Carnegie Mellon University

CARNEGIE INSTITUTE OF TECHNOLOGY

THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF Doctor of Philosophy

TITLE **Mobile Behaviometrics: Behavior Modeling from Heterogeneous Sensor Time-series**

PRESENTED BY **Jiang Zhu**

ACCEPTED BY THE DEPARTMENT OF

Electrical and Computer Engineering

Joy Zhang — ADVISOR, MAJOR PROFESSOR — 5/2/14 — DATE

Jelena Kovacevic — DEPARTMENT HEAD — 5/2/14 — DATE

APPROVED BY THE COLLEGE COUNCIL

Vijayakumar Bhagavatula — DEAN — 5/1/14 — DATE

Mobile Behaviometrics: Behavior Modeling from Heterogeneous Sensor Time-series

Submitted in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the
Electrical and Computer Engineering

Jiang Zhu

M.S., Electrical Engineering, Stanford University
M.S., Management Science and Engineering, Stanford University
B.E., Control Engineering, Tsinghua University

Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, Pennsylvania

May, 2014

Mobile Behaviometrics: Behavior Modeling from Heterogeneous Sensor Time-series

ABSTRACT

In the era of ubiquitous and mobile computing, the penetration of mobile devices equipped with various embedded sensors makes it possible to capture the physical and virtual context of the user and his/her surrounding environment. There is also an increasing demand to model human behaviors based on those data to enable context-aware computing and people-centric applications, which utilize users' behavior pattern to improve the existing services or develop new services.

Over the decades, we have seen tremendous success in biometrics technologies being used in all types of applications based on the physical attributes of the individual such as face, fingerprints, voice and iris. Inspired by this, we introduce a new concept *Mobile Behaviometrics*, which uses algorithms and models to measure and quantify unique human behavioral patterns in place of human bio-attributes. Behaviometrics algorithms take multiple data from various sensors as input and fuse them to build behavioral models which are capable of producing application specific quantitative analysis on the unique individuals that were the originators of the data.

We have three main challenges: statistical modeling of the sensor data, difficulty of multi-dimensional sensor fusion and structural activity recognition. Based on our observation of the similarity between human behavior and natural language, we apply natural language processing (NLP) techniques to statistically model heterogeneous sensor data and investigate various applications based on the derived models. We also investigate adapting conventional machine learning techniques to human behavioral modeling for non time-sensitive behavior sequences. To overcome the difficulty of multi-dimensional sensor fusion, we develop a hybrid approach to first learn behavioral factors for feature dimension reduction and then infer optimal way of fusing models from different factors. We study a novel *Helix* algorithm to induce the underlying grammar of human activities and ultimately solve the structural activity recognition.

Our analytical work has been incorporated into a mobile security application *SenSec*, which

allows us to conduct extensive experiments on behavioral modeling and its applications through passive sensing. Through the deployment and field tests of *SenSec*, we conclude that the proposed modeling algorithms can effectively identify users and their unique behavioral patterns and eventually detect anomalies from suspicious behaviors. Our *SenSec* application and its foundation application *MobiSens*[1] are publicly available. They continuously collect huge amount of mobile sensing data which will enable a broader range of research in modeling and analyzing human behaviors in the near future.

[1] SenSec is built on the same framework of MobiSens application is originally designed and implemented. Therefore, publicly, we usually refer MobiSens as both the framework and the application.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Professor Ying Zhang, for his support, inspiration, encouragement and trust, without which this thesis can never be possible. It was my fortune and privilege to work with him. Professor Zhang has always been a fountain of knowledge for me no matter what issue challenged me during the course of my work. I am truly grateful for the research freedom he gave me on developing expertise of my own, his willingness to share his deep knowledge and strong analytical skills whenever I was at stuck, and his encouragement and support on my research and choice of career path.

I am very grateful for my outstanding thesis committee members – Professor Jason Hong, Professor Patrick Tague, Dr. Fabio Maino and Dr. Flavio Bonomi. Professor Hong has been a wonderful source of insight and enthusiasm that has helped me identify the strengths and improve the weaknesses of my thesis work. I would miss the enjoyable individual discussions with Professor Tague, and would like to thank him for the valuable advice and support for the course project where a major portion of this research work is based on. I have learned immensely from collaborations with Dr. Maino and Dr. Bonomi at Cisco Research during the past 3 years, and truly appreciate their deep and insightful comments, which have greatly improved this thesis.

I would like to thank my colleagues and friends at Carnegie Mellon. Thanks to Huan-Kai Peng and Dongzhen Piao, who have always been willing to offer help during my first year; to Pang Wu, Sean Wang and Ben Draffin, for great research discussions and their great support on MobiSens, SenSec and KeySens system implementations; to Le Nguyen, Ming Zeng and Fei Xiong, thanks to all of you for the warm and friendly lab atmosphere and the sharing of your knowledge during the group discussions. Another special thanks to Dr. Hao Hu from Cisco Research and Jiatong Zhou from University of New South Wales for carefully proofreading and editing the manuscript. I would also like to thank the group of friends and colleagues in the ECE department for making my study

in Carnegie Mellon such an enjoyable and rewarding process.

This work would not have been possible without the support from my family. I am indebted to the support and encouragement from my parents. My Mom and Dad have always cared for me, tried their best to understand every detail of my work and provide suggestions. I am blessed to have the companionship of my wife, Yuechuan She, and my lovely daughter, Alina, for going through the ups and downs with me throughout these years at Carnegie Mellon. It is my fortune to always have you two by my side.

Finally, this thesis research was supported by CyLab at Carnegie Mellon under grants DAAD19-02-1-0389 and W911NF-09-1-0273, from the Army Research Office (ARO), Nokia research award on "Mobile Sensing and Behavior Modeling for Social Computing," Google research award on "Social Behavior Sensing and Reality Mining," and a Cisco research award on "Behavior Modeling for Human Network." I greatly appreciate the financial support from these sponsors.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Nearly 40 years ago, IBM suggested that a computer user could be recognized at a computer terminal by something he knows or memorizes, by something he carries, or even by a personal physical characteristic. This analysis was done in the context of computer data security - remotely recognizing those authorized to access stored data - and specifically referenced voice recognition as a personal physical characteristic useful for human recognition, although automated handwriting, fingerprint, face, and hand geometry systems were, by 1970, also under development. Since that time, automatically recognizing people by physical and behavioral characteristics has come to be known as biometric authentication and its applications have broadened far beyond the remote recognition of computer terminal users. Today, biometric technologies are being used in all types of applications not foreseen by its early pioneers especially in areas where using something you know or something you carry is either not secure enough or not feasible such as visa and passport issuance, social service administration, and entertainment ticket management systems.

By formal definition, *biometrics* is the science of recognizing the identity of a person based on the physical or behavioral attributes of the individual such as face, fingerprints, voice and iris. With the pronounced need for robust human recognition techniques in critical applications such as secure access control, international border crossing and law enforcement, biometrics has positioned itself

as a viable technology that can be integrated into large-scale identity management systems. Biometric systems operate under the premise that many of the physical or behavioral characteristics of humans are distinctive to an individual, and that they can be reliably acquired via *appropriately designed sensors* and represented in a numerical format that lends itself to automatic decision-making in the context of identity management. Now, biometric technology is a rapidly evolving field, resulting in the development of innovative sensors, novel feature extraction and matching algorithms, enhanced test methodologies and cutting-edge applications, even though it has yet penetrated everyone's daily life.

On the other hand, with the advancement and proliferation of digital electronics and portability having become a major driving factor for innovation, mobile devices and their applications are becoming ubiquitous and will increasingly change the way we live, work and play. Those devices, such as smartphones, tablets and laptops, make our lives convenient in ways that were unimaginable before and have become an indispensable part of our lives. Keeping in mind the successfulness of biometrics, can we improve existing applications and enable new services for the future ubiquitous computing?

Modern mobile devices are equipped with various embedded sensors that make it possible to capture the physical and virtual context of the user and surrounding environment. The fact that these embedded sensors have become so pervasive combined with the increasing popularity of context-aware computing and people-centric applications, has created the possibility to model human behaviors based on sensory data. With models developed, we can utilize a user's behavior pattern to improve existing services, such as casual authentication and anomaly detection, or enable new services, such as customized intelligence and patient monitoring and support in healthcare.

In this thesis, we introduce a new concept, *Mobile Behaviometrics*, which comes from mobile "behavior" and "biometrics", uses algorithms and models to measure and quantify the unique human behavioral patterns and natural rhythms that every user has when operating and interacting with their mobile devices. Many of the characteristics we explore here through *Behaviometrics* are highly

related to biometrics, such as how a user retrieves the phone, how she holds it while using certain applications, and how she types on the keyboard. Behaviometric algorithms take multiple streams of data collected from various sensors (GPS, WiFi, Accelerometers, Gyroscopes, Bluetooth, etc) and other context-related information sources (Mouse movement and clicks, screen touches and slides, keyboard strokes, application usage, etc) as input, and fuses them to build behavioral models which are capable of producing application specific quantitative analysis on the unique individuals that were the originators of the data.

1.1 RESEARCH OVERVIEW

The fundamental question we are trying to answer is: given a set of sequences of sensory data, how to build a statistical model to represent the user who directly or indirectly generated those sequences, such that the model can identify if a new sequence of sensory data is generated by the original user? Empirically, we observe similar structure between human behavior and natural language, i.e., both the sequence of behaviors and the sequence of words are generally following certain patterns. This inspired us to investigate the application of natural language processing (NLP) techniques to Behaviometric modeling. It is also possible to adapt and improve conventional machine learning techniques, e.g., neural networks, to efficiently build non time-sensitive behavior models.

Once we start to build sensory data based user identifications algorithms, we observe one key challenge, i.e., **heterogeneous sensor fusion**. In standard sequence classification, input data are usually one-dimension sequences, such as words, phrases or DNA nucleotides. For mobile Behaviometrics, behaviors are captured and represented by various sensors such as accelerometers, gyroscopes, GPS receiver, WiFi receiver, microphone, keyboard, touch screen, and so on. These sensors are of heterogeneous data format (e.g., real number value or location coordinates), sampling rate, and semantic meaning. In this thesis, we consider a hybrid approach to deal with this problem, which consists of two phases. In the learning phase, we model user behaviors for each predefined category of factors separately. Within the category, we use various techniques to reduce the

dimensionality of the feature space. In the inference phase, we adopt a voting system that combines scores adaptively, boosting or filtering, from models that represent different factors.

Another practical challenge is the need for unsupervised structural activity recognition. Supervised methods use samples with predefined labels such as "walking" or "running". Practically, however, not only it is unrealistic to assume large amounts of labeled data, but also it's limited to recognize only the predefined activities. In many applications, it is more useful to recognize activities at high-level, or more generally, in a structured manner. Such an approach will make it possible to identify high-level activities like "playing a tennis game", lower-level ones like "a play", or even breaking down to "step" or "swing", etc. Inspired by grammar induction in natural language processing, we study a novel approach called *Helix* to induce the underlying grammar of human activities.

In our language approach, at first, we convert the raw sensory data into a behavior text representation as sequences of behavior labels. Each behavior label is considered as a word in the language. We then train a continuous n-gram language model on these labels. We also evaluate several NLP techniques in this new problem context and adapt them in performing activity segmentation, recognition, classification, prediction and anomaly detection. To overcome the aforementioned practical challenge, our proposed unsupervised algorithm *Helix* induces the underlying grammar or structure of human behaviors using partial or unlabeled heterogenous sensor data and discovers the hierarchy of the activities considering various granularities. It first generates an initial vocabulary using unlabeled sensory data, followed by iteratively combining statistically collocated sub-activities across multiple sensor dimensions and grouping similar activities together to discover higher level activities. We also perform a separate study which focuses specifically on modeling motion data, and by taking advantage of its structure we are able to produce a different method for activity classification. For some sensory data categories and application contexts, we also apply traditional machine learning approaches, such as Naive Bayes, Mixed Gaussian, and Neural Networks, in modeling mobile behaviors to complement the approaches mentioned in the aforementioned language models.

We apply these *Behaviometric* models and algorithms in the design and development of a mobile security application *SenSec*, which builds user behavioral models through passive sensing. These models are then used to perform user identification and casual authentication. Through the deployment and field tests of *SenSec*, we conclude that we are able to model and identify users via *Behaviometric* analysis of the sensor data using the above techniques. We first focus on the factors accessible from motion and posture sensors, for example, accelerometers, gyroscope, orientation sensors and magnetometers by building generative models of user's motion and posture characteristics to use for identification and authentication. This generative model is based on the "language" as a motion principle which we show can be used for activity segmentation and recognition, anomaly detection. Then we build on this model by introducing location and mobility aspects with WiFi traces, which can be used to enhance the anomaly detection capabilities of the model as well as to predict future behavior. Then we take a look at user identification and anomaly detection through the approach of using classic machine learning algorithms by deriving a method to extract an "activity level" from data and build models based on these levels. We improve this by adding location data into the mix, showing that with this new factor, we are able to lower the false positive rate and boost the accuracy of the model. Finally, we include the factors obtained from user interactions with the mobile devices, specifically, software keyboard inputs. These factors describe physical aspects of individuals. From the experiments conducted with human subjects using these applications, we are also able to provide guidelines in selecting different models, tuning model parameters and improving user experience, especially in handling errors and false-positives in prediction and anomaly detection.

1.2 THESIS STATEMENT

In this thesis, we study the fundamental scientific problem of *modeling an individual's behavior from heterogeneous sensor time-series*. Sensor information is monitored from physical sensors such as accelerometers and gyroscopes on mobile phones and "soft sensors" such as the key-typing

patterns on laptops. The users' behavior models are primarily used in this work to identify the user and detect anomalies in usage patterns.

Our contributions to the theoretical development of Behaviometrics include:

1. We propose a novel language-based approach to study mobile user behaviors. We develop key steps to convert the raw sensory data into behavior text representation as sequences of behavior labels and then to apply NLP techniques in performing activity segmentation, recognition and classification, prediction and anomaly detection.

2. We propose a novel unsupervised algorithm *Helix* to induce the underlying "grammar" of human behaviors using partial or unlabeled heterogenous sensory data and to discover the hierarchy of the activities considering various granularities. We also propose an extension *Helix* Tree that use such structures to efficiently perform classification and regression in various granularities.

3. We investigate how conventional machine learning techniques can be adapted and applied to this new Behaviometrics modeling when the language based approach is less efficient. We apply data preprocessing techniques, e.g., DFT and entropy-based feature selection algorithms for user identification/authentication based on activity recognition. We also investigate sensor fusion technologies to improve the performance.

Our contributions to the advancement of Behaviometrics applications include:

1. We develop a Behaviometrics framework which incorporates sensing, data preprocessing, data modeling, application and evaluation. This generic framework can be utilized to implement future Behaviometrics applications.

2. We build and deploy a mobile security application SenSec based on our mobile sensing platform, MobiSens, and release it to the public domain.

3. We conduct extensive experiments with SenSec collecting location, motion and user interaction traces to build mobile user's behaviometrics for user identification and anomalous behavior detection.

4. Based on our experiments and real user experience, we provide guidelines in selecting different models, tuning model parameters and improving user interaction, especially in handling errors and false-positives in prediction and anomaly detection in related application areas.

5. Our applications deployed in the field, including SenSec and MobiSens, continuously collect huge amounts of mobile sensing data, some of them are labelled or partially labelled for various purposes including security, lifelog, or mental status. These valuable data sets will enable a broader range of research in modeling and analyzing human behaviors in the mobile computing context.

1.3 THESIS ROADMAP

The roadmap of the thesis is shown in Figure 1.1. The thesis is organized as follows: In Chapter 2, the background and related work of user behavior modeling is presented. Then an overview of the proposed *Behaviometrics* framework is first presented in Chapter 3. Later in Chapter 3, we go through the sensor data factors collected, and introduce the *SenSec* app. In Chapters 4, 5, and 6, we will go through the design considerations and the details of the proposed approaches. Chapter 4 explores the analysis techniques and models used throughout this work. We explain the mathematically derived principles used in these models and outline the methods used to interpret them. Chapter 5 extends Chapter 4 by evaluating the effectiveness of the proposed approaches. We present the procedure for dataset collection, evaluation methodologies and experimental results. Finally, in Chapter 6 we discuss challenges and limitations of our work as well as the future research directions. We conclude this thesis in Chapter 7.

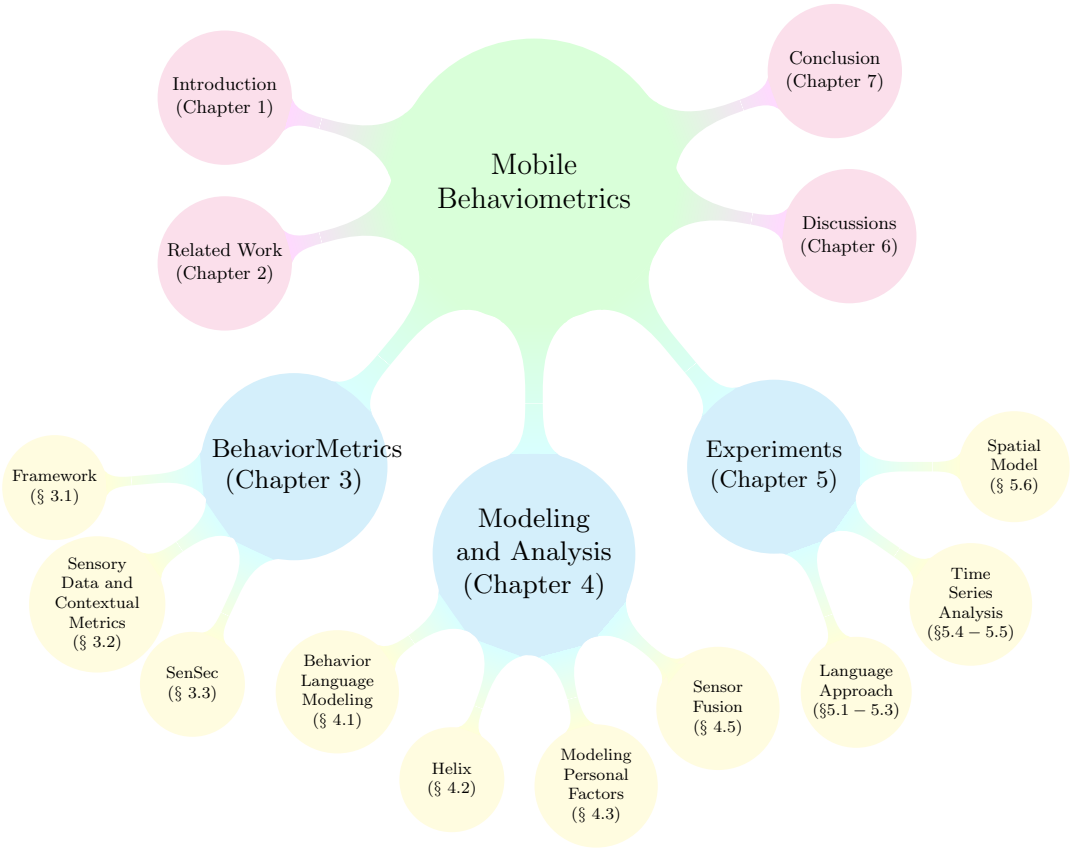

Figure 1.1: Thesis Roadmap

CHAPTER 2

RELATED WORK

There has been extensive prior work on human behavior modeling and analysis in various application domains, especially in the field of biomedical and psychological studies. In the past, interviews, surveys and medical examinations were intensively used to collect information from the human subjects. However, these approaches are not suitable for mobile internet environment, because 1) they are very domain-specific and not flexible enough to capture complex behaviors from human day-to-day activities and routines. 2) some approaches rely on structured and labeled data sets, such as results from medical examinations and questionnaires, which takes a long time to obtain, requires special facility and incurs significant cost. Therefore, practically, the data sets for those studies were relative small and only represent limited sources of information. 3) most of the recent work did leverage data sets collected from the low cost sensors on the mobile devices, but focused on using single sensory data stream. In this section, we review some of the past research that are related to our Behaviometric work.

2.1 MOBILITY BEHAVIOR MODELING AND ANALYSIS

The use of probabilistic methods for modeling user mobility behavior in wireless network including WiFi and Cellular environment has been attempted in various previous work [1–5] with predominate focus on using probabilistic approaches such as Bayesian and Markov models.

Work in [6] by Kim et al. is one of the first attempts to construct a WLAN mobility model from real-world wireless user traces. Using *syslog* approach, they collected traces containing sequences of WLAN association records. They explored several methods to extract mobility tracks, including *triangle centroid*, *time-based centroid* and *Kalman filter*. They also developed a heuristic to extract pause time from mobility tracks. They validated these methods through controlled walks where the tracks (via GPS) and pause-time were also recorded. Using the similar data set, Lee and Hou presented a semi-markov model [4] for characterizing user mobility both in the temporal and spatial domains. With the semi-markov model, they were able to capture and analyze both the steady-state and transient behaviors of user mobility. From the steady-state analysis, they were able to identify the long-term mobility characteristics, such as the steady-state user distribution over APs. Among other contributions, the effect of ping-pong phenomena in user mobility was analyzed and a solution to identify and remove ping-pong transitions was proposed.

As suggested by the close ties between human activity and language, predictive models used in language modeling can also be applied to mobility behavior modeling. Aipperspach et al in [7] showed that smoothed n-grams can provide a fast and accurate method for making single-step predictions on binary sensor output and user's location in a smart-home environment. The results also showed that the mapping between language and human behavior, evidenced through the Zipf distribution of movement sequences and the "local structure" of those sequences, supports the application of language models to user behavior modeling. The simplicity of their approach inspire us to explore similar language approach to model user mobility behavior with fine-grain sensory data with heterogenous nature. We have demonstrated that such simple language approaches can be used to model user activities in our lifelogger system [8] and to perform anomaly detection in our geo-tracing system [9].

2.2 MOBILE APPLICATION SECURITY THROUGH PASSIVE SENSING

Biometric authentication, which applies context-awareness [10, 11] to security applications, verifies user identity by leveraging the uniqueness of behavioral characteristics and/or physical trait. Traditional schemes include fingerprint scanners, iris recognition, voice recognition, face recognition and so forth. Since last decade, novel approaches have sprung out. Orr and Abowd [12] designed a system which can identify users based on their footstep force profiles. Work by Peacock et al. [13] concentrated on keystroke patterns and the underlying typing characteristics to perform user identification. Work by Hayashi et al. [14] examined context-aware scalable authentication, combining a number of passive factors such as location for authentication. Work by Davrondzhon et al. [15] proposed a scheme that uses accelerometers mounted on lower leg for gait recognition and uses it to enforce authentication. Other biometrics including blinking pattern, writing style, etc. were also explored in the past which focus on individuals' behaviors.

More recently, as sensing and computing capabilities become standard on smartphones, researchers have begun to collect more types of sensory data on devices to build user behavior models and use the model to infer certain contexts, including user authentication. In the work of Zheng et al. [16], the GPS readings are used to detect whether a person is walking, running, driving a car, or riding a bus. Schmidt et al. [17] monitors various system parameters on a mobile device, including system free memory, running process count, user inactivity, CPU usage and SMS count, for anomaly detection and IDS. Keng-hao et al. [18] predicts whether the user is interruptible based on the input from microphone and the user's keyboard activities by detecting the unique pattern of holding the television remote control using the accelerometers attached. Jakobsson et al. [19] put forward the notion of implicit authentication in that the authentication strategy is carried out based on what applications and features on a mobile phone are being used. Shi et al. [20] also devised an implicit authentication architecture in which a user model is constructed from a user's past behavior recorded by the mobile device. Then with this model and recently observed user activities, device

can score the trustworthiness of current user and respond accordingly. SenGuard [21], a similar system to *SenSec*, leverages availability of multiple sensors on smartphones and passively use them as sources of user identification in background. It invokes active user authentication when there is a mounting evidence that the phone user has changed.

Our work differs from the aforementioned efforts in that 1) we collected Received Signal Strength (RSS) readings of the beacons emitted from mobile devices and aggregate the data from multiple Wireless Access Points (WAPs) to form sequences of fingerprints, which can be used to easily infer the actual locations of the devices. This approach reduces the overhead to estimate mobility tracks and pause time. 2) We converted heterogenous sensory data into behaviorial text and adopt "n-gram" model to efficiently construct sufficient statistics of user mobility behavior, capturing both the steady-state (uni-gram) and transient behavior (bi-gram and n-gram) simply through counting. 3) We used "*skipped* n-gram" model to capture the long-term dependency in user mobility behavior. Ping-pong phenomena can also be partially mitigated through this method if ping-pong transitions are skipped.

Our work extends beyond the aforementioned efforts in that 1) Instead of using devices mounted to various part of the human body, we explored the possibility of using the sensor data collected from generic smart phones to model user's gesture patterns while users are using the devices. 2) Different from SenGuard, which uses JigSaw engine [22] to detect 5 common physical activities, stationary, walking, cycling, running, and in a vehicle (i.e., car, bus), *SenSec* uses a novel and simple n-gram model to capture individual users' motion gesture. 3) One of our experiments was based on the data collected from a group of users performing the same tasks with the device. This further verified that gesture patterns are very personal and can be used for user identification and classification. 4) We integrated our approach into a prototype system and tested it in participants' the day-to-day usage. Our system can identify non-owner anomaly in less than 5 seconds, faster than existing work [19], and still maintain reasonable True Positive Rate at 85% with False Positive Rate at 10%, which makes our approach more practical for real deployments.

2.3 MICRO-BEHAVIOR MODELING OF KEYBOARD INTERACTION

One of the key areas of research on behavioral models for computer security is to model user interaction through input devices. These include 1) desktop keystroke dynamics using traditional methods [23], 2) mobile phone keypress dynamics using traditional methods [24] and 3) key inference using side-channel sensors [25]. This research can focus on either *structured text* such as passwords/shared secrets or *dynamic text* that tracks keystroke patterns over the duration of a typing session. The testing environment in these studies could be either *controlled* or *uncontrolled*, describing whether the users are in a lab or on their personal machines. Additionally, studies can either focus on *authentication* of a single user or *identification* from among a pool of users [23]. They can also study a wide variety of features, depending on researcher interest and the capabilities of the target equipment.

Desktop computer keystroke dynamics research has a long and rich history. A wide variety of extracted features, learning algorithms, success metrics and testing environments have been studied [13, 23]. While more research has been focused on static text that can observe the patterns of password typing, there are a number of studies that focus on uncontrolled environments with dynamic text [26]. These studies, limited by the capabilities of physical keyboards, tend to gather information about typing latencies, cadences, or error rates. Some special keyboards can also provide pressure information, though this is not common in consumer-grade products. With our application, we leverage the additional data gathering power of a touch screen keyboard and can extend beyond these primarily timing-based features.

Mobile phones have been getting increased attention over the last few years as security threats develop and risks are further revealed. The access to additional data from their touchscreens defines a research environment with both added benefits (e.g. more data) and added challenges (e.g. greater mobility, more dynamic environments). A number of studies have addressed keypress dynamics on mobile phones, though many use similar feature sets as on desktops [27]. Some, however, including

a quite successful one on PINs by Zahid et al., analyze additional features such as the difference in *digraph time* between adjacent and non-adjacent keys [24]. Our work builds on this existing work by focusing on the mobile keypress features not possible on traditional desktops.

Another growing area of research is the ability to leverage the other sensors on mobile phones (notably accelerometers and gyroscopes) to infer keypresses without direct access to keypress data. By measuring the changes in orientation [28], accelerometer readings [25, 28], and/or gyroscopes [28], applications with very few special privileges [1] can provide attackers insight into passwords or PINs. Applications posing as games or with other innocuous guises could tap into these sensor channels in the background and infer keypresses. We have internally discussed, though not examined, how a gaming application could generate its own training data through in-game menus. While we focus on the direct analysis of touches, we leverage this area of research to guide the processing of our own orientation data.

Our work extends beyond the aforementioned efforts in that 1) Instead of using devices mounted to various part of the human body, we explored the possibility of using the sensory data collected from generic smart phones to model user's gesture patterns while users are using the devices. 2) Different from SenGard, which uses JigSaw engine [22] to detect 5 common physical activities, stationary, walking, cycling, running, and in a vehicle (i.e., car, bus), *SenSec* uses a novel and simple n-gram model to capture individual users' motion gesture. 3) One of our experiments was based on the data collected from a group of users performing the same tasks with the device. This further verify that gesture patterns are very personal and can be used for user identification and classification. 4) We integrated our approach into a prototype system and tested it in participants' the day-to-day usage. Our system can identify non-owner anomaly in less than 5 seconds, faster than existing work [19], which makes our approach practical for real deployments.

[1] Android Security Overview, `http://source.android.com/tech/security/index.html`

CHAPTER 3

BEHAVIOMETRICS

The word "Behaviometrics" derives from the terms behavioral and biometrics [29]. Behavioral refers to the way a human person behaves, and biometrics, in an information security context, refers to technologies and methods that measure and analyze biological characteristics of the human body, usually for authentication purposes, for example fingerprints, retina and voice patterns. Behaviometrics, or behavioral biometrics, is a measurable behavior used to recognize or verify the identity of a person or his/her individual behaviors. Unlike biometrics, Behaviometrics focuses on behavioral patterns rather than physical attributes and can be extended to area beyond the aforementioned security context.

A human behavioral pattern consists of a variety of different unique semi-behaviors; all mixed together into a larger and utterly more unique profile. Each person has a unique pattern in how they interact with a mobile device's input components, its keyboard, touch screen, mouse and microphone. Since every person's unique Behaviometric pattern is formed not only by biometric features, like the way you move your hand, but is also influenced by more social and psychological means, like if you are native in the language you write, it is just about impossible to copy or imitate somebody else's behavior when using mobile devices. This makes Behaviometrics very personal and unique, which could be leveraged for various applications.

In this chapter, we first introduce the concept of *Behaviometrics* and its enabling framework. Then, we discuss various types of the sensory data on modern mobile devices and contextual metrics that can be derived from them. Applying the Behaviometric framework in real world scenario, we propose a particular use case of Behaviometrics, SenSec, to passively secure mobile device and application from unauthorized users.

3.1 FRAMEWORK

A generic Behaviometric framework is shown in Figure 3.1. There are five major components or stages:

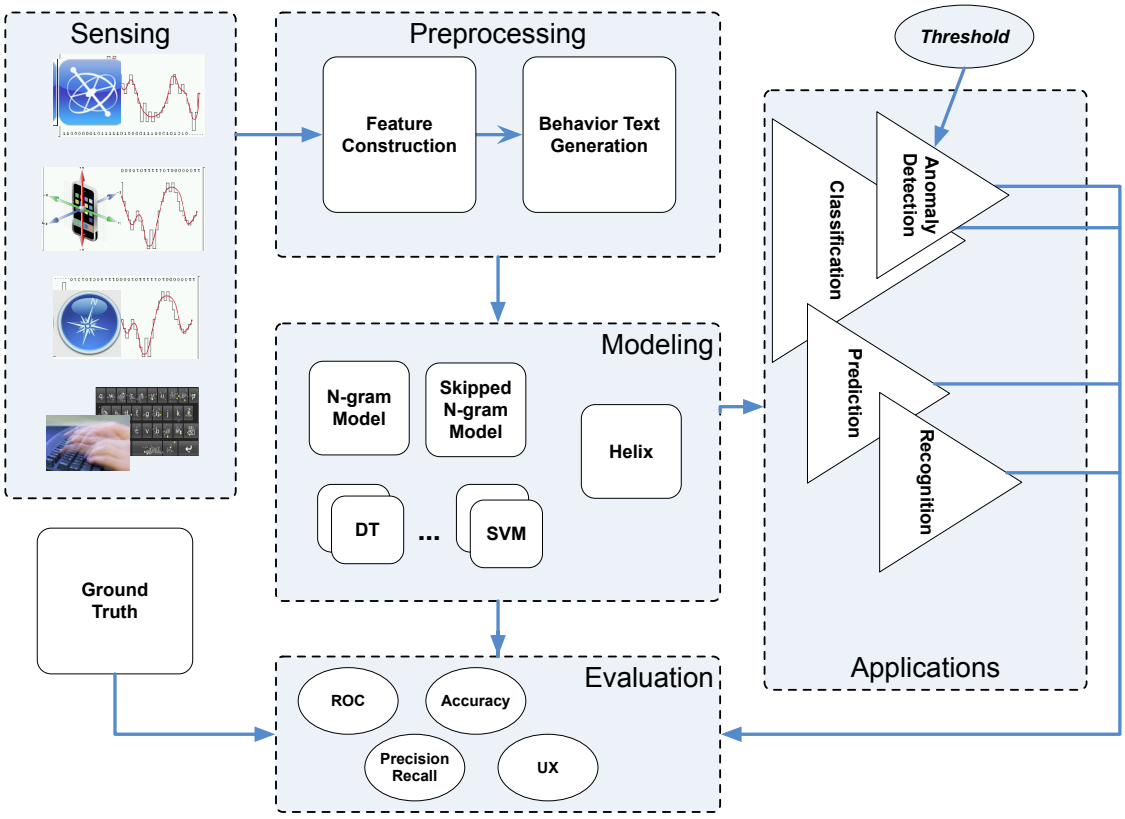

Figure 3.1: Behaviometric Framework Diagram

(1) Sensing: in order to build a behavior model and eventually a behavior-aware application,

we need to first collect necessary data as input to the framework. The data may come from hardware sensors embedded in smart devices, for example, GPS, gyroscope, etc., or from software running status and device input/output interface, as illustrated in Figure 3.1.

(2) Preprocessing: This component preprocesses the raw sensor data (or human-device interactions) from various sensor inputs. For example, the data may be discretized and quantized into time-series for later processing. Selecting the right set of features is important for improving the target system performance and reducing overhead. Additionally, features may also be sensitive to modeling tools. Therefore, we need to carefully construct features to use and convert the data to behavior text if necessary.

(3) Modeling: behavior modeling is the most challenging part. Given the constructed feature set, various modeling tools may be applied; some can be superior to the others. In this thesis, language-based behavior modeling method is extensively investigated. Compared with legacy learning algorithms, e.g., neural network, etc., it has better performance and lower complexity for some features, with time-series involved. We also proposed Helix, a novel approach to induce underlying grammar of human activities for macro-level behavior modeling.

(4) Applications: There are many use scenarios that can benefit from Behaviometric study, including enterprise IT, mobile security, etc. Typical use-cases include anomaly detection, user classification, user behavior prediction and regression.

(5) Evaluation: With ground truth data and behavior learned with suitable models, we will be able to evaluate the performance of each application.

The first stage, 'Sensing', is the prerequisite of the following stages of Behaviometrics. Without sensory data collected from user devices, it is impossible to model user behavior and consequently, impossible to realize applications such as user classification, recognition, anomaly detection, etc. The actual modeling involves data preprocessing and model development, i.e., stages 2 and 3. The

applications will also be highly tied to the specific model developed.

3.2 SENSOR DATA AND CONTEXTUAL METRICS

Modern mobile devices come with various mobile sensors such as accelerometer, gyroscope, GPS receiver, WiFi receiver, etc. We consider contextual information, i.e., contextual metrics being collected from these "hardware sensors" when building the Behaviometrics of a mobile user. Additionally, we also consider including the contexts from "virtual sensors" such as information obtained from the operating system like which application is being active or how much network traffic an application is transmitting or receiving, etc. In the following, we discuss various widely utilized sensors, how to retrieve sensor data systematically and what contextual metrics we can derive from these sensor data.

3.2.1 Sensors

3.2.1.1 Accelerometer

An accelerometer is a device that measures proper acceleration experienced by an object relative to a free-falling frame of reference. A triaxial accelerometer installed on a mobile or wearable device returns a real-valued estimate of acceleration along the x, y, and z axes (as shown in Figure 3.2[1]) in units of meter per second squared (m/s^2).

By measuring the amount of static acceleration due to gravity, it is possible to find out the angle the device is tilted at with respect to the earth. By sensing the amount of dynamic acceleration, we can analyze the way the device is moving and eventually infer the bearer's movement. Because of the information that the accelerometer can offer, it can be employed as a high-bandwidth side channel to learn certain behavioral patterns, e.g., user tap and gesture-based input pattern [30].

[1] `http://s.techtunes.com.bd`

Figure 3.2: Illustration of the coordinate system of an accelerometer on a mobile phone.

3.2.1.2 Gyroscope

A gyroscope is a device for measuring or maintaining orientation, based on the principles of angular momentum. Also known as angular rate sensors or angular velocity sensors, gyroscopes can sense the angular velocity along the x, y, and z axes of a mobile or wearable device, corresponding to pitch, roll and yaw respectively (as shown in Figure 3.3 [2]), in units of radian per second (rad/s).

The introduction of gyroscopes into the mobile devices has allowed for more accurate recognition of movements within 3D space than lone-accelerometer devices. That is why modern smartphones are usually equipped with both accelerometer and gyroscope, for example, HTC, Nexus, iPhone, Nokia, etc. Gyroscopes plays a significant role in the gaming arena by providing a richer experience in handling the game controls than a smartphone without it. It can also improve the accuracy of behavior model with better movement recognition data.

[2] `http://b2b.cbsimg.net`

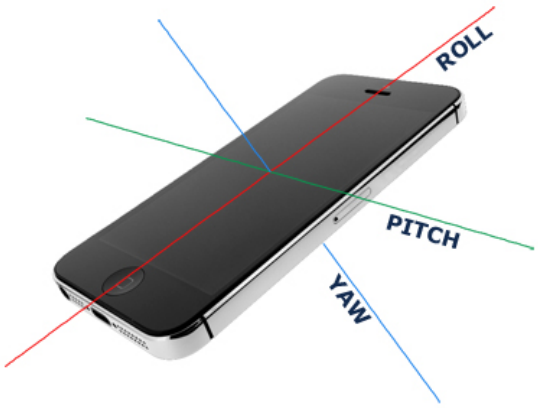

Figure 3.3: Illustration of the coordinate system of a gyroscope on a mobile phone

3.2.1.3 Location Sensors

Location sensors detect the location of the mobile device using either GPS, Lateration/Triangulation of cell towers or wifi networks (with a database of known locations for towers and networks), Location of associated cell tower or wifi network. Typical GPS precision is 20-50 meters with maximum precision of 10 meters. However, cell towers or wifi is preferred over GPS in urban or indoor environments due to its availability and higher precision.

These location sensors play an essential role for Location-based services (LBS). By using these sensors, applications on the smart devices can customize information and services for the user based on the location, surroundings and more. For example, with a location sensor, you may be able to find a nearby restaurant, get directions to that restaurant, send the directions to a friend, and then follow the directions on a map as you travel to your destination. The location sensor data provides a macro-level view of human activity trajectory. By properly using that, we can enable a variety of novel applications.

3.2.1.4 Virtual Sensors

Other than the "hardware" sensors, there are a variety of "channels" to collect other side information from the smart devices, referred as "virtual" sensors. One obvious medium is the device I/O,

for example, keyboard, mouse or touch screen. Based on the collected key press and/or mouse click pattern data, one is able to develop a keystroke dynamics model for different applications, through analyzing information about typing latencies, cadences, or error rates.

Mobile smart devices normally come with a touch screen and some form of soft keyboard. Soft keyboards are a relatively new form of input method editor, as shown in Fig. 3.4. While typing, users have a variety of different typing rhythms and the physiological traits of their hands, joints and fingertips ensure that no two users type exactly alike. Alongside traditional desktop-like features such as experience level and posture, soft-keyboard typing seems to be influenced by hand size, finger length, fingertip size, muscle development, posture and position, one-handed or two-handed typing, orientation of the phone, user tiredness, coldness of fingers, focus of the user (mobile users are often partially engaged in other activities), and whether the user is walking, standing, sitting, lying down, or riding in a car, bus or subway. Additional influences are size/shape of the phone, screen, and external phone cases.

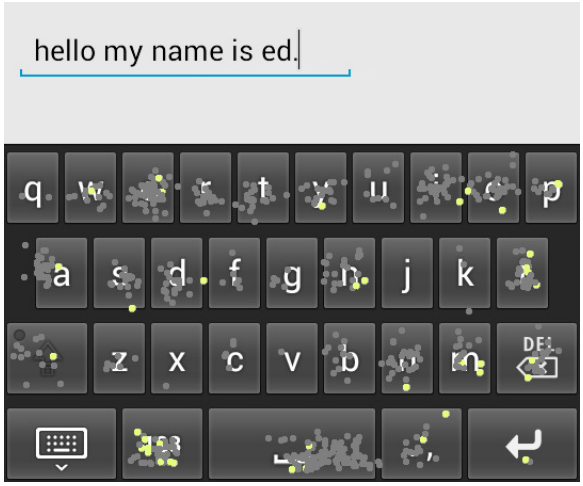

Figure 3.4: Illustration of the touch screen and soft keyboard on a mobile phone

Another category of virtual sensors is referred to as “soft” sensors, meaning they are not physically present but can potentially provide useful information. Network packet traces are a typical example in this category. It is possible to extract and analyze network packet headers and on-off

patterns for desired applications. Additionally, we can also track how human interacts with running applications through OS level message or applications API.

3.2.1.5 Reliability and Variability of Sensors

One important fact to note about both physical and virtual sensors is that devices of from different manufacturers or different models from the same manufacturer may have different reliabilities. And for some sensors the scale and sensibility may be different yielding variability in sensor readings. For example, accelerometers, gyroscopes and GPS sensors are more calibrated, while the WiFI radios, touch screen may need further calibrations to produce reliable and comparable readings from different devices.

3.2.2 Data Acquisition

On a mobile device, the sensory data may be collected through different means. For the aforementioned sensory data, we can collect them through /emphPlatform API, /emphSyslog and /emphDaemon Services.

3.2.2.1 Platform API

The proliferation of sensors in modern mobile devices has driven the evolution of platforms with necessary drivers and exposed certain APIs for developers to access the sensory data. Modern operating systems targeted for portable devices, e.g., iOS, Android, come with a rich set of APIs with supports a broad categories of sensors, including motion sensors, environmental sensors, and position sensors as mentioned above. Android, for example, offers an Android sensor framework [3], providing several classes and interfaces that help developers perform a wide variety of sensor-related tasks:

- Determine which sensors are available on a device.

[3] `http://developer.android.com/guide/topics/sensors/sensors_overview.html`

- Determine an individual sensor's capabilities, such as its maximum range, manufacturer, power requirements, and resolution.
- Acquire raw sensor data and define the minimum rate at which you acquire sensor data.
- Register and unregister sensor event listeners that monitor sensor changes.

3.2.2.2 Syslog

Syslog is a standard for computer message logging. It permits separation of the software that generates messages from the system that stores them and the software that reports and analyzes them. Within syslog, messages are labeled with a facility code (one of: auth, authpriv, daemon, cron, ftp, lpr, kern, mail, news, syslog, user, uucp, local0 ... local7) indicating the type of software that generated the messages, and are assigned a severity (one of: Emergency, Alert, Critical, Error, Warning, Notice, Info, Debug). Specific configuration may permit directing messages to various devices (console), files (/var/log/) or remote syslog servers. It is also possible to filter and display matching syslog messages only. Syslog also defines a client/server protocol. A logging application can transmit a maximum 1024 byte text message to the syslog receiver, which is commonly referred as syslogd, or syslog daemon.

3.2.2.3 Daemon Service

Daemon, by definition, is a computer program that runs as a background process, rather than being under the direct control of an interactive user. Therefore, a daemon service will have close to none negative impact on a user's experience at using the devices. A daemon service can also assist in gathering non-skewed data from virtual sensors as the users may not notice such processes running in the background. Depending on a specific requirement, a daemon service can perform certain actions at predefined times or in response to certain events, for example, sending out an alert message or logging the status and shutting down the device if an anomaly is detected. Normally, a daemon service is very light-weight by consuming little system resources. Thus, we can develop

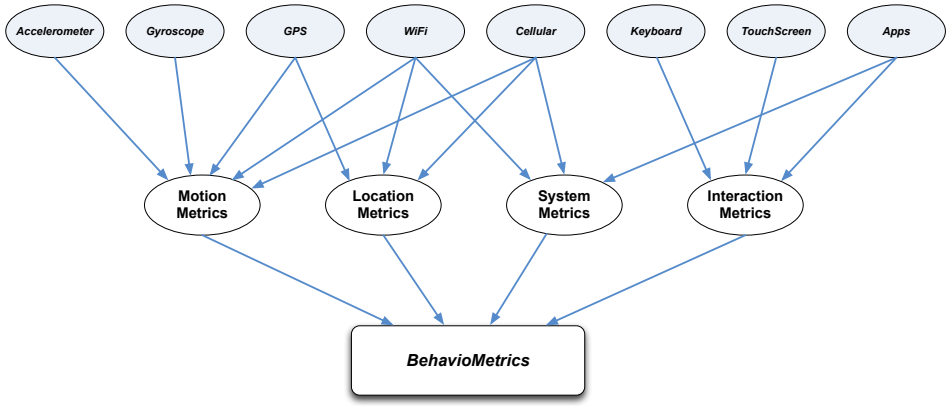

Figure 3.5: An illustration of contextual metrics

data acquisition daemons using supported platform APIs.

3.2.3 Contextual Metrics

Depending on specific application requirements and resources constraints, e.g., available sensors, power constraints, etc., we can choose a set of sensory data and apply machine learning techniques to them. As a first and generic step, we organize them into four categories, based on the characteristics of these physical and virtual contexts. As shown in Fig. 3.5, we proposed four contextual metrics: Motion Metrics, Location Metrics, Interaction Metrics and System Metrics. Combined, these four metrics provide a wide spectrum towards generalized Behaviometrics.

Motion Metrics describe physical aspects of individuals. For example, when a group of users perform the same sequence picking up a mobile phone and starting an application, the actual data readings recorded through the accelerometer will normally differ from person to person, from which we can learn a specific motion model for each person. One application from such a model is that by comparing a sequence of motion sensors readings, we can infer whether they come from the authorized user or not.

Oftentimes, motion metrics is not appropriate or efficient for certain applications due to the limited scope of physical aspects of individuals. By studying *Location Metrics*, also examined by

Hayashi et al. [14], we can enable or augment other mobile security services. From the location sensors of mobile devices, we will be able to construct a mobility pattern that is routinely followed by users. Later, when a new event is detected and it is not likely due to the authorized user based on his mobility model, certain security measures will be evoked to protect privacy and information integrity. Location Metrics will also be very useful to augment solely motion metrics based solutions, by providing additional features for the learning process.

Interaction Metrics, unlike the previous two metrics that are normally time-series data, how a user may interact with his/her devices, may generally not be correlated from one action to the next. For example, a user can interact with a keyboard or touch screen of a device. If we model the typing sequence as time-series, it can be very ineffective. However, we can apply data analysis in the spatial domain to model a user's interaction pattern. In the touch screen example, the finger pressure, press time, key press location, etc., will differ from user to user. Thus, these features can be efficiently used for pattern recognition.

System Metrics, reflect how system resources are used on the mobile devices. They are largely tied to the frequent application invocations by the mobile users and the interactions between the systems, the applications and the users. These metrics can be derived from system specific sensory and trace data, including network IO statistics, power consumption, system and user process statistics and memory usage, etc. These are indirect reflections on the mobile users' preferences and usage behaviors.

Behaviometrics is a generalization of those contextual metrics. They can be modeled alone as discussed above or selectively fused together in appropriate way towards an overall Behaviometric model. We will cover more details in Chapter 4 and Chapter 5.

3.3 SENSEC, SECURE MOBILE DATA AND APPLICATION THROUGH BEHAVIOMETRICS

Based on the Behaviometric framework and the outline of factors which influence a user's unique Behaviometric profile, we realize a mobile application, the SenSec App. SenSec can passively secure mobile data and applications through data sensing, model learning and anomaly detection.

3.3.1 Mobile Security: Motivation and Background

While mobile devices such as smartphones make our lives convenient in ways that were unimaginable before, applications such as email, web browsing, social network, shopping and online banking know too much about our private lives. Mobility introduces additional security and privacy challenges in being able to provide services in a way that neither compromises the environment of users nor their data.

Recently, a new survey [4] has revealed that 36 percent of consumers in the United States have either lost their mobile phone or had it stolen. Another survey [5] has also revealed that 329 organizations polled had collectively lost more than 86,000 devices with an average cost of *lost data* at $49,246 per device, worth $2.1 billion or $6.4 million per organization. Given the high loss rate and high cost associated with these losses, accountable schemes are needed to protect the data on the mobile devices.

Reliable and convenient authentication is an essential requirement for a mobile device and its applications. Today, passwords are the most common form of authentication. This results in two potential problems. First, passwords are major sources of security vulnerabilities, as they are often easily guessed, re-used, often forgotten, often shared with others, and are susceptible to social engineering attacks. Secondly, to secure the data and applications on a mobile device, the mobile

[4] Strategy One survey conducted among a U.S. sample of 3,017 adults age 18 years or older on September 21-28, 2010, with an oversample in the top 20 cities (based on population)

[5] "The Billion Dollar Lost-Laptop Study" conducted by Intel Corporation and the Ponemon Institute, analyzed the scope and circumstances of missing laptop PCs

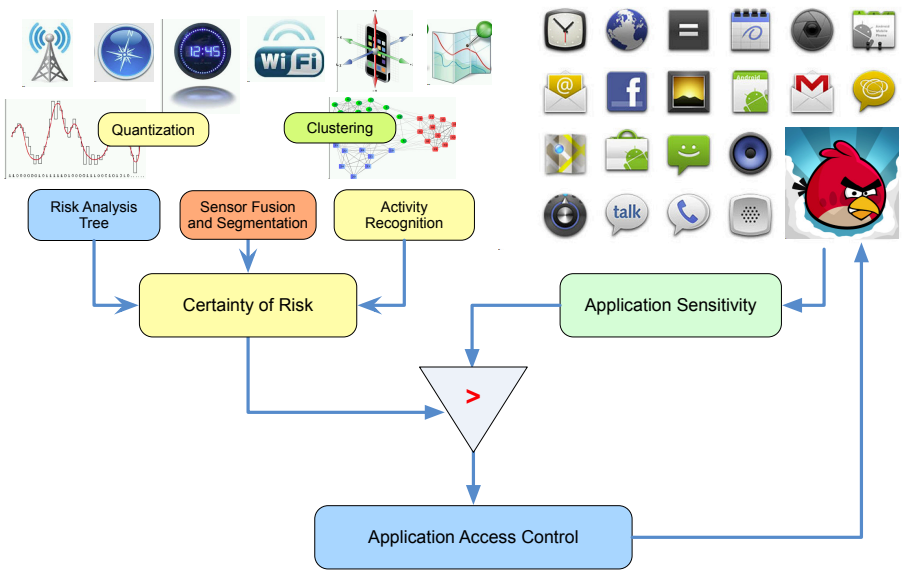

Figure 3.6: SenSec Overview: A Motivating Example.

system would prompt the user for authentication quite often and this results in serious usability issues. Therefore, protecting a user's privacy, data integrity and ensuring the accountability of mobile applications in seamless and non-intrusive ways poses great interests and values to next generation mobile computing platforms.

3.3.2 SenSec Architecture

Mobile BehavioMetrics and its various techniques create new opportunities for simplifying and strengthening mobile device and application security. Our *SenSec* application collects data from accelerometer, gyroscope, GPS, WiFi radio and other sensors and builds its owner's behavior model. Once the model is built, *SenSec* also continuously evaluates the *sureness* of whether the mobile device is under the control of the owner. This information can be used subsequently by on-device security sub-systems to control the access to resources, services and data as shown in Figure 3.6.

SenSec, which uses passive sensory data to ensure mobile device and application security running in the background, provides an additional layer of security mechanism on top of existing au-

thentication methods. A typical use scenario of *SenSec* would be like this: You are at a party and mistakenly leave your phone unattended. The phone is picked up by one of your friends. Out of curiosity, he/she starts to browse your applications and data. You certainly would not mind if she only plays a mobile game, but would feel extremely uncomfortable if she starts to view your contact list and uses your mobile banking apps. Because the gesture patterns are so different between you two, the *SenSec* system would detect this incident and automatically trigger further authentication, if any *sensitive* (defined by the user) applications are invoked.

3.3.3 SenSec Prototype

We built a functional *SenSec* Android mobile application, which keeps track of sensory readings from the accelerometer, gyroscope and magnetometer, builds the context of a user's gesture/motion pattern and uses that to verify his or her identity. It also collects data streams from GPS, WiFi radios, touch screen and software keyboard to build statistical models on user's location and interaction behaviors. We further fuse these location metrics and interaction metrics with gesture and motion factors to archive the same level of mobile security with better user experience. Table 3.1 shows a list of contextual metrics of our interests organized into four categories, each associated with some examples. In the sequel, we will discuss how to model these contextual metrics and how we design and carry out the experiments and their performance evaluation in detail.

Type	Example Context	Description of Certainty
Motion Metrics	Touchscreen and Mouse Movement	One's unique of using the touch screen or mouse to operate certain functions of the device.
	Walking Gait	One's personal walking style, based on accelerometer data.
	Device Tilt and Holding Pattern	One's personal device angle based on accelerometer and gyroscope data.
Location Metrics	WiFi MAC Address	Known WiFi environment
	Cellular Tower ID	Known Cellular environment
	Outdoor Mobility Patterns	The movement pattern of the mobile devices based on GPS-trace.
	Indoor Mobility Patterns	Movement patterns and indoor places users usually go to, via WiFi RSS and inertial navigation.
	Daily Activity Routines	When and where a user does what on a daily basis.
Interaction Metrics	Typing Pattern	One's unique pattern of typing on the keyboard on the mobile device based on keystrokes.
	Touchpad swiping	A user's swiping pattern based on pressure, length, and speed.
System Metrics	Application Usage	When and where a user uses a certain application on the mobile device.
	Device power consumption	Whether the device is idling or active
	Application Network Behavior	Frequency of data packets sent by an App.

Table 3.1: Contextual Metrics considered in SenSec.

CHAPTER 4

BUILDING BEHAVIOMETRICS: MODELING AND ANALYSIS

Behavior modeling and analysis is the most critical phase of our methodology (see Figure 3.1) and is concerned with the application of mathematically derived principles to Mobile BehavioMetrics for the purposes of characterization, classification, prediction and anomaly detection in mobile user and group behaviors.

In this chapter, we define our BehavioMetric models and outline the methods used to analyze and interpret these models in a quantitative manner. We start by proposing a novel language approach that converts multi-dimensional sensory data into behavioral text representation and utilizes NLP techniques to perform modeling and inference tasks. We then illustrate a hierarchical scheme, Helix, to segment and classify sensory time-series. This novel language-based approach is then augmented in our SenSec framework with traditional time-series based methods. Finally, we tackle the case where time-series analysis is not particularly useful with spatial analysis techniques. For each of the modeling techniques developed, we also show their application to various tasks, e.g., classification, identification, authentication, etc.

Natural Language	Behavior Language	Example
Word	Atomic Movement	Device tilt and movement
Phrase	Movement	Picking up the phone
Sentence	Action	Making a phone call
Paragraph	Activity	Search for the nearest Pizza restaurant and place a phone order
Document	Event	Prepare for a lunch meeting

Table 4.1: Behavior as language at different levels.

4.1 BEHAVIOMETRICS LANGUAGE MODELING

4.1.1 Motivations

The similarity between human behavior and language had been articulated by Burke [31] and Wertsh [32]. Based on the "principle of language as action", natural language and human action are really the same thing. They are both "mediational means" or tools by which we achieve our ends. They exhibit structure and satisfy "grammars". Table 4.1 illustrates that ambulatory behavior shares a lot in common with natural languages at all levels. Atomic movements form the vocabulary of the behavior language. A sequence of atomic movements performed in meaningful order creates a *movement* such as an *action* of "picking up the device". *Actions* such as "making a phone call" are created by performing actions in a right order similar to create a "sentence". A sequence of actions builds up an *activity*. Higher level behavioral concept *event* is composed of a series of activities in a similar way as a *document*.

To empirically evaluate the similarity between ambulatory behavior and natural language, we check if the behavior language corpus, a corpus composed of words mapped from atomic behaviors, follows the Zipf's law. Zipf's law states that given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table. In other words, the log frequency of a word is linear to its rank in a natural language corpus. Figure 4.1 plots the log frequency of word types in the behavior language corpus for word type ranked 1, 2, 3, etc. Though not exactly linear, it does plot a line similar to Zipf's distribution. The implication of such analogous

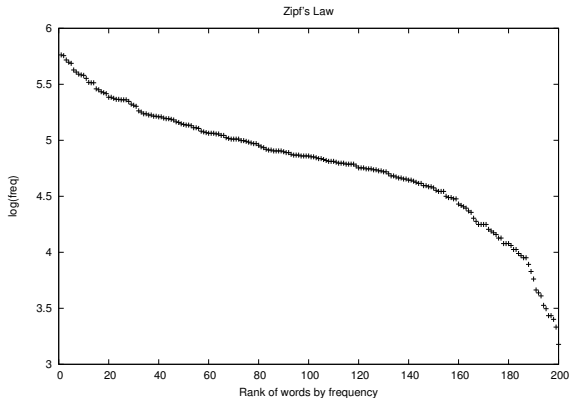

Figure 4.1: Log(freq) vs. rank of frequency of word types in behavior language corpus.

is that we may reuse the rich collection of algorithms developed by NLP researchers to model the time-series as a sequence and a structure problem rather than developing algorithms from scratch.

4.1.2 Behavioral Text Representation and Language Model

Shannon in [33] established that a language could be approximated by an n-th order Markov model (n-gram). Using an n-gram model trained on the English text, we can estimate whether "United" or "house" is more likely to follow the phrase "the president of the" by comparing the probability P("United" | "the president of the") and P("house" | "the president of the"). Such an n-gram model has also been proven to be very robust in modeling sequences of data other than language. Similar to [34–37], in our work, we adopt a scheme by converting real-valued sensor readings to *symbols* through quantization, vectorization or clustering and using n-gram models to capture the patterns in a user's behaviors in various scales, macro locations, micro locations, motion gestures, activities and tasks, etc.

Data produced by sensors usually are multi-dimensional and real-valued. This may prevent us from using them directly in our language approach due to computational complexity. To overcome this, we first segment the raw sensor readings into a series of chucks of fixed size in time. For each chunk, we then construct a set of features and build feature vector for that time slot. After that, these

feature vectors are clustered into V classes using K-means algorithm. The centroids of the resulting V classes form the vocabulary **V**. Thus, the sequence of feature vectors can be mapped to a series of class labels $\{l_0, l_1, l_2, \ldots, l_n, \ldots\}$, where $l_i \in \mathbf{V}$. This forms the behavior labels for the n-gram processing, which will be covered in next section.

In a nutshell, we convert the raw sensory data into ***behavior text representation*** as sequences of behavior labels. Each behavior label is considered as a "word" in the language. We then train a ***language model*** on those traces and use the trained model for various applications.

4.1.3 Behavioral n-gram Model

An n-gram model is an n-order Markov model for predicting the next item in a sequence. The two core advantages of n-gram models are 1) relative simplicity and 2) the ability to scale up. Moreover, n-gram model is a generative model. Compared with discriminative models such as SVM, it also enables us to perform user authentication task with only positive training samples collected from the owner.

In our context, the n-gram model estimates the probability of the next behavior label l_i given the previous $n-1$ behavior labels from the traces denoted as in Equation 4.1.

$$P(l_i|l_{i-n+1}^{i-1}) = P(l_i|l_{i-n+1}, l_{i-n+2}, \ldots, l_{i-1}) \tag{4.1}$$

The model probabilities $P(l_i|l_{i-n+1}^{i-1})$ can be estimated through the Maximum Likelihood Estimation (MLE) from the training data by counting the occurrences of behavioral text labels:

$$P_{\text{MLE}}(l_i|l_{i-n+1}^{i-1}) = \frac{C(l_{i-n+1}, \ldots, l_{i-1}, l_i)}{C(l_{i-n+1}, \ldots, l_{i-1})} \tag{4.2}$$

MLE assigns probability zero to any unseen n-grams if a data set contains n-grams that have never occurred in the training data. To address this issue, Good-Turing discounting and Katz backoff smoothing [38] can be applied to discount the MLE probability for each observed n-grams in the training data and reserve some probability mass for unseen events.

Similar to the n-gram language model, our n-gram model is based on the Markovian assumption that a user's future behavior solely dependent on his/her previous $n-1$ behaviors.

4.1.4 Prediction and Skipped n-gram model

The aforementioned continuous n-gram model can be used to predict user's next behavior given adjcent previous behaviors as given in Equation 4.1.

In natural language, words in a sentence may have *long-distance dependencies*. For example, the sentence "I hit the tennis ball" has three word level tri-grams: "I hit the", "hit the tennis" and "the tennis ball". However, it is clear that an equally important tri-gram implied by this sentence, "hit the ball", is not normally captured because of the separator "tennis". If we skip the word "tennis", we can form this important trigram. Similarly, our continuous n-gram model assumes that a user's next action is dependent solely on his/her previous $n-1$ actions. However, in many cases one's future mobility behavior depends on behaviors that happened a while ago, while the intermediate behaviors have little relevance or influence on the present and future behaviors. For example, knowing that a user is leaving the break room and entering the hallway which leads to his office, we can predict that he will be in his office soon. In other words, his intermediate actions along the hallway and his actions right before entering the office are not that important once we know that he is leaving the break room.

To model such long-distance dependencies, we can train a skipped n-gram model. A skipped n-gram is a pair (L, l) extracted from the behavior text. L is a label sequence of $n-1$ labels and l is the label that after skipping d labels after L (Figure. 4.2). Usually L and l have strong correlations. In other words, the occurrence of L is *associated with* the occurrence of l in the future.

4.1.5 Similarity of Behaviometrics

Each sentence is vectorized using the frequencies of all its n-grams. Each dimension of the vector represents one n-gram instance. For example, consider an activity sentence P ("NB NB P P

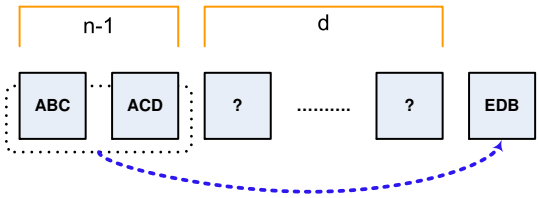

Figure 4.2: Skipped n-gram to model long-term dependent mobility behaviors

P P P P NB NB") and another Q ("NB P P NB NB P NB P P P"). The vectors for sentence P and Q are constructed as

$$V_P = (4, 6, 2, 1, 5, 1, 1, 1, 1, 1, 4, 1, 1, 1, 1, 3, 1, 1, 1, 1, 2)$$

$$V_Q = (4, 6, 1, 3, 3, 2, 1, 2, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0)$$

Each dimension of the these vectors is representing one of the instances of the n-gram column of Table 4.2. Note that if a particular n-gram does not appear in a sentence, the value of that dimension would be zero. e.g. n-gram ("NB NB P P P") in sentence Q.

Table 4.2 shows an example of calculating the similarity between activity sentence P ("NB NB P P P P P P NB NB") and Q ("NB P P NB NB P NB P P P").

The centroid of cluster is constructed by taking the the ***centroid*** of all the vectors in the given cluster.

In natural language, it is a common task to compare multiple documents for similarity with the applications in document classification, forgery detection, etc. Similarly, in our context, when BehavioMetric models are built from sequences of sensory data, similarity between two BehavioMetric models can be calculated by the distance of their corresponding behavior text strings.

Inspired by the BLEU metric where averaged n-gram precision is used to measure the similarity between a machine translation hypothesis and human generated reference translations, we use *averaged n-gram precision* to estimate the similarity between two lifelog segments.

Assuming that P and Q are two activity language sentences of the same length l. P is the sequence of P_1, P_2, ..., P_L and Q is the sequence of Q_1, Q_2, ..., Q_L. Denote the *similarity*

n	n-gram	$freq_P$	$freq_Q$	min	Prec$_n$
1	NB	4	4	4	
	P	6	6	6	
					10/10=1.0
2	NB NB	2	1	1	
	NB P	1	3	1	
	P P	5	3	3	
	P NB	1	2	1	
					6/9 = 0.67
3	NB NB P	1	1	1	
	NB P P	1	2	1	
	P NB NB	1	1	1	
	P P NB	1	1	1	
	P P P	4	1	1	
					5/8 = 0.63
4	NB NB P P	1	0	0	
	NB P P P	1	1	1	
	P P NB NB	1	1	1	
	P P P NB	1	0	0	
	P P P P	3	0	0	
					2/7 = 0.29
5	NB NB P P P	1	0	0	
	NB P P P P	1	0	0	
	P P P NB NB	1	0	0	
	P P P P NB	1	0	0	
	P P P P P	2	0	0	
					0/6 = 0.0
					$S(P, Q) = 0.52$

Table 4.2: Calculating the similarity between two activity sentences using averaged n-gram precision.

between P and Q as $S(P,Q)$. Define the n-gram precision between P and Q as $\text{Prec}_n(P,Q) =$

$$\frac{\sum_{\tilde{p}\in\{\text{All } n\text{-gram types in } P\}} \min(freq(\tilde{p},P), freq(\tilde{p},Q))}{\sum_{\tilde{p}\in\{\text{All } n\text{-gram types in } P\}} freq(\tilde{p},P)}, \tag{4.3}$$

and the similarity between P and Q is defined as:

$$S(P,Q) = \frac{1}{N}\sum_{n=1}^{N} \text{Prec}_n(P,Q) \tag{4.4}$$

$\text{Prec}_n(P,Q)$ calculates the percentage of n-grams in P that can also be found in Q and $S(P,Q)$ averages the precision over N-gram of all N orders.

4.1.6 Classification and Recognition

We can build n-gram models for M different users or user groups (or classes). When they perform the same tasks that cause similar motions and postures. Additionally, we can group traces from multiple users that belongs to the same class, being gender, occupation or age group, and train a n-gram model for this class. Without losing generality, let's denote **V** as the vocabulary and **G** as the set of all n-grams from such a behavioral text data set of M users, the models of these M classes can be described as probability vectors as $\overrightarrow{P}_0, \overrightarrow{P}_1, \overrightarrow{P}_2, \ldots, \overrightarrow{P}_{M-1}$ where

$$\overrightarrow{P}_m = \{P_m(l_i|l_{i-n+1}^{i-1}) | l_i \in \mathbf{V}, l_{i-n+1}^{i-1} \in \mathbf{G}\} \tag{4.5}$$

For a sequence of behavioral labels of size N, $L = \{l_1, l_2, \ldots, l_N\}$, we estimate the probability that L is generated by a given n-gram model $\overrightarrow{P}_m$ as

$$P(L,m) = P(l_1, l_2, \ldots, l_N, m) = \prod_{i=1}^{N} P_m(l_i|l_{i-n+1}^{i-1}) \tag{4.6}$$

or average log probability as

$$\frac{1}{N}\sum_{i=1}^{N} \log P_m(l_i|l_{i-n+1}^{i-1}) \tag{4.7}$$

where $P_m(\ldots)$ are the model probabilities of user m as in Equation 4.12.

Given a behavior text sequence L, the user classification problem can be formulated as

$$\hat{u} = \underset{m}{\text{argmax}}\, P(L,m) \tag{4.8}$$

where $P(L, m)$ is the probability the behavior text sequence L is generated by mth user's n-gram model as denoted in Equation 4.1. Therefore,

$$\hat{u} = \underset{m}{\operatorname{argmax}} \frac{1}{N} \sum_{i=1}^{N} \log P_m(l_i | l_{i-n+1}^{i-1}) \tag{4.9}$$

$$= \underset{m}{\operatorname{argmax}} \sum_{i=1}^{N} \log P_m(l_i | l_{i-n+1}^{i-1}) \tag{4.10}$$

4.1.7 Anomaly Detection

Once we constructed a model of a user's behaviometrics through learning, we can continue monitoring user's behaviometrics and compare them with the learned model. If the new behaviometrics deviate from the learned model, we may choose to trigger an anomaly alert. However, variations in sensory data streams could also be caused by noise and new behaviors in addition to anomalous behaviors. Variations caused by noise are less significant and can be smooth out statistically. On the other hand, to distinguish between anomalous and new behaviors, we need to evaluate if those unseen patterns can be incorporated into the model over time. Failing to identify such a distinction might yield false positive temporarily, but if certain feedback mechanisms are in place to correct those false positives, we are still able to build a robust anomaly detection system in various application domains such as theft detection and prevention, casual authentication, emergency detection and healthcare monitoring.

The anomaly detection problem can be formulated as binary classification problem, classifying a behavior text sequence as normal behavior ($\hat{a} = 1$) or not ($\hat{a} = -1$). We can model this scheme probabilistically as $\hat{a} = sign[P(u = 1|r) > \theta]$, i.e. given an observation r, evaluate the probability of a given sequence is generated from a learned model ($\hat{a} = 1$) and check to see if it exceeds a certain threshold θ.

$$\hat{a} = sign[P(u = 1|r) > \theta] \tag{4.11}$$

Similarly as shown in Equation 4.5, a user u's behaviometric model can be expressed as a

probability vector as $\overrightarrow{P}_u$ where

$$\overrightarrow{P}_u = \{P_u(l_i|l_{i-n+1}^{i-1})|l_i \in \mathbf{V}, l_{i-n+1}^{i-1} \in \mathbf{G}\} \tag{4.12}$$

Then, given a sequence of behavior text L, and a sensitivity threshold θ, we want to validate if these sequence is generated by model of user u as

$$\hat{a}(L|u,\theta) = sign[P(L,u) > \theta)] \tag{4.13}$$

while $P(L,u) = P(l_1, l_2, \ldots, l_N, m) = \prod_{i=1}^{N} P_m(l_i|l_{i-n+1}^{i-1})$ as in Equation 4.1:

$$\hat{a}(L|u,\theta) = sign(\sum_{i=1}^{N} \log P_u(l_i|l_{i-n+1}^{i-1}) > \log\theta) \tag{4.14}$$

The sensitivity threshold θ is crucial towards the precision and recall of a given anomaly detection model. A suboptimal θ may lead to false positives or may cause anomalous behaviors to be undetected.

4.1.8 Behavioral Text Segmentation

Similar to natural language processing where the text corpus needs to be segmented to paragraphs, sentences and words in order to infer their meanings, behavioral textual representation of sensory data sequence also need to be segmented, so that we can group labels that belong to the same activity and build behaviometrics for that activity.

Unlike most of the natural languages, which have specific escape tokens to indicate the boundary of language entities, we relay on similarity of adjacent behavior textual representations to identify if a change of activity occurs. If we assume that when a user switches his/her activity at time t, there should be a significant change from behavior text string $[t-w, t-1]$ to string $[t, t+w]$. For a window size w, define the "change of activity" at time t as:

$$H(t,w) = -logS([t-w, t-1], [t, t+w-1]), \tag{4.15}$$

where $S(P,Q)$ (Eq. 5.1) measures the similarity between behavior text string P and Q. The higher

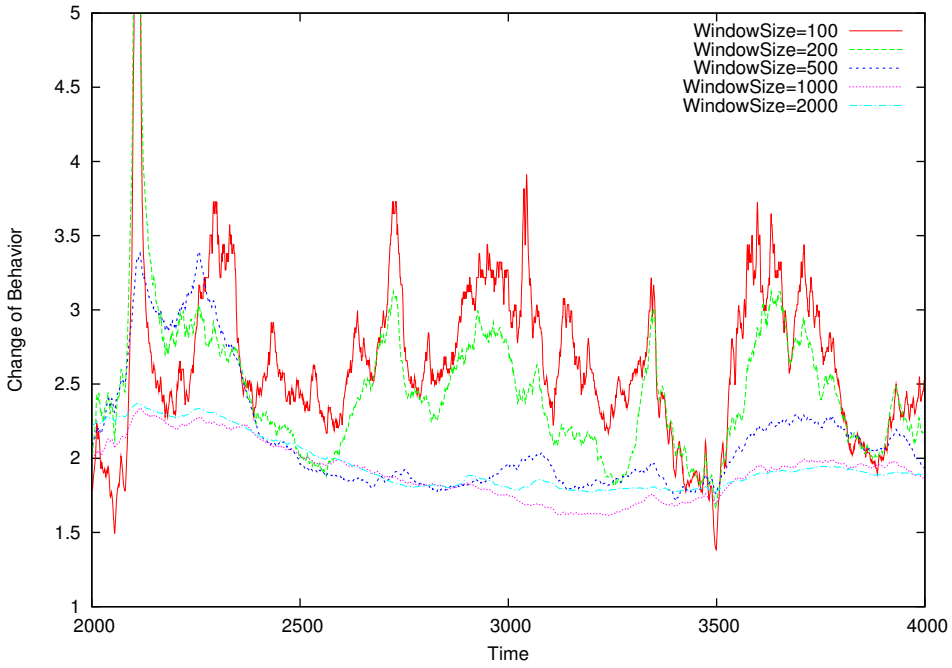

Figure 4.3: Activity changes calculated by different sizes of sliding windows.

the value of $H(t, w)$, the more likely is the user to have changed his/her activity at time t.

Figure 4.3 shows an example of H values at each data given different window sizes for a sequence of behavior text built from a sequence of accelerometer readings from a mobile device. Notice that: (1) peaks of activity change identified by larger windows are also peaks identified by smaller windows but not vice versa; and (2) activity changes over larger windows are smoother than smaller windows. Intuitively, larger window size captures changes of larger-scale activities whereas smaller window captures changes of smaller activities. Based on these, we can first segment the data using large window sizes and then recursively segment the data using smaller windows.

A smoothed Hidden Markov Model (HMM) can also be used to segment behavior textual representation. After specifying the topology of a Markov model, an HMM can be trained on unlabeled sequences of text through the Baum-Welch algorithm [39]. which estimates model parameters such as the probability of emitting a symbol from a certain state and the transition probability from one state to another. A trained HMM can then be used to "parse" a sequence of text and estimate the

most likely "states" sequence (e.g., "S1 S1 S1 S2 S2 S1 S1 S3 ...") that generates the observed text. The "state" is then used as the label for each observed symbols or in our case, the underlying activity for the observed sensor readings. When the state labels changes, we consider the underlying activity has changed and segment the data to reflect this change.

In our implementation, each state in the HMM emits single behavior text symbols. This leads to a problem where HMM segments the input data into too many activities. To smooth out these noise, we apply a sliding window of size $2w$ over the recognized state sequence. At time t, we use the dominant activity symbols within the window $[t - w, t + w]$ as the smoothed activity symbol for time t and segment the sequence to activities over the smoothed activity/state symbols.

The activity recognition approach proposed in [40] is implemented as the bootstrapping activity recognition algorithm in our system. In this approach, sensors readings will be clustered and quantized into a series of labels, called *activity text*. The activity text will be used as the training corpus to train a smoothed $n - gram$ language model for activity recognition.

This approach was selected based on its following capabilities: It can provide acceptable activity boundary segmentation in a cold boost and lead ways to similar activity retrieval, which can help users annotate activity automatically after getting sufficient labels. In addition, it provides a uniformed representation of heterogeneous sensor input, like accelerometer and GPS are converted into a single type of "activity language" after data preprocessing, which will facilitate the fusion of different sensors.

4.2 HELIX: HIERARCHICAL SEGMENTATION AND CLASSIFICATION

With illustrations in Figure 4.4, we first define the problem as follows: given d-dimensional sensor readings $S = \{S_1, \ldots, S_{|d|}\}$ with some underlying activities, we define the multi-level *Activity Text* as $A = \{A^1, \ldots, A^L\}$ of L levels, where $A^l = \{a^l_1, \ldots, a^l_{|A^l|}\}$ in which each a^l_i represents the i-th activity in level l. Here A^1 denotes lowest-level activities (e.g., "twisting the waist") while A^L denotes the highest (e.g., "playing a game"). Subject to A, we define an *Activity Grammar*

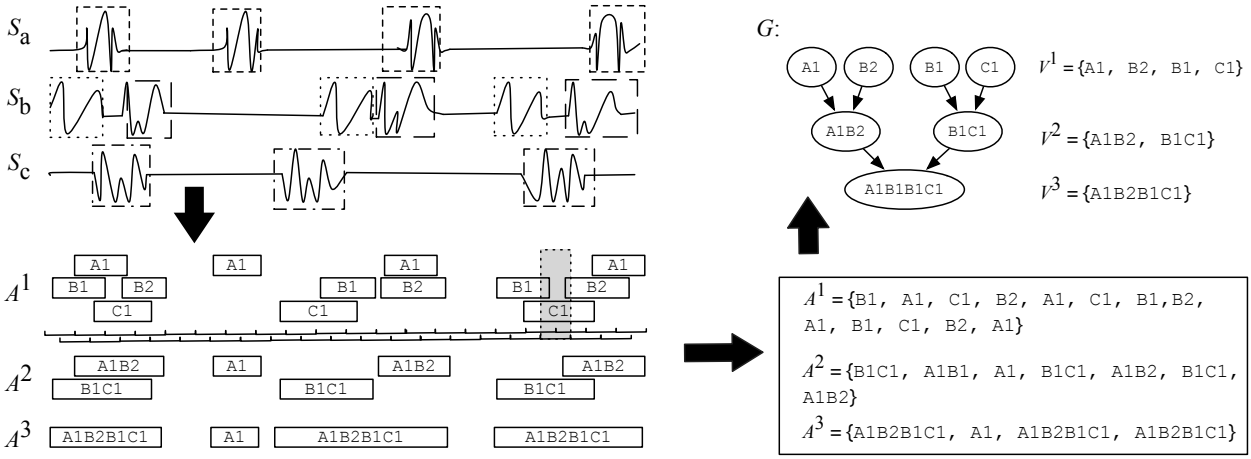

Figure 4.4: Grammar induction using Helix. A, V, and G denote the multi-level activities, vocabulary, and induced grammar, respectively.

$G = \{V, R\}$[1] , where V denotes a multi-level *Activity Vocabulary* of L levels and R denotes the *Relations* specifying the structure of the grammar. Specifically, $V = \{V^1, \ldots, V^L\}$, where V^l represents the activity vocabulary of the l-th semantic level, such that each l-th level activity $a_i^l \in \{V^1 \cup \cdots \cup V^l\}$. Accordingly, any directed relationship $(v_i^{l_1} \rightarrow v_j^{l_2}) \in R$ iff $l_2 > l_1$ and $v_j^{l_2}$ is a *super-activity* or a *generalized activity* [2] are $v_i^{l_1}$, that is, $v_i^{l_1}$ is implied to be performed as part of the activity $v_j^{l_2}$.

Definition. **Unsupervised Activity Structure Learning**. Subject to a set of unlabeled multi-dimensional sensor readings S generated by some activities, the *Unsupervised Activity Structure Learning Problem* is defined as finding a grammar $G = \{V, R\}$ that best describes the hierarchical activity structure embedded in S.

We then describe how to find such a grammar G using an unsupervised algorithm consisting of three main steps: (1) vocabulary initialization, (2) collocation discovery, and (3) vocabulary

[1] The grammar definition in the NLP community is typically of the 4-tuple form $G' = \{V', R', \Sigma, S\}$, where Σ is the set of *terminals*, V' is the set of *variables*, and S is the starting symbol. For simplicity, and with a slight abuse of notation, our grammar definition is of the 2-tuple form $G = \{V, R\}$ where $V^1 = \Sigma$, $V^L = S$, $R = R'$, and $V = \{\Sigma \cup V' \cup S\}$.
[2] A super activity or generalized activity is a combination of smaller sequenced activities. A intuitive example would be, as we illustrated in the "playing a game" example, "playing a game" is a super activity that consists of "twisting the waist" other smaller actions

generalization.

4.2.1 Vocabulary Initialization using Time-series Motifs

An established approach to build a vocabulary from time-series is by finding *time-series motifs* [41], the recurring patterns of similar subsequences. As in the top-left of Figure 4.4, we extend [41] to extract similar subsequences. They are then assigned labels to form the initial vocabulary. In the example, $V^1 = \{\texttt{A1}, \texttt{B1}, \texttt{B2}, \texttt{C1}\}$ and $A^1 = \{\texttt{B1}, \texttt{A1}, \texttt{C1}, \texttt{B2}, \texttt{C1}, \texttt{B1}, \texttt{B2}, \texttt{A1}, \texttt{B1}, \texttt{C1}, \texttt{B2}, \texttt{A1}\}$.

4.2.2 Super-Activity Discovery by Statistical Collocation

In Statistical Natural Language Processing (SNLP), *Collocation* [42] is the "expression of two or more words that correspond to some conventional way of saying things". In the context of super-activity discovery, collocation is the combination of two activities where their joint-occurrence frequency is significantly larger than what could be resulted randomly from their marginal frequencies.

To discover super-activities using statistical collocation, we first assume that the activity pairs occurring jointly within a window may be components of one super-activity, like an intertwined double helix. For example, the grey dotted box in Figure 4.4 includes 3 activity pairs ([B1,B2], [B1,C1], and [B2,C1]). Note that the order within the pair matters: two sub-activities from the same dimension are ordered by time ([B1,B2]), whereas two sub-activities from different dimensions are ordered by their dimension rank ([B2,C1]). The rationale is that for a super-activity, the ordering between its sub-activities in the same dimension usually matters (e.g., leaning forward then stand up), but not so much for those from different dimensions (doing an exam while feeling pressure).

By sliding the grey window along the current activity text (A^1), we accumulate the joint frequencies of all activity pairs and record them in a *Joint Frequency Table* (upper-left of Figure 4.5). The *Marginal Frequency Table* (upper-right of Figure 4.5) is also accumulated similarly. For example, the pair [B1,C1] will account for one [B1,−] and one [−,C1]. Although the content of the two tables depends on the length and the step size of the sliding window, we found that the collocation

Joint Freq. Table

w1	w2	Freq.
A1	B1	5
A1	B2	12
A1	C1	6
B1	B2	3
B1	C1	10
B2	B2	6

Marginal Freq. Table

w1	w2	Freq.
A1	-	23
B1	-	13
B2	-	6
-	B1	5
-	B2	15
-	C1	22

Contingency Table of [A1, B2]

	A1	¬ A1	
B2	12	11	23
¬ B2	3	16	19
	15	27	42

$$\chi^2 = \frac{42 \times (12 \times 16 - 11 \times 3)}{23 \times 19 \times 15 \times 27} = 5.99 \geq \alpha_{0.95} = 0.3841$$

Figure 4.5: Joint and marginal frequencies of activity pairs, with the contingency table of the pair (A1,B2).

discovery results are not sensitive to them, because as the size of a super-activity grows, the region within which other activities can co-occur with it also stretches. Empirically, setting the sliding window's step size to an atomic activity length (e.g., 0.5s) and the window size to a multiple of it (e.g., 5s) gives good results.

With the joint and marginal frequency tables, we test each activity pair for its statistical collocation significance. We do so by constructing the contingency table for each activity pair and calculating the χ^2 (chi-square) statistics [42]:

$$\chi^2 = \sum_{i,j} \frac{(O_{ij} - E_{ij})^2}{E_{ij}} \tag{4.16}$$

where O_{ij} denotes the (i, j)-th entry of the contingency table and E_{ij} denotes the expectation of the entry derived from their marginals. For [A1,B2] in the Figure 4.5, $\chi^2 = 5.99 \geq \chi^2_{0.05}(1) = 3.841$. Therefore, we can merge [A1,B2] to form a super-activity with 95% confidence level, here we use $\alpha_{TH} = \chi^2_{0.05}(1)$ as the threshold parameter controlling the merging strength. Repeating this step for three iterations, we obtain the grammar in the right-bottom of Figure 4.4.

4.2.3 Vocabulary Generalization

The collocation discovery works when there are repeating sequences of sub-activities. For real human activities, however, repetitions are unlikely to be exact. For example, when one is playing

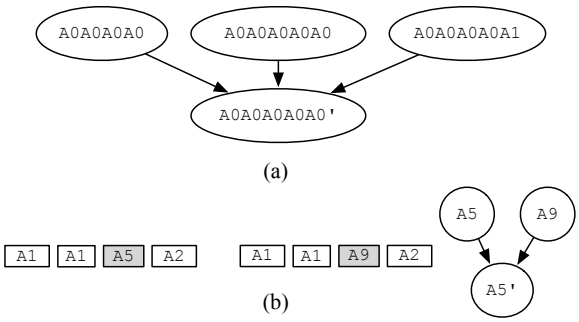

Figure 4.6: (a) Content-based and (b) context-based generalizations.

ping-pong, he may perform a cut, a lift, or a regular play, which can have various sequence combinations. In order to detect higher-level activities in such a condition, we need to *generalize* similar activities before constructing the joint and marginal frequency tables. Such a clustering is done according to two similarity measures: (1) Content Similarity (ϕ_e) and (2) Context Similarity (ϕ_x). We employ a clustering algorithm according to two similarity measures: (1) Content Similarity (ϕ_e) and (2) Context Similarity (ϕ_x). For content similarity, as shown in Figure 4.6(a), we we first calculate their n-gram precision and recall and then calculate the similarity score as .

$$\begin{aligned} Precision &= \frac{|nGram(v_1) \cap nGram(v_2)|}{|nGram(v_1)|} = 1 \\ Recall &= \frac{|nGram(v_1) \cap nGram(v_2)|}{|nGram(v_2)|} = 0.75 \\ \phi_e &= 2 \times \frac{Precision \times \sqrt{Recall}}{Precision + \sqrt{Recall}} = 0.93 \end{aligned} \tag{4.17}$$

The square root of the recall term is to award that case when two phrases are very similar in content but differs in length. For context similarity, we take into account of activities occurring within similar context, as shown in Figure 4.6(b), we count the number of times every activity co-occurs with the activity and then calculate the context similarity ϕ_x between two activities as the cosine distance between their context vectors:

$$\phi_x = \frac{\vec{c_1} \cdot \vec{c_2}}{|\vec{c_1}||\vec{c_2}|} \tag{4.18}$$

Finally, we aggregate the content and context similarities into one single similarity measure ϕ using arithmetic means and use ϕ in a complete-link clustering algorithm where a link exists between two activities only if their similarity ϕ is larger than a threshold ϕ_{TH}

4.2.4 Content similarity (ϕ_e)

As in Figure 4.6(a), the first generalization accounts for activities similar in content, including two main cases. The first case is when multiple activities are both repetitions of the same sub-activity, e.g., `A0A0A0A0`, and `A0A0A0A0A0` (left two nodes at the top). The second case is when two activities differ in only minor portion in their composition, e.g., `A0A0A0A0A0`, and `A0A0A0A0A1` (right two nodes).

To consider both of these cases of two similar phrases (activities) v_1 and v_2, we first calculate their n-gram precision and recall. W.l.o.g., we assume $|v_1| \leq |v_2|$ and $n = 2$. For the example in Figure 4.6(a), v_1 = `A0A0A0A0` and v_2 = `A0A0A0A0A1`, such that $nGram(A)$ consists of 3 `A0A0`'s and whereas $nGram(B)$ consists of 3 `A0A0`'s and 1 `A0A1`. Accordingly, we have:

$$\begin{aligned} Precision &= \frac{|nGram(v_1) \cap nGram(v_2)|}{|nGram(v_1)|} = 1 \\ Recall &= \frac{|nGram(v_1) \cap nGram(v_2)|}{|nGram(v_2)|} = 0.75 \\ \phi_e &= 2 \times \frac{Precision \times \sqrt{Recall}}{Precision + \sqrt{Recall}} = 0.93 \end{aligned} \tag{4.19}$$

The square root of the recall term is to award that case when two phrases are very similar in content but differs in length.

4.2.5 Context similarity (ϕ_x)

The second generalization accounts for activities occurring within similar context. For example, `A5`, and `A9` in Figure 4.6(b) are both preceded by 2 `A1`'s and followed by `A2`, suggesting that these two activities are likely to be semantically similar. We represent the context of an activity as $\vec{c} = (n_1, \ldots, n_{|V|})$, representing the number of times every activity co-occurs with the activity.

Then the context similarity ϕ_x between two activities is the cosine distance between their context vectors:

$$\phi_x = \frac{\vec{c_1} \cdot \vec{c_2}}{|\vec{c_1}||\vec{c_2}|} \tag{4.20}$$

Before clustering, we need to aggregate the content and context similarities into one single similarity measure ϕ. Intuitively, the arithmetic, geometric, and harmonic means are all reasonable candidates. However, in order to make each of ϕ_e and ϕ_x sufficient for generalization (instead of requiring both), we use the arithmetic mean. This aggregate ϕ is then used in a complete-link clustering algorithm where a link exists between two activities only if their similarity ϕ is larger than a threshold ϕ_{TH}. Consequently, all activities grouped together in a cluster are all highly similar to each other. Algorithm 4.2.1 summarizes the overall Helix algorithm.

Algorithm 4.2.1 Hierarchical Activity Structure Discovery

Input: $S = \{S_1, \ldots, S_d\}$: d-dimensional sensor readings
Input: α_{TH} : merging threshold parameter
Input: δ_{TH} : generalization threshold parameter
Output: $G = \{V, R\}$: discovered hierarchical grammar
Output: A : hierarchical activities labeled using V

(A^1, V^1) = initialize the vocabulary by motif discovery
$(A', V') = (A^1, V^1)$
$l = 1$
while TRUE **do**
  $l = l + 1$
  (A^l, V^l) = discover collocations from (A', V')
  break if $|V^l|$ == 0
  for all $v_i \in V^l$ **do**
    add edges (v_i, v_j) into R for all collocations
  end for
  (A', V') = generalize the vocabulary from (A^l, V^l)
end while
$V = \{V^1, \ldots, V^{l-1}\}$
return $G = \{V, R\}$, A

4.3 MODELING PERSONAL FACTORS FROM MOTION SENSOR DATA

In our language as action study, we are training a single model for all types of activities. Depending on how we calculate the feature vector for each fixed size window when building our vocabulary, we may end up with elements in **V** such that two conceptually different actions map to the same element. In this way, there may be the possibility that two separate people perform a different sequence of actions but they get mapped to a similar behavioral 'sentence'. While HELIX works well with the language as action principle, it is more of a general approach for activity classification. In this section, we perform a separate study which focuses specifically on motion sensor data to develop an approach to behavioral modeling. By doing this, we are able to exploit the structure of motion data and build a different model for each individual class. This allows for the models to learn the subtleties in the particular way each user performs actions from each activity class. For example, if we can know that a set of data comes exclusively from people walking, then we can infer from this data distinctive qualities of each walk. However, without knowing this, we may unwittingly compare a walk to a run which when performed by different people may exhibit similar time-series data.

4.3.1 Activity Classification via Time-Series Analysis

We begin this study by focusing on human ambulatory behavior as an example, but the developed approach generalizes to all actions. Figure 4.7 shows the accelerometer readings for three specific activities: standing, walking and running. By analyzing the time series data (left column) in this figure (taken from accelerometer readings of a phone placed in the front pocket with a sampling rate of once every 50ms), it can easily be seen that standing, walking and running produce vastly different patterns. We propose that in order to extract an 'activity level', we should look at two distinguishing factors obvious in this figure. One is the raw magnitude of acceleration and the other is period with which patterns repeat, which represents the frequency of how this acceleration changes

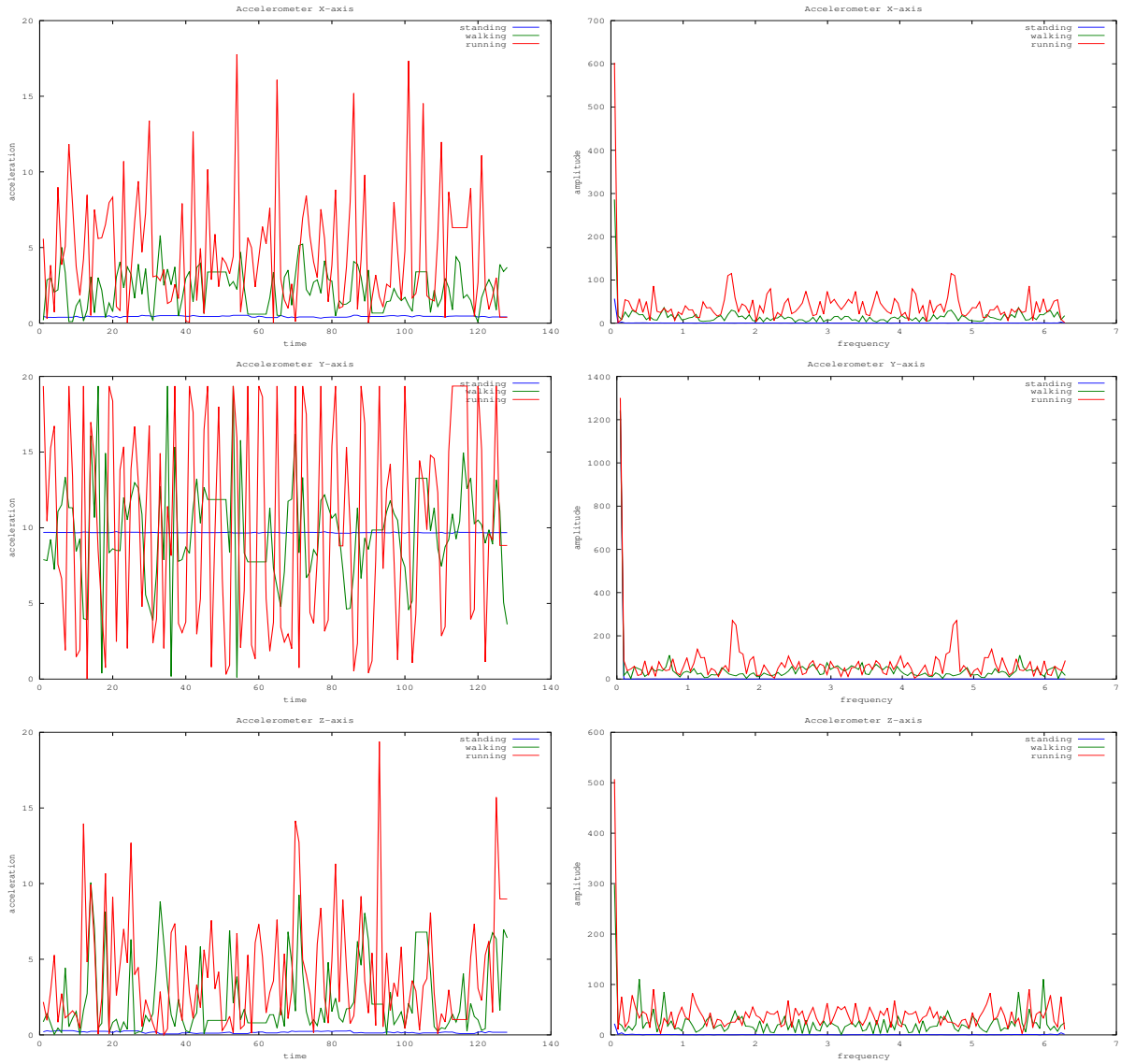

Figure 4.7: Accelerometer readings for standing, walking and running along with the discrete fourier transformation

over time. In Figure 4.7, we see that standing produces a relatively flat line in the time-series data where the magnitude of acceleration is low with no repeating patterns of high acceleration. This contrasts with walking data which shows some irregular patterns with moderate acceleration and running data with produces regular high frequency patterns of high acceleration. We can extract these two factors more easily by applying the Discrete Fourier Transform (DFT) and mapping the data into the frequency domain.

The result of this transformation is also shown in the right hand column of Figure 4.7 which graphs the modulus of the complex amplitudes of the ambulatory activities. With these figures, we demonstrate the transform allows us to more easily extract the level of activity for each window. Windows with high levels of activity such as running will have higher amplitudes with peaks closer to π while a low level of activity will produce a curve with low amplitude. We make the classification by transforming the DFT data into a real valued number between 0 and 1 where 1 represents a window where there is high frequency activity of high amplitude and 0 represents either low frequency, or low amplitude activity or both. To do this, we take the m largest peaks in the range $(0, \pi]$ (since the DFT is symmetric) and average the x and y values. The intuition for doing this lies in the fact that we want to look at the largest m sinusoids which contribute to the original wave. By averaging the x values we obtain the average frequency and by averaging the y values we obtain the average amplitude contributed by these waves. Now to map these values onto the range $[0, 1]$ we use the hyperbolic tangent function because of the desirable shape of its graph (lying on the range $[0, 1)$ in the domain $[0, \infty)$).

$$\begin{aligned} ActivityClass &= \tanh(\alpha \times \bar{x}) \times \tanh(\beta \times \bar{y}) \\ &= \frac{(e^{\alpha \times \bar{x}} - e^{-\alpha \times \bar{x}}) \times (e^{\beta \times \bar{y}} - e^{-\beta \times \bar{y}})}{4} \end{aligned}$$

Using the above equation (where $\bar{x}$ is the average of x values, and $\bar{y}$ is the average of y values), we map the DFT of each axis to a value on the range $[0, 1)$. This range can now be split arbitrarily

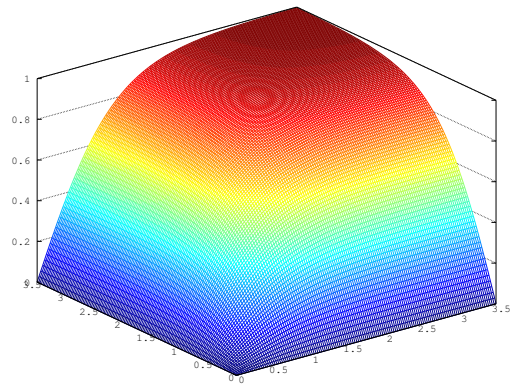

Figure 4.8: ActivityClass function with $\alpha = \beta = 1$

depending on the number of activity classes needed and the properties of each of these activity classes. A plot of this function can be seen in Figure 4.8 to give an intuition of how some $\bar{x}$, $\bar{y}$ pair will get mapped. It can be seen from this plot that a value below 1 for either axis brings about a steep drop in the result, as we want to penalize data with either low acceleration or low frequency patterns. Now we get a final value for each window by taking the maximum $ActivityClass$ value from all axes of the accelerometer.

4.3.2 User Identification based on Activity Recognition

A rich body of machine learning algorithms has been developed over the years. Many of the new learning algorithms, such as support vector machines, Bayesian and neural networks, are routinely used in commercial systems. In some scenarios, we can also utilize and adapt the standard approach to model human behavior. Being able to do activity class recognition allows for the modeling of each class for different users. Given that the model is built for a distinct class, it is possible to use existing machine learning classification techniques to learn the different characteristics of that particular class, say for example a walk. In this section we describe a feature set to help capture the individual traits of each person performing a similar action.

The feature set described in this section does not rely on the sensor input type, it just needs the input to produce a stream of real valued samples. From this stream we can form collections of n consecutive samples starting with some sample s_{t_0} using a sliding window. For each window w we can compute a set of features which represent the action a of time $t_{n-1} - t_0$ starting at time t_0. The features calculated include:

- The root mean squared value, which is a statistical measure of the magnitude of a varying quantity:
$$x_{rms} = \sqrt{\frac{1}{n}\sum_{t=1}^{n} x_t^2}$$
- The root mean squared error, which is a measure of the differences between values predicted by a model and the values actually observed:
$$x_{rmse} = \sqrt{\frac{1}{n}\sum_{t=1}^{n} (x_t - \hat{x})^2}$$
(where $\hat{x}$ is the mean value for that dimension)
- The minimum value of window w:
$$x \in w : x \leq y, \forall y \in w$$
- The maximum value of window w:
$$x \in w : x \geq y, \forall y \in w$$
- The average sample by sample change of window w:
$$\bar{\Delta} = \frac{1}{n}\sum_{t=2}^{n} (x_t - x_{t-1})$$
- The number of local peaks:
$$Count(t \in \{1 \dots n\} : x_t \in w, x_t > x_{t-1}, x_t > x_{t+1})$$

- The number of local crests:

$$Count(t \in \{1 \dots n\} : x_t \in w, x_t < x_{t-1}, x_t < x_{t+1})$$

- The average time from a sample to a peak:

$$\frac{1}{n} \sum_{i=1}^{|Peaks|} \sum_{Peaks_{i-1} < j < Peaks_i} (Time(x_{Peaks_i}) - Time(x_j))$$

- The average time from a sample to a crest:

$$\frac{1}{n} \sum_{i=1}^{|Crests|} \sum_{Crests_{i-1} < j < Crests_i} (Time(x_{Crests_i}) - Time(x_j))$$

- The root mean square cross rate:

$$x_{rcr} = Count(t \in \{1 \dots n\} : x_t \in w, x_t < x_{rms}, x_{t+1} > x_{rms})$$

- The signal magnitude area:

$$SMA = \max(w) - \min(w)$$

4.3.3 Feature Selection

With this feature set, we can further perform feature selection once we know the input sensor type and the application of the model, to increase the performance of any algorithms using the set and also make it more relevant to the problem. Our approach for feature selection is based on measuring the information gain of a feature with respect to class. To measure information gain, we use the concept of entropy which provides a measure for the average unpredictability in a random variable, equivalent to its information content. Entropy can be defined as:

$$H(X) = \sum_{x \in X} -P(x) \times \log(P(x))$$

From this, we can rank the features by measuring the information gain which is defined as follows:

$$I(Y, X) = H(Y) - H(Y|X)$$

which in this context translates to:

$$G(Class, Feature) = H(Class) - H(Class|Feature)$$

4.3.4 A Gaussian Model for User Authentication

We approach the user authentication problem by building a model with gaussian distribution to detect anomalies in user behavior. Here we take an unsupervised and generative approach, where in some period of learning, all data is assumed to be normal data from one user, to form a set of positive samples. After this training period, new data is tested against this model and a probability is calculated for the likelihood of it being generated by the model. We will then classify all data with a probability less than some ϵ an anomaly and assume it was caused by someone other than the primary user.

To create the gaussian model, we use the same features described above. We calculate the feature vector for m training examples $x^1 \dots x^m$ indicative of normal usage and for each feature, we will fit the parameters μ (mean) and σ^2 (standard deviation squared) as per the following equations.

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^i$$

$$\sigma_j^2 = \frac{1}{m} \sum_{i=1}^{m} (x_j^{(i)} - \mu_j)^2$$

To calculate the probability of some feature j taking a value x, we use the gaussian equation:

$$p(x : \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Now by making the assumption that all of our features are reasonably independent, we can calculate the probability of some window of data being caused by the primary user by calculating

its feature vector and computing:

$$p(x) = \prod_{j=1}^{n} p(x_j; u_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(x_j-\mu_j)^2}{2\sigma_j^2}}$$

4.3.5 Augmenting Motion Data with Location Knowledge

Using motion data by itself is sometimes not enough to discriminate between two users. This is evident especially for those actions with relatively low levels of activity. There isn't enough information available for proper analysis. As part of our SenSec framework, in this section, we seek to mitigate this problem by augmenting motion data with location knowledge. We hypothesize that people spend the majority of their time in only a few regular locations (for example home, office, school) and thus actions performed in a location can be tied to specific users. The rationale behind this is most people frequent different locations, if a phone is stolen, and taken to a rarely visited or not visited area, then a high probability that it is not being used by the primary user should be reflected in the framework. Even within a location, there are different probabilities with which we perform different actions. For example, when one is at the movies, the phone is relatively stationary, as it is not courteous to use their phone. On the other hand, when a person is at work, depending on profession, he/she may frequently check for updates on work related messages. If this is true, by including location context, we should be able to lower the number of false positive results improving the usability of our framework.

The approach taken to integrate location into the previous anomaly detection system is similar to a naive bayes approach. We still divide the samples into groups based on their frequency and amplitude. For each group, we are able to model a gaussian distribution for each of the motion features. As before, we can use these distributions to calculate the probability that some samples were generated by that model. Let us denote this probability as $P_{m,M}(fv)$ which represents the probability that some feature vector fv was generated based on its motion features, given some model M.

We now use a naive bayes approach to calculate $P_{l,M}(fv)$ which is the probability that some feature vector fv was generated based on its location features, given some model M.

Using the chain rule in probability theory:

$$P(A_n, \ldots, A_1) = P(A_n|A_{n-1}, \ldots, A_1) \times P(A_{n-1}, \ldots, A_1)$$

We can calculate the probability that some action class C was performed at some location L with:

$$P(C \cap L) = P(C|L) \times P(L)$$

Here we calculate $P(L)$, the probability that the user was at location$L$, by counting the number of times M has seen the location L and dividing it by the number of times M has seen any location. Similarly we calculate $P(C|L)$ by counting the number of times action class C was performed at location L divided by the total number of times any action class was performed at location L. To combine the two factors together, we use a simple, but in this case empirically effective, concatenation, which just multiplies the two values together. Hence our new anomaly detection system measures the probability that some model M generated some motion vector and location pair fv by calculating:

$$P_M(A) = P_{m,M}(fv) \times P_{l,M}(fv) = P_{m,M}(fv) \times P(C|L) \times P(L)$$

4.4 SPATIAL MODELING ON USER INTERACTION PATTERNS

As part of SenSec's user interaction metrics, we take into account how a user's soft keyboard usage contributes to their unique behaviometric profile. While typing, users have a variety of different typing rhythms and the physiological traits of their hands, joints and fingertips ensure that no two users type exactly alike. Alongside traditional desktop-like features such as experience level and posture, soft-keyboard typing seems to be influenced by hand size, finger length, fingertip size, muscle development, posture and position, one-handed or two-handed typing, orientation of the phone, user tiredness, coldness of fingers, focus of the user (mobile users are often partially en-

gaged in other activities), and whether the user is walking, standing, sitting, lying down, or riding in a car, bus or subway. Additional influences are size/shape of the phone, screen and external phone cases. Our analysis of soft keyboard usage takes a different tact from previous methods which had a focus on time-series data. In this section, we have considered these influencing factors and developed several spatial models to tackle the challenges that arise when learning how different users interact with their mobile device.

4.4.1 Feature Selection

After analyzing the prior research on keyboards as well as the smartphone-specific information, we developed our set of features to capture and analyze. These will be used to help train our model and be will analyzed for patterns between users.

Location pressed on key is our primary micro-behavior metric and is recorded per key as an ordered pair $(X_{\text{offset}}, Y_{\text{offset}})$, expressed from that key's center.

Length of press from finger down to finger lift, a traditional metric, has much greater variation on mobile phones, but does improve model performance.

A user's *force of press* or *pressure* is available as a unitless value between 0 and 1 on many Android devices [43]. This allows for tracking of metrics such as distributions of *maximum touch pressure*, but is complicated by inconsistent scales between device models.

Similarly, the user's *size of touched area* is also available as a unitless value between 0 and 1 [43].

We can also analyze variability in *size* and *pressure* information within a single keypress. This allows analysis of *pressure and size dynamics* for the user.

Another feature, *drift*, records how much the user's finger moves between pressing down and lifting up and at what angle they slid.

Additionally, with easily available sensor data from accelerometers and gyroscopes, information about *orientation* (where the phone faces while held) could help improve comparisons and

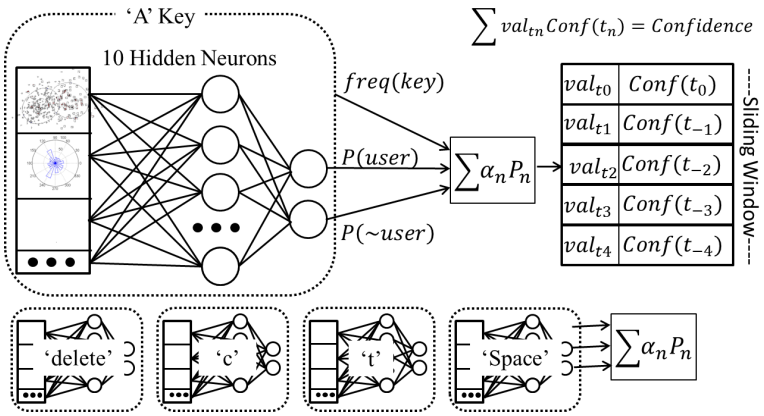

Figure 4.9: A high-level diagram of how the discriminant algorithm is designed. Per-key neural networks generate confidence scores weighted against training size. These are aggregated into a combined score for a 5-key sliding window to generate likelihood of non-authorized user.

anomaly detection. This proved to be challenging to correlate between handsets, but the addition of a calibration session could enable successful consideration of this feature.

4.4.2 Discriminant Model

We developed two models for comparison testing between users—each suited for different environments. The method of analysis we used is offline supervised learning trained with data from other users. While not as scalable as a purely generative version, this has some additional advantages—better non-authorized user recognition, better battery life by reducing on-phone computation, and more securely stored behavioral data (on server rather than directly on phone). This technique trains small neural networks for each key, using samples of other users' behavior as 'non-authorized user' training examples [44]. A high-level diagram explaining this technique can be seen in **Figure 4.9** and the following paragraph contains a detailed explanation.

This model leverages the multi-user data collection environment and takes advantage of the high number of training keypresses. While more computationally expensive, this facilitates much higher recognition rates and fewer errors. Each key with a reasonable amount of training data (we set threshold at 15 keypresses) is trained with its own two-layer, feed-forward neural network. Scaled

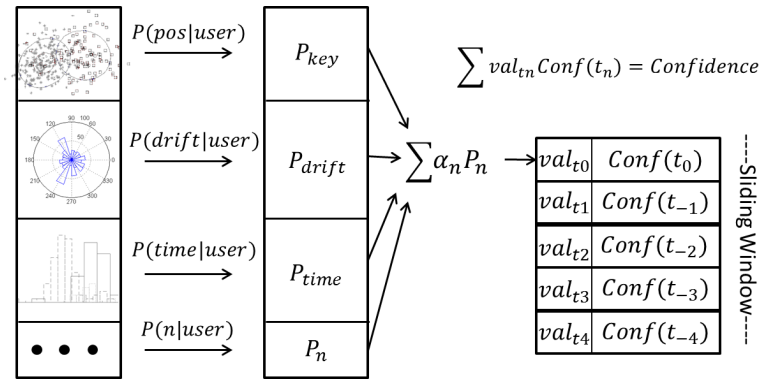

Figure 4.10: A high-level diagram of how the generative algorithm is designed. It calculates feature probabilities individually, aggregates them to a confidence score for a single key press and uses a 5-key sliding window to generate likelihood of non-authorized user.

conjugate gradient back-propagation is the learning method [44]. We used 10 neurons for the one hidden layer to ensure there is sufficient power in the analysis. The performance function is the *mean squared error*, but another performance function checks after training to ensure the network is not ill-fitted. This may ask networks to retrain if they settle in poor local minima (e.g. classifying everything as the regular user). The output from these neural networks are then weighted by how many keypresses were used to train that key and are averaged across the last 5 keypresses (using a simple mean). This aggregate confidence is compared against a threshold. The threshold (and the per-key weighting) is set by using a regularized logistic regression algorithm [44]. The cost function for the algorithm is shown in **Equation 4.21**. To improve scalability and to allow for non-networked computation, this model will later be replaced with a single-user generative model (described below).

$$\min_{\theta} \frac{1}{m} \left[\sum_{i=1}^{m} y^{(i)} \left(-\log h_\theta(x^{(i)}) \right) + (1 - y^{(i)}) \left(-\log(1 - h_\theta(x^{(i)})) \right) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2 \qquad (4.21)$$

4.4.3 Generative Model

Our intention is to further develop our generative model that can detect anomalies without network access or comparison to other users. A generative model is needed for this approach. For a single user's features, historical distributions can be determined and models (Gaussian or otherwise) developed [45]. Then, upon new data input, the probability that each feature has come from the user is calculated. Aggregated for all of the features, these probabilities give a per-key confidence score. Additionally, as per an evaluation by Killourhy and Maxion, an *outlier count* feature was added that tallies the number of recent outliers (shown to be effective at detecting anomalies on desktops) [45]. In a similar fashion to the discriminant model, number of historical keypresses at that key serves as the weight relative to other recent keypresses. See **Figure 4.10** for a visual explanation. This weighting is done through the same logistic regression as described in the 'discriminant model' section but is expanded to include the input features processed by that model's neural network [44].

4.5 HETEROGENOUS SENSOR FUSION

In this chapter we have discussed methods and techniques used to model multiple different sensors. However, effective modeling requires the system to consider different factors together. Intuitively, fusing information from different types of sensors should improve the discriminative power of the passive authentication classifier. For example, we can obtain the user's outdoor location trace and her traveling speed with just a GPS sensor. However, if her mobile phone is stolen by an attacker that is familiar with the user's commuting pattern, then the attacker can just walk along the same route as the user to break into the system. In this case, fusing the location information with other sensor information such as gait and phone tilting can help to identify the anomaly.

The most straightforward straightforward way of sensor fusion is to concatenate several sensor readings into one value and train the classifier on the concatenated input. For example, after fusing the location, acceleration and time information, the input may look like "location=37.378535 N

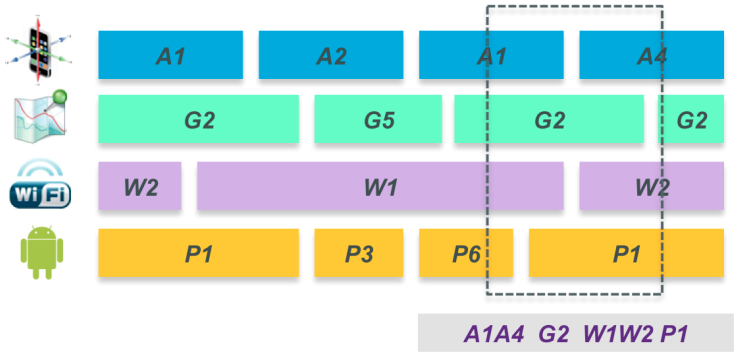

Figure 4.11: Sensor fusion by concatentation

122.086585 W; acceleration-x=0.017g; acceleration-y=0.002g; acceleration-z=-0.014g; time=2:05pm". However, this approach suffers the *curse of dimensionality problem*. When the number of fused sensors increases, the total number of possible combined values increases exponentially. The fused sensor representation requires enormous amounts of training data to train a reliable statistical model in order to be general enough to classify unseen data. Even if we preprocess the sensor data by quantizing the sensor readings after which real number values are converted to discrete symbolic labels, the vocabulary's size can still be in the range of hundreds or thousands and the combination of all sensor values can easily grow to millions or billions. Thus fusing by concatenating is only practical to fuse a small number of sensors. An example can be seen in **Figure 4.11**.

Another problem of this approach is over partitioning. Over partitioning happens when the classifier partitions the value space into too many classes. For example, if we fused "application usage: information with "location" and "time" information through concatenation, the classifier might partition the "checking email" activity to multiple activities such as "checking email in office at 2:13pm", "checking email in the hallway at 10:30am", and "checking email at home at 11:23pm". This can be overkill as checking email can happen at any time. Knowing the habit of where the user usually checks email is sufficient for contextual security.

We may mitigate this problem by having each sensor generate its own confidence score and combining them with some linear or nonlinear combination. For example, say we have the confidence

scores for location s_l, accelerometer s_a and gyroscope s_g, we can just take the overall confidence score to be

$$s = \alpha \times s_l + \beta \times s_a + \gamma \times s_g \tag{4.22}$$

This essentially turns the sensor fusion problem into an optimization problem, where we try to find the appropriate weights to maximize the detection of anomalous usage while minimizing the number of false positives. We may even choose to use some nonlinear function for example the step function for which $s = 1$ if $s_l > th_l \wedge s_a > th_a \wedge s_g > th_g$ for some arbitrary thresholds and $s = 0$ otherwise. The current approach used in the *SenSec App* uses a hybrid model which combines the concatenation approach with this combination technique. For sensors which generate multidimensional values, for example the accelerometer, we use the concatenation approach to generate a confidence score for that sensor. Then we combine these confidence scores together using a simple linear combination. This seems to work relatively well in practice (See Chapter 6)

To further explore this area, we tried another method for sensor fusion. Similar to decision trees, a technique widely used for classification tasks, we can automatically construct a risk analysis tree for each user to determine whether the system is at risk or not by querying multiple sensors. **Figure 4.12** shows an example Risk Assessment Tree. Here, the RAT first queries the sensors to get the user's outdoor location. If the user is not at home or at work, the RAT then queries the GPS to obtain her traveling speed. In the case that she is most likely driving given her traveling speed, the RAT checks which application is running on the mobile device and applies a risk assessment function that fuses the sensor readings from geo-trace, app-category and time information. Intuitively, the RAT partitions the full feature spaces into small regions where certain combination of sensors are most effective to assess the risks of the mobile device. To train the RAT model, we take two approaches with slight differences:

We then build a decision-tree classifier to partition the feature space into sub-spaces where a subset of features are responsible for the risk assessment. For each subspace, we will then train a

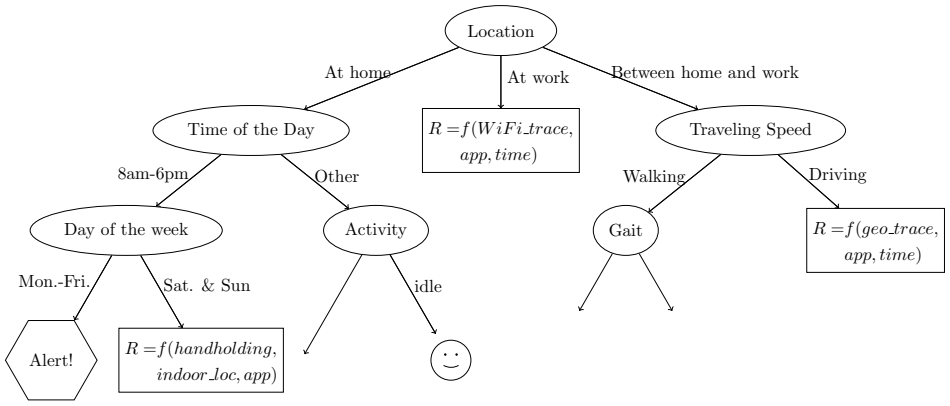

Figure 4.12: Sensor fusion through Risk Analysis Tree.

regressor to optimize weights for each sensor feature to best quantify the risk level of the mobile device. Moreover, we can systematically select subsets of the features and build multiple RATs and use a weighted average or voting method to combine the decision from these RATs to quantify the risk level.

4.6 SUMMARY

In this chapter, we have explained the theory behind our behaviometric models and described methods with which we can use to interpret these models mathematically. We first introduced a language approach to behavior modeling that utilizes NLP techniques to perform inference. This approach allowed for the conversion of multi-dimensional sensory data into a behavioral text representation and used a novel hierarchical scheme, Helix, to segment and classify sensory time series. We continued with techniques to analyze sensory data that is based off of more traditional time-series methods. Finally we introduced spatial analysis techniques for contexts such as soft-keyboard modeling where time-based methods may not be as effective. Through this, we have explained the most critical phase of our methodology, making clear how to perform characterization, classification, prediction and anomaly detection in mobile user and group behaviors.

CHAPTER 5

EXPERIMENTS, RESULTS AND EVALUATION

In the previous chapters, we have covered the major research problems and challenges in Mobile Behaviometrics, and have presented the details of a mobile application, *SenSec* to ensure mobile device and application security using Behaviometrics techniques.

In this chapter, the effectiveness of the proposed approaches will be evaluated through *SenSec*'s deployment, data collection and experiments. We implemented several experiments to evaluate these approaches in a variety of different situations. We begin with three experiments which demonstrate the language based approach to human behavior modeling. The first experiment examines using WiFi traces to model a user's location and mobility patterns, the second experiment explores performing user classification on users performing the same task and the third experiment studies automatic segmentation and recognition of unlabeled activities. We then perform an experiment which focuses specifically on motion data to demonstrate the improvements that can be made to user identification and authentication by first performing activity recognition and then further improve upon this by adding the location factor. Finally, we evaluate our spatial model for soft keyboard modeling by developing and deploying our own soft keyboard as part of *SenSec*.

5.1 USER MOBILITY BEHAVIOMETRICS

In this experiment, we introduce the practical aspects of the *behavior as language* theme by exploring how system metrics can be used to build a mobile users' location and mobility behaviometrics. We start with a set of experiments which leverage WiFi traces. We adopted the *syslog* [6] approach to collecting data for these experiments. Unlike [6], in which only the associated WAPs is recorded, we collect RSS of the beacons for mobile devices on multiple WAPs. Because all of these WAPs' internal clocks are synchronized, it allows us to aggregate traces from different WAPs and serialize them on a single time line. This approach allows us to infer mobile devices' location in finer granularity, in addition to their associated WAP. In our study, we constructed a series of RSS vectors $\vec{s_t} = (s_{1,t}, s_{2,t}, \ldots, s_{N,t})$, where $s_{k,t}$ is the RSS of a mobile device's beacon detected by the kth WAP at time t. We also used two types of information to model a user's mobility behavior: (pseudo) location and time. We adopted the fingerprinting approach similar to [46] to assign a pseudo location label for each segment of RSS readings to represent the device's current location. Compared with [6], our approach removes the overhead of estimating the actual mobility path from the associated WAPs path and the pause time at each location.

5.1.1 Label Generation

5.1.1.1 Pseudo location label generation

We assume that the infrastructure in the building is relatively stable, i.e., positions of WAPs, devices used as WAPs and radio environment are not changed over time, which usually holds true for office buildings. Therefore, RSS vectors with high similarities will indicate the device is within proximity. We used k-means clustering algorithm with a distance function inspired by Redpin [47] and WASP system [48] to convert the trace for each mobile user into a series of pseudo location labels, where each label represents a class from the clustering results. Our experiments showed that choosing $3N$ as the number of clusters, where N is the RSS vector dimension, produces the best quality/performance trade-offs.

Using the algorithm described above, we convert the trace for each mobile user c into a series of pseudo location labels $L_c = \{l_0, l_1, l_2, ...l_t\}$.

5.1.1.2 Collapsing recurring pseudo location labels

Due to the recurring nature of the data collection process, if a user stays at a pseudo location for a long period of time, there will be sequences of repeating labels in the data. The n-gram model would fail to model the transition of actions and the probability mass will be dominated by the most visited places.

To mitigate this effect, we collapsed repeating pseudo location labels into a set of new pseudo location labels. Time spent at a location can be user-dependent. For example, a faculty member may stay in his office for a long time, while a student visiting him may spend less time at the same location. The process described above may result in the loss of this information. To overcome this, we constructed another *Duration* feature by counting the repeating pseudo location labels before collapsing.

5.1.1.3 Sequence Segmentation and Time-of-day label

Users' behavior follows certain regularities due to regular meetings and routines associated with their responsibilities and roles as shown in Figure 5.1. For example, a group of students who work on the same course project may get together in a conference room or a common area after the class; faculty members may have quick group meetings at certain time of the day to get status reports from the students.

In our raw data set, every RSS vector is marked with a time stamp (epoch). It provides extra information to assist the modeling of user behaviors especially with routine and fixed schedules. We partitioned the duration of 24 hours into multiple non-equal-size sections and used the tags NA, AM, PM, EV as the time-of-day (TOD) labels. While not the focus of this study, we found that the universal division of the duration may cause problems if the users have shifted working hours by which a series of associated mobility behaviors are right across the section boundary. We

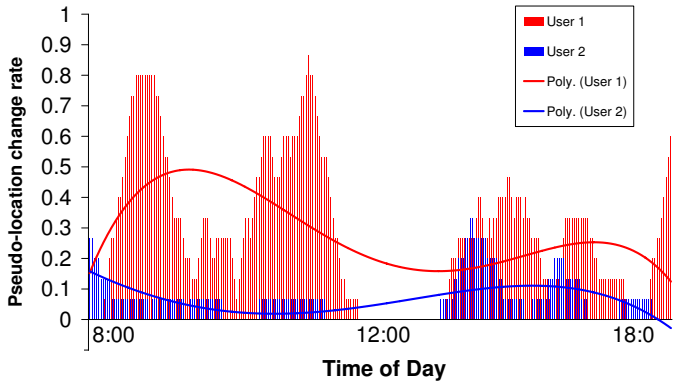

Figure 5.1: Regular patterns from user's daily mobility behaviors

proposed an Activity-Level Adjusted Segmentation (ALAS) scheme where we adjusted the section boundary based on a given user's mobility behavior. The rationale is to cut the section boundary so that the continuity of user mobility behavior is maximized.

5.1.1.4 Multi-label fusion based on Pair-Wise Mutual Information

As described in Section 5.1.1.2 and 5.1.1.3, we have multi label streams in addition to pseudo location labels from the RSS trace. Blindly concatenating the labels for different features together will either inadequately reduce data resolution or necessarily increase the model complexity if the sampling rates are very different and the granularity of the data source is not comparable.

Therefore, we explored heterogenous sensor fusion techniques to combine these different streams of labels into a single stream of behavior text before fit it to our n-gram model. We use Mutual Information, a simple metric, to determine whether time-at-location features should be combined with the pseudo location feature. This approach is to include the duration label only if it is highly correlated with the location label, to capture the high level activities associated with this location/time pair.

The change rate of the Time-of-Day label is small compared to the other two features. Therefore, we decided to concatenate them with the fused label in this study.

This multi-label fusion process allows us to add more features, such as the type of applications

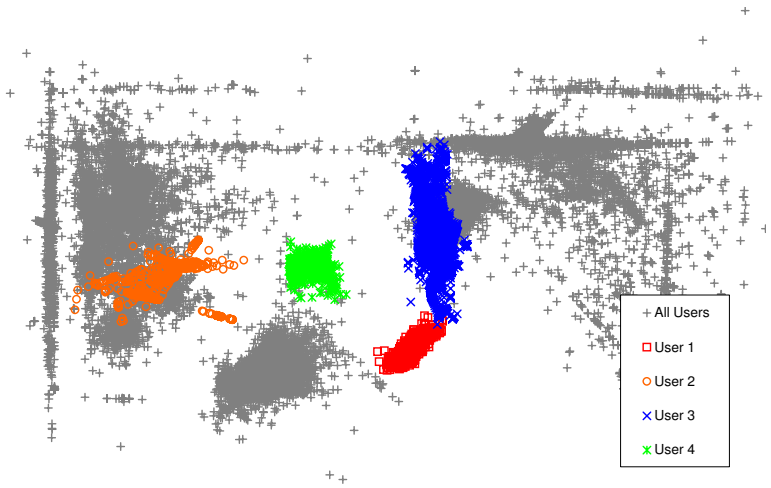

Figure 5.2: Pseduo-location density from 4 selected users vs all users

running on the mobile device, their network resource consumption and battery information, into behavior text streams without losing their importance in the n-gram modeling.

5.1.2 Data Set

We obtained series of pseudo location labels of 40 users, who were selected from the traces of 587 users collected from 87 WAPs over 5 days. If two users' mobility areas are entirely separable, it is relatively easy to identify the users by their pseudo location label histogram. As shown in Figure 5.2, user 2 and user 4 are completely partitioned into two separate areas, while user 1 and user 3 have some overlap. We calculated the cross entropy of those labels among all these users. To test our n-gram model's strong capability in anomaly detection, we choose 10 users that have the least cross entropy. This ensures that users' moving paths strongly overlap and the location label histogram can not be used to efficiently identify users.

5.1.3 Anomaly Detection

5.1.3.1 Simulated testing data through splicing

We used 3/4 of the samples as the training set and the remaining 1/4 data to create the testing data. Since there were no real device stolen events recorded in the data we collected, to make the experiments more generalizable, we created simulated device stolen events by *splicing* two users'

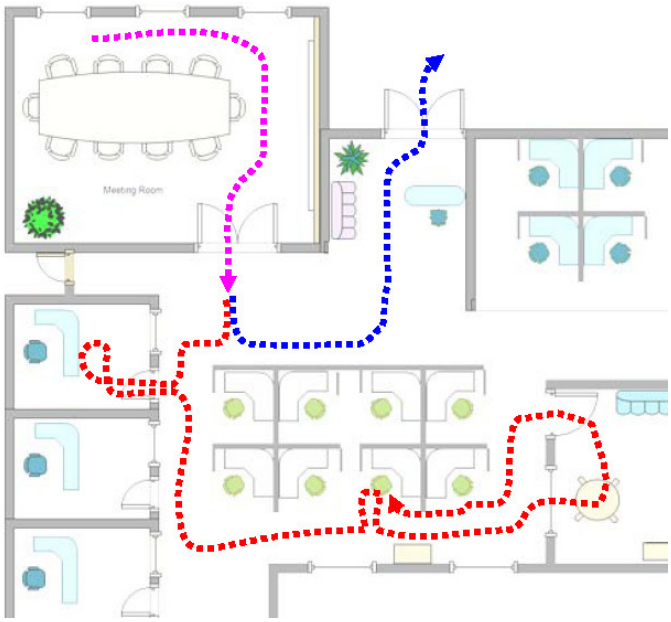

Figure 5.3: Two users' mobility patterns are concatenated to create a sample for a simulated stolen event. Originally pink and red are user A's trace. blue is user B's trace. We splice pink with the blue to simulate a device stolen event.

trace segments at their intersection point. This simulated device-stolen trace has the real moving patterns from both the original owner and the "unauthorized user" and those patterns intersect at the *event time* as show in Figure 5.3. We combined these simulated device-stolen samples and the normal samples to create a testing set.

5.1.3.2 Anomaly detection and threshold

In the testing phase, we adopted a sliding window scheme with a window size of 20 seconds and step size of 2 seconds. As the sliding window advances, we continuously calculate the likelihood (average log probability as shown in Equation 4.7) of the behavior text in the sliding window that is generated by the n-gram model of the original user. We will trigger an anomaly detection alert when the log probability drops below a predefined threshold. When such situation is detected, we record the location of the sliding window and the "response time". In other words, this estimates how much time it takes for the system to detect the mobile device has been stolen.

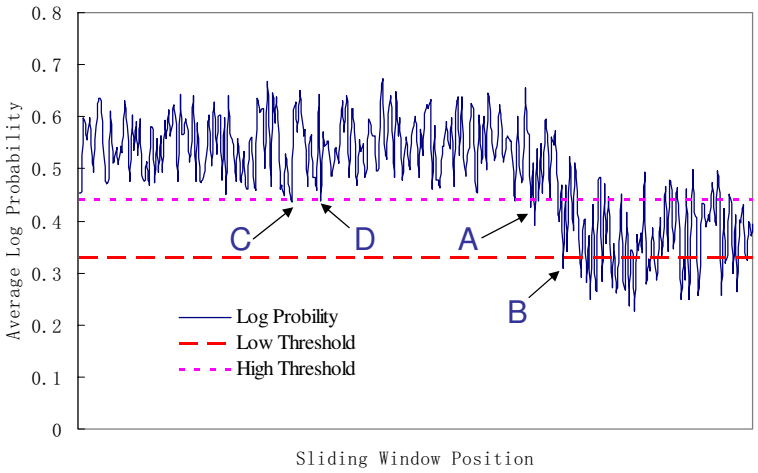

Figure 5.4: Anomaly detection experiment results for a user with different thresholds. The graph shows average log probability within a sliding window over time. The stolen event happens at A. C and D are false positives for threshold 0.42. B is the true positive for threshold 0.37. The response time is the distance between B and A.

As shown in Figure 5.4, at first the log probability of the testing segment is high because the data matches the user's behavior model. As the sliding window moves closer to the event point A, the log probability drops and eventually goes completely below the threshold, where an alarm is triggered. We performed these experiments for all 10 users with different training data sizes (by means of hours L), different history lengths (n) and different detection thresholds. Among the users with high cross entropy in pseudo location labels, the n-gram model correctly detected 86% of simulated device-stolen events with $n = 10$ and just 12 hours of training data.

About 10% of the non-stolen testing samples trigged the alerts as false positive cases. These results were obtained using 0.40 (average log probability) as the detection threshold and 10 labels as maximum allowable response time. If an event is undetected at the 10th label after the event happens, we consider this testing sample undetected, and classify it as a false negative.

Figure 5.5 shows Receiver Operating Characteristic (ROC) curves for anomaly detection results. Each points on the ROC curve corresponds to a certain detection threshold. The upper left corner represents perfect detection, while diagonal represents the model that is performing no better than a

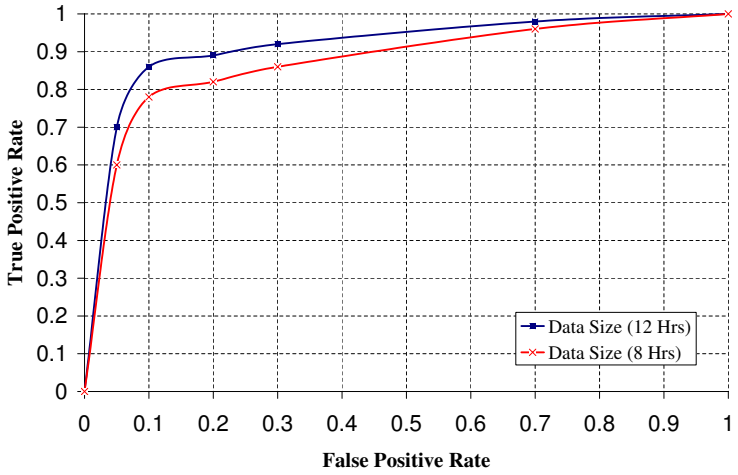

Figure 5.5: Receiver Operating Characteristic (ROC) curves of the n-gram model for training size of 4 and 12 hours.

coin-flip. Figure 5.5 also illustrates the trade-offs on detection threshold and data size.

5.1.3.3 Order of n-gram and training data size

We studied the impact of history length n on the detection accuracy. A larger n captures more context in the n-gram model which increases the accuracy. However, it also increases the size of the model. Figure 5.6 shows that anomaly detection performance would saturate when $n > 5$. This suggests that in the majority of the cases, user mobility behavior only depends on the most recent 5 pseudo locations. Figure 5.6 also shows the effect on the training size. If the training size is as small as 4 hours, it may not capture user's mobility behavior, yielding poor accuracy. The closeness of the two curves with training size 8 and 12 hours also suggests that the system only needs to monitor the user's behavior for less than one day in order to build up a reliable mobility behavior model for anomaly detection.

5.1.3.4 Future Location Prediction

In addition to accountability, this mobile behavior model can also be used to predict the future activity of the user based on his/her past and current behaviors. Predicting future activity can be used to optimize resource allocations, such as spectrum and bandwidth, to improve quality of expe-

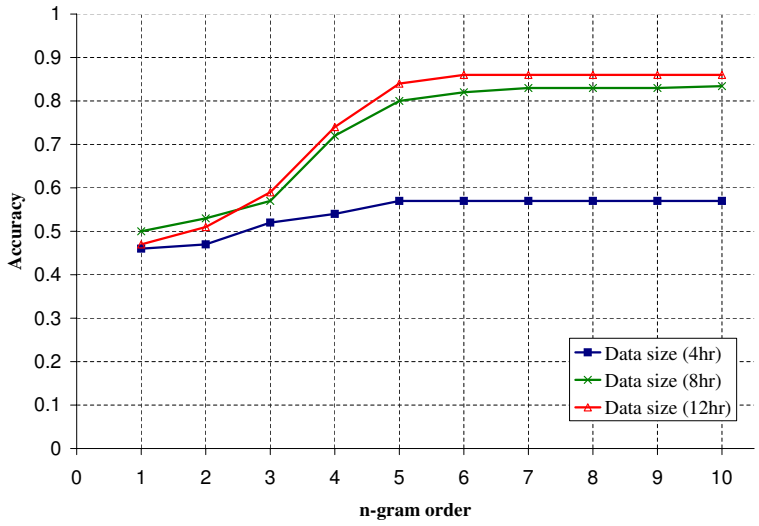

Figure 5.6: Variation of anomaly detection accuracy vs n-gram order. Threshold is set to .59

UID	1	2	3	4	5	6	7	8	9	10
(%)	74.1	60.5	63.3	38.4	72.3	64.9	61.7	65.0	32.8	70.2

Table 5.1: Prediction Accuracy for a selected set of users.

rience of multimedia services including VoIP and video streaming, and to preserve user privacy and maintain network security.

To demonstrate the n-gram model's predictive capabilities, we divided each user's trace into 4 equal-length consecutive segments. We used the first 2 segments to train the n-gram model and tried to predict the user's behavior for the remaining 2 segments. In our study, we empirically set $n = 3, d = 7$. The results are shown in Table 5.1.

We found that the prediction accuracy dropped significantly for some users, specifically, user 4 and user 9 in Table 5.1. Further investigation revealed that our segmentation scheme to divide the training and testing data might have cut behavior changing boundary. As a result, their mobility patterns are different between the training and testing set. After swapping a segment between testing and training, we achieved 63.1% and 65.7% accuracy for user 4 and user 9 respectively similar to other users.

5.1.3.5 Experiment Summary

We collected RSS readings of the beacons emitted from mobile devices. We aggregated the data from multiple WAPs to form sequences of fingerprints. These sequences were used to infer the actual locations of the devices on a fine-grain scale. This approach reduces the overhead in estimating mobility tracks and pause time using more complex models or ad-hoc heuristics. We converted these heterogenous sensor data into behavioral text and adopted a *skipped* n-gram model to capture the long-term dependency in user mobility behavior. In addition to single pseudo location information, we leveraged simple heterogenous data fusion techniques to incorporate other context information, such as time-of-day and duration to mobility behavior text representation.

We applied our model to mobile anomaly detection and future behavior prediction. Through experimenting with data sets captured from production networks, we have shown that this simple n-gram approach can capture the personal mobility patterns. Since our data set did not contain any samples from a device-stolen event, we tested our model in anomaly detection by including a series of simulated device-stolen scenarios. The results showed that the model can detect such an anomaly by examining the log probability calculated with the n-gram model. We also analyzed how certain parameters would affect the performance of the algorithm in terms of scale of the model and accuracy of detections.

5.2 USER CLASSIFICATION BASED ON GESTURE PATTERNS

We now move from the high level modeling of location and mobility to more fine grain modeling of a user's movement. The SenSec framework enables us to model user's gesture patterns and use it to do user classification and user authentication. In this set of experiments, we investigated *SenSec*'s effectiveness in conducting user classification to distinguish users by their gesture patterns when they are performing the same tasks.

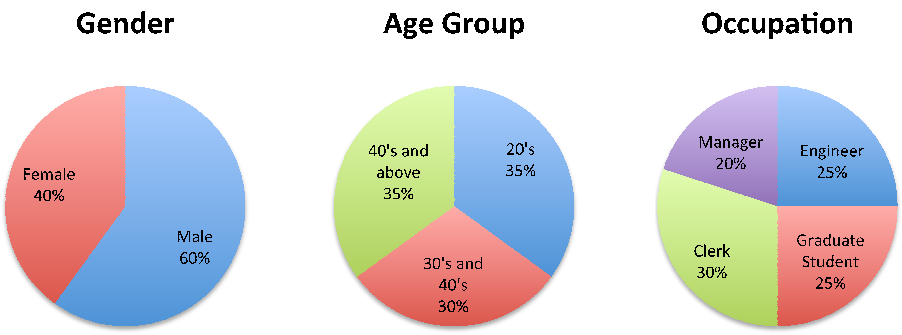

Figure 5.7: Distributions of demographic groups for user classification

5.2.1 Experimental Setup

20 volunteers [1] from different demographic groups (genders, occupations, and age) participated in the study. The distribution of their demographics are shown in Figure 5.7. Due to the planned, systematic nature of this experiment, it serves to be a proof of concept that user classification can be achieved through learning gesture patterns.

All participants were given the same Google Nexus S smartphone loaded with our sensing application and were asked to perform the same task for 5-10 times. Specifically, they were asked to perform the following tasks in a sequential order:

1. Pick up the device from a desk
2. Unlock the device using the right slide pattern [2]
3. Invoke Email app from the ”Home Screen”
4. Lock the device by pressing the ”Power” button
5. Put the device back on the desk

[1] User studies were conducted with the approval of IRB application HS11-094, ”Human Behavioral modeling using Mobile Sensors” from the IRB Board at Carnegie Mellon University. [2] No password or lock patterns are configured on the phone

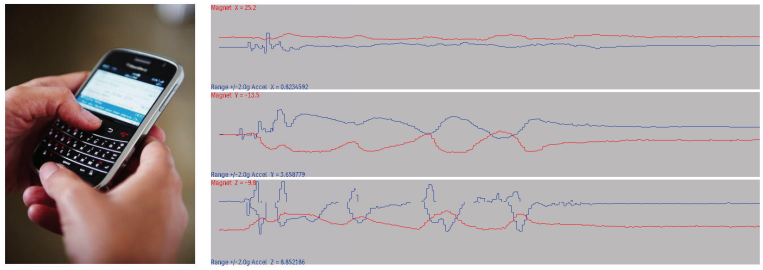

Figure 5.8: Users perform the same sequences of activities and SenSec is monitoring and building Motion Metrics

While the users are performing this specific sequences of activities, SenSec application is recording the motion sensor data time-series as illustrated in Figure 5.8. The sensor data are used later to build user motion metrics to experiment on user identification and authentication tasks. It is worthwhile to note that this experiment is **neither realistic nor robust**. The only purpose of this experiment is to build a **proof of concept** system.

5.2.2 Data Set and Parameters

During the experiments, our sensing application monitors the accelerometers, orientations, gyroscopes and magnetometers. We set the sample rate of these sensors at 4Hz, i.e. the application produces 4 samples per second and each sample contains 12 real valued readings, including Accelerometer (X, Y, Z), Orientation (Azimuth, Pitch, Roll), Compass (X, Y, Z), and Gyroscope (X, Y, Z). Each experiment lasts about 20-30 seconds, which produces about 1250 time-stamped samples. These samples are then uploaded to our server and labeled with the hashed user ID to hide participant's real identity.

We adopt a sliding window approach in constructing the features. We chose the window size of 2 seconds and stepping offset of 500 ms. Using the data in the window, each feature vector is composed by various statistical features [49], including Root Mean-Square, Root Mean-Square error, minimum value, maximum value, average sample-by-sample change, number of local peaks, number of local crests and combined signal magnitude, etc., to capture sufficient *statistics* of those

Classification Target	No. of Classes	Accuracy
Gender	2	0.81
Age Group	3	0.79
Occupation	4	0.76
User ID	20	0.75

Table 5.2: Experiment Results for User Classifications

micro movements. These motion feature vectors are clustered in V classes using K-means clustering algorithm. The centroids of the resulting V Classes are mapped to behavior text labels and become the dictionary of the model.

For a given classification tasks to predict participant's gender, occupation, age group and user IDs [3], we randomly reserve 20% of the samples from each class as testing set $\{E\}$ and use the rest as training set $\{R\}$ to train a N order n-gram model for that class. For each sample L from $\{E\}$, we calculate $P(L, m)$ for the models of all the classes where $m = 1, 2, \ldots, M$ and select m with the maximum $P(L, m)$ as the class label prediction. We chose $V = 200$ and $N = 5$ due to performance and complexity trade-offs.

5.2.3 Results

The experiment was repeated 5 times, which is equivalent to a 5-fold cross validation (CV). The results are shown in Table 5.2. By using the data from a small group of participants, we can achieve a reasonable accuracy in classifying users of different classes. We also found that most of the misclassifications happened when the experiments were conducted back-to-back. There were chances that the latter participants may have observed previous participants' motion and posture and subconsciously tried to mimic what he or she had done [50], which ideally may disappear in experiments with an isolated environment and with a large number of participants.

[3] We would like to understand how the methodology we adopt here would work for identification problems with different complexity, where gender identification is assumed to be the easier and user ID identification to be the hardest

5.2.4 Online User Authentication

We integrated the aforementioned data preprocessing and modeling schemes into our *SenSec App*, loaded it to a smartphone and gave it to a group of participants to conduct the online user authentication experiments.

5.2.4.1 Training and Testing

These experiments were carried out in the following stages:

1. Training Stage: Each of the participants uses the phone for 24 hours while the *SenSec* app is collecting sensor readings in the background and build the behavioral n-gram model.

2. Positive Testing Stage: In this stage, each participants continually uses her phone for 24 hours. This time the *SenSec* app is switched to testing mode. It collects the sensors reading the same way as in training mode, but also construct behavior text sequences and feeds it to the learned n-gram model. A *sureness* score is calculated as described in Equation 4.1. If it falls below a preset threshold while a certain operation is performed, an authentication screen will be pop up asking user to enter a passcode. The *sureness* score and the authentication decision are recorded for logging and result reporting purposes.

3. Negative Testing Stage: The phones are given to other participants to use for another 24 hours. As in the previous stage, the same operations are performed on the phone and all authentication events are recorded for further analysis.

5.2.4.2 Outcome

We examined the logs generated by these experiments. At each authentication decision point, the sureness score is recorded. By varying the threshold, we can evaluate how well our *SenSec* models perform user authentication under different threshold values. For each threshold value, we can calculate False Positive Rate or FPR and True Positive Rate or TPR and plot Receiver Operating Characteristic (ROC) curves.

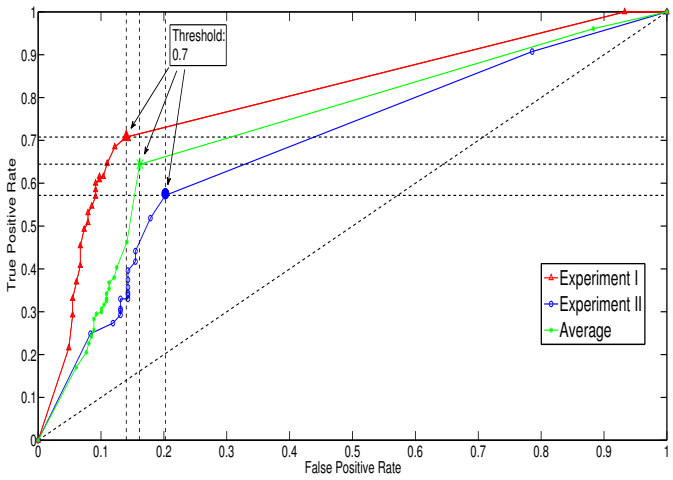

Figure 5.9: Receiver Operating Characteristic (ROC) curves of the n-gram model for training size of 4 and 12 hours.

Figure 5.9 shows ROC curves for user authentication experiments. Each points on the ROC curve corresponds to a certain threshold. The upper left corner represents perfect authentication, while diagonal represents the model that is performing no better than a coin-flip.

The True Positive Rate (TPR) represents the cases that the system successfully detected non-owner access, while the False Positive Rate (FPR) represents the false alarms when the owner is detected as a non-owner. To ensure the security of the system while still keeping the usability to a certain level, we need to maximize TPR and minimize FPR. The ROC curves provide a guideline on choosing the right thresholds to fulfill such a requirement. For example, the system may have a requirement to effectively detect 70% of the unauthorized access, but it may need to keep the false alarm rate below 15%. As shown in Figure 5.9, we can choose a data point from the allowable region on the ROC curve: threshold 0.7 can achieve 71.3% TPR and only 13.1% FPR. We also examined the detection delay at this threshold by deliberately passing the phone between the two users more often and recording the time. By comparing these recorded time and the timestamps of triggered anomalies from the log, we found *SenSec* bears an average 4.96 seconds detection delay.

Table 5.3: Sensors on the mobile client and their sampling rate.

Sensor	Sampling Rate
Accelerometer, magnetometer	
Ambient light, temperature	20Hz (every 50 ms)
Microphone	8KHz
Camera	
GPS, WiFi	every 2 minutes

5.3 HIERARCHICAL ACTIVITY SEGMENTATION AND RECOGNITION

We continue our language as action based modeling in this section, extending the classification of users performing the same task from the previous experiment to automatic activity segmentation and recognition. The data for this experiment was collected with the MobiSens[4] mobile client, part of the MobiSens mobile sensing platform designed for real-world mobile sensing applications using novel algorithms, running on a *Motorola Droid*. Table 5.3 shows sensors used and their sampling rates. Although video and audio data could help, we decided not to use them in MobiSens for three reasons. First of all, the media stream will create a large data storage and transmission overhead, which greatly reduces battery life. Another reason is privacy concerns, especially concerning the recording of video. In addition, we realized that capturing video on Android requires that the application stays in the foreground, which will prevent the user using other functionalities of the phone.

5.3.1 Experimental Setup

We collected 36 hours of real life data in 5 consecutive weekdays from two graduate students to verify and analyze the system. The data collection process lasted about 7 hours each day. Table 5.4 summarizes activities done by the users.

The users carried two phones during the data collection stage (Figure 5.10). One was tied to the

[4] MobiSens is a framework developed as part of our research to collect sensor data from mobile devices. Refer to Appendix A for more details

Table 5.4: Activity instance count and their average time

Activity	Instance Count	Avg. Time per Instance (minutes)
Walk	7	50.29
Working on Computer	15	66.07
Cooking	4	19.75
Eating	9	20.78
Washing Dishes	2	4.5
Cycling	2	21.5
Video Gaming	2	47.5
Presentation	2	29
Having Class	2	79
Meeting	2	69.5
Talking to Somebody	5	15
Driving	3	8.67
Printing Paperwork	1	44

user's right arm and another phone was used in the normal way: most of the time the phone was in user's pocket and from time to time the user took it out to make phone calls or check emails.

At the end of each day, the user will use the web interface to annotate his activities during the day in two settings:

- Without looking at the unsupervised segmentation, the user listens to the recorded audio and creates from scratch his daily activity summary on the calendar. This segmentation/annotation is used as the ground truth in our experiments.
- Based on the unsupervised segmentation (starting/ending time information) and activity clustering (clustering similar activities and visualizing using the same color) label segmented activities.

The goal of the experiments is to evaluate 1) whether the automatic activity segmentation matches the ground truth, and 2) whether the similar activity coloring helps the user to recall what has happened before.

Figure 5.10: Two mounting positions of phones in experiments: stowed in pocket (left) and mounted on upper arm (right)

Table 5.5: Sensors and their sampling rate used for the single-sensor classification experiment.

Sensor	**Sampling Rate**
Accelerometer, Ambient Light	20Hz
GPS, Speed	every 2 minutes

5.3.2 Modeling the Data

5.3.2.1 Unsupervised Activity Segmentation based on Motion

Activity segments are the basis units to train activity recognition models. MobiSens first segments the sensor time-series into a sequence of activities in a unsupervised manner without any prior knowledge of users' activities. The automatically generated activity segments reduce the cognitive load of users as it is very challenging for a user to remember the start and end time of an activity but relatively easy to recall what they did during a time interval.

MobiSens uses motion information for activity segmentation. Intuitively, when one changes his/her activities, motion usually changes [51]. Motion is also the most distinguishing modality for most of the human activities. Table 5.6 shows the activity recognition accuracies based on a single-modality. Using motion-only information, we can achieve 72%, much higher than other modalities such as location (GPS).

Table 5.6: Activity classification accuracy using single sensor modality. J48 decision tree is used in this example. Motion is the most informative single sensor in this experiment.

Data Modality	**Accuracy**
Accelerometer (Motion)	0.72
GPS (Location)	0.41
Speed	0.32
Light	0.17

MobiSens' activity segmentation algorithm operates on motion sensor (accelerometer) time-series data.

5.3.2.2 Behavior Text Representation

We apply a sliding window (width=24 samples, step size=6 samples) over the accelerometer time-series and extract motion features from each window to represent the motion characteristics at each time.With the sampling rate at 4 Hz, each window corresponds to 6 seconds. We extract 6 features from each window including the *mean* and *standard deviation* of each of the three axes of accelerometers. During the offline training process, motion feature vectors from many users' data are clustered into V classes using K-Means clustering. During the runtime, we map an incoming motion feature vector to its closest class and use the class label to represent this motion feature vector for the follow up process. The input accelerometer reading stream is thus converted to a sequence of cluster labels which we refer as *"behavior text"* in later sections.

Needless to say, such clustering and quantization process loses information and precision compared to using the raw motion feature vectors. The benefit of discretizing feature vectors to a symbol is the convenience to model the sequential and structural patterns in the motion time-series.

The number of clusters V is the *vocabulary size* of the behavior text. The selection of V affects the overall system performance. If V is too small, the quantization process loses too much information which decreases the segmentation accuracy. If V is too large, the symbolic representation loses generalization and it is difficult to capture the patterns in activities. Also, the quantization process

needs more computation to calculate the distance between the input feature vector with each of the V centroids and drains the battery fast. In a working system, V needs to be manually optimized based on the system requirements, i.e. the minimum granularity of activity and battery life. In our experiments, we set $V = 200$ which can balance the power consumption and segmentation accuracy. The detail of parameter selection is discussed in Section 5.3.3.1.

5.3.2.3 Change of Activities

When a user changes his/her activity at time t, there should be a significant change from behavior text string $[t-w, t-1]$ to string $[t, t+w]$. Here w is the window size that controls the granularity of detected activities. We refer to w as the *segmentation window size*.

To compare the similarity between two behavior strings, we use the classic Vector Space Model (VSM) [52]. Each behavior string is represented by a vector over the whole vocabulary $\mathbf{b} = (b_1, b_2, \ldots, b_i, \ldots, b_V)$ where b_i is the weight of word i (term-weight) in documents. Here we want to partially model the temporal patterns between symbols. In addition to words, we also consider consecutive words (n-grams) in the string. Each behavior string is then represented as a vector in the n-gram space and b_i is the normalized frequency over 1-gram, 2-gram and up to N-gram.

The similarity of two behavior text vectors **b**, **q** is calculated by their cosine distance:

$$S(\mathbf{b}, \mathbf{q}) = \frac{\mathbf{b} \cdot \mathbf{q}}{||\mathbf{b}||\ ||\mathbf{q}||} \tag{5.1}$$

where $S(\mathbf{b}, \mathbf{q})$ (Equation 5.1) is the n-gram similarity between string B and Q. The lower the value of $S(\mathbf{b}, \mathbf{q})$, the more likely the user changes his/her activity in $[t-w, t+w]$. A threshold Θ was defined to determine the activity change. For single level segmentation on the smartphone, we empirically set $\Theta = 0.8$.

5.3.2.4 Hierarchical Activity Segmentation

An advantage of using this activity segmentation framework is that the window size w can be used to control the granularity of segmented activities. Since the length of activities varies in real

life, we can use different window sizes to generate hierarchical activity segmentation: After getting the segmentations in level one with a large segmentation window size, a smaller window will be used to identify fine-grain activities recursively on each of the generated segment (Figure 5.11). The hierarchical activity segmentation will enable the user to view and annotate activities in different granularities, he/she can "zoom" in or out to change the granularity of activities.

Figure 5.11: An example of 3-level top-down hierarchical activity segmentation with $w_1 = 20$minutes, $w_1 = 10$minutes and $w_1 = 5$minutes.

In our experiment, we implemented a four level hierarchical activity segmentation using segmentation window sizes of 30, 20, 10 and 5 minutes.

We discover that the activity change "peaks" (similarity lower than average) identified by larger segmentation windows are also peaks identified by smaller segmentation windows but not vice versa; and activity changes over larger windows are smoother than smaller windows so setting one threshold for all levels is not a good approach. Instead of setting a hard threshold, we obtain threshold T_l for level l dynamically using the following equation:

$$\Theta_l = 0.9 \cdot (\max(\{S\}) - \min(\{S\})) + \min(\{S\}) \tag{5.2}$$

where $\{S\}$ are the similarities obtain from two neighboring segmentation windows of level l.

5.3.3 Performance Evaluation

We evaluate the accuracy of the activity segmentation and recognition by calculating the F1-Score of user's annotation with the ground truth.

Each identified activity in the system's annotation A is a triple of $< s, e, l >$ where s is the

starting time of the activity, e is the ending time and l is the label of the activity such as "walking". Similarly, we can represent each activity in the ground truth G also as a triple of $< s, e, l >$.

For each activity A in the system's annotation, if there exists a ground truth activity G such that $A_l = G_l$, i.e., the two activities have the same label, and $A_s = G_s \pm \Delta$, $A_e = G_e \pm \Delta$ where two activities have roughly the same starting time and ending time within the allowed margin Δ, then we consider A matches the ground truth activity G and is a true positive (Figure 5.13). With the precision and recall value calculated for each activity type, we can estimate the harmonic mean and report the F1 score. High F1 scores indicate the system's segmentation/label matches the ground truth.

5.3.3.1 Impact of Vocabulary Size V and N-gram Size

Table 5.7 shows the system power consumption[5] with different vocabulary size.

$V =$	**50**	**100**	**200**	**300**
Recognition $F1$	0.33	0.45	0.57	0.49
Power(%)	41	45	51	63

Table 5.7: The impact of V to the system, in terms of power consumption, overall recognition performance. N is set to 3 in these experiments.

Overall, recognition accuracies increase when V increases. It reaches the max when V is 200 and then drops slightly when $V = 300$. It could be that when V is too large, the system loses generalization and the trained activity recognition model over fits the training data.

The power consumption increases significantly when V increases. Because larger V requires more computation and CPU is one of the major sources that consumes power.

By fixing $V = 200$, we test different size of N (1, 2, 3, 4) for n-grams.Table 5.8 shows that when N reaches 3, activity recognition performance does not improve any more. The power consumption doesn't seem to have very significant change because the additional computation needed with larger

[5] Power consumption is measured using Android's built-in power measurement tool under "Settings". The percentage is recorded on the same device, Samsung S4, after a full charge

N is trivial compared to the increase of V.

$N =$	**1**	**2**	**3**	**4**
Recognition $F1$	0.51	0.51	0.57	0.55
Power(%)	48	48	49	51

Table 5.8: The impact of n-gram size N to on power consumption and activity recognition accuracy.

5.3.3.2 Impact of Phone Position in Activity Recognition

We run experiments to study whether the mounting position of the phone has any impact on the activity recognition accuracy. As shown in Figure 5.10, participants carry two phones at the same time, one in the pants pocket and another one mounted on the upper arm. Figure 5.12 shows the recognition accuracy of different activities with two mounting positions. Overall, that system performs better when the phone is attached to users upper arm. For activities where motion comes mainly from hand movements, l(e.g., "cooking" and "working on computer"), the upper-arm setting performances better. For activities where motion comes mainly from legs, such as "cycling" and "walking", pocket position performs better than the upper-arm setting. It makes sense because arms are relatively stable while riding a bike yet legs are moving more frequently and regularly. For activity "walking", the motion patterns of upper arm are more similar to other activities whereas as the motion patterns of legs (pockets) are more distinguishable.

5.3.3.3 Hierarchical Activity Segmentation vs. Smoothed HMM

Hidden Markov Models (HMM) have been widely used to model sequential data that have inherent structure as state transitions. We compare activity segmentation using the proposed hierarchical activity segmentation approach in MobiSens with a *smoothed single-layer Hidden Markov Model (HMM)*. After training a single-layer HMM using the EM algorithm over the unlabeled motion label sequence, we apply the learned HMM on the testing data (also a sequence of motion labels). During the decoding time, the HMM finds the state sequence that best explains the observed sensor

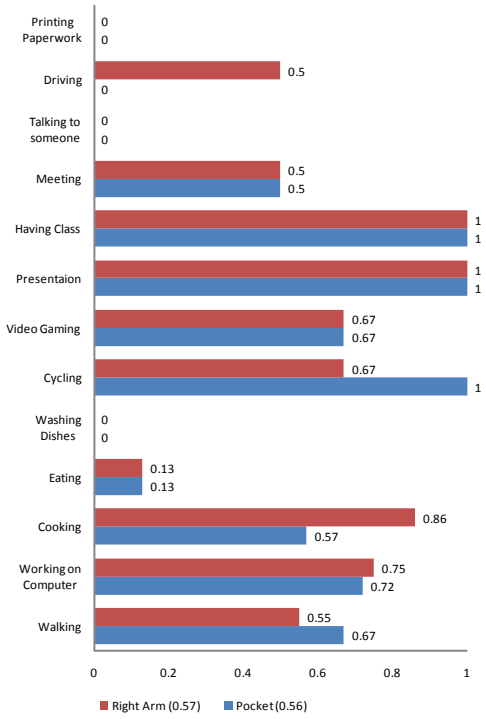

Figure 5.12: F1 scores of activity recognition accuracies for two phone mounting positions: pants pocket and upper arm. Activity recognition uses only accelerometers data. For pocket settings, the average F1 score is 0.57. For upper-arm setting, F1=0.56.

stream. For each input motion label, its corresponding HMM state becomes an "activity label". We apply a sliding window over the activity label sequence and use the majority label in window $[t - w/2, t + w/2]$ as the label for position t to smooth out noises. Segmentation is then done over the "smoothed" activity label sequence to identify positions where activities change.

The best configuration of HMM in our experiment has 10 states and the smoothing window size $w = 800$ which corresponds to 6 minutes in time. Averaged over all activity types, HMM performs worse than the hierarchical segmentation approach (Figure 5.14). In particular, HMM performs badly on high-level activities such like *"Having Class"*, *"Meeting"*, *"Working on Computer"* and *"Presentation"*. These activities are usually composed by multiple low-level activities and have

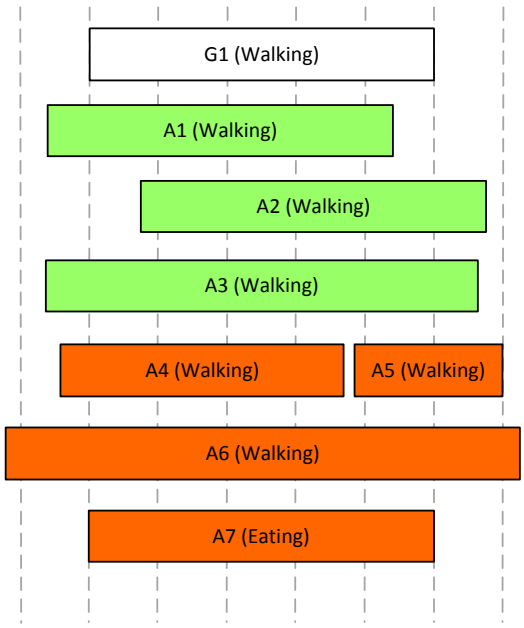

Figure 5.13: Ground truth labeled as G_1 with three true annotations(A_1-A_3) and four false annotations(A_4-A_7)

multiple motion patterns. A single-layer HMM models the lowest-level sensor readings and doesn't have the capability to merge these similar patterns to higher level activities.

5.4 ACTIVITY RECOGNITION THROUGH TIME SERIES ANALYSIS

In this experiment, we forgo the *language as action* principle to evaluate a different approach to activity recognition. Here, we study only motion data and by leveraging certain properties of this data we are able to derive a method to extract an *activity level*. Then by recognizing different activity classes of physical action (say standing, walking, or running etc.), we can build a system which compares two actions from the same class. This will allow for models which can discriminate between the minute differences in the way different people perform the same action, which can be used to identify between different users and detect unknown unauthorized users.

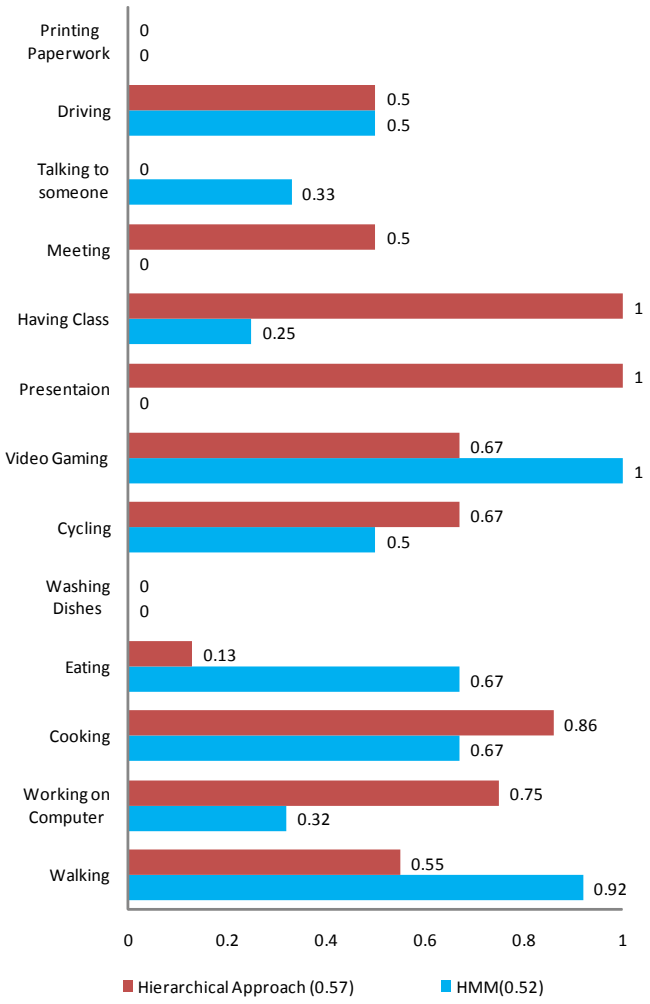

Figure 5.14: The F-Score of user annotation on Hierarchical activity segmentation result vs. HMM, on right arm dataset, motion only.

5.4.1 Experimental Setup

The experiment was set up such that a small group of people performed a set of ambulatory activities, standing, walking and running, with the same Nexus 5 phone in their pockets collecting accelerometer data. The experiments were conducted with a 50ms sampling rate, except for the set of walking experiments which were performed with both a 50ms and 20ms sample rate. We also used a separate dataset where the ground truth labels were not known. This dataset contained 110GB of data, after taking only the accelerometer part and discretizing the time series data into feature vectors for windows of size 128, we were left with a dataset of more than 2GB.

5.4.2 Activity Class Construction

We first test the approach to ambulatory activity recognition, explained in Section 4.3, by calculating the correct number of classification made on our dataset of 381 windows, each window containing 128 samplings from a triaxial accelerometer with a sampling rate of 50ms. In this dataset, there are 113 windows of standing data, 149 windows of walking data and 119 windows of running data, contributed by eight different users. The experiment is set up such that for each window, the DFT is extracted to calculate the $ActivityClass$ function. We choose $\alpha = 2$ and $\beta = \frac{1}{10}$ to scale both inputs to roughly the same range, and to account for the fact that the $\tanh$ function approaches 1 very fast so we want the inputs to be relatively small. Since we are looking to distinguish between three different classes, we sort the outputs of the $ActivityClass$ function and try to find the 2 largest gaps between these outputs which do not lie close to each other. For our particular dataset this happens at 0.1754 and 0.9050. We choose these two numbers as the boundaries between classes so we classify any window between the ranges $[0, 0.1754]$ as standing, $(0.1754, 0.9050]$ as walking and $(0.9050, 1]$ as running. Comparing this to the ground truths for this dataset, we correctly classify 359 of the 381 windows, giving a classification accuracy of 94.23%.

5.4.3 Feature Selection

To speed up the performance of any algorithms, and make the feature vector more relevant to the problem of classifying between users, we perform feature selection on the features discussed in the Section 4.3.2. We used data collected from 7 different users walking and each data sample is formed by taking the 11 features from the motion feature set for each of the X, Y and Z axes of the accelerometer, creating a 33 dimension feature vector for each window. Table 5.9 shows the rankings of each feature. From this experiment, it was apparent that the $avgDelta$, $numCrests$, $numPeaks$ features were not very useful and they are therefore discarded in the remaining experiments of this section.

5.4.4 User Identification Evaluation

To evaluate the effectiveness of these features on ambulatory data, we will first visualize how data from different people get mapped in feature space after dimensionality reduction using t-Distributed Stochastic Neighbor Embedding (t-SNE), and then perform classification with 10-fold cross validation on the dataset.

We collected the data by having a small group of people perform a normal walk across a hallway. The experiment was conducted with two sampling intervals, one of 20ms and another of 50ms, and the data collected was divided into windows of 128 samples. Feature vectors, based on the features identified in the feature selection section, were created for each axis of the accelerometer forming a feature vector of size 24 for each window. For the t-SNE visualization, the features were also log-normalized. In the case of the 50ms experiments, a group of 7 people participated, each contributing a different number of windows for a total of 149 data points. For the 20ms experiments, a group of 3 people participated, contributing a total of 211 data points.

Figures 5.15 and 5.16 plots the result of applying t-SNE dimensionality reduction on our dataset to reduce it to 2 dimensions. Each mark represents a window of data in the 2-dimensional space. From these figures, it can be seen that there does seem to be natural clusters, but there is also some

Ranking	Feature
1	RMS_x
2	$RMSE_x$
3	max_y
4	RMS_z
5	$RMSE_z$
6	sma_y
7	max_z
8	sma_z
9	min_y
10	rcr_x
11	sma_x
12	max_x
13	RMS_y
14	$RMSE_y$
15	ttp_z
16	$numPeaks_x$
17	ttc_y
18	ttc_x
19	$numCrests_x$
20	min_x
21	rcr_z
22	min_z
23	rcr_y
24	$avgDelta_x$
25	ttc_z
26	$numCrests_z$
27	ttp_y
28	ttp_x
29	$numCrests_y$
30	$avgDelta_y$
31	$numPeaks_z$
32	$avgDeltas_z$
33	$numPeaks_y$

Table 5.9: Feature Rankings

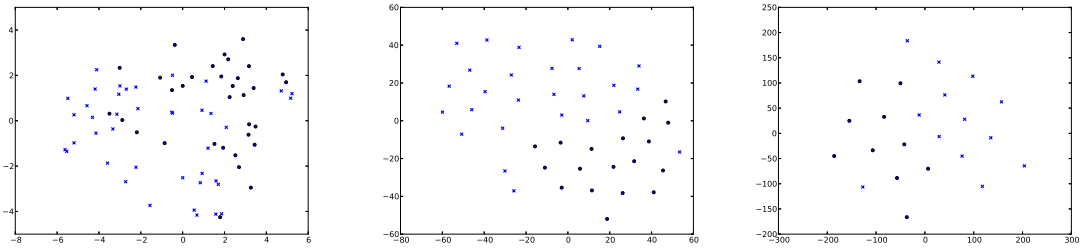

Figure 5.15: t-SNE visualization of a subset of the data collected with a 50ms sampling rate

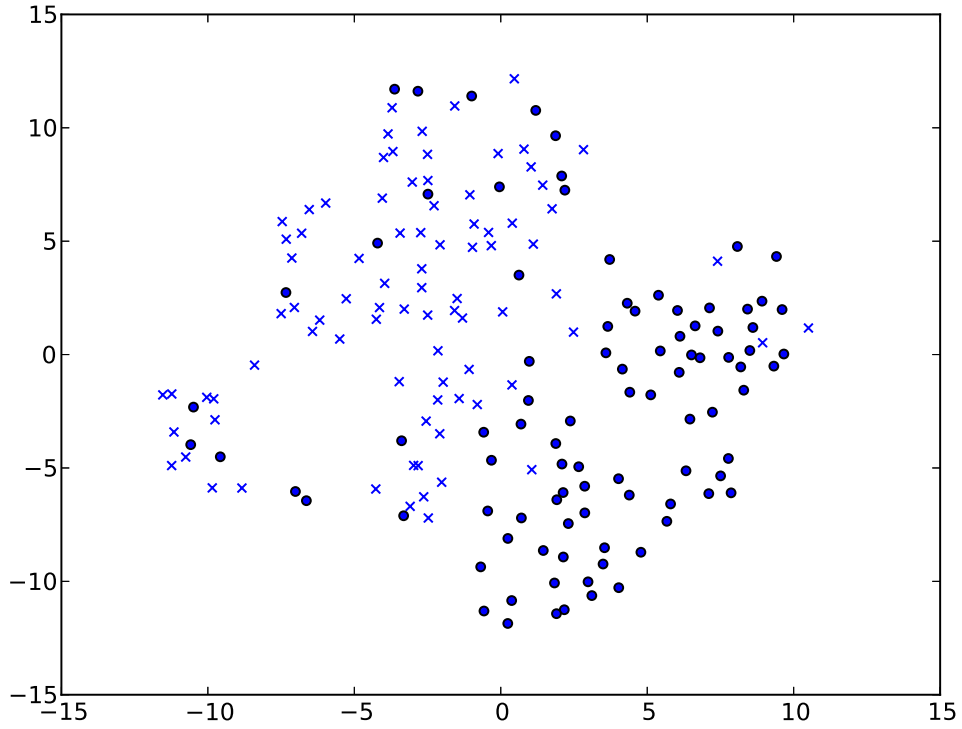

Figure 5.16: t-SNE visualization of a subset of the data collected with a 20ms sampling rate

	TP rate	FP rate	ROC area	Precision	Recall	F-Score
J48 Decision Tree	0.664	0.076	0.825	0.670	0.664	0.664
Sequential Minimal Optimization	0.832	0.047	0.941	0.835	0.832	0.817
Multilayer Perceptron	0.799	0.050	0.941	0.800	0.799	0.792
Naive Bayes	0.732	0.071	0.916	0.750	0.732	0.730
Bayes Net	0.722	0.048	0.930	0.779	0.772	0.773

Table 5.10: 10-fold cross validation of data collected with 50ms sampling rate

	TP rate	FP rate	ROC area	Precision	Recall	F-Score
J48 Decision Tree	0.886	0.063	0.932	0.888	0.886	0.887
Sequential Minimal Optimization	0.915	0.039	0.946	0.920	0.915	0.916
Multilayer Perceptron	0.924	0.038	0.979	0.927	0.924	0.925
Naive Bayes	0.839	0.092	0.919	0.837	0.839	0.827
Bayes Net	0.882	0.055	0.971	0.891	0.882	0.884

Table 5.11: 10-fold cross validation of data collected with 20ms sampling rate

overlap in the data. This can be explained by the both the data loss inherent in dimensionality reduction and the fact that even though most of the time when walking, people will exhibit behavior that is common as a group, but different to their previous normal behavior, such as when a person turns a corner.

Now we look at how these features perform when doing actual classification. Tables 5.10 and 5.11 list the results of running the dataset through some common classification algorithms. We can see from these results that the above feature set performs quite well in identifying the original users. Most of the algorithms will produce an accuracy within the 70% to 80% range for the 50ms sampling rate and 80% to 90% range with 20 ms sampling rate, with both high precision and recall. What is interesting to note is that the recognition rate for different users vary modestly, showing that some users are easier to recognize than others. This may be due to the fact that different users contributed different sizes of data, so for some users the algorithms may not have enough to data to work with.

We've so far only looked at user identification when we know beforehand the set of actions that have been performed. However it would be much more useful to be able to do user identification

ActivityLevel Range	TP rate	FP rate	F-Score
$[0.9, 1)$ Bayes Net	0.794	0.013	0.81
$[0.9, 1)$ SMO	0.785	0.03	0.761
$[0.8, 0.9)$ Bayes Net	0.7	0.016	0.714
$[0.8, 0.9)$ SMO	0.671	0.041	0.635
$[0.7, 0.8)$ Bayes Net	0.634	0.024	0.643
$[0.7, 0.8)$ SMO	0.593	0.048	0.551
$[0.6, 0.7)$ Bayes Net	0.539	0.034	0.546
$[0.6, 0.7)$ SMO	0.527	0.056	0.486
$[0.5, 0.6)$ Bayes Net	0.487	0.045	0.496
$[0.5, 0.6)$ SMO	0.488	0.091	0.429
$[0.4, 0.5)$ Bayes Net	0.434	0.056	0.429
$[0.4, 0.5)$ SMO	0.41	0.106	0.343
$[0.3, 0.4)$ Bayes Net	0.378	0.068	0.369

Table 5.12: User classification on unknown activities

on data which either don't have activity labels, or aren't based on some already known activity. Now, we explore using the activity recognition process described in Section 4.3, on a much larger dataset, to create arbitrary classes of similar activities and evaluate the effectiveness of performing user identification on these groups.

For this test we use the second, much larger dataset, consisting of more than 2GB of feature vectors of unknown activities. The feature vectors were then divided into 10 activity classes based on the output of the $ActivityClass$ function, with a class for each of the ranges $[0, 0.1)$, $[0.1, 0.2)$, $\ldots$, $[0.9, 1)$. We trained and tested some classification algorithms with 10-fold cross validation on these classes to determine if it was possible to do user classification without knowing the ground truth activity labels. The results are shown in Table 5.12.

For all tested classes, this approach to user identification performs reasonably well compared to randomly guessing the user which should only yield a true positive rate of 0.04. An obvious pattern in these results is that the accuracy of the classification algorithms decrease as the output of the $ActivityClass$ function decreases. This can be intuitively explained if we think of the $ActivityClass$ function as a function which measures the *level of motion* within any window. Now,

User	Accuracy	TP+TN	FP	FN
User 1	91.4%	117	4	7
User 2	89.4%	118	3	11
User 3	83.7%	113	2	20

Table 5.13: Anomaly Detection Metrics

for those windows with less activity, there are less features which are able to distinguish between different users. It is for this reason that the classes with lower motion levels weren't classified, they would predictably yield much lower accuracy and because most of the time your phone is stationary, (eg. laying on a table), these classes were much bigger and would've taken significantly longer to classify.

5.4.5 Anomaly Detection Evaluation

We continue this experiment by evaluating how this activity recognition approach performs when doing anomaly detection, a core part of *SenSec*'s user authentication functionality. We evaluate the approach mentioned in Section 4.3.4 by taking 3 of 7 users in the 50ms walking dataset and building gaussian models for them on half of their data. Then we take the rest of the dataset and test it on these models with an epsilon of 1.0×10^{-10}. The primary objective here is for this anomaly detection technique to act as a user authentication system, hence we want to maximize the detection rate, consisting of true positives (anomalies which were successfully detected) and true negatives (normal usage which was successfully detected), while minimizing false positives (since we don't want to incorrectly classify the primary user as an anomaly) and false negatives (marking anomalous usage as normal usage). For this experiment, we will consider the true positive + true negative, false positive and false negative metrics.

Table 5.13 shows promising results, demonstrating both high accuracy and a low false positive rate. However, again, we are using data for which there is only a single activity being performed. Now, we tackle the problem of anomaly detection for windows of data which have undetermined activity (so for data without labels or for previously unseen classes). Using the larger dataset, the

ActivityLevel Range	Accuracy	TPR	FPR
$[0.1, 0.2)$	88.2%	0.898	0.812
$[0.2, 0.3)$	85.4%	0.869	0.811
$[0.3, 0.4)$	83.7%	0.851	0.791
$[0.4, 0.5)$	88.9%	0.903	0.803
$[0.5, 0.6)$	89.3%	0.905	0.753
$[0.6, 0.7)$	90.3%	0.915	0.706
$[0.7, 0.8)$	90.7%	0.918	0.691
$[0.8, 0.9)$	91.9%	0.932	0.760
$[0.9, 1)$	95.8%	0.976	0.815

Table 5.14: Unknown Activity Anomaly Detection

experiment is set up so that for each class c and user u, a model is built using half of u's data from class c. Then for each class we average the TPR, FPR and $accuracy$ statistics of all users with data in that class. In this experiment, we empirically set epsilon to 5.0×10^{-4}, to maximize the accuracy and minimize the number of false positives. An interesting note here is that this epsilon differs from the one used in the above anomaly detection experiment by many orders of magnitude. The reason for this can be explained by the fact that this dataset is so much bigger, with presumably many more windows of different types. This translates to a gaussian model with a larger standard deviation which will increase the probability of new data being generated by the model, hence we have to set a much larger epsilon. The results of this experiment are detailed in Table 5.4.5.

Although the results show very high accuracy with an average of 89.36% across all classes, unfortunately the false positive rate is also extremely high with an average of 0.771. This means that the primary user will get incorrectly flagged around $\frac{3}{4}$ of the time which is generally not acceptable for anomaly detection systems. This shows that we will still need to combine this with other methods or factors of authentication, to provide increased user experience.

5.5 LOCATION AUGMENTED PASSIVE AUTHENTICATION

As shown in the previous section, performing user authentication through anomaly detection based solely on accelerometer data presents a very high false positive rate. In this experiment, we

seek to mitigate this problem by including location as a factor in the anomaly detection model.

5.5.1 Experimental Setup

The experiment is based on data collected from 25 Android phone users. The Android application took both accelerometer and GPS samples, with an accelerometer sampling rate of $50ms$ and GPS sampling rate of $1s$. Each sample was logged with its corresponding timestamp. Because the application may not always be operating at highest priority (most common usage shows it will operate at background priority most of the time), these sampling rates may not be very precise. With the collected data, we further processed the location trace by parsing the GPS coordinates and placing them into distinct areas. Specifically we divided latitude and longitude space into 0.002×0.002 grids as done in [14]. This resulted in grids of size $200m \times 200m$ in the North America region where most of our participants come from.

5.5.2 Location Data Analysis

First, we will use the location traces in the above dataset to determine if we can confirm the fact that the majority of users spend most of their time in a few locations. For each user, we calculate the amount of time they spent at each recorded location. As mentioned above, the collection of data may not be particularly consistent, since we cannot control the thread priority of the application, whether GPS tracking was turned on, whether the application was killed or not and a variety of other factors. We manage this by defining a period of time, of five minutes, where if two consecutive samples were recorded at the same location within this period, we would assume that the user was at this location for the full duration between the two samples. If, however, two samples s_1, s_2 which indicate the user was in location A were recorded outside of this time interval, we assume that the user left location A at s_1 and came back at s_2. This is to prevent the scenario where a user goes to work, turns on the GPS because they need it for something work related, turn it off before they go back home, and turn it back on again at work the next day. We don't want to calculate them as having

Place	Mean	Standard Deviation
1	50.66%	16.33%
2	15.14%	9.70%
3	6.76%	3.44%
4	3.73%	2.51%
5	3.03%	2.34%

Table 5.15: Percentage of time spent in the top 5 locations across 25 users

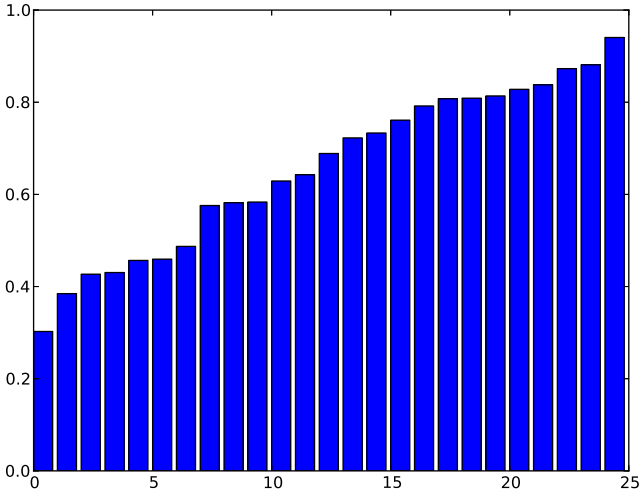

Figure 5.17: Percentage of time in top two locations

been at work for the whole night because two subsequent GPS samples say they were at the same location.

Using this strategy, we were able to collect an average of 1428 hours of location data for each user, roughly 60 days worth.

Table 5.15 shows the distribution of time spent between in the top 5 places. It can be seen that for the majority of people, a large amount of time (approximately 50%) is spent in one location. This is promising for this study because it implies we can indeed use location data as a factor when performing user authentication, a conclusion shared by the findings in [14]. The smaller the percentage of time the primary user spends in his/her less frequented areas the more the likelihood

of such an event occurring being an anomaly. Hence such a distribution of time spent by our participants gives the combined advantage of increasing true anomaly detection rates and increasing user experience, since the larger the amount of time the user spends in more frequented 'safe' areas the less the number of false positive triggers.

We can see a more detailed and precise picture of where each user spends the majority of their time in Figure 5.17. As shown in the graph, a good majority, 18 of the 25 users spend 55% or more of their time at their top two most frequented locations, presumably home and work/school.

5.5.3 Location as a Passive Authentication Factor

Here we evaluate the approach detailed in Section 4.3.5 by augmenting the previous anomaly detection system which is based on segmenting motion into different activity groups with location data to see if we can improve the accuracy of the anomaly detection system. In combining the motion and location data, we need some way of assigning each motion a discrete location. In this experiment the motion data is in the form of a feature vector. These feature vectors were created as before, taking a window of 128 samples and calculating the motion features from this window. Each feature vector is assigned a timestamp of when the first sample in its window was taken. We assigned all feature vectors to the closest location with a timestamp that is both less than and within 5 minutes of the timestamp of the feature vector. We then discarded all those feature vectors for which there did not exist a recorded location 5 minutes prior to its timestamp. This resulted in ~ 2.1 million motion feature and location pairs, with each feature vector accounting for about 6 seconds of data.

The participants who collected that data came from a variety of backgrounds. While most of them were situated in the United States, there were also some people from other continents. It is important to note that this approach does not take the distance of frequented locations into account, since each location is a different discrete entity, so given some user on the west coast of the United States, the probability that some action was performed on the east coast is the same as the probability

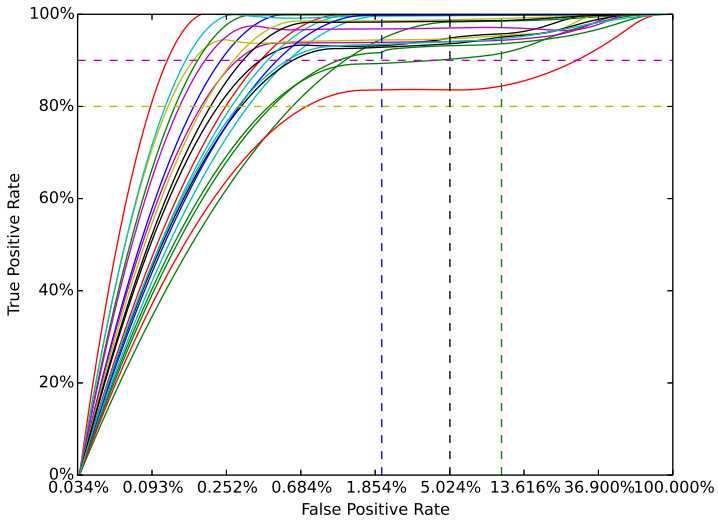

Figure 5.18: ROC Curve

of the same action performed in Antarctica, given model M has seem the same number of each action class in both locations.

To evaluate the suggested approach, we performed the following experiment. For each class c and user u we randomly selected half of u's samples for class c and calculated the gaussian distributions for each feature in the feature set. Then we trained the model on all of u's samples for activities that weren't in class c and the previously randomly selected set of feature vectors, by recording a count for the number of samples in each location, in each class. We then evaluated this model by taking the remainder of u's feature vectors in class c combined with every other user's feature vectors of class c and calculated $P_m(A)$. We then selected a threshold and labelled all those feature vectors with probability less than the threshold an anomaly.

We chose a number of different thresholds in testing the approach. Figure 5.18 shows shows the receiver operating characteristic curve of the different users. We calculated this curve by summing the true positive, false positive, false negative and true negative results for all classes together for each user. Although each user has a different number of feature vectors for each class, we don't

Range	Accuracy	TPR	FPR
$[0, 0.1)$	95.78%	0.9606	0.1990
$[0.1, 0.2)$	97.57%	0.9801	0.3311
$[0.2, 0.3)$	97.64%	0.9810	0.3335
$[0.3, 0.4)$	97.70%	0.9810	0.2997
$[0.4, 0.5)$	98.05%	0.9849	0.3301
$[0.5, 0.6)$	98.07%	0.9858	0.4108
$[0.6, 0.7)$	97.54%	0.9815	0.3875
$[0.7, 0.8)$	97.33%	0.9811	0.5255
$[0.8, 0.9)$	96.24%	0.9714	0.4607
$[0.9, 1)$	97.16%	0.9832	0.4933

Table 5.16: Anomaly detection for each class

apply any weighting when adding the classes together because we assume that the distribution of feature vectors among classes for each user is representative of future usage. From this, we deduce that once the threshold goes past a certain point, we are able to see very promising results. In fact, before we even reach a false positive rate of 0.2, the true positive rate exceeds 0.8 for all users.

Table 5.16 shows the performance of the anomaly detection system for each class when the threshold is set to 1×10^{-30}. Comparing these results to those of the anomaly system without location as a factor, we see that the accuracy has improved across the board. What is interesting to note is that the decrease in false positive rate is much higher for the lower ranges than for higher ranges. This seems to be indicative of the fact that the previous anomaly detection system was not able to distinguish between user classes on the lower activity class ranges based solely on motion alone. This can be seen intuitively since in low activity windows, for example of a person standing, there are less distinguishing motion features. By introducing location data, a factor independent from motion, we allow the system to recognize a person standing in North America is very different to a person standing in Asia, and thus we are able to dramatically lower the false positive rate for these range of activities.

5.6 USER IDENTIFICATION BASED ON INTERACTION METRICS

The experiments thus far, have all considered time-series data. However, *SenSec*'s user interaction modeling component takes into account how a user's use of the soft keyboard also contributes to their behaviometric profile. As such, this section explores the usages of the spatial modeling techniques described in Section 4.4 to supplement our time-series models. We developed a soft-keyboard application that can *passively authenticate* users in real-world environments based on micro-behavioral metrics modeling. We analyze not only how rapidly a user types, but also a variety of soft-keyboard specific micro-behavior features. These include *where* on the key the user pressed, how much the user *drifts* over the course of a keypress, and the *orientation* of the phone.

Using this data, plus a variety of statistical tools, we can generate a *certainty score* of whether the user's phone is in a stranger's hands. This score, when combined with a larger application and decision engine, will be used in future work to block access to applications or ask for additional authentication information when an non-authorized user is detected. For this demonstration we leverage other users' keypress data to help train the a micro-behavioral model, but we envision a system where all data is collected and stored on the phone and a model is trained offline separate from other user data.

5.6.1 Experimental Setup

5.6.1.1 Soft Keyboard Application

One of the advantages of the Android platform is the availability of lower level sensors and View-level data. Compared with iOS or other mobile operating systems, Android exposes more touchscreen data, allows for greater application configuration, and enables developers to design a huge variety of alternative input devices. We took advantage of these features to build an trial *input method editor* (IME) to gather example user data. This application (implementing a custom KeyboardView for raw touchscreen access) records absolute pointer positioning and size/pressure data for each finger or stylus. On most Android phones, there can be up to 4 or 5 touchscreen events

per keypress, giving us an abundance of data with which to work. By implementing a custom KeyboardView class, we can capture all of the MotionEvents that Views receive from the touchscreen, rather than just KeyEvents. Additionally, features can be analyzed on a per-key and a holistic basis to identify patterns with fine-grained resolution. If the regular user always presses spacebar in the bottom right and a keypress occurs in the bottom left, the application can flag that with a different significance than a slightly atypical press of the 'q' key, which may not have nearly as much training data.

To ensure there are a sufficient number of test users and to encourage natural typing behaviors, our application was designed to closely mirror the features of other Android keyboards: Standard QWERTY layout, typical enter, shift, delete and 'symbols' key locations, and a typical key size are all familiar attributes. Additionally, as a promotional feature—and one that has received very positive reviews—we included 6 configurable 'hot-strings' that can output frequently typed strings such as emails, mailing addresses, or phone numbers. We feel that spending time on the user experience side of application development was a worthwhile investment for the increased user retention and more comfortable typing environment. If users have to spend a long time adjusting to the new system, their early data can be misleading or even counterproductive when developing a training model.

5.6.1.2 Data Collection and Modeling

A short recruitment drive [6] and three week long collection period resulted in a group of 13 active users contributing a total of roughly 86,000 keypresses (with highly variable contributions from each user) and about 430,000 touch data points. This gives some indication about how much data can be collected in a short period of time, especially when gathering keypress information in all contexts, not just from passwords or controlled phrases. To model the data, we utilize the generative

[6] User studies were conducted with the approval of IRB application HS11-094, "Human Behavioral modeling using Mobile Sensors" from the IRB Board at Carnegie Mellon University. Recruitment were through on-campus advertisements with symbolic incentives.

or discriminant models developed in Chapter 3.

5.6.1.3 Training System Design

When designing learning algorithms, it is very important to ensure good training practices are followed [53]. Model training, validation and testing are important parts of the process and should be considered carefully. When analyzing, user data is split into five sections, the first four randomly sampled from the first two weeks of collection. The final set of testing data is from at least 3 days after the rest (to test in user environments independent from training data). Our results are generated from the 'final testing data.'

1. Training data for primary generative or discriminant models (50% $\simeq$ 3000 keypresses from user, 2000 from each of 3 other random users)
2. Cross validation for primary model (15%)
3. Testing data for model and training data for key-frequency scaling (10%)
4. Cross-validation for key-frequency scaling (10%)
5. Final testing data (15%)

5.6.2 Performance Evaluation

5.6.2.1 Variability in Keypress Location

Figure 5.19 displays differrent patterns that develop for users on specific keys. 'User 1' is almost always in the bottom right, while 'User 2' is usually right around the vertical center of the key but with wide horizontal variability. Observe also that there are two area of concentration for 'User 2.' Based on some contextual knowledge about that user's behavior we found that the left concentration is from the right finger, while the right concentration is from the left finger. **Figure 5.20** compares 5 different users' press locations on a single key, holding 'User 1' constant for reference.

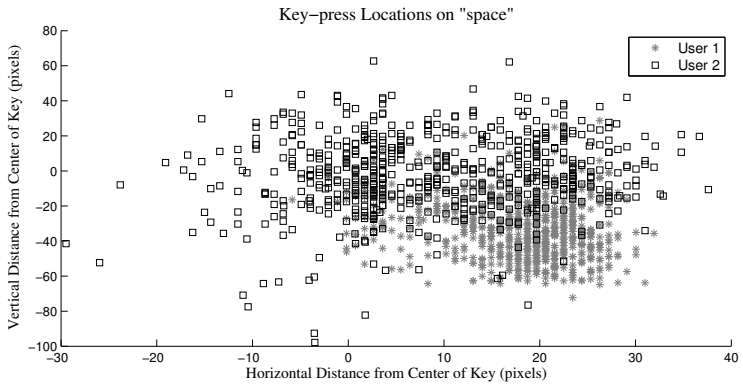

Figure 5.19: Two-User Comparison of Keypress Locations on 'spacebar'. Observe how users tend to clump, and how 'User 2' presses in seemingly distinct areas with their left thumb versus right thumb.

With the key press location data, we applied a *bivariate Gaussian distribution* for each key. The formula for bivariate covariance is in **Equation 5.3** [44].

$$cov = \frac{1}{n}\sum_{i=1}^{n}(w_i - \bar{w})(w_i - \bar{w})', \tag{5.3}$$

5.6.2.2 Keypress Length

While experienced desktop users tend to type faster than inexperienced ones [26], mobile users type in a much wider range of circumstances and at a much wider range of rates. An interesting exception was found with the dual-purpose period ('. ,') key. To access a comma, users can hold the period key for 500 ms. The distributions of hold times around this value had some discriminatory merit, perhaps because users had a different sense of how long it really took. See **Figure 5.21** for a visualization.

5.6.2.3 Drift

Drift was found to have different angle distributions depending on the user. Due to touchscreen noise, drift is only counted if a finger's last contact is more than 4 pixels from point of first contact. The numerical cutoff is referred to as the *drift threshold*. See **Figure 5.22** for a visualization of drift

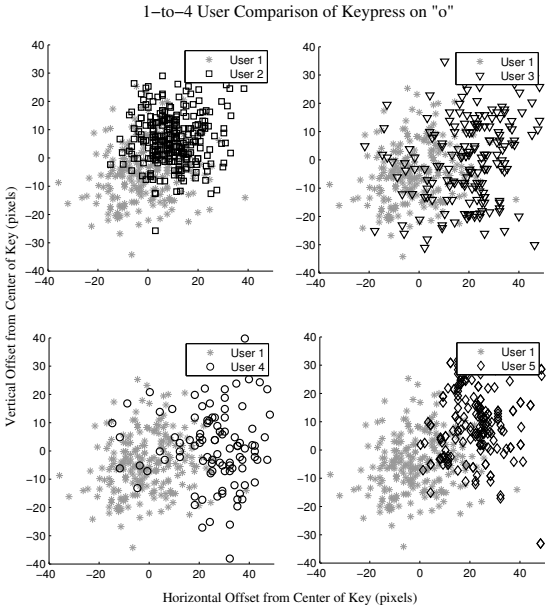

Figure 5.20: Quad-comparison of Keypress Locations on ‘o’. ‘User 1’ and the axes are static. This shows how different users have (sometimes widely) varying spreads and centers.

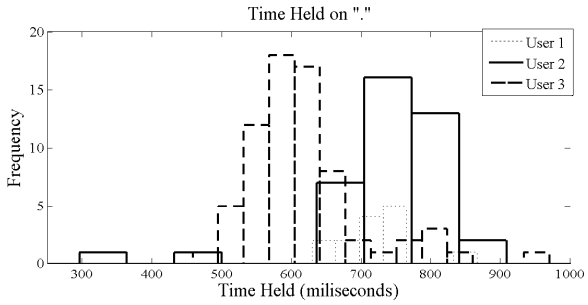

Figure 5.21: Comparison of hold time between users on the period key.

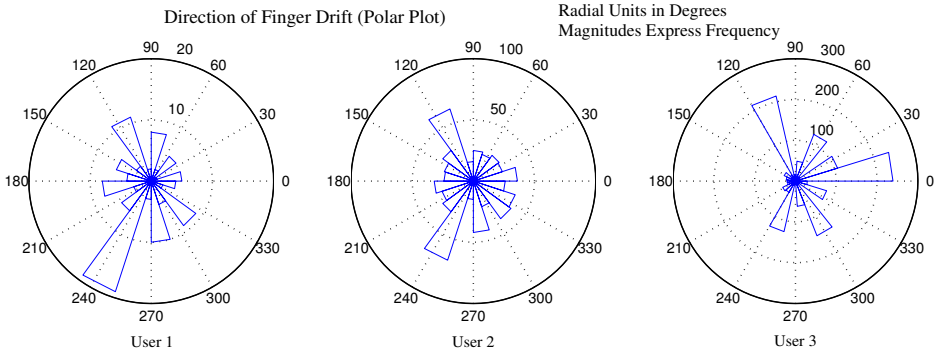

Figure 5.22: Comparison of finger *drifts* between users. The center of each circle is the normalized location where the user first pressed down. The angles describe which direction the user drifted (but not how far). The ray lengths are the frequency of drifts in that direction.

angle distributions where the origin is normalized location of first contact. There are a very similar number of tap samples from each user. Observe how 'User 3' drifts down to the left much less often (as a percentage of total drifts) than the other two in the figure. Observe, however, that 'User 3' has a much higher total number of drifts above the threshold as the other two (the number by each ring is frequency).

The data for *drift* is by no means normally distributed, but tends to clump into a number of distinct areas for particular users. By grouping angles into a finite number of buckets (akin to the rose histograms below), we can calculate the historical probability of a particular drift direction against which to compare new data. Our formula for such analysis is below:

$$P(drift|user;\theta) = \alpha_{drift} * P(anyDrift|user) * P(\theta|user) \tag{5.4}$$

5.6.2.4 User Identification Results

5.6.2.4.1 Success and Error Metrics

When evaluating the success of a model or a process, it is important to define the metrics used. Our primary objective is to identify 'non-authorized users' quickly, accurately, and repeatably. This is essential in creating a functional authentication system. Our other main objective is to minimize

how often the primary user is flagged as another user—an inconvenience. For these metrics we define *Detection Rate* as the frequency of successfully detecting 'non-authorized users', *False Rejection Rate* (FAR) as the frequency of flagging the primary user as 'non-authorized' and the *False Acceptance Rate* (FAR) as the frequency of failing to flag a 'non-authorized user' as such [53].

5.6.2.4.2 Attack Detection

Our discriminant algorithm trained on multiple users performed well in testing, detecting an median of 67.7% of simulated 'non-authorized users' within 5 keypresses. Our False Acceptance Rate (FAR) was 32.3% and we had a False Rejection Rate (FRR) of only 4.6%. For longer input sessions of 15 keypresses, our detection rate rose to 86.0% with a FAR of 14.0% and a FRR of only 2.2%. Models were trained with 3000 keypresses from the 'primary user' and 2000 from each of 3 other users. It was then tested against 550 'primary user' keypresses and 500 'non-authorized user' keypresses from a variety of other users. A few keypresses from each user were ignored due to lack of training data for those keys (a potential concern for analyzing symbol-heavy passwords). The 'non-authorized users' in the testing sets were not used for training data. The test data from the primary trainee was from at least 3 days after the training data to ensure independent (though not assuredly distinct) environments. The performance matrix and receiver operating characteristics (ROC) curves (for a number of different users) are in **Figure 5.23**.

As seen in the ROC curves (a subset of all tests), there is a sizable spread to the recognition rates between users. This needs to be further explored, however it indicates that some users are easier to tell apart than others. Running a model trained on 'User 1' with two different sets of example 'non-authorized users' can generate very different detection rates.

Our results do suffer from a significant hole, however. Because we strip timestamp information from the data before logging, we cannot recreate exact strings to use as testing data. We counter this as best we can by creating each test string with data from a single log file (which typically contain only a few minutes of typing data). Further testing needs to be done on data known to be in ordered

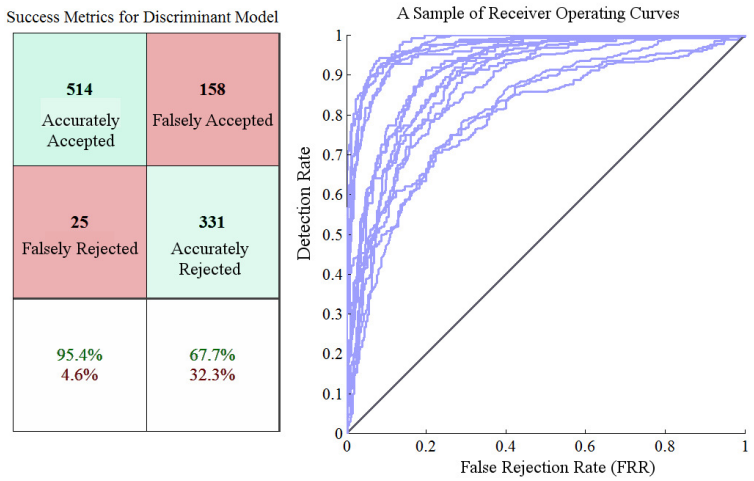

Figure 5.23: Testing performance and receiver operating characteristics for discriminant model. 67.7% of simulated 'non-authorized users' were caught within a 5 keypress sliding window with a False Acceptance Rate of 32.3% and a False Rejection Rate of only 4.6%. The ROC curves represent a subset of all tests indicating variability of detection rates between users.

strings, however actual keylogging will only be acceptable in more limited environments.

Going forward, our generative model can use these results as a benchmark against which to measure it's own success. More work will be required to develop robust, yet not overly sensitive decision engines that need not be trained on other users' data.

5.7 SUMMARY

In this chapter, we evaluated the effectiveness of our proposed approaches and models through several experiments. We started by focusing on the language as action principle demonstrating that we can indeed use this for location and mobility modeling, user identification, and activity segmentation and recognition. Then we demonstrate our second model for user identification and authentication, one which focuses on motion data as input. We augment this model with location data and demonstrate that by doing so we are able to increase accuracy and decrease the false positive rate. Finally, we evaluate our spatial model for user interaction by building and testing a soft keyboard capable of passively authenticating users.

CHAPTER 6

DISCUSSION

In the previous chapter, we describe the experiments that we conduct for the particular use scenario of Behaviometrics, *SenSec*, and how we evaluate the methodologies, models and algorithms we discussed in Chapter 4. During these experiments and evaluations, we have encountered several practical issues and/or limitations that are either related to the BehavioMetric models or the application scenario itself. We classify the issues and limitations into the following categories:

1. Usability
2. Data Privacy and Security
3. Power Consumption

In this chapter, we will discuss these issues and their solutions. We will also present some limitations to our work and propose some future research directions.

6.1 USER INTERFACE AND USER EXPERIENCE CHALLENGES

6.1.1 Usability Issues from Alpha Testing

We developed and released *SenSec* version 1.0 based on motion behaviometrics. Figure 6.1 shows the UI. There are 3 major views: Application Protection Configuration View (Figure 6.1.a),

Training and Threshold Settings (Figure 6.1.b), and Passcode prompt (Figure 6.1.c). The UI follows the older version of Android UI patterns with a darker background and standard UI elements without a lot of changes in visualization and interaction flows. One particular modification we have done is the *scrabbled passcode* layout, as shown in Figure 6.1, where the numeric keys' locations are randomized. The intention was to added extra overhead for attackers, making it harder to guess the passcode by looking at relative key stroke movements on the screen.

We started the alpha test in June 2012 and then released on the Google Play Store in October the same year. As mentioned in Chapter 5, our initial tests and experiments show an average 13% False Positive Rate or FPR. From a modeling perspective, this FPR may be reasonable. However, after we surveyed the participants of the experiments, we found that 75% of the users felt very annoyed when SenSec prompted them for the passcode, especially when it affected their normal operations of the device. In an extreme case, one participant complained that the system blocked him from answering important phone calls multiple times in a row because he was occupied with other tasks and could only use one hand to operate the phone. With the scrambled keyboard, it is not easy to enter the code correctly. Another participant mentioned SenSec sometimes prompted for the passcode even after she had successfully authenticated herself just a minute piror. Other participants shared frustrations in understanding how to use the application, especially the "training/testing" switch on the Settings view.

In the next section, we will illustrate the evolution of SenSec user experience, UX, based on this feedback.

6.1.2 User Interface and Interaction Evolution

A great mobile application should constantly adapt to user feedback. SenSec is not an exception. In SenSec version 2, we redesigned SenSec's user interface as well as its interaction flows based on the feedback we collected from alpha tests and initial release.

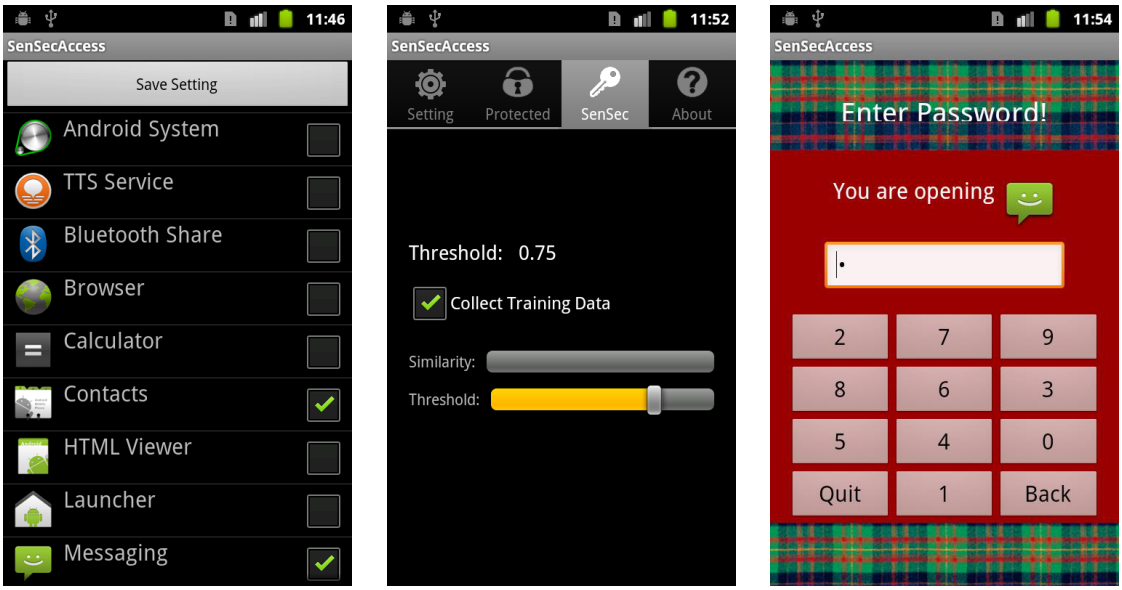

Figure 6.1: User Interface for SenSec Alpha

6.1.2.1 Models and System Flows

To address the aforementioned usability issues related to high FPR, we have made the following revisions in our system and interaction flows: 1) We adopt an online and adaptive model, in which we add the trace data shortly before a false positive to the training data and update the model. This change would eliminate the need for the training switch in the Settings view in such a way that the SenSec app will learn and adapt to a user's behaviometrics gradually and use the false positive samples (labeled by user's successful authentication) to update the model. 2) We also add logic to grant user a "Free Ride" period if they just successfully pass the authentication due to the detection of changes in motion behaviometrics. This is under the assumption that a user who just authenticated him or her self with the system should be controlling the device for a given period of time. To prevent a sudden event that could breach mobile device security, such a "Free Ride" period will end immediately if an abrupt context change is detected, such as high acceleration, sudden change of location, etc.

6.1.2.2 SenSec Tutorial

We added a tutorial in SenSec version 2. This tutorial will be triggered when user starts the application for the first time. As shown in Figure 6.2, the tutorial walks through the key elements on the SenSec app and shows the new user the standard usage flow.

6.1.2.3 Multiple Protection Options

We also added the options to use a sliding pattern in addition to the passcode when SenSec is prompting the user for authentication. Our intention is to let the user decide which applications need "softer" security, for which a sliding pattern would be adequate to protect, and which applications require strict authentication via passcode from a scrabbled keypad. The new UI for this change is shown in Figure 6.3.

6.1.2.4 Home Screen and Monitoring History

The "sense of security" is very important for security products [54] like *SenSec*. From the focus groups, survey and interviews of SenSec's potential users[1], we learned that it is critical to show the users what is under protection by the system and what is not. Therefore, we redesigned the home screen of SenSec to have a pie chart showing how many applications are secured (via key code), how many are protected (via sliding patterns) and how many are unprotected as shown in Figure 6.4.a. We also added the protection history as part of the home screen, Figure 6.4.b, to give user the behaviometrics, in form of certainty or sureness scores, over time and the threshold relatively to these scores.

6.1.2.5 Status and Notification

We noticed that users sometime need to turn off protection completely during certain time periods for various reasons. We added this flexibility following the standard Android UI design patterns: Start/Stop options in the Status Bar and Notification Area as shown in Figure 6.5. It not only pro-

[1] User studies were conducted with the approval of IRB application HS11-094, "Human Behavioral modeling using Mobile Sensors" from the IRB Board at Carnegie Mellon University.

Figure 6.2: SenSec Tutorial

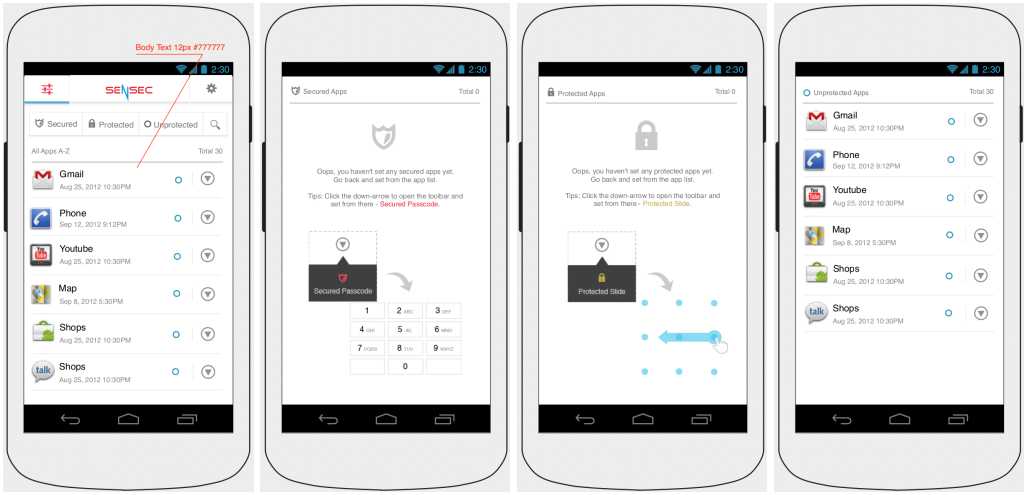

Figure 6.3: SenSec Protection Options: Secured, Protected via either Passcode or Sliding Pattern

Figure 6.4: SenSec Home Screen

Figure 6.5: SenSec Status Bar and Notification

vides a way to start or stop SenSec service, but also shows a summary of anomaly detection counts and the accuracy rate.

6.1.2.6 Sensor Fusion for Behaviormetrics

Also, in this version, we combined location metrics, interaction metrics along with motion metrics in the anomaly detection process. As we mentioned in Chapter 4, *SenSec* App adopts a simple hybrid model to combine behaviometrics from different sensory sources: the motion sensors are generating location argumented confidence score $S_{m,l}$, separated from the confidence score S_k of the user interaction metrics, i.e. software keyboard patterns. *SenSec* App will trigger user authentication prompt whenever either of the scores drop below a threshold. Due to the nature of these different types of sensors and the corresponding interaction flows, users will have different perceptions when an anomaly is detected and an authentication screen is prompted:

1. Reactive triggering: When user is trying to invoke an application either secured or protected,

the certainty score is computed and compared against the threshold. User are blocked from invoking such an application by the authentication screen if the score falls below the threshold.

2. Proactive triggering: When user is using the software keyboard, presumably within an already running application, if the certainty score, from interaction metrics, motion metrics and location metrics combined, drops below threshold, user will be blocked from continued use of the application until the authentication is successfully completed.

6.2 DATA PRIVACY AND SECURITY

The most critical factor that might hinder the SenSec applications is data privacy and security. For example, in the KeySens application, by collecting so much data about each touch event on the keyboard, we are able to compare a wide variety of features between users and develop an improved model for user authentication. However, much of this information is highly sensitive or confidential. We cannot expect users to willingly use a keylogger on their personal devices and we certainly do not want to expose scores and scores of key sequences in the unfortunate event of server compromise. Given that we do comparisons key-by-key, it was necessary to find an obfuscation strategy that allows us to know which key was pressed, but cannot let someone find out *when* it was pressed. Thus, we remove timestamp information from the keypresses and order them by a cryptographically secure random number generator (Java's SecureRandom class) so that the original sequences are not reorder-able [55]. This, combined with block-level encryption and large-batch HTTPS transfers to our server helps protect user data while allowing us access to the information we need.

From a platform point of view, we can rely on a hybrid cloud architecture in which personal sensitive data will be kept in a private cloud fully controlled by the user [56]. He/she opts for certain data to be exposed to a public cloud service to enable certain applications. Alternatively, a third party widget can be deployed by the user into the private cloud and only the final analytics results will be published. E.g., information consumers such as health monitor organizations; car

insurance companies and utility companies can develop application widgets to extract statistics from users' sensor data. A widget 1) declares what kind of information is going to be extracted from users' data (e.g., "this widget will estimate your average commuting distance from home to work"); 2) is downloaded and installed to the user's personal cloud once the user grants information access to the information consumer; 3) runs on the personal cloud and processes the raw sensor data using the data analytics API; and 4) uploads the extracted analytics to public cloud. Finally, the aggregated statistics will be used to derive behavior models for their own applications.

6.3 DATA AND POWER CONSUMPTION

Mobile devices are highly sensitive to power consumption considerations. Users expect that applications not only improve their mobile experience, but do so with limited impact on battery life [57]. Users are less willing to forgo longevity for common applications such as keyboards, than for immersive ones (such as games). It was observed from our users that the soft keyboard application was consuming less than 4 percent of the phone's battery life. For most, it did not even show up in the listing (alongside power hogs like Maps, Music—even Phone made the cut). Further analysis of current draw is needed, but power consumption is markedly low. This will likely increase, however, once the behavioral model computation is done directly on the handset.

In the *SenSec* App, we measure the power consumption due to data acquisition and transmission on a Google Nexus S running Android 4.0. With a polling rate of 4Hz, *SenSec* App collects 2MB of sensor data per hour, which is the major storage and power overhead. The following chart shows the power consumed by sensor (accelerometers, gyroscope and magnetometers) data collection (includes writing to SD card), GPS (using an activity aware power optimization algorithm) and wireless sensor data upload (using WIFI).

As we can see, sensor data collection (including writing to the SD card) and WIFI transmission makes up 65% of *SenSec*'s power consumption. After leveraging the activity aware GPS sampling scheme, the power consumed by GPS only takes 10% of the total power consumed by *SenSec*. If

we use a constant rate of 3 minutes per GPS request, this number will grow to more than 50%. The computation overhead brought by activity segmentation and recognition, as well as other system-wide power consumption also make up 17% of SenSec's power consumption.

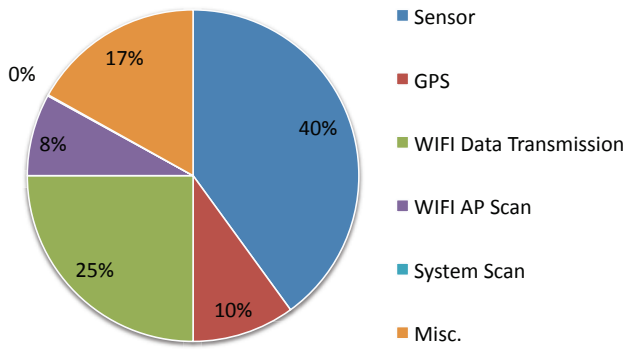

Figure 6.6: The power consumed by different module/functionalities of the SenSec mobile client. "Sensor" refers to physical sensors including accelerometers, gyroscope and magnetometers (digital compass)."System Scan" is the scanning of running applications in the system, which consumes very little power.

This experiment indicates that we should reduce the wireless data transmission via the mobile device to lower power consumption. In *SenSec* app, we have built the modeling and inference logic as discussed in Chapter 4 and Chapter 5, which does not rely on cloud services to perform user identification and casual authentication. However, we still upload all the raw sensor data to cloud services in order to keep all sensory data and traces for our Behaviometrics research and for any future research work that could use the data sets. We envision that in future, *SenSec* will adopt a hybrid approach, in which it processes and consumes part of the sensory data on the mobile devices and sends some of the raw sensory data and some processed data to the cloud services. The cloud services will aggregate sensory time-series from a larger population of users and try to model the *group behaviometrics* in a larger scale. The resulting behaviometric models will then be pushed back to the mobile devices and be used for more accurate user identifications and casual authentications. In [56], we propose such a hybrid architecture called "Fog Computing based Mobile Sensing".

Among other aspects and challenges [56] of such a *Fog* based approach, one critical question is how to balance the ”local” processing vs. ”cloud” processing in a way to optimize transmission and computation costs on the mobile devices in a practical manner. This remains a future research direction.

CHAPTER 7

CONCLUSION

This thesis explores the problem of capturing user behavioral patterns and its applications by conducting scientific study on human behavior patterns in the digital age, which is known as BehavioMetric. Unlike the traditional approaches to measure and quantify such *metrics*, we established a new ground to measure, model and analyze human's behaviors through sensing techniques in a mobile computing environment.

A key step towards this end is defining the *BehavioMetrics*, outlining its enabling framework, identifying various sensors on modern mobile devices and finding factors and metrics that can be derived from them. Applying the *BehavioMetrics* framework in real world scenario, we proposed a particular use case, SenSec, to passively secure mobile device and application from unauthorized users.

We discussed in detail our BehavioMetric models and outlined the methods used to analyze and interpret these models in a quantitative manner. We started by proposing a novel language approach that converts multi-dimensional sensory data into behavioral text representation and utilizes NLP techniques to perform modeling and inference tasks. Further exploiting such representations, we proposed a hierarchical scheme, Helix, to segment and classify sensory time-series. This novel language-based approach is then augmented with traditional time-series based methods. Moreover,

we tackled the case where time-series analysis is not particularly useful with spatial analysis techniques. For each of the modeling techniques developed, we also showed their application to various tasks, e.g., classification, identification, authentication, etc. We built SenSec app based on MobiSens platform, a mobile behavior monitoring, annotation and data collection tool developed as part of this research, to carry out some of experiments to evaluate these models and algorithms. We also conducted user studies to drive the usability issues arise from these experiments and early field trails of the SenSec App.

In summary, we have made the following contributions:

1. We proposed a language approach in studying mobile user behaviors. We developed several techniques to convert raw sensory data into behavior text representation and apply NLP algorithms to perform activity segmentation, recognition, classification, prediction and anomaly detection.

2. We proposed a new unsupervised algorithm *Helix* to induce the underlying grammar or structure of human behaviors using partial or unlabeled heterogenous sensory data and to discover the hierarchy of the activities considering various granularities. This hierarchical structure of activity recognition enable us to model user behaviors not only at the micro level, but also at the macro level.

3. We investigated how existing legacy machine learning algorithms can be adapted and applied to the Behaviometrics context when the NLP approach is less efficient.

4. We derive a method for extracting an "activity level" from time series data which can be used in performing user identification and anomaly detection. We also added location as a factor to augment our models to reduce the false positive rate and improve accuracy.

5. We built and deployed a mobile security application, SenSec, based on our research on mobility anomaly detection, mobile user behavior modeling through passive sensing, and micro

behavioral modeling using software keyboard interactions (KeySens) to release to the public domain. From the experiments we conduct with human subjects using these applications, we are also able to provide guidelines in selecting different models, tuning model parameters and improving user experience, especially in handling errors and false-positives in prediction and anomaly detection for mobile device and application security.

6. The dataset collected by SenSec, and its backend system, which has, in the field, continuously collected huge amounts of mobile sensory data, some of them are labelled or partially labelled for various purposes including security, lifelog, and mental status. These valuable data sets will enable a broader range of research in modeling and analyzing human behaviors in the mobile computing context.

As an effort to enhance the extend to which mobile devices can be proactive and assistive in our day-to-day lives, we believe that being able to observe and understand mobile users' behaviors will play a key role in the future era of ubiquitous computing. In this regards, we particularly focused on formalizing the computational models of human behaviors that can perform well in the face of data uncertainty and complex behavioral dynamics. With the framework, theoretical foundations and specific applications developed in this thesis, we have a good starting point towards bridging physical world gestures and activities to the digital world mobility, security as well as intelligence.

APPENDIX A

MOBISENS FRAMEWORK

A.1 MOBISENS ARCHITECTURE

MobiSens is a full-blown mobile sensing platform designed for a range of real-world mobile sensing applications using novel algorithms and user interface. MobiSens addresses common challenges in mobile sensing including robust sensor data sampling, power optimization, indexing and understanding the sensor time-series and interaction with users.

After releasing MobiSens on Android Market for five months, we have collected 13,993 hours of sensor data submitted from 310 users for the mobile lifelogger project. Several other research projects have been using the MobiSens platform to collect user data to develop behavior-based anomaly detection [36], future activity prediction [58], remote elderly care [59] and behavior-driven passive authentication [60].

The MobiSens platform is a client/server system consisting of two major components: an Android mobile Application and a two-tier back-end system. Figure A.1 illustrates the system architecture. In this architecture, Smart phones are used as both sensing and controlling units. Mobile clients collect various sensor data from users' phones, apply activity segmentation, lightweight adaptive activity recognition and directly interacts with the user. This sensor data is uploaded to the MobiSens server to index and process the sensor data using heavy-weight algorithms. Second-tier (application

and service) servers use information processed by MobiSens server to provide application specific services such as lifelogging, senior care and ground reporting etc.

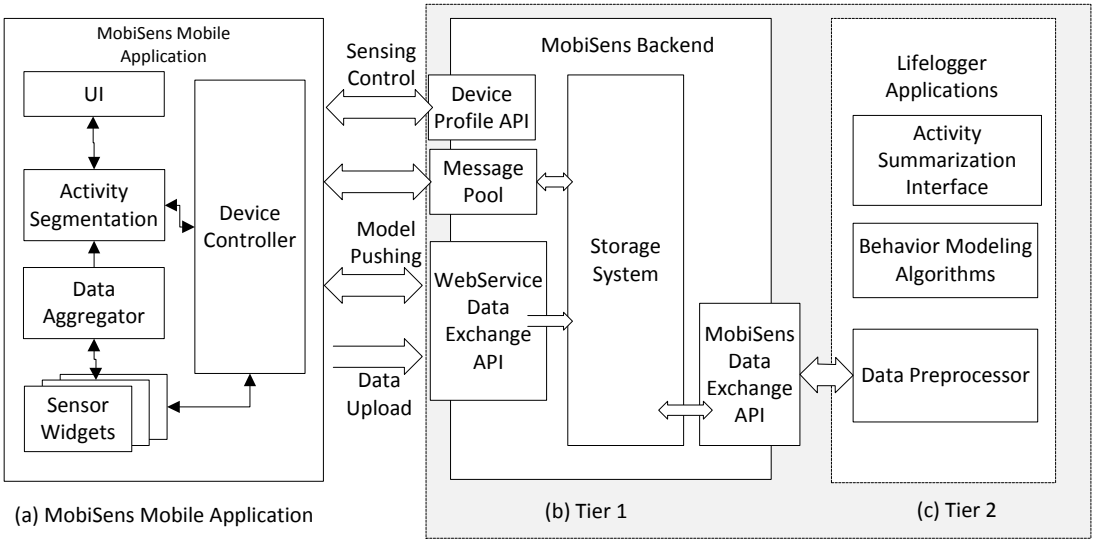

Figure A.1: MobiSens mobile application and back-end system architecture. MobiSens is a client/server system with Android App as client and two-tier server systems.

A.2 MOBILE CLIENT COMPONENTS

The MobiSens mobile client has four components: Data Widgets, the Data Aggregator, the Device Controller and the Activity Recognition Module. The client is designed as data-centric and message-driven, i.e., components are both message receivers and broadcasters. They register as listeners of a certain set of messages and then process data passed with these messages and broadcast the processing result as new messages.

A.2.1 Data Widgets

In MobiSens, we consider all information sources as "sensors". Sensors include "hardware sensors" such as accelerometers and GPS, and "soft sensors" such as time, date, calendar, phone charging status, ringtone settings etc. Data Widgets are collectors of sensor data. Data Widgets

collect raw sensor readings (Table A.1) and broadcast the information to data consumer components using Android's inter process communication mechanism (IPC) if the sensor sampling rate is low or using RAM directly for high-frequency sensors such as the accelerometer, magnetometer and gyroscope.

Table A.1: Examples of sensor data collected by MobiSens client. All these sensory data are recorded with time-stamps. For *rotation matrix*, we used Motorola Droids which do not have gyroscope sensors. The orientation of the phone is estimated by the Android SDK based on the accelerometer readings and the gravity g-value.

Sensors	**Information Sensed**
3-axis accelerometer	motion
Magnetometer	azimuth value of heading direction
GPS coordinates	outdoor locations and trace
Rotation matrix	orientation of the phone
Ambient light level	lighting level
Shuffled sound recording	environmental acoustic features
Charging state	whether the phone is being charged or not
Temperature sensor	environment temperature
WiFi RSS	indoor location and WiFi provider
Nearby bluetooth IDs	whether the user is in a crowded place
Battery level	power consumption of the device
Process list	which application/service is running
Network stats	network traffic to/from applications

A.2.2 Data Aggregator

Mobile devices are not always connected to the Internet. We can not stream the sampled sensor data to the MobiSens Backend at all time. MobiSens first stores all the data locally and uploads them once a network connection is available and the phone is being charged. Two components on the client side manage the data storage and upload process:

- Data Aggregator: a local data storage manager. The Data Aggregator receives all raw readings from different Data Widgets and saves them to multiple files. It also pushes the raw data file queue to the Data Uploader and notifies the Uploader which files are ready to be uploaded.
- Data Uploader uploads and removes the uploaded file to free local storage space for incoming

sensor data.

A.2.3 Sensing Profile Pulling

There are hundreds of parameters controlling the mobile client's behavior. For example, the sampling interval and sampling rates of each sensor impact the sensitivity of the activity recognition and the battery life of the whole system. Different mobile sensing applications and different mobile devices require different sensing profiles. For instance, mobile lifelogging systems need to run at least 12 hours without charging for users who leave home in the morning and come back from work in the late afternoon. For behavior-driven security systems (Section 3.3) using the user's gripping pattern, the sensing only needs to be activated for a few seconds after the screen is unlocked.

On the other hand, different Android devices have different CPU frequencies (which significantly impact the power consumption rates) and battery capacities, MobiSens clients need to adjust their sampling profiles to ensure the desired battery life on different devices. To avoid publishing new versions of MobiSens on the Android Market each time when we want to adjust the client's parameter settings, MobiSens mobile client has a feature called *profile pulling* which dynamically updates its configurations from the backend server. As the core component of this functionality, the Device Controller periodically *pulls* configuration profile from the backend MobiSens server, including list of sensors need to be sampled, sensor sampling rate and sampling strategy, data push interval, etc.

The client profile can be customized for individual devices by specifying different configuration files. This allows the system to adjust the sensing configuration for each user or each type of devices if needed.

A.2.4 Activity Recognition Module

The Activity Recognition Module monitors process events from the Data Aggregator and performs simple activity modeling tasks, such as motion-based activity segmentation and recognition.

The recognized activities are shown on the mobile client for the user to view and annotate. Details of activity recognition is discussed in Section 4.1.6 and 4.3.

A.3 MOBISENS BACKEND AND MOBILE SENSING APPLICATION SERVERS

The two-tier back-end server contains the first-tier MobiSens Backend and the second-tier Mobile Sensing Application servers. MobiSens Backend communicates with mobile devices directly and provides three key functions: device control, data storage and data access management.

MobiSens Backend receives the sensor feature data pushed/uploaded from mobile devices and stores them in a file system. The data can then be fed via MobiSens Data Exchange API to Mobile Sensing Application servers to derive and infer meanings for sensing applications, such as lifelogger activity summarization and social behavior aggregation. Results are pushed back to MobiSens Backend which can be accessed by mobile devices for users to check their activity summary on their smart phones. Mobile Sensing Application Servers access data processed by MobiSens Backend and provide high-level services with new analytical components.

It is also possible that applications can be developed and supported without MobiSens Backend, in which case, the analytics are done on the MobiSens client instance on the mobile devices and the corresponding actions can be taken locally.

REFERENCES

[1] L. Song, U. Deshpande, U. C. Kozat, D. Kotz, and R. Jain, "Predictability of wlan mobility and its effects on bandwidth provisioning," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, April 2006, pp. 1 –13.

[2] G. Liu and G. Maguire, Jr., "A class of mobile motion prediction algorithms for wireless mobile computing and communication," *Mobile Networks and Applications*, vol. 1, no. 2, pp. 113–121, 1996.

[3] M. Shin, A. Mishra, and W. A. Arbaugh, "Improving the latency of 802.11 hand-offs using neighbor graphs," in *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, ser. MobiSys '04. New York, NY, USA: ACM, 2004, pp. 70–83. [Online]. Available: http://doi.acm.org/10.1145/990064.990076

[4] J.-K. Lee and J. C. Hou, "Modeling steady-state and transient behaviors of user mobility: formulation, analysis, and application," in *Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, ser. MobiHoc '06. New York, NY, USA: ACM, 2006, pp. 85–96. [Online]. Available: http://doi.acm.org/10.1145/1132905.1132915

[5] L. Vu, K. Nahrstedt, S. Retika, and I. Gupta, "Joint bluetooth/wifi scanning framework for characterizing and leveraging people movement in university campus," in *Proceedings of the 13th ACM international conference on Modeling, analysis, and simulation*

of wireless and mobile systems, New York, NY, USA, 2010. [Online]. Available: http://doi.acm.org/10.1145/1868521.1868563

[6] M. Kim, D. Kotz, and S. Kim, "Extracting a mobility model from real user traces," in *Proceedings of the 25th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, Barcelona, Spain, April 2006. [Online]. Available: http://www.cs.dartmouth.edu/~dfk/papers/kim:mobility.pdf

[7] R. Aipperspach, E. Cohen, and J. Canny, "Modeling human behavior from simple sensors in the home," in *Proceedings of IEEE Conf. on Pervasive Computing*, Dublin, Ireland, April 2006, pp. 337–348.

[8] S. Chennuru, P.-w. Chen, J. Zhu, and Y. Zhang, "Mobile Lifelogger - recording , indexing , and understanding a mobile user ' s life," *MobiCase*, 2010.

[9] S. Buthpitiya, Y. Zhang, A. Dey, and M. Griss, "n-gram geo-trace modeling," in *Proceedings of Ninth International Conference on Pervasive Computing*, San Francisco, CA, June 12-15 2011.

[10] M. Baldauf and S. Dustdar, "A survey on context-aware systems," *INTERNATIONAL JOURNAL OF AD HOC AND UBIQUITOUS COMPUTING*, p. 2004, 2004.

[11] K. Wrona and L. Gomez, "Context-aware security and secure context-awareness in ubiquitous computing environments."

[12] R. Orr and G. Abowd, "The smart floor: A mechanism for natural user identification and tracking," *ACM Press*, 2000.

[13] A. Peacock, X. Ke, and M. Wilkerson, "Typing patterns: a key to user identification," *Security Privacy, IEEE*, vol. 2, no. 5, pp. 40 –47, sept.-oct. 2004.

[14] E. Hayashi, S. Das, S. Amini, J. Hong, and I. Oakley, "Casa: Context-aware scalable authentication," in *SOUPS '13 Proceedings of the Ninth Symposium on Usable Privacy and Security*, 2013.

[15] D. Gafurov, K. Helkala, and T. Sondrol, "Biometric gait authentication using accelerometer sensor," *Journal of Computers*, vol. 1, 2006.

[16] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma, "Understanding mobility based on gps data," in *Proceedings of the 10th international conference on Ubiquitous computing*, ser. UbiComp '08. New York, NY, USA: ACM, 2008, pp. 312–321. [Online]. Available: http://doi.acm.org/10.1145/1409635.1409677

[17] A.-D. Schmidt, F. Peters, F. Lamour, and S. Albayrak, "Monitoring smartphones for anomaly detection," in *Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, ser. MOBILWARE '08. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007, pp. 40:1–40:6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1361492.1361542

[18] K. hao Chang, J. Hightower, and B. Kveton, "Inferring identity using accelerometers in television remote controls," in *In Proceedings of the International Conference on Pervasive Computing*, 2009.

[19] M. Jakobsson, E. Shi, P. Golle, and R. Chow, "Implicit authentication for mobile devices," in *Proceedings of the 4th USENIX conference on Hot topics in security*, ser. HotSec'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 9–9. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855628.1855637

[20] E. Shi, Y. Niu, M. Jakobsson, and R. Chow, "Implicit authentication through learning user behavior," in *Proceedings of the 13th international conference on Information security*, ser. ISC'10. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 99–113.

[21] W. Shi, J. Yang, Y. Jiang, F. Yang, and Y. Xiong, "Senguard: Passive user identification on smartphones using multiple sensors," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2011 IEEE 7th International Conference on*, oct. 2011, pp. 141 –148.

[22] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell, "The jigsaw continuous sensing engine for mobile phone applications," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '10. New York, NY, USA: ACM, 2010, pp. 71–84. [Online]. Available: http://doi.acm.org/10.1145/1869983.1869992

[23] S. P. Banerjee and D. L. Woodard, "Biometric authentication and identification using keystroke dynamics: A survey," *Journal of Pattern Recognition Research*, 2012.

[24] S. Zahid, M. Shahzad, S. Khayam, and M. Farooq, "Keystroke-based user identification on smart phones," in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, E. Kirda, S. Jha, and D. Balzarotti, Eds. Springer Berlin Heidelberg, 2009, vol. 5758, pp. 224–243. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-04342-0_12

[25] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang, "Accessory: password inference using accelerometers on smartphones," in *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications, HotMobile'12*. New York, NY, USA: ACM, 2012, pp. 9:1–9:6. [Online]. Available: http://doi.acm.org/10.1145/2162081.2162095

[26] F. Bergadano, D. Gunetti, and C. Picardi, "User authentication through keystroke dynamics," *ACM Trans. Inf. Syst. Secur.*, vol. 5, no. 4, pp. 367–397, Nov. 2002. [Online]. Available: http://doi.acm.org/10.1145/581271.581272

[27] E. Maiorana, P. Campisi, N. González-Carballo, and A. Neri, "Keystroke dynamics authentication for mobile phones," in *Proceedings of the 2011 ACM Symposium on Applied Computing*, ser. SAC '11. New York, NY, USA: ACM, 2011, pp. 21–26. [Online]. Available: http://doi.acm.org/10.1145/1982185.1982190

[28] L. Cai and H. Chen, "On the practicality of motion based keystroke inference attack," in *Trust and Trustworthy Computing*, ser. Lecture Notes in Computer Science, S. Katzenbeisser, E. Weippl, L. Camp, M. Volkamer, M. Reiter, and X. Zhang, Eds. Springer Berlin Heidelberg, 2012, vol. 7344, pp. 273–290. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30921-2_16

[29] L. Wang and X. Geng, *Behavioral Biometrics For Human Identification: Intelligent Applications (Premier Reference Source)*, 1st ed. Medical Information Science Reference, Sep. 2009. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/1605667250

[30] A. J. Aviv, B. Sapp, M. Blaze, and J. M. Smith, "Practicality of accelerometer side-channel on smartphones," in *Proceedings of Annual Computer Security Applications Conference. (ACSAC 2012)*, 2012.

[31] K. Burke, *Language as Symbolic Action*. University of California Press, 1966.

[32] J. V. Wertsch, *Mind As Action*. Oxford University Press, USA, 1998.

[33] C. E. Shannon, "A mathematical theory of communication," Bell Systems Technical Journal, Tech. Rep., 1948.

[34] P.-W. Chen, S. K. Chennuru, and Y. Zhang, "A language approach to modeling human behavior," in *Proceedings of The seventh international conference on Language Resources and Evaluation (LREC)*, Valletta, Malta, May 19-21 2010.

[35] S. Buthpitiya, Y. Zhang, A. Dey, and M. Griss, "n-gram geo-trace modeling," in *Proceedings of Ninth International Conference on Pervasive Computing*, San Francisco, CA, June 12-15 2011.

[36] J. Zhu and Y. Zhang, "Towards accountable mobility model: A language approach on user behavior modeling in office wifi networks," in *Proceedings of The IEEE International Conference on Computer Communications and Networks (ICCCN 2011)*, Maui, Hawaii, July 31 - August 4 2011.

[37] K. Farrahi and D. Gatica-Perez, "Extracting mobile behavioral patterns with the distant n-gram topic model," *Wearable Computers, IEEE International Symposium*, vol. 0, pp. 1–8, 2012.

[38] C. Zhai and J. Lafferty, "A study of smoothing methods for language models applied to information retrieval," *ACM Transactions on Information and System Security*, vol. 22, no. 2, pp. 179–214, 2004.

[39] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains," *The Annals of Mathematical Statistics*, vol. 41, no. 1, pp. 164–171, 1970. [Online]. Available: http://dx.doi.org/10.2307/2239727

[40] P.-w. Cheng, S. Chennuru, S. Buthpitiya, and Y. Zhang, "A Language-Based Approach to Indexing Heterogeneous Multimedia Lifelog," *International Conference on Multimodal Interfaces*, 2010.

[41] A. Mueen, E. Keogh, Q. Zhu, and S. Cash, "Exact discovery of time series motifs," *Proceedings of the SIAM*, 2009. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.155.1026\&rep=rep1\&type=pdf

[42] C. D. Manning and H. Schuetze, *Foundations of Statistical Natural Language Processing*, 1st ed. The MIT Press, Jun. 1999.

[43] A. O. S. Project, "Touch devices." [Online]. Available: http://source.android.com/tech/input/touch-devices.html

[44] R. O. Duda, P. E. Hart, and D. G. Stork, "Multi-layer neural networks," in *Pattern Classification, 2nd Edition*. John Wiley and Sons, Inc., 2001, vol. 2.

[45] K. Killourhy and R. Maxion, "Comparing anomaly-detection algorithms for keystroke dynamics," in *Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on*, 2009, pp. 125–134.

[46] E. Mok and G. Retscher, "Location determination using wifi fingerprinting versus wifi trilateration," *J. Locat. Based Serv.*, vol. 1, pp. 145–159, June 2007. [Online]. Available: http://portal.acm.org/citation.cfm?id=1451879.1451884

[47] P. Bolliger, "Redpin - adaptive, zero-configuration indoor localization through user collaboration," in *MELT '08: Proceedings of the first ACM international workshop on Mobile entity localization and tracking in GPS-less environments*. New York, NY, USA: ACM, 2008, pp. 55–60.

[48] H. Lin, Y. Zhang, M. Griss, and I. Landa, "Wasp: an enhanced indoor locationing algorithm for a congested wi-fi environment," in *Proceedings of the 2nd international conference on Mobile entity localization and tracking in GPS-less environments*, ser. MELT'09, Berlin, Heidelberg, 2009, pp. 183–196. [Online]. Available: http://portal.acm.org/citation.cfm?id=1813141.1813158

[49] J. Han, E. Owusu, L. T. Nguyen, A. Perrig, and J. Zhang, "Accomplice: Location inference using accelerometers on smartphones," in *COMSNETS*, 2012, pp. 1–9.

[50] L. Van Boven, A. Kamada, and T. Gilovich, "The perceiver as perceived: Everyday intuitions about the correspondence bias." *Journal of Personality and Social Psychology*, vol. 77, no. 6, pp. 1188–1199, 1999.

[51] B. Logan, J. Healey, M. Philipose, E. Tapia, and S. Intille, "A long-term evaluation of sensing modalities for activity recognition," in *Proceedings of the 9th international conference on Ubiquitous computing*. Springer-Verlag, 2007, pp. 483–500.

[52] P. Soucy, "Beyond TFIDF weighting for text categorization in the vector space model," in *In Proceedings of the Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005*, 2005, pp. 1130–1135. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.105.4957

[53] I. S. Organization, "Biometric performance testing and reporting," 2006.

[54] A. Whitten, J. D. Tygar, A. Whitten, and J. D. Tygar, "Usability of security: A case study," Tech. Rep., 1998.

[55] J. Knudsen, "Securerandom," in *Java Cryptography*. O'Reilly Media, 2010, vol. 1, pp. 38–49.

[56] J. Y. Zhang, P. Wu, J. Zhu, H. Hu, and F. Bonomi, "Privacy-preserved mobile sensing through hybrid cloud trust framework," in *Proceedings of the 6th IEEE International Conference on Cloud Computing*, 2013.

[57] D. Gordon, J. Czerny, and M. Beigl, "Activity recognition for creatures of habit," *Personal and Ubiquitous Computing*, pp. 1–17, 2013. [Online]. Available: http://dx.doi.org/10.1007/s00779-013-0638-2

[58] L. T. Nguyen, H.-T. Cheng, P. Wu, S. Buthpitiya, and Y. Zhang, "Pnlum: System for prediction of next location for users with mobility," in *Procedings of Mobile Data Challenge by Nokia Workshop at the Tenth International Conference on Pervasive Computing*, Newcastle, UK, June 2012.

[59] P. Wu, H.-K. Peng, J. Zhu, and Y. Zhang, "Senscare: Semi-automatic activity summarization system for elderly care," in *International Conference on Mobile Computing, Applications, and Services (MobiCASE)*, 2011.

[60] J. Zhu, P. Wu, X. Wang, and J. Y. Zhang, "Sensec: Mobile application security through passive sensing," in *Proceedings of International Conference on Computing, Networking and Communications. (ICNC 2013)*, San Diego, CA, USA, January 28-31 2013.