Proximity Detection Using Wi-Fi Fingerprints and Smartphone Magnetometers, With Applications to COVID-19 Surveillance

by Zach Van Hyfte

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

[illegible]

Professor Avideh Zakhor
Research Advisor

5/22/2022

(Date)

* * * * * * *

Professor Prabal Dutta
Second Reader

(Date)

Proximity Detection Using Wi-Fi Fingerprints and Smartphone Magnetometers, With Applications to COVID-19 Surveillance

Zach Van Hyfte

May 20th, 2022

Abstract

Smartphone apps for exposure notification and contact tracing have been shown to be effective in controlling the COVID-19 pandemic. However, Bluetooth Low Energy tokens similar to those broadcast by existing apps can still be picked up far away from the transmitting device, making them ineffective for COVID-19–related proximity detection applications in some scenarios. In this thesis, we present two new classes of methods for detecting whether or not two devices are in immediate physical proximity, i.e. 2 or fewer meters apart, as established by the U.S. Centers for Disease Control and Prevention (CDC). One method uses Wi-Fi RSSI fingerprints, and the other uses magnetometer traces. Both of these types of data can be recorded by almost all modern smartphones. Our ultimate goal is to enhance the accuracy of smartphone-based exposure notification and contact tracing systems.

We first design a set of binary machine learning classifiers that take as input pairs of Wi-Fi RSSI fingerprints. These classifiers distinguish between pairs of RSSI fingerprints recorded 2 or fewer meters apart and pairs recorded further apart but still in Bluetooth range. We empirically verify that a single classifier cannot generalize well to a range of different environments with vastly different numbers of detectable Wi-Fi Access Points (APs). However, specialized classifiers, tailored to situations where the number of detectable APs falls within a prescribed range, are able to detect physical proximity significantly more accurately. As such, we design three classifiers for situations with low, medium, and high numbers of detectable APs. We characterize their balanced accuracy for proximity detection to be between 66.8% and 77.8%.

Next, we design a second set of binary machine learning classifiers, which take as input pairs of 10-second traces of smartphone magnetometer readings. These classifiers distinguish between pairs of trace segments for which the two recording devices are 2 or fewer meters apart for at least 75% of the segment duration and pairs for which the two devices are further apart but still in Bluetooth range. We first evaluate these classifiers' performance on traces from the MagPIE dataset, a dataset for evaluating magnetometer-based localization algorithms; we characterize their balanced accuracy for homogeneous-device proximity detection to be between 89.3% and 93.3%. We show that our classifiers can generalize well to different buildings whose traces are not present in their training data. We introduce a simple method of compensating for different magnetometer biases in heterogeneous devices, and evaluate our approach with this added mitigation by training and evaluating classifiers on different disjoint subsets of traces from 4 different smartphone models. We characterize their balanced accuracy for heterogeneous-device proximity detection with non–tilt-compensated traces to be between 93.8% and 96.9%; these results indicate that our classifiers can generalize well to devices whose traces are not present in their training data.

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I'd like to thank my research advisor, Professor Avideh Zakhor, whose thoughtful guidance, expertise, and dedication to her students made this work possible, and whose mentorship has helped me grow immensely as a student and researcher.

I'd also like to thank my family, whose constant support is what has gotten me through five years at Berkeley and what keeps me moving forward each day.

I'm grateful for the work of all my colleagues at UC Berkeley who've contributed to this project. Many thanks to Oleksii Volkovskyi and Jerome Quenum for collecting Wi-Fi fingerprint data in Cory Hall; to Anderson Hansen for helping determine the design of these proximity detection systems; to Richard Huang for implementing the RE3 algorithm from [10]; to Willis Wang for evaluating the suitability of several RSSI fingerprint datasets for this project; and to countless other students whose ingenuity and creativity has inspired me, and from whom I've learned so much.

Computational resources for this work, on the Microsoft Azure platform, were generously provided by Microsoft as an AI for Health grant.

Chapter 1

Introduction

The COVID-19 pandemic sparked a proliferation of “exposure notification” or “contact tracing” smartphone apps designed to alert users if they came within close proximity of an individual infected with COVID-19 [28, 7, 5, 27]. Many were backed by public health authorities, and were used to help automate the traditional manual contact tracing process. Most of these apps continuously track, in an oblique and privacy-preserving manner, which other devices a specific smartphone has been near. When an individual tests positive for COVID-19, these apps can send a notification to every device to which that individual’s smartphone had recently been in close proximity. Those contacts can self-quarantine until they confirm that they are not carrying the virus, thus limiting the disease’s spread.

Authors in [28], studying the effects of the NHS COVID-19 app for England and Wales, estimated that “for every percentage point increase in app users, the number of cases can be reduced by [between] 0.8 ... [and] 2.3 per cent ... These findings provide evidence for continued development and deployment of such apps in populations that are awaiting full protection from vaccines.” COVID-19 has been shown to be much more transmissible indoors than outdoors [26, 19], and most people spend roughly 80% of their waking hours in indoor environments. Since many public indoor environments contain a substantial number of Wi-Fi Access Points, Wi-Fi data has the potential to become a key method for exposure notification and contact tracing apps to accurately measure proximity indoors. Likewise, steel is used in the construction of many of the larger modern commercial and residential buildings in which people spend large parts of their day; the fine-grained magnetic field disturbances caused by both these steel components and other metallic objects or furniture are another source of largely untapped data suitable for accurate proximity measurement.

Currently, the majority of existing exposure notification and contact tracing apps use Bluetooth Low Energy (*BLE*) to determine whether two users are in close proximity. Specifically, a device running an exposure notification app periodically broadcasts, via BLE, a token that can be picked up by other nearby devices running the same app. The distance from which a transmitted BLE token similar to those broadcast by the apps can be received varies based on the radio configuration and the characteristics of the surrounding environment, but can be over 20 meters. Given the 2-meter social distancing guidelines set by the

U.S. Centers for Disease Control and Prevention (CDC), BLE-based proximity detection methods might not be the most appropriate for exposure notification and contact tracing applications across all environments.

This thesis presents two new classes of methods for detecting whether or not two devices are in immediate physical proximity, i.e. 2 or fewer meters apart, as established by the CDC. These guidelines are based on research indicating a high probability of COVID-19 transmission between two people who are within 2 meters of each other for at least 15 minutes. Both of our proposed methods involve binary machine learning classifiers that take as input data collected by smartphone sensors about the surrounding environment and infrastructure; the classifiers use this data to predict whether or not two devices are within roughly 2 meters of each other.

Our first proposed method uses Wi-Fi RSSI fingerprints to determine whether or not two devices are within immediate physical proximity. Each classifier takes as input two Wi-Fi RSSI fingerprints — two sets of MAC addresses of *APs* and their respective received signal strengths — and predicts whether or not the two fingerprints were captured within roughly 2 meters of each other. Our second proposed method uses magnetometer traces to accomplish a version of this same task. Each classifier takes as input two 10-second segments of smartphone magnetometer traces and predicts whether or not the two recording devices were within roughly 2 meters of each other for at least 75% of the segment duration. Both of our methods can work across a range of device types, and are designed to be robust in dealing with a wide range of physical environments as well as heterogeneous devices. The Wi-Fi–based method can be deployed across a wider range usage scenarios and device types, including notebook computers, while the magnetometer-based method has a significantly lower impact on device battery life than the Wi-Fi–based method.

Our proposed methods could supplant or augment existing Bluetooth-based proximity detection methods employed by exposure notification apps. They could improve the accuracy of smartphone-based contact tracing by more precisely determining which individuals were actually in very close physical proximity to an infected individual.

The outline of this thesis is as follows: In Chapter 2, we discuss the design of our Wi-Fi–based machine learning classifiers and characterize their performance. In Chapter 3, we discuss the design of our magnetometer trace–based machine learning classifiers, and present preliminary performance results from evaluations involving a range of different buildings and device models. In Chapter 4, we conclude the thesis and discuss future directions for the work described within it.

Chapter 2

Proximity Detection Using Wi-Fi RSSI Fingerprints

In this chapter, we design a set of binary machine learning classifiers that use Wi-Fi RSSI fingerprints to determine whether or not two devices are within immediate physical proximity, and characterize their performance for real-world COVID-19 proximity detection tasks with heterogeneous devices.

The outline of this chapter is as follows: In Section 2.1, we discuss how our methods differ from previous work. In Section 2.2, we outline where the RSSI fingerprint data used in our experiments was sourced and how our new sets of RSSI fingerprint data were collected. In Section 2.3, we describe how our training and evaluation sets were created from the RSSI fingerprint data. In Section 2.4, we discuss each of the fingerprint similarity measurements that are passed to our classifiers as input features. In Section 2.5, we detail the additions and adjustments we made to the input features to ensure that our classifiers work across a heterogeneous array of devices. In Section 2.6, we describe the algorithms, software, and settings used for training the classifiers. In Section 2.7, we present empirical evidence indicating that a single machine learning classifier cannot generalize well to a range of different environments with vastly different numbers of detectable APs. In Section 2.8, we present a set of three binary machine learning classifiers, each tailored for a situation where the number of APs detected by the device falls into a given range, and quantify their performance.

2.1 Related Work

The development of systems for localizing Wi-Fi–enabled devices in indoor environments using a Wi-Fi RSSI fingerprint — a list of the Wi-Fi Access Points (*APs*) detected by a particular device, and the received signal strength indicator (*RSSI*) value for each of those APs — has been an active research area over the past 15 years [15]. Existing Wi-Fi localization systems can determine a device's position within a building with an average error of several meters. These systems typically require the creation of a database containing at least one RSSI fingerprint from every single location or position within the building. Wi-Fi–enabled devices within the building can submit a fingerprint to a central server, which employs an algorithm such as k-nearest neighbors to find the location in the database whose fingerprint is most similar to the one submitted by the device. The classifiers we present in this thesis are concerned with determining whether or not two users are in close physical proximity, rather than estimating a user's location on a map, but they make use of established techniques designed to increase the accuracy of RSSI fingerprint–based localization systems. Crucially, too, the proximity detection methods explored in this thesis do not require the *a priori* collection of a database of fingerprints from every possible reference point — they can be utilized in a new location without having to perform any training or setup specific to that location.

Similar to this chapter, [23] presents a number of binary machine learning classifiers that use various features extracted from a pair of Wi-Fi fingerprints to determine whether or not they were recorded in "close physical proximity" to each other. However, the sole criterion used in [23] to determine whether two fingerprints were classified as within close physical proximity was whether the two smartphones recording the fingerprints were both able to detect each other via Bluetooth around the time the two fingerprints were recorded; this, it was noted, generally happens when the two phones are 10 or fewer meters away from each other. In contrast, because the data used in our training and evaluation sets includes precise, meter-level ground truth position information, we assign a label of "Close" only to samples recorded within 2.25 meters of each other — the goal of our classifiers is to determine whether two devices are not just nearby but in *immediate* physical proximity to each other, i.e. less than the safe social distance defined by the U.S. Centers for Disease Control and Prevention (CDC). In addition, our systems expand upon the set of similarity measurements evaluated in [23], making use of additional features and pre-processing steps to further enhance accuracy.

2.2 Data Sources

A variety of Wi-Fi fingerprint datasets used to develop and evaluate Wi-Fi localization systems are publicly available online, as shown in Table 2.1; we used Wi-Fi fingerprints from several of these datasets to create training and evaluation data for our classifiers.

To the extent possible, we also collected our own training and evaluation data, optimized for training proximity detection classifiers. We developed an Android app for data collection, which uses Android's `WifiManager` API to perform scans of nearby APs on-demand and save the fingerprints to the device for later use.

We were able to collect our own RSSI fingerprint data in Cory Hall, a five-story academic building on the UC Berkeley campus that houses classrooms, labs, common work spaces, and offices for the EECS department. We collected fingerprints from the first, third, and fourth floors of the building over the course of three days in late November and early December 2020. Each device captured data in "bursts," recording 9 fingerprints at a time in rapid succession.

Table 2.1: Wi-Fi RSSI Fingerprint Datasets

Name	**Median APs**[a]	**Source**
Miskolc	10	UCI ML Repository [16]
JUIndoorLoc[b]	15	JUIndoorLoc Paper [20]
UJIndoorLoc	17	UJI IndoorLoc Platform [22]
IPIN 2016 Tutorial	32	UJI IndoorLoc Platform [22]
TampereU	38	UJI IndoorLoc Platform [22]
Alcalá Tutorial	40	UJI IndoorLoc Platform [22]
Suburban Home	8	Collected by Us
Cory Hall	66	Collected by Us

[a] The median number of APs detected in a given fingerprint from the dataset.

[b] Only the test data from this dataset was used.

Table 2.2: Devices Used to Collect Wi-Fi RSSI Fingerprints

Device	**Android Version**	**Cory Hall**			**Suburban Home**
		Floor 1	**Floor 3**	**Floor 4**	
Google Pixel 3	10	225	576	198	567
Nokia 2.2[a]	10	225	576	198	567
Google Pixel XL	10	225	342	198	—
Oppo RX17 Pro	8.1	63	27	9	—
	Total	738	$1,521$	603	$1,134$

[a] Model TA-1179, in the 3 GB RAM configuration

The devices were kept stationary for the duration of each burst. At each position where a set of fingerprints were recorded, a Leica DISTO E7100i laser distance measurement device was used to measure the distance from each point to two reference walls on the current floor, which were later used to calculate the distance between samples from the same floor. The median number of APs observed in a given fingerprint from the first, third, and fourth floors are 76, 69, and 52, respectively. The median number of APs observed in a given fingerprint from the entire set of data collected from Cory Hall is 66.

Data was also collected from a single-story suburban home in San Diego, California, over an area spanning roughly 1,800 square feet, also using our Android data collection app. As in Cory Hall, data was captured in 9-fingerprint "bursts," with devices kept stationary during

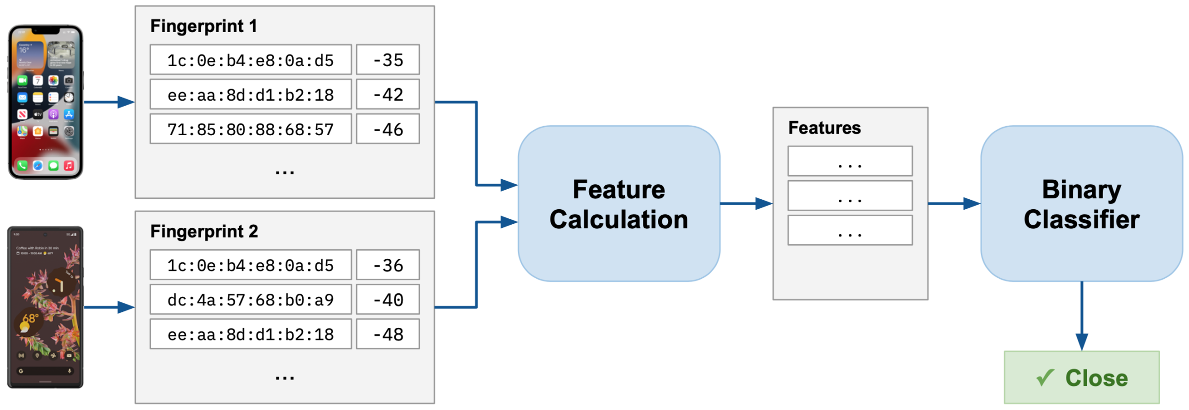


Figure 2.1: The high-level design of our Wi-Fi–based proximity detection system.

each burst. Within the home itself are two APs, one located roughly in the northwest corner of the floorplan, and the other roughly in the southeast corner of the floorplan. However, at any given location within the home, APs from several nearby homes are also detectable; across all of the samples collected, the median number of APs observed in a given fingerprint is 8.

Table 2.2 shows the devices we used to collect fingerprint data, the versions of Android they were running at the time of data collection, and the number of fingerprints collected with each device in each location.

No filtering of the APs within the fingerprints — e.g. filtering out APs with very weak signal strengths, or filtering out smartphones in Wi-Fi "hotspot" mode — was performed.

2.3 Classifier Design and Proximity Classes

Our classifiers evaluate two RSSI fingerprints at a time, determining whether or not they were recorded in immediate physical proximity to each other. Each classifier takes as input two Wi-Fi RSSI fingerprints — two sets of MAC addresses of *APs* and their respective received signal strengths — and predicts whether or not the two fingerprints were captured within roughly 2 meters of each other. Figure 2.1 shows the high-level design of our Wi-Fi–based proximity detection system.

To prepare training and evaluation sets from a given pool of fingerprints, we isolate fingerprints into different subsets — one for each floor of each building of each dataset. Within each subset, we enumerate every possible pairing of two distinct fingerprints from the subset, and calculate the two-dimensional distance d between the locations where the two fingerprints were recorded, using the meter-level coordinates that are either provided in the dataset or recorded by our team. We assign each pairing of fingerprints a "proximity class" according to the value of d for that pairing. If 0 meters $\leq d \leq$ 2.25 meters, the fingerprint pairing's proximity class is set to "Close." If 3.25 meters $\leq d \leq$ 20 meters, it is set to "Far."

Fingerprint pairings are dropped from the training and evaluation sets if the two fingerprints were recorded more than 20 meters apart, in an effort to focus the training process on the fine-grained differentiation between pairings recorded in immediate physical proximity and those recorded in somewhat close physical proximity, i.e. within Bluetooth range of each other. This is in contrast to an approach which would optimize our classifiers for the coarse-grained differentiation of pairings recorded in immediate physical proximity and those recorded very far away from each other. Fingerprint pairings are also dropped from the training and evaluation sets if the two fingerprints were recorded between 2.25 and 3.25 meters apart, in an effort to focus the training process on differentiating between pairings recorded in immediate physical proximity and those recorded in somewhat close physical proximity, rather than on border cases that are nearly at the distance cutoff for the "Close" label.

2.4 Classifier Input Features

For each fingerprint pairing, we calculate an expansive set of features, which are the inputs passed directly to the classifiers described in Sections 2.7 and 2.8. The majority of these input features were selected because we hypothesized that they could potentially serve as similarity measurements — quantities that numerically express the level of similarity between the two fingerprints in a pairing, which can be a rough proxy for how close together the fingerprints were recorded. In this section, we describe our chosen features in detail.

Consider a pair of RSSI fingerprints, F_X and F_Y. Define the set of *shared APs* $\{S_1, S_2, \ldots, S_N\}$ as the APs that are detected in both F_X and F_Y. Furthermore, define $\text{RSSI}(a, F_X)$ and $\text{RSSI}(a, F_Y)$ as the RSSI of the AP a in fingerprints F_X and F_Y, respectively.

2.4.1 AP Detection–Based Features

The classifiers are provided with the following input features derived from the number of APs detected in F_X, the number of APs detected in F_Y, and the number of shared APs:

1. The *shared AP count*: N, the number of shared APs from above.
2. The *union AP count*: the total number of APs detected in at least one of the fingerprints.
3. The *non-shared AP count*: the total number of APs detected in exactly one of the two fingerprints, but not both.

4. The *detected AP count difference*: the absolute value of the difference between the number of the APs detected in F_X and the number of APs detected in F_Y.

5. The *Jaccard similarity* of the sets of APs detected in the two fingerprints: the shared AP count divided by the union AP count.

2.4.2 Basic RSSI Value–Based Features

The *Manhattan distance* and the *Euclidean distance* between two fingerprints F_X and F_Y, calculated exactly as in [23], are both provided to the classifiers as input features.

Define the *top AP(s)* of a fingerprint as the AP(s) whose measured RSSI value(s) in that fingerprint are the highest among all APs detected in that fingerprint. We define a number of feature types related to the difference in dBm between the shared APs' measured RSSI values in F_X and in F_Y:

1. The feature *Has shared top AP within Z dBm* for fingerprints F_X and F_Y is equal to 1 if there exists at least one shared AP S_i such that (a) the measured RSSI value of S_i in F_X is at most Z dBm below the maximum RSSI of any AP in F_X, and (b) the measured RSSI value of S_i in F_Y is at most Z dBm below the maximum RSSI of any AP in F_Y. Otherwise, the feature is equal to 0. Features of the type *Has shared top AP within Z dBm* are provided to the classifiers for $Z = 1, 2, \ldots, 15$. This feature type is similar to the "top AP $\pm$ 6 dB" feature described in [23]. It is designed to allow a classifier to determine whether there is an AP that both fingerprints were likely closer to than most other APs, while accounting for differences in RSSI measurement scales or minor RSSI fluctuations that could change which specific AP has the highest RSSI.

2. The feature *RSSIs within Z dBm percentage* for fingerprints F_X and F_Y is equal to the percentage of shared APs whose RSSI values in F_X are within Z dBm of their RSSI values in F_Y. Features of the type *RSSIs within Z dBm percentage* are provided to the classifiers for $Z = 1, 2, \ldots, 15$.

3. The feature *Has shared top K APs* for fingerprints F_X and F_Y is equal to 1 if and only if the K highest-RSSI APs in F_X are the same as the K highest-RSSI APs in F_Y, regardless of ordering differences between the two fingerprints. Features of the type *Has shared top K APs* are provided to the classifiers for $K = 1, 2, \ldots, 8$.

2.4.3 Redpin Score–Based Features

Two input features based on the Redpin score are also provided to the classifiers. The Redpin score is a measurement of fingerprint similarity that was developed for and used in the Redpin crowdsourced Wi-Fi localization system [2], with a reference implementation available at [3]. The Redpin score calculation is not a commutative operation, so we provide two Redpin scores to the classifiers: `RedpinScore`$(\max(F_X, F_Y), \min(F_X, F_Y))$ and

`RedpinScore`$(\min(F_X, F_Y), \max(F_X, F_Y))$, where min and max select the fingerprint with the lower and higher number of detected APs, respectively.

2.4.4 Correlation-Based Features

In addition, we define the following pairs of vectors:

1. The *shared AP RSSI value vectors* are a pair of vectors containing the measured RSSI of each shared AP in F_X and in F_Y:
$$\Big[\,\mathrm{RSSI}(S_1, F_X),\ \mathrm{RSSI}(S_2, F_X),\ \ldots\ \mathrm{RSSI}(S_N, F_X)\,\Big]$$
$$\Big[\,\mathrm{RSSI}(S_1, F_X),\ \mathrm{RSSI}(S_2, F_X),\ \ldots\ \mathrm{RSSI}(S_N, F_X)\,\Big]$$

2. The *shared AP pair difference vectors* are a pair of vectors containing the absolute value of the difference between the RSSI values within each fingerprint for every possible pairing of two distinct shared APs:
$$\Big[\,|\,\mathrm{RSSI}(S_i, F_X)\ -\ \mathrm{RSSI}(S_j, F_X)\,|\ \ \ldots\ \ \forall(i \neq j) \leq N\,\Big]$$
$$\Big[\,|\,\mathrm{RSSI}(S_i, F_Y)\ -\ \mathrm{RSSI}(S_j, F_Y)\,|\ \ \ldots\ \ \forall(i \neq j) \leq N\,\Big]$$

3. The *shared AP pair ratio vectors* are a pair of vectors containing the ratios of the RSSI values within each fingerprint of every possible pairing of two distinct shared APs. They are similar to the modified type of RSSI fingerprints used in Hyperbolic Location Fingerprinting [13]:
$$\Big[\,\frac{\mathrm{RSSI}(S_i, F_X)}{\mathrm{RSSI}(S_j, F_X)}\ \ \ldots\ \ \forall(i, j) \leq N,\ i \neq j\,\Big]$$
$$\Big[\,\frac{\mathrm{RSSI}(S_i, F_Y)}{\mathrm{RSSI}(S_j, F_Y)}\ \ \ldots\ \ \forall(i, j) \leq N,\ i \neq j\,\Big]$$

4. Define the *rank* of a shared AP S_i within a particular fingerprint as the number of shared APs, including S_i itself, in the fingerprint whose measured RSSI values are at least as weak as S_i's. The *normalized ordered shared AP rank vectors* are a pair of unit vectors, $\hat{R_X}$ and $\hat{R_Y}$. They are the normalized forms of the following vectors:
$$\begin{aligned} R_X &= \big[\,N,\ N-1,\ N-2,\ \ldots\ \ 2,\ 1\,\big] \\ R_Y &= \big[\,\mathrm{Rank}_Y(N),\ \mathrm{Rank}_Y(N-1),\ \ldots\ \ \mathrm{Rank}_Y(1)\,\big] \end{aligned}$$

   $\mathrm{Rank}_Y(i)$, as used above, is the rank within fingerprint F_Y of the shared AP with rank i in fingerprint F_X. Thus, any index i represents some shared AP, and $R_X[i]$ and $R_Y[i]$ are the ranks of that shared AP in the two different fingerprints. The correlation coefficient of $\hat{R_X}$ and $\hat{R_Y}$ therefore measures how similar the signal strength rankings of the shared APs are across the two fingerprints.

For each of the pairs of vectors above — the shared AP RSSI vectors, the shared AP pair difference vectors, the shared AP pair ratio vectors, and the normalized shared AP rank vectors — the following measurements of similarity between the two vectors are passed as input features to the classifiers:

1. The cosine similarity of the two vectors.
2. The Pearson correlation coefficient of the two vectors.
3. The Spearman correlation coefficient of the two vectors.
4. The Kendall correlation coefficient of the two vectors.

2.4.5 Difference-Based Features

Lastly, we define a set of individual vectors as follows:

1. The *shared AP RSSI difference vector* is a vector containing the absolute value of the difference between the measured RSSI in F_X and in F_Y of each shared AP:

$$\Big[\, |\, \mathrm{RSSI}(S_i,\, F_X) \;-\; \mathrm{RSSI}(S_i,\, F_Y)\, | \;\; \ldots \;\; \forall\, i \leq N \Big]$$

2. Define $\mathrm{PD}(S_i,\ S_j,\ F_X)$ and $\mathrm{PD}(S_i,\ S_j,\ F_Y)$ as the shared AP pair difference, as defined in Section 2.4.4, of shared APs S_i and S_j in fingerprints F_X and F_Y respectively. The *shared AP pair difference comparison vector* contains the absolute value of the difference between the pair difference in F_X and the pair difference in F_Y of every possible pairing of two distinct shared APs:

$$\Big[\, \big|\mathrm{PD}(S_i,\ S_j,\ F_X) \;-\; \mathrm{PD}(S_i,\ S_j,\ F_Y)\,\big| \;\; \ldots \;\; \forall\, i \leq N \Big]$$

3. Similarly, define $\mathrm{PR}(S_i, S_j, F_X)$ and $\mathrm{PR}(S_i, S_j, F_Y)$ as the shared AP pair ratio, as defined in Section 2.4.4, of shared APs S_i and S_j in fingerprints F_X and F_Y respectively, and define the *shared AP pair ratio comparison vector* as follows:

$$\Big[\, \big|\mathrm{PR}(S_i,\ S_j,\ F_X) \;-\; \mathrm{PR}(S_i,\ S_j,\ F_Y)\,\big| \;\; \ldots \;\; \forall\, i \leq N \Big]$$

For each of the individual vectors above — the shared AP RSSI difference vector, the shared AP pair difference comparison vector, and the shared AP pair ratio comparison vector — the following similarity measurements are passed as input features to the classifiers:

1. The smallest element of the vector.
2. The largest element of the vector.

3. The mean of all of the vector's elements.
4. The median of all of the vector's elements.
5. The harmonic mean of all of the vector's elements.
6. The standard deviation of all of the vector's elements.
7. The population standard deviation of all of the vector's elements.

2.5 Mitigating the Effects of Device Heterogeneity

A common challenge in creating indoor positioning systems that use RSSI fingerprints is dealing with the effects of device heterogeneity. Since different devices and different Wi-Fi chips have different levels of sensitivity and different signal strength measurement scales, two RSSI fingerprints recorded in the same place with different devices can generally have (a) less of an overlap in the set of APs detected, and (b) a larger difference between their RSSI values for shared APs than a pair of RSSI fingerprints recorded with the same device model. We employ several measures to compensate for these effects of device heterogeneity, including inputting several additional features to the classifiers:

1. In both publicly available datasets and in our own collected data, among the metadata stored with each fingerprint is the model of the device that recorded it. Therefore, an additional feature passed into all of our classifiers is *Same device model*, whose value is 1 if the two fingerprints were recorded by the same device model and 0 otherwise. For instance, if the two fingerprints were recorded by two different Google Pixel 3 handsets, the value of this feature would be 1. Likewise, if the two fingerprints both came from a single Google Pixel 3 handset, the value of this feature would also be 1. This additional feature allows a classifier to select different decision sequences or feature weights and cutoff values for device-heterogeneous fingerprint pairings than for device-homogeneous ones.

2. An additional similarity measurement, designed to be more robust in the face of inputs from heterogeneous devices, is the Refined Relative RSSI Relationship (*RE3*), as proposed for use in Wi-Fi localization systems in [10]. Since it is calculated using the ranks of the APs detected in a fingerprint rather than their explicit RSSI values, it is more likely to remain the same across different devices that have different RSSI measurement scales. In addition to all of the classifier input features described above, the RE3 of the two fingerprints is calculated and passed as an input feature to the classifier.

3. It is established that in many cases, given two fingerprints recorded in the same location with different devices, the measured RSSI values in both fingerprints are roughly

linearly dependent. Applying a linear transformation to the RSSI values from one of the fingerprints can bring the two fingerprints in line and compensate for differences in the two devices' sensitivities and signal strength measurement scales. So, in actuality, for each of the 80 classifier input features described above that depends on the APs' RSSI values themselves, four separate input features are passed into the classifiers:

a) *No transformation*, with the feature value calculated with the raw, original RSSI values from F_X and F_Y.

b) *Single-fingerprint least squares*, with the feature value calculated after every RSSI value r_X in the fingerprint F_X has been replaced with $Ar_X + B$, with the values of the constants A and B determined by using the method of least squares to fit the RSSI values from F_X to the RSSI values from F_Y.

c) *Single-fingerprint 50% least squares*, with the feature value calculated after every RSSI value r_X in the fingerprint F_X has been replaced with $\frac{A}{2}r_X + \frac{B}{2}$, where A and B are computed as in (b) above.

d) *Double-fingerprint least squares*, with the feature value calculated after every RSSI value r_X in the fingerprint F_X has been replaced with $Ar_X + B$ and every RSSI value r_Y in the fingerprint F_Y has been replaced with $Cr_Y + D$, where A and B are the same as in (b) and (c) above, and C and D are similarly determined by using the method of least squares to fit the original, raw RSSI values from F_Y to the original, raw RSSI values from F_X.

2.6 Experiment Setup

We now present a series of binary machine learning classifiers whose aim is to predict, based on the extensive set of input features they are provided, the proximity class — "Close" or "Far" — of input fingerprint pairings.

The high number of input features provided to the classifiers has the potential to make our data subject to overfitting and the "curse of dimensionality," summarized in [21] as the phenomenon where "increasing the number of features fed into a machine learning model usually exponentially increases the search space and hence, the probability of fitting models that cannot be generalized." To mitigate this, while preserving the ability to take advantage of a large set of available features, we trained classifiers using the attribute bagging method, introduced in [4]. Attribute bagging is "a wrapper method that can be used with any learning algorithm," in which many different subsets of features, usually small ones, are randomly selected and used to train a set of smaller base estimators, usually decision trees. Base estimators with the highest possible performance are combined into an ensemble classifier. When a sample is input to an attribute bagging classifier for prediction, the base estimators each individually predict, or "vote," based on their particular set of input features, which class a sample belongs to. The class predicted by the most base estimators, after optionally factoring in weights assigned to each base estimator's vote, is chosen as the final predicted class. Scikit-learn [17], a popular Python machine learning library, provides a flexible `BaggingClassifier` class that can be configured to perform attribute bagging. All of the classifiers outlined below were trained with instances of the `BaggingClassifier` class from scikit-learn 0.22.1, with `DecisionTreeClassifier` instances as the base estimators. Through manual tuning across a number of experiments, we empirically found a set of hyperparameters that generally work well for this problem: at most 3 features per base estimator, and 300 individual voting base estimators — i.e. `max_features = 3` and `n_estimators = 300`.

To further reduce the dimensionality of the data — and thus reduce both the probability of overfitting and the time it takes to train a classifier, when necessary — we used the minimum Redundancy Maximum Relevance (*mRMR*) feature selection algorithm presented in [18]. The mRMR algorithm analyzes a dataset and aims to identify a set of features that are both "maximally relevant," i.e. that "have the largest mutual information. . . with the target class," and "minimally redundant," i.e. that have the smallest amount of mutual information amongst each other. We used version 0.1.11 of the pymRMR Python package provided by the authors of [18], with the "mutual information difference" (*MID*) feature selection method, to select the top features.

For every experiment, we began by training on a perfectly class-balanced training set, where exactly 50% of the samples in the training set belong to the "Close" class and the remaining 50% belong to the "Far" class. However, in some cases, training on a perfectly class-balanced training set produced a classifier that was overly biased towards one particular class. In the experiments discussed below, when applicable, we manually tuned the class balance of the training set to compensate for any biases and produce a classifier with better overall performance.

We began by training and evaluating a single classifier on data from a wide variety of different environments. After verifying that the generic classifier approach was infeasible, we developed specialized classifiers for use with fingerprints containing different numbers of detected APs.

2.7 Generic Classifier Experiments

Initially, we trained a single generic classifier on a combination of data from all of the publicly available Wi-Fi localization datasets listed in Table 2.1. Table 2.3 shows the number of usable samples yielded by the training set generation process described in Sections 2.3 and 2.4 for each dataset.

We created a training set with samples evenly distributed among the datasets listed in Table 2.1 by randomly selecting 9,000 "Close" and 8,000 "Far" fingerprint pairings from each of the datasets. We trained a `BaggingClassifier` on this training set, and individually evaluated the classifier on the remaining samples — i.e. those not selected for use in the training set — from each dataset. The percentage of true negative samples correctly identi-

Table 2.3: Number of Fingerprint Pairings Generated from Individual Datasets

Dataset Name	**"Far" Samples**	**"Close" Samples**
Miskolc	227,716	11,630
JUIndoorLoc	554,011	335,002
UJIndoorLoc	2,644,089	387,186
IPIN 2016 Tutorial	90,964	30,845
TampereU	374,556	17,193
Alcalá Tutorial	826,009	110,442
Suburban Home	491,832	97,443
Cory Hall, Floor 1	107,892	24,336
Cory Hall, Floor 3	314,847	34,029
Cory Hall, Floor 4	65,124	16,263

Table 2.4: Experiment Results for Generic Classifier on Individual Datasets

Dataset Name	**True Negatives**	**True Positives**	**Balanced Accuracy**
Miskolc	75.76%	42.47%	59.11%
JUIndoorLoc	36.00%	68.69%	52.34%
UJIndoorLoc	64.30%	57.82%	61.06%
IPIN 2016 Tutorial	71.71%	39.21%	55.46%
TampereU	75.92%	42.14%	59.03%
Alcalá Tutorial	84.48%	44.59%	64.53%
Suburban Home	42.75%	72.74%	57.74%
Cory Hall	59.06%	70.09%	63.57%
Average	63.75%	54.72%	59.27%

fied, the percentage of true positive samples correctly identified, and the balanced accuracy of the classifier for these "evaluation" portions of all of the individual datasets are shown in Table 2.4. The generic classifier's performance is not only low but inconsistent, varying across different environments even though data from each of those environments is present in equal capacity in the training set.

The results of the the generic classifier experiment indicate that a single, one-size-fits all classifier is insufficient for guaranteeing consistent performance across a wide range of environments with different average numbers of APs per fingerprint.

2.8 Specialized-Classifier Experiments

To pursue solid performance across different environments, we developed an ensemble of three classifiers, each tailored specifically for environments with a particular "AP density," or number of APs typically detected per fingerprint. We developed specialized classifiers for three different types of target environments:

1. Locations with a low AP density — 5 to 15 APs detected per fingerprint on average.
2. Locations with a moderate to high AP density — 30 to 70 APs detected per fingerprint on average.
3. Locations with a very high AP density — 70 to 90 APs detected per fingerprint on average.

Table 2.5 shows the percentage of true negative samples correctly identified, the percentage of true positive samples correctly identified, and the balanced accuracy for all of the experiments ran with the specialized classifiers. Figure 2.2 shows the precision–recall curves of the specialized classifiers on perfectly class-balanced subsets of each evaluation set, along with the analogous precision–recall curves of the generic classifier on perfectly class-balanced subsets of the evaluation sets described in Section 2.7.

In the following sections, we describe the details of the specialized classifiers developed for each of the three types of target environments above.

Table 2.5: Experiment Results for Specialized Classifiers

#	Dataset Name (Features)	Classifier Type	True Negatives	True Positives	Balanced Accuracy
1	Miskolc (All)	Low AP	71.66%	84.12%	77.89%
2	Suburban Home (All)	Low AP	63.45%	70.25%	66.85%
3	Miskolc (Top 7)	Low AP	71.31%	78.53%	74.92%
4	TampereU (All)	Med. AP	70.77%	68.06%	69.41%
5	IPIN 2016 Tutorial (All)	Med. AP	69.62%	66.02%	67.82%
6	Cory Hall, Floor 3	High AP	69.26%	70.43%	69.84%
7	Cory Hall, Floors 1 and 4	High AP	70.38%	73.47%	71.92%

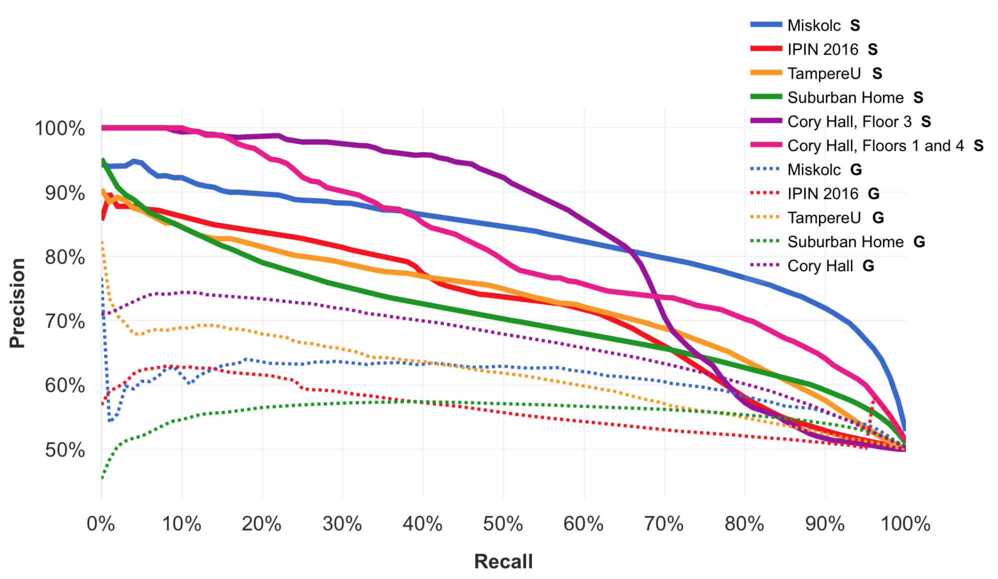


Figure 2.2: Precision–recall curves of the specialized classifiers on perfectly class-balanced subsets of each evaluation dataset from Section 2.5, along with the analogous precision–recall curves of the generic classifier. (S = specialized classifier, G = generic classifier)

2.8.1 Low AP Density Classifier

The median number of APs detected in a fingerprint from the suburban home and in a fingerprint from the publicly available Miskolc dataset is 8 and 10, respectively, making them both low AP density environments. Following the process described in Section 2.6, we created a training set from the suburban home data, containing all $97,443$ "Close" and $97,443$ randomly chosen "Far" fingerprint pairings, and used it to train a `BaggingClassifier`. We then evaluated the performance of the classifier on the entire set of fingerprint pairings extracted from the publicly available Miskolc dataset. The results of this evaluation, shown in row 1 of Table 2.5, indicate that a model trained on data from a single location can achieve fairly high recall on data from a completely different location with a similar AP density.

For comparison, we swapped the training and evaluation datasets, creating a class-imbalanced training set with all $11,630$ "Close" and $8,000$ randomly selected "Far" fingerprint pairings from the Miskolc dataset. This classifier performed worse than the previous one, as detailed in row 2 of Table 2.5. We believe that the main cause of this performance disparity is that the data we collected from the suburban home is specially tailored to allow learning algorithms to determine the signifiers of close proximity. Because fingerprints were recorded in "bursts" of 9 at a time, our own collected data yields many more pairs of fingerprints that were recorded in the exact same location, allowing learning algorithms to more easily establish a starker contrast between "Close" and "Far" samples. The Miskolc dataset's lack of extremely close fingerprint pairings may also be the reason why shifting the class balance slightly in favor of "Close" samples yielded better performance when using a training set derived from it. A secondary cause of the performance disparity may be the relative size of the training sets; the training set created from the suburban home data is more than 9 times larger than the one derived from the Miskolc dataset.

We also trained a second `BaggingClassifer` on the same suburban home training set, with the input feature set reduced to only the top 7 features identified by the mRMR algorithm when run on the same class-balanced data from the suburban home. Reducing the input feature set substantially reduced the time it took to train and evaluate the classifier, but reduced the classifier's recall by 5% to 6%, as shown in row 3 of Table 2.5.

2.8.2 Medium AP Density Classifier

The median number of APs detected in a fingerprint from the TampereU and IPIN 2016 Tutorial datasets is 38 and 32, respectively; thus, both datasets are of medium AP density. We created a class-balanced training set from the fingerprint pairings from the IPIN 2016 Tutorial dataset, consisting of all $30,845$ "Close" and $30,845$ randomly selected "Far" fingerprint pairings, and used it to train a `BaggingClassifier`. We evaluated the performance of that classifier on the entire set of fingerprint pairings extracted from the TampereU dataset; the results are shown in row 4 of Table 2.5. We then trained an identical classifier on a training set derived from the TampereU dataset, containing all $17,193$ "Close" and $15,000$ randomly selected "Far" fingerprint pairings, and evaluated it on the full set of fingerprint pairings extracted from the IPIN 2016 Tutorial dataset. The results of this evaluation are shown in row 5 of Table 2.5.

2.8.3 High AP Density Classifier

The median number of APs detected in a fingerprint from Cory Hall is between 52 and 76, depending on the floor, making it a high AP density environment. As detailed in Section 2.2, the data from Cory Hall can be organized into a set of 9-scan "bursts," each associated with a single physical position. A burst of fingerprints can yield data about significantly more APs than a single fingerprint; across all of the data collected from Cory Hall, the median number of APs detected in the first fingerprint of a burst is 65, while the median number of APs detected in at least one of the first four fingerprints of a burst is 86.

To take advantage of the additional data provided by bursts, without substantially reducing the size of the eventual training set, we divided each burst into two smaller "sub-bursts" — one comprised of the first 4 fingerprints recorded, and the other comprised of the next 4 fingerprints recorded. The final, ninth fingerprints from each burst were not used. During the feature extraction process described in Section 2.3, the fingerprints paired up for the Cory Hall datasets were "pseudo-fingerprints," one for each of the aforementioned sub-bursts. The pseudo-fingerprint for a given sub-burst contains an entry for every access point detected in at least one of the sub-burst's fingerprints. The RSSI value for a given AP in the pseudo-fingerprint is the median of all of the observed RSSI values for that AP from all of the fingerprints within the sub-burst.

We created a training set containing all $1,889$ "Close" and 300 randomly chosen "Far" pseudo-fingerprint pairings from the first and fourth floors of Cory Hall, and used it to train a `BaggingClassifier`. We evaluated the performance of the classifier on the entire set of pseudo-fingerprint pairings from the third floor of Cory Hall. The results of this evaluation are shown in row 6 of Table 2.5. The reason for choosing this particular class imbalance is that, unlike other datasets, for the vast majority of the "Close" fingerprint pairings from these particular floors, the two fingerprints were recorded in the exact same place; thus, a classifier trained on a training set derived from this dataset is more likely to misclassify samples recorded 1 or 2 meters apart as "Far." As such, when a perfectly class-balanced training set is used, over 50% of "Close" samples in the evaluation set are misclassified as "Far."

As in the preceding sections, for comparison, we swapped the training and evaluation datasets. We trained another `BaggingClassifier` on a training set derived from the pseudo-fingerprint pairings from the third floor. This training set contained all $1,549$ "Close" and $2,500$ randomly selected "Far" pseudo-fingerprint pairings. We evaluated this classifier on the full set of pseudo-fingerprint pairings from the first and fourth floors; it performed slightly better than the previous classifier, as indicated in row 7 of Table 2.5. In contrast, the balanced accuracy of a `BaggingClassifier` trained and evaluated on versions of the same training and evaluation sets that only used the first fingerprint of each sub-burst was only 65.90%.

Chapter 3

Proximity Detection Using Magnetometer Traces

In this chapter, we design a set of binary machine learning classifiers that use short segments of magnetometer traces to determine whether or not two devices are within immediate physical proximity, and discuss preliminary performance results from evaluations involving different buildings and and different device models.

The outline of this chapter is as follows: In Section 3.1, we discuss how our methods relate to and expand upon previous work. In Section 3.2, we detail where the existing magnetometer data used in our experiments was sourced, and how our new sets of magnetometer data were collected. In Section 3.3, we outline how we process the raw sensor data collected from devices to create coherent magnetometer traces from which training and evaluation sets can be compiled. In Section 3.4, we describe how we create training and evaluation sets from the magnetometer traces. In Section 3.5, we discuss the similarity measurements that are passed to our classifiers as input features. In Section 3.6, we present a set of binary machine learning classifiers, each trained and evaluated on data collected by a single smartphone in different buildings, and quantify their performance. In Section 3.7, we detail the pre-processing we apply to devices' magnetometer traces to compensate for the different magnetometer biases of different devices. In Section 3.8, we present an additional set of binary machine learning classifiers, each trained and evaluated on data from different sets of smartphone models, and characterize their performance.

3.1 Related Work

Over the past decade, there has been a growing amount of research on developing systems that localize devices within a building using sequences of magnetometer readings over time, i.e. magnetometer "traces." Disturbances in the Earth's magnetic field — caused by the steel used in the construction of many larger buildings, as well as by metallic objects and furniture — introduce distortions into smartphone magnetometer measurements. The particular pattern of distortions seen in a device's magnetometer readings over time serves as a distinct "fingerprint" of a given path being walked through a building.

Companies such as IndoorAtlas [11] have developed commercial indoor positioning platforms and APIs that localize devices primarily using their magnetometer traces. A number of magnetometer trace–based localization systems [25, 1, 29] have also been proposed and evaluated in the literature, with a typical average error of 1–3 meters. Similar to the Wi-Fi localization systems referenced in Section 2.1, before they can be deployed, these magnetometer trace–based localization systems require creating a database of reference traces collected by walking different paths throughout the building. Devices within the building can submit a sequence of magnetometer readings, usually spanning several seconds, to a central server, which finds the location or path in the database for which the nearby magnetometer readings from the reference traces are most similar to the ones submitted by the device.

Similar to the Wi-Fi–based classifiers we designed in Chapter 2, the classifiers we present in this chapter are concerned with determining whether or not two users are in immediate physical proximity, rather than estimating a user's location on a map, and as such can be used in any new location without first compiling a database of traces or performing other location-specific training or setup.

The authors in [12] present a method to detect whether two users are in close physical proximity by comparing segments of smartphone magnetometer traces; their proposed method is also intended for use in digital contact tracing. However, to measure the similarity between two trace segments, the authors' proposed method relies only on the Pearson correlation coefficient of the norms of the three-element magnetic field vectors that make up the segment. A threshold value is empirically determined based on the length of the trace segments being compared; a pair of segments is classified as "Close" if the Pearson correlation coefficient of their constituent vectors' norms is above that threshold. In contrast, our methods make use of additional pre-processing steps, as well as a much wider variety of similarity measurements that account for similarities or dissimilarities between each of the three elements of the magnetic field vectors, in addition to the norms; we use machine learning classifiers to, in effect, automatically determine the combination of features and threshold values that yields the highest proximity detection accuracy. Furthermore, the authors in [12] only evaluated their approach on pairs of segments that either (a) had the exact same start location, trajectory, and end location, or (b) were collected in two distinct areas very far away from each other. We go beyond this, evaluating our methods on segment pairings with a broader range of distances between their two constituent segments. As opposed to differentiating between segment pairings with the same exact trajectory and segment pairings recorded in completely different areas, the classifiers we present perform proximity detection at a finer level of granularity, with a more diverse set of "Close" traces; they differentiate between pairings recorded 2 or fewer meters apart on average and those recorded further apart but still within Bluetooth range. Lastly, all of the traces used in the evaluation of the method from [12] were collected with the same smartphone model; in this chapter, we introduce a way of compensating for different magnetometer biases in heterogeneous devices, and evaluate our methods with this added mitigation by training and evaluating classifiers on different subsets of traces from 4 different smartphone models.

3.2 Data Sources

Several magnetometer trace datasets, intended for use in developing and benchmarking magnetometer trace–based localization systems, are publicly available online. We used traces from the MagPIE dataset [9, 8] to create training and evaluation sets for our classifiers. This dataset contains raw traces from a single smartphone's magnetometer, recorded in 3 different academic buildings across the campus of the University of Illinois Urbana-Champaign. Some of the traces were recorded while a person walked throughout the buildings holding the smartphone, and others were recorded while the smartphone was mounted on a wheeled robot. Some of the traces designated as "Test Data" are "live load cases"; for these traces, to simulate changes in an environment over time, the authors deliberately moved certain objects to positions different from those that they had occupied during the recording of all of the other traces. In a hypothetical exposure notification app based on our proximity detection methods, most if not all of the traces that would be input to the classifiers would be recorded as users walked around with their smartphones somewhere on their person, and by default only pairs of traces recorded around the same time would be input to classifiers for comparison. Thus, to compile our training and evaluation sets, we used only the non-"live load" traces designated "Training Data" that were recorded while a person walked holding the smartphone.

In a real-world exposure notification app, the traces that would be input to the classifiers would be recorded by a diverse range of devices, but all of the traces in the MagPIE dataset were recorded by the same smartphone. In order to be able to evaluate how well our methods hold up to inputs from heterogeneous devices, we also collected new magnetometer traces, using the same tools employed in the creation of the MagPIE dataset. We collected our own magnetometer trace data in Cory Hall, a five-story academic building on the UC Berkeley campus. Over the course of two days in late April 2022, we recorded magnetometer traces in a roughly 30m $\times$ 15m lab space on the third floor of the building, using 4 different Android smartphones.

Table 3.1 shows the devices we used to collect fingerprint data and the versions of Android they were running at the time of data collection. Magnetometer traces from the 4 devices were recorded using a slightly modified version of the "MagnetometerV2" data collection app [14] released by the authors of the MagPIE dataset. Ground-truth position data was simultaneously recorded using the MagPIE authors' "BRG_Trajectory" app [6], which uses the Google Tango API, on a Tango–equipped Lenovo Phab 2 Pro running Android 6.0.1. At the start of each of the two data collection sessions, prior to recording any magnetometer traces, the entire lab area was slowly walked with the BRG_Trajectory app in "Learning Mode" to perform area learning. The area learning process generates an Area Description File (ADF) describing the space, which was subsequently used to continuously localize the Phab 2 Pro within the mapped lab area while each magnetometer trace was recorded.

Table 3.1: Devices Used to Collect Magnetometer Traces

Device	**Android Version**
Samsung Galaxy S8	9
Oppo RX17 Pro	8.1
Google Pixel 3	10
Google Pixel	10

Magnetometer traces were recorded while walking a number of intersecting linear and non-linear paths — navigating around cubicles, desks, office chairs, electronic and mechanical equipment, and computers — with the smartphone recording the trace held flat in the palm of one hand, and the Phab 2 Pro held vertically upright, i.e. with its rear cameras facing forward, in the other hand. Since the clocks on any given pair of 2 of our test devices usually differ by 1 to 2 seconds, in order to align the timestamps of the position and magnetometer logs, we ensured that both the Phab 2 Pro and the phone recording the magnetometer trace began recording at the exact same time by tapping the "Start Recording" button on both devices at as close to the same instant as we were able to. For each log file generated by one of the 4 test devices or by the Phab 2 Pro, we subtracted the timestamp of the first entry from all of the timestamps in the log file — essentially making all of the log file timestamps relative to the moment at which recording began.

3.3 Trace Synthesis and Preparation

In this section, we describe how we synthesize the raw sensor data in the log files output by the MagnetometerV2 and BRG_Trajectory apps into coherent magnetometer traces whose trajectories and magnetometer readings can easily be compared.

For each recording, the output of the MagnetometerV2 app consists of three log files, which contain timestamped magnetometer, accelerometer, and gyroscope readings, respectively. Future work will involve using the accelerometer and gyroscope log files to tilt-compensate the magnetometer readings, but for the experiments discussed in this chapter, those files were not used. Likewise, for each recording, the BRG_Trajectory app on the Phab 2 Pro produces a log file of timestamped position measurements, in local coordinates relative to the point at which area learning was started. Each of these log files contains data from different hardware sensors that operate at different frequencies — the timestamps are not aligned across log files, and there are often cases in which there are no position measurements with the same exact timestamp as a particular magnetometer reading. In order to produce a single coherent trace — a single sequence of entries, where each entry contains a single timestamp, a single position measurement, and a single magnetometer reading — we match each magnetometer reading in the magnetometer log file with a "nearby" position measurement from the corresponding position log file, eliminating magnetometer readings from the final trace if there were no position measurements recorded sufficiently close in time to the magnetometer reading.

To perform this matching, for each recording, we iterate through every magnetometer reading m_i in the magnetometer log file, starting at the very beginning of the log. For each reading m_i, we search the position log file to find the position measurement $p_{c,i}$ whose timestamp is closest to that of m_i. If the timestamp of $p_{c,i}$ is within δ seconds of the timestamp of m_i, we add a new entry to the final trace containing both m_i and $p_{c,i}$, and set its timestamp equal to m_i's timestamp. Otherwise, we ignore m_i and move on to the next magnetometer reading.

For all of the experiments in this chapter, we set δ to 0.01 seconds. This allows us to preserve most of the magnetometer readings from the original log file in the final trace, while ensuring that the position measurement paired with each magnetometer reading m_i represents the state of the device at a moment in time sufficiently close to that at which m_i was recorded, and thus represents the approximate state of the device at the exact moment at which m_i was recorded. In addition, once we have paired some position measurement p_i with a magnetometer reading that is added to the final trace, we remove it from the list of position log file entries to search. This guarantees that the final trace contains exactly one magnetometer reading for each instantaneous position measurement.

3.4 Training Sets and Proximity Classes

Our classifiers evaluate two magnetometer trace segments at a time, determining whether or not they were recorded in immediate physical proximity to each other. Each classifier takes as input two magnetometer trace segments of equal temporal length L_S and predicts whether or not the two traces were captured within roughly 2 meters of each other. Figure 3.1 shows the high-level design of our magnetometer trace–based proximity detection system.

In this section, we describe how we generate training and evaluation sets from the magnetometer traces output by the process described in the previous section.

3.4.1 Segment Extraction

Once the various log file entries for each recording have been synthesized into a single trace, segments of length L_S are extracted from the trace for use in training and evaluation sets. For each trace, two sets, or "tracks," of segments are extracted, where one track is offset from the other by $\frac{L_S}{2}$ seconds. Specifically, each track corresponds to a particular starting position in the trace; the starting position for the first track is 0 seconds and the starting position for the second track is $\frac{L_S}{2}$ seconds. For each track, the trace is divided into contiguous, non-overlapping segments of length L_S, with the first segment covering the L_S seconds immediately following the track's starting position, the second segment covering

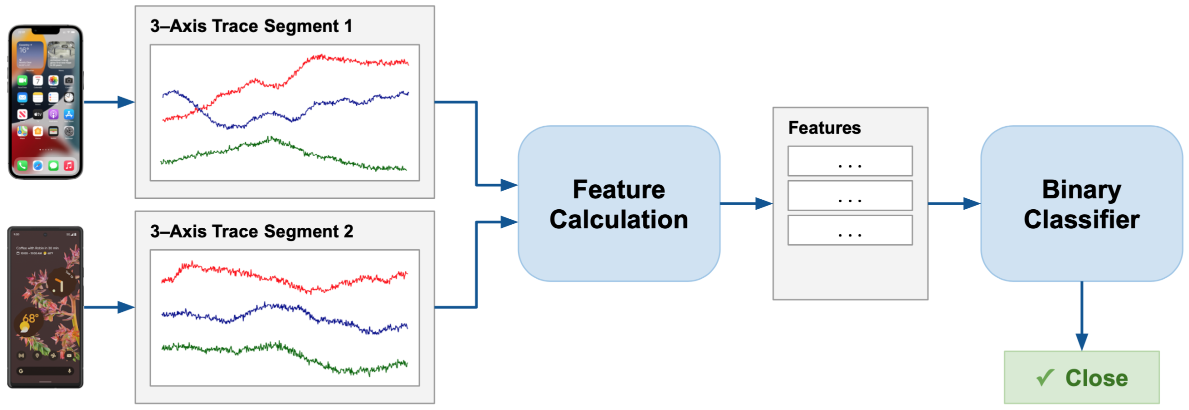


Figure 3.1: The high-level design of our magnetometer trace–based proximity detection system.

the L_S seconds immediately following the end of the first segment, and so on, with the last segment ending at second $L_S \times \lfloor \frac{L_T}{L_S} \rfloor$ of the trace, where L_T is the length of the trace.

Figure 3.2 shows the segments that would be extracted from a 25-second trace, i.e. $L_T = 25$ seconds, when $L_S = 10$ seconds. The segments from the first track cover the intervals $[0\text{s}, 10\text{s}]$ and $(10\text{s}, 20\text{s}]$. The segments from the second track cover the intervals $[5\text{s}, 15\text{s}]$ and $(15\text{s}, 25\text{s}]$.

Compared to a sliding-window method, this method of segment extraction yields fewer segments, and thus smaller training and evaluation sets. However, it bounds the amount of overlap between segments — a given pair of segments share at most 50% of their magnetometer readings, and a given pair of segments from the same track share none of their magnetometer readings. This ensures that, after all possible segment pairings are enumerated and then split into a training set and an evaluation set, the contents of the segment pairings that make up the training set are substantially different than the contents of those that make up the evaluation set. As we detail later in Section 3.4.3, we do not pair up segments that come from different tracks, so a segment pair in the training set will share at most 50% of its entries with any given pair in the evaluation set. Extracting two tracks of segments, and not pairing up segments from different tracks, allows us to double the size of our training and evaluation sets without introducing many substantially similar segment pairs into the training and evaluation sets.

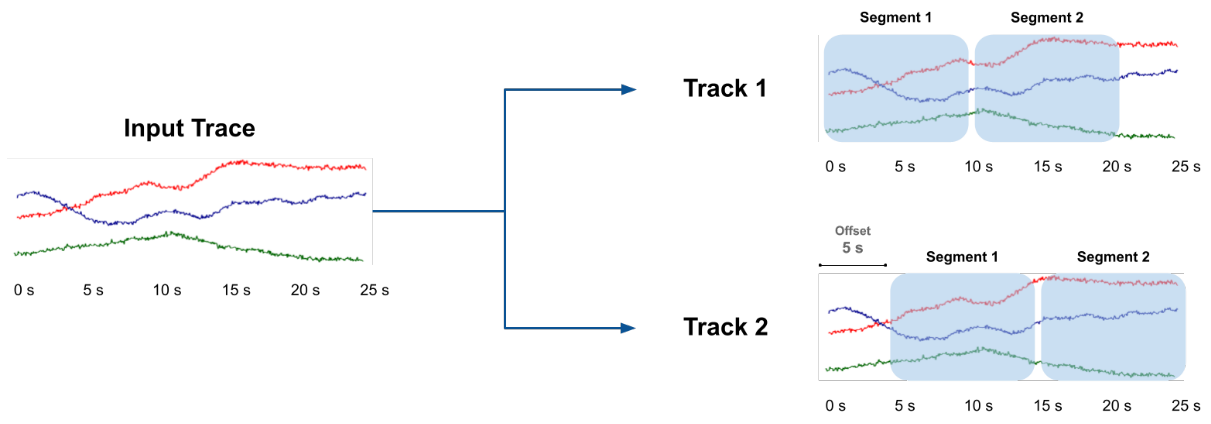


Figure 3.2: An example of our segment extraction process, showing all 4 of the segments that would be extracted from a 25-second trace when the segment length $L_S = 10$ seconds.

3.4.2 Segment Length (L_S)

For all of the experiments in this chapter, we set the segment length $L_S = 10$ seconds. This time interval is long enough to allow distinct signatures of different paths to appear in segments, but short enough that if two users were close together long enough for transmission to occur, that period of contact would represent a significant portion of the segment. Using a short segment length also enables a single system to easily adapt to a wide range of fine-grained contact duration thresholds for different infectious diseases or variants; to determine if two individuals were in immediate physical proximity for some arbitrary amount of time that is a multiple or near-multiple of L_S, it is trivial to keep a running total of the number of contiguous L_S-second intervals for which the two were estimated to have been in immediate physical proximity. Note that the CDC's guidelines indicate a high probability of COVID-19 transmission between two people who are within 2 meters of each other for at least 15 minutes.

3.4.3 Proximity Classes

To prepare training and evaluation sets from a given pool of segments, we first isolate segments into different groups. For the segments from the MagPIE dataset, we created 3 groups — one group for each building in which traces were recorded, such that each group contained all of the segments that were recorded in that particular building. For the segments we recorded in Cory Hall, we created 2 groups — one group for each day on which traces were recorded, such that each group contained all of the segments that were recorded on that particular day.

We then enumerate every possible pairing of two distinct segments that are both from the same group *and from the same track*. Then, for each segment pairing (S_i, S_j), we iterate through every entry of S_j. For each entry e_k of S_j, we find the entry $e_{c,k}$ of S_i whose timestamp, relative to the start of the segment, is closest to that of e_k. We then calculate the two-dimensional distance d_k between the position measurements in e_k and $e_{c,k}$. We assign each segment a "proximity class" based on an analysis of all of the entry distances d_k for the pairing; the sampling frequency for our magnetometer traces is 50 Hz, so roughly 500 entry distances d_k are compared. If the percentage of entries e_k for which $0\text{ meters} \leq d_k \leq 2.25\text{ meters}$ is at least P, the fingerprint pairing's proximity class is set to "Close." If the percentage of entries e_k for which $3.25\text{ meters} \leq d_k \leq 20\text{ meters}$ is at least P, it is set to "Far." For each of the sets of experiments in this chapter, we evaluate our classifier design with two different choices of P: $P = 75\%$ and $P = 95\%$.

Segment pairings are dropped from the training and evaluation sets if neither of these two criteria are met. This has the effect of excluding from the training and evaluation sets any pairs of segments for which a significant portion of the corresponding entries were recorded more than 20 meters apart on average. As with the training sets for the Wi-Fi–based classifiers, we do this in an effort to focus the training process on the fine-grained differentiation between pairings recorded in immediate physical proximity and those recorded in somewhat close physical proximity, i.e. Bluetooth range. This is in contrast to an approach which would optimize our classifiers for the coarse-grained differentiation of pairings recorded in immediate physical proximity and those recorded very far away from each other. Our method of classifying segments also excludes from the training and evaluation sets any fingerprint pairings for which a significant portion of the corresponding entries were recorded between 2.25 and 3.25 meters apart. Again, as with the training sets for the Wi-Fi–based classifiers, we do this in an effort to focus the training process on differentiating between pairings recorded in immediate physical proximity and those recorded in somewhat close physical proximity, rather than on border cases that are nearly at the distance cutoff for the "Close" label.

The fact that we only consider segment pairs (S_i, S_j) where both S_i and S_j are from the same track guarantees that none of the entries in S_i also appear in S_j. This more closely mimics the segment pairs that would be input to the classifiers in a real-world usage scenario, wherein each segment would belong to a different user's trace, and there wouldn't be any entries that would appear in both segments. Likewise, for the Cory Hall traces, only considering segment pairs where both segments were recorded on the same day ensures that, for any given segment pairing (S_i, S_j) in the training or evaluation sets, the segments S_i and S_j were recorded within a few hours of each other, and thus that the physical environment in which they were recorded remains the same across the two segments. This ensures that the training and evaluation sets more closely reflect real-world inputs — in a hypothetical proximity detection system based on our methods, only segments recorded at or around the same time would be input to the classifiers.

Table 3.2 shows the total number of "Close" and "Far" segment pairings generated from

Table 3.2: Number of Segment Pairings Generated from Trace Subsets

Subset Name	**P = 75%**		**P = 95%**	
	“Far” Samples	**“Close” Samples**	**“Far” Samples**	**“Close” Samples**
“CSL First Floor”	44,297	1,669	31,837	1,256
“Loomis First Floor”	19,278	1,460	9,270	1,116
“Talbot Third Floor”	12,378	698	6,857	462
Cory Hall, Day 1	4,188	587	2,237	367
Cory Hall, Day 2	2,174	455	929	202

each of the 3 buildings from the MagPIE dataset and each of the 2 recording sessions in Cory Hall, both when $P = 75\%$ and when $P = 95\%$.

3.5 Classifier Input Features

For each segment pairing, we calculate a set of features, which are the inputs actually passed directly to the classifiers described in Sections 3.6 and 3.8. As with the Wi-Fi–based classifiers, we chose to include the majority of these input features because we hypothesized that they could accurately quantitatively express of the level of similarity between the two segments in a pairing. In this section, we describe our chosen features in detail.

A single smartphone magnetometer reading defines a vector in 3D space; the direction of the vector represents the direction of the Earth's magnetic field — after accounting for any distortions caused by nearby metallic objects — while the length of the vector represents the intensity of Earth's magnetic field. Thus, each reading has three components — an X component, a Y component, and a Z component, which represent the magnetic field intensity measured by the magnetometer along the X, Y, and Z axes defined by Android's `Sensor` APIs [24].

For a given magnetometer trace segment S, let $|S|$ denote the number of entries in S. Most of the 50 Hz magnetometer traces we use contain around 500 entries; the actual length can differ slightly — it might be, for example, 496 entries — due to very slight variations in the sampling rate over time. Define $M_{S,i}[x]$, $M_{S,i}[y]$, and $M_{S,i}[z]$ as the X, Y, and Z components of the magnetometer reading in the i-th entry of the segment S. Additionally, define $|M_{S,i}|$ as the norm of the magnetic field vector defined by the magnetometer reading in the i-th entry of S.

Let $M_{\text{X}}(S)$, $M_{\text{Y}}(S)$, and $M_{\text{Z}}(S)$ be vectors containing only the X, Y, and Z components from the magnetometer readings in the entries of S, in the same order as their corresponding entries:

$$\begin{aligned}
M_{\mathrm{X}}(S) &= \left[\, M_{S,1}[x],\ M_{S,2}[x],\ M_{S,3}[x],\ \dots\ M_{S,|S|}[x] \,\right] \\
M_{\mathrm{Y}}(S) &= \left[\, M_{S,1}[y],\ M_{S,2}[y],\ M_{S,3}[y],\ \dots\ M_{S,|S|}[y] \,\right] \\
M_{\mathrm{Z}}(S) &= \left[\, M_{S,1}[z],\ M_{S,2}[z],\ M_{S,3}[z],\ \dots\ M_{S,|S|}[z] \,\right]
\end{aligned}$$

Additionally, let $M_{\mathrm{N}}(S)$ be a vector containing only the norms of the magnetometer readings in the entries of S, in the same order as their corresponding entries:

$$M_{\mathrm{N}}(S) = \left[\, |M_{S,1}|,\ |M_{S,2}|,\ |M_{S,3}|,\ \dots\ |M_{S,|S|}| \,\right]$$

Consider a pair of magnetometer trace segments, S_i and S_j. For each set of the segment pair's single-axis vectors — $(M_{\mathrm{X}}(S_i), M_{\mathrm{X}}(S_j))$, $(M_{\mathrm{Y}}(S_i), M_{\mathrm{Y}}(S_j))$, and $(M_{\mathrm{Z}}(S_i), M_{\mathrm{Z}}(S_j))$ — as well as the set of the pair's norm vectors $(M_{\mathrm{N}}(S_i), M_{\mathrm{N}}(S_j))$, the following measurements of similarity between the two segments' vectors are passed as input features to the classifiers:

1. The cosine similarity of the two vectors.
2. The Pearson correlation coefficient of the two vectors.
3. The Spearman correlation coefficient of the two vectors.
4. The Kendall correlation coefficient of the two vectors.

These similarity measurements are intended to provide the classifiers with indications of how well correlated the two segments are along each axis — how much the fluctuations in the measured magnetic field strength in one segment match or align with the fluctuations in the other segment.

Furthermore, for a pair of traces (S_i, S_j), let $D_{\mathrm{X}}(S_i, S_j)$, $D_{\mathrm{Y}}(S_i, S_j)$, and $D_{\mathrm{Z}}(S_i, S_j)$ be vectors containing the differences between the X, Y, and Z components of the magnetometer readings in the entries of S_i and S_j, respectively:

$$\begin{aligned}
D_{\mathrm{X}}(S_i, S_j) &= \left[\, M_{S_i,k}[x] \,-\, M_{S_j,k}[x]\ \dots\ \forall\, k \le \min(|S_i|, |S_j|) \,\right] \\
D_{\mathrm{Y}}(S_i, S_j) &= \left[\, M_{S_i,k}[y] \,-\, M_{S_j,k}[y]\ \dots\ \forall\, k \le \min(|S_i|, |S_j|) \,\right] \\
D_{\mathrm{Z}}(S_i, S_j) &= \left[\, M_{S_i,k}[z] \,-\, M_{S_j,k}[z]\ \dots\ \forall\, k \le \min(|S_i|, |S_j|) \,\right]
\end{aligned}$$

Additionally, let $D_{\mathrm{N}}(S_i, S_j)$ be a vector containing the differences between the norms of the magnetometer readings in the entries of S_i and S_j:

$$D_{\mathrm{N}}(S_i, S_j) = \left[\, |M_{S_i,k}| \,-\, |M_{S_j,k}|\ \dots\ \forall\, k \le \min(|S_i|, |S_j|) \,\right]$$

For each of these individual vectors of differences — for $D_{\mathrm{X}}(S_i, S_j)$, $D_{\mathrm{Y}}(S_i, S_j)$, $D_{\mathrm{Z}}(S_i, S_j)$, and $D_{\mathrm{N}}(S_i, S_j)$ — the following similarity measurements are passed as input features to the classifiers:

1. The mean of all of the vector's elements.
2. The median of all of the vector's elements.
3. The standard deviation of all of the vector's elements.

These features are intended to provide the classifiers with a high-level summary of how far apart the magnetometer readings in the two segments tend to be along each axis, and an indication of whether or not this difference between the two segments' readings tends to remain the same — i.e. whether or not the fluctuations along a particular axis in each segment follow a similar pattern.

3.5.1 Segment Alignment

In our empirical analysis of pairs of segments from the MagPIE dataset, we found that, in a significant number of "Close" segment pairs, both segments contain almost exactly the same pattern of magnetic field strength fluctuations, but the pattern begins at a different time within each segment, such that one segment "lags" another, usually by at most 1 second. This phenomenon likely appears in segment pairs where two users are walking together for some time, but one is slightly ahead of the other.[1] To make this relatively common case easier for the classifiers to detect, we developed a method to align each pair of segments in the training and evaluation sets for all of the experiments described in Sections 3.6 and 3.8.

For a given segment S, of length L_S seconds, define $S[a,b]$, where $a < b$, as the subsequence of S starting with the first entry whose timestamp is $\geq a$ seconds after the start of S and ending with the last entry whose timestamp is $\leq b$ seconds after the start of S. Then, define $T((S_i, S_j), \Delta)$ as follows:

$$T((S_i, S_j), \Delta) \quad = \quad \begin{cases} (S_i[-\Delta, L_S], S_j[0, L_S + \Delta]) & \text{if } \Delta < 0 \\ (S_i[0, L_S - \Delta], S_j[\Delta, L_S]) & \text{if } \Delta > 0 \end{cases}$$

In essence, the transformation T "shifts" S_i either forwards (if $\Delta > 0$) or backwards (if $\Delta < 0$) temporally relative to S_j, such that the two segments only overlap temporally for $L_S - \Delta$ seconds, and then removes the length-Δ portions of S_i and S_j that now lie outside this overlap region, thus producing two completely overlapping segments of length $L_S - \Delta$.

For each pair of segments (S_i, S_j), after calculating all of the features described above, we enumerate all pairs of transformed segments of the form $T((S_i, S_j), \Delta)$ where -2 seconds $\leq \Delta \leq +2$ seconds and $\Delta \mod 0.25 = 0$. We then find Δ^*, the value of Δ for which the Pearson correlation coefficient of the norms of the magnetometer readings in the segment pair $T((S_i, S_j), \Delta)$ is maximized. Finally, we calculate all of the features described above for $T((S_i, S_j), \Delta^*)$. Thus, the full input to the classifiers consists of two sets of the features

[1]Or, in this specific case, considering how the MagPIE dataset was collected, when one person is walking very close to where they had previously walked while recording a different trace.

Table 3.3: Results of Homogeneous-Device Experiments

Evaluation	**P = 75%**			**P = 95%**		
Building Name	**TN**	**TP**	**BalAc**	**TN**	**TP**	**BalAc**
"CSL First Floor"	99.69%	78.97%	89.33%	99.37%	82.56%	90.96%
"Loomis First Floor"	89.02%	97.74%	93.38%	86.03%	98.66%	92.34%
"Talbot Third Floor"	90.47%	91.83%	91.15%	89.25%	94.59%	91.92%
Average	93.06%	89.51%	91.28%	91.55%	91.93%	91.74%

described above — one for the original, unshifted S_i and S_j, and another for $T((S_i, S_j), \Delta^*)$ — plus a feature *Best shift offset* whose value is equal to Δ^*.

We chose to limit the values of Δ we examine to those between -2 and $+2$ seconds, because applying T should only be useful for "Close" segment pairs — if the two segments of a "Far" pair happen to contain the same pattern of fluctuations, shifting the segments enough to perfectly align the two instances of the pattern would only risk getting the pair misclassified as a "Close" pair. Since the average human walking speed is roughly 1 meter per second, shifting a segment temporally by up to ± 2 seconds compensates for a distance gap of up to ± 2 meters between the segments, which is nearly the largest possible distance gap for a "Close" pair.

3.6 Homogeneous-Device Experiments

To determine how well our approach performs in a baseline setting without the complications introduced by heterogeneous devices, and how well it generalizes to different buildings, we trained a set of three classifiers on data from the MagPIE dataset. Each classifier was trained on data from 2 of the 3 buildings in which traces were recorded, and evaluated on data from the remaining building.

For each of the three cases, we created an evenly class-balanced training set consisting of all "Close" segment pairings and an equal number of randomly selected "Far" pairings from the 2 training buildings. We then used this training set to train a random forest classifier — specifically, an instance of the `RandomForestClassifier` class from scikit-learn version 0.22.1. Then, we evaluated the classifier on the entire set of "Close" and "Far" segment pairings from the evaluation building.

Table 3.3 shows the results of these evaluations — the percentage of true negative samples correctly identified, the percentage of true positive samples correctly identified, and the balanced accuracy for each classifier — when P, as defined in Section 3.4.3, is set to 75% and additionally when it is set to 95%. The results indicate that classifiers developed using our methods accurately detect immediate physical proximity in this baseline setting involving traces from homogeneous devices, and that those classifiers perform well overall when evaluated on data from buildings other than those whose data was included in the training set. The classifiers' performance remains strong across both looser and stricter definitions of "Close" and "Far," i.e. lower and higher values of P, which indicates that our methods are adaptable to a range of different proximity class definitions and contact duration thresholds.

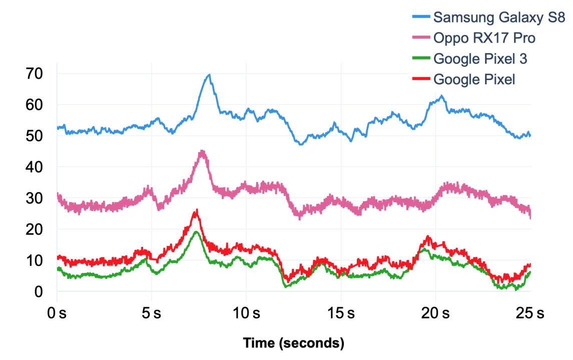
3.7 Mitigating the Effects of Device Heterogeneity

As with the Wi-Fi–based approach, a common challenge in creating indoor positioning systems that use magnetometer traces is dealing with the effects of device heterogeneity. Different magnetometers in different devices have different biases — different baseline values around which magnetometer readings fluctuate based on nearby objects. Furthermore, different magnetometers have different sensitivities — the amplitude of the fluctuations seen in magnetometer readings when walking the same exact path can vary across different device manufacturers and smartphone models. Examples of these varying biases and sensitivities can be seen in Figure 3.3, which shows the X components, Y components, Z components, and norms of the 4 test devices' magnetometer readings over time as each of the devices was individually taken down the same corridor in Cory Hall. Along each of the three axes, the pattern of fluctuations is the same in all 4 lines, but there are fairly large vertical gaps between the lines. Furthermore, there are small variations in the height of the "peaks" and depths of the "valleys" across the 4 traces.

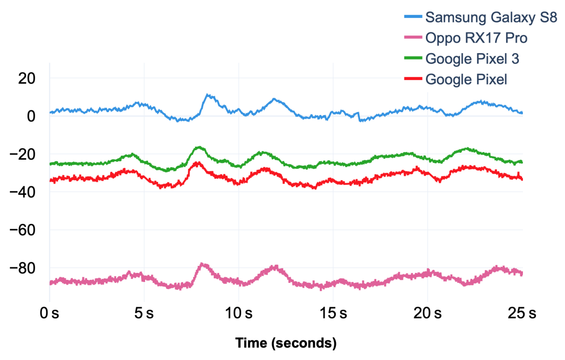
To make our methods as robust as possible in the face of inputs from heterogeneous devices, we explicitly compensate for the biases of individual magnetometers by determining a single baseline magnetometer reading for each device, and then subtracting that baseline reading from all of the magnetometer readings in the traces recorded by that device.

To determine the baseline magnetometer readings, the MagnetometerV2 app was used to record the 4 devices' magnetometer traces as they were all held steady in the exact same location — in the center of a field on the UC Berkeley campus, relatively far from buildings and other sources of magnetic anomalies — and in the same orientation in which they were held while traces were recorded in Cory Hall. For each device, we generated a trace from the log files output by the MagnetometerV2 app using the process described in Section 3.3. Figure 3.4 shows the X components, Y components, Z components, and norms of the 4 test devices' magnetometer readings over time as each of the devices was held steady in the field. We set the X, Y, and Z components of the baseline magnetometer to the average of those respective components across all of the magnetometer readings in the trace. Table 3.4 shows the baseline magnetometer measurements — these averages — for each of our 4 test devices.

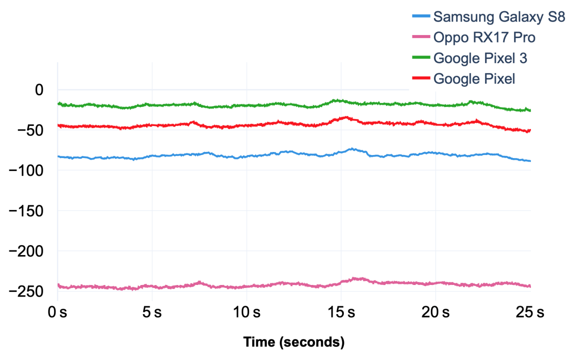
While the baseline subtraction described above compensates for the varying biases of different magnetometers, we have not yet implemented any measures explicitly aimed at compensating for the varying sensitivities of different magnetometers.



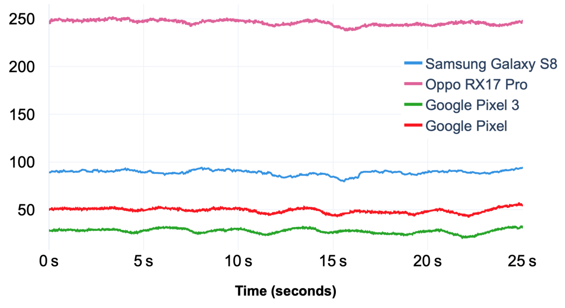
(a) X components (μT)



(b) Y components (μT)



(c) Z components (μT)

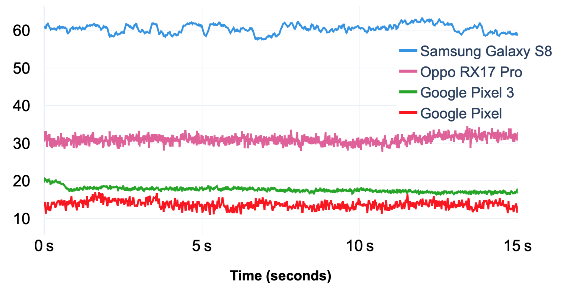


(d) Norms (μT)

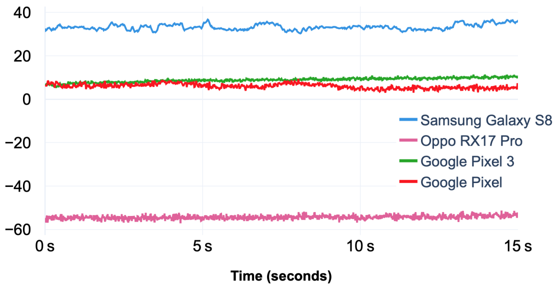
Figure 3.3: The X components, Y components, Z components, and norms of the 4 test devices' magnetometer readings, in μT, over time as each of the devices was individually taken down the same corridor. *Note that, because the plots' scales are different, fluctuations that appear larger than others on the plots may not have a larger actual magnitude.*

Table 3.4: Baseline Magnetometer Readings for Devices

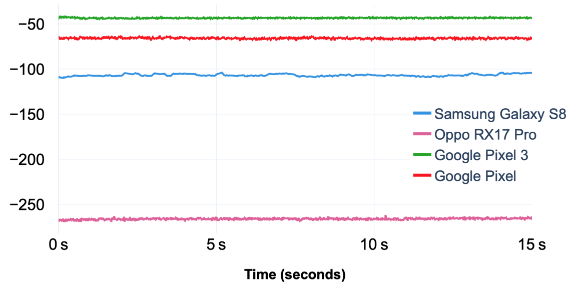
Device	**X**	**Y**	**Z**
Samsung Galaxy S8	60.2	34.2	−106.0
Oppo RX17 Pro	30.8	−54.4	−265.9
Google Pixel 3	17.5	9.1	−43.2
Google Pixel	13.6	5.5	−65.8



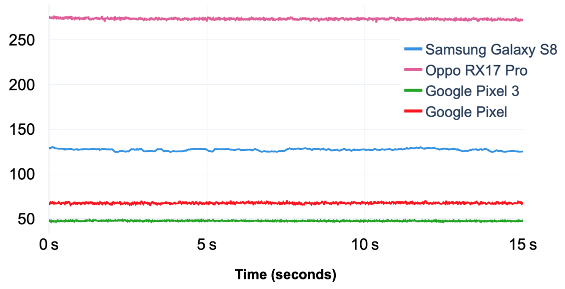
(a) X components (μT)



(b) Y components (μT)



(c) Z components (μT)



(d) Norms (μT)

Figure 3.4: The X components, Y components, Z components, and norms of the 4 test devices' magnetometer readings, in μT, over time as each of the devices was held steady in the center of a field far from buildings and other sources of magnetic anomalies. *Note that, because the plots' scales are different, fluctuations that appear larger than others on the plots may not have a larger actual magnitude.*

3.8 Heterogeneous-Device Experiments

To evaluate how well our methods — including the baseline subtraction described in the preceding section — perform in a heterogeneous-device setting, we trained a set of classifiers on the data recorded by our 4 test devices in Cory Hall. We trained 4 classifiers in total; each classifier was trained on segment pairs from 3 of the devices and evaluated only on segment pairs involving the remaining device. For each classifier, a different smartphone from among our 4 test devices was selected to the the "evaluation device." To compile the training and evaluation sets for each classifier, we started with the entire pool of segment pairs that we had previously generated from all of the Cory Hall traces using the process described in Section 3.4. We assigned all segment pairs for which at least one of the two segments was recorded by the evaluation device to the evaluation set. Then, from the remaining pairs not assigned to the evaluation set — the pairs for which neither of the two segments was

Table 3.5: Results of Heterogeneous-Device Experiments

Evaluation	**P = 75%**			**P = 95%**		
Device Name	**TN**	**TP**	**BalAc**	**TN**	**TP**	**BalAc**
Samsung Galaxy S8	98.52%	92.21%	95.36%	97.60%	94.52%	96.06%
Oppo RX17 Pro	98.37%	92.36%	95.36%	99.15%	93.53%	96.34%
Google Pixel 3	97.46%	96.15%	96.80%	97.23%	96.67%	96.95%
Google Pixel	98.83%	88.93%	93.88%	99.87%	89.36%	94.61%
Average	98.29%	92.41%	95.35%	98.46%	93.52%	95.99%

recorded by the evaluation device — we compiled a training set. This training set consists of all “Close” pairs from the set of remaining segment pairs, plus just enough randomly selected “Far” pairs to make the ratio of “Close” to “Far” pairs in the training set 1 : 1. We then trained a `RandomForestClassifier` on this class-balanced training set, and evaluated it on the entire evaluation set.

Table 3.5 shows the results of these heterogeneous-device experiments — the percentage of true negative samples correctly identified, the percentage of true positive samples correctly identified, and the balanced accuracy for each classifier — when P, as defined in Section 3.4.3, is set to 75% and additionally when it is set to 95%. they confirm that classifiers developed using our methods can distinguish between “Close” and “Far” segment pairs recorded by heterogeneous devices with very high accuracy, and that their performance is solid even when evaluated on data from devices other than those whose data was included in the training set.

Initially, in addition to subtracting baseline magnetometer readings from all of the magnetometer readings within each segment, we also calculated and provided to the classifiers three additional input features for each segment pair. These three features’ values were equal to the differences between the X, Y, and Z components of the two device-specific baseline measurements subtracted from the two segments, respectively. However, this did not improve the performance of the classifiers.

Notably, the Cory Hall traces used in these experiments were not tilt-compensated. As established in Section 3.5, a smartphone magnetometer reading represents a vector in 3D space. At any given position, the direction of Earth’s magnetic field relative to the smartphone — and thus the magnetometer reading — varies based on the roll and pitch of the smartphone. The traces input to a real-world proximity detection system will have been recorded by devices in countless different orientations; to enable accurate trace comparisons, each individual vector within each trace will need to be individually rotated to where it would be if the device were held in some arbitrary reference orientation. In this case, though, all 4 test devices were held in the same horizontal orientation during the recording of all of the traces, so tilt compensation is not strictly necessary to enable accurate comparison of their magnetometer readings. Future work will include using the accelerometer and gyroscope data recorded by the MagnetometerV2 app to tilt-compensate the traces, likely using one of several well-established tilt-compensation procedures.

Chapter 4

Conclusions and Future Work

Our evaluations of the Wi-Fi RSSI fingerprint–based classifiers in Chapter 2 show that, while a generic classifier is unable to generalize well to a wide variety of environments, an ensemble of specialized classifiers for environments with different AP densities can detect immediate physical proximity more accurately, with roughly 70% balanced accuracy on average.

Meanwhile, our evaluations of the magnetometer trace–based classifiers in Chapter 3 show that classifiers developed using our magnetometer trace–based methods can distinguish between "Close" and "Far" segment pairs from non–tilt-compensated traces with over 90% balanced accuracy on average, indicating that magnetometer trace–based methods are a more promising means of performing immediate proximity detection than the comparison of Wi-Fi RSSI fingerprints. Our results also show that our classifiers can generalize well to different buildings and devices whose traces were not present in their training data.

Many of the same fingerprint-based techniques discussed in Chapter 2 could also be applied to fingerprints constructed from the MAC addresses and RSSI values of nearby BLE devices. Generally, the range in which BLE devices can be detected is smaller than that in which typical Wi-Fi APs can be detected, so the addition of BLE fingerprints may prove valuable for improving the accuracy of immediate proximity detection. Meanwhile, future work for the magnetometer trace–based methods includes using the accelerometer and gyroscope measurements captured by the MagnetometerV2 app to tilt-compensate magnetometer traces; as noted above, tilt compensation is necessary in a real-world proximity detection system whose input traces are recorded by devices in many different orientations. For both the Wi-Fi–based and magnetometer–based methods, future directions could include exploring additional classifier input features, pre-processing steps, or learning algorithms that have the potential to boost the classifiers' accuracy. A final strand of future work could explore sensor fusion approaches for building a single combined proximity detection system that makes use of magnetometer traces, Wi-Fi RSSI fingerprints, and possibly other sensor data simultaneously.

Lastly, in an actual exposure notification contract tracing system, participating devices would need to continuously record and save RSSI fingerprints or magnetometer traces to later periodically compare with those recorded by infected individuals. Those infected individuals' fingerprints or traces would likely need to be distributed to participating devices by centralized servers. Another practical consideration is that even if two devices are physically close to each other, their two users might not necessarily be in the same room; they could be separated by a wall or door. As such, the methods we have described in this chapter are merely the basic building blocks of any practical exposure notification or contact tracing system, and require additional enhancements before they can be deployed.

Bibliography

[1] Han Jun Bae and Lynn Choi. "Large-Scale Indoor Positioning using Geomagnetic Field with Deep Neural Networks". In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. 2019, pp. 1–6. DOI: `10.1109/ICC.2019.8761118`.

[2] Philipp Bolliger. "Redpin - Adaptive, Zero-Configuration Indoor Localization through User Collaboration". In: *Proceedings of the First ACM International Workshop on Mobile Entity Localization and Tracking in GPS-Less Environments*. MELT '08. San Francisco, California, USA: Association for Computing Machinery, 2008, pp. 55–60. ISBN: 9781605581897. DOI: `10.1145/1410012.1410025`. URL: `https://doi.org/10.1145/1410012.1410025`.

[3] Pascal Brogle and Philipp Bolliger. *Redpin on SourceForge*. URL: `https://sourceforge.net/projects/redpin/`.

[4] Robert Bryll, Ricardo Gutierrez-Osuna, and Francis Quek. "Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets". In: *Pattern Recognition* 36.6 (2003), pp. 1291–1302. ISSN: 0031-3203. DOI: `https://doi.org/10.1016/S0031-3203(02)00121-8`. URL: `https://www.sciencedirect.com/science/article/pii/S0031320302001218`.

[5] California Department of Technology. *CA Notify*. 2020. URL: `https://canotify.ca.gov`.

[6] Alex Faustino. *alex-faustino/BRG_Trajectory: Android app for a Google Tango enable device.* 2017. URL: `https://github.com/alex-faustino/BRG%5C_Trajectory`.

[7] Government of Singapore. *TraceTogether*. 2020. URL: `https://www.tracetogether.gov.sg`.

[8] Bretl Research Group. *Magnetic Positioning Indoor Estimation (MagPIE) Dataset*. URL: `http://bretl.csl.illinois.edu/magpie`.

[9] David Hanley et al. "MagPIE: A dataset for indoor positioning with magnetic anomalies". In: *2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. 2017, pp. 1–8. DOI: `10.1109/IPIN.2017.8115961`.

[10] Zhengyong Huang et al. “Clustering combined indoor localization algorithms for crowdsourcing devices: Mining RSSI relative relationship”. In: *2014 Sixth International Conference on Wireless Communications and Signal Processing (WCSP)*. 2014, pp. 1–6. DOI: `10.1109/WCSP.2014.6992130`.

[11] *IndoorAtlas*. URL: `https://www.indooratlas.com/`.

[12] Seungyeon Jeong, Seungho Kuk, and Hyogon Kim. “A Smartphone Magnetometer-Based Diagnostic Test for Automatic Contact Tracing in Infectious Disease Epidemics”. In: *IEEE Access* 7 (2019), pp. 20734–20747. DOI: `10.1109/ACCESS.2019.2895075`.

[13] Mikkel Baun Kjærgaard and Carsten Valdemar Munk. “Hyperbolic Location Fingerprinting: A Calibration-Free Solution for Handling Differences in Signal Strength (concise contribution)”. In: *2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)*. 2008, pp. 110–116. DOI: `10.1109/PERCOM.2008.75`.

[14] *MagnetometerV2.tar.gz ||PoweredbyBox*. URL: `https://app.box.com/s/8g155irfv4ldbqcc5arnzh`

[15] Germán Martın Mendoza-Silva, Joaquın Torres-Sospedra, and Joaquın Huerta. “A Meta-Review of Indoor Positioning Systems”. In: *Sensors* 19.20 (2019). ISSN: 1424-8220. DOI: `10.3390/s19204507`. URL: `https://www.mdpi.com/1424-8220/19/20/4507`.

[16] Germán Martın Mendoza-Silva et al. “Long-Term WiFi Fingerprinting Dataset for Research on Robust Indoor Positioning”. In: *Data* 3.1 (2018). ISSN: 2306-5729. DOI: `10.3390/data3010003`. URL: `https://www.mdpi.com/2306-5729/3/1/3`.

[17] Fabian Pedregosa et al. “Scikit-Learn: Machine Learning in Python”. In: *J. Mach. Learn. Res.* 12.null (Nov. 2011), pp. 2825–2830. ISSN: 1532-4435.

[18] Hanchuan Peng, Fuhui Long, and C. Ding. “Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.8 (2005), pp. 1226–1238. DOI: `10.1109/TPAMI.2005.159`.

[19] Hua Qian et al. “Indoor transmission of SARS-CoV-2”. In: *medRxiv* (2020). DOI: `10.1101/2020.04.04.20053058`. eprint: `https://www.medrxiv.org/content/early/2020/04/07/2020.04.04.20053058.full.pdf`. URL: `https://www.medrxiv.org/content/early/2020/04/07/2020.04.04.20053058`.

[20] Priya Roy et al. “JUIndoorLoc: A Ubiquitous Framework for Smartphone-Based Indoor Localization Subject to Context and Device Heterogeneity”. In: *Wirel. Pers. Commun.* 106.2 (May 2019), pp. 739–762. ISSN: 0929-6212. DOI: `10.1007/s11277-019-06188-2`. URL: `https://doi.org/10.1007/s11277-019-06188-2`.

[21] Omer Sagi and Lior Rokach. “Ensemble learning: A survey”. In: *WIREs Data Mining and Knowledge Discovery* 8.4 (2018), e1249. DOI: `https://doi.org/10.1002/widm.1249`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1249`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1249`.

[22] E. Sansano et al. *UJI Indoor positioning and navigation repository.* 2016. URL: `http://indoorlocplatform.uji.es`.

[23] Piotr Sapiezynski et al. "Inferring Person-to-person Proximity Using WiFi Signals". In: *CoRR* abs/1610.04730 (2016). arXiv: `1610.04730`. URL: `http://arxiv.org/abs/1610.04730`.

[24] *Sensors Overview ||AndroidDevelopers.* URL: `https://developer.android.com/guide/topics/sensors/sensors_overview#sensors-coords`.

[25] Shervin Shahidi and Shahrokh Valaee. "GIPSy: Geomagnetic indoor positioning system for smartphones". In: *2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN).* 2015, pp. 1–7. DOI: `10.1109/IPIN.2015.7346761`.

[26] Mike Weed and Abby Foad. "Rapid Scoping Review of Evidence of Outdoor Transmission of COVID-19". In: *medRxiv* (2020). DOI: `10.1101/2020.09.04.20188417`. eprint: `https://www.medrxiv.org/content/early/2020/09/10/2020.09.04.20188417.1.full.pdf`. URL: `https://www.medrxiv.org/content/early/2020/09/10/2020.09.04.20188417.1`.

[27] WeHealth. *Covid Watch Arizona.* 2020. URL: `https://www.wehealth.org/arizona`.

[28] Chris Wymant et al. "The epidemiological impact of the NHS COVID-19 App". In: *Nature* (2021). (Pre-print from journal website). ISSN: 1476-4687. DOI: `10.1038/s41586-021-03606-z`. URL: `https://doi.org/10.1038/s41586-021-03606-z`.

[29] Sheng-Cheng Yeh et al. "Study on an Indoor Positioning System Using Earth's Magnetic Field". In: *IEEE Transactions on Instrumentation and Measurement* 69.3 (2020), pp. 865–872. DOI: `10.1109/TIM.2019.2905750`.

Appendix A

Most Relevant Features for Wi-Fi Fingerprint Data

This appendix lists the most relevant individual features — not the features that form the most relevant set — for training sets derived from different Wi-Fi fingerprint datasets, as identified by a feature of the pymRMR package. These rankings indicate which features' values have the highest correlation with close proximity.

The 8 features from a perfectly class-balanced training set derived from the suburban home dataset that are individually most relevant are:

1. Euclidean distance (No transformation)
2. Manhattan distance (No transformation)
3. Mean shared AP RSSI difference (No transformation)
4. Largest shared AP RSSI difference (No transformation)
5. Euclidean distance (Single-fingerprint least squares)
6. Standard deviation of shared AP RSSI pair difference comparison vector (No transformation)
7. Euclidean distance (Double-fingerprint least squares)
8. Population standard deviation of shared AP pair difference comparison vector (No transformation)

The 8 features from a perfectly class-balanced training set derived from the IPIN 2016 Tutorial dataset that are individually most relevant are:

1. RE3 (Double-fingerprint least squares)

2. RE3 (No transformation)
3. Redpin score (max, min) (Double-fingerprint least squares)
4. RE3 (Single-fingerprint least squares)
5. Shared AP count
6. Redpin score (min, max) (Double-fingerprint least squares)
7. RE3 (Single-fingerprint 50% least squares)
8. Redpin score (max, min) (No transformation)

The 8 features from the training set derived from fingerprints from the third floor of Cory Hall that are individually most relevant are:

1. RE3 (No transformation)
2. RE3 (Double-fingerprint least squares)
3. RE3 (Single-fingerprint least squares)
4. RE3 (Single-fingerprint 50% least squares)
5. Euclidean distance (No transformation)
6. Euclidean distance (Single-fingerprint least squares)
7. Largest element of shared AP pair difference comparison vector (No transformation)
8. Mean of shared AP RSSI difference vector (No transformation)

Appendix B

Most Relevant Features for Magnetometer Trace Data

This appendix lists the most relevant individual features — not the features that form the most relevant set — for training sets derived from different magnetometer trace datasets, as identified by a feature of the pymRMR package. These rankings indicate which features' values have the highest correlation with close proximity.

The 8 features from a perfectly class-balanced training set derived from the entire MagPIE dataset (i.e. containing segments from all 3 buildings) that are individually most relevant are:

1. Median of magnetometer reading X-component differences (Aligned segments)
2. Mean of magnetometer reading X-component differences (Aligned segments)
3. Median of magnetometer reading Y-component differences (Aligned segments)
4. Median of magnetometer reading X-component differences (Original segments)
5. Mean of magnetometer reading Y-component differences (Aligned segments)
6. Mean of magnetometer reading X-component differences (Original segments)
7. Median of magnetometer reading Y-component differences (Original segments)
8. Standard deviation of magnetometer reading X-component differences (Aligned segments)

The 8 features from a perfectly class-balanced training set derived from the entire set of Cory Hall data (i.e. containing segments from all 4 test devices) that are individually most relevant are:

1. Standard deviation of magnetometer reading Z-component differences (Aligned segments)
2. Standard deviation of magnetometer reading Y-component differences (Aligned segments)
3. Standard deviation of magnetometer reading X-component differences (Aligned segments)
4. Standard deviation of magnetometer reading Z-component differences (Original segments)
5. Standard deviation of magnetometer reading norm-component differences (Aligned segments)
6. Standard deviation of magnetometer reading Y-component differences (Original segments)
7. Standard deviation of magnetometer reading X-component differences (Original segments)
8. Mean of magnetometer reading Z-component differences (Aligned segments)