5-2015

Using Unrestricted Mobile Sensors to Infer Tapped and Traced User Inputs

Trang Duyen Nguyen

Recommended Citation

Nguyen, Trang Duyen, "Using Unrestricted Mobile Sensors to Infer Tapped and Traced User Inputs" (2015). *Theses and Dissertations*. 178.
https://csuepress.columbusstate.edu/theses_dissertations/178

USING UNRESTRICTED MOBILE SENSORS TO INFER TAPPED AND TRACED USER INPUTS

Trang Duyen Nguyen

Columbus State University

The D. Abbott Turner College of Business

The Graduate Program in Applied Computer Science

Using Unrestricted Mobile Sensors to Infer Tapped and Traced User Inputs

A Thesis in

Applied Computer Science

by

Trang Duyen Nguyen

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2015

I have submitted this thesis in partial fulfillment of the requirements for the degree of Master of Science

May 4, 2015 | [illegible]
Date | Trang Nguyen

We approve of the thesis of Trang Duyen Nguyen as presented here.

May 4, 2015 | [illegible]
Date | Radhouane Chouchane, Ph.D.
| Associate Professor,
| Thesis Advisor

May 4, 2015 | [illegible]
Date | Yesem Peker, Ph.D.
| Assistant Professor

April 30, 2015 | [illegible]
Date | Charles Turnitsa, Ph.D.
| Senior Research Scientist at
| Georgia Tech Research Institute

May 4, 2015 | [illegible]
Date | Wayne Summers, Ph.D.
| Distinguished Chairperson and
| Professor of Computer Science

ABSTRACT

As of January 2014, 58% of Americans over the age of 18 own a smart phone. Of these smart phones, Android devices provide some security by requiring that third-party application developers declare to users which components and features their applications will access. However, many of the real-time environmental sensors on devices are exempt from this requirement. We evaluate the possibility of exploiting this freedom to discretely use these sensors and expand on previous work by developing an application that can use the gyroscope and accelerometer to interpret what the user has written, even if trace input is used. Trace input is an option available on Samsung's default keyboard as well as in many popular third-party keyboard applications, such as Swype, SwiftKey, TouchPal, and GO Keyboard. "Tracing" an input involves the user dragging from the first letter of the intended word to the last letter without lifting his or her finger. The inclusion of trace input in a key logger application increases the amount of personal information that can be captured since users may choose to use the time saving trace-based input as opposed to the traditional tapping-based input. In this work, we attempt to interpret user input using accelerometer and gyroscope data given single letter tap and full word trace inputs.

Keywords: mobile malware; key logger; mobile security; spyware; motion sensors; Android

TABLE OF CONTENTS

List of Figures

Chapter 1

Introduction

According to McAfee, approximately half of all adults in the United States had personal data of some sort exposed in 2014, primarily through website breaches, point-of-sale theft, and falling victim to malicious software and social engineering [1]. Although there are some types of personal information that users may not be opposed to being collected (such as their application preferences), exposure of a user's passwords, credit card numbers, and other identifying information can have negative consequences for the user. Malware and in particular *spyware*, which is a type of malware that focuses on covertly gathering information, can be used to acquire these kinds of sensitive personal information and pose a potential risk for organizations that allow for the "bring your own device" policy [2].

The number of malware samples has been growing and their targets have been changing:

- there was a 112% increase from 2013 to 2014 raising the total number to over 5 million samples detected by McAfee as seen Figure 1.1 [1]
- there was a 75% increase in Android malware encounters in the US from 2013 to 2014 [3]
- the majority of new threat families, 275 out of the 277, found by F-Secure in the first quarter of 2014 specifically targeted Android devices [4]

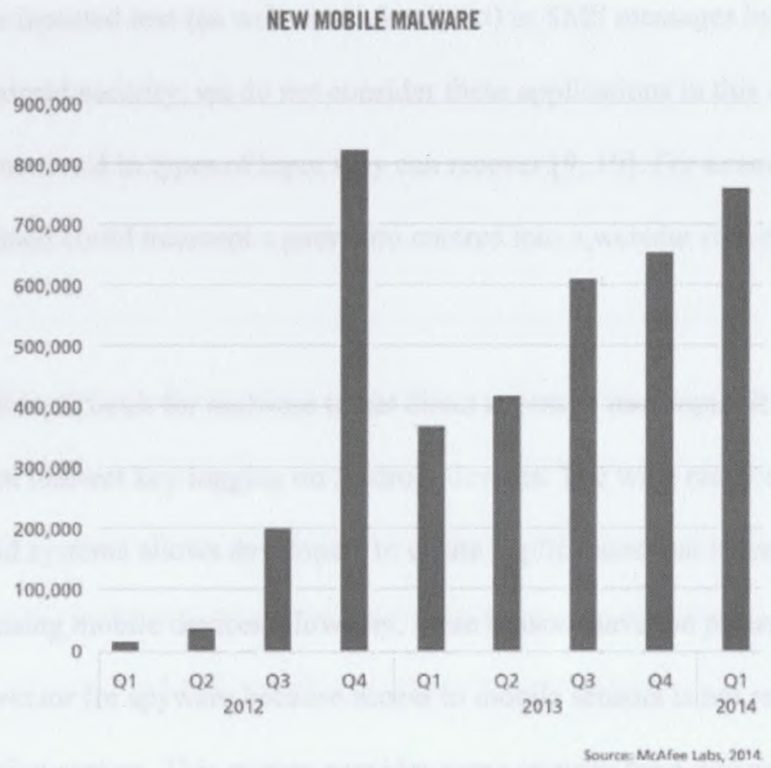

Figure 1.1: The number of new malware samples detected each quarter by McAfee [5].

1.1 Current State of Android Key Loggers

Surprisingly, while the portion of malware that perform activities to spy on users has increased from 12% to 28% between 2012 and 2013 and commercially marketed spyware for monitoring children's, a spouse's, or employees' activities are available, there has not been an Android key logger in the wild to our knowledge [6, 7]. A *key logger* is a type of spyware that secretly records the keys pressed by a user on a keyboard.

The lack of an Android specific key logger is potentially due to the strict restricted access allowed to applications on the Android system, which enables only the application view that is in focus to intercept keystrokes [8]. Although there are "*key logger-like*" malware applications, such as MisoSMS and Andr/FakeKRB-H, which can gain access to user inputted text (as well as received text) in SMS messages by circumventing Android security, we do not consider these applications in this study because they are restricted in types of input they can recover [9, 10]. For example, neither of the apps mentioned could intercept a password entered into a website viewed on a mobile browser.

Although it is difficult for malware to get direct access to user input, it may be possible to perform indirect key logging on Android devices. The wide range of available sensors on Android systems allows developers to create applications that enhance a user's experience when using mobile devices. However, these sensors have the potential to be used as an attack vector for spyware because access to mobile sensors is not restricted by Android's permission system. This system provides some security for a device by allowing only applications with explicit permission from the user to access certain restricted resources. Two examples of restricted resources are the device's camera and microphone; these restrictions help prevent applications from photographing or recording the user without his or her knowledge, since most would consider these actions to be a breach in personal privacy.

1.2 Mobile Sensors and Their Impact on Privacy

Sensor data has the potential to be used to collect information about users that may be considered private, albeit not as directly as a photograph or voice recording. There are multiple works that demonstrate this potential misuse of sensors.

- **Mobile accelerometer used to spy on PC keyboard:** Work by Marquardt, et al. indicated that the accelerometer readings taken from a smartphone placed on the same desk as a personal computer keyboard can be sufficient to recover the typed text with rates up to 80% [11].
- **Accelerometer readings used as smartphone fingerprint:** A recent study suggests that accelerometer data could be used to identify different individual devices in the same way that a fingerprint can be used to identify individuals [12]. This identification is based on the differences in accelerometer readings across devices for the same stimuli and could allow a user and their application usage to be tracked.

Whether the data comes from a nearby keyboard or a device's own screen, the ability to recover user entered text puts the user's passwords and other private information at risk. Because motion sensors on Android devices are not restricted and do not require the user's permission, the user could download and use applications that collect this data without ever knowing. This makes these unrestricted sensors a potential side channel that could be exploited in an Android key logging malware.

1.3 Our Contributions

1.1.1 Interpretation of Tap Input

Several studies have demonstrated the potential for sensor data to be used to interpret text that is entered using a mobile device (see Section 2.2). A common feature of these past works is that input was assumed to be entered one letter at a time by tapping

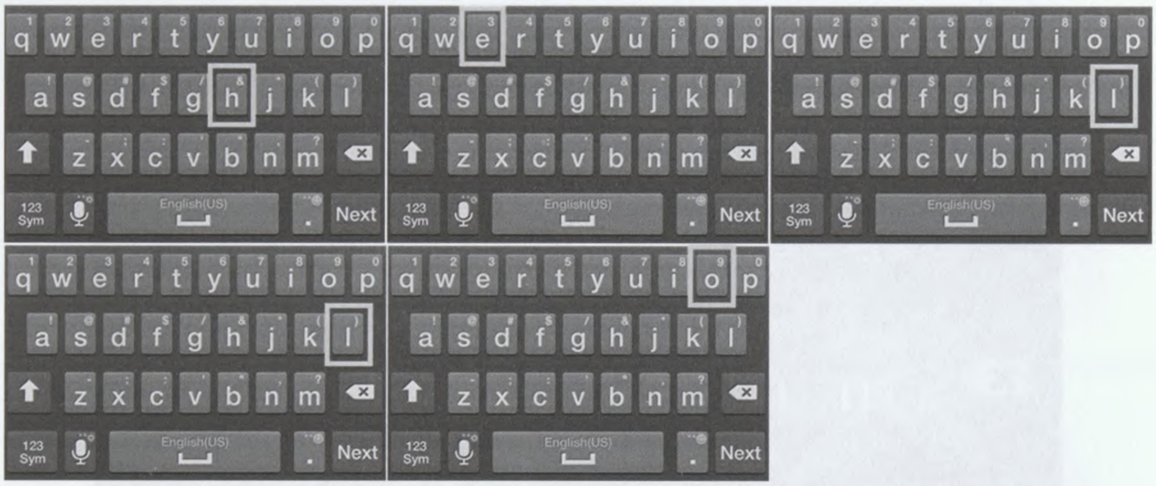

Figure 1.2: An example of a tapped input, in this case "hello". Each letter must be tapped individually and one-at-a-time.

on the screen of the device. An example of tap input is demonstrated in Figure 1.2. Because tap input has been a focus of previous work, the first phase of our work is to verify that our model is able to reasonably interpret tap inputs. However, the default software keyboards found on Samsung and Nexus devices as well as third-party keyboard applications, such as Swype, SwiftKey, TouchPal, and GO Keyboard, also have the option for users to enter text using a trace. This new type of input is the focus of the second phase of our work.

1.1.2 Interpretation of Trace Input

Trace input involves the user dragging from the first letter of the intended word to the last letter without lifting his or her finger, as shown in Figure 1.3. Unlike tap input which can only be a single letter, a trace input is always a word and must contain at least two letters. It is realistic to assume that while some users do only use the tap style of

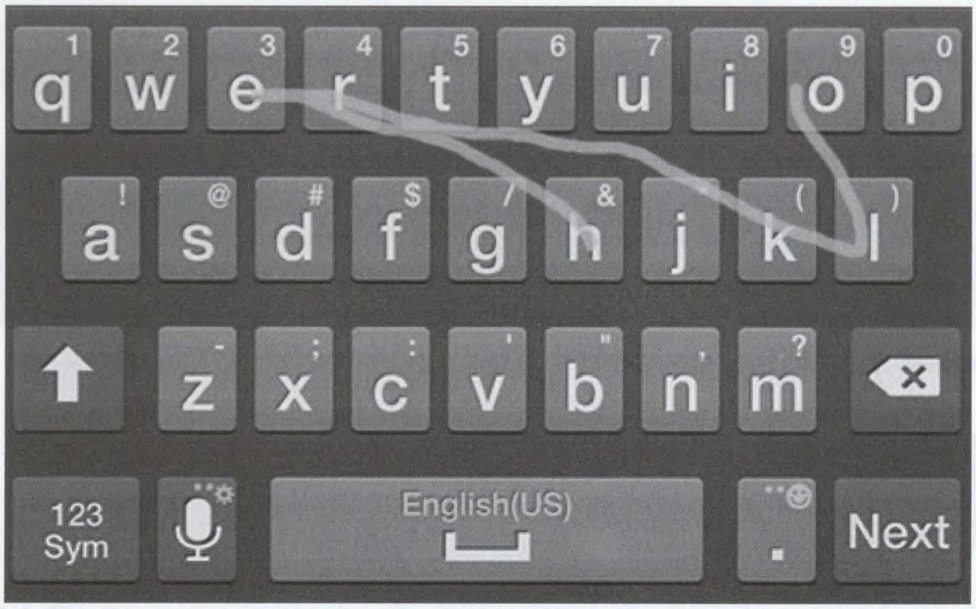

Figure 1.3: An example of a trace input, in this case "hello". Tracing an input requires that a user drag their finger to each letter in the word without lifting their finger. The blue path represents the finger movement needed to input this word.

inputting words into their devices, some users will also choose to use the trace style of input that has been made available by popular keyboard applications. For this reason, in the second phase of our work we evaluated the feasibility of interpreting trace input using a mobile device's motion sensors, which was not previously done in past works.

As can be seen in Figure 1.4, both tap and trace input (letters and words) are handled similarly throughout our work in that 1) the motion data must be collected while the user taps or traces, 2) characteristics (also known as features) about each tap or trace must be calculated and stored, and 3) these characteristics are used by a classification algorithm that has been trained on sample words and letters to recreate the original word or letter from the motion data.

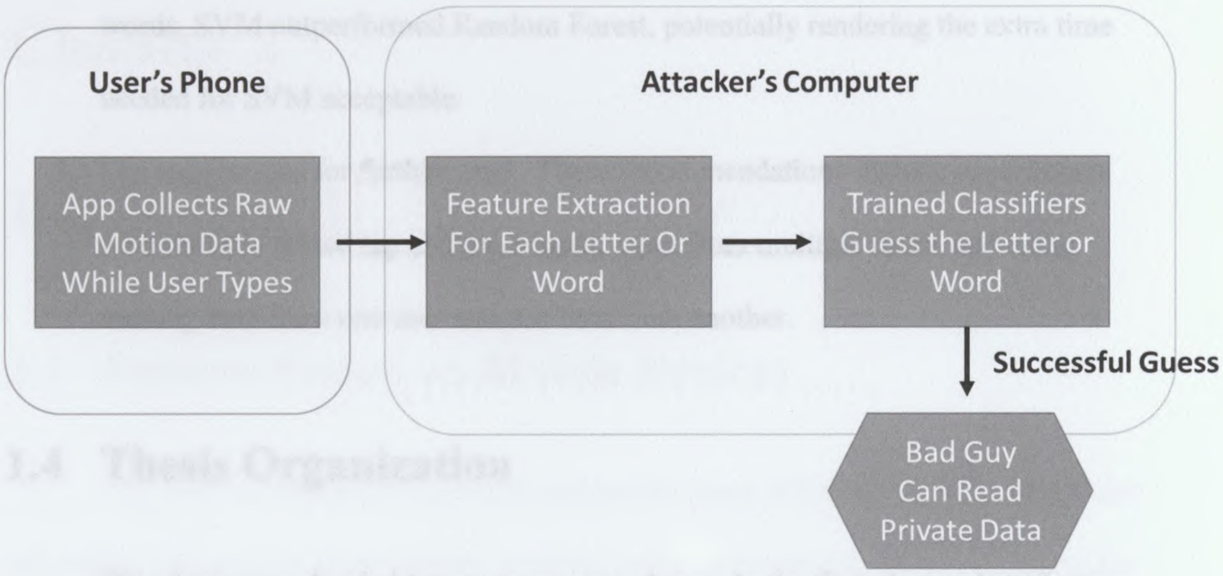
User's Phone
Attacker's Computer
App Collects Raw Motion Data While User Types
Feature Extraction For Each Letter Or Word
Trained Classifiers Guess the Letter or Word
Successful Guess
Bad Guy Can Read Private Data

Figure 1.4: This figure gives an overview of how tap and trace inputs are interpreted in our work. Motion sensor data collection occurs in the user's device and processing and interpretation takes place on the attacker's computer. The differences in how tap and trace inputs are handled occur in how features are calculated for letters versus words.

Our main contributions with this thesis are:

1. An evaluation of whether the use of motion sensor data collected from the accelerometer and gyroscope is sufficient to infer user entered text when trace input is used. Our results using a pangram and common words as input showed 6 to 47 times improvement over random guessing, depending on the number of words tested. This suggests that motion sensors do contribute to data leakage.
2. An evaluation of the impact of using different classification algorithms on the accuracy and speed of the text inference. Our results showed that Random Forest and Support Vector Machine (SVM) classifiers consistently provided the best predictions, although SVM also required the most time. At a greater number of

words, SVM outperformed Random Forest, potentially rendering the extra time needed for SVM acceptable.

3. Our suggestions for further work. These recommendations include experiments with realistic mixed tap and trace input, input from multiple users, and using training data from one user and test data from another.

1.4 Thesis Organization

This thesis was divided into two separate phases. In the first phase, classification was performed for individual letters entered using tap input. In the second phase, classification was performed for only trace inputted words. Chapter 2 contains the background for this thesis work, including an overview of the sensor available on Android devices and an overview of similar work published by other authors. Chapter 3 corresponds to the first phase and contains the details of our tap input only experimental setup and results. Chapter 4 describes the second phase of this thesis and is broken into two main trace input experiments. Chapter 5 concludes this work and our suggestions for further work. There is also an appendix with elementary information about the classification algorithms used.

Chapter 2

Background

2.1 Sensors Found on Mobile Devices

The Android framework provides support for many different types of sensors that can collect information about motion, position, or the environment, thereby allowing for the development of applications that can respond to the user and his or her environment. These sensors may be one of the following:

- **Hardware-based sensors:** physical components in the device that gather their data by direct measurement
- **Software-based sensors:** use one or more hardware-based sensors to generate their data [13]

Although there are many sensors available, the number and type of sensors present on a device depend on the device itself and the Android version of the device. For example, of the 83 devices analyzed by Teardown.com, 94% contained an accelerometer and 71% contained a gyroscope; other sensors, such as thermometers and barometers, are less common [13, 14]. Examples of mobile sensors and their type can be found in Table 2.1.

Sensors		
Hardware-based	*Hardware or Software-based*	*Software-based*
Accelerometers	Gravity detection	Orientation sensor*
Thermometers	Linear acceleration	
Gyroscopes	Rotational vector measurement	
Light sensors		
Magnetic field sensors		
Barometers		
Proximity sensors		
Humidity sensors		

Table 2.1: Examples of sensors that can be found on mobile phones. *Deprecated in API level 8

2.1.1 Motion Sensor Coordinate System

A 3-axis coordinate system, as shown for both smartphones and tablets in Figure 2.1, is used by motion and position sensors such as the accelerometer, gyroscope, and gravity sensor. This coordinate system does not change as the device is moved, even if the screen orientation of the device changes. With respect to this coordinate system, the *accelerometer* is the sensor that records the linear acceleration along each axis as $^{meters}/_{second^2}$, allowing it to register shaking or tilting of the device. The *gyroscope* is the sensor that measures the rotational speed around each axis of this coordinate system in $^{radians}/_{second}$, which translates to turning or spinning of the devices.

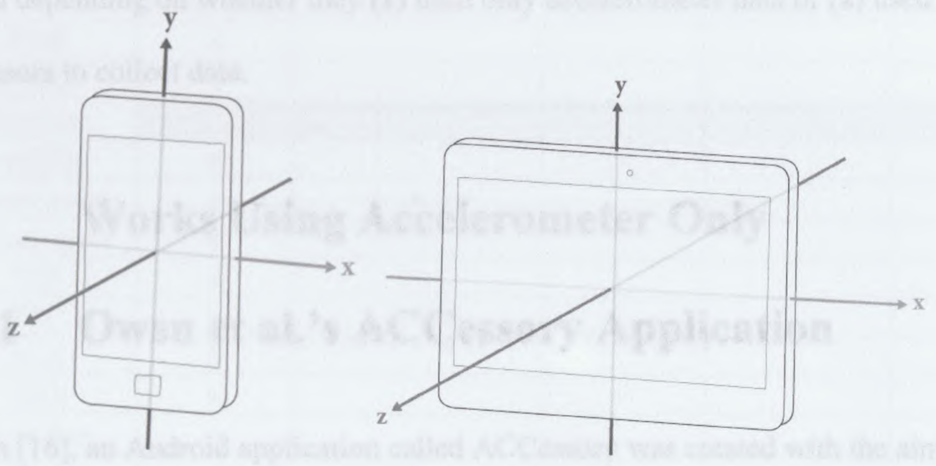

Figure 2.1: The 3-axis coordinate system used by Android device sensors. Source MathWorks [15].

2.1.2 Sensor Sensitivity and Sampling Rate

While the sensitivity and sampling ability of a sensor depends on the device in question, it is possible to use methods provided in the Android API to determine the minimum sampling delay available and to set the desired sampling rate to a specific range depending on the application's purpose. These include normal delay, delay suitable for gaming, delay for a user interface, and the least delay possible for the sensor [13].

2.2 Related Works

Previous works have shown that data collected using the unrestricted motion sensors in mobile devices can be used for inferring either a user's keystrokes or the location on the device's screen that was tapped. These works were able to achieve their goals using different classification algorithms and feature combinations. These differences are enumerated in the following two sections, in which the works are separated depending on whether they **(1)** used only accelerometer data or **(2)** used one or more sensors to collect data.

2.2.1 Works Using Accelerometer Only

2.2.1.1 Owsu et al.'s ACCessory Application

In [16], an Android application called ACCessory was created with the aim to infer which area of the screen was tapped as well as the characters inputted using taps. This work used the random forest algorithm for classification, which was trained using 46 features. The features used were extracted from the three axis component and the magnitude of acceleration, and can be seen in Table 2.2.

The model presented was able to correctly infer 6 out of 99 passwords consisting of six characters each in a median of 4.5 trials.

2.2.1.2 Xu, Bai, and Zhu's TapLogger Application

The application known as TapLogger was developed by another set of researchers [17]. In this work, the goal was to evaluate if sensor data provided enough information to distinguish between taps on the number pad, which would allow for such activities as cracking a user's lock screen pin or determining credit card numbers entered onto the device. The 2-norm acceleration vector from the accelerometer data was used to describe the tap pattern. The orientation sensor (now depreciated) was also used in their work to infer the position of the tap. This model was able to correctly predict screen lock

		Owsu [16]	Cai and Chen [18]	Miluzzo [19]	Al-Haiqi [20]	Our Work
Features	RMS	●				
	RMSE	●				
	Min/Max	●		●	●	●
	ΔASbS	●				
	# local peaks	●	●			
	# local crests	●				
	TTP	●				
	TTC	●				
	RCR	●				
	SMA	●				
	Total time	●				
	# samples	●				
	Segment length		●			
	ΔPeak Time		●			
	Peak interval		●			
	Attenuation rate		●			
	Vertex angles		●			
	CPI			●		
	Moments			●		
	Skewness			●	●	●
	Kurtosis			●		●
	1-norm			●		
	Infinity norm			●		
	Forbenius norm			●		
	FFT			●		
	Mean				●	●
	Median				●	●
	StD				●	

Table 2.2: Features used by previous works. These features include root mean square value (RMS), root mean square error (RMSE), average sample-by-sample change (ΔASbS), average time from a sample to a peak (TTP), average time from a sample to a crest (TTC), RMS cress rate (RCR), cubic spline interpolation (CPI), Fast Fourier Transform (FFT), and standard deviation (StD). Note that some of these features are extracted for each axis of multiple sensors, so the total number of features used in a work does not equal the number of features marked in this table. For example, Miluzzo et al. used a total of 273 features in their work.

passwords with a four character length with an average coverage rate of 40% and eight character length passwords with a rate of 45% [17].

2.2.2 Works Using Multiple Sensors

2.2.2.1 Cai and Chen's Application

In Cai and Chen's paper, the impacts of different classification algorithms and features, device types, key sets (such as alphabet-only keyboards compared to number-only keyboards), the device's screen orientation, and the keyboard layout on the performance tap-input inference based off sensor data were evaluated [18]. This work's pre-processing on the raw sensor data included de-jittering, low-pass filtering, calibration (such as removing the influence of gravity), and segmentation (separate each of the keystrokes). The authors also chose not to consider the z-axis component for either the accelerometer or the gyroscope. They also state that for motion data, magnitude is a poor feature. They instead chose to use the six features shown in Figure 2.2.

This work also included another feature that the authors calculated as shown in Equation 1 below.

$$\mathrm{hi} = \arctan\left(\frac{\mathrm{viy}}{\mathrm{vix}}\right) \times 180/\pi \qquad (1)$$

The two classification algorithms compared in this work were

1) Dynamic Time Warping
2) Support Vector Machines (SVM)

The authors concluded that both algorithms performed similarly and were effective for inferring user's tap-based input, although accuracy was affected by keyboard and device differences (for example, the accuracy increases when a device's screen is in the landscape orientation). The authors also noted that gyroscope data provided more accurate inference than accelerometer data [18].

2.2.2.2 Miluzzo et al.'s TapPrints Application

The framework presented in [19], known as TapPrints, uses a combination of accelerometer and gyroscope data to infer user input for devices with different operating systems (iOS and Android) and for both smartphones and tablets. In this work, 273 features were extracted from the time and frequency domains of the raw data. The authors used cubic spline interpolation to ensure that the number of sensor readings used for each tap were the same before feature extraction. The time domain features they chose to use were divided into column features (which used components of each sensor axis) and matrix features (which used the correlation between the three-axis sensor vectors).

The column features included cubic spline interpolation, moments, the minimum and maximum values, skewness, and kurtosis; matrix features included the 1-norm, the Infinity norm, and the Frobenius norm. Other features were also extracted from processed data. For the frequency domain features, they performed a Fast Fourier Transform (FFT) on each of the sensor axis components and computed features from the power spectrum of the FFT values. These features are summarized in Table 2.2.

For classification, TapPrints used an ensemble classification approach in an attempt to increase the accuracy and robustness of their input predictions. The authors chose to use the following types of multi-class classifiers:

1) k-nearest neighbor
2) multinomial logistic regressions
3) Support Vector Machines (both linear and with radial basis function kernels)
4) random forests
5) bagged decision trees.

This model was able to achieve an average of above 50% accuracy for inferring sequentially tapped-inputted letters in landscape orientation and 27% when inferring a pangram while in portrait orientation [19].

2.2.2.3 Al-Haiqi et al.'s Application

In a work comparing the effectiveness of different sensors and sensor combinations by Al-Haiqi et al., 18 features per sensor from the time domain were chosen [20]. These features can be seen in Table 2.2.

The authors state that no set of features used in previous research for keystroke classification appears to clearly outperform the others used. For classification, the authors of this work determined that the Bagging classifier used with a Functional Trees based model performed best for their dataset. The taps used were estimated to be approximately 80ms long, which, with the sampling rates used, resulted in five sensor samples for each tap. The authors concluded that the gyroscope data was more effective for tapped key inference than data collected using the linear accelerometer, the rotational vector sensor, or a combination of the accelerometer and the magnetic field sensor [20].

Chapter 3

Interpretation of Tap Input

In this chapter, we describe our method for inferring user text from accelerometer and gyroscope readings taken while the user taps each letter on a device. We describe our experimental approach and provide the details of our data collection, feature selection, feature extraction, and classification algorithm selection. We end this chapter with an evaluation of our results.

3.1 Our Verification Experiment

3.1.1 Motivation

The aim of the first phase of this thesis was to confirm that our model was capable of producing tap-input inference accuracy similar to those seen in previous works before proceeding with the novel inference of trace-input. We also wanted to determine if there was a significant difference in performance when using other classifications algorithms that previously were not compared, which would allow us to potentially optimize text inference.

3.1.2 Approach

The first step of our approach was to create an Android application that could be used to collect gyroscope and accelerometer data on a device while a user entered text.

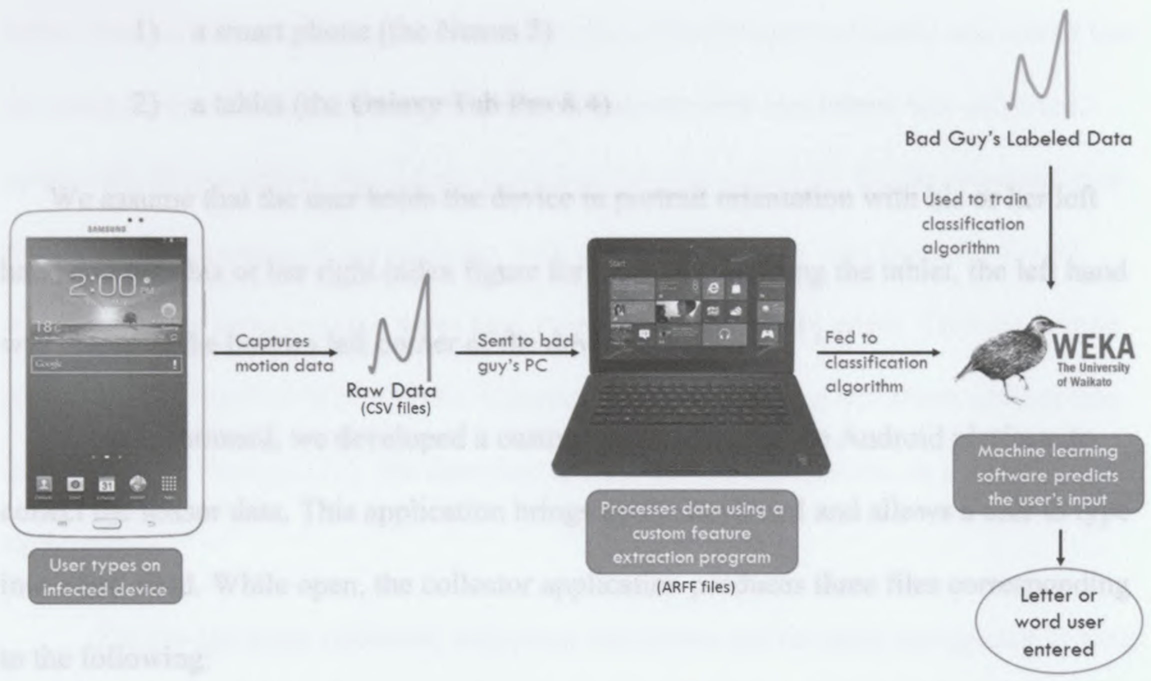
Bad Guy's Labeled Data
Used to train classification algorithm
Captures motion data
Raw Data (CSV files)
Sent to bad guy's PC
Fed to classification algorithm
WEKA The University of Waikato
Machine learning software predicts the user's input
Processes data using a custom feature extraction program
(ARFF files)
User types on infected device
Letter or word user entered

Figure 3.1: The overview of our approach. Data flows from the "infected" mobile device to a PC, where processing and classification occurs.

Our application was responsible for outputting the comma-separated values (CSV) files containing the sensor data as well as the end time of each key tap. These files were then transferred from the device to a computer where all processing, learning, and prediction were performed. We then developed a program to extract our selected features from these data CSV files and output feature CSV files. These feature CSV files could then be converted to attribute-relation file format (ARFF) files. The ARFF files were then used with the machine learning software Weka, which we used for classification learning and prediction. This approach is also presented visually in Figure 3.1.

3.1.2.1 Data Collection

Two devices were initially used in this experiment:

1) a smart phone (the Nexus 5)
2) a tablet (the Galaxy Tab Pro 8.4)

We assume that the user holds the device in portrait orientation with his or her left hand and uses his or her right index figure for input. When using the tablet, the left hand was placed at the bottom left corner of the device.

As mentioned, we developed a custom application for the Android platform to collect the sensor data. This application brings up the keyboard and allows a user to type into a text field. While open, the collector application produces three files corresponding to the following:

- input completion timing,
- the gyroscope readings
- the accelerometer readings

For the tap's completion time, we used a "TextWatcher" and a "TextChangedListener". Both the gyroscope and the accelerometer were set to use the lowest delay possible ("SENSOR_DELAY_FASTEST"), which for both sensors was approximately 5 microseconds. This collection application requires "WRITE_EXTERNAL_STORAGE" permission because the created files are stored locally in a user accessible folder. This is for our convenience and would likely not be present in an application aiming for attack. After data collection process was complete, we retrieved the data files for off-line processing.

Raw accelerometer and gyroscope data was collected on the tablet device for each letter of the alphabet entered using tap-style input. This was done with 50 repetitions per letter. Another 50 repetitions per letter were also collected approximately one month later for comparison. For the smart phone, data for only the first two letters was collected, again with 50 repetitions. We did not complete the data collection for the smartphone at this time point because of the difficulty with retrieving the data files from the device, which made the process much more time consuming than on the tablet. This is a known issue with the Android MTP (Media Transfer Protocol) and does not affect devices that use USB Mass Storage [21]. We therefore excluded the Nexus 5 from all further experiments.

For the tap-input collected, individual tap events can be easily recognized in both the raw accelerometer and gyroscope data, as shown in Figure 3.2 and Figure 3.3. This is particularly true in the z-axis of the accelerometer sensor readings and the x-axis and y-axis sensor readings of the gyroscope. Note that the end time of each tap in these figures is denoted by a vertical line.

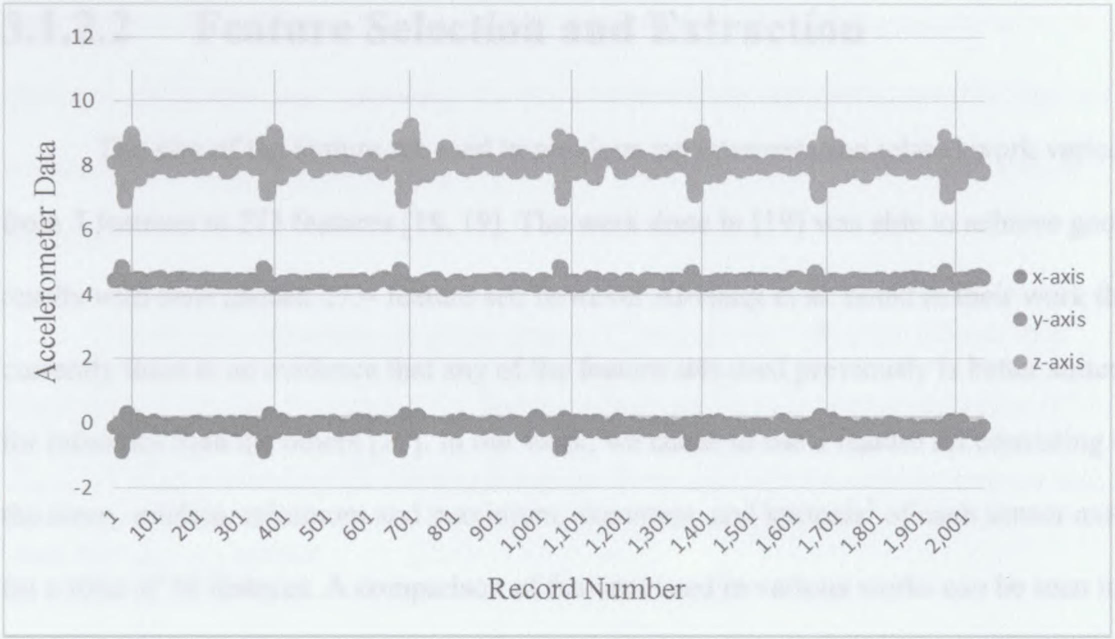

Figure 3.2: Raw accelerometer data showing seven tap-input letters. The end of each tap is denoted by a vertical bar.

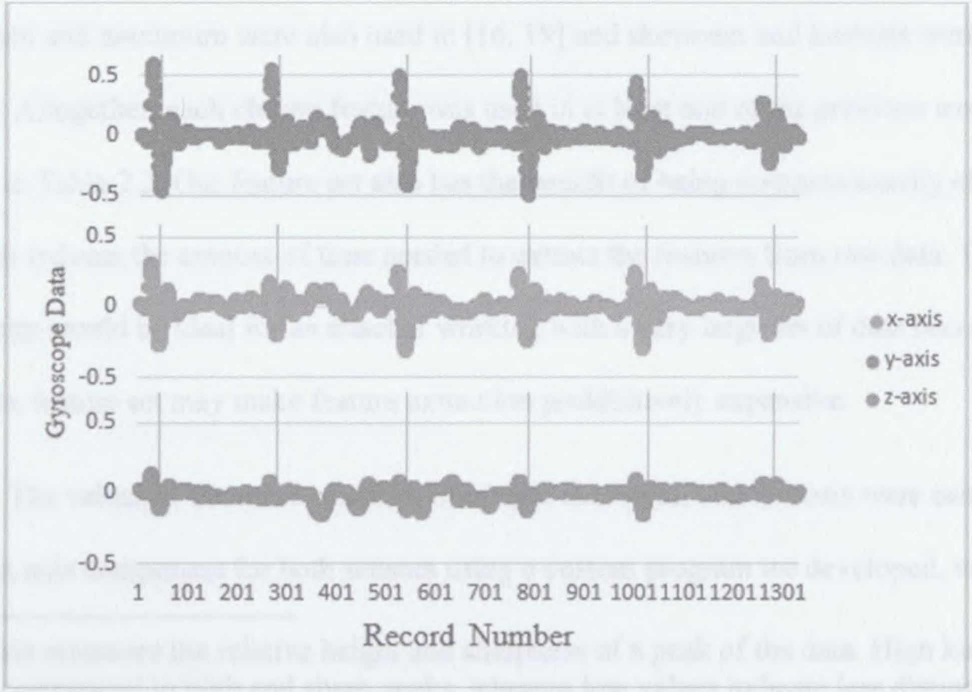

Figure 3.3: Raw gyroscope data showing six tap-input letters. The end of each tap is denoted by a vertical bar.

3.1.2.2 Feature Selection and Extraction

The size of the feature set used in previous tap interpretation related work varied from 7 features to 273 features [18, 19]. The work done in [19] was able to achieve good results with their chosen 273- feature set; however Al-Haiqi et al. noted in their work that currently there is no evidence that any of the feature sets used previously is better suited for inference than the others [20]. In our work, we chose to use a feature set consisting of the mean, median, minimum and maximum, skewness, and kurtosis[1] of each sensor axis for a total of 36 features. A comparison of features used in various works can be seen in Chapter 2 in Table 2.2.

The feature set used in our study is very similar to the one used in [20], with the exception that we did not use the standard deviation and they did not use kurtosis. The minimum and maximum were also used in [16, 19] and skewness and kurtosis were used in [19]. Altogether, each chosen feature was used in at least one of the previous works, as shown in Table 2.2. Our feature set also has the benefit of being computationally efficient in that it reduces the amount of time needed to extract the features from raw data. This simplicity would be ideal for an attacker working with a very large set of data because a complex feature set may make feature extraction prohibitively expensive.

The values of the mean, median, min/max, skewness, and kurtosis were calculated for each axis component for both sensors using a custom program we developed, which

[1] Kurtosis measures the relative height and sharpness of a peak of the data. High kurtosis values correspond to high and sharp peaks, whereas low values indicate less distinctive peaks. The skewness measurement indicates whether the data is skewed to the left or to the right (the asymmetry). Positive skewness means the data is skewed to the right, negative skewness means it is skewed left, and a skewness value of zero indicates a symmetrical dataset [33].

used the DescriptiveStatistics API available from Apache [22]. This program takes as input two CSV files, one containing the raw accelerometer and gyroscope data and one containing the input end times, and outputs one CSV file containing all the extracted features.

Our feature extraction program assumed that each tap lasted for 200ms; this resulted in 42 records being used for feature extraction for each tap for all letters [17]. The CSV files containing the extracted features were also labeled with the corresponding letter the tap represented.

3.1.2.3 Learning and Classification

Before our labeled feature data could be used for classification, the CSV files were first converted into ARFF files using an online converter [23]. All classification learning and prediction was then performed using Weka version 3.6.10 [24].

We elected to compare the performance with regard to classification accuracy and the time required to build the classification model of the following classification algorithms:

- *k*-Nearest Neighbor (*k*-NN)
- Support Vector Machine (SVM)
- Random Forest

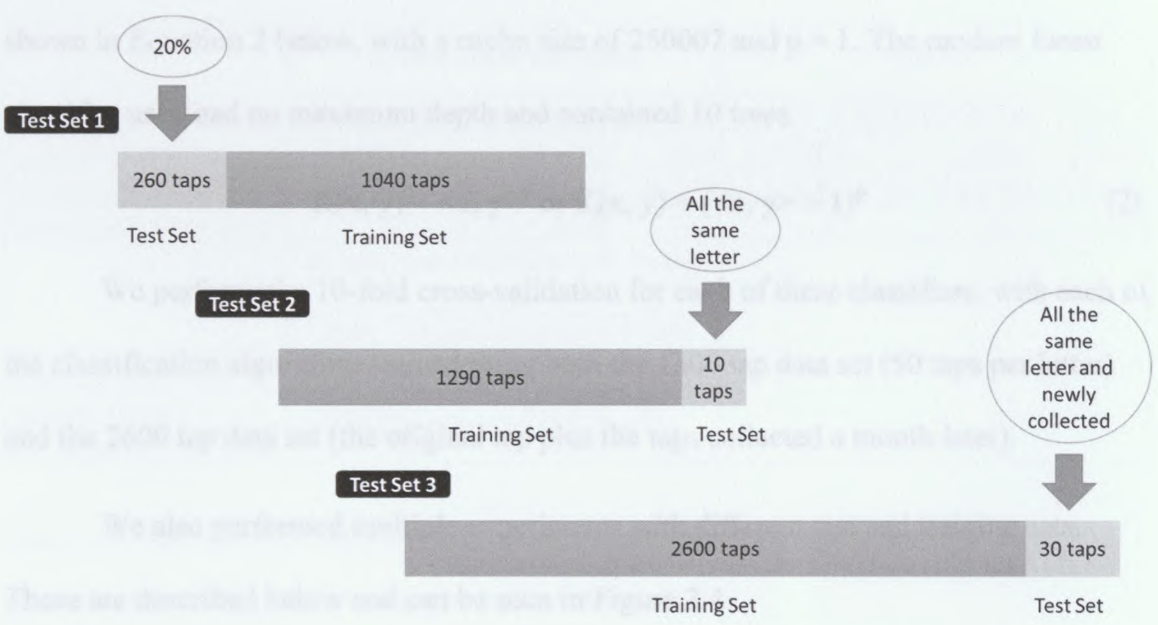

Figure 3.4: The breakdowns of the three different test and training sets used. Note that the first two sets used only the original 1300 taps collected.

These classifiers were chosen because random forest was used in [16], SVM was used in [18], and all three were part of a large ensemble classifier in [19], but their performance was not previously compared. The *k*-NN algorithm was chosen for its relative simplicity, which allowed us to compare a lower complexity algorithm (*k*-NN) with a higher complexity algorithm (SVM).

The *k*-NN and SVM algorithms correspond to the instance-based *k* (IBK) and sequential minimal optimization (SMO) classifiers in Weka, respectively. We used the default parameters available in Weka for all classification algorithms through our experiments. We made the decision not to optimize for two reasons: 1) to aid in repeatability and 2) because optimization for one user's input style may not be generalizable to other users. For *k*-NN, the default settings correspond to $k = 1$, with no distance weighting, and Euclidean distance function. The SVM had a polynomial kernel shown in Equation 2 below, with a cache size of 250007 and p = 1. The random forest classifier used had no maximum depth and contained 10 trees.

$$K(x, y) = \langle x, y\rangle^p \text{ or } K(x, y) = (\langle x, y\rangle + 1)^p \quad (2)$$

We performed a 10-fold cross-validation for each of these classifiers, with each of the classification algorithms trained using both the 1300 tap data set (50 taps per letter) and the 2600 tap data set (the original tap plus the taps collected a month later).

We also performed multiple experiments with different test and training sets. These are described below and can be seen in Figure 3.4.

- **A 20/80 split test and training set:** We tested the classification algorithms by splitting our 1300 tap data set into a training set and a testing set. The testing set consisted of 20% (260 taps) from the original set, and the training set contained the remaining instances (1060 taps).
- **A 10 instance test set per letter:** We split individual letters into testing/training sets, such that the test set contained 10 instances of the letter of interest and the training set contained 1290 taps (the original data set minus the test set).
- **A 30 new instance test set per letter with the doubled training set:** We collected 30 new taps of individual letters and used them as a test set for the classifiers and the doubled 2600 tap data set as the training set.

	10-Fold Cross-Validation			Test Set		
	1-NN	Random Forest	SVM	1-NN	Random Forest	SVM
Correctly Classified	74.69%	83.23%	82%	71.92%	78.85%	80.38%
Relative Absolute Error	27.92%	37.10%	96.10%	30.91%	38.74%	96.06%
Model Build Time (sec)	0	0.29	1.61	0	0.25	1.72
	10-Fold Cross-Validation w/ Double Data					
	1-NN	Random Forest	SVM			
Correctly Classified	77.92%	86.50%	62.77%			
Relative Absolute Error	23.81%	32.72%	96.38%			
Model Build Time (sec)	0	0.71	4.34			

Figure 3.5: Accuracies and time requirements of the three classification algorithms we compared (*k*-nearest neighbor, support vector machine, and random forest).

3.1.3 Results and Discussion

For the 10-fold cross-validation experiments with the 1300 tap data set, the classification accuracies in terms of the percentage of correctly classified instances achieved using the SVM and the random forest were very similar, with the 1-NN being less accurate than both. The k-NN classifier was approximately 8% less accurate than the other two classifiers; however, it showed the smallest relative absolute error of the three classifiers. Doubling the dataset size resulted in only a slight increase in accuracy for *k*-NN and random forest classifiers. However, the accuracy of the SVM classifier decreased by almost 20%. While these results could suggest that collecting a large number of labelled tap inputs recorded from a single user may be unnecessary, this may also be an artifact of combining data collected a month apart, indicating a time sensitivity issue.

When we split the data set into an 80/20 percent split training and test set, both SVM and random forest classifiers again performed better than k-NN, although in this

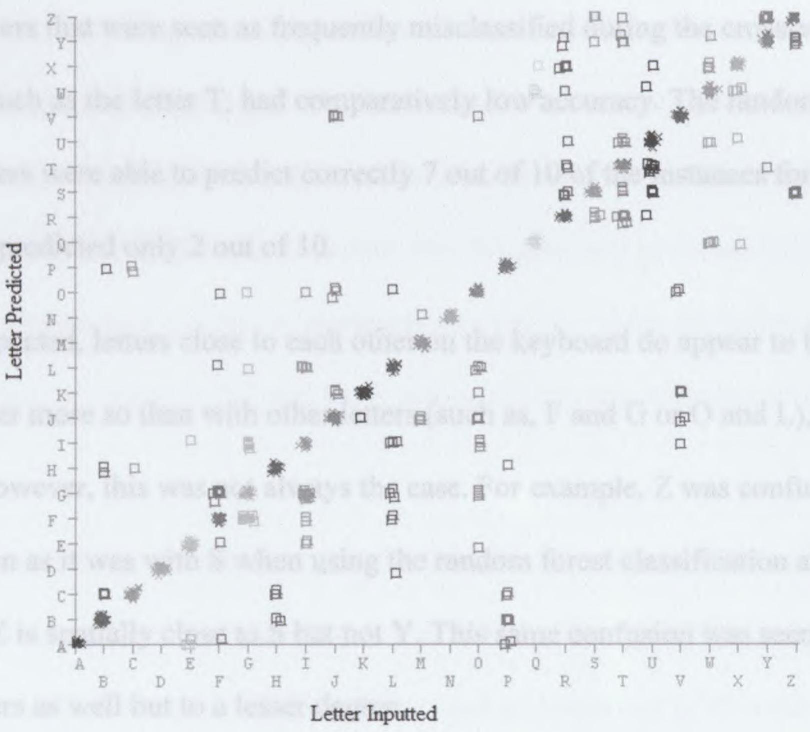

Figure 3.6: Letter confusion seen during the 10-fold cross-validation of the random forest classifier.

evaluation SVM outperformed the random forest classifier by a small amount. The k-NN classifier again showed the smallest relative absolute error of the three. These accuracies, as well as the relative absolute error and time each classifier took to build, can be seen in Figure 3.5.

Accuracy when the test set consisted of 10 instances of one letter and the training set consisted of the 1290 remaining taps varied depending on the letter. For example, for the letter A both the SVM and random forest classifiers were able to correctly predict all 10 instances and the k-NN predicted 9 out of 10 correctly. As can be seen in the confusion matrix (which shows both the correct and incorrect predictions made by a classifier) of the random forest classifier (Figure 3.6), A was also not misclassified as any other letter during the cross-validation experiment so this result was not unexpected.

Similarly, letters that were seen as frequently misclassified during the cross-validation experiment, such as the letter T, had comparatively low accuracy. The random forest and k-NN classifiers were able to predict correctly 7 out of 10 of the instances for the letter T, but the SVM predicted only 2 out of 10.

As expected, letters close to each other on the keyboard do appear to be confused with each other more so than with other letters (such as, F and G or O and L), as seen in Figure 3.4. However, this was not always the case. For example, Z was confused with Y almost as often as it was with S when using the random forest classification algorithm, even though Z is spatially close to S but not Y. This same confusion was seen using the other classifiers as well but to a lesser degree.

While the classifiers all performed well in our other tests, when we introduced newly collected tap data (30 instances) as the test set and used the doubled data set for training, all of the classifiers achieved around 65% classification accuracy (66.7% by SVM and random forest and 63.3% by k-NN; data not shown). As previously mentioned, we believe this could be due to differences in the feature values over time because the taps used to double the data set were collected closer to time to the new test set taps than the original data set taps were. We have not yet found the cause of this difference, though a change in the feature set may allow for stored labelled tap data to be used for more long term interpretation.

Aside from the time-sensitive nature of our current feature set, all three classifiers were able to infer the letters represented by the tap input with a much greater accuracy than the 1⁄26 (3.85%) accuracy expected for randomly guessing English letters. For the results using the 1300 tap data set, the accuracy of tap classification was 18-20 times better than random guessing. Our results are similar to those found in the literature; for example, in [19] a mean accuracy of 65.11% was achieved when taps were collected on a tablet in landscape orientation. As expected, our results were better than those that were obtained using only the gyroscope, with only 30-33% accuracy achieved in [18] for tapped letters collected on a tablet in landscape orientation. Although higher accuracies of over 90% were achieved in [8], this work evaluated only the inference of tapped numbers.

In terms of the performance of the classification algorithms, SVM and random forest consistently performed slightly better than k-NN, although k-NN always showed the least relative absolute error. While both the random forest and k-NN built quickly, the SVM classifier was comparatively sluggish, taking over five times longer to build than the random forest in all tests. This difference in computation time lead us to conclude that random forest and k-NN are better suited for this type of inference.

Chapter 4

Interpretation of Trace Input

In this chapter, we describe our method for inferring user text from accelerometer and gyroscope readings taken while the user traces whole words on a device. We performed two experiments for trace input: one using pangrams as input and one using common English words. In this chapter we explain our approach and detail our data collection, feature selection and extraction, classification algorithm selection and learning, and evaluate our results.

4.1 Pangram Experiment

4.1.1 Motivation

We chose to perform an experiment using a pangram as the input to ensure that all letters of the alphabet were included. Not all letters were inferred with the same accuracy during our tap experiment, so we wanted to use words that covered all of the keyboard locations.

4.1.2 Approach

The same approach was used for interpreting trace-input as was used for tap-input (shown in Figure 3.1), with a few modifications that are enumerated in Section 4.1.2.1.

We used our application to output CSV files containing the sensor data and the end time of each word trace. These files were transferred from the device to a computer, where all processing, learning, and prediction was performed. Our selected features were computed from these data CSV files using our custom feature extraction application and contained in the outputted feature CSV files. These feature CSV files could then be converted to ARFF files and used with the classification algorithms to infer the inputted words.

4.1.2.1 Data Collection

Trace input data was collected using the same mobile device, screen orientation, and custom Android application and by the same user as the tap input data. Raw accelerometer and gyroscope data was collected for the trace of each word in the pangram "*The quick brown fox jumps over the lazy dog*." with 50 repetitions per word. Because our collection application used a "TextWatcher" and a "TextChangedListener" to record the end time of each trace, time entries for the space added automatically by the Samsung after a word is traced were also in the outputted CSV file and had to be removed manually before feature extraction.

4.1.2.2 Feature Selection and Extraction

The feature set we used to infer the collected trace data were the same as those used for tap input inference: the mean, median, minimum and maximum, skewness, and kurtosis values of each sensor axis (also shown in Figure 2.2). We made this decision to see if the same feature set could be used to interpret both tap and trace input.

We modified our feature extraction program with the assumption that each word took 1 second to trace. This choice came from the original claims of the Swype keyboard, the first that used trace input, which stated that a user could input 55 words per minute [25]. Because the time a user begins the trace is not available to use, this second is collected starting from the end of the trace and goes backwards. As a result, 210 records are used to compute the features of every traced word regardless of whether the trace was longer or shorter than 1 second. As with the tap data, the feature-containing CSV files were converted to ARFF files for classification and learning.

4.1.2.3 Learning and Classification

The SVM, k-NN, and random forest classification algorithms were used to determine their potential for inferring words inputted using a trace. Their inference performance and required computation times were also compared.

A 10-fold cross-validation was performed with each classifier trained using the full 400 trace data set. We also evaluated the classifiers by splitting the data set into a test set containing 20% of the original data set (40 traces) and a training set (360 traces).

4.1.3 Results and Discussion

Classifiers using any of the three classification algorithms were able to infer the inputted words with better accuracy than randomly guessing one of the eight words (12.5%). The SVM had the highest percentage of correctly identified words (89.75%), followed by the random forest (84.75%), and then the k-NN classifier (79%) for the 10-

	10-Fold Cross-Validation		
	k-NN	Random Forest	SVM
Correctly Classified	79.00%	84.75%	89.75%
Relative Absolute Error	25.65%	32.86%	86.29%
Model Build Time (sec)	0	0.22	0.68
	Test Set		
	k-NN	Random Forest	SVM
Correctly Classified	75.00%	90.00%	85.00%
Relative Absolute Error	30.05%	33.64%	86.13%
Model Build Time (sec)	0	0.23	0.75

Figure 4.1: Accuracies and time requirements of the three classification algorithms (k-nearest neighbor, support vector machine, and random forest) for trace input.

fold cross-validation. As in the results for the tap input, the smallest relative absolute error was seen with the k-NN classifier.

When compared using the split test and training sets, the random forest performed best in terms of accuracy (90%), followed by the SVM (85%) and the k-NN (75%) classifiers. The random forest and k-NN classifiers showed similar relative absolute error; both were much lower than SVM. As seen in the tap interpretation experiments, SVM took several times longer to build than the random forest classifier. These results and those from the cross-validation are summarized in Figure 4.1. Taking into account both its inference accuracy and build time, the random forest algorithm appears to be the best choice out of the three for trace-inputted word interpretation. However, we found it interesting that all three algorithms were able to identify the traces. It should be noted, of

	k-NN	SVM	Random Forest
Word	**Confused with**	**Confused with**	**Confused with**
quick	over/fox/dog	dog	lazy
the	jumps	jumps	jumps
over	quick	lazy	quick
lazy	fox	brown	fox
jumps	the	the	the
fox	dog	dog	lazy
dog	fox	over	over
brown	lazy	lazy	lazy

Figure 4.2: Letter confusion seen during the 10-fold cross-validation of all three classifiers for trace input.

course, that while the inference accuracy is high, these results demonstrate only 6-7 times improvement over random guessing because of the low number of words tested.

When comparing the confusion matrices for the cross-validation trials, it can be seen that misclassification is not as easy to interpret as with the tap inputs. The most frequently confused word for each trace with each classifier is shown in Figure 4.2. Note that with k-NN, "quick" was confused with three other words with the same frequency. Length of the word does not appear to contribute to the confusion, which is expected because we treated each trace as the same length when extracting the features. Interestingly, only "jumps" and "the" were misclassified with each other in both directions; for example, "brown" was confused with "lazy" the most frequently with all classifiers but "lazy" was not most often confused with "brown" in two of the three classifiers. While we would expect the misclassification to be due to similarities in the motion of the trace pattern of each word, it is difficult to visualize these similarities. For example, although both “the” and “jumps” end with the trace going to the left of the keyboard, “the” is more centralized on the keyboard and the distance between the keys is comparatively small. In contrast, “jumps” starts at the far right of the keyboard and ends going down and left all the way from “P” to “S.” Unlike with tap inference where a misclassified letter is likely a letter spatially close on the keyboard, misclassification of trace input appears to be much less intuitive.

4.2 Common Word Experiment

4.2.1 Motivation and Approach

The aim of this experiment was to determine if common, short words could be inferred using sensor data when traced by a user. Because these words are considered very common, using them as input allows us to determine how effective trace inference would be for everyday text input. We also included a greater variety words in our data set, again to be more representative of a user’s realistic usage. For the approach, refer back to Section 4.1.2.

4.2.2.1 Data Collection

Trace input data was collected using the same mobile device, screen orientation, and custom Android application and by the same user as the tap input data. Raw accelerometer and gyroscope data was collected for the trace of each of the 110 most

1	the	31	or	59	know	87	work
2	be	32	an	60	take	88	first
3	to	33	will	61	people	89	well
4	of	34	my	62	into	90	way
5	and	35	one	63	year	91	even
7	in	36	all	64	your	92	new
8	that	37	would	65	good	93	want
9	have	38	there	66	some	94	because
11	it	39	their	67	could	95	any
12	for	40	what	68	them	96	these
13	not	41	so	69	see	97	give
14	on	42	up	70	other	98	day
15	with	43	out	71	than	99	most
16	he	44	if	72	then	100	us
17	as	45	about	73	now	*2*	*person*
18	you	46	who	74	look	*6*	*thing*
19	do	47	get	75	only	*7*	*man*
20	at	48	which	76	come	*8*	*world*
21	this	49	go	77	its	*9*	*life*
22	but	50	me	78	over	*10*	*hand*
23	his	51	when	79	think	*11*	*part*
24	by	52	make	80	also	*12*	*child*
25	from	53	can	81	back	*13*	*eye*
26	they	54	like	82	after	*14*	*woman*
27	we	55	time	83	use	*15*	*place*
28	say	56	no	84	two	*17*	*week*
29	her	57	just	85	how		
30	she	58	him	86	our		

Figure 4.3: The 110 words used as input for the "common words" trace experiment. The number indicates how common the word is (lower is more common), according to the Oxford English Dictionary [26]. Italicized words are from the list of most common content words.

common words, according to the Oxford English Dictionaries and shown in Figure 4.3, with 50 repetitions per word [26]. Note that we excluded single letter words and included some of the most common content words for a total of 110 words. Because our collection application used a “TextWatcher” and a “TextChangedListener” to record the end time of each trace, time entries for the space added automatically by the Samsung after a word is traced were also in the outputted CSV file and had to be removed manually before feature extraction.

4.2.2.2 Feature Selection and Extraction

As in the pangram experiment, the feature set we used to infer was the same as the one used for tap input inference: the mean, median, minimum and maximum, skewness, and kurtosis values of each sensor axis.

Unlike in the pangram experiment, we chose to assume that each word took 0.5 seconds to trace in our feature extraction program, rather than 1 second per trace. This decision was made because of the nature of the words used as input; that is, many of the words were short and contained only two to three characters (see Figure 4.3). Although reducing the time assumed for each trace decreases the amount of information we can collect from longer word traces, it also helps prevent more than one trace being included in what we assume to be the features of only one trace. As a result, 105 records were used to compute the features of every traced word regardless of whether the trace was longer or shorter than 0.5 seconds. As in the tap and pangram experiment, the feature-containing CSV files were converted to ARFF files for classification and learning.

4.2.2.3 Learning and Classification

As in the other experiments, the SVM, k-NN, and random forest classification algorithms were used to determine their potential for inferring words inputted using a trace. Their inference performance and required computation times were also compared.

A 10-fold cross-validation was performed with each classifier trained using the full 5500 trace data set. We also evaluated the classifiers by splitting the data set into a test set containing 20% of the original data set (1100 traces) and a training set (4400 traces).

4.2.3 Results and Discussion

Although the classification accuracies of all three classification algorithms were quite low, using the classifiers to infer the inputted words had significantly higher accuracy than randomly guessing one of the 110 words (0.91%). As in the pangram experiment, the SVM had the highest percentage of correctly identified words (43.58%), followed by the random forest (37.27%), and then the *k*-NN classifier (27.29%) for the 10-fold cross-validation. The smallest relative absolute error was again seen with the k-NN classifier, although all of the classifiers had greater than 70% error.

The results for the split test and training set runs were very similar to the cross-validation. The SMV still performed best in terms of accuracy (41.09%), followed by the random forest (32.64%) and the *k*-NN (25.36%) classifiers. The relative absolute error was also still above 70% for all classifiers, with SVM having the highest and *k*-NN

	10-Fold Cross-Validation		
	k -NN	Random Forest	SVM
Correctly Classified	27.29%	37.27%	43.58%
Relative Absolute Error	73.95%	80.90%	99.15%
Model Build Time (sec)	0	4.22	13.31
	Test Set		
	k -NN	Random Forest	SVM
Correctly Classified	25.36%	32.64%	41.09%
Relative Absolute Error	75.91%	82.99%	99.14%
Model Build Time (sec)	0	3.05	13.55

Figure 4.4: Accuracies and time requirements of the three classification algorithms compared (*k*-nearest neighbor, support vector machine, and random forest).

having the lowest. As expected from the pangram experiment, the SVM took approximately three times longer to build than the random forest classifier. These results and those from the cross-validation are summarized in Figure 4.4. Although it is the slowest of the classification algorithm examined, SVM allowed for an inference accuracy that was over 6% higher than the accuracy obtained using the random forest algorithm. Unlike with the pangram, where the random forest outperformed the SVM when using the test set, the best accuracy was achieved using SVM in all cases. Because all accuracies are below 50%, the benefit of increased accuracy when using SVM outweighs its higher time requirements, making it the best choice in this experiment.

Although the inference accuracy was low, it should be noted that using the motion sensor data was 28-47 times better than random guessing. This improvement was achieved without any optimization or customization of the classification algorithms. By simply changing the complexity factor used with the SVM algorithm from 1 to 2, the inference accuracy was increased to 46.80% (10-fold cross-validation).

Chapter 5

Conclusion and Future Work

We have collected gyroscope and accelerometer data from an Android tablet while a user inputs text and have demonstrated the potential for this data to be used to infer the user text whether it was tapped or traced. Although our work is only in its early stages, it adds to the evidence garnered by previous works that leaving a mobile device's motion sensors unrestricted may be a threat to users' privacy. An attacker could potentially modify a legitimate application to collect sensor data in the background with the user's knowledge and use it to rebuild user text containing passwords or other sensitive material regardless of whether the user's preference is for tapping letter-by-letter or tracing full words. Due to the low battery consumption of the sensors, the ever increasing data allowance on mobile plans, and the ability to use mobile devices with Wi-Fi, this kind of activity may go unnoticed by users [27].

5.1 Impact of Classifier on Input Prediction

We compared three classification algorithms, namely *k*-Nearest Neighbor (*k*-NN), Random Forest, and Support Vector Machine (SVM), for use with inferring both tap and trace inputs on an Android tablet. Our results suggest that although classification accuracy is fairly similar among the three algorithms, random forest consistently had a

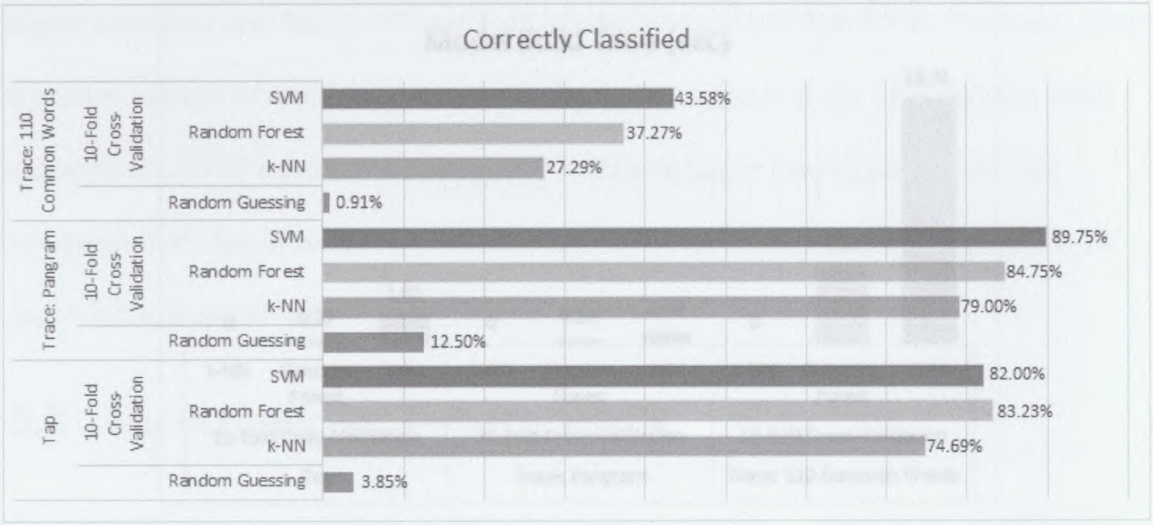

Figure 5.1: Classification accuracies of the three classification algorithms (*k*-NN, SVM, and random forest) and the accuracy of random guessing for the 10-fold cross validation run of the three main experiments (Tap input only, trace input using a pangram, and trace input using 110 common words).

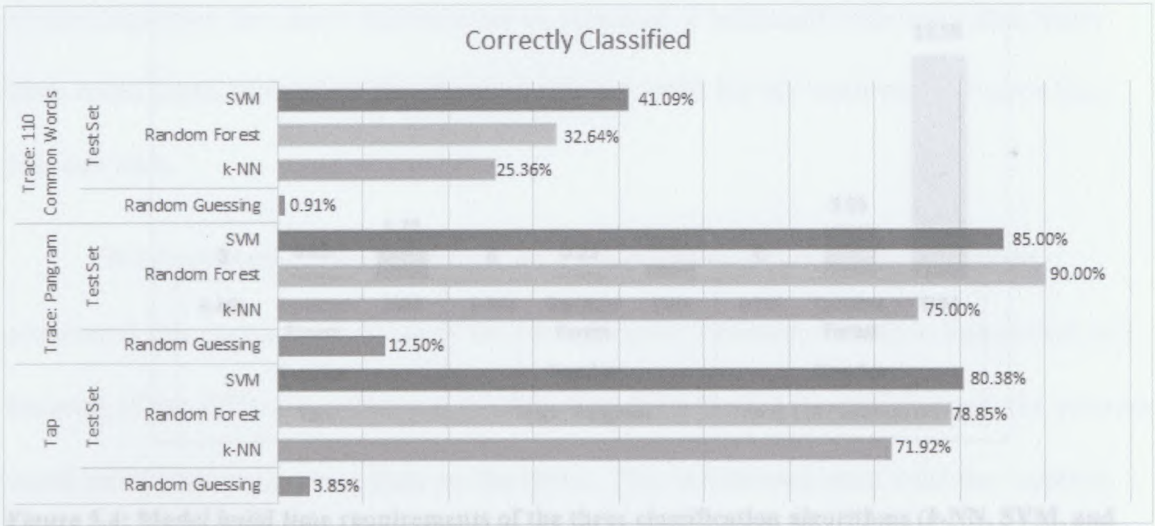

Figure 5.2: Classification accuracies of the three classification algorithms (*k*-NN, SVM, and random forest) and the accuracy of random guessing for the test set/training set run of the three main experiments (Tap input only, trace input using a pangram, and trace input using 110 common words).

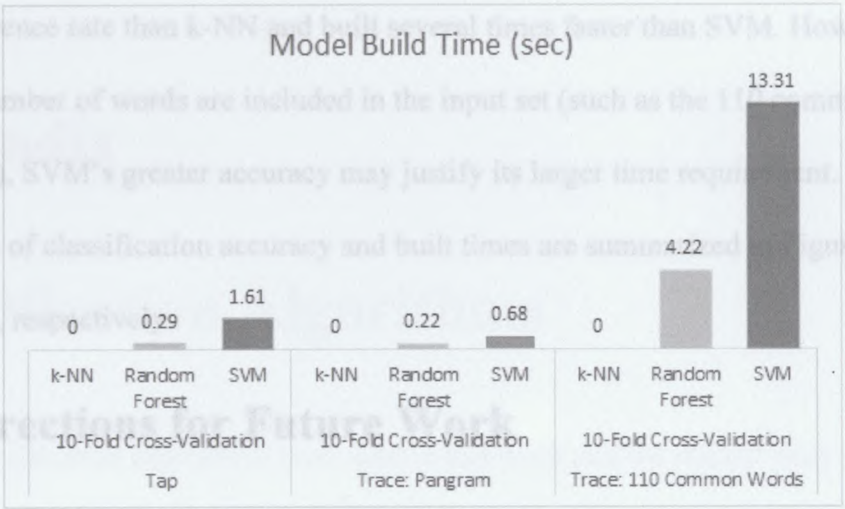

Figure 5.3: Model build time requirements of the three classification algorithms (*k*-NN, SVM, and random forest) guessing for the 10-fold cross validation run of the three main experiments (Tap input only, trace input using a pangram, and trace input using 110 common words).

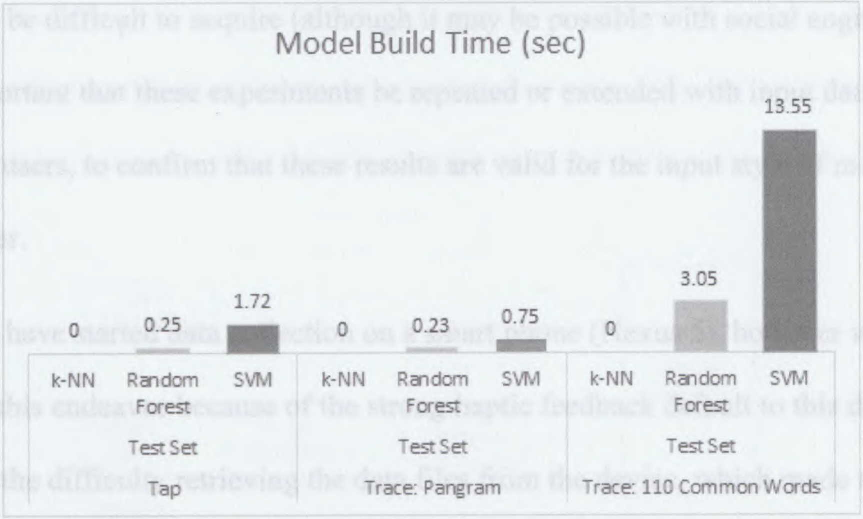

Figure 5.4: Model build time requirements of the three classification algorithms (*k*-NN, SVM, and random forest) guessing for the test set/training set run of the three main experiments (Tap input only, trace input using a pangram, and trace input using 110 common words).

higher inference rate than k-NN and built several times faster than SVM. However, when a greater number of words are included in the input set (such as the 110 common word experiment), SVM's greater accuracy may justify its larger time requirement. This comparison of classification accuracy and built times are summarized in Figure 5.1-5.2 and 5.3-5.4, respectively.

5.2 Directions for Future Work

In the future, we plan to attempt to modify our system so that it can interpret mixed tap and trace input. We believe it would be beneficial to determine whether it is possible to use one person's input data to infer another user's input. If this is possible, it would allow an attacker to build a training set without labeled input from the target, which may be difficult to acquire (although it may be possible with social engineering). It is also important that these experiments be repeated or extended with input data taken from more users, to confirm that these results are valid for the input style of more than just one user.

We have started data collection on a smart phone (Nexus 5), however we postponed this endeavor because of the strong haptic feedback default to this device and because of the difficulty retrieving the data files from the device, which made the process much more time consuming than on the tablet. This is a known issue with the Android MTP (Media Transfer Protocol), and it does not affect devices that use USB Mass Storage [21]. Tablets were shown to have better inference accuracy than smart phones previously in [19], however it may be interesting to determine how much haptic keyboard feedback affects the ability of tap and traces to be inferred.

Appendix

Classification Algorithms

Classification algorithms were used in this work and the related works to relate the sensor data to the indicated output (the translation to text). The chosen algorithm or algorithms use features derived from a training set of data to "learn" the relationship between members within different categories. After this training phase, the membership (or label) of new instances of data (or objects) can be predicted. The choice of algorithm used for learning can affect how accurately membership is predicted, as can the training and testing data sets and the features chosen from these sets [28]. The following three sections give an overview of the specific classification algorithms use in this thesis.

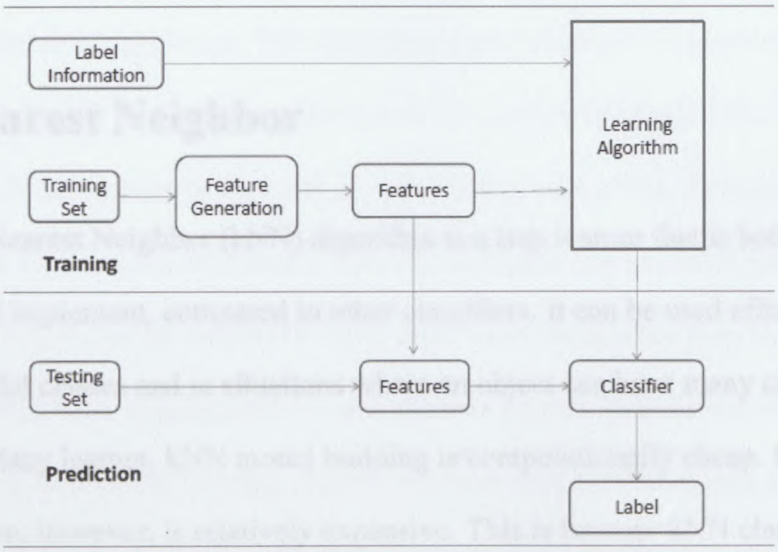

Figure A.1: General overview of data classification. Source Tang [28].

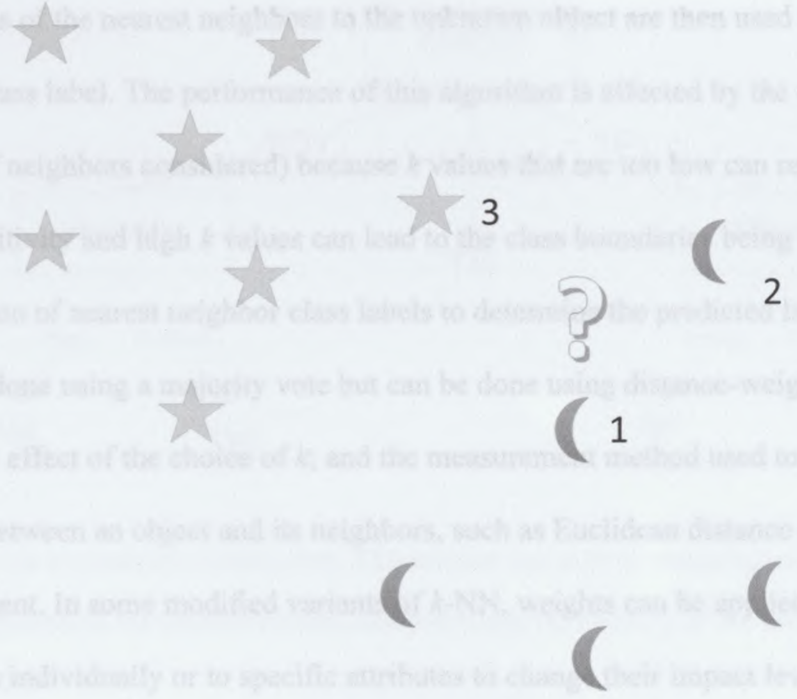

Figure A.2: *k*-NN example. The question mark represents the unknown object, which has two possible class labels: a star or a moon. If *k* = 1, only the moon labeled "1" is considered as a neighbor. If *k* = 2, the two moons labeled "1" and "2" are considered. If *k* = 3, the star labeled "3" is also considered, in addition to the two moons. In all three of these cases, the unknown object would be predicted to be a moon because the majority of its nearest neighbors are moons.

A.1 *k*-Nearest Neighbor

The *k*-Nearest Neighbor (kNN) algorithm is a lazy learner that is both easy to understand and implement, compared to other classifiers. It can be used effectively for both multi-modal classes and in situations where an object can have many class labels. Because it is a lazy learner, kNN model building is computationally cheap. Using kNN for classification, however, is relatively expensive. This is because kNN classifies an unknown object by computing the distance between it and known (labeled) objects. The class labels of the nearest neighbors to the unknown object are then used to determine the object's class label. The performance of this algorithm is affected by the choice of *k* (the number of neighbors considered) because *k* values that are too low can result in increased noise sensitivity and high *k* values can lead to the class boundaries being unclear; the combination of nearest neighbor class labels to determine the predicted label, which is normally done using a majority vote but can be done using distance-weighted votes to reduce the effect of the choice of *k*; and the measurement method used to determine the distance between an object and its neighbors, such as Euclidean distance or a cosine measurement. In some modified variants of *k*-NN, weights can be applied to the training set objects individually or to specific attributes to change their impact levels [28, 29]. An example of *k*-NN can be seen in Figure A.2.

A.2 Support Vector Machines

Support Vector Machines (SVM) are considered to be one of the most robust and accurate classification algorithms. This algorithm has the benefit of requiring only a fairly small training set, and it is not sensitive to the number of dimensions. The main idea behind SVM algorithms is to create an optimal hyperplane that distinguishes between groups with a maximal margin. This classification can be either linear or non-linear (or kernel) [29]. In the non-linear case, three basic kernels are used with SVM: the polynomial, radial basis function (RBF), and sigmoid kernels [30]. However, these are not the only kernels available. Using a kernel function with a SVM allows for nonlinear input relationships to be defined. SVM classifiers can also be both binary and multi-class. While SVM algorithms can be slow due to their computational inefficiency, there have

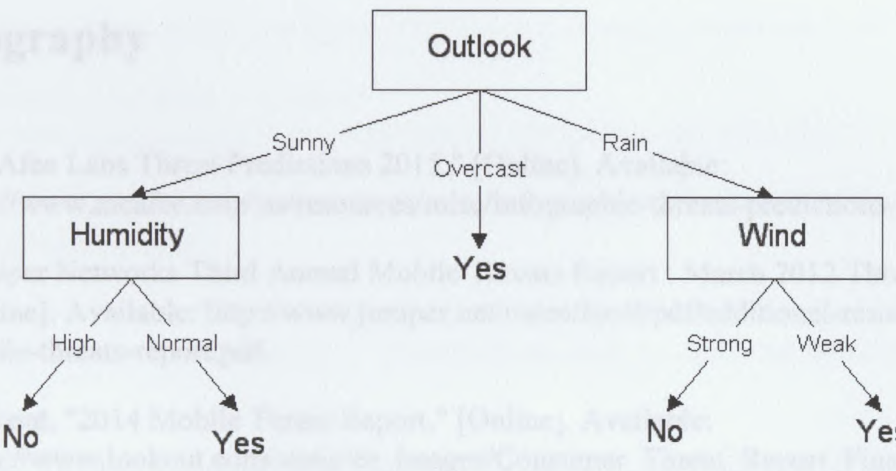

Figure A.3: An example of a decision tree. This decision tree is for determining whether to play outside depending on the weather outlook, and potentially the humidity or the wind depending on the outlook [31].

been some successful efforts to optimize their speed and their accuracy may outweigh the higher time requirement [29].

A.3 Random Forest

The random forest algorithm is an ensemble approach that involves the use of multiple decision trees. These decision trees are created from different subsets of the original feature set. The class labels predicted by each of the decision trees is compared and the most commonly predicted class label is chosen as the final prediction. The goal of this technique is to use a collection of weak learners (the decision trees) to build a strong learner (the random forest) [32].

Bibliography

[1] "McAfee Labs Threat Predictions 2015," [Online]. Available: http://www.mcafee.com/us/resources/misc/infographic-threats-predictions-2015.pdf.

[2] "Juniper Networks Third Annual Mobile Threats Report : March 2012 Through March 2013," [Online]. Available: http://www.juniper.net/us/en/local/pdf/additional-resources/jnpr-2012-mobile-threats-report.pdf.

[3] Lookout, "2014 Mobile Threat Report," [Online]. Available: https://www.lookout.com/static/ee_images/Consumer_Threat_Report_Final_ENGLISH_1.14.pdf

[4] "F-Secure MobileThreat Report Q1 2014," [Online]. Available: http://www.f-secure.com/static/doc/labs_global/Research/Mobile_Threat_Report_Q1_2014.pdf.

[5] M. Labs, "Threats Report June 2014," [Online]. Available: http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q1-2014.pdf.

[6] "Symantec Internet Security Threat Report Appendix 2014," [Online]. Available: http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_appendices_v19_221284438.en-us.pdf.

[7] "Treand Micro Security Predictions for 2014 and Beyond," [Online]. Available: http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/reports/rpt-trend-micro-security-predictions-for-2014-and-beyond.pdf.

[8] A. Al-Haiqi, M. Ismail and R. Nordin, "Keystrokes Inference Attack on Android: A Comparative Evaluation of Sensors and Their Fusion," *Journal of ICT Research and Applications,* 2013.

[9] "FireEye," [Online]. Available: http://www.fireeye.com/blog/technical/botnet-activities-research/2013/12/misosms.html.

[10] "Sophos," [Online]. Available: http://blogs.sophos.com/2014/02/05/sophoslabs-android-malware-intercepts-sms-messages-to-steal-banking-info/.

[11] P. Marquardt, A. Verma, H. Carter and P. Traynor, "(sp) iPhone: decoding vibrations from nearby keyboards using mobile phone accelerometers.," in *18th ACM Conference on Computer and communications Security*, 2011.

[12] S. Dey, N. Roy, W. Xu, R. R. Choudhury and S. Nelakuditi, "Accelprint: Imperfections of accelerometers make smartphones trackable," in *Network and Distributed System Security Symposium*, 2014.

[13] "Sensors Overview: Android Developers," [Online]. Available:

http://developer.android.com/guide/topics/sensors/sensors_overview.html.

[14] "Teardown," [Online]. Available: http://www.eetimes.com/author.asp?section_id=36&doc_id=1321925.

[15] "MathWorks," [Online]. Available: http://www.mathworks.com/help/simulink/samsung-galaxy-android-devices.html.

[16] E. Owsu, J. Han, S. Das, A. Perrig and J. Zhang, "ACCessory: Password Inference using Accelerometers on Smartphones," *Workshop on Mobile Computing Systems and Applications*, 2012.

[17] Z. Xu, K. Bai and S. Zhu, "TapLogger: Inferring User Inputs on Smartphone Touchscreens Using On-board Motion Sensors," in *Security and Privacy in Wireless and Mobile Networks*, 2012.

[18] L. Cai and H. Chen, "On the Practicality of Motion Based Keystroke Inference Attack," in *5th International Conference on Trust and Trustworthy Computing*, 2012.

[19] E. Miluzzo, A. Varshavsky, S. Balakrishnan and R. R. Choudhury, "TapPrints: Your Finger Taps Have Fingerprints," in *International Conference on Mobile Systems, Applications, and Services*, 2012.

[20] A. Al-Haiqi, M. Ismail and R. Nordin, "On the Best Sensor for Keystrokes Inference Attack on Android," in *International Conference on Electrical Engineering and Informatics*, 2013.

[21] "Android Open Source Project Issue Tracker," [Online]. Available: http://code.google.com/p/android/issues/detail?id=38282.

[22] "Apache Commons," [Online]. Available: http://commons.apache.org/proper/commons-math/userguide/stat.html.

[23] "CSV2ARFF," [Online]. Available: http://slavnik.fe.uni-lj.si/markot/csv2arff/csv2arff.php.

[24] "Weka," [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/.

[25] "CNET," [Online]. Available: http://www.cnet.com/news/move-over-t9-here-comes-swype/.

[26] "Oxford Dictionaries," [Online]. Available: http://www.oxforddictionaries.com/words/the-oec-facts-about-the-language.

[27] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu and M. Musolesi, "Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application.," in *6th ACM Conference on Embedded Network Sensor Systems*, 2008.

[28] J. Tang, S. Alelyani and H. Liu, "Feature selection for classification: A review.," in *Data Classification: Algorithms and Applications*, 2014.

[29] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda and D. Steinberg, "Top 10 Algorithms in Data Mining," *Knowledge and Information Systems,* 2008.

[30] C. Hsu, C. Chang and C. J. Lin, "A Practical Guide to Support Vector Classification".

[31] T. Mitchell , Machine Learning, New York, NY: McGraw-Hill, Inc, 1997.

[32] L. Breiman and A. Cutler, "Random Forests," [Online]. Available: http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm.

[33] "Interpreting Results: Skewness and Kurtosis," [Online]. Available: http://www.graphpad.com/guides/prism/6/statistics/index.htm?stat_skewness_and_kurtosis.htm.