Lifespace Tracking and Activity Monitoring on Mobile Phones

by

Ian Dewancker

BSE., The University of Waterloo, 2011

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

March 2014

Abstract

Daily patterns of behaviour are a rich source of information and play an important role in establishing a person's quality of life. Lifespace refers to measurements of the frequency, geographic extent and independence of an individual's travels. While difficult to measure and record automatically, lifespace has been shown to correlate to important metrics relating to physical performance, nutritional risk, and community engagement.

MobiSense is a mobile health research platform that aims to improve mobility analysis for both ambulating and wheelchair users. The goals of the system were to be simple for users to collect mobility data, provide accessible summaries of daily behaviours and to enable further research and development in this area. The system is capable of lifespace summaries relating to indoor and outdoor mobility as well as activity trends and behaviours.

For indoor reporting, we investigated robust classification algorithms for room level indoor localization using WiFi signal strengths. We pursue topological map localization as it requires simpler map models while preserving useful semantic information associated with location. Personalized maps are easy to create by capturing training observations in areas of interest. Outdoor summaries are captured by periodically recording GPS fixes.

For activity monitoring, a decision tree classifier was learned using a combination of accelerometer and GPS features. The classifier can differentiate between stationary, wheeling (in a wheelchair), walking or vehicle motion.

To capture the relevant sensor data, we extended an open source logging application which records data streams locally before uploading data to a web service to process and visualize results. The custom web service processes the data and generates summary files which can then be visualized either for each individual day or over a user selected date range. We employed a heat map visualization for outdoor lifespace to understand the geographic extent of a user's mobility. For indoor and activity summaries, we employed temporal line charts to understand trends in a user's mobility.

Preface

The design and evaluation of the experiments outlined in Chapter 2 and 3 were primarily done by me with assistance from Dr. Ian Mitchell. All data collection for use in this thesis was performed primarily by me with assistance from Dr. Jaimie Borisoff. I extended an open source sensor logging application [13] to suit our needs. The backend for the web application was designed and developed entirely by me. Tom Jin worked heavily on improving my early versions of the data visualizations.

A paper has been accepted for publication at RESNA 2014 that summarizes the work in this thesis [9]. Dr. Jaimie Borisoff wrote most of the paper, basing it on a near complete version of this thesis, and had assistance from Dr. Ian Mitchell and I.

Table of Contents

List of Tables

List of Figures

List of Programs

Acknowledgements

I would like to thank my supervisors Dr. Ian Mitchell and Dr. Jaimie Borisoff for their guidance and suggestions during my study at UBC. The path towards this thesis was certainly not direct and I am very appreciative of their patience for my technical meandering. Much of this work would have been difficult or impossible without Dr. Borisoff's data collection and validation. I would also like to thank Dr. Nando de Freitas and Jordan Frank for productive discussions and useful suggestions.

Tom Jin was an excellent collaborator and was instrumental in taking the web client visualizations to the next level. Great thanks as well to my LCI labmates : Junaed, Pouria, Eric, Pooja, Dave and Mauricio for bouncing ideas off of. A very important thanks to my other friends at UBC, Felix and John for inspiring technical discussions and countless Vancouver adventures. Thanks as well to Josh and Ryan for California distractions and technical brainstorming. Much of the analysis and evaluation for this thesis would not have been possible without the help of Arianne, Nikole, Vanessa, Rachel, and Alanna for helping with data collection and being great friends during my time in Vancouver.

Thanks to my family in Belgium and Canada, especially Luc and Brigitte, and Oma and Opa for offering their homes and Internet connections during my trip to Belgium as well as Nancy and Probal for always welcoming me in Toronto. Finally, thanks to my parents and sisters for their unconditional support and love throughout my studies and travels.

This research was supported by CanWheel (the Canadian Institutes of Health Research Emerging Team in Wheeled Mobility for Older Adults Grant), the National Science and Engineering Research Council of Canada (NSERC) Discovery Grant, an NSERC Undergraduate Student Research Award, and Dr. Borisoff's Canada Research Chair in Rehabilitation Engineering Design.

Chapter 1

Introduction

Mobile devices now surround us and have transformed into important tools in our daily lives. Our activities and behaviours are becoming increasingly coupled to devices with ever improving sensors. With current devices, mobility patterns and physical activity levels are particularly amenable to inference and analysis by leveraging the integration of GPS, accelerometer and other sensors. Mobile health systems that facilitate longterm mobility studies and reporting would potentially be of great value to health care, assistive technology and rehabilitation science. Highly personalized, detailed mobility traces are not only of interest to researchers, but also individuals motivated to learn more about their own trends and behaviours. The quantified self movement aims to empower these individuals with accessible data and analysis tools. Towards these ends, we investigate the design of a system to track lifespace and monitor mobility of both ambulating and wheelchair users with standard mobile phones. Lifespace refers to measurements of the frequency, geographic extent and independence of an individual's travels [37]. We concern ourselves with models and approaches to summarize indoor and outdoor movement as well as activity levels. We focus on techniques that perform well on real sensors and in real-world environments.

1.1 Contributions

We evaluate four classifiers for room level localization using WiFi signal strength using datasets from actual residences. We propose an effective set of features for use with a decision tree classifier to differentiate four activity classes. Finally, we describe MobiSense, a mobile-to-web lifespace tracking system and its data visualizations.

1.2 Organization

In Chapter 2 we present our investigation into robust classifiers for WiFi based indoor localization. Chapter 3 describes our approach to activity recognition. Chapter 4 presents the entire MobiSense system, and Chapter 5 discusses future work and extensions.

Chapter 2

WiFi Indoor Localization

Indoor localization is an important problem in the realms of robotics, ubiquitous computing and assistive technology. A robust indoor positioning system allows for context-aware assistive applications in the home and work place. Even with room level localization, one could imagine useful applications related to cognitive aids, navigation assistance, and tracking.

In particular, an indoor localization component would be useful for systems designed to measure lifespace and mobility. While various systems have explored outdoor mobility summaries using GPS, indoor location extends mobility summarization capabilities, and provides richer contextual information. Rooms within the home and office have strong associations with specific activities (e.g. kitchen and cooking). High level activity recognition would therefore also be enhanced by indoor localization.

To perform localization, a map or model is required to infer position. Localization is primarily performed on two types of maps : metric and topological. Metric maps model continuous spaces and are capable of reporting relative distances (e.g. 2m from south east wall of kitchen) in contrast with topological maps which only model discrete locations (e.g. in kitchen). We pursue topological map localization as it requires simpler map models while preserving useful semantic information associated with location. The key elements required for probabilistic localization are shown below [42] :

$$P(x_i \mid x_{i-1}, u) = \text{motion model}$$
$$P(z \mid x_i) = \text{observation model}$$

We concern ourselves with the investigation of a robust observation model for residential indoor locations. We do not assume connectivity information between places to be present in the map and so motion models are of limited use for localization. We are interested in low cost sensing devices and observation models that work well in practice. In addition, it would be desirable to require only minimal human effort for mapping and model construction as it would decrease deployment efforts.

Vision sensors are popular as they are low cost and have been well studied in this context [34] [35] [8], but mounting cameras on people is potentially awkward and risks capturing sensitive or private information. While GPS repeaters can be installed indoors, in most situations, GPS functionality will not be available inside buildings. Radio frequency signals can also be used to perform indoor localization, and since WiFi has become highly integrated into most urban and suburban infrastructures, it has the advantage of requiring little deployment effort. Signal measurements can be made by any mobile devices equipped with WiFi modems including mobile phones. While cameras require an unobstructed view of their surroundings, measuring WiFi signals can be done with a phone placed in a pocket or bag.

To investigate room-level WiFi indoor localization, we evaluated four probabilistic classification algorithms. Performance was measured on a dataset collected with an off-the-shelf Android phone and consists of real signal strengths recorded in several residential environments. We also considered the problem of novelty detection, that is determining whether a WiFi observation belongs to a known indoor location or not, and evaluated five novelty metrics.

2.1 Related Work

The seminal work in using WiFi signals to localize indoors was [1] where a large fingerprint database of signal strengths was captured and location was inferred by performing a nearest neighbour search. More recently, a system proposed in [31] looked at organically growing an indoor fingerprint model by requiring intermittent input from users to establish ground truth location, thereby distributing the burden of site survey across multiple users. A probabilistic model was proposed in [35] that fused vision with WiFi on low-cost mobile devices, however this approach required a detailed site map and survey process. Ekahau, a commercial system [17], requires placing WiFi device tags within a building.

Localization can only be achieved in conjunction with a map and so there has also been research into posing the problem in the simultaneous localization and mapping (SLAM) context. The use of Gaussian process models for indoor WiFi localization was first proposed in [11], which required labeled ground truth positions. These requirements were later removed to achieve full SLAM [10]. Recently, work in [16] showed state of the art results by posing the problem as a GraphSLAM instance.

Topological or room based maps have also been of interest. In [45], a system attempts to cluster similar WiFi fingerprints to generate a connected topological map. Work in [14] explored using hierarchical Dirichlet processes to learn clusters associated with indoor locations and rooms of interest.

We approach the problem in a similar way to [2] in that we are interested in defining an explicit training phase for map construction and using probabilistic classification algorithms to infer room-level topological location.

2.2 WiFi Overview

WiFi networks have become popular in the home to service the growing number of personal mobile devices. Laptops, mobile phones and tablets all use WiFi networks to connect to the Internet. Access points connect WiFi enabled devices to wired networks. There are three key pieces of information about access points visible to other WiFi enabled devices. The service set identifier (SSID) is a human-readable string specifying what network the AP is part of. The basic service set identification (BSSID), also known as the media access control (MAC) address in WiFi networks is a globally unique identifier for the network interface of the access point. The received signal strength indication (RSSI) is a generic metric used in telecommunications to represent the power present in a radio signal [24]. Example access point information is shown in Table 2.1. Note that the SSID is not unique, however the BSSID is. The RSSI will vary depending on the relative position of the AP and the listening device as well as the physical environment (location of walls and other obstacles).

Table 2.1: Example WiFi Access Point Information

SSID (Name)	**BSSID (MAC Address)**	**RSSI (Signal Strength)**
ubc_secure	00:12:44:b0:4f:bb	-73
ubc_secure	00:14:f1:ac:74:23	-36
ubc_visitor	00:1c:0e:42:47:45	-84
bbox2-3964	00:10:7f:13:55:75	-66

While most households only employ a single access point, neighbouring access points are often also visible. WiFi routers are cheap and emit a signal even when not connected to the Internet.

In probabilistic localization, observations represent sensor measurements of physical environments. A common choice for observation vector when using WiFi signals is to simply use the raw RSSI values returned by the modem for each visible access point [16] [10] [2]. We assume that for a given building or indoor location there exists a set of observable access points with unique BSSIDs. The observation vector z is simply formed by recording the signal strengths of the set of N access points associated with the indoor location (which depends on each location) as shown below.

$$z = [\ RSSI_0,\ RSSI_1,\ \cdots,\ RSSI_N\ ]$$
$$RSSI_i = \text{RSSI of BSSID}_i$$

The SSID information of the access point is ignored if signal strength information for a given BSSID is absent during a reading. The signal strength for the absent access point is cast to -100 as proposed in [2].

2.3 Localization as Probabilistic Classification

We pose the problem of learning an observation model as an instance of supervised learning. Given a set of M training examples $\{(z_0, x_0)$, (z_1, x_1) ... $(z_M, x_M)\}$ where z_i is the observation vector described in the previous section and x_i is the place/room label (eg kitchen), we are interested in learning a suitable model of $P(z \mid x)$ and more importantly arriving at $P(x \mid z)$ to infer location. In our domain, the target variable is x (location) and so we are not necessarily concerned with modeling the statistics of all observation variables z (WiFi signal strengths) as generative models do. A generative model of $P(z \mid x)$ would allow sampling of WiFi observations for a given room. For example, one could sample possible signal strengths of all known access points given that you were in the kitchen. With $P(z \mid x)$, one can use Bayes rule to arrive at $P(x \mid z)$ and then localize to the most probable room. In contrast, discriminative models learn a direct mapping to the target distribution $P(x \mid z)$ and do not necessarily learn the statistics associated with all variables.

In this section, we describe two generative models of $P(z \mid x)$, naive Bayes and Chow-Liu trees, and two discriminative models for $P(x \mid z)$, random forests and k-nearest neighbours, and evaluate their classification performances on real-world data. Our comparison of classifiers is similar to work in [2]. While it is often the case that discriminative models perform better on classification tasks [28], generative models can give deeper insights into variable statistics and offer a useful comparison.

2.3.1 Naive Bayes

One of the simplest and most common approaches to classification is the naive Bayes classifier. Naive Bayes is a generative model that makes a simplifying assumption about the conditional independence of feature variables [28]. In our context this means assuming that the signal strengths of each access point are conditionally independent given the room the user is located in. Shown below in Figure 2.1 is a graphical model depicting the naive Bayes structure. The nodes are variables and the arrows denote conditional dependence. In this example, the six signal strength observations (0-5) depend only on the location (x) in which they are read.

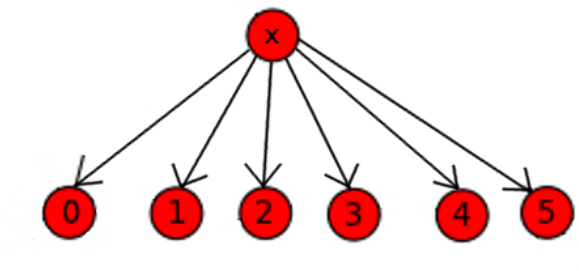

Figure 2.1: Example naive Bayes graphical model for localization.

The probability of the entire WiFi observation conditioned on the room is equal to the product of the individual conditional probabilities of each access point's signal strength conditioned on the room as shown below.

$$P(z[0], z[1], ..., z[N] \mid x) = \prod_{i=0}^{N} P(z[i] \mid x)$$

The choice remains as to what model or distribution to use for the individual conditional probabilities. We investigate two alternatives : modeling the conditional with a Gaussian distribution or as a normalized histogram distribution. The representation of the conditional probabilities can dramatically alter the classifier's performance. Also worth remembering is that we have made a strong assumption in treating the signal strength variables as independent. While this assumption often works well in practice [28], the data may be better modeled with weaker assumptions.

Gaussian Distribution

One option is to model each conditional probability of the naive Bayes as a one dimensional Gaussian distribution, as defined below.

$$P(z[i] \mid x) = \frac{1}{\sqrt{2\pi\sigma^2_{z[i]}}} \exp\left(-\frac{(z[i] - \mu_{z[i]})^2}{2\pi\sigma^2_{z[i]}}\right)$$

Each conditional then is parameterized by the mean, $\mu_{z[i]}$ and variance, $\sigma^2_{z[i]}$ of the Gaussian. These parameters are estimated using maximum likelihood estimates which is to say the empirical values calculated from the training samples.

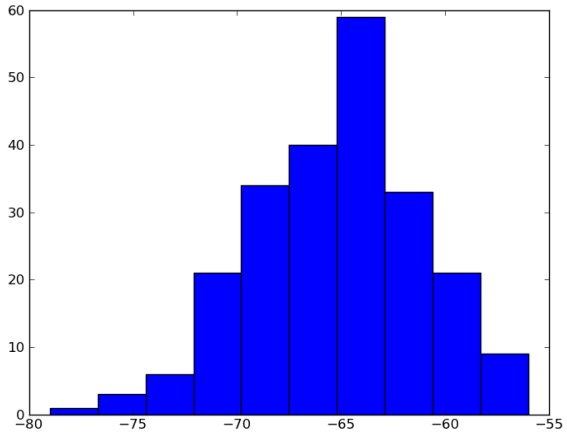

Figure 2.2: Frequency of signal strengths of single access point in a given room. No absent signals strength readings (-100) present

A histogram showing the frequency of signal strengths from a single access point in a room is shown in Figure 2.2. One can see that the distribution looks somewhat Gaussian, however it also looks slightly skewed. Another consideration is the situation where an access point would be visible in one room and not seen in another. This signal strength would have only absent values (-100) in the second room in the training data, so the Gaussian estimated would be degenerate with 0 variance. In practice, a small epsilon is added to the empirical variance to prevent this [32].

Histogram Distribution

As is often the case when dealing with real sensor data, the Gaussian model may not accurately represent the underlying data. A common approach in robotics is to learn a mixture model using expectation maximization as outlined in [42]. In our scenario, the Gaussian assumption very obviously breaks down when an access point is sometimes visible in a room and sometimes not. The access point's signal strength alternates between real strengths (-95 to -10) and the absent placeholder strength (-100). This is demonstrated in Figure 2.3 below and one can clearly see that a Gaussian fit to these readings would not represent the data well. Additionally, in residential environments, most access points seen will likely have weaker signal strength since they are not located within the same building.

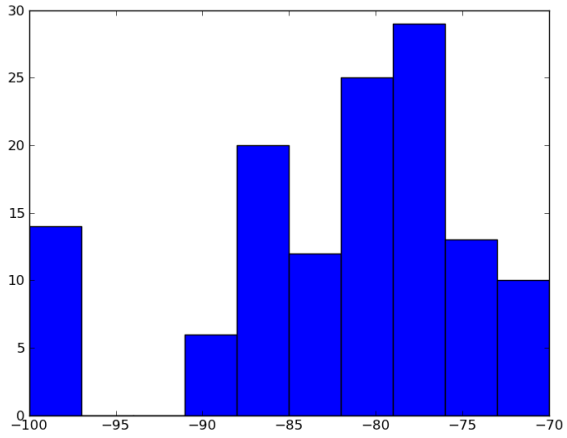

Figure 2.3: Frequency of signal strengths of single access point in a given room. Absent signal strength readings (-100) are present

Instead of learning a mixture model, we simply use a normalized histogram of the empirical signal strength for each room and discretize the signal strength into bins of 5 RSSI units.

$$P(z[i] \mid x = r) = \text{Hist}(z[i], r)$$

We return a small epsilon if the signal strength for a given room was not observed in training. As we will see, this histogram approach (referred to as robust Naive Bayes in the Section 2.5.2) works much better than the Gaussian model.

2.3.2 Chow-Liu Trees

Introduced in [6], the Chow-Liu algorithm approximates a probability distribution by the closest tree-structured Bayesian network. The topology of the Bayesian network is determined by the maximum-weight spanning tree of the complete graph of N nodes for the N variables. The weights are calculated as the mutual information between variables (eq 2.1).

$$I(z[i], z[j]) = \sum_{z[i]\in\Omega,\ z[j]\in\Omega} p(z[i], z[j]) \log \frac{p(z[i], z[j])}{p(z[i])p(z[j])} \tag{2.1}$$

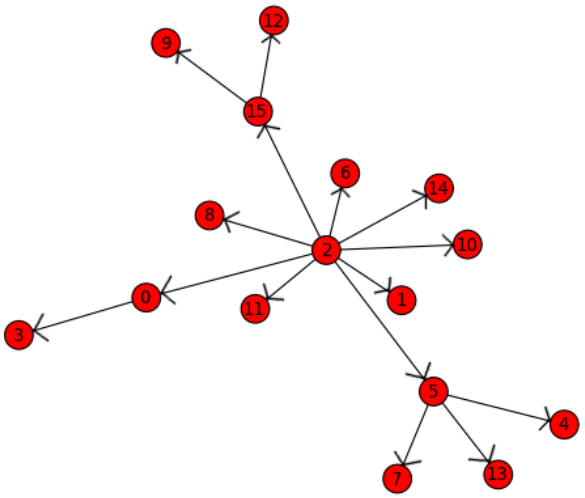

Figure 2.4: Example graphical model of learned Chow-Liu tree

This structure saw interest again in robotics research in [8] for visual appearance models and to our knowledge this is the first time Chow-Liu trees have been evaluated for WiFi indoor localization.

Why would this be an appropriate model for indoor localization? The structure could encode the belief about the signal strengths of neighbouring access points as dependent on the perceived signal strength of a central access point in the home or building of interest.

This is illustrated in Figure 2.4 where access point 2 is learned as the central AP for a given room (indeed this was the case) and the probability distributions of the other access points' signal strengths depend on access point 2's strength as defined by the network.

To find the probability of the entire WiFi observation conditioned on the room, the structure of the learned tree is used where $parent(i)$ represents the parent variable in the tree as shown below.

$$P(z[0], z[1], ..., z[N] \mid x) = \prod_{i=0}^{N} P(z[i] \mid z[parent(i)], x)$$

This is in contrast with the naive Bayes factorization as the signal strengths are no longer conditionally independent; they depend on exactly one other access point's signal strength, and the Chow-Liu algorithm attempts to learn which access point that should be. We use a normalized histogram in the same way as described for the naive Bayes conditional probabilities to store the resulting model.

$$P(z[i] \mid z[parent(i)] = s, x = r) = \text{Hist}(z[i], s, r)$$

2.3.3 K-Nearest Neighbours

The k-nearest neighbour classifier is a non-parametric classifier that compares test input to training examples and finds the k most similar. It then returns the majority class label of those k training examples. While nearest neighbour methods are not inherently probabilistic, they can be converted into a probabilistic model by returning the percentage of the k neighbours voting for each room as shown below. Euclidean distance is used as a similarity metric as described in [1]. The performance of the classifier can depend on the parameter k and so we perform a parameter study in section 2.5.2.

$$NN(z) = \text{k} \arg\min_{(z_i, x_i) \in O} ||z_i - z||_2 \tag{2.2}$$

$$P(x = r \mid z) = \frac{|\{(z, x) \in NN(z) \mid x = r \} |}{\text{k}} \tag{2.3}$$

2.3.4 Random Forests

Introduced in [41] and shown to be useful for WiFi localization in [2], random forests are ensemble learners that use a collection of decision trees to classify test input. They are called random because each tree in the forest is learned on a random subset of the training data. Decision trees are grown by looking for the best way to successively split data. In most cases, this means finding a feature θ and a threshold τ to partition the data set on. We will refer to the pair as a splitting candidate as shown below.

$$\text{splitting candidate } \phi = (\theta, \tau)$$

The normal decision tree learning algorithm attempts to find the best feature and threshold to split the data on. However, in random forests, the feature that is chosen to split on is the best among a random subset of the features [32]. All suitable thresholds are evaluated for the random subset of features however. In our context, these splitting candidates refer to an access point θ and a signal strength τ to split on. For each potential split, two partitions of the dataset are formed: the left set Q_l that contains training pairs having feature θ less than threshold τ and the right set Q_r which contains the rest.

$$\begin{aligned} Q &= \{(z, x)\} \\ Q_l(\phi) &= \{\ (z, x) \mid z[\theta] < \tau\ \} \\ Q_r(\phi) &= Q \setminus Q_l(\phi) \end{aligned}$$

With the data set partitioned into two sets, the split that maximizes the information gain $G(\phi)$ is selected for the current node of the tree. The entropy, $H(Q)$ of the sets is used in calculating the information gain as shown below.

$$\begin{aligned} \phi^* &= \arg\max_{\phi} G(\phi) \\ G(\phi) &= H(Q) - \sum_{s \in \{l,r\}} \frac{|Q_s(\phi)|}{|Q|} H(Q_s(\phi)) \\ H(Q) &= - \sum_{r \in \{rooms\}} p(x = r) \log p(x = r) \end{aligned}$$

The best split is found and the process continues recursively until a stopping criteria is met.

During classification, each tree in the forest votes for the room that it classifies the current sensor reading as, so one can form an estimate of $P(x \mid z)$ simply by counting the number of trees that vote for each room as shown below.

$$P(x = r \mid z) = \frac{|\text{Trees voting for } r|}{\text{Total Trees}}$$

Figure 2.5 shows an example of a single decision tree in the forest. In each node, a specific access point of the test input is compared to a threshold and if less than or equal to the value the process moves down the left child of the node, otherwise down the right. This continues until a leaf node is reached that has a class, or in our case a room, associated with it.

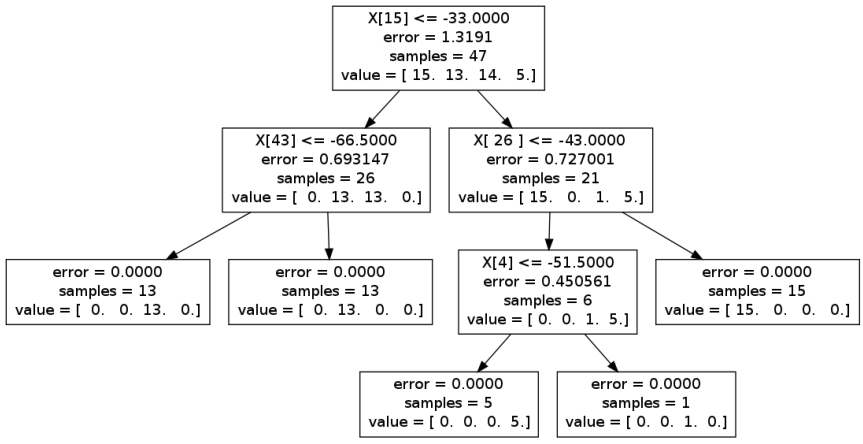

Figure 2.5: A decision tree compares specific access point signal strengths to thresholds to decide location

In the random forest implementation in [32], three key parameters govern their performance and behaviour. The number of estimators is the number of decision trees that will be used. The maximum features controls the number of features in the random subset to be considered for each split. Trees may also optionally consider all features in generating split candidates. Finally, the maximum depth controls how many successive decisions or splits should be made at most for any tree. The trees may optionally continue generating decisions until the training data is perfectly partitioned. The performance of the classifier depends significantly on these three values and so we conducted a parameter study, presented in section 2.5.2.

2.4 Novelty Detection

Localizing to an indoor room is important, however so is knowing when to report that a user is probably not located in one of the trained rooms. In a large building with many rooms or offices or perhaps even a large house, it is likely that a user will not train on all rooms within the building. Therefore the system should only localize when confident that the readings represent observations from within rooms trained on. This is known as the problem of novelty detection in machine learning. We can pose the problem as learning a function that reports either an observation is novel or not, as shown below.

$$f(z) = \begin{cases} 1, & \text{inlier, observation from known room} \\ -1, & \text{outlier, novel observation} \end{cases}$$

The use of one-class SVMs is a common approach to this problem [32], however their performance largely depends on the selection of a kernel function which is a comparison metric for observation vectors. Motivated by these kernel functions, we concern ourselves with investigating useful distance metrics for comparing test observation vectors to the training data.

One observation that the system can be confident to be novel (at least as long as the WiFi environment is not degenerate) is the observation where all the WiFi access points are not visible and every signal strength has the absent value -100 as shown below.

$$z_{neg} = [\ -100,\ -100,\ \cdots,\ -100\ ] \tag{2.4}$$

We evaluate distance metrics used for novelty detection with thresholds computed relative to the novelty reported for z_{neg}. Here $\{z_i\}$ represents the set of training observations and z is the measurement under consideration.

$$f(z) = \begin{cases} 1, & \text{if min } dist(z,\ \{z_i\}) < \tau \\ -1, & \text{otherwise} \end{cases}$$
$$\tau = \alpha \cdot (\text{min } dist(z_{neg},\ \{z_i\}))$$

We propose five different novelty detection schemes and outline them in the following sections.

2.4.1 Nearest Neighbour Threshold

The first novelty detection approach we describe finds the nearest neighbour in the training set to the test vector and tests whether the Euclidean distance to this nearest neighbour is less than a threshold.

$$f(z) = \begin{cases} 1, & \text{if } \min_{i \in O} ||z_i - z||_2 < \tau \\ -1, & \text{otherwise} \end{cases}$$
$$\tau = \alpha \cdot (\min_{i \in O} ||z_i - z_{neg}||_2)$$

The threshold τ is defined as a fraction α of the minimum Euclidean distance of z_{neg} to the training data. O is the set of indices of training observations.

2.4.2 Nearest Centroid Thresholds

Instead of comparing a test vector directly to the training data, we could compare observations to a set of centroids computed from the dataset. We find the mean observation for each room as shown below.

$$\mu_i = \frac{1}{|R_l|} \sum_{i \in R_l} z_i \tag{2.5}$$

R_l is the set of indices of observations labelled as room l. We also learn not one threshold for comparison, but separate thresholds for each centroid.

Euclidean Distance

The first distance metric for comparing test vectors to centroids is the euclidean distance as used in the nearest neighbour threshold.

$$f(z) = \begin{cases} 1, & \text{if } ||\mu_i - z||_2 < \tau_i \\ -1, & \text{otherwise} \end{cases}$$
$$\tau_i = \alpha \cdot ||\mu_i - z_{neg}||_2$$
$$i = \arg\min_{i \in U} ||\mu_i - z||_2$$

The distance between test vectors and their closest centroid is compared to a centroid specific threshold. U is the set of indices of centroids.

Mahalanobis Distance

Gaussian mixture models are a standard generative model for probability distributions [28]. If we had a good measure of the probability of an observation z, given the training data, that is $P(z \mid \{z_i\})$, we could likely determine which are novel. An overview of Gaussian mixture models for novelty detection is presented in [36]. A GMM is defined by M Gaussians each having their own mean, covariance and weight as shown below.

$$P(z \mid \{z_i\}) = \sum_{i=1}^{M} w_i \; g(z|\mu_i, \mathbf{\Sigma_i}) \tag{2.6}$$

$$g(z|\mu_i, \Sigma_i) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2}(z-\mu_i)^T \Sigma_i^{-1}(z-\mu_i)\right) \tag{2.7}$$

An important decision when using a GMM model is how many components, M, to use. In [36] the author suggests growing the number of components until some stopping criteria is met. In our context, we choose to use the number of rooms as the number of components and allow the means to be the same as those calculated in eq 2.5. For the covariance matrices, one could use the empirical estimates from the data, however we found better results by using the robust covariance estimator method of [25] implemented in [32]. The key distance metric in the GMM is the Mahalanobis distance (shown below) in the exponent and so we focus on using that alone in our novelty detection scheme.

$$d(z, \; \mu_i) = \sqrt{(z-\mu_i)^T \Sigma_i^{-1} (z-\mu_i)}$$

The novelty detection then proceeds in much the same way as for the Euclidean distance centroid comparison, except now we use the Mahalanobis distance to compare test vectors to the centroids.

$$f(z) = \begin{cases} 1, & \text{if } d(z, \; \mu_i) < \tau_i \\ -1, & \text{otherwise} \end{cases}$$
$$\tau_i = \alpha \cdot d(z_{neg}, \; \mu_i)$$
$$i = \arg\min_{i \in U} \; d(z, \; \mu_i)$$

2.4.3 Naive Bayes and Chow-Liu Tree Thresholds

As we saw in the previous section, finding $P(z \mid \{z_i\})$ for an observation could be a useful step towards detecting novelty. Fortunately we have two other models, the naive Bayes and Chow-Liu trees, that are capable of determining this quantity since they model the distribution $P(z \mid x, \{z_i\})$. We can marginalize out the location variable x to be left with only $P(z \mid \{z_i\})$ as shown below. We treat $P(x \mid \{z_i\})$ as uniform as we have no prior assumptions about the probability of being localized in a given room.

$$\begin{aligned} P(z \mid \{z_i\}) &= \sum_x P(z, x \mid \{z_i\}) \\ &= \sum_x P(z \mid x, \{z_i\}) P(x \mid \{z_i\}) \\ &= \sum_x P(z \mid x, \{z_i\}) \end{aligned}$$

For novelty detection, we compare the probability of a test vector to the probability of z_{neg}. We take the negative log of the probability for numerical stability and avoiding underflow as suggested in [28]. Computing these probabilities for the Chow-Liu tree and Naive Bayes involves a large product of conditional probabilities, one for each variable.

$$f(z) = \begin{cases} 1, & \text{if } -\log(\ P(z \mid \{z_i\})\ ) < \tau \\ -1, & \text{otherwise} \end{cases}$$
$$\tau = -\alpha \cdot \log(\ P(z_{neg} \mid \{z_i\})\ )$$

2.5 Experiments

For many reasons, an indoor localization component would useful in an overall system for measuring lifespace and summarizing mobility. The indoor environments such a system would be used in would likely be offices and residential buildings such as homes and apartments. We are therefore interested in evaluating the performance of these approaches on real data collected from representative environments.

2.5.1 Data Collection

To collect evaluation data, an Android application was written that allowed for the collection of WiFi signal strength data and images of the environ-

ment. The phone used was an off-the-shelf Nexus 4 phone using standard Android libraries and OpenCV for accessing the camera.

Figure 2.6 shows a screen shot of the data collection application. The app can create rooms (+ button), and record training data with that room (target button). The app continuously captures WiFi signal strengths every 600ms (summarized in top right) and an image every 1.8 seconds from the back facing camera. The images were recorded to provide ground truth for evaluation. The app saves the data to a simple text format.

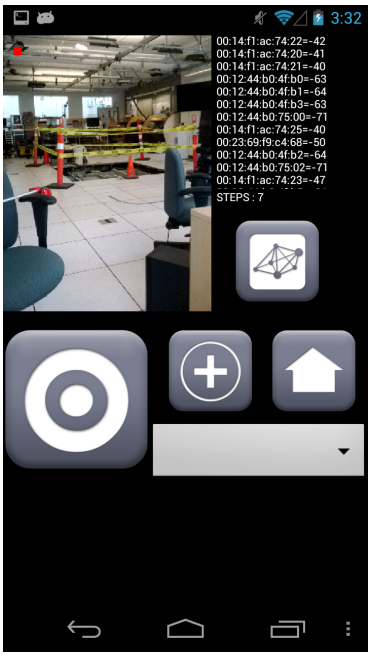

Figure 2.6: A screen shot of the data collection application used to generate training and evaluation datasets for localization.

Five datasets produced using the application are summarized in Table 2.2. The datasets were collected in a variety of indoor environments, including 3 homes, 1 apartment and 1 dataset (UBC) combines readings from an office building with a cafe external to the building. Most datasets included WiFi readings taken on multiple floors since it is important that the localization component be able to handle this situation. The number of readings is the total WiFi measurements taken during the duration of the data collection session. For most datasets we collected around 100 measurements of training data per room.

Dataset	Building	Multi-floor	Rooms	WiFi APs	Readings
(Al)	house	yes	6	15	2100
(Ar)	aptm	yes	5	42	2094
(Fe)	house	yes	4	7	1269
(UBC)	multi-bldg	yes	4	189	3138
(Va)	house	no	5	20	1140

Table 2.2: Indoor evaluation dataset summary (training and test).

We can also see the range in the number of visible access points across the datasets. The datasets from houses have relatively few access points, whereas the apartment and especially the office buildings at UBC have many more. We kept the number of rooms trained on somewhat low as we do not anticipate a large number of rooms of interest in most applications. Each dataset consisted of a training and test portion. In each collection we aimed for at least twice as many test observations as training data.

2.5.2 Evaluation

The datasets collected provided a good start for evaluating the localization approaches outlined in section 2.3, however indoor WiFi environments are not static and can easily change over time. Access points may be moved, replaced or shut off after the training data has been collected and so there is a risk that the observation model would become out of date and incapable of properly localizing further observations.

While it would be very difficult to simulate access points changing position as it would require inferring their original position and a propagation model of the signal strength through the indoor environment, a simpler more feasible test would be to remove a random subset of access points from test observation vectors. In other words, simulate as if access points had been turned off and now only reported the absent signal strength value of -100. We created five simulated datasets that are modified versions of the originals where each observation vector has 20% of its access point readings set to -100. The access points selected to be turned off are chosen at random and a new random subset is generated for each observation. Where Va would refer to the original dataset, Va_S refers to the Va dataset with the simulated defects. Access points that would appear in an environment after training would not be considered in the observation. We evaluate room classification and novelty detection on both the original datasets and their respective simulated defect versions.

To assist in manually labeling ground truth novel and known room locations within the test datasets, an application was developed that displayed the sequence of images and used sliders to quickly label ranges of readings as one of the trained rooms or a novel room. Figure 2.7 shows a screen capture of the application ground truthing the (Ar) dataset.

Figure 2.7: A screen shot of the application used to ground truth the location of the WiFi observations and label novel locations

The two sliders under the images allowed us to quickly select large ranges of measurements and label them all at once. Novel measurements were also manually labeled for all three datasets. With the ground truth for datasets, it was now possible to conduct an evaluation on the standard and simulated defect datasets.

Room Classification

To compare the algorithms for room classification, we looked at reporting the accuracy of the classifier which is simply defined as the total measurements correctly classified divided by the total number of test measurements.

$$\text{Accuracy} = \frac{\text{Total Correct}}{\text{Total Test Measurements}}$$

Table 2.3 summarizes the performance of the various algorithms across the datasets. Two of the classifiers, k-nearest neighbour and random forests, required parameters in their specification and so small parameter studies

were done to determine a good selection across the datasets. These parameter studies are summarized in Table 2.4 for k-nearest neighbour where the parameter searched over is k, and in Table 2.5 for random forests where the three parameters searched over are number of estimators, maximum features, and maximum depth. Based on the results in these tables, in the remainder of our analysis for k-nearest neighbours we select k to be 20 and for random forests we select the parameter tuple to be (250, 10, 2). These configurations had the highest average accuracy across the test datasets. The other algorithms did not have relevant parameters.

Table 2.3: Summary of Room Classification Accuracies

Classifier	**Avg.**	**Al**	**Ar**	**Fe**	**UBC**	**Va**	**Al**$_S$	**Ar**$_S$	**Fe**$_S$	**UBC**$_S$	**Va**$_S$
Random Forest	0.916	0.939	0.969	0.898	0.984	0.955	0.863	0.896	0.76	0.98	0.917
Robust NB	0.890	0.789	0.936	0.843	0.992	0.940	0.759	0.922	0.817	0.987	0.910
Chow-Liu	0.870	0.892	0.888	0.897	0.985	0.960	0.713	0.774	0.747	0.981	0.854
K-NN	0.826	0.926	0.796	0.96	0.982	0.955	0.727	0.46	0.757	0.953	0.744
Naive Bayes	0.193	0.189	0.116	0.313	0.216	0.120	0.189	0.116	0.313	0.237	0.120

On analysis of the results, it would seem random forests are the superior approach for this problem. They have the best overall performance on both the normal datasets and the simulated defect datasets. A close second would be Robust naive Bayes and Chow-Liu trees as they too have strong performance across all datasets. K-nearest neighbour has one dataset on which their performance is not competitive. Naive Bayes with Gaussian distributions performs the worst and likely suffers from issues discussed in Section 2.3.1.

Table 2.4: Parameter Study Accuracies for K-NN Classifier

Param	**Avg.**	**Al**	**Ar**	**Fe**	**UBC**	**Va**	**Al**$_S$	**Ar**$_S$	**Fe**$_S$	**UBC**$_S$	**Va**$_S$
(1)	0.813	0.923	0.784	0.943	0.970	0.932	0.699	0.475	0.742	0.931	0.734
(2)	0.817	0.929	0.772	0.943	0.974	0.940	0.706	0.466	0.744	0.931	0.769
(3)	0.817	0.919	0.788	0.952	0.974	0.942	0.702	0.466	0.741	0.930	0.759
(4)	0.820	0.920	0.778	0.948	0.975	0.945	0.703	0.466	0.746	0.951	0.769
(5)	0.817	0.926	0.783	0.946	0.967	0.945	0.700	0.471	0.741	0.938	0.751
(10)	0.818	0.922	0.771	0.936	0.980	0.950	0.706	0.458	0.747	0.949	0.761
(20)	0.826	0.926	0.796	0.96	0.982	0.955	0.727	0.46	0.757	0.953	0.744

Table 2.5: Parameter Study Accuracies for Random Forest Classifier

Params	Avg.	Al	Ar	Fe	UBC	Va	Al_S	Ar_S	Fe_S	UBC_S	Va_S
(50, None, None)	0.7881	0.936	0.893	0.866	0.745	0.882	0.804	0.748	0.722	0.622	0.663
(50, None, 2)	0.9074	0.928	0.938	0.901	0.984	0.96	0.848	0.85	0.762	0.981	0.922
(50, None, 4)	0.8984	0.938	0.951	0.866	0.984	0.955	0.861	0.856	0.728	0.978	0.867
(50, None, 10)	0.8675	0.939	0.935	0.873	0.982	0.887	0.833	0.841	0.73	0.967	0.688
(50, 2, None)	0.722	0.823	0.708	0.863	0.753	0.802	0.69	0.551	0.731	0.651	0.648
(50, 2, 2)	0.8474	0.829	0.777	0.889	0.976	0.932	0.749	0.722	0.752	0.959	0.889
(50, 2, 4)	0.8528	0.877	0.851	0.863	0.966	0.922	0.79	0.77	0.731	0.931	0.827
(50, 2, 10)	0.8192	0.866	0.867	0.863	0.96	0.829	0.742	0.74	0.731	0.913	0.681
(50, 4, None)	0.7663	0.886	0.849	0.873	0.741	0.887	0.756	0.672	0.728	0.613	0.658
(50, 4, 2)	0.9041	0.919	0.933	0.921	0.979	0.945	0.849	0.864	0.763	0.974	0.894
(50, 4, 4)	0.8851	0.932	0.924	0.873	0.971	0.942	0.843	0.851	0.728	0.955	0.832
(50, 4, 10)	0.8571	0.901	0.929	0.873	0.969	0.897	0.768	0.825	0.728	0.94	0.741
(50, 10, None)	0.7855	0.935	0.865	0.868	0.747	0.882	0.809	0.728	0.723	0.63	0.668
(50, 10, 2)	0.9095	0.938	0.949	0.905	0.983	0.96	0.863	0.856	0.762	0.972	0.907
(50, 10, 4)	0.8936	0.938	0.941	0.866	0.974	0.952	0.867	0.856	0.722	0.966	0.854
(50, 10, 10)	0.8677	0.935	0.938	0.87	0.986	0.897	0.824	0.841	0.723	0.965	0.698
(100, None, None)	0.7896	0.935	0.894	0.87	0.738	0.882	0.811	0.75	0.728	0.62	0.668
(100, None, 2)	0.9144	0.936	0.96	0.905	0.977	0.97	0.871	0.88	0.76	0.975	0.91
(100, None, 4)	0.8987	0.936	0.961	0.866	0.98	0.952	0.858	0.874	0.728	0.97	0.862
(100, None, 10)	0.8687	0.936	0.943	0.873	0.98	0.889	0.84	0.85	0.727	0.953	0.696
(100, 2, None)	0.7238	0.829	0.722	0.863	0.749	0.802	0.693	0.553	0.731	0.648	0.648
(100, 2, 2)	0.8482	0.873	0.791	0.846	0.974	0.927	0.802	0.712	0.703	0.957	0.897
(100, 2, 4)	0.845	0.861	0.822	0.863	0.957	0.92	0.771	0.744	0.731	0.932	0.849
(100, 2, 10)	0.825	0.86	0.893	0.863	0.953	0.837	0.736	0.769	0.731	0.917	0.691
(100, 4, None)	0.7729	0.9	0.853	0.873	0.74	0.887	0.774	0.687	0.728	0.621	0.666
(100, 4, 2)	0.908	0.919	0.942	0.911	0.986	0.957	0.848	0.868	0.765	0.982	0.902
(100, 4, 4)	0.8897	0.9	0.938	0.873	0.982	0.947	0.845	0.86	0.728	0.972	0.852
(100, 4, 10)	0.8589	0.9	0.922	0.873	0.975	0.905	0.759	0.852	0.728	0.946	0.729
(100, 10, None)	0.7903	0.934	0.886	0.873	0.737	0.887	0.818	0.749	0.73	0.621	0.668
(100, 10, 2)	0.9119	0.942	0.954	0.901	0.981	0.965	0.871	0.868	0.762	0.973	0.902
(100, 10, 4)	0.9002	0.936	0.957	0.87	0.982	0.952	0.866	0.874	0.73	0.968	0.867
(100, 10, 10)	0.8731	0.936	0.949	0.87	0.979	0.892	0.845	0.852	0.727	0.955	0.726
(250, None, None)	0.7888	0.936	0.887	0.87	0.741	0.882	0.811	0.748	0.723	0.622	0.668
(250, None, 2)	0.9148	0.941	0.96	0.901	0.982	0.957	0.866	0.894	0.762	0.978	0.907
(250, None, 4)	0.9016	0.944	0.956	0.866	0.979	0.955	0.864	0.886	0.725	0.974	0.867
(250, None, 10)	0.8697	0.938	0.945	0.87	0.98	0.887	0.845	0.856	0.723	0.96	0.693
(250, 2, None)	0.7266	0.849	0.707	0.863	0.751	0.802	0.709	0.551	0.731	0.655	0.648
(250, 2, 2)	0.8667	0.87	0.831	0.884	0.974	0.937	0.796	0.766	0.736	0.956	0.917
(250, 2, 4)	0.851	0.827	0.834	0.863	0.972	0.92	0.777	0.783	0.731	0.951	0.852
(250, 2, 10)	0.8237	0.882	0.902	0.863	0.959	0.809	0.744	0.778	0.731	0.903	0.666
(250, 4, None)	0.7712	0.903	0.854	0.873	0.738	0.887	0.768	0.68	0.728	0.62	0.661
(250, 4, 2)	0.9127	0.923	0.954	0.924	0.981	0.952	0.86	0.884	0.766	0.976	0.907
(250, 4, 4)	0.8932	0.911	0.948	0.873	0.976	0.952	0.848	0.884	0.728	0.96	0.852
(250, 4, 10)	0.86	0.901	0.93	0.873	0.982	0.892	0.767	0.857	0.728	0.954	0.716
(250, 10, None)	0.7909	0.935	0.892	0.87	0.742	0.882	0.814	0.761	0.723	0.622	0.668
(250, 10, 2)	0.9161	0.939	0.969	0.898	0.984	0.955	0.863	0.896	0.76	0.98	0.917
(250, 10, 4)	0.8984	0.941	0.953	0.863	0.981	0.952	0.858	0.872	0.725	0.972	0.867
(250, 10, 10)	0.8749	0.941	0.949	0.87	0.982	0.899	0.84	0.864	0.723	0.965	0.716

Novelty Detection

Receiver operator characteristic curves are a useful visualization for information retrieval systems. They are 2D projections of a system's performance in the space of true positive rate and false positive rate. True positive rate (TPR) and false positive rate (FPR) are defined in terms of the system's returned true positive (TP), true negative (TN), false negative (FN), and false positive (FP) counts.

$$TPR = \frac{TP}{TP + FN}$$
$$FPR = \frac{FP}{FP + TN}$$

For our novelty detection system, true positives are defined as correctly classified WiFi observation inliers (not novel), true negatives are defined as correctly classified observation outliers (novel), false positive are incorrectly classified outliers, and false negatives are incorrectly classified inliers.

A good classifier is one that has a high true positive rate with a relatively low false positive rate. All the approaches described in section 2.4 are parameterized by α, the fraction of the novelty reported for z_{neg} to be used as the novelty threshold. We searched the parameter space of α and plotted the TPR and FPR of the system with each parameter setting.

ROC curves for the three test datasets that had novel readings labeled are summarized in Figure 2.8. On analysis of the results, we see that most approaches are competitive on the datasets. However, the performance of all approaches is somewhat poor on the (Al) dataset, and this is likely due to the fact that the novel readings were collected on the second floor directly above the first floor which the observation model was trained on. It would seem that all of these approaches have a difficult time discerning between observations made on the first floor (inliers) and observations made on the second floor (outliers).

In terms of overall performance, the best approach would likely be the centroid thresholds using Euclidean distances. While competitive on the other datasets, the Mahalanobis distance seems to suffer on the UBC environment which models larger rooms and has many more visible access points. The robust Bayes threshold seems competitive on all but the (Al) dataset. The Chow-Liu tree threshold is the weakest of the five and never achieves competitive performance.

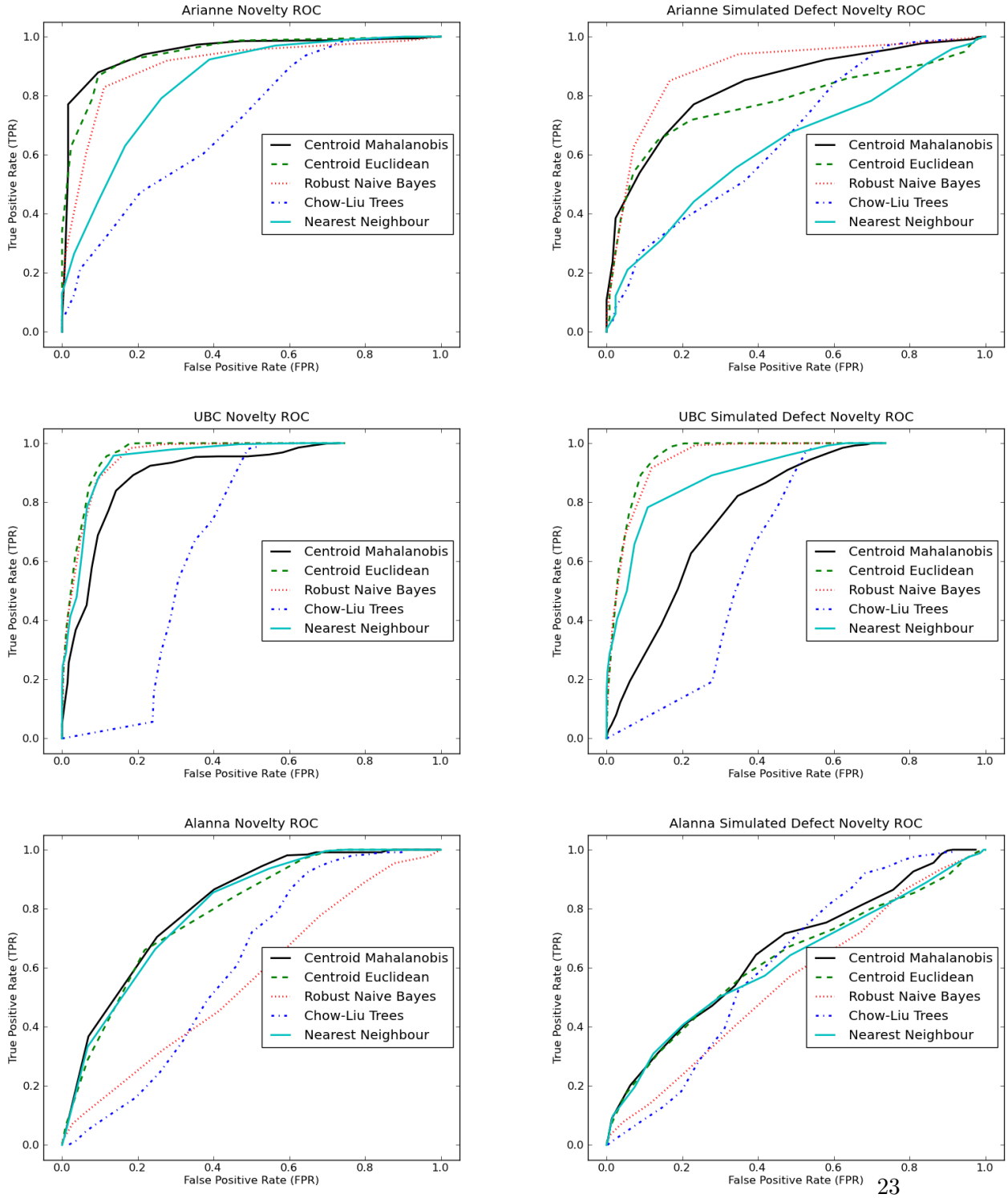

Figure 2.8: ROC Curves for Novelty Detection

Chapter 3

Activity Recognition

Context awareness is an ever improving feature of assistive technologies and ubiquitous computing. Inference of user activities and environmental interactions remains an important problem if only to serve higher level goals of a particular system or application.

In particular, the idea of using mobile phones and other wearable devices to perform actigraphy (non-invasive activity monitoring) has become a common theme in mobile health applications. Standard mobile phones are now well equipped to measure and report summaries of human rest and activity cycles. While most of these systems have focused on ambulating users only, wheelchair users would certainly also benefit from similar summaries.

Figure 3.1: The activity classes of interest for our system. Stationary, walking, wheeling and vehicle use

We extended the approach of activity recognition proposed in [15] and introduced the ability to classify wheeling motion for manual wheelchair users in addition to the other classes shown in Figure 3.1. We have made no attempt to optimize the approach and it could easily be substituted for another. A decision tree is learned that considers a combination of accelerometer and GPS features to infer activity. The approach computes features on 2.8 second windows of accelerometer data sampled at 20Hz and combines them with an average speed estimate from a stream of GPS fixes. We trained on data collected with the phone placed in the subject's pocket and backpack to reduce dependence on specific placement or orientation of the phone. The algorithm works offline to analyze accelerometer and GPS streams over the course of an entire day and is therefore able to correctly classify vehicle motion when GPS might be temporarily unavailable; for example, when taking the subway or driving through a tunnel.

3.1 Related Work

In much the same way that section 2.3 posed the localization problem, the problem of activity recognition can broadly be seen as performing classification on sequential sensor data. A good summary of modern activity recognition techniques is presented in [5]. Two general methods are defined for creating activity recognition models. Data-driven approaches where a model is learned from training data is contrasted with knowledge-driven methods where the model is primarily created from prior domain knowledge and expertise. The overview in [5] discusses the generative models: naive Bayes classification (NBC), hidden Markov models (HMMs), general dynamic Bayesian networks (DBNs), as well as some useful discriminative models : decision trees, support vector machines (SVMs), conditional random fields (CRFs), and nearest neighbour (NN) approaches. Work in [43] presents a tutorial on CRFs applied to activity recognition and compares results with HMMs on a simulated dataset. Work in [26] extracts activities and significant places from GPS traces using hierarchically structured CRFs. We adopted decision trees for their simplicity and ease of interpretation.

Work from [3] looks at wearing wireless accelerometers on the body, discusses the difficulties in obtaining labeled data for genuine activities outside of a lab environment and demonstrates the effectiveness of decision trees. More recently, [38] compares various machine learning approaches to solve the problem of gait recognition efficiently on mobile phones.

State of the art classification on mobile devices is achieved in [12] with a method called geometric template matching (GeTeM) which uses time-delay embeddings to construct a dynamical systems model capable of scoring similarity between time-series data.

Activity recognition has also moved well beyond being a purely academic endeavour. Most deployed, working systems rely on analyzing data streams of accelerometer readings. Wearable actimetry systems have existed in the consumer space for some time. Two recent examples of popular wearable devices for activity monitoring include the Fitbit [18], and Nike+ Fuelband [23]. Google Now summarizes distances biked and walked [21] and very recently, Google introduced activity recognition into Android location APIs [19]. Pedometers and cyclometers also been available for quite some time, however they only have limited recognition capability (steps and rotations).

While these products are marketed towards consumers as quantified self or training tools, activity traces are certainly also of interest to the assistive technology, gerontology, and rehabilitation science communities ([40], [37], [15]). It is this application domain that we are concerned with.

3.2 Sensor Overview

Modern smartphones are certainly impressive in terms of their array of sensors. While the sensors are low-cost and consumer-grade, their level of integration within the device and operating system is attractive. For instance, most smartphones come equipped with accelerometers, gyroscopes, magnetometers, GPS receivers as well as light and proximity sensors, recently even barometers. It is quite remarkable that what were once expensive inertial measurement units only available in the domains of aerospace and military engineering now sit in low-cost form in nearly everyone's pocket. For the activity recognition component of our system, we employed the accelerometer and GPS receiver.

3.2.1 Tri-Axis Accelerometer

An accelerometer is a device capable of measuring acceleration forces. They exist on most mobile devices as a means of controlling the user interface, sensing display orientation or detecting falls for hard drive protection.

The signal that we initially compute is total force magnitude. The gravitational constant g is subtracted from that scalar value to put the mean value near zero. The resultant signal is neither the acceleration nor the applied force without gravity, since that would properly require a vector subtraction, which is not possible with the data available on the device.

$$m = \sqrt{X^2 + Y^2 + Z^2} - g$$
$$g = 9.81\ m/s^2$$

However, the signal is useful for our purposes, and this usage is common in the field. This is the same univariate signal used in [15]. This approach has the advantage of producing values that should be independent of the orientation of the phone, which allows for greater flexibility in placement and orientation by the user. It also reduces the amount of data produced by the system as we only store a single float to represent a sensor reading. For sample windows of this data stream see Figure 3.2

The Android API specifies several high level sample rates that can be set when using the accelerometer [22]. We selected one that gives us ∼20Hz sampling rate on Nexus 4 phones.

3.2.2 Global Positioning System

The global positioning system or GPS is a satellite navigation system that is capable of locating users anywhere that there is line of sight with at least four of the GPS satellites orbiting Earth. A fix typically consists of the latitude and longitude of the receiver, usually represented by two floating point numbers. Latitude is an angle which ranges from -90° to 90° (0° at equator), and longitude ranges from westmost -180° to eastmost 180° (0° at Prime Meridian). For example, downtown Vancouver is [49.2839, -123.1200].

$$\text{fix} = [LAT,\ LON]$$

Limitations of GPS include either too few satellites being visible or the signal being reflected too many times and increasing its time of flight unpredictably. The second effect often occurs in downtown areas with many tall buildings. When too few satellites are visible the receiver will be unable to determine a fix. GPS therefore does not work well indoors or underground. However, GPS repeaters can be installed to provide indoor functionality [24]. In addition to determining a fix using only the receiver, most location services on mobile phones allow for improved positioning by consulting large databases of stored associations between known WiFi access points and cell towers and their locations. If the device is connected to the Internet, it can consult this service and receive potentially more accurate location information. However, we only considered unassisted GPS readings in our experiments. Often location services can also return estimates for speed, bearing and altitude with a reported accuracy of these measurements [19].

3.3 Feature Selection and Sensor Fusion

As previously discussed, the problem of activity recognition can be posed as a classification of sequential sensor data. In section 2.3 the features used for observations were simply the raw signal strengths associated with the WiFi access points. While one could use raw sequential data as features in this context, better results typically are achieved by pre-processing the sequential data into a smaller set of representative features.

$$z = [\ f_0,\ f_1,\ \cdots,\ f_8\ ]$$

We propose an observation vector made up of nine features: eight are extracted from fixed-length windows of accelerometer measurements and one incorporated GPS measurements, nearly the same as [15]. In this way, sensor fusion is handled naturally by the classification algorithm.

3.3.1 Accelerometer Features

The most common approach for analyzing sequential accelerometry data is to break the stream up into non-overlapping windows and perform the classification on those smaller series [12] [38] [15].

The first aspect of this approach that must be determined is the time length of the window. We chose 3 seconds for the window length because it lies between the 1 second time span used in [15] and results shown in [40] that show most manual wheelchair bouts of mobility last 5 seconds.

$$\text{ACC} = [\ m_0,\ m_1,\ \cdots,\ m_{N-1}\ ]$$

The window, ACC, is simply N real numbers representing the motion signal as defined in section 3.2.1, where N depends on the desired time span of the window and the sampling frequency. In our case, sampling at 20Hz with a desired window length of 3 seconds leads to N = 60 samples for each window. We can now consider features for this small time-series.

One simple feature of a series is the variance. This is the first feature we consider and is defined below in equation 3.1, where μ_m is the mean of the time series.

$$f_0 = \frac{1}{N-1}\sum_{n=0}^{N-1}(m_n - \mu_m)^2 \tag{3.1}$$

Fourier coefficients can also act as useful features for signal analysis. The Fourier transform is a standard approach that gives a breakdown of the frequency components present in a time-domain signal. Other similar but more efficient approaches include the Goertzel algorithm used in [15] or counting zero-crossings as a simple approximation to the signal's fundamental frequency.

$$X_k = \sum_{n=0}^{N-1} m_n \cdot e^{-i2\pi kn/N} \tag{3.2}$$

$$|X_k| = \sqrt{X_k.Re^2 + X_k.Im^2} \tag{3.3}$$

$$f_1 = \sum_{n=0}^{N-1} |X_k| \tag{3.4}$$

We employ the standard discrete Fourier transform and compute the N complex coefficients as defined above. The magnitude of the coefficients is known as the spectrum and we include the energy (3.4) as a feature.

In addition to an aggregate feature like energy, it would also be useful to include some information about the individual frequency components of the signal as features. Typically in spectral analysis, the sample frequency is plotted with the spectral response. The N sample frequencies for a discrete Fourier transform can be produced as shown in (3.5) , where d is the sample spacing [29]. In our case we simply leave $d = 1$.

$$\text{FREQ} = [\ 0,\ 1,\ \cdots,\ N/2-1,\ -N/2,\ -(N/2-1),\ \cdots,-1\ ]\frac{1}{d\cdot N} \quad (3.5)$$

$$\text{SPEC} = [\ |X_0|, |X_1|, \cdots, |X_{N/2-1}|, |X_{N/2}|, |X_{N/2-1}|, \cdots, |X_1|\ ] \quad (3.6)$$

We then select the three largest sample frequencies in terms of their spectral response as features and include the corresponding spectral responses as features as well. Due to the symmetry of the response we need only consider N/2 elements of SPEC

$l_0 =$ index of largest element in SPEC[0:N/2]
$l_1 =$ index of 2nd largest element in SPEC[0:N/2]
$l_2 =$ index of 3rd largest element in SPEC[0:N/2]

$f_2 = \text{FREQ}[l_0]$ $\qquad$ $f_5 = \text{SPEC}[l_0]$
$f_3 = \text{FREQ}[l_1]$ $\qquad$ $f_6 = \text{SPEC}[l_1]$
$f_4 = \text{FREQ}[l_2]$ $\qquad$ $f_7 = \text{SPEC}[l_2]$

To better illustrate these features we considered the sample windows from three of the activity classes shown in Figure 3.2

As can be seen from the figures, the variance, energy and spectrum responses are much larger in the walking examples than the other two classes. In terms of differentiating wheeling from stationary the variance and energy are also much larger in the former. It seems as though even just the variance and energy would be enough to reliably distinguish these classes.

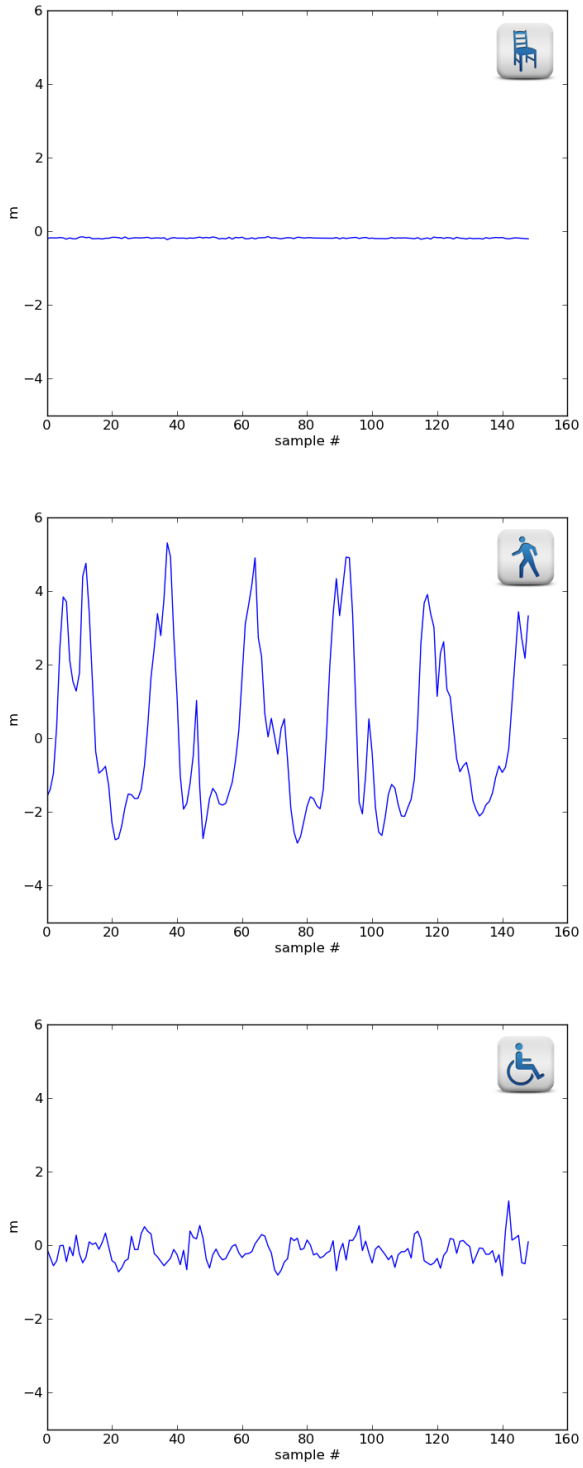

Figure 3.2: Sample motion signal windows for three activity classes.

3.3.2 GPS Feature and Sensor Fusion

The accelerometry features seem suitable for differentiating between the first three motion classes, however the classifier requires a way to distinguish vehicle motion from the other profiles.

This is addressed in [15] by including an instantaneous speed estimate returned from the GPS receiver and phone location service. We propose an offline approach which assumes that a sequence of GPS fixes are available. Specifically, we generate a list of time intervals where each interval has an average GPS speed associated with it. Assuming we have a list of all K GPS fixes taken throughout the day in order, we generate a list of average speed intervals. This interval list consists of a 3-tuple of the format (start time, end time, average GPS speed). The Haversine formula is an equation important in navigation, giving great-circle distances between two points on a sphere from their longitudes and latitudes [44].

Program 3.1 Function to generate average speed list

```
def genAvgSpeed(GPS_FIX):
    idx = 0
    AVG_SPEED = []
    while ( idx <= K ):
        nxt  = findFutureFix5Min( GPS_FIX, idx )
        dist = haversine( GPS_FIX[idx], GPS_FIX[nxt])
        start_time = GPS_FIX[idx].time
        end_time   = GPS_FIX[nxt].time
        avg_speed  = dist / (end_time - start_time)
        AVG_SPEED.append( ( start_time , end_time , avg_speed ) )
        idx = nxt
    return AVG_SPEED
```

Intuitively, the stream of GPS fixes are broken down into intervals of 5 minutes or greater and the average speed is calculated for this interval. This approach has several advantages over the use of the instantaneous estimates from the phone's location service. The instantaneous speed estimates are often inaccurate, and GPS readings may be temporarily unavailable for periods of time during travel for a number of reasons. For example, traveling underground in a subway, long tunnel or other situations where GPS data is only sporadically available will misclassify vehicle motion when using the instantaneous approach.

The final feature for the activity classifier is determined from where the non-overlapping accelerometer windows lie in the GPS data stream. The average speed of this interval is returned as the last feature of the observation vector. This process is illustrated in Figure 3.3

$$f_8 = \text{findSpeed}(\text{ AVG\_SPEED, ACCWindowTime}) \tag{3.7}$$

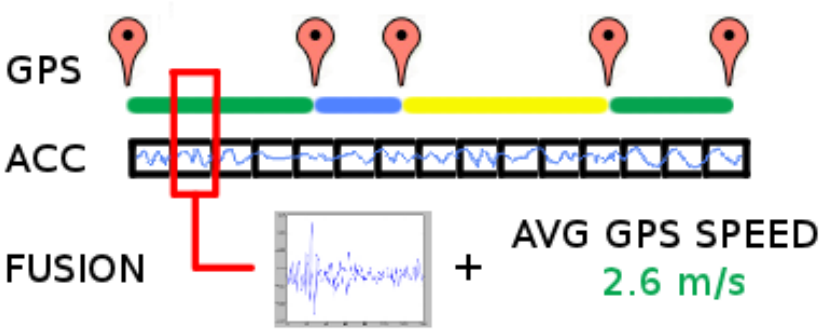

Figure 3.3: Final feature vector is produced for each accelerometer signal window by combinining accelerometer features with the average GPS speed recorded during the same time interval

3.4 Experiments

We are primarily interested in having an activity recognition component in our overall system. To build a representative dataset, ideally one would collect data from multiple subjects on multiple devices. The classifier we chose to evaluate is the decision tree as employed in [15]. The classifier is not state of the art, but works reasonably well for our purposes and could be replaced with a superior approach.

We compare our ability to classify wheeling motion with a wheel-mounted actimetry system similar to the one proposed in [40]. We also compare our ability to classify vehicle motion by using our proposed offline approach versus one that employs speed estimates returned by the Android location services.

3.4.1 Data Collection

Accelerometry and GPS data streams were collected using a Nexus 4 Android phone and the MobiSense App 4.3. The phone was carried in a variety of positions and orientations as shown in Figure 3.4

Wheeling data was collected using a manual wheelchair and included indoor and outdoor movement on a variety of surfaces around the UBC campus. The subject was, however, not an actual wheelchair user. Walking data was collected while moving in and out of buildings. Effort was made to walk with different gaits and speeds. Vehicle motion data was collected while traveling on public buses in Vancouver. Stationary data was collected with the phone left on a desk. All data was collected by a single user on a single phone.

Figure 3.4: The four configurations for which the training and test data were collected. Wheeling and walking training data was collected with the phone in a pocket and a backpack

3.4.2 Evaluation

With the dataset collected, we split the data into 75% training and 25% test. A decision tree was learned using the CART algorithm as provided in the sci-kit learn library for python [32] with a maximum depth of 3 decision nodes. The learned tree is shown in Figure 3.5. Decision trees have several nice properties: they are inexpensive to store, fast to evaluate, and easy to interpret.

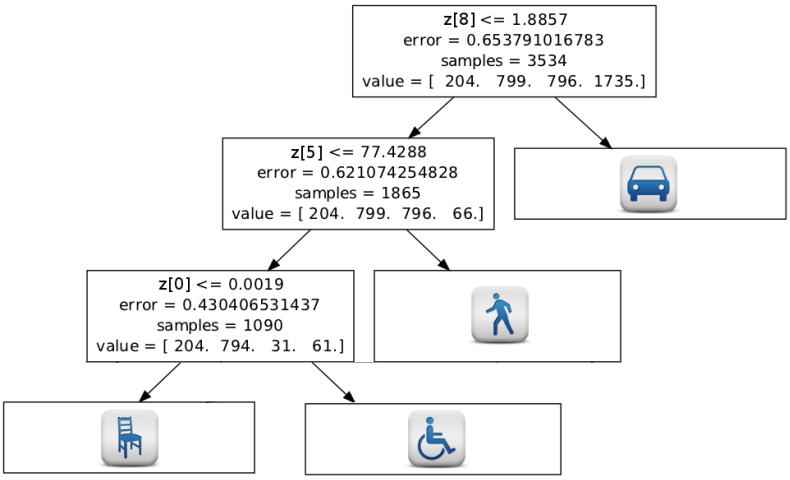

Figure 3.5: Visualization of the final activity classification decision tree. Left branches are taken when condition is true, right branches when false.

The learned tree gives us some insight into the activity classes. The first decision is made on the average GPS speed feature. This feature value is in meters per second so we can see that if the GPS speed is above 1.89 m/s or 6.77 km/h then the tree will report the motion as vehicle. This may seem too low perhaps, but there was no training data for jogging or running present and this threshold could easily be adjusted. For the other three classes the two features selected were the variance of the signal and the largest of all coefficient magnitudes. It is interesting that the energy and actual frequencies were not found to be useful by the algorithm.

To visualize the results of the classifier on the test dataset we employed a confusion matrix incorporating the four classes of interest shown in Figure 3.6. The results are quite promising for this simple approach in that an average of 97% accuracy is achieved across the classes. Most of the confusion lies between walking and wheeling.

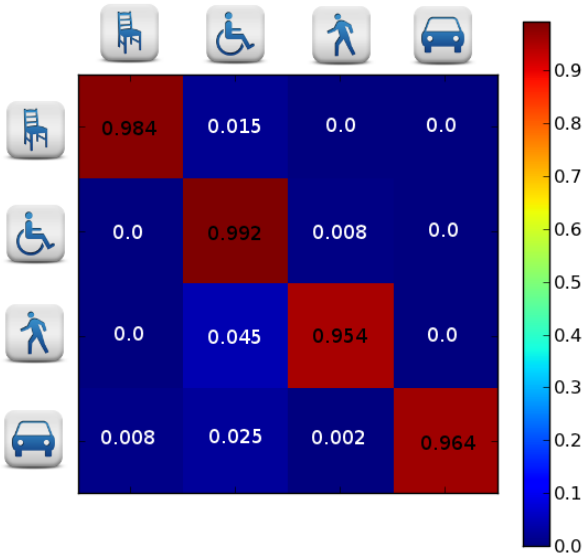

Figure 3.6: Confusion matrix across activity classes of interest. Rows and columns represent the ground truth and classified activities respectively.

If instantaneous GPS speed measurements are used instead of our approach, the vehicle motion accuracy drops to around 67% on the test set with high confusion between vehicle motion and wheeling.

Likely the main issue with respect to misclassification in our approach is that there was no noise or fidget data present in the training set. The datasets were continuous recordings of the specific activity. For example, when the data for walking was collected, no time was spent standing still waiting briefly or tapping a foot impatiently. For the same reasons explained in [3] it is difficult to capture and ground truth authentic fidgeting. In the classifier's current state, such actions would likely be classified as wheeling since they would have low average GPS speed, low spectral response and slightly higher variance than a stationary signal. The classifier would also likely be more robust if it had been trained using multiple devices and users.

To further validate our classifier, we compared our results with a summary produced by another wheelchair activity measurement system. This system used an accelerometer mounted to a wheel. The signature from a wheel mounted accelerometer is much cleaner and obvious when compared to phone accelerometry measurements made from a user's pocket or bag. In Figure 3.7, we see similar reported time periods of activity and inactivity.

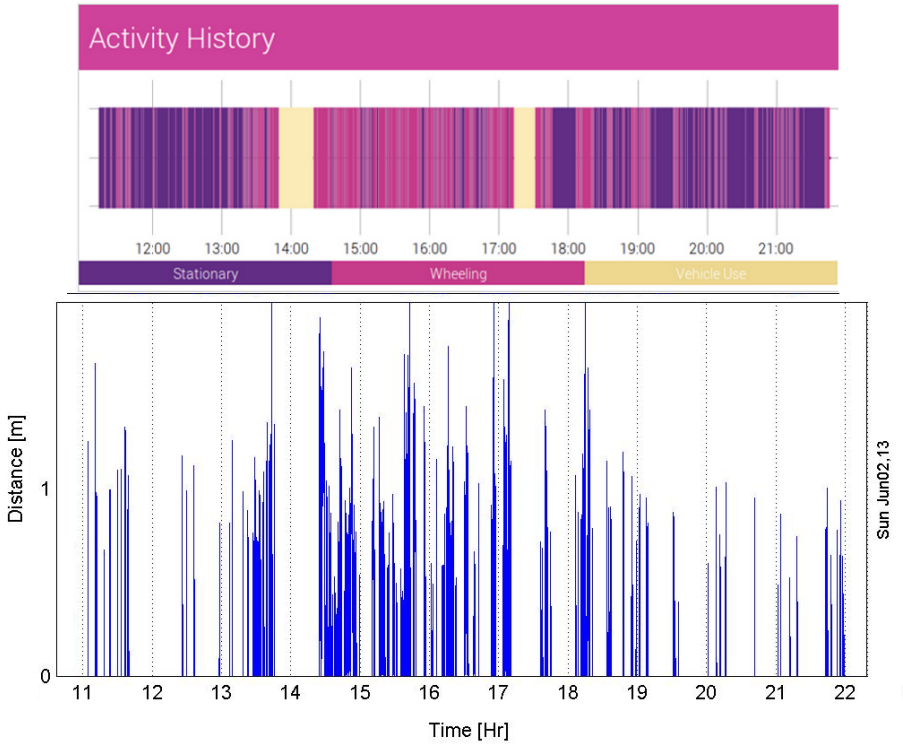

Figure 3.7: Comparison of phone based activity classifier (top) with custom wheel mounted actimetry system (bottom).

Chapter 4

MobiSense System

Mobile technology has reached a level of sophistication and ubiquity that it now presents an attractive platform for assistive technology research and health-related applications. Mobile devices support a rich integration of sensing, display, and Internet connectivity hardware and have become widely adopted by the general public.

The electronic health or eHealth movement has made great strides in improving access to health care information by using Internet technologies. An obvious next step in this progression is to leverage both Internet and mobile technologies and investigate mobile health, or mHealth applications [27] that move towards further personalization and assistive capabilities. In a more general sense, the quantified self movement [39] aims to promote self-improvement through consistent data collection and analysis. The underlying philosophy of the quantified self movement is that consistent measurement and reflection of important personal metrics leads to an almost instinctive improvement in behaviour. With sensing and recording becoming easier than ever, there is a growing number of quantified self tools for improving lifestyle or behavioural habits such as diet, exercise, and time management. Of particular interest to assistive technologists and rehabilitation scientists is the measure of a person's mobility, which is an important factor in assessing quality of life [37].

MobiSense is a mobile health research platform that aims to improve mobility analysis for both ambulating and wheelchair users. The goals of the system are to be simple for users to collect mobility data, provide easily accessible summaries and analysis of daily behaviours and finally to enable further research and development by providing a sandbox environment for rapid prototyping and experimentation. The paradigm of distributed data collection and centralized analysis stands to greatly benefit health care and assistive applications. With cloud computing and storage services becoming cheap and accessible, systems that harness these resources could be of great interest to health researchers and professionals. Investigating systems of this nature requires a broad skill set; including familiarity with machine learning, user interfaces and distributed systems.

4.1 Related Work

Several systems have been built in recent years that explored mobility sensing and reporting to varying degrees. One of the earlier realized systems was UbiFIT [7], a project designed to track physical activity and provide on-phone visual feedback with a simple intuitive display. Users wore a separate pedometry device that communicated with the phone to summarize activity using a visualization that was quick to interpret. As sensor integration with phones became more sophisticated, systems relied less on external hardware and solely on integrated sensors within the mobile device. FUNF, an open source sensing framework for Android [33], provided an extensible framework for capturing a wide variety of sensor data from a device and publishing it to a web service. HumansSense [13] is an open source data collection platform on Android, capable of logging sensor data and periodically uploading to a web service. Work done by [37] demonstrated a proof of concept system that utilized pre-placed bluetooth beacons in the home to infer room level localization, as well as activity levels using mobile phones. This work also presented several succinct visualizations for summarizing the time series lifespace data. A custom wearable device with GPS and activity logging functionality was proposed in [4]. This device was designed to be used for lifespace reporting and recorded logs on a memory card to be removed and analyzed later.

Recently, the open mHealth initiative [27] has proposed a standardized architecture for mHealth and reporting applications. Ohmage, their participatory sensing platform [15], allows for data collection on the phone with the ability to upload data to a web service for summarizing and visualizing reports for users. The application has been used to track users' activity levels and has also been applied in other studies relating to post-traumatic stress disorder and monitoring stress levels in new mothers [30].

MobiSense builds on the HumanSense project and couples it with a new web service in the spirit of Ohmage and the open mHealth architecture. The system provides user feedback and summarization visualizations through a custom web client. Many of the visualizations are styled after the lifespace visualizations presented in [37]. The system uses an off the shelf Android Nexus 4 phone with no hardware modifications and the web client can be viewed from any browser. To our knowledge, MobiSense is the first lifespace tracking system running on standard mobile phone hardware that provides indoor, outdoor and activity summarizations for both ambulating and wheelchair users.

4.2 System Architecture

The open mHealth initiative identifies three key abstractions in the architecture of an mHealth system [27]. Data storage units (DSUs) define how the data is stored and accessed, data processing units (DPUs) perform transformations and summarizations on the collected information and data visualization units (DVUs) define graphical representations. While MobiSense does not adhere exactly to the specifications of the units outlined by the initiative, in part because the specifications are still in flux, they provide a useful frame of reference for discussing the system's components and certainly future work could be done to achieve full compliance. Figure 4.1 outlines the system's overall architecture and data flow. Sensor data is collected on the user's Android phone and uploaded to a web service hosted in the cloud. The data is processed on the server and transformed into succinct summarizations. A web client can then request those summaries for visualization to the user. Users login to the web client with the unique identifier (UID) associated with each phone.

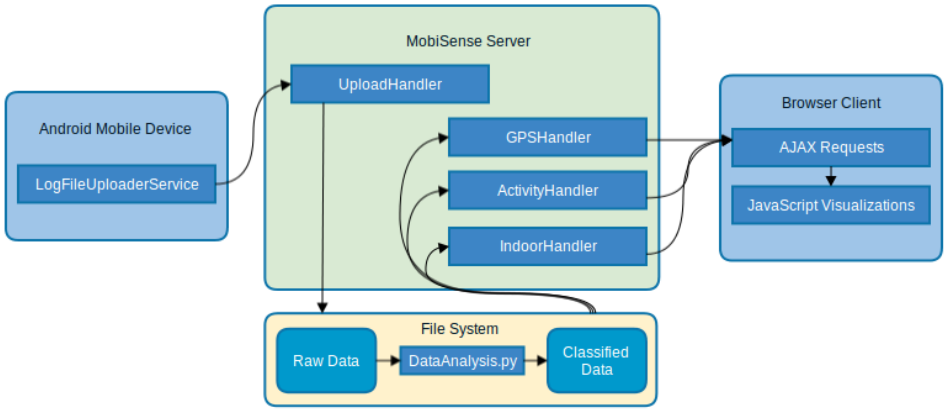

Figure 4.1: Overview of MobiSense system architecture

Several decisions in the design of the system were influenced by the preference for a flexible, research friendly environment. The operating system's file system was used as the primary storage unit instead of a database to reduce technical overhead and complexity. The data processing could be done in the mobile application, but a centralized approach eased experimentation and prototyping. Python scripts running on the server analyze and transform stored data and are relatively simple to write and experiment with. Similarly, JavaScript visualization libraries are easy to use and customize for the web client.

4.3 Phone Application

The MobiSense phone application is a modification of the HumanSense project [13]. HumanSense logs many of the sensors supported by Android but for the lifespace measurements that MobiSense focuses on only WiFi, GPS and accelerometer readings are stored. The readings are stored as binary files compressed with gzip on the phone's SD card.

4.3.1 Sensor Logging

Each sensor can be configured with a different polling rate and MobiSense sets suitable defaults for the different streams based on the way the data is used. The GPS sensor is polled once a minute and records the following:

```
<unixtimestamp> (8 bytes, long int)
<accuracy> (4 bytes, float)
<bearing> (4 bytes, float)
<speed> (4 bytes, float)
<altitude> (8 bytes, double)
<latitude> (8 bytes, double)
<longitude> (8 bytes, double)
```

Here, `<unixtimestamp>` is a standard unix timestamp and represents the milliseconds since Jan 1 00:00:00 1970 GMT. Longitude and latitude represent the GPS fix as described in Section 3.2.2. Accuracy, bearing, altitude and speed are other estimates returned by the GPS API in Android, however MobiSense does not make use of these. The WiFi modem is polled every 10 seconds for MobiSense and stores the following data on file:

```
<numAP> (4 bytes, int)
<unixtimestamp> (8 bytes, long int)
(numAP entries):
    <signal_strengh> (4 bytes, int)
    <SSID_len> = (2 bytes, int)
    <SSID_string> = (SSID_len bytes, string)
    <BSSID_len> = (2 bytes, int)
    <BSSID_string> = (BSSID_len bytes, string)
```

Since the number of visible access points varies from reading to reading, each entry is potentially of a different length. Here, `<numAP>` encodes the number of access points observable for the current reading. The signal strength, SSID, and BSSID fields are described in Section 2.2.

The accelerometer is polled at a rate of 20Hz and records the following:

```
<unixtimestamp> (8 bytes, long int)
<mag> (4 bytes, float)
```

Here `<mag>` represents the accelerometer signal as defined in Section 3.2.1. The final important group of files created by MobiSense is the training data used to learn a classifier for indoor localization as outlined in Section 2.3. The training file is stored in human-readable plaintext and records labeled WiFi observations in the following format:

```
R <unixtimestamp>
L <roomName>
<BSSID_0> <SS_0>
...
<BSSID_N> <SS_N>
```

Training files are generated whenever the user decides to add a new room to their localization model or to record more training data for a given room.

Since the application is expected to collect data over the course of a full day, an important consideration is the amount of data that the sensor logging would produce. As it is polled most frequently, the accelerometer sensor stream creates by far the most data. For an 18 hour recording period with full accelerometer, WiFi logging and GPS logging ∼15 MB of gzip compressed sensor data would be collected (close to 50 MB uncompressed). Only the compressed files are ever uploaded to the web service, and only when the phone has an Internet connection through WiFi to prevent accidental transfer costs on a user's data plan. Once the data has been processed as explained in section 4.5, its summarization format is much more compact, each day taking up only ∼100 KB.

Another important consideration for any mobile application is battery usage. With little screen use and no calls made or received, MobiSense was able to run on a single battery charge on a Nexus 4 phone for 22 hours.

Sensor logging files are continuously updated until either the user stops recording or the current time reaches midnight. If the user has not stopped recording at midnight, new logging files are created to reflect the start of a new day and logging continues using these new files. An upload service periodically checks for an Internet connection over WiFi and if there are un-uploaded files, it sends HTTP post request with the gzipped sensor files as the body. The user can also manually start the uploading process. Along with these files, the timezone of the device is also uploaded to the service.

4.3.2 User Interface

To simplify the application's use, a basic interface was designed to provide hooks into the most important actions associated with the application. Figure 4.2 below is a capture of the main screen.

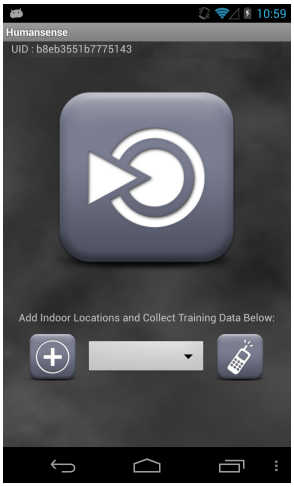

Figure 4.2: Screenshot of the MobiSense phone application.

Displayed in the top left of the screen is the UID (unique identifier) associated with the device that is used to differentiate data streams and login to the server. MobiSense also allows the user to change the UID of the device, which allows for several different models and logs to be created while still using the same device. The central button is used to start the logging service. When pressed, the application will begin recording sensor streams and storing the data locally on the device. To stop the service, the user presses the same button again and another service will attempt to upload the recorded files. The user can also manually trigger this upload service. Below the central start button, is the area of the UI used for creating an indoor location model. The plus (+) button is used to create a new labeled room. Pressing it will create a dialog where the user can enter the name for a new room they would like to train on (e.g. "kitchen"). The drop down menu after the plus button lists all the labeled rooms, so the user can add more training data to a previously created room. Finally, the phone button starts recording WiFi observations for training data on the currently selected room. It will record observations every 800 ms and store the training data in the format described in Section 4.3.1

4.4 Data Storage

The centralized processing and storage of the data streams is done on a single Amazon EC2 instance running Ubuntu Linux. The Tornado web server is used to handle data uploads and requests for lifespace summaries. When the MobiSense phone application uploads compressed sensor data to the server, the upload handler decides where to store the data on disk. Figure 4.3 below outlines the organization of the filesystem on the server.

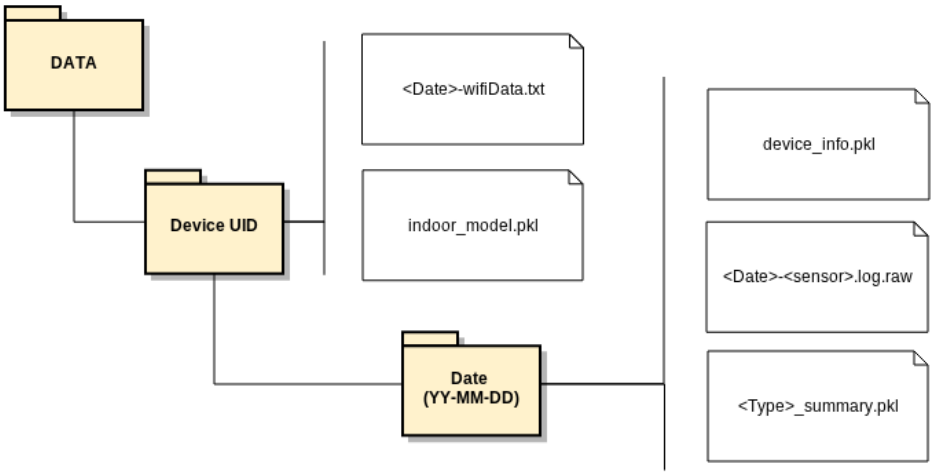

Figure 4.3: Filesystem Organization on MobiSense Server

Data and summaries are grouped by device UID and date. Indoor training files (<Date>-wifiData.txt) are stored for each device, where <Date> represents a timestamp for the file. The indoor WiFi classifier and novelty detector described in Chapter 2 are represented in a python pickle file (indoor_model.pkl). A shared activity model (not device specific) stores the decision tree classifier explained in Chapter 3 for activity recognition. The file device_info.pkl stores the timezone that the device was set to. Sensor data is uploaded as compressed gzip files which are placed into folders corresponding to the date they were created. The gzip files get decompressed to .raw files and then summary pickle files (<Type>-summary.pkl) are generated using the raw uncompressed data. Three summary files are generated: one for activity, indoor, and outdoor data. The process that generates the summary files is discussed in more detail in the following section.

4.5 Data Processing

Data processing is scheduled on the server using a single cron job that searches the filesystem for unprocessed files. The steps at a high level are shown below in Program 4.1. The job is scheduled to run every three minutes and there is a locking mechanism in place to ensure that no new job begins execution while another is still running.

Program 4.1 Data Analysis Process

```
FILES_2_UNZIP = collectNewCompressedFiles()
gunzip( FILES_2_UNZIP )

NEW_INDOOR_TRAINING_FILES = collectNewIndoorTrainingFiles()
updateIndoorModels ( NEW_INDOOR_TRAINING_FILES )

NEW_GPS_FILES  = collectNewGPSReadings()
NEW_WIFI_FILES = collectNewWifiReadings()
NEW_ACC_FILES  = collectNewAccelerometerReadings()

doOutdoorSummary( NEW_GPS_FILES )
doActivitySummary( NEW_ACC_FILES )
doIndoorSummary( NEW_WIFI_FILES )
```

The unzip step decompresses all sensor stream files ending with a .gz extension and stores the decompressed data with .raw extension.

All new indoor training files are processed and then converted from .txt extensions to .proc (for processed) to ensure that no file is processed twice by the server (.proc files are ignored).

The sensor streams are then analyzed and summarized by running scripts on the uncompressed .raw files. The .raw files are then re-labeled as .proc and the scripts create or update their respective summary pickle files.

Another important consideration is the data dependencies between the different summarizations. As explained in Chapter 3, the activity recognition relies on having the GPS stream already processed to allow for sensor fusion. The data analysis process schedules the GPS sensor streams to be analyzed before the accelerometer streams. Similarly, the indoor summary could likely benefit from having activity summary information and so this is generated last. The device_info.pkl file is created by the web server when the upload requests from the phone application come in.

4.5.1 Outdoor Summary

The outdoor summary stores the sequence of GPS fixes, the average speed for given time intervals and the total km traveled over the day. The structure of the summary file is a dictionary with three main entries:

```
RAW : [ <time>, <lat>,  <lon>, <s_est>, <acc>, <t_delta>, <d_delta> ]
INT : [ <start_time>, <end_time>, <avg_speed> ] (intervals >= 5min)
STATS : { TOT_FIXES : <num_fixes>,  TOT_KM : <tot_km_travelled> }
```

The `RAW` entry is a list of GPS fixes, with information relating to accuracy, speed estimate, and the time and distance from previous entry. The `INT` entry is a list of time intervals and the average speed calculated over that interval. As explained in Section 3.3.2, this information is needed for activity recognition. The `STATS` entry is itself a dictionary and stores the total fixes and distance travelled for the day.

4.5.2 Activity Summary

The activity summary has a similar dictionary structure with two entries.

```
INT : [ <start_time, <end_time>, <class>  ] (intervals >= 2s)
TOT : { <class> : <tot_time> }
```

The decision tree outlined in Chapter 3 is used to process windows of accelerometer readings fused with average GPS speed, to decide on the activity class for each interval. If two subsequent intervals have the same classification, they are condensed into one larger interval with the same classification. Krumm suggests using a two-state HMM to smooth transitions between active and stationary states [24]. For our purposes, motion classifications less than 2.5 seconds in duration are suppressed if surrounded by stationary classifications.

4.5.3 Indoor Summary

The stream of WiFi observations is processed in a similar way using the novelty detection and classifier model outlined in Chapter 2.

```
INT : [ <start_time>, <end_time>, <room_name> } (intervals >= 10s)
TOT : { <room_name> : <tot_time> }
```

Each interval that is identified as not novel has a room associated with it. The `TOT` entry stores the total time spent in each room for the day.

4.6 Data Visualization

One of the goals of MobiSense was to provide users with rich summaries of their daily patterns and behaviours. Visualizations play an important role in communicating this information and careful thought is required to present succinct, quickly interpretable charts and plots. A visualization component is present in many lifespace and actigraphy applications and is key to guiding the user's exploration and interpretation of their own data [15] [37] [18] [23].

As outlined in Section 4.2, all the logic for the visualizations lives in the web client which is written in HTML and JavaScript. Summarization data is requested from the server, and returned in human readable JSON format which is then rendered into the appropriate visuals.

4.6.1 Timeline Selection

The first step before data can be visualized is for the user to specify the data range to be summarized. A slide bar with two adjustable ends allows for selection over the range of dates that the user collected data. Shown below in Figure 4.4 is a screen shot of the timeline selector.

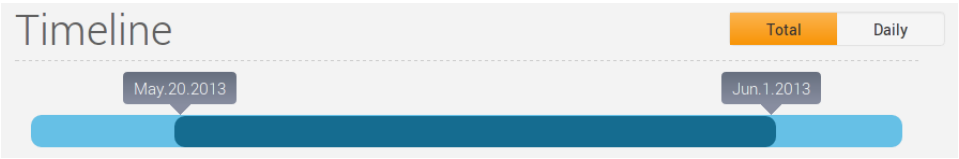

Figure 4.4: Timeline Selection Slider User Interface

The user also has the option of visualizing each individual day over that range (Daily), or an aggregate summarization (Total) that is explained in the subsequent sections. Once a range has been selected the site generates requests for the three summary streams from the MobiSense server. These are simple HTTP GET requests and three are sent, one to each summarization handler on the server. For example, a request URL for the outdoor or GPS summary from May 20 to June 1 would look like the following:

```
http://mobisense.ca/gps/?start=13-05-20&end=13-06-01
```

The server will populate summary data and send it back in JSON format. The JSON format closely follows the structure of the summary files presented in Section 4.5

4.6.2 Outdoor Summary

The outdoor summary data is a stream of GPS fixes consisting of latitudes and longitudes. The Google Maps API [20] has built in functionality to visualize a heat map distribution of logged GPS locations. Frequently logged locations produce larger, darker heat centers. In addition, trace lines can be added to connect consecutive fixes. An example screen shot of the visualization is shown below in Figure 4.5:

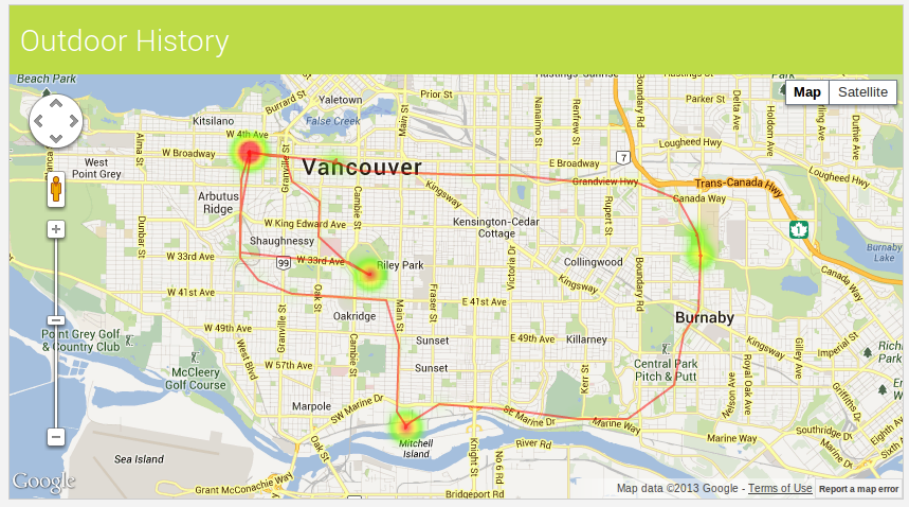

Figure 4.5: Outdoor Summary Visualization

The data visualized here was collected by a single user over the course of two days. Clear centers around the user's home and work place can be seen as well as two trips to parks in the city. For both individual day summaries and total aggregates over a date range, the nature of the visualization presentation is essentially the same. Using an embedded instance of Google Maps for the outdoor visualization affords the user all the features and familiar functions of Google Maps. For example users can zoom in and out and pan around the map, as well as investigate street level views of areas of interest. The total kilometres travelled for each day and the average number of kilometres per day are also shown to the user. When the user chooses to view each day individually across a timeline, a separate visualization page is generated for each day. When the total summarization of a timeline is requested, a single map is generated for the entire period.

4.6.3 Activity and Indoor Summary

The visualizations for the activity and indoor timelines are closely related in that they are summarizing classifications over the course of the day. Pie charts visualizing a break down of the distribution of time spent in each activity class for the activity summarization and each room for the indoor summarization are created for the user. Figure 4.6 shows an example of each chart for a given timeline.

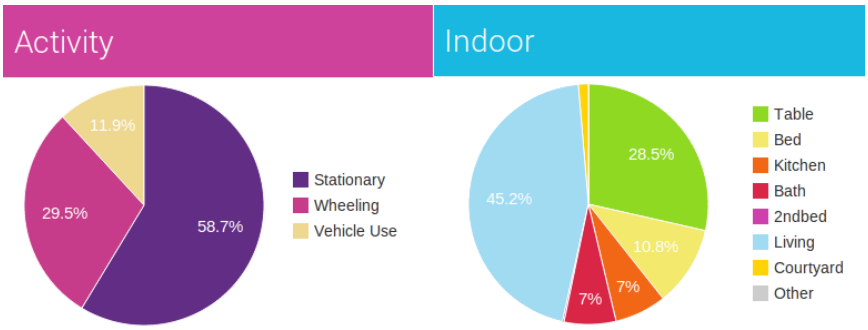

Figure 4.6: Pie Charts Summarizing Distribution of Activities and Rooms

We can see that for this period, over 58% of the user's time was spent not moving, about 30% of the time wheeling (since this was a wheelchair user) and 11% driving or using public transportation. Similarly, from the indoor chart, one can see the user spent most of his time indoors in the living room or desk table. This pie chart visualization of activities was employed in the prototype system proposed in [37].

Users can also mouse over the sections of the pie chart and see absolute minutes logged for each activity class or room. Here, as for the outdoor summary, the visualization stays the same whether the user requests individual day summaries or the total aggregation over the entire timeline. The pie chart works well for a quick summary of the user's time; however, interesting temporal trends are also likely of interest to users for behaviour analysis. The following visualizations are designed to show specifically when classifications occur during the day.

As the summary of the data streams is stored as a sequence of classifications, it made sense to visualize this sequence explicitly in addition to the aggregate pie charts. Each classification interval is visualized directly in a color-coded timeline. The approach is also employed in the work of [37]. Figure 4.7 presents an example summary for May 19, 2013.

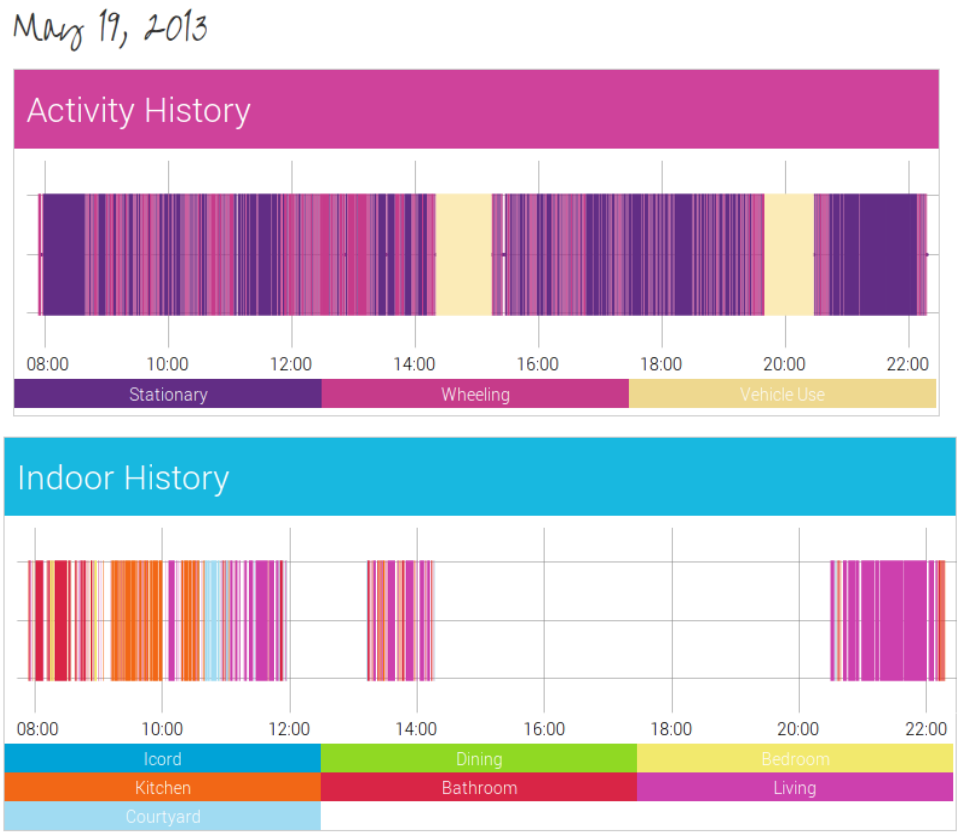

Figure 4.7: Screenshot of Activity and Indoor Daily Timeline Summary

There are only three classes of interest for the activity summary, while the number of rooms for the indoor summary depends on the training model. The JavaScript library DyGraph was used to create the graphs and allows for zooming in or out. In this example we can see the descriptive power of this style of visualization. The user begins the day in the bathroom and bedroom for about 50 minutes, then spends some time in the kitchen and living room and then leaves the house at around 12 noon. The user returns to the house at 1 and then leaves by vehicle at 2 and returns around 9.

The previous visualizations were used to plot single days, however if the user requests a total aggregation over the dates of interest, a different summary is presented. Instead of sending back separate sequences for each day, the MobiSense server aggregates over each day in the range and sends a single sequence that represents the entire date range. The timeline is populated with predetermined intervals of 10 minutes over the course of 24 hours and each interval stores a probability distribution of the classes for that time over the date range. For example, if the user selects a range of 10 days and for 3 out of 10 of those days the user was classified as being in their bathroom from 8:10 to 8:20, then for that interval the bathroom would be visualized with a probability of 0.3.

Figure 4.8: Screenshot of Total Summary Timeline Visualization

The total 24 hour timeline is visualized as a 100% stacked line chart, where each interval is a distribution of classifications that add up to 1. An example summary using this visualization technique is shown in Figure 4.8

Here again we can see the descriptive power of this kind of visualization in this example. The activity summary shows us that there is a tendency for the user to be using vehicle transportation in the morning between 9 and 10, likely representing the morning commute. There is a smaller corresponding peak for the return trip home between 5 and 6. After about 11 at night, the probability of stationary activity dominates and represents the user going to sleep or at least leaving their phone somewhere for the night. For the rest of the day, wheeling and stationary are mostly distributed equally.

Analyzing the indoor plot we can see similar trends. With more rooms in the distribution the plot becomes somewhat harder to read, but one can still see that there is a definite trend towards being in the living room in the evening and similarly the bedroom at night. Since every interval across the days of interest will not necessarily have a classification (novel locations) we also included a subplot below the summary that visualizes data density for each interval. This subplot shows for the dates of interest, how many have non-novel WiFi readings during each time interval. In this example, we see that the data density it quite low throughout the day, until the user returns home in the evening. The data density drops to 0 when the user has no non-novel readings during the interval or when the phone was not collecting data. In the example we can see that no data was collected from 6 to 8 in the morning over the range as the activity data is also absent there.

Currently the server populates the aggregate structure over the date range each time a summary is requested. With relatively small ranges, the time spent processing is negligible; however, if larger date ranges were considered, more thought would be required to cache or store the aggregate results in a query optimized format.

Chapter 5

Conclusion

Evaluating indoor localization on real and simulated datasets has grounded our investigation and given us more confidence that the approaches would perform well in a variety of residences and non-ideal environments. Random forests were demonstrated to be highly effective as an approach to room level localization using WiFi signal strengths. They had the best overall performance on both the normal datasets and those with simulated defects. For novelty detection, the best approach was less clear; however, the nearest centroid approaches seemed to be more robust than the others. Of the evaluated approaches, nearest centroid with Euclidean distance thresholds appeared to be most robust across the environments evaluated.

The activity classifier was able to differentiate between wheeling, walking, stationary and vehicle motion to a high degree of accuracy using the test data collected. However, in practice, it was possible to confuse walking with wheeling. To further validate our classifier, we compared our results with a summary produced by another wheelchair activity measurement system. In this system an accelerometer was mounted directly to one wheel and we saw that our approach gave comparable actigraphy summaries.

We proposed MobiSense as a research platform for lifespace summaries using mobile phones. The combination of a simple timeline selection with rich visualizations could act as a powerful explorative tool for researchers and users interested in their own patterns and behaviours. Date range aggregates and daily summaries for activity and indoor location were visualized as 100% stacked line charts and line charts respectively. Heat maps served as visualizations for outdoor activity and all visualizations were effective in quickly understanding lifespace trends and patterns. The timeline selection control provides a clear, simple point of entry for manipulating and exploring date ranges. The server-side data storage and processing is organized in a straight forward way to reduce technical overhead and complexity, while following the spirit of the architecture abstractions of the openMHealth specifications. Extensions and improvements to the system would be relatively straight forward to implement and experiment with. A test instance of the system is currently hosted at `http://mobisense.ca`

5.1 Future Work

When proposing an entire system, many avenues for future improvements and investigation present themselves. In particular, more effort could be spent in the development of a truly robust classifier between walking and wheeling activity classes. As explained in Section 3.4.2, in practice the signals are confused for each other somewhat frequently.

While the design of MobiSense was influenced a great deal by the architecture patterns presented by the openMHealth initiative, full adherence to the openMHealth abstraction specification would be preferable to promote standards and collaboration. A very serious aspect of any openMHealth system is the concern about system security and user privacy. Mobile health systems may potentially be storing sensitive and private user information and so great care must be taken to ensure that data is not compromised. Currently MobiSense does not have any security measures in place.

As a tool for exploring and summarizing lifespace, a primary goal of MobiSense is for researchers and users to identify trends and common behaviours in their daily lives. While passive summarization of information is an important step towards making adjustments and behavioural improvements to lifestyle and activity, future systems may not only respond to user queries, but actively request information and make suggestions to users directly. Currently the visualizations are the only meaningful interaction users have with the MobiSense system. It is exciting to imagine future assistive capabilities of MobiSense as an interactive agent, instead of only as a passive system. Extending the predictive and interactive abilities of MobiSense could, for example, enable the creation of very useful personalized reminders and alerts to users. The development of long-term personal data offers an exciting context for future work in unsupervised machine learning and wearable systems. Ohmage [15] and HumanSense [13] can query the user, however these requests are simple prompts for data or labeling.

The introduction of Google Glass presents another interesting future platform for data collection and visualization. With video recording and a small heads up display, interactions and data logging could potentially become much richer. We are beginning to see devices and systems that are more personalized and responsive than ever before. It is an exciting opportunity to apply some of these ideas to assistive technology and possibly improve quality of life and overall health.

Bibliography

[1] Paramvir Bahl and Venkata N Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 775–784. IEEE, 2000.

[2] B. Balaguer, G. Erinc, and S. Carpin. Combining classification and regression for WiFi localization of heterogeneous robot teams in unknown environments. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 3496–3503, 2012.

[3] Ling Bao and Stephen S. Intille. Activity recognition from user-annotated acceleration data. In Alois Ferscha and Friedemann Mattern, editors, *Pervasive Computing*, volume 3001 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin Heidelberg, 2004.

[4] P. Boissy, S. Briere, M. Hamel, M. Jog, M. Speechley, A. Karelis, J. Frank, C. Vincent, R. Edwards, and C. Duval. Wireless inertial measurement unit with GPS (WIMU-GPS); Wearable monitoring platform for ecological assessment of lifespace and mobility in aging and disease. In *Engineering in Medicine and Biology Society,EMBC, 2011 Annual International Conference of the IEEE*, pages 5815–5819, 2011.

[5] Liming Chen, J. Hoey, C.D. Nugent, D.J. Cook, and Zhiwen Yu. Sensor-based activity recognition. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 42(6):790–808, 2012.

[6] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14:462–467, 1968.

[7] Sunny Consolvo, Predrag Klasnja, David W. McDonald, Daniel Avrahami, Jon Froehlich, Louis LeGrand, Ryan Libby, Keith Mosher, and James A. Landay. Flowers or a robot army?: encouraging awareness

& activity with personal, mobile displays. In *Proceedings of the 10th international conference on Ubiquitous computing*, UbiComp '08, pages 54–63, New York, NY, USA, 2008. ACM.

[8] Mark Cummins and Paul Newman. FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance. *The International Journal of Robotics Research*, 27(6):647–665, 2008.

[9] Ian Dewancker, Jaimie Borisoff, Boyan Tom Jin, and Ian M. Mitchell. MobiSense: Lifespace racking and activity monitoring on mobile phones. In *Proceedings of Rehabilitation Engineering and Assistive Technology Society of North America Annual Conference, RESNA 2014*, 2014.

[10] Brian Ferris, Dieter Fox, and Neil D Lawrence. WiFi-SLAM Using Gaussian Process Latent Variable Models. In *International Joint Conference on Artificial Intelligence*, volume 7, pages 2480–2485, 2007.

[11] Brian Ferris, Dirk Hähnel, and Dieter Fox. Gaussian processes for signal strength-based location estimation. In *In Proc. of Robotics Science and Systems*. Citeseer, 2006.

[12] J. Frank, S. Mannor, J. Pineau, and D. Precup. Time series analysis using geometric template matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012.

[13] Jordan Frank. Humansense android app. `http://jwf.github.io/Humansense-Android-App/`, August 2013.

[14] Jordan Frank, Shie Mannor, and Doina Precup. Generating storylines from sensor data. *Pervasive and Mobile Computing*, 2013.

[15] John Hicks, Nithya Ramanathan, Hossein Falaki, Brent Longstaff, Kannan Parameswaran, Mohamad Monibi, Donnie H Kim, Joshua Selsky, John Jenkins, Hongsuda Tangmunarunkit, et al. Ohmage: An open mobile system for activity and experience sampling. Technical report, CENS Technical Reports, 2011.

[16] Joseph Huang, David Millman, Morgan Quigley, David Stavens, Sebastian Thrun, and Alok Aggarwal. Efficient, generalized indoor WiFi GraphSLAM. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1038–1043. IEEE, 2011.

[17] Ekahau Inc. Ekahau : Business intelligence through location. `http://www.ekahau.com/`, 2013.

[18] Fitbit Inc. Fitbit. `http://www.fitbit.com/us`, July 2013.

[19] Google Inc. Google Location APIs. `http://developer.android.com/google/play-services/location.html`, July 2013.

[20] Google Inc. Google Maps API. `https://developers.google.com/maps/`, August 2013.

[21] Google Inc. Google Now. `http://www.google.com/landing/now/`, July 2013.

[22] Google Inc. Motion Sensors. `http://developer.android.com/guide/topics/sensors/sensors_motion.html`, July 2013.

[23] Nike Inc. Nike+ Fuelband. `http://www.nike.com/us/en_us/c/nikeplus-fuelband`, July 2013.

[24] John Krumm. *Ubiquitous Computing Fundamentals*. Chapman & Hall/CRC, 1st edition, 2009.

[25] Olivier Ledoit and Michael Wolf. A well-conditioned estimator for large-dimensional covariance matrices. *Journal of multivariate analysis*, 88(2):365–411, 2004.

[26] Lin Liao, Dieter Fox, and Henry Kautz. Extracting Places and Activities from GPS Traces Using Hierarchical Conditional Random Fields. *Int. J. Rob. Res.*, 26(1):119–134, January 2007.

[27] Open mHealth. Open mHealth. `http://openmhealth.org/`, August 2013.

[28] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective (Adaptive Computation and Machine Learning series)*. The MIT Press, August 2012.

[29] Numpy and Scipy Contributors. Discrete Fourier Transform. `http://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fftfreq.html`, July 2013.

[30] Ohmage. Ohmage projects. `http://ohmage.org/projects.html`, August 2013.

[31] Jun-geun Park, Ben Charrow, Dorothy Curtis, Jonathan Battat, Einat Minkov, Jamey Hicks, Seth Teller, and Jonathan Ledlie. Growing an organic indoor location system. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 271–284. ACM, 2010.

[32] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine Learning in Python. *The Computing Research Repository*, abs/1201.0490, 2012.

[33] Alex Pentland, Nadav Aharony, Wei Pan, Cody Sumter, and Alan Gardner. Funf: Open sensing framework. `http://www.media.mit.edu/research/groups/1448/funf-open-sensing-framework`, August 2013.

[34] Andrzej Pronobis, Oscar M. Mozos, Barbara Caputo, and Patric Jensfelt. Multi-modal semantic place classification. *The International Journal of Robotics Research (IJRR), Special Issue on Robotic Vision*, 29(2-3):298–320, February 2010.

[35] Morgan Quigley, David Stavens, Adam Coates, and Sebastian Thrun. Sub-meter indoor localization in unmodified environments with inexpensive sensors. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2039–2046. IEEE, 2010.

[36] Stephen J Roberts. Extreme value statistics for novelty detection in biomedical data processing. In *Science, Measurement and Technology, IEE Proceedings-*, volume 147, pages 363–367. IET, 2000.

[37] Ana Katrin Schenk, Bradley C Witbrodt, Carrie A Hoarty, Richard H Carlson, Evan H Goulding, Jane F Potter, and Stephen J Bonasera. Cellular telephones measure activity and lifespace in community-dwelling adults: proof of principle. *J Am Geriatr Soc*, 59(2):345–52, 2011.

[38] Oliver S. Schneider, Karon E. MacLean, Kerem Altun, Idin Karuei, and Michael M.A. Wu. Real-time gait classification for persuasive smartphone apps: structuring the literature and pushing the limits. In *Proceedings of the 2013 International Conference on Intelligent User Interfaces*, IUI '13, pages 161–172, New York, NY, USA, 2013. ACM.

[39] Quantified Self. Quantified self : Self knowledge through numbers. `http://quantifiedself.com/about/`, August 2013.

[40] Sharon Eve Sonenblum, Stephen Sprigle, and Ricardo A Lopez. Manual wheelchair use: bouts of mobility in everyday life. *Rehabil Res Pract*, 2012.

[41] Leo Breiman Statistics and Leo Breiman. Random forests. In *Machine Learning*, pages 5–32, 2001.

[42] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[43] Douglas L. Vail, Manuela M. Veloso, and John D. Lafferty. Conditional random fields for activity recognition. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, AAMAS '07, pages 235:1–235:8, New York, NY, USA, 2007. ACM.

[44] Wikipedia. Haversine formula. `http://en.wikipedia.org/wiki/Haversine_formula`, 2013. [Online; accessed 19-July-2013].

[45] Chenshu Wu, Zheng Yang, Yunhao Liu, and Wei Xi. Will: Wireless indoor localization without site survey. In *INFOCOM, 2012 Proceedings IEEE*, pages 64–72. IEEE, 2012.