate dynamic plan selecting best k classifiers where $k = 1, 2, 3, 4$. We note that the time complexity of generating dynamic plan with best k classifiers is linear in k and exponential in N where N is the number of contexts in the DBN. We show the results in Figure 4.10. From the figure, we can see that the *plan evaluation time* increases quickly with the number of contexts in DBN and requires more than a minute to evaluate best 4 classifiers for a DBN consisting 10 contexts. This might still be practical if we generate a plan only when the set of queries change, which might be infrequent. However, if the plan is re-evaluated every few minutes, this approach quickly becomes infeasible. This problem can be addressed in two ways: i) using an approximate inference algorithm with better time complexity, which is part of our ongoing work, or ii) evaluating the plan in the cloud when its complexity exceeds a certain threshold. Of course, dynamic plan evaluation on the cloud has the overhead of data transfer of the current classifier outputs with the appropriate confidence values. However, the DBN model itself is not required to be sent to the cloud as it is already trained on the cloud.

4.7 Discussion

In our results, we have demonstrated the utility of *CQue* framework for energy-efficient and accurate continuous sensing of user context while understanding the privacy implications. In this section, we discuss potential opportunities for improvement and various interesting questions posed by our framework for further exploration.

Feature-level DBN structure: Our current implementation of the DBN supports fusion of high-level categorical contexts that are the outputs of classifiers. While this is beneficial in that we can take advantage of extensive work on the design of individual activity or behavior classifiers, the downside is that if a classifier outputs inaccurate uncertainty estimates, we may have incorrect inferences that can impact the accuracy of all the re-

lated contexts. One way to address this issue may be to use both categorical contexts from classifiers as well as low-level signal features extracted from the sensor data (e.g. spectral features, mean, variance, etc), and to use combination of these different inputs towards information fusion for context inference.

Privacy leakage: We have touched upon the issue of privacy leakage by leveraging correlations across contexts, but *CQue* does not capture all possible attacks that can use multi-sensor inference techniques. The privacy leak detection in *CQue* for a sensitive context is based on a model of adversary who is aware of the context relationships in DBN, but it is possible to have an undetected privacy leak if there are relationships that cannot be modeled by the DBN. For example, the DBN can model and evaluate privacy leakage over a short window of W adjacent time steps but it cannot detect privacy leaks before W time steps. A powerful adversary with alternative long-term models may still be able to execute privacy attacks. Thus, much work remains before we fully understand how to effectively understand the extent of privacy leakage that is possible using sophisticated multi-sensor inference techniques.

Duty-cycling classifiers: One question that we have not fully explored is the interaction between duty-cycled classifiers and the DBN. Several techniques have been proposed to leverage temporal consistency in classifiers for duty-cycling *i.e.* if the context value is not expected to change for some time period, then reduce the sampling rate of the sensor. *CQue* does not currently take advantage of such approaches to duty-cycle classifiers. One method to incorporate such approaches is to use the last observed classifier output as evidence in DBN for next few time-steps while the classifier is inactive. Another interesting question is whether the DBN can be used to learn the duty-cycle for each classifier. Intuitively, if the current state of DBN indicates that some context, say c, is unlikely to change soon then we can have a lower duty-cycle for context c where its classifier will remain inactive for a long time.

DBN learning time: How much training data is needed to fully learn an accurate DBN structure? Although one week of trace proved to be sufficient to learn good structures in our experiments, it might not be the case in general since there may be some longer-range patterns that cannot be captured, and the patterns may change over time. To answer this question, we need larger-scale datasets across more individuals that can help us understand how to learn the DBN effectively across a population while limiting burden to provide labels, as well as the learning period.

4.8 Conclusions

Context awareness distinguishes smartphones from traditional computing platforms and can play a key role in creating smart mobile applications. We argue that user contexts across several dimensions are correlated, and if we can learn these correlations in a personalized manner, we can leverage it to improve performance as well as understand the privacy implications of revealing seemingly unrelated contexts. In this chapter, we described our context querying framework, *CQue*, that exploits probabilistic relationships across contexts for each individual user to improve energy-efficiency, accuracy and reliability of context sensing, as well as the ability to understand privacy implications.

The ability to understand relations across contexts can have far-reaching consequences. A growing area of healthcare is personalized behavioral monitoring using smartphones and on-body sensors, where behavioral scientists seek to understand relations between addictive or unhealthy behavior, and the individual's state in the real world (activities, social interactions, location, etc) . Our work can enable such understanding, and is a step towards a "big data inference toolkit" for mobile sensing.

CHAPTER 5

PREPP: PRACTICAL PREDICTION AND PREFETCH FOR FASTER ACCESS TO APPLICATIONS ON MOBILE DEVICES

Mobile phones have evolved from communication devices to indispensable accessories with access to real-time content. The increasing reliance on dynamic content comes at the cost of increased latency to pull the content from the Internet before the user can start using it. While prior work has explored parts of this problem, they ignore the bandwidth costs of prefetching, incur significant training overhead, need several sensors to be turned on, and do not consider practical systems issues that arise from the limited background processing capability supported by mobile operating systems. In this chapter, we make app prefetch practical on mobile phones. Our contributions are two-fold. First, we design an app prediction algorithm, APPM, that requires no prior training, adapts to usage dynamics, predicts not only which app will be used next but also when it will be used, and provides high accuracy without requiring additional sensor context. Second, we perform parallel prefetch on screen unlock, a mechanism that leverages the benefits of prediction while operating within the constraints of mobile operating systems. Our experiments are conducted on long-term traces, live deployments on the Android Play Market with installs on more than 50,000 phones, and user studies, and show that we outperform prior approaches to predicting app usage, while also providing practical ways to prefetch application content on mobile phones.

5.1 Introduction

The success of smartphones has resulted in an explosive increase in the number of apps available in app marketplaces. Currently, Apple's iOS app store and Google's play

store both have more than 1,000,000 apps available while the Window Phone app store is catching up with more than 200,000 available apps. Among the most popular apps in the marketplace are real-time content-driven applications such as News, Email, Facebook, Twitter, and others that provide timely information to users.

While the utility provided by installed apps has made the smartphone an indispensable companion to users, it comes with a new set of user experience challenges. Mobile phones are largely used during idle times in between real-world tasks, and users want information instantly during these periods. Yet, the reality is that delays in refreshing popular apps like *Email, Facebook, Twitter, News* and *Weather* that rely on network connectivity to download app content is often of the order of seconds or tens of seconds. The fickle nature of cellular connectivity means that these apps can have highly variable wait times before the content is displayed to the user. Prior work has shown that the top apps available in the market have an average network latency greater than 11 seconds [186]. In short, we face a dire challenge — app loading times are increasing as apps fetch more content from the cloud, and the cellular data networks continue to play catch up with these trends.

There have been several prior efforts at tackling the increase in app usage latency, but ultimately they fall short of addressing a plethora of practical considerations. First, prior work ignores the freshness of loading predicted apps — for example, *Email* may be predicted to be the next application to be used, but should new emails be prefetched now or an hour later? Second, several past efforts propose to modify either the mobile OS or apps in order to speed up launch time [164, 186] — these methods lack easy deployability through app stores or require modifications of 3rd party app source code. Third, prior approaches propose to use contextual data such as location to improve prediction accuracy. However, users are often reluctant to install apps that require contextual information, in particular location context, diminishing the broader appeal of such methods. Fourth, prediction methods proposed in prior work often have high training requirements (several weeks of data), but users expect system adaptation within days, not weeks.

In response, we have developed *PREPP*[1], a preeminently practical approach to prefetch. By practical, we mean that the two major facets inherent to fast app launch — App Prediction, and Prefetch Execution — have been designed for performance on today's stock mobile devices with users seeing immediate launch time speedups. At the core of our system, we have two models. The first is *App Prediction by Partial Match (APPM)*, a *prediction model* that adaptively learns the probability distribution of the apps to be used next and requires no privacy-sensitive or power-hungry contextual information like location. The second model is a *Time Till Usage (TTU) temporal model* that utilizes the learnt distribution of time spent before app use to estimate appropriate time for prefetch in a bandwidth cost-aware manner – an aspect which is of key importance for prefetching dynamic application content.

The outputs of the prediction and TTU models are combined in *PREPP* to dynamically decide when to prefetch app content, thereby delivering significant fresh content with low energy overhead, while requiring neither app nor OS modifications. We are able to accomplish this by designing *PREPP* to take advantage of app foreground-background execution semantics that are common across mobile OSs while operating within their constraints.

We use a combination of a deployment in the Android app store with 7,630 active users, controlled user study with 22 participants, trace-based analysis and micro-benchmarks of an Android smartphone to show that *PREPP*:

- Reaches over 80% accuracy when predicting a top 5 ranking for the next app to be used. In fact, we perform better than widely used approaches such as MRU, as well as previous technical proposals [164, 186] while requiring neither privacy-sensitive location information nor offline training time.

- Delivers content to the user that is on average fresh within 3 minutes, and even up to 1.5 minutes fresh on average for some users.

[1] Predictive Practical Prefetch

- Causes negligible additional energy expenditure.

5.2 Background and Shortcomings

In this section, we outline three drawbacks of existing systems that we address in this work.

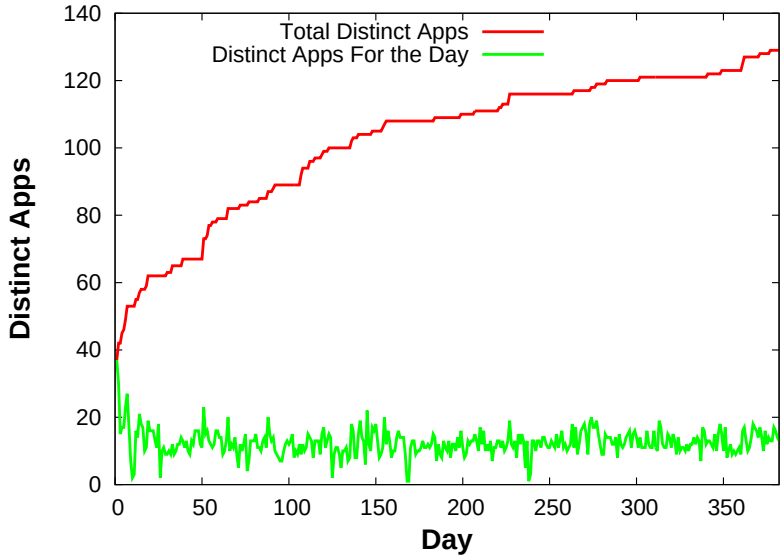


Figure 5.1. Cumulative number of distinct apps used by a representative user and the number of distinct apps used each day by the same user over a period of 382 days.

Slow User Adaptation. Approaches that use prediction-based strategies have suffered from prediction model training cold-start: once a user starts to use the system, it has taken as long as 3-6 weeks of data collection before meaningful predictions can be made [186]. Unfortunately, for mobile systems where time and attention spans are limited, users expect benefits within days, not weeks, particularly since app usage patterns continually change over time. For example, long-term traces of app usage (e.g. LiveLab iPhone usage dataset [163]) show that users download more than one hundred apps each year, and use them only for a few weeks or months. Figure 5.1 plots how app usage varies over the course of a year for a representative user in the RiceLabs iPhone usage dataset [163]. Clearly, the cumulative number of distinct apps increases over the period of a year, motivating the

need to adapt. Therefore, prediction systems that require lengthy training periods are not suitable.

Lack of Freshness. Prior work only answers the question of which application will be used next and not *when* it will be used. Predicting when an application will be used is particularly important when optimizing freshness of prefetched application content. For example, *Email* may be predicted to be the next application to be used, but should it be prefetched now or an hour later? Prefetching content frequently would incur bandwidth and energy overhead, and infrequently would result in stale content. Therefore, temporal prediction is an important missing piece for cost-aware prefetch.

Ease of Deployability. Mobile OSs have unique constraints that make it difficult to implement a practical prefetch system. Hence existing approaches either propose modifications to mobile OSs [186], or modifications to the app source to prefetch content [77]. However, this hurts their deployability — even with an open source system like Android, manufacturers often lock down the OS bootloader such that unsigned OS changes are not installable. Similarly, a potential approach that aimed to modify only the server push decision algorithm would also lack deployability because push notification infrastructure is under OS vendor control [81] and is not accessible for modification. Finally, approaches that propose to modify only apps are challenged with the necessity for access to and modification of 3rd party app source [77]. These constraints from the OS and Apps are unlikely to change anytime soon. While modifications to the OS or apps represent useful design alternatives, a key question that has remained unanswered is whether it is possible to design an easily deployable, store-compatible prefetching system for mobile devices.

5.3 Requirements

Our goal is to design *PREPP*, a practical prediction and prefetch system that satisfies the following key requirements:

- *PREPP* should require a *small training* overhead, and be able to converge to high accuracy within a small number of days of use, while adapting to changing usage patterns and applications quickly.
- *PREPP* should maximize freshness for applications that need to update their content, while minimizing the resource overhead incurred in terms of network access and energy cost.
- *PREPP* should work on unmodified off-the-shelf phones and bring speedups to existing apps, thereby making it immediately deployable to mobile app stores, installable on current mobile OSs and usable by all mobile users.

The next three sections address these requirements.

5.4 PREPP System Design

The problem that we solve in *PREPP* can be defined as follows: given a sequence of content-based apps that a user has used, and the times when the user has used them, can we prefetch content in a timely manner while keeping the overall network prefetch costs low. This high-level problem can be divided into three sub-problems: a) can we accurately predict what app is going to be used next? b) can we predict when that app is going to be used next? and c) can we decide when to prefetch to maximize freshness while keeping network access costs bounded? We describe next the three key system components in *PREPP* that accomplish these goals.

5.4.1 App Prediction Algorithm

The problem of app prediction is as follows: given the app usage sequence for an individual, can we predict what app the user is likely to use next? Intuitively, this problem has parallels with text compression, where the preceding character sequence can be used to determine the most likely next character, which in turn can be leveraged to compress the text. (For example, a character following the sequence *natio* in English language is highly

likely to be *n*). Similarly, one can view each app as a "character" and the sequence of app usages as a character stream, and apply text compression techniques to our problem. We now look at what text compression algorithm to leverage, and how to adapt it to our needs.

One of the widely used methods in text compression is Prediction by Partial Match (PPM) [41]. At a high level, PPM operates by scanning character sequences, and building up a variable-length Markov-based predictor on the fly. PPM uses the longest preceding character sequence to compute the conditional probability distribution for the following character. To compute this conditional probability distribution, PPM maintains frequencies for characters that have been seen before in all prefix-sequences that have occurred before, up to some maximum order, where order is the length of the prefix. For example, PPM with order 3 maintains frequencies for all prefixes of length 3, 2, 1, and 0 as shown in Table 5.1. To predict what character is likely to appear next in the stream, PPM computes the conditional probability for each character given the highest order node having a non-zero frequency for the character and scales it using weights, with weights being highest for the longest prefix, since longer matches often provide more precise contexts for character prediction.

Since PPM is designed for text compression rather than app usage patterns, we need to address some of the differences between these two cases. The main difference is that in text prediction, the longest prefix is often the most relevant, whereas in app usage, both long and short prefixes can be highly informative. For example, we found that in some cases, a user tends to strongly favor recently used items whereas in other cases, a user exhibits highly sequential app usage behavior.

We address this issue by allowing the prediction accuracy from the different orders in PPM determine how they are weighted. Thus, we do not make strong assumptions about what length prefixes are likely to be useful, and instead let the empirical accuracies from prior predictions dictate which nodes are favored. With accuracy-driven weights, we compute weighted sum of the outputs of predictions from each of the prefix nodes. In

Table 5.1. Example of PPM nodes for current context sio in string *"sionofionsionofofioroforofsiotofsio"*. Nodes for prefixes *sio, io, o* and empty sequence ϕ along with character frequencies, sum of frequencies for characters in node that are not present in higher order node (T_m), PPM weights $w_m = \prod_{i=m+1}^{3} \frac{1}{T_i+1}$ and APPM weights given by the prediction accuracy observed for the node are shown.

Order	prefix	character frequency						T_m	PPM w_m	APPM w_m
		n	t	r	f	s	o			
3	sio	2	1					3	1	0.5
2	io	3	1	1				1	$\frac{1}{4}$	0.5
1	o	3	1	2	6			6	$\frac{1}{4.2}$	0.64
0	ϕ	3	1	2	6	4	12	16	$\frac{1}{4.2.7}$	0.38

this manner, APPM's vocabulary is personalized to each user, and can learn their preferred patterns over time. Table 5.1 shows an example scenario where APPM weights are given by accuracy obtained using the top-2 predictions for each node.

In summary, APPM is an appealing app prediction algorithm that satisfies our key requirements: (1) it has low training overhead — incremental training requires O(1) operations that involve a simple increment of counts, (2) it continuously adapts to changes in the sequence's character distribution by discounting older history, and (3) it learns what prefixes are likely to be more useful for prediction and places its bets appropriately.

5.4.2 Temporal Modeling

One of the limitations of prior work on app prediction is that it does not consider *freshness*, i.e. how recently an application's content was prefetched prior to an application use. Consider the following naïve prefetch example — every time *CQue* predicts that Email will be used as the next app, prefetch Email. This has high freshness if Email is used immediately but has poor freshness if the user opens Email an hour later. An alternative might be to prefetch Email every, say 10 minutes, until the user opens Email. But this suffers from high network access overhead if the delay is of the order of an hour or more. Clearly, if we want to optimize cost, we need to first predict *when* an app is likely to be opened next.

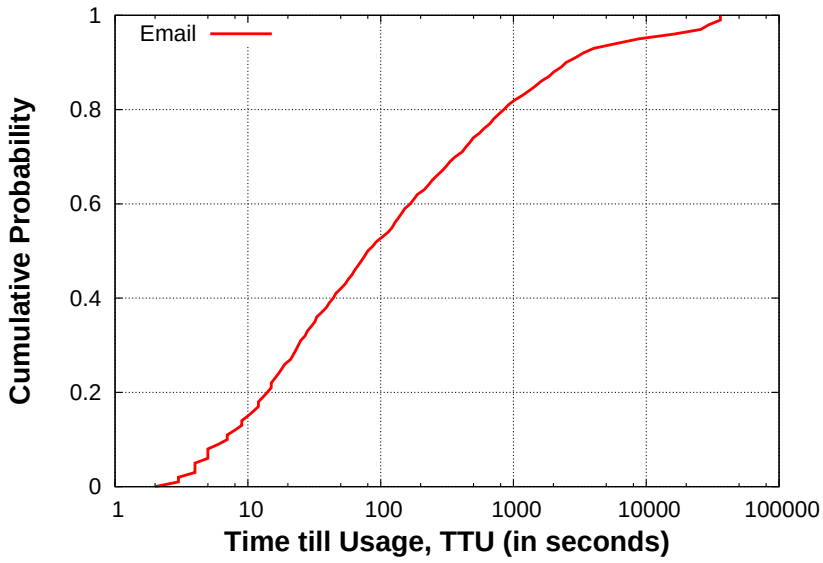


Figure 5.2. The cumulative probability distribution of *Email* to be launched as the next app as a function of time till usage, obtained for a representative user.

The question we are addressing here is: If we have just opened an app, and if we know that the next app to be used is e, what is the probability that e will be used in time interval Δt from the current time (denoted $p(TTU \leq \Delta t | nextapp = e)$, where TTU, time till usage for $nextapp$, is the time that elapses between the current time and when the $nextapp$ is used). Note that we only need to evaluate this probability every time an app is used (we refer to this as an *app-change event*), since no additional information is available between app usage events.

To answer this, we learn the conditional cumulative distribution function (CDF) $F_{TTU|nextapp=e}$ for an app e from the app usage history. Given an app usage history $[app_1, ..., app_n]$ where app_j is the j^{th} app used and launched at time t_j, we can learn the conditional CDF using the distribution of all durations $(t_j - t_{j-1})$ such that $app_j = e$. Figure 5.2 shows an example of the conditional CDF obtained for a representative user where $nextapp$ being modeled is *Email*. The conditional CDF is computed from a user trace of app usages obtained from LiveLab iPhone usage dataset [163].

5.4.3 Decision Engine

Our decision engine executes on each app-change event and utilizes the prediction model and the temporal model to decide when to prefetch an app such that it balances the need to improve freshness, while at the same time limits the network bandwidth costs associated with frequently connecting to a server and downloading the data from it. The output of the decision engine is the time Δt we should wait from the app-change event to execute prefetch.

Recall that our prediction model predicts *what* app is going to be used next and the temporal model predicts *when* some app will be used assuming it knows what app will be used next. Our decision fuses the uncertainty in *what* and uncertainty in *conditional when* to get the joint estimate of the probability of some target app $nextapp$ to be used next within some time interval(Δt). Let us denote this probability using p_{fetch}. Now, if we choose to prefetch after time Δt has elapsed, then with probability $1 - p_{fetch}$, the next app will be used after prefetch and will see a freshness benefit of Δt, and with probability p_{fetch}, the next app is opened prior to prefetch and it will see no benefits in freshness. If we select a small value for p_{fetch}, it has two effects: a) more apps are prefetched since even apps that are predicted to be opened next with low probability will qualify for prefetch, and b) Δt will be smaller and hence, we will typically prefetch apps quickly. This means higher bandwidth cost and apps used a while later see less prefetch benefit. On the other hand, a higher value of p_{fetch} means fewer app prefetches and longer Δt and hence, we prefetch less frequently but the fewer target app usages that benefit from it see better freshness due to larger Δt benefit.

Thus, the key question for the decision engine is how to choose p_{fetch} to balance freshness and bandwidth cost. To address this, we set a target bandwidth cost for each app, and learn from history the largest possible value of p_{fetch} that would have been within the target cost while ensuring the least missed opportunities. This learning is not expensive since we

only need to make a decision upon each app change event, which restricts our search space. We now present our intuition in a more formal manner.

Algorithm

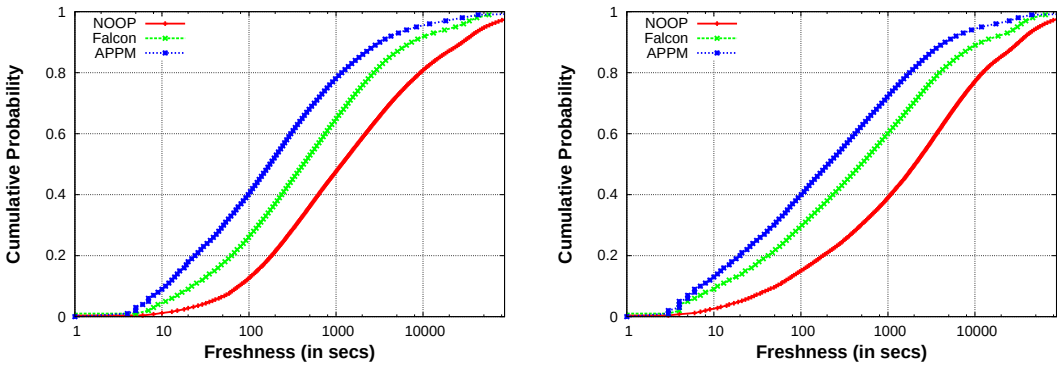
Our prediction model can be used to compute the probability $p(nextapp = e)$ and our temporal model gives us the probability of $nextapp = e$ being used within time Δt conditioned on e being the next app. This probability is provided using the conditional cumulative distribution function $F_{TTU|nextapp=e}(\Delta t)$. The decision engine fuses these two outputs multiplicatively to obtain a cumulative distribution function ($F_{TTU}(\Delta t)$) that can be used to compute the probability of e being used as the next app within the time interval Δt.

$$F_{TTU}(\Delta t) = p(nextapp = e) \times F_{TTU|nextapp=e}(\Delta t)$$

Once the distribution function F_{TTU} is known, we need to choose a threshold p_{fetch} that balances freshness and cost as described earlier. We define *network bandwidth cost* (C) as the multiplicative factor of the rate of use of the target app. Thus, if a target app is used N times then the number of prefetches can be at most $N \times C$. Given such a cost C and app usage history, we can estimate the appropriate value for p_{fetch} to limit the cost to C. The detailed description of our algorithm can be found in Algorithm 2, but intuitively, if there are N occurrences of $nextapp$ in the user's history, the multiplicative cost is C and we know the probability values attained by the joint distribution function F_{TTU} just before each app-change event in the history, we pick the NC^{th} largest probability value as p_{fetch} since it results in exactly NC prefetches on the historical trace.

Algorithm 2 Compute Time To Prefetch

1: **Input:** Network Bandwidth Cost C; TTU distribution function for target app F_{TTU}; TTU probability history $d[1...L]$; Count of target app in user's history N.
2: **Output:** Time to wait for prefetch Δt.
3: Sort d in decreasing order.
4: $p = d[N_e * C]$ i.e. NC^{th} highest TTU probability.
5: $\Delta t = F^{-1}_{TTU}(p)$.
6: **return** Δt



(a) Freshness in *Facebook* app

(b) Freshness in *Email* app

Figure 5.3. Freshness observed using i) No prefetching (NOOP) ii) Previously proposed scheme (Falcon), and iii) our APPM prefetch algorithm. While the previous scheme improves median *Email* and *Facebook* freshness to 7 and 8 minutes, respectively, APPM further improves freshness to 3 minutes at the same bandwidth cost.

5.5 Trace-driven Evaluation

In this section, we evaluate the benefits of our techniques using long-term app usage traces provided by the LiveLab project at Rice University [163]. This dataset consists of traces for thirty four volunteers collected on iPhone 3GS for a period of up to fourteen months. We look at the app-usage traces from this dataset, which provide the start time and the duration of each app usage observed on the phone. In addition, we use location traces to obtain the location where apps are used. Since location traces are obtained at a coarse granularity compared to app usage data, we interpolate location information to find the missing location information for app usage data. In all, we have 576,491 records of app use across 14 months.

Prefetch Effectiveness We evaluate the effectiveness of our prefetch algorithm and compare it with Falcon [186] which is the only other known scheme for app prefetch. Figure 5.3 shows the results for two heavily used apps that can leverage prefetch, *Email* and *Facebook*. Since Falcon cannot control the bandwidth costs, for a fair comparison, we evaluate our APPM+TTU model based prefetch scheme using the same bandwidth cost as observed in Falcon. The median freshness (i.e. the time period between the latest network data fetch

Table 5.2. *PREPP* w/ TTU model delivers better freshness, especially for low bandwidth budgets. Three quartiles of observed freshness values for *Facebook* app are shown at varying bandwidth budgets of 2x, 4x and 8x (Evaluated on 34 users from LiveLab dataset).

Cost	**Algorithm**	Freshness(in secs)		
		25-%ile	50-%ile	75-%ile
2x	APPM w/o TTU model	83	383	1966
	APPM w/ TTU model	68	286	1334
4x	APPM w/o TTU model	41	190	928
	APPM w/ TTU model	37	155	691
8x	APPM w/o TTU model	26	111	506
	APPM w/ TTU model	26	104	452

and the time of actual app use) with the Falcon scheme is near 8 minutes for both the apps. Our scheme improves freshness by reducing median freshness to 3 minutes. Note that for the 'No prefetching' case, we computed freshness as the time elapsed since last use of the app assuming that the app is refreshed by a user upon each use.

In addition, we evaluate the effectiveness of temporal modeling in our scheme by comparing it against a APPM-based scheme which prefetches on app-change events whenever the probability of the target app predicted by APPM algorithm is above a certain threshold. The threshold value in this scheme is determined by the bandwidth cost. Table 5.2 shows the benefits of using TTU model in 3 quartiles of observed freshness in *Facebook* app for various costs. We see that the benefits are significant, particularly at low bandwidth costs. The median freshness improves by 25% for $2\times$ and $4\times$ cost, demonstrating the benefit of incorporating TTU into the algorithm.

Table 5.3. Aggregate prediction accuracy over all users obtained using APPM with and without various contexts. Additional contexts like Time of Day and Location provide only a marginal improvement in accuracy.

Context Used	**Prediction Accuracy**
None	80.85%
Location	81.10%
Time of Day	81.23%
Location, Time of Day	81.35%

Utility of location context As described earlier, several prior efforts at predicting application usage have pointed out that location context is very useful [164, 178, 186]. Yet, obtaining location is often undesirable due to user privacy concerns as well as energy overhead. Therefore, we investigate the actual utility of location data and whether we can forgo it. Interestingly, we find that app prefixes largely subsume the need for explicit location.

To understand whether location contexts improve prediction accuracy, we design a multi-context version of APPM which includes a different predictor for each location context (e.g. home, workplace, etc), and one for each time-segment of the day (e.g. morning, afternoon, etc). Given several context-specific predictors, we compute a weighted combination of these models to predict the next application. Table 5.3 shows the performance of a single predictor that does not use any context, as well as predictors that use additional contexts. The benefits of additional context is surprisingly small, a puzzling observation given observations made in prior work. A closer look at app usage patterns in different location contexts provides an interesting explanation. As it turns out, "contextual" information is partially captured by the app sequences that are used only in a specific context e.g. Angry Birds at home. We further validated this by looking at the distribution of location contexts for each prefix used by APPM for a representative user who had 14 semantic locations (Figure 5.4). We observed an 80^{th}-percentile entropy of 2.0 bits indicating that a large number of app prefixes have location information encoded in them. If the prefixes had no correlation with location, then the entropy would be close to 3.8 bits corresponding to the 14 semantic locations.

Table 5.4. Comparison of APPM with previous app prediction algorithms when making Top-5 predictions. APPM performs better than schemes using location and time context.

(a) Predicting all apps in the user traces

Algorithm	Prediction Accuracy
MFU	48.81±1.08 %
2-NB	74.87±1.60 %
3-NB	78.81±1.34 %
APPM	80.85±1.23 %

(b) Predicting follower apps in the user traces

Algorithm	Prediction Accuracy
Falcon	70.16±1.56 %
APPM	74.37±1.41 %

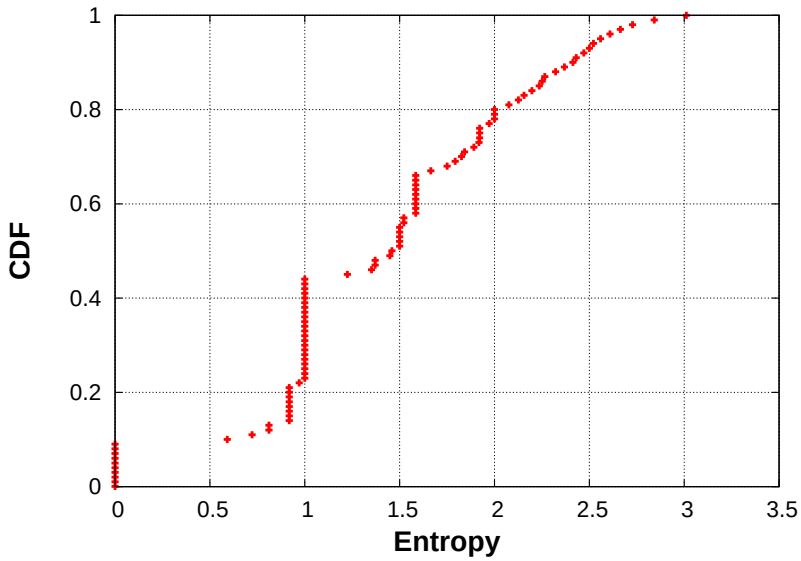


Figure 5.4. CDF of entropies observed in locations for each app prefix for a representative user. This user had 14 semantic locations that should result in a maximum entropy of $\log_2 14 = 3.807$. The 80%-percentile entropy of 2.0 indicates that large number of app prefixes have location information encoded in it.

Prediction Evaluation Tables 5.4(a) and 5.4(b) present a comparison of prefix-only *CQue* against previously proposed prediction techniques including: a) Falcon, which uses location and time contexts [186], b) 2-NB: a Naive-Bayes prediction model that uses location and time-of-day as features [178], c) 3-NB: a Naive-Bayes prediction model that uses location, time-of-day and previous-app-used as features [27], and d) a strawman *Most Frequently Used* (MFU) algorithm.

Table 5.4(a) compares prefix-only APPM against all schemes except Falcon, and Table 5.4(b) compares against Falcon. We separated these results because Falcon only predicts occurrences of apps that follow the first app used upon unlocking the screen. The results show that the prefix-only version of APPM performs just as well as algorithms that use additional contexts including location.

5.6 Practical Prefetch Considerations

Mobile operating systems such as Android, iOS and Windows Phone support a set of runtime semantics distinct from their desktop OS counterparts. A central challenge in the design of a practical prefetching system is managing the limitations posed by these mobile operating systems. While the remainder of our discussion centers around Android, the design decisions are in fact standard to modern mobile OSs, and apply to iOS and Windows Phone as well.

Addressing Android constraints: Android places several constraints on the scheduling and resource usage of foreground and background threads and processes. The constraints that directly impact the design of prefetch mechanisms are two-fold. First, main threads are not scheduled when the device enters the standby state (which occurs either due to an idle activity timeout or an explicit user button toggle). This means that main threads can only run when the device is active. Second, apps are restricted from performing networking activities on the main thread as it can make an app unresponsive if there is slow or no network connectivity. Thus, the standard model is that apps fetch network data by spawning background threads from the main thread.

We examined the behavior of several apps that could be potential prefetch candidates and found that all of them rely on the main thread to be active to fetch content or load state. For example, *Facebook* and *Email* rely on network fetch and initiate background threads for network calls from the main thread when the app is opened and appears on the screen. Thus, it was clear that we could not initiate prefetch during times that the device was turned off, leaving us with the times that the device was actually unlocked and in use.

A closer look at applications that require content refresh reveals that they often auto-refresh upon opening the app and/or upon re-starting the app. This means that once auto-refresh is triggered by the main thread, the asynchronous network fetch task is created and the app can be pushed to the background. Thus, we only needed to bring the app to the foreground for the small amount of time that it needed to initiate auto-refresh. An empirical

study of the minimum amount of time needed for an app to be brought to the foreground showed that it ranges from 200ms for apps like *Email* to up to 1 second for *Facebook*. Note that even if auto-sync is turned off for an application, it is possible to kill and re-start an app, at which time it automatically updates its content.

Minimizing disruptiveness to user: Next, we ask what is the most suitable time for prefetch such that it is minimally disruptive to user experience. During periods when the device is actively used, three opportunities present themselves: a) when the screen is unlocked and before an application is started, b) transition times between usage of applications in the same session, and c) between an app being closed and the phone being turned off. While all three are viable, we find that the most convenient option is prefetch upon screen unlock since it allows us to prefetch immediately prior to a user opening an app. Our approach uses k-step prediction, where APPM predicts which apps to prefetch among the next k applications that were going to be used. Previous work has analyzed typical session lengths and found that these are typically short [186], therefore a two-step prediction is sufficient for deciding which apps to prefetch upon screen unlock.

Parallel prefetch to minimize energy overhead: The energy consumption in executing prefetch is another consideration that impacts our design. Prior work has shown that data transfer using cellular incurs a tail-energy overhead as the phone's radio remains in active mode for another 10-20 seconds after data transfer is complete [51]. Therefore, batching transfer can provide energy savings in the range 62-75% for 3G networks. We leverage this insight to reduce prefetch cost by executing a number of prefetches in parallel. In *PREPP*, two-step look ahead allows us to merge all the prefetches that are scheduled to execute within 30 seconds of each other and execute them in parallel.

5.7 Practical UI Considerations

Users not only want the right app to start quickly, they also want to find the right app quickly from amongst the many they have installed. In response, we employ APPM predic-

tions for a second purpose: deciding which app shortcuts to display to users on an *adaptive shortcut menu*.

The idea of adaptive shortcut menus for smartphones has existed for some time [26, 30, 179]. Our approach is a synthesize of the best practices from this line of work, in light of the practical UI constraints imposed by a representative commodity OS, Android. In addition, our use of APPM with its ability to operate without training means that our adaptive shortcut menu adapts much more quickly than previous menus. The *PREPP* adaptive shortcut menu design addresses the following user-facing questions: when, where and what to display.

Where: While it is possible to supplant the Android shell with a non-standard shell, such a change would significantly limit potential users because users seldom install new shells. Moreover, recent studies have found that too much home screen dynamicism leaves users feeling confused and out of control [164]. Instead, users prefer split menus [159] that augment rather than replace their existing distinct icon arrangements [28]. Therefore, our adaptive shortcut menu is presented as an Android widget which can be added to an existing home screen palette.

When: Just like apps, a widget's main thread is only scheduled when they are in the foreground. Therefore, a widget's opportunities to execute are the same as mentioned for apps in Section 5.6. Since APPM predictions can be made very efficiently (as shown in Section 5.9), we opt to have the adaptive shortcut menu update upon unlock and between app transitions. As a result, users always see a very adaptive prediction of upcoming apps.

What: Figure 5.5 shows a screenshot of our *PREPP* adaptive shortcut menu Android prototype. Shortcut icons for the top five predicted apps are shown, and the user can click on any of them to jump directly to it with immediate prelaunch speedup. In addition, a timer beneath each shortcut indicating time since prefetch informs users about content freshness.

5.8 Implementation

To implement the APPM algorithm on Android, we ported the PPMII implementation from D. Shkarin et al. [165] in C++, and modified it to implement our ranking scheme for predictions. Since Android apps are written in Java, we implemented a Java wrapper around this code using Java Native Interface so that other components implemented in Java can interact seamlessly with the predictor.

In addition to prefetching content automatically, we also integrate a dynamic home screen widget that leverages predictions from APPM to allow users to quickly find apps among the many that they have installed. The idea of adaptive shortcut menus for smartphones has existed for some time [26, 30, 164, 179]. Our *PREPP* widget builds on prior work, with the change that we update the widget not only when the device is unlocked, but also when the user returns to his homescreen after closing an app. This provides the user with a higher degree of adaptiveness.

Figure 5.5 shows a screenshot of our *PREPP* adaptive shortcut menu Android prototype. Shortcut icons for the top five predicted apps are shown, and the user can click on any of them to launch it directly. In addition, a timer beneath each shortcut indicating time since prefetch informs users about content freshness.

User Control: Since a user may have sensitive apps like banking applications or apps that use privacy-sensitive information like location upon start, we do not prefetch any app by default. Instead, we provide users with an interface to select the apps they want to prefetch. This interface provides users with a greater control regarding which apps get prefetched.

5.9 Experimental Evaluation

In this section, we present an evaluation of our implementation through two user studies. We first describe our controlled user study where we evaluate benefits of prefetch in detail. Next, we describe our experiences deploying our prediction algorithm part of

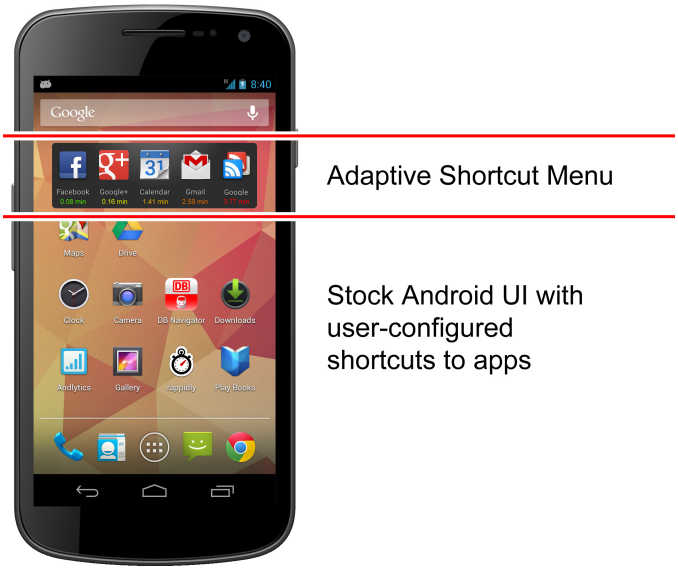


Figure 5.5. *PREPP's* adaptive shortcut menu placed on the homescreen as a widget.

PREPP to Google Play Store and study its impact on real users in the wild. Finally, we conclude with benchmarks characterizing system overhead for our implementation.

5.9.1 Controlled User Study

In order to perform a detailed evaluation of the prefetch benefits from *PREPP*, we conducted a user study[2] consisting of 22 users. Of these 22 users, 6 users used *PREPP* for at least two weeks, segmented as follows: (1) one week of control data with no prefetch, and (2) one week of data with prefetch active, and the multiplicative bandwidth cost C set to 2. The remaining 16 users had a control and test period of 3 days and 4 days respectively. Recall that *PREPP* performs prefetch by bringing an app to the foreground for a brief period upon unlocking the phone screen. To make users unaware of whether it was the control or experimental period, apps were brought to the foreground during the control period without actually performing any content fetch activity. Note that for the controlled user study, we disabled the adaptive shortcut menu shown in Figure 5.5.

[2]Study conducted with appropriate IRB approval

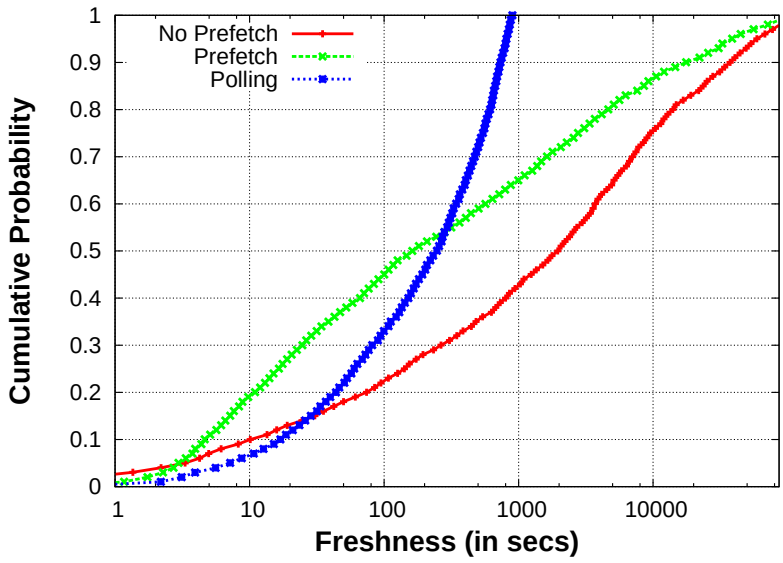


Figure 5.6. Prefetch achieves better content freshness than 15-minutes polling at a fraction of cost (number of polls were $3.58\times$ number of prefetches). The median freshness improves from 1955s (no prefetch) to 247s with polling and to 162s with prefetch.

The primary metric for evaluating the benefit of prefetch during app usage is the increase in freshness of content. Content freshness reflects how recently content was retrieved at time of use, and is measured as the time between prefetch and app usage.

Figure 5.6 shows distributions of freshness observed for all users during the control period with no prefetching and the test period with active prefetching. Also, we show the distribution that can be achieved via 15-minutes polling for the test period as polling is the most popular way of refreshing app content on phones. We see substantial freshness benefits across the board, with an order of magnitude improvements across all three quartiles. The median freshness improves from 32.6 minutes to 4.1 minutes with polling and to 2.7 minutes with prefetch. Most importantly, the number of polls during test period are $3.58\times$ the number of prefetches. Thus, APPM achieves better freshness than polling at a much smaller cost. Also, the median freshness with prefetch for 16 users with 3-days control period and 6 users with 7-days control period are 3.6 minutes and 2.06 minutes respectively.

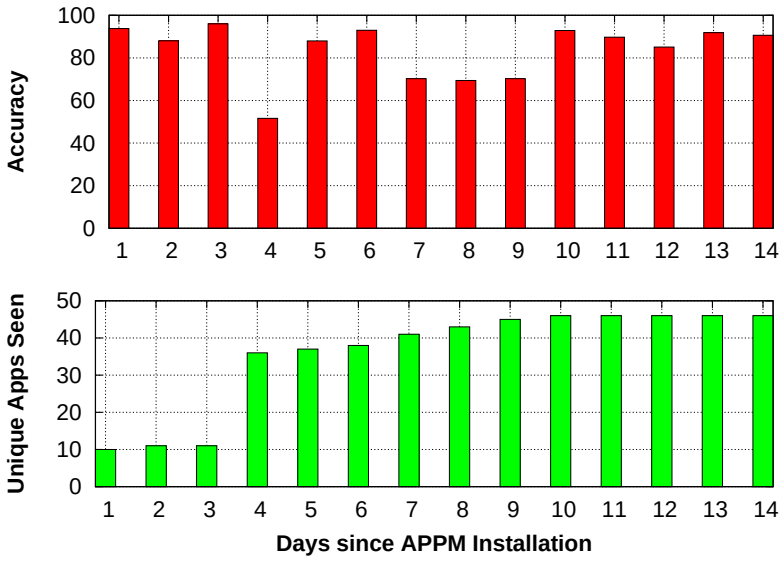


Figure 5.7. APPM yields high accuracy and adapts quickly to usage of new apps.

Additionally, we measure prefetch effectiveness using *precision* i.e. the fraction of prefetches that were followed by a respective app usage and *recall* i.e. the fraction of app usages that were preceded by a respective prefetch. The average precision and recall values observed for all the users are 22.12% and 46.87% respectively. In contrast, 15-minutes polling has 3.82% precision and 28.97% recall.

Prediction Accuracy and Adaptivity

We now show that APPM requires very little training and adapts rapidly to changing user behavior. Figure 5.7 shows the time-series trend for a representative user — the bar graph below shows the number of unique apps seen by APPM over time, and the upper bar graph shows APPM's accuracy on those days based on top-5 predictions.

The results suggest two interesting conclusions. First, we see that accuracy with top-5 predictions is extremely high (95%) even on the first day, and our average prediction accuracy is 81.89$\pm$3.9%, both of which are very high. Second, we see that APPM adapts well to usage dynamics — for example, we see that on day 4, the user invokes a large number of new apps that are previously unseen by APPM, As a result, prediction accuracy

drops for the day, but rapidly improves again on day 5. We see similar fluctuations between day 7 to day 10 where new apps are used. These results validate the adaptive capability of APPM

5.9.2 User Study in the Wild

Next, we study *PREPP* in the wild, and specifically focus on the prediction performance. We packaged the adaptive shortcut menu widget with APPM prediction and other state of the art prediction algorithms and released them as a standalone download — called *AppKicker*[3] — on the Google Play Store.

Methodology

We compared APPM with two other prediction algorithms: i) Most Recently Used (MRU), which is used by all major mobile platforms to show MRU app shortcuts; and ii) Sequential (SEQ), which uses only the previously used app to predict the next app, as suggested by previous work [27, 164].

We used a within-subject A/B/C design to test and compare the different prediction strategies; whenever a new prediction is requested, we randomly choose one of APPM, MRU or SEQ for prediction. We tracked how users interacted with the icon menu with each prediction algorithm. However, our evaluation is limited in that we cannot know what other icons users have on their home screens. For example, we might recommend apps that the user has already pinned to his home screen, or we might not catch app launches that the user initiates outside of our launcher widget.

[3] http://goo.gl/wZO0C

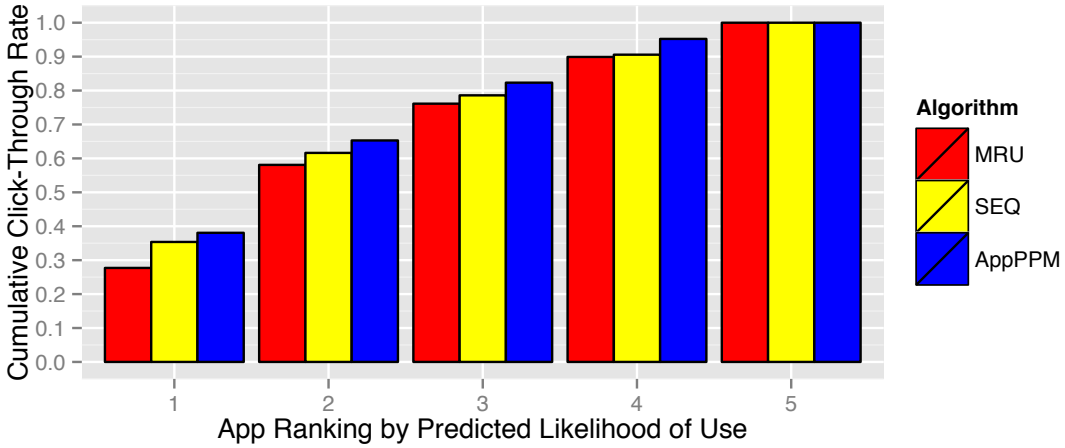


Figure 5.8. APPM yields better click-through rates for more highly ranked apps.

Results

At the current time, the widget has been installed more than 50,000 times and has more than 7,630 active users.[4] Note that both *PREPP* installation and participation in the research study are voluntary. As soon as a user starts *PREPP*, we present a research study disclaimer and offer opt-in consent. Following the Two Buttons Approach [141], users can decline from contributing data but still use the application. For a fair comparison of the different prediction algorithms, we report results for the top 100 users as ranked by click volume. These users had a median of 112.5 clicks (min 62, max 1,807). This resulted in 16,991 clicks in total.

Figure 5.8 shows the click-through rate as a function of the prediction algorithm's ranking. APPM yields a click-through rate of 38.1% on its top ranked app, whereas MRU and SEQ produce only 27.7% and 35.4% respectively. Similarly, APPM outperforms other prediction algorithms at other ranked positions as well. APPM's more accurate predictions and better app rankings result in more opportunities for effective prefetch.

[4] According to Android Developer Console

Table 5.5. System overhead

Binary Size	0.96MB
Memory	6.5MB
Time for prediction	$<250\mu s$
Time for prefetch decision	<5ms

5.10 Microbenchmarks

System Overhead: We measure the system overhead of our implementation on the Samsung Galaxy Nexus Phone. Table 5.5 shows the overhead summary of *PREPP*. Using APPM for prediction and prefetch decision requires less than $250\mu s$ and $5ms$ respectively. This overhead is sufficiently low that a prefetch upon unlock is not an issue. The memory requirement of 6.5MB is modest and remains stable during execution. This suffices for running all required background services, keeping statistics for prediction in memory, and keeping uplink state to collect user trace.

Data Overhead: In this evaluation, we define two categories of apps that can use prefetching and evaluate the data overhead. We use data overhead observed for each app to set the app-specific multiplicative bandwidth cost C to be used by the decision engine.

Persistent data apps

In this category of apps, when refreshed, the apps fetch all the new-arrived data since it was last updated and save the data locally for later retrieval. Example of such apps include Email, Dropbox and Evernote. The data usage in these apps is proportional to the size of payload being fetched. These apps can fetch their payload in one or more prefetches but the actual cost of payload fetch remains independent of the number of prefetches. However, each prefetch incurs a constant overhead that is needed for the synchronization protocol and makes multiple prefetches expensive to execute. We can compute this constant overhead by observing the total data usage as we increase the payload size linearly. By fitting straight

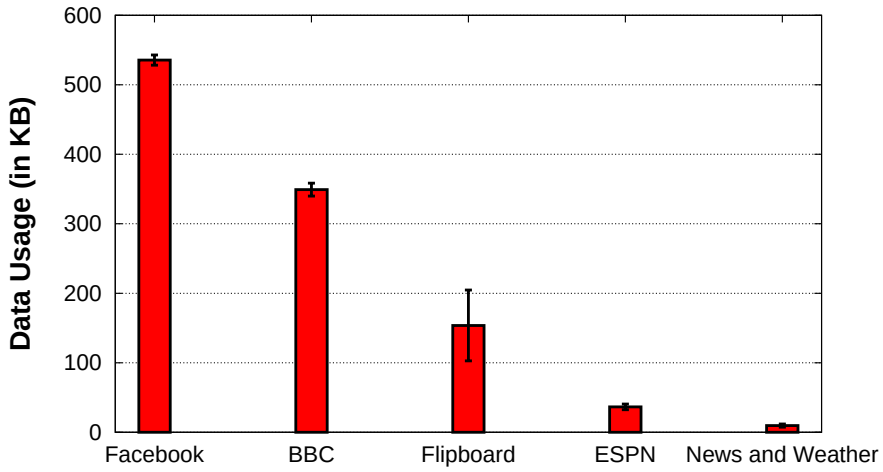


Figure 5.9. Data usage per refresh for popular apps.

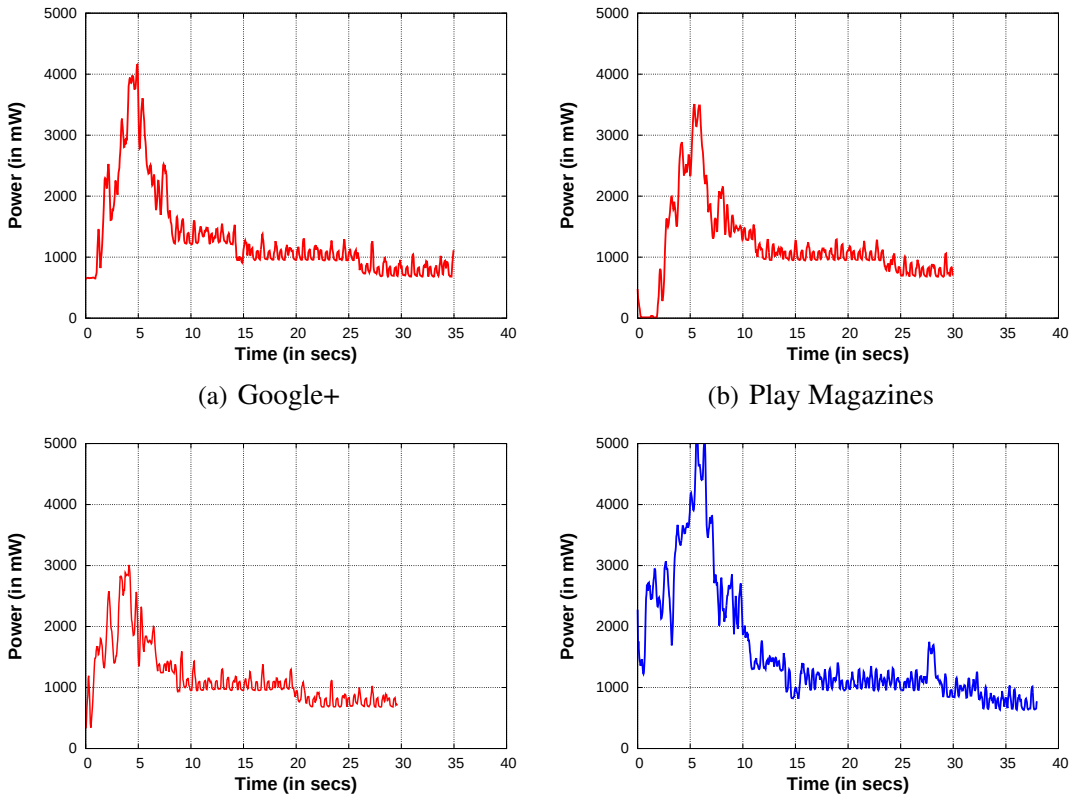
lines to these observations, we can solve for the constant. Using this, we estimate constant overhead to be 8447 bytes for the Email app and 2659 bytes for Gmail app.

Live feed apps

This category of apps receives the latest feed from the server. In contrast to Email app, these apps do not fetch all the new feeds since the last update but only the most recent ones. Example of these apps include Facebook, News and Weather, BBC news, etc. Depending on the nature of feed, the size of the feed can be near constant (e.g. live game scores) or it can vary significantly (e.g. Facebook where the number of images and the amount of text in a feed vary). While most of these apps will discard the previous feed upon refresh and fetch the feed again, there are a few apps like Flipboard that fetch only the change in feed since the last update. Overall, the data overhead in these apps is equal to the data used in receiving the new feed. Figure 5.9 shows average data usage for popular apps observed at various times of day.

Energy Consumption in 3G: We now compare energy consumption for refreshing app content in i) prefetch scheme, and ii) manual refresh. We note that a user refreshes app by opening each app in a sequence. In contrast, our prefetch scheme identifies all the apps

that need to be refreshed as soon as the phone is unlocked and starts fetching content for all the identified apps in parallel. Consequentially, the power consumption characteristics are different for prefetch and manual refresh. Now, we show that our prefetch scheme using parallel fetches reduces the energy overhead per app-prefetch, and can save energy up to 47% in comparison to manual refresh if the app predictions are accurate.



(a) Google+ (b) Play Magazines

(c) News & Weather (d) All apps in parallel

Figure 5.10. Power consumption characteristics for fetching network data shown for 3 apps: *Google+, Play Magazines* and *News & Weather*. Google+ has highest data usage(>300KB) whereas News has the least data usage(<10KB). Figure 5.10(d) gives characteristics when all these 3 apps are fetched in parallel.

Figures 5.10(a)-5.10(c) show power consumption characteristics of network data-fetch for 3 distinct apps on a 3G network with different levels of data usage. Also, Figure 5.10(d) shows power characteristics when data is fetched for all these 3 apps in parallel. We measured this using Monsoon Power monitor on Galaxy Nexus phones running on AT&T's 3G network. In these figures, the first phase with high power consumption is the data transfer

Table 5.6. Energy consumed in parallel fetch vs sequential fetch of content for 3 distinct apps. Parallel fetch saves 39.34% and 47.14% energy in data transfer phase and total respectively.

	Energy Consumption(in μAh)	
	Data Transfer Phase	**Total**
Sequential	2320.04	3547.25
Parallel	1407.31	1875.00

phase, followed by a period called the *tail* up to 17s in length when there is no data transfer but the power consumed remains above 1000mW. In the end, we see a baseline power of 783mW required to keep the screen and cpu running on the phone. It is easy to see that parallel fetch has an advantage over sequential fetch as it incurs an energy overhead of 1 tail instead of 3 tails in the worst case for a sequential fetch. Apart from energy consumed in the tail, the energy consumed during the data transfer phase in parallel fetch is smaller than the total energy consumed in all the data transfer phases of sequential fetch of all the apps. Table 5.6 shows the average energy consumed in parallel and sequential fetch obtained over several runs. In this computation, we subtract the baseline power to accurately estimate the energy consumption for network fetch. We see that the parallel fetch uses 47.14% less energy than sequential fetch. Thus, if apps are predicted accurately, our prefetch scheme can potentially save energy. But if the predictions are inaccurate, energy in prefetch can go waste. In the worst case, the energy consumption of $1875\mu Ah$ for prefetching 3 apps in parallel accounts for 0.13% of the $1400mAh$ battery available on the Galaxy Nexus phone. This overhead is further reduced if upon unlocking the phone, a user starts some app that connects to the network to fetch data or an advertisement or if some widget initiates a network connection. This is because a prefetch will occur in parallel with a user-initiated or a widget-initiated fetch.

5.11 Related Work

PREPP relates to three distinct lines of prior work:

App Prediction: In the wake of increasing availability of apps in phone's app stores, a number of algorithms have been proposed for utilizing contextual information in app usage prediction. Verkasalo et al. [178] found that location and time of day are the most useful contexts in app usage prediction. The study by Böhmer et al. [27] showed that in addition to location and time contexts, the last app used by a user is a useful predictor for the next app. Yan et al. [186] proposed Falcon that uses the first app opened upon phone unlocking to predict the following apps. Most recently, Shin et al. [164] analyzed a variety of contextual information like accelerometer, WiFi, SMS, GPS, last used app, etc. to build app-specific naive-bayes predictors by selecting the best set of contextual features for each app. In contrast to all these works, our APPM algorithm does not use any of power-hungry or privacy-sensitive contexts but utilizes only the history of sequence of app usages in prediction, and does not require a long training period unlike previous efforts.

Mobile Web Content Prefetch: Prefetching in mobile phones has recently received attention from the research community [19, 77, 186]. Higgins et al. [77] argue for an OS-supported API for prefetch where third party apps are expected to provide hints to execute prefetch. Such a model for prefetch requires modification of third party apps to learn the temporal model. In contrast, *PREPP* does not require any modification of the OS or third party apps. Yet, *PREPP* is complementary to this work in that APPM can provide hints to a potential OS-supported API on behalf of legacy third party apps. FALCON [186] considers prefetch under a fixed energy budget, but requires extensive modification of the OS and some apps, requires long training periods, and does not offer as accurate prediction as *PREPP*. Balasubramania et al. [19] proposed the TailEnder protocol that minimizes energy usage while prefetching and meeting user-specified delay tolerance. However, they do not take usage behavior into account, performing prefetch even when a user is unlikely to use apps. In the non-mobile case, browsers have utilized content prefetching and caching for reducing page load times [10], but were not targeted for interoperatibility with mobile apps nor OSs.

Adaptive Shortcut Menus: Bridle and McCreath [30] investigated injecting shortcuts such as calling a specific person into users' smartphone menus. Shortcuts included common tasks such as calling a specific person or sending a specific text message. They found that a dual approach presenting frequently used communication options mixed with Naïve Bayes-filtered options worked best. Vetek et al. [179] extended this approach with automatically generated context-aware menus based on unsupervised learning. St. Amant et al. [170] investigate the optimization of hierarchical text-based menus for cell phones that can be traversed by keyboards input. Based on their findings they propose options for optimizing menu structures that result in reducing traversal time, e.g. by putting commonly used items higher in the hierarchy. Their approach for menu redesign results in a time saving of 30% in simulation studies. Also Matsui and Yamanda [113] present an algorithm for optimizing the menu structure for hierarchical menus on mobile devices. In their experiment they minimize menu item selection time by changing menu structures. Böehmer et al. investigated the impact of context on users' icon arrangements [26], and found that when given a chance to arrange their icons according to context, users place icons associated with a context in prominent positions. While informative as a set of guidelines, none of the preceding work takes into account the design considerations for adaptive shortcut menus presented in Section 5.7, and hence are not complete solutions to the task of reducing visual app search time.

5.12 Conclusion

Mobile apps now serve a dazzling array of functions, but are becoming increasingly complex and reliant on network connectivity to provide up-to-date content and rich interaction. For mobile users where seconds of sluggish loading can dissuade many users, *PREPP* mitigates long network content retrieval times by accurately predicting which apps will be used, and prefetching their app content to improve the user experience. Moreover, *PREPP's* careful design, cognizant of the constraints posed by commodity OSs, has al-

lowed us to implement, evaluate and deploy a *PREPP* prototype on commodity Android devices. Through a combination of a controlled user study and a study "in the wild" of the prediction algorithm, we show that we can accurately predict app usage, and deliver system-level speedups even without modification to current OSs and apps.

CHAPTER 6

SUMMARY AND FUTURE WORK

6.1 Thesis Summary

This thesis has explored the problems and challenges in three aspects of behavior-aware applications: behavior detection, behavior understanding and behavior prediction. This thesis presented the design and implementation of systems and techniques for addressing challenges in these three aspects of behavior-aware applications. These systems and techniques have been developed to work on off-the-shelf smartphones. We have evaluated our systems and techniques by using data collected from real users and also by conducting real user studies.

- **RisQ: Gesture Recognition System** In Chapter 3, we described the use of mobile devices as a platform to detect health behavior contexts namely, smoking a cigarette. Our mobile application called *RisQ*, uses a wristband equipped with inertial sensors to accurately count the number of cigarette puffs taken by a user and to identify the smoking session boundaries reliably. Our experimental evaluation done using data collected from users showed that we can detect natural gestures like eating and smoking with high accuracy and precision. The user studies conducted to evaluate the *RisQ* data processing pipeline showed that *RisQ* can detect smoking gestures in real-time with high precision and recall and can be used for real-time interventions. This work has appeared in the Proceeding of MobiSys 2014 [133].

- **CQue: Context Query Engine** In Chapter 4, we presented a *CQue*, a context-query engine that improves the accuracy and energy consumption characteristics of context

inference algorithms through information fusion and reduces the privacy risks associated with context-aware mobile applications. *CQue* supports continuous context-sensing on a limited energy budget enabling behavior analysis over a stream of user contexts data. *CQue* provides a query interface to the mobile applications, enabling applications to obtain accurate context results while remaining agnostic of "what" classifiers and sensors to execute and "when" to execute. Our experimental evaluation using real user traces confirms the benefits of *CQue* in context-sensing. This work has appeared in the Proceedings of MobiSys 2013 conference [134].

- **PREPP: Predictive Prefetch System** In Chapter 5, we discussed the design and implementation of a practical prefetch system called *PREPP* that improves the responsiveness of the mobile applications by keeping the application data fresh through network data fetch at appropriate times determined by the app usage behavior. Our system utilizes Markov-model based prediction algorithm to predict app usage behavior and a temporal model to prefetch the data for the predicted apps in a cost-aware manner such that the network costs in keeping app content fresh is bounded. Our evaluation using user studies showed the effectiveness of *PREPP*'s modeling techniques in achieving high freshness for mobile apps with a low data and energy overhead. This work has appeared in the Proceedings of UbiComp 2013 [132].

6.2 Future Work

While this thesis presented novel techniques for building energy-efficient, accurate and responsive behavior-aware mobile systems, there are new research directions that opened up from the work described in this dissertation. In this section, we identify and describe a few feasible directions of future research in context-aware mobile computing.

- **Natural Gesture Recognition** Our evaluation of *RisQ* data processing pipeline showed promising results for detecting gestures like smoking and eating. An interesting ex-

tension of this approach is to detect a broad range of natural hand gestures such as various types of exercises performed in a gymnasium, drinking, typing on a computer, etc. We anticipate that this extension will require a new segmentation approach to identify segments containing a gesture, and also require use of other sensing modalities like accelerometer to extract features relevant to different types of gestures. Another interesting direction for this work is to use more than one inertial measurements unit (IMU) on the human body to detect activities based on body postures. Data fusion from more than one IMUs can add capabilities to discriminate between various gestures and activities that look similar under features derived from a single IMU.

- **Context Query Engine** Our work on *CQue*, context-query engine opens up several avenues of future research. First, our current model of Dynamic Bayesian Network (DBN) can be extended to include low-level signal features extracted from the sensor data for information fusion in order to further improve the accuracy of context-inference. Second, an interesting extension of this work involves incorporating sensors embedded in the external environment. Being able to use sensors available from the public infrastructure can reduce the energy requirements in sensing on the mobile device. Another interesting research question to explore is how can we use a Dynamic Bayesian Network model to decide when to duty-cycle classifiers. Intuitively, the state of DBN can tell us which context is unlikely to change in the near future and we can use this information to have a lower duty cycle ratio for the specific context. Finally, there is a need to explore how to reduce the time required to learn the DBN structure. A related question to this problem that remains to be explored is: how to decide which context labels to get from a human user in order to reduce the learning time while minimizing the number of labels required.

- **Predictive Prelaunch for Responsive Mobile Applications** Our work on predictive prefetch showed that our prediction model can predict the next mobile application a user is likely to use with high accuracy. This prediction model can be used by the mobile operating system to decide which mobile applications to pre-load and keep in memory so that the applications can be launched quickly by the user. One limitation of our prediction model is that it only utilizes the last sequence of apps used and the time of day information to make predictions. However, mobile operating systems now provide many more signals and contexts that can be useful in improving the app predictions. For instance, many applications in Android OS now post notifications with audible sounds to grab a user's attention. These notifications often lead users to launch mobile applications via the shortcuts embedded in the notification itself. Integration of such signals in the prediction model is a promising direction that can improve the accuracy of the model and in turn, improve the responsiveness of the mobile applications.

BIBLIOGRAPHY

[1] CDC fact sheet on smoking. http://www.cdc.gov/tobacco/data_statistics/fact_sheets/fast_facts/.

[2] Empatica E3 sensor. http://www.empatica.com.

[3] Fitbit Wristband. http://www.fitbit.com.

[4] Google maps app. http://www.google.com/mobile/maps/.

[5] Google Now. http://www.google.com/landing/now/.

[6] Java bayes. http://www.cs.cmu.edu/~javabayes/.

[7] Street bump. http://apps.citizapps.com/apps/bump.

[8] Up by jawbone. http://jawbone.com/up/.

[9] Urban spoon. http://www.urbanspoon.com.

[10] Aggarwal, Charu, Wolf, Joel L., and Yu, Philip S. Caching on the world wide web. *IEEE Trans. on Knowl. and Data Eng. 11*, 1 (Jan. 1999), 94–107.

[11] Agrawal, Sandip, Constandache, Ionut, Gaonkar, Shravan, Roy Choudhury, Romit, Caves, Kevin, and DeRuyter, Frank. Using mobile phones to write in air. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2011), MobiSys '11, ACM, pp. 15–28.

[12] Ali, Amin Ahsan, Hossain, Syed Monowar, Hovsepian, Karen, Rahman, Md. Mahbubur, Plarre, Kurt, and Kumar, Santosh. mpuff: Automated detection of cigarette smoking puffs from respiration measurements. In *Proceedings of the 11th International Conference on Information Processing in Sensor Networks* (New York, NY, USA, 2012), IPSN '12, ACM, pp. 269–280.

[13] Android Activity Recognition API. http://developer.android.com/training/location/activity-recognition.html.

[14] Android Sensors. http://developer.android.com/reference/android/hardware/Sensor.html.

[15] Annavaram, M, Medvidovic, N, Mitra, U, Narayanan, S, Sukhatme, G, Meng, Z, Qiu, S, Kumar, R, Thatte, G, and Spruijt-Metz, D. Multimodal sensing for pediatric obesity applications. *Proceedings of UrbanSense08* (2008), 21–25.

[16] AppKicker. https://play.google.com/store/apps/details?id=org.appkicker.app.

[17] Armstrong, Trevor, Trescases, Olivier, Amza, Cristiana, and de Lara, Eyal. Efficient and transparent dynamic content updates for mobile clients. In *Proceedings of the 4th International Conference on Mobile Systems, Applications and Services* (New York, NY, USA, 2006), MobiSys '06, ACM, pp. 56–68.

[18] Azizyan, Martin, and Choudhury, Romit Roy. Surroundsense: Mobile phone localization using ambient sound and light. *SIGMOBILE Mob. Comput. Commun. Rev. 13*, 1 (June 2009), 69–72.

[19] Balasubramanian, Niranjan, Balasubramanian, Aruna, and Venkataramani, Arun. Energy consumption in mobile phones: A measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference* (New York, NY, USA, 2009), IMC '09, ACM, pp. 280–293.

[20] Ban, Zhijie, Gu, Zhimin, and Jin, Yu. An online ppm prediction model for web prefetching. In *Proceedings of the 9th Annual ACM International Workshop on Web Information and Data Management* (New York, NY, USA, 2007), WIDM '07, ACM, pp. 89–96.

[21] Bao, L., and Intille, S. S. Activity Recognition from User-Annotated Acceleration Data. *Pervasive Computing* (2004), 1–17.

[22] Bargh, Mortaza S., and de Groote, Robert. Indoor localization based on response rate of bluetooth inquiries. In *Proceedings of the First ACM International Workshop on Mobile Entity Localization and Tracking in GPS-less Environments* (New York, NY, USA, 2008), MELT '08, ACM, pp. 49–54.

[23] Barra Chicote, Roberto, Fernández Martínez, Fernando, Lutfi, Lebai, Binti, Syaheerah, Lucas Cuesta, Juan Manuel, Macías Guarasa, Javier, Montero Martínez, Juan Manuel, San Segundo Hernández, Rubén, and Pardo Muñoz, José Manuel. Acoustic emotion recognition using dynamic bayesian networks and multi-space distributions. ISCA.

[24] Benbasat, Ari Y., and Paradiso, Joseph A. A framework for the automated generation of power-efficient classifiers for embedded sensor nodes. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems* (New York, NY, USA, 2007), SenSys '07, ACM, pp. 219–232.

[25] Benjelloun, Omar, Sarma, Anish Das, Halevy, Alon, and Widom, Jennifer. Uldbs: Databases with uncertainty and lineage. In *Proceedings of the 32nd International Conference on Very Large Data Bases* (2006), VLDB '06, VLDB Endowment, pp. 953–964.

[26] Böhmer, Matthias, and Bauer, Gernot. Exploiting the icon arrangement on mobile devices as information source for context-awareness. In *Proceedings of the 12th International Conference on Human Computer Interaction with Mobile Devices and Services* (New York, NY, USA, 2010), MobileHCI '10, ACM, pp. 195–198.

[27] Böhmer, Matthias, Hecht, Brent, Schöning, Johannes, Krüger, Antonio, and Bauer, Gernot. Falling asleep with angry birds, facebook and kindle: A large scale study on mobile application usage. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services* (New York, NY, USA, 2011), MobileHCI '11, ACM, pp. 47–56.

[28] Böhmer, Matthias, and Krüger, Antonio. A study on icon arrangement by smartphone users. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2013), CHI '13, ACM, pp. 2137–2146.

[29] Breiman, Leo. Random forests. *Machine learning 45*, 1 (2001), 5–32.

[30] Bridle, Robert, and McCreath, Eric. Inducing shortcuts on a mobile phone interface. In *Proceedings of the 11th international conference on Intelligent user interfaces* (New York, NY, USA, 2006), IUI '06, ACM, pp. 327–329.

[31] Campbell, AT., Eisenman, S.B., Lane, N.D., Miluzzo, E., Peterson, R.A, Lu, Hong, Zheng, Xiao, Musolesi, M., Fodor, K., and Ahn, Gahng-Seop. The rise of people-centric sensing. *Internet Computing, IEEE 12*, 4 (July 2008), 12–21.

[32] Cao, Pei, Felten, Edward W., and Li, Kai. Implementation and performance of application-controlled file caching. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 1994), OSDI '94, USENIX Association.

[33] Cao, Xin, Cong, Gao, and Jensen, Christian S. Mining significant semantic locations from gps data. *Proc. VLDB Endow. 3*, 1-2 (Sept. 2010), 1009–1020.

[34] Chen, Mingyu, AlRegib, G., and Juang, Biing-Hwang. Feature processing and modeling for 6d motion gesture recognition. *Multimedia, IEEE Trans. on 15*, 3 (2013), 561–571.

[35] Chen, Yin, Lymberopoulos, Dimitrios, Liu, Jie, and Priyantha, Bodhi. Fm-based indoor localization. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2012), MobiSys '12, ACM, pp. 169–182.

[36] Choudhury, Tanzeem, Borriello, Gaetano, Consolvo, Sunny, Haehnel, Dirk, Harrison, Beverly, Hemingway, Bruce, Hightower, Jeffrey, Klasnja, Predrag "Pedja", Koscher, Karl, LaMarca, Anthony, Landay, James A., LeGrand, Louis, Lester, Jonathan, Rahimi, Ali, Rea, Adam, and Wyatt, Danny. The mobile sensing platform: An embedded activity recognition system. *IEEE Pervasive Computing 7*, 2 (Apr. 2008), 32–41.

[37] Choudhury, Tanzeem, Rehg, James M, Pavlovic, Vladimir, and Pentland, Alex. Boosting and structure learning in dynamic bayesian networks for audio-visual speaker detection. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on* (2002), vol. 3, IEEE, pp. 789–794.

[38] Chu, David, Deshpande, Amol, Hellerstein, Joseph M., and Hong, Wei. Approximate data collection in sensor networks using probabilistic models. In *Proceedings of the 22nd International Conference on Data Engineering* (Washington, DC, USA, 2006), ICDE '06, IEEE Computer Society, pp. 48–.

[39] Chu, David, Kansal, Aman, Liu, Jie, and Zhao, Feng. Mobile apps: its time to move up to condos. In *Proceedings of the 13th USENIX conference on Hot topics in operating systems* (2011), USENIX Association, pp. 16–16.

[40] Chu, David, Lane, Nicholas D., Lai, Ted Tsung-Te, Pang, Cong, Meng, Xiangying, Guo, Qing, Li, Fan, and Zhao, Feng. Balancing energy, latency and accuracy for mobile sensor data classification. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems* (New York, NY, USA, 2011), SenSys '11, ACM, pp. 54–67.

[41] Cleary, John G., Ian, and Witten, Ian H. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications 32* (1984), 396–402.

[42] Constandache, I., Gaonkar, S., Sayler, M., Choudhury, R.R., and Cox, L. Enloc: Energy-efficient localization for mobile phones. In *INFOCOM 2009, IEEE* (april 2009), pp. 2716 –2720.

[43] CReSS. `http://borgwaldt.hauni.com/en/instruments/smoking-machines/smoking-topography-devices/cress-pocket.html`.

[44] Criminisi, Antonio, Shotton, Jamie, and Konukoglu, Ender. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Found. Trends. Comput. Graph. Vis. 7*, 2–3 (Feb. 2012), 81–227.

[45] Curewitz, Kenneth M., Krishnan, P., and Vitter, Jeffrey Scott. Practical prefetching via data compression. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 1993), SIGMOD '93, ACM, pp. 257–266.

[46] Dagum, Paul, and Galper, Adam. Forecasting sleep apnea with dynamic network models. In *Proceedings of the Ninth International Conference on Uncertainty in Artificial Intelligence* (San Francisco, CA, USA, 1993), UAI'93, Morgan Kaufmann Publishers Inc., pp. 64–71.

[47] Dagum, Paul, Galper, Adam, and Horvitz, Eric. Dynamic network models for forecasting. In *Proceedings of the Eighth International Conference on Uncertainty in Artificial Intelligence* (San Francisco, CA, USA, 1992), UAI'92, Morgan Kaufmann Publishers Inc., pp. 41–48.

[48] Dalvi, Nilesh, and Suciu, Dan. Efficient query evaluation on probabilistic databases. *The VLDB Journal 16*, 4 (Oct. 2007), 523–544.

[49] Dan, Asit, Yu, Philip S., and Chung, Jen Yao. Characterization of database access pattern for analytic prediction of buffer hit probability. *The VLDB Journal 4*, 1 (Jan. 1995), 127–154.

[50] Dean, Thomas L, and Kanazawa, Keiji. Probabilistic temporal reasoning. In *AAAI* (1988), pp. 524–529.

[51] Deng, Shuo, and Balakrishnan, Hari. Traffic-aware techniques to reduce 3g/lte wireless energy consumption. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies* (New York, NY, USA, 2012), CoNEXT '12, ACM, pp. 181–192.

[52] Deshpande, Amol, Guestrin, Carlos, Hong, Wei, and Madden, Samuel. Exploiting correlated attributes in acquisitional query processing. In *Proceedings of the 21st International Conference on Data Engineering* (Washington, DC, USA, 2005), ICDE '05, IEEE Computer Society, pp. 143–154.

[53] Deshpande, Amol, Guestrin, Carlos, Madden, Samuel R., Hellerstein, Joseph M., and Hong, Wei. Model-driven data acquisition in sensor networks. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30* (2004), VLDB '04, VLDB Endowment, pp. 588–599.

[54] Deshpande, Mukund, and Karypis, George. Selective markov models for predicting web page accesses. *ACM Trans. Internet Technol. 4*, 2 (May 2004), 163–184.

[55] Dong, Yujie, Hoover, Adam, Scisco, Jenna, and Muth, Eric. A new method for measuring meal intake in humans via automated wrist motion tracking. *Applied psychophysiology and biofeedback 37*, 3 (2012), 205–215.

[56] Dutta, Prabal, Aoki, Paul M., Kumar, Neil, Mainwaring, Alan, Myers, Chris, Willett, Wesley, and Woodruff, Allison. Common sense: Participatory urban sensing using a network of handheld air quality monitors. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems* (New York, NY, USA, 2009), SenSys '09, ACM, pp. 349–350.

[57] Elmezain, Mahmoud, Al-Hamadi, Ayoub, Sadek, Samy, and Michaelis, Bernd. Robust methods for hand gesture spotting and recognition using hidden markov models and conditional random fields. In *Proceedings of the The 10th IEEE International Symposium on Signal Processing and Information Technology* (Washington, DC, USA, 2010), ISSPIT '10, IEEE Computer Society, pp. 131–136.

[58] Eriksson, Jakob, Girod, Lewis, Hull, Bret, Newton, Ryan, Madden, Samuel, and Balakrishnan, Hari. The pothole patrol: Using a mobile sensor network for road surface monitoring. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2008), MobiSys '08, ACM, pp. 29–39.

[59] Facebook. `http://www.facebook.com`.

[60] Fan, Li, Cao, Pei, Lin, Wei, and Jacobson, Quinn. Web prefetching between low-bandwidth clients and proxies: Potential and performance. In *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (New York, NY, USA, 1999), SIGMETRICS '99, ACM, pp. 178–187.

[61] Feiertag, Richard J, and Organick, Elliott I. The multics input/output system. In *Proceedings of the third ACM symposium on Operating systems principles* (1971), ACM, pp. 35–41.

[62] FieldStream. `http://www.fieldstream.org`.

[63] Fonseca, Rodrigo, Dutta, Prabal, Levis, Philip, and Stoica, Ion. Quanto: Tracking energy in networked embedded systems. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2008), OSDI'08, USENIX Association, pp. 323–338.

[64] Forbes, Jeff, Huang, Tim, Kanazawa, Keiji, and Russell, Stuart. The batmobile: Towards a bayesian automated taxi. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2* (San Francisco, CA, USA, 1995), IJCAI'95, Morgan Kaufmann Publishers Inc., pp. 1878–1885.

[65] Foursquare. `http://www.foursquare.com`.

[66] Friedman, Nir, Murphy, Kevin, and Russell, Stuart. Learning the structure of dynamic probabilistic networks. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence* (San Francisco, CA, USA, 1998), UAI'98, Morgan Kaufmann Publishers Inc., pp. 139–147.

[67] Ganti, Raghu K., Jayachandran, Praveen, Abdelzaher, Tarek F., and Stankovic, John A. Satire: A software architecture for smart attire. In *Proceedings of the 4th International Conference on Mobile Systems, Applications and Services* (New York, NY, USA, 2006), MobiSys '06, ACM, pp. 110–123.

[68] Garg, Ashutosh, Pavlovic, Vladimir, and Rehg, James M. Audio-visual speaker detection using dynamic bayesian networks. In *Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition 2000* (Washington, DC, USA, 2000), FG '00, IEEE Computer Society, pp. 384–.

[69] Google Cloud Messaging for Aandroid. `http://developer.android.com/google/gcm/index.html`.

[70] Götz, Michaela, Nath, Suman, and Gehrke, Johannes. Maskit: privately releasing user context streams for personalized mobile applications. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2012), SIGMOD '12, ACM, pp. 289–300.

[71] Griffioen, James, and Appleton, Randy. Reducing file system latency using a predictive approach. In *Proceedings of the USENIX Summer 1994 Technical Conference on USENIX Summer 1994 Technical Conference - Volume 1* (Berkeley, CA, USA, 1994), USTC'94, USENIX Association, pp. 13–13.

[72] Gupta, Rahul, and Sarawagi, Sunita. Creating probabilistic databases from information extraction models. In *Proceedings of the 32nd International Conference on Very Large Data Bases* (2006), VLDB '06, VLDB Endowment, pp. 965–976.

[73] Hall, Mark, Frank, Eibe, Holmes, Geoffrey, Pfahringer, Bernhard, Reutemann, Peter, and Witten, Ian H. The weka data mining software: An update. In *SIGKDD Explorations* (2009), pp. 10–18.

[74] Hammond, David, Fong, Geoffrey T, Cummings, K Michael, and Hyland, Andrew. Smoking topography, brand switching, and nicotine delivery: results from an in vivo study. *Cancer Epidemiology Biomarkers & Prevention 14*, 6 (2005), 1370–1375.

[75] Hernandez, Javier, Hoque, Mohammed (Ehsan), Drevo, Will, and Picard, Rosalind W. Mood meter: Counting smiles in the wild. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (New York, NY, USA, 2012), UbiComp '12, ACM, pp. 301–310.

[76] Herning, Ronald I, Jones, Reese T, Bachman, John, and Mines, Allan H. Puff volume increases when low-nicotine cigarettes are smoked. *British medical journal (Clinical research ed.) 283*, 6285 (1981), 187.

[77] Higgins, Brett D., Flinn, Jason, Giuli, T. J., Noble, Brian, Peplin, Christopher, and Watson, David. Informed mobile prefetching. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2012), MobiSys '12, ACM, pp. 155–168.

[78] Hua, Nan, Lall, Ashwin, Romberg, Justin, Xu, Jun (Jim), al'Absi, Mustafa, Ertin, Emre, Kumar, Santosh, and Suri, Shikhar. Just-in-time sampling and pre-filtering for wearable physiological sensors: Going from days to weeks of operation on a single charge. In *Wireless Health 2010* (New York, NY, USA, 2010), WH '10, ACM, pp. 54–63.

[79] iOS Background App Refresh. `http://support.apple.com/kb/ht4211`.

[80] iOS Core Motion Framework. `https://developer.apple.com/library/ios/documentation/coremotion/reference/coremotion_reference/`.

[81] iOS Push Notification Service. https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html.

[82] Karlen, W., Mattiussi, C., and Floreano, D. Sleep and wake classification with ecg and respiratory effort signals. *Biomedical Circuits and Systems, IEEE Transactions on 3*, 2 (April 2009), 71–78.

[83] Kay, Matthew, Choe, Eun Kyoung, Shepherd, Jesse, Greenstein, Benjamin, Watson, Nathaniel, Consolvo, Sunny, and Kientz, Julie A. Lullaby: A capture & access system for understanding the sleep environment. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (New York, NY, USA, 2012), UbiComp '12, ACM, pp. 226–234.

[84] Khalil, Faten, Li, Jiuyong, and Wang, Hua. Integrating recommendation models for improved web page prediction accuracy. In *Proceedings of the Thirty-first Australasian Conference on Computer Science - Volume 74* (Darlinghurst, Australia, Australia, 2008), ACSC '08, Australian Computer Society, Inc., pp. 91–100.

[85] Kistler, James J., and Satyanarayanan, M. Disconnected operation in the coda file system. *ACM Trans. Comput. Syst. 10*, 1 (Feb. 1992), 3–25.

[86] Koller, Daphne, and Friedman, Nir. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[87] Krause, Andreas, and Guestrin, Carlos. Near-optimal Nonmyopic Value of Information in Graphical Models. In *UAI* (2005), pp. 324–331.

[88] Krause, Andreas, Horvitz, Eric, Kansal, Aman, and Zhao, Feng. Toward community sensing. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks* (Washington, DC, USA, 2008), IPSN '08, IEEE Computer Society, pp. 481–492.

[89] Kroeger, Tom M, and Long, Darrell DE. Design and implementation of a predictive file prefetching algorithm. In *USENIX Annual Technical Conference, General Track* (2001), pp. 105–118.

[90] Lafferty, John D., McCallum, Andrew, and Pereira, Fernando C. N. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning* (San Francisco, CA, USA, 2001), ICML '01, Morgan Kaufmann Publishers Inc., pp. 282–289.

[91] Lane, Nicholas D, Mohammod, Mashfiqui, Lin, Mu, Yang, Xiaochao, Lu, Hong, Ali, Shahid, Doryab, Afsaneh, Berke, Ethan, Choudhury, Tanzeem, and Campbell, Andrew. Bewell: A smartphone application to monitor, model and promote well-being. In *5th International ICST Conference on Pervasive Computing Technologies for Healthcare* (2011), pp. 23–26.

[92] Lane, Nicholas D., Xu, Ye, Lu, Hong, Hu, Shaohan, Choudhury, Tanzeem, Campbell, Andrew T., and Zhao, Feng. Enabling large-scale human activity inference on smartphones using community similarity networks (csn). In *Proceedings of the 13th International Conference on Ubiquitous Computing* (New York, NY, USA, 2011), UbiComp '11, ACM, pp. 355–364.

[93] Lee, Chul Min, and Narayanan, S.S. Toward detecting emotions in spoken dialogs. *Speech and Audio Processing, IEEE Transactions on 13*, 2 (March 2005), 293–303.

[94] Lei, Hui, and Duchamp, Dan. An analytical approach to file prefetching. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 1997), ATEC '97, USENIX Association, pp. 21–21.

[95] Lester, Jonathan, Choudhury, Tanzeem, and Borriello, Gaetano. A practical approach to recognizing physical activities. In *Proceedings of the 4th International Conference on Pervasive Computing* (Berlin, Heidelberg, 2006), PERVASIVE'06, Springer-Verlag, pp. 1–16.

[96] Lester, Jonathan, Choudhury, Tanzeem, Borriello, Gaetano, Consolvo, Sunny, Landay, James, Everitt, Kate, and Smith, Ian. Sensing and Modeling Activities to Support Physical Fitness. In *UbiComp '05 Workshop: W10 – Monitoring, Measuring, & Motivating* (Sept. 2005).

[97] Lester, Jonathan, Choudhury, Tanzeem, Kern, Nicky, Borriello, Gaetano, and Hannaford, Blake. A hybrid discriminative/generative approach for modeling human activities. In *Proceedings of International Joint Conference on Artificial Intelligence - Volume 5*, IJCAI'05.

[98] Li, Ming, Ganesan, Deepak, and Shenoy, Prashant. Presto: Feedback-driven data management in sensor networks. In *Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3* (Berkeley, CA, USA, 2006), NSDI'06, USENIX Association, pp. 23–23.

[99] Liu, D. C., and Nocedal, J. On the limited memory bfgs method for large scale optimization. *Math. Program. 45*, 3 (Dec. 1989), 503–528.

[100] Liu, Jiayang, Zhong, Lin, Wickramasuriya, Jehan, and Vasudevan, Venu. uwave: Accelerometer-based personalized gesture recognition and its applications. vol. 5, Elsevier Science Publishers B. V., pp. 657–675.

[101] Liu, Juhong, Wolfson, Ouri, and Yin, Huabei. Extracting semantic location from outdoor positioning systems. In *Proceedings of the 7th International Conference on Mobile Data Management* (2006), IEEE Computer Society, p. 73.

[102] Logan, Beth, Healey, Jennifer, Philipose, Matthai, Tapia, Emmanuel Munguia, and Intille, Stephen. A long-term evaluation of sensing modalities for activity recognition. In *Proceedings of the 9th International Conference on Ubiquitous Computing* (Berlin, Heidelberg, 2007), UbiComp '07, Springer-Verlag, pp. 483–500.

[103] Lopez-Meyer, Paulo, Tiffany, Stephen, and Sazonov, Edward. Identification of cigarette smoke inhalations from wearable sensor data using a support vector machine classifier. In *Engineering in Medicine and Biology Society (EMBC), 2012 Annual International Conference of the IEEE* (2012), IEEE, pp. 4050–4053.

[104] Lu, Hong, Frauendorfer, Denise, Rabbi, Mashfiqui, Mast, Marianne Schmid, Chittaranjan, Gokul T., Campbell, Andrew T., Gatica-Perez, Daniel, and Choudhury, Tanzeem. Stresssense: Detecting stress in unconstrained acoustic environments using smartphones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (New York, NY, USA, 2012), UbiComp '12, ACM, pp. 351–360.

[105] Lu, Hong, Pan, Wei, Lane, Nicholas D., Choudhury, Tanzeem, and Campbell, Andrew T. Soundsense: Scalable sound sensing for people-centric applications on mobile phones. In *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2009), MobiSys '09, ACM, pp. 165–178.

[106] Lu, Hong, Yang, Jun, Liu, Zhigang, Lane, Nicholas D., Choudhury, Tanzeem, and Campbell, Andrew T. The jigsaw continuous sensing engine for mobile phone applications. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems* (New York, NY, USA, 2010), SenSys '10, ACM, pp. 71–84.

[107] Lymberopoulos, Dimitrios, Bamis, Athanasios, and Savvides, Andreas. Extracting spatiotemporal human activity patterns in assisted living using a home sensor network. In *Proceedings of the 1st International Conference on PErvasive Technologies Related to Assistive Environments* (New York, NY, USA, 2008), PETRA '08, ACM, pp. 29:1–29:8.

[108] Lymberopoulos, Dimitrios, Riva, Oriana, Strauss, Karin, Mittal, Akshay, and Ntoulas, Alexandros. Pocketweb: Instant web browsing for mobile devices. *SIGARCH Comput. Archit. News 40*, 1 (Mar. 2012), 1–12.

[109] Ma, Ling, Milner, Ben, and Smith, Dan. Acoustic environment classification. *ACM Transactions on Speech and Language Processing (TSLP) 3*, 2 (July 2006), 1–22.

[110] Ma, Ling, Smith, Dan, and Milner, Ben. Context awareness using environmental noise classification. In *Eighth European Conference on Speech Communication and Technology* (2003).

[111] Madhyastha, Tara M., and Reed, Daniel A. Input/output access pattern classification using hidden markov models. In *Proceedings of the Fifth Workshop on I/O in Parallel and Distributed Systems* (New York, NY, USA, 1997), IOPADS '97, ACM, pp. 57–67.

[112] Markatos, Evangelos P, Chronaki, Catherine E, et al. A top-10 approach to prefetching on the web. In *Proceedings of INET* (1998), vol. 98, pp. 276–290.

[113] Matsui, Shouichi, and Yamada, Seiji. Genetic algorithm can optimize hierarchical menus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2008), CHI '08, ACM, pp. 1385–1388.

[114] McKusick, Marshall K., Joy, William N., Leffler, Samuel J., and Fabry, Robert S. A fast file system for unix. *ACM Transactions on Computer Systems (TOCS) 2*, 3 (Aug. 1984), 181–197.

[115] McVoy, Larry W, and Kleiman, Steve R. Extent-like performance from a unix file system. In *USENIX Winter* (1991), vol. 91.

[116] Meliou, Alexandra, Guestrin, Carlos, and Hellerstein, Joseph M. Approximating sensor network queries using in-network summaries. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks* (Washington, DC, USA, 2009), IPSN '09, IEEE Computer Society, pp. 229–240.

[117] Moffat, Alistair. Implementing the ppm data compression scheme. *IEEE Trans. on Communications 38* (1990), 1917–1921.

[118] Mohan, Prashanth, Padmanabhan, Venkata N., and Ramjee, Ramachandran. Nericell: Rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems* (New York, NY, USA, 2008), SenSys '08, ACM, pp. 323–336.

[119] Montoliu, R., Blom, J., and Gatica-Perez, D. Discovering places of interest in everyday life from smartphone data. *Multimedia Tools and Applications* (2012), 1–29.

[120] Moves Android Application. `https://www.moves-app.com`.

[121] Mowry, Todd C., Demke, Angela K., and Krieger, Orran. Automatic compiler-inserted i/o prefetching for out-of-core applications. In *Proceedings of the Second USENIX Symposium on Operating Systems Design and Implementation* (New York, NY, USA, 1996), OSDI '96, ACM, pp. 3–17.

[122] Mozilla Firefox browser. `https://www.mozilla.org/en-US/firefox/desktop/`.

[123] Mozilla Link prefetching. `https://developer.mozilla.org/en-US/docs/Web/HTTP/Link_prefetching_FAQ`.

[124] Mun, Min, Reddy, Sasank, Shilton, Katie, Yau, Nathan, Burke, Jeff, Estrin, Deborah, Hansen, Mark, Howard, Eric, West, Ruth, and Boda, Péter. Peir, the personal environmental impact report, as a platform for participatory sensing systems research. In *Proceedings of the 7th international conference on Mobile systems, applications, and services* (2009), ACM, pp. 55–68.

[125] Nanopoulos, Alexandros, Katsaros, Dimitrios, and Manolopoulos, Yannis. A data mining algorithm for generalized web prefetching. *IEEE Trans. on Knowl. and Data Eng. 15*, 5 (Sept. 2003), 1155–1169.

[126] Natarajan, Annamalai, Parate, Abhinav, Gaiser, Edward, Angarita, Gustavo, Malison, Robert, Marlin, Benjamin, and Ganesan, Deepak. Detecting cocaine use with wearable electrocardiogram sensors. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (New York, NY, USA, 2013), UbiComp '13, ACM, pp. 123–132.

[127] Nath, Suman. Ace: exploiting correlation for energy-efficient and continuous context sensing. In *Proceedings of the 10th international conference on Mobile systems, applications, and services* (New York, NY, USA, 2012), MobiSys '12, ACM, pp. 29–42.

[128] Nemhauser, G. L., Wolsey, L. A., and Fisher, M. L. An analysis of approximations for maximizing submodular set functions-I. *Mathematical Programming 14*, 1 (Dec. 1978), 265–294.

[129] Nodelman, Uri, Shelton, Christian R, and Koller, Daphne. Continuous time bayesian networks. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence* (2002), Morgan Kaufmann Publishers Inc., pp. 378–387.

[130] Padmanabhan, Venkata N., and Mogul, Jeffrey C. Using predictive prefetching to improve world wide web latency. *SIGCOMM Comput. Commun. Rev. 26*, 3 (July 1996), 22–36.

[131] Palmer, Mark. Fido: A cache that learns to fetch. In *In Proceedings of the 17th International Conference on Very Large Data Bases* (1991).

[132] Parate, Abhinav, Böhmer, Matthias, Chu, David, Ganesan, Deepak, and Marlin, Benjamin M. Practical prediction and prefetch for faster access to applications on mobile phones. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (New York, NY, USA, 2013), UbiComp '13, ACM, pp. 275–284.

[133] Parate, Abhinav, Chiu, Meng-Chieh, Chadowitz, Chaniel, Ganesan, Deepak, and Kalogerakis, Evangelos. Risq: Recognizing smoking gestures with inertial sensors on a wristband. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2014), MobiSys '14, ACM, pp. 149–161.

[134] Parate, Abhinav, Chiu, Meng-Chieh, Ganesan, Deepak, and Marlin, Benjamin M. Leveraging graphical models to improve accuracy and reduce privacy risks of mobile sensing. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2013), MobiSys '13, ACM, pp. 83–96.

[135] Parate, Abhinav, and Miklau, Gerome. A framework for safely publishing communication traces. In *Proceedings of the 18th ACM conference on Information and knowledge management* (New York, NY, USA, 2009), CIKM '09, ACM, pp. 1469–1472.

[136] Park, Chulsung, Chou, Pai H, Bai, Ying, Matthews, Robert, and Hibbs, Andrew. An ultra-wearable, wireless, low power ecg monitoring system. In *Proceedings of Biomedical Circuits and Systems Conference, 2006. BioCAS 2006. IEEE* (2006), IEEE, pp. 241–244.

[137] Patterson, R. H., Gibson, G. A., Ginting, E., Stodolsky, D., and Zelenka, J. Informed prefetching and caching. In *Proceedings of the fifteenth ACM symposium on Operating systems principles* (New York, NY, USA, 1995), SOSP '95, ACM, pp. 79–95.

[138] Pearl, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco, USA, 1988.

[139] Peltonen, Vesa, Tuomi, Juha, Klapuri, A, Huopaniemi, Jyri, and Sorsa, Timo. Computational auditory scene recognition. In *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on* (May 2002), vol. 2, pp. II–1941–II–1944.

[140] Picard, Rosalind W. *Affective Computing*. MIT Press, Cambridge, MA, USA, 1997.

[141] Pielot, Martin, Henze, Niels, and Boll, Susanne. Experiments in app stores - how to ask users for their consent? In *CHI '11 Workshop on Ethics, Logs and Videotape: Ethics in Large Scale Trials & User Generated Content* (2011).

[142] Plarre, K., Raij, A, Hossain, S.M., Ali, AA, Nakajima, M., Al'absi, M., Ertin, E., Kamarck, T., Kumar, S., Scott, M., Siewiorek, Daniel, Smailagic, A, and Wittmers, L.E. Continuous inference of psychological stress from sensory measurements collected in the natural environment. In *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on* (April 2011), pp. 97–108.

[143] Poh, Ming-Zher, Loddenkemper, Tobias, Reinsberger, Claus, Swenson, Nicholas C., Goyal, Shubhi, Sabtala, Mangwe C., Madsen, Joseph R., and Picard, Rosalind W. Convulsive seizure detection using a wrist-worn electrodermal activity and accelerometry biosensor. *Epilepsia 53*, 5 (2012), e93–e97.

[144] Popa, Raluca Ada, Balakrishnan, Hari, and Blumberg, Andrew J. Vpriv: protecting privacy in location-based vehicular services. In *Proceedings of the 18th conference on USENIX security symposium* (Berkeley, CA, USA, 2009), SSYM'09, USENIX Association, pp. 335–350.

[145] Qian, Feng, Wang, Zhaoguang, Gerber, Alexandre, Mao, Zhuoqing, Sen, Subhabrata, and Spatscheck, Oliver. Profiling resource usage for mobile applications: a cross-layer approach. In *Proceedings of the 9th international conference on Mobile systems, applications, and services* (2011), ACM, pp. 321–334.

[146] Quwaider, Muhannad, and Biswas, Subir. Body posture identification using hidden markov model with a wearable sensor network. In *Proceedings of the ICST 3rd International Conference on Body Area Networks* (ICST, Brussels, Belgium, Belgium, 2008), BodyNets '08, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 19:1–19:8.

[147] Rabbi, Mashfiqui, Ali, Shahid, Choudhury, Tanzeem, and Berke, Ethan. Passive and in-situ assessment of mental and physical well-being using mobile sensors. In *Proceedings of the 13th International Conference on Ubiquitous Computing* (New York, NY, USA, 2011), UbiComp '11, ACM, pp. 385–394.

[148] Rachuri, Kiran K, Efstratiou, Christos, Leontiadis, Ilias, Mascolo, Cecilia, and Rentfrow, Peter J. Metis: Exploring mobile phone sensing offloading for efficiently supporting social sensing applications. In *Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on* (2013), IEEE, pp. 85–93.

[149] Rachuri, Kiran K., Musolesi, Mirco, Mascolo, Cecilia, Rentfrow, Peter J., Longworth, Chris, and Aucinas, Andrius. Emotionsense: A mobile phones based adaptive platform for experimental social psychology research. In *Proceedings of the 12th ACM International Conference on Ubiquitous Computing* (New York, NY, USA, 2010), Ubicomp '10, ACM, pp. 281–290.

[150] Rahman, Md. Mahbubur, Ali, Amin Ahsan, Plarre, Kurt, al'Absi, Mustafa, Ertin, Emre, and Kumar, Santosh. mconverse: Inferring conversation episodes from respiratory measurements collected in the field. In *Proceedings of the 2nd Conference on Wireless Health* (New York, NY, USA, 2011), WH '11, ACM, pp. 10:1–10:10.

[151] Ravi, Nishkam, Dandekar, Nikhil, Mysore, Preetham, and Littman, Michael L. Activity recognition from accelerometer data. In *Proceedings of the seventeenth Innovative Applications of Artificial Intelligence, IAAI 2005.* (2005), pp. 1541–1546.

[152] Romberg, Justin. Compressive sensing by random convolution. *SIAM J. Img. Sci. 2*, 4 (Nov. 2009), 1098–1128.

[153] Russell, Stuart J., and Norvig, Peter. *Artificial Intelligence: A Modern Approach*, second ed. Prentice Hall, 2002.

[154] Samsung Galaxy Gear. `http://www.samsung.com/us/mobile/wearable-tech`.

[155] Sano, Akane, and Picard, Rosalind W. Stress recognition using wearable sensors and mobile phones. In *Proceedings of the 2013 Humaine Association Conference on Affective Computing and Intelligent Interaction* (Washington, DC, USA, 2013), ACII '13, IEEE Computer Society, pp. 671–676.

[156] Sazonov, E., Metcalfe, K., Lopez-Meyer, P., and Tiffany, S. Rf hand gesture sensor for monitoring of cigarette smoking. In *Sensing Technology (ICST), 2011 Fifth International Conference on* (Nov 2011), pp. 426–430.

[157] Scholl, Philipp M., Kücükyildiz, Nagihan, and Laerhoven, Kristof Van. When do you light a fire?: Capturing tobacco use with situated, wearable sensors. In *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication* (New York, NY, USA, 2013), UbiComp '13 Adjunct, ACM, pp. 1295–1304.

[158] Schuller, B., Villar, R.J., Rigoll, G., and Lang, M. Meta-classifiers in acoustic and linguistic feature fusion-based affect recognition. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on* (March 2005), vol. 1, pp. 325–328.

[159] Sears, Andrew, and Shneiderman, Ben. Split menus: effectively using selection frequency to organize menus. *ACM Transactions on Computer-Human Interaction (TOCHI) 1*, 1 (Mar. 1994), 27–51.

[160] Sen, Prithviraj, Deshpande, Amol, and Getoor, Lise. Prdb: managing and exploiting rich correlations in probabilistic databases. *The VLDB Journal 18*, 5 (2009), 1065–1090.

[161] Sen, Souvik, Choudhury, Romit Roy, Radunovic, Bozidar, and Minka, Tom. Precise indoor localization using phy layer information. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks* (New York, NY, USA, 2011), HotNets-X, ACM, pp. 18:1–18:6.

[162] Sen, Souvik, Lee, Jeongkeun, Kim, Kyu-Han, and Congdon, Paul. Avoiding multipath to revive inbuilding wifi localization. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2013), MobiSys '13, ACM, pp. 249–262.

[163] Shepard, Clayton, Rahmati, Ahmad, Tossell, Chad, Zhong, Lin, and Kortum, Phillip. Livelab: measuring wireless networks and smartphone users in the field. *SIGMETRICS Perform. Eval. Rev. 38*, 3 (Jan. 2011), 15–20.

[164] Shin, Choonsung, Hong, Jin-Hyuk, and Dey, Anind K. Understanding and prediction of mobile application usage for smart phones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (New York, NY, USA, 2012), UbiComp '12, ACM, pp. 173–182.

[165] Shkarin, Dmitry. Ppm: One step to practicality. In *Proc. of the Data Compression Conference* (2002), DCC '02, pp. 202–.

[166] Silveira, Fernando, Eriksson, Brian, Sheth, Anmol, and Sheppard, Adam. Predicting audience responses to movie content from electro-dermal activity signals. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (New York, NY, USA, 2013), UbiComp '13, ACM, pp. 707–716.

[167] SleepBot Android Application. `https://mysleepbot.com`.

[168] Smith, Alan Jay. Sequentiality and prefetching in database systems. *ACM Transactions on Database Systems (TODS) 3*, 3 (1978), 223–247.

[169] Smith, Dan, Ma, Ling, and Ryan, Nick. Acoustic environment as an indicator of social and physical context. *Personal Ubiquitous Comput. 10*, 4 (Mar. 2006), 241–254.

[170] St. Amant, Robert, Horton, Thomas E., and Ritter, Frank E. Model-based evaluation of cell phone menu interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2004), CHI '04, ACM, pp. 343–350.

[171] Stikic, Maja, Van Laerhoven, Kristof, and Schiele, Bernt. Exploring semi-supervised and active learning for activity recognition. In *Wearable Computers, 2008. ISWC 2008. 12th IEEE International Symposium on* (Sept 2008), pp. 81–88.

[172] Sutton, Charles, and McCallum, Andrew. An introduction to conditional random fields. *Foundations and Trends in Machine Learning 4*, 4 (2012), 267–373.

[173] Thawani, Amit, Gopalan, Srividya, Sridhar, V, and Ramamritham, Krithi. Context-aware timely information delivery in mobile environments. *The Computer Journal 50*, 4 (2007), 460–472.

[174] Tropp, Joel A., Laska, Jason N., Duarte, Marco F., Romberg, Justin K., and Baraniuk, Richard G. Beyond nyquist: Efficient sampling of sparse bandlimited signals. *Information Theory, IEEE Transactions on 56*, 1 (Jan. 2010), 520–544.

[175] Tropp, Joel A, Wakin, Michael B, Duarte, Marco F, Baron, Dror, and Baraniuk, Richard G. Random filters for compressive sampling and reconstruction. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on* (2006), vol. 3, IEEE, pp. III–III.

[176] Varkey, John Paul, Pompili, Dario, and Walls, Theodore A. Human motion recognition using a wireless sensor-based wearable system. *Personal and Ubiquitous Computing 16*, 7 (Oct. 2012), 897–910.

[177] Vartiainen, Elina, Roto, Virpi, and Popescu, Andrei. Auto-update: A concept for automatic downloading of web content to a mobile device. In *Proceedings of the 4th International Conference on Mobile Technology, Applications, and Systems and the 1st International Symposium on Computer Human Interaction in Mobile Technology* (New York, NY, USA, 2007), Mobility '07, ACM, pp. 683–689.

[178] Verkasalo, Hannu. Contextual patterns in mobile service usage. *Personal and Ubiquitous Computing 13*, 5 (June 2009), 331–342.

[179] Vetek, Akos, Flanagan, John A., Colley, Ashley, and Keränen, Tuomas. Smartactions: Context-aware mobile phone shortcuts. In *Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction: Part I* (Berlin, Heidelberg, 2009), INTERACT '09, Springer-Verlag, pp. 796–799.

[180] Vitter, Jeffrey Scott, and Krishnan, P. Optimal prefetching via data compression. *J. ACM 43*, 5 (Sept. 1996), 771–793.

[181] Vlasic, Daniel, Adelsberger, Rolf, Vannucci, Giovanni, Barnwell, John, Gross, Markus, Matusik, Wojciech, and Popović, Jovan. Practical motion capture in everyday surroundings. *ACM Transactions on Graphics (TOG) 26*, 3 (July 2007).

[182] Wang, Daisy Zhe, Michelakis, Eirinaios, Garofalakis, Minos, and Hellerstein, Joseph M. Bayesstore: managing large, uncertain data repositories with probabilistic graphical models. *Proceedings of the VLDB Endowment 1*, 1 (Aug. 2008), 340–351.

[183] Wang, He, Sen, Souvik, Elgohary, Ahmed, Farid, Moustafa, Youssef, Moustafa, and Choudhury, Romit Roy. No need to war-drive: Unsupervised indoor localization. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2012), MobiSys '12, ACM, pp. 197–210.

[184] Wang, Sy Bor, Quattoni, Ariadna, Morency, Louis-Philippe, and Demirdjian, David. Hidden conditional random fields for gesture recognition. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2* (2006), pp. 1521–1527.

[185] Wang, Yi, Lin, Jialiu, Annavaram, Murali, Jacobson, Quinn A, Hong, Jason, Krishnamachari, Bhaskar, and Sadeh, Norman. A framework of energy efficient mobile sensing for automatic user state recognition. In *Proceedings of the 7th international conference on Mobile systems, applications, and services* (2009), ACM, pp. 179–192.

[186] Yan, Tingxin, Chu, David, Ganesan, Deepak, Kansal, Aman, and Liu, Jie. Fast app launching for mobile devices using predictive user context. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2012), MobiSys '12, ACM, pp. 113–126.

[187] Yang, Hee-Deok, Park, A-Yeon, and Lee, Seong-Whan. Gesture spotting and recognition for human-robot interaction. *Robotics, IEEE Transactions on 23*, 2 (2007), 256–270.

[188] Yang, Jun, Lu, Hong, Liu, Zhigang, and Boda, PterPl. Physical activity recognition with mobile phones: Challenges, methods, and applications. In *Multimedia Interaction and Intelligent User Interfaces*, Ling Shao, Caifeng Shan, Jiebo Luo, and Minoru Etoh, Eds., Advances in Pattern Recognition. Springer London, 2010, pp. 185–213.

[189] Ye, Mao, Shou, Dong, Lee, Wang-Chien, Yin, Peifeng, and Janowicz, Krzysztof. On the semantic annotation of places in location-based social networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2011), KDD '11, ACM, pp. 520–528.

[190] Yelp. `http://www.yelp.com`.

[191] Yin, Ying, and Davis, Randall. Toward natural interaction in the real world: Real-time gesture recognition. In *International Conference on Multimodal Interfaces and the Workshop on Machine Learning for Multimodal Interaction* (New York, NY, USA, 2010), ICMI-MLMI '10, ACM, pp. 15:1–15:8.

[192] Zephyr BioHarness 3 Chestband. http://zephyranywhere.com/products/bioharness-3/.

[193] Zhang, Yongmian, and Ji, Qiang. Active and dynamic information fusion for multisensor systems with dynamic bayesian networks. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 36*, 2 (Apr. 2006), 467–472.

[194] Zhao, Zhongtang, Chen, Yiqiang, Liu, Junfa, Shen, Zhiqi, and Liu, Mingjie. Cross-people mobile-phone based activity recognition. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three* (2011), IJCAI'11, AAAI Press, pp. 2545–2550.