Technische Universität Berlin

Institut fü Softwaretechnik und Theoretische Informatik
Quality and Usability Lab

Fakultät IV
Franklinstrasse 28-29
10587 Berlin

Master Thesis

M.Sc. Computer Science

Predicting tap locations on touch screens in the field using accelerometer and gyroscope sensor readings

Emanuel Schmitt

Matriculation Number: 333772

Examined by:
Prof. Dr.-Ing. Sebastian Möller
Prof. Dr. Axel Küpper

Supervised by:
Dr.-Ing. Jan-Niklas Voigt-Antons
Carola Trahms

I would like to thank the following persons and institutions for the support of my master thesis.

Dr. Jan-Niklas Voigt-Antons for the last 2.5 years I was a research assistant at the Quality and Usability Lab. I am grateful for all the projects we worked on together and for the support throughout the last years.

Carola Trahms for the supervison and the initial idea of the topic.

Irene Huber-Achter and Yasmin Hillebrenner for handling the vouchers which were given to all participants of this study.

Prof. Dr. Sebastian Möller and the whole Quality and Usability Lab.

My family and friends, especially Lisa Brust, for the unconditional support at any time of my studies. A big thank you to my parents, without whom all this would not have been possible.

Hereby I declare that I wrote this thesis myself with the help of no more than the mentioned literature and auxiliary means.

Berlin, 15.10.2017

. . . . . .

(Signature [Emanuel Schmitt])

Abstract

Recent research has shown that the location of touch screen taps on modern smartphones and tablet computers can be identified based on sensor recordings from the device's accelerometer and gyroscope. This security threat implies that an attacker could launch a background process on the mobile device and send the motion sensor readings to a third party vendor for further analysis. Even though the location inference is a non-trivial task requiring machine learning algorithms in order to predict the tap location, previous research has shown that PINs and passwords of users could be successfully obtained. However, as the tap location inference was only shown for taps generated in a controlled setting not reflecting the environment users naturally engage with their smartphones, the attempts in this work bridge this gap.

In this work, I propose TapSensing, a data acquisition system designed to collect touch screen tap event information with corresponding accelerometer and gyroscope readings. Having performed a data acquisition study with 27 subjects and 3 different iPhone models, a total of 45,000 labeled taps could be acquired from a laboratory and field environment enabling a direct comparison of both. The overall findings show that tap location inference is generally possible for data acquired in the field, hence, with a performance reduction of approximately 20% when comparing both environments. As the tap inference has therefore been shown for a more realistic data set, this work yet again confirms that smartphone motion sensors could potentially be used to comprise the user's privacy.

Zusammenfassung

Kürzliche Untersuchungen haben gezeigt, das die Position eines Taps auf einem Smartphone oder Tablet Displays über eine Aufzeichnung der Lagesensoren bestimmt werden kann. Diese Sicherheitslücke deutet darauf hin, dass ein potentieller Angreifer einen Hintergrundprozess auf dem Zielgerät installieren und die aufgezeichneten Daten an eine Server Applikation zur weiteren Analyse schicken könnte. Obwohl die Bestimmung der Postion auf dem Touchscreen keine einfache Aufgabe darstellt, die nur mit Hilfe von Maschinellem Lernen möglich ist, ist es Forschern dennoch gelungen PINs und Passwörter von einzelnen Nutzern auszuspähen. Da jedoch die Daten zur Vorhersage in den bisherigen Versuchen von Nutzern stammen, die in einem Laborversuch aufgezeichnet wurden und die Daten hierbei keine reale Nutzerumgebung darstellen, schliesst diese Arbeit diese Lücke.

In dieser Arbeit stelle ich TapSensing vor, eine Applikation die Touch Screen Interaktionen mit den entsprechenden Lagesensor-Daten aufzeichnet. In einem Labor sowie in einem Feldversuch mit insgesamt 27 Teilnehmer, wurden 45.000 einzelne Taps aufgenommen. In einer Analyse konnte gezeigt werden, dass eine Vorhersage der Tap-Position auf dem Display ebenso für Daten aus dem Feld möglich ist. Im direkten Vergleich zwischen Feld und Labor wurden jedoch Performance-Einbußen von rund 20% für Daten aus dem Feld gemessen. Da die Tap-Positions-Inferenz nun für reale Konditionen gezeigt werden konnte, können wir uns bisherigen Untersuchungen anschließen und erneut bestätigen, dass Lagesensoren ein potentielles Sicherheitsrisiko für den Nutzer darstellen.

Contents

List of Figures

List of Tables

1 Introduction

The utilization of smartphones has become an integral part of our everyday life. We use them to perform various tasks ranging from highly privacy-sensitive tasks such as for bank transactions or personal communication to more casual tasks such as setting an alarm clock or checking the weather. This universal applicability is one important factor that has contributed to the immense success of the smartphone. Another factor is the rich set of embodies sensor, such as an accelerometer, digital compass, gyroscope, GPS, microphone and camera [27] which have enabled developers to introduce highly interactive applications.

Location based services, for instance, utilizing the GPS sensor [31] can lead users on the fastest route to their desired destination while health tracking applications [12], enabled by the motion sensors, can recommend health beneficial behavior based on the amount of physical activity sensed. Furthermore, newly introduced augmented reality applications utilize the camera and the motion sensors to enhance our perception of our immediate surroundings resulting in a whole new interactive experience. As these are positive examples for sensor usage, there have also been reports of sensor utilization with a more malicious intent.

The motion sensors, gyroscope and accelerometer, which are typically used for detecting the device orientation and for gaming applications [16], can be used to infer the locations of touch-screen taps. As the striking force of a tapping finger creates an identifiable signature on the 3-axis motion sensors, previous research has shown that the granularity of inference is adequate to obtain PINs and passwords [11, 35, 44] or at least significantly reduce the search space to do so. The situation is furthermore reinforced by the fact that motion sensors do not require special privileges or access rights on operation system level. Due to this purpose, the motion sensors expose a vulnerable side-channel for potential attackers to eavesdrop on user interactions.

However, as the motion data used to train the inference systems in the previous research was acquired from users in a controlled setting [35, 11, 44], the feasibility of tap location inference has not been shown for a more realistic data set that is capable of modeling natural user behavior as well as their changing environments. It is plausible that when a user interacts with the touch screen, for instance, while walking in the park or during

a public transportation ride, the sensory data will be effected by this activity potentially mitigating an eavesdropping attack.

In order to address this issue, I would like to propose *TapSensing*. TapSensing is a data acquisition system designed to acquire tap information with corresponding accelerometer and gyroscope readings. After having conducted a laboratory and field study, I have collected over 45,000 taps from 27 different subjects to investigate if the proposed security threat posed by leaking motion sensor also applies to an uncontrolled environment.

1.1 Motivation

In recent years, the number of smartphone users has rapidly increased. According to a report published by Smart Insights[1], the number of smartphone users grew from 400 million users in 2007, to more than 1,800 million in 2015. In addition, the report claims that at the end of 2017, 97% of adults, aged 18 to 34, in the US were mobile device users. Due to this rapid adaption and the smartphones rich set of sensors, smartphones have increasingly become targets for various attacks [13, 3, 11].

As gyroscope and accelerometer are commonly used for all genres of applications ranging from gaming [16] to productivity apps [46], the mobile operating system providers offer easy-to-use access via standard APIs[2]. Consequently, a smartphone application designed for a real-world attack could sense the user's motion in the background and send the information to a server-side application for machine learning analysis. This is possible due to the fact that background tasks are fully supported on the Android platform while on iOS, methods have been developed to run initially prohibited background tasks[3] for long durations than usually supported. Notably, all this can be achieved without the user's consent due to the lack of access restriction for motion sensors.

A further point emphasizing the security threat is that in 2017 the World Wide Web Consortium, W3C, has released the so-called Device Orientation specification [1] for JavaScript. This specification allows browser applications to access the motion sensor hardware of smartphones opening further possibilities for an attacker. A potentially harmless website could therefore obtain the motion sensor data and reveal private information of the user.

[1] http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/
[2] Application Programming Interface
[3] https://github.com/yarodevuci/backgroundTask

Besides all the privacy issues motion sensors raise, tap location inference could also be used for usability research purposes. Assuming that the inference provides high enough accuracies, it could be used as an in-app behavior tracking for applications where the source code is publicly not available. The inference system could run in a background task to evaluate the user behavior in a target application.

1.2 Outline

This work is structured as follows:

- **Related Work**: In chapter 2, previous research concerning side-channel attacks is presented. As this work is based on similar studies, they will be outlined in this chapter.
- **Machine Learning Fundamentals**: In chapter 3, I will review the machine learning fundamentals required to understand the technical aspects of this thesis. Since learning the relation between the sensory data and the tap location is a supervised learning task, the classification algorithms used in this thesis will be introduced.
- **Data Aquisition System**: In chapter 4, I will introduce *TapSensing* which is the data acquisition system used for obtaining taps and sensor readings for both the laboratory and the field environment.
- **Methodology**: In chapter 5, the methodology of the data acquisition, data pre-processing and machine learning classification is explained.
- **Results**: In chapter 6, meaningful results are presented from the data acquired in the user study.
- **Conclusion**: In the final chapter 7, I will conclude the thesis with an overview of the results and ideas for future studies.

2 Related Work

A tap location inference is a form of eavesdropping, this chapter will shed light into similar approaches of side-channel attacks that have been revealed by researchers in the past. The first section deals with different kinds of device emanations that enable attackers to obtain confidential information. The second section covers the foundations of this work as it discusses previous similar attempts to predict tap locations on smartphone screens.

2.1 Eavesdropping on Emanations

Eavesdropping is defined as the practice of secretly listening to private conversations of others without their consent [9]. As this definition originally refers to conversations between humans, eavesdropping can also be seen in terms of human-computer-interaction. In this context, the computer is seen as one conversational partner whereas the user interacting with the device is seen as the other. The channel in which human-computer interaction takes place has various forms. Typically, a human may enter information on a peripheral device while the computer gives feedback through an image representation. However, as in recent times, conversations though natural language interfaces are also gaining popularity [50].

In order to spy on these conversations, various techniques have been revealed in the past of which a subset involves the utilization of leaking emanations. These emanations can be monitored in order to carefully reconstruct the contained information. As the emanations occur in various formats, a categorization has been made based on the sensory channel they transpire. Therefore, the following sections will cover eavesdropping techniques based on acoustic, optical, electromagnetic and motion emanations.

2.1.1 Acoustic Emanations

One way of spying on electrical devices is by utilizing the acoustic channel [6, 2, 56]. Many electronic devices deploy tiny mechanics that generate sounds as a byproduct during in-

teractions or during operation. These distinct sounds can differ in their characteristics making them adequate to identify the original information currently being processed by the machine. In these scenarios a potential eavesdropper targets a microphone in near proximity of the target device to capture the audio signals the device is currently exposing. A learning algorithm is then applied to the audio signals to reconstruct information.

In 2010, Backes et al. examined the problem of acoustic emanations of dot matrix printers, which where, at that time, still commonly used in banks and medical offices. By using a simple consumer-grade microphone, the researchers were able to recover whole sentences the printer was processing [6]. By processing the audio samples and extracting features from the frequency domain, the researchers were able to train a Hidden Markov Model to reconstruct individual characters based on the sound inputs. Consequently, in order to demonstrate a potential attack, the researchers deployed the system in a medical office being able to obtain up to 72% of the sentences being printed on medical subscriptions [6].

Being inspired by the findings regarding the dot matrix printer, Asonov and Agrawal investigated acoustic emanations produced by hitting keystrokes on desktop and notebook keyboards. Following their hypothesis claiming that each keystroke has a macroscopic difference in its construction mechanics as well as a distinct reverberation caused by the position on the keyboard, individual keystrokes were recorded [2]. Researchers then extracted features from the audio signals and passed them into a neural network. In an experiment performing 300 keystrokes, 79% of the characters could be correctly recognized [2]. As this technique required substantial training before recognition, other studies have reached similar accuracies using an unsupervised approach [56] and by using acoustic dictionaries [8].

2.1.2 Optical Emanation

Besides acoustic emanations, optical emanations can also pose a valuable source of information for a potential eavesdropper. Most electronic devices, such as notebook, smartphones and tablet computers, provide graphical user interfaces through their own built-in screens. Even though these screens are meant to target the human eye, they can reflect off other surfaces. The reflections can be caught by high resolution camera sensors, which can unintendedly reveal private information.

One example of the use of optical emanations has been developed by the researcher Kuhn aiming to eavesdrop on cathode-ray-tube(CRT) monitors at distance [25]. The researcher has shown that the information displayed on the monitor can be reconstructed

from its distorted or even diffusely reflected light. In an experiment, Kuhn targeted a screen against a wall while the reflections of the screen were captured using a photomultiplier. The results showed that enough high-frequency content remained in the emitted light for a computer to reconstruct the original image [25]. Moreover, a similar approach that comprises reflections has been shown by Backes et al., however, focusing on LCD displays. In this experiment, the researchers caught reflections in various objects that are commonly to be found in close proximity to a computer screen. Such objects included eyeglasses, tea pots, spoons and even plastic bottles. This work was later extended to additionally capture screens based on the reflections on the human eye's cornea [5].

2.1.3 Electro-magnetic Emanation

As electric currents flow through computer components, they emit electromagnetic waves to their near surrounding. These electromagnetic radiations can be picked up as a side channel using sensitive equipment in order to retrieve data. Electromagnetic emanations have been a research topic that has been present for several decades while the first research conducted reaches far back in time.

Back in 1943, a research group under the codename TEMPEST, a subdivision of the NSA[1], were able to infer information from the infamous Bell Telephone model 131-B2, a teletype terminal which was used for encrypting wartime communication [43]. Using an oscilloscope, researchers could capture leaking electromagnetic signals from the device and by carefully examining the peaks of the recorded signals, the plain message the device was currently processing could be reconstructed [43]. This technique was later advanced and used in the Vietnam war where the US military could detect approaching Viet Cong trucks giving them an immense competitive advantage over their enemies [39]. Today, TEMPEST is a security standard for electronic devices ensuring that certified devices do not accidentally emanate confidential information [40].

A second prominent finding of eavesdropping on electromagnetic emanations was shown by van Eck discovering that cathode-ray-tube monitors could be spied upon from a distance [53]. By using general market equipment, such as antennas and standard receivers, signals emitted from the monitor cable could be received. Since these cables only transmit the video signals for visualization, the researcher could display the visual output of the target monitor revealing a full screen cast of the original image. This attack is referred to in literature as *Van Eck Phreaking* [20, 24].

[1] National Security Agency

Moreover, research has shown that side-band electromagnetic emanations are present in keyboards [54], computer screens [53, 26], printers [45], computer interfaces, such as USB 2 [42] and the parallel port [52] and in so-called Smart Cards [47].

2.1.4 Motion Emanation

As mentioned in the introduction section, modern devices, such as tablets or smartphones, are increasingly equipped with highly responsive sensors enabling the devices to sense interactions with their environment. As touch screen interactions, such as typing the keyboard or tapping icons, require the user to apply a certain force while entering information, this motion can be captured by the sensors in order to be used as a side-channel attack [35, 44, 11].

Marquardt et al. conducted an experiment placing an Apple iPhone capturing accelerometer motion was placed next to a desktop keyboard. Subjects then had to enter sentences while the application was monitoring the user's motion. The researchers could decode the accelerometer signals by mapping the vibration caused by the typing motion to individual keystrokes of the keyboard. Then the decoded characters were matched based on a dictionary to obtain words the user was typing. As a result, words could successfully be obtained with an accuracy of up to 80% [33].

As motion emanations are highly relevant for this work, the next section will be dedicated to further work regarding this topic.

2.2 Eavesdropping on touch screen user interactions

As we have seen in the previous section, confidential information can be obtained utilizing emanations from various channels. However, with the rise in software keyboard usage, the same discussed methods that extract information from keyboards do not apply to tap interactions on a touch screen surface. Since a touch screen does not embody fine mechanics producing sounds or emanating cables, the research community has developed nouvelle methods to spy on these kind of user interactions.

The general idea behind the three approaches, that are going to be discussed in the following, is that a tap or to be more precise the magnitude of the force of a tap, on a specific touch screen location creates an identifiable pattern on the motion sensors that can be sufficient to infer the initial tap location.

Touchlogger

The first paper regarding this security threat was published by Cai and Chen. In their proof-of-concept study they created an Android application which displays a 10-digit PIN-pad. During interactions with the PIN-pad, the accelerometer signals were monitored and used for later data analysis. Having observed that a tap movement affects the rotation angle of the screen, the researchers handcrafted features based on the path of the *pitch*[2] and the *roll*[3] angles of the accelerometer. Both angles were intersected to find a dominating edge on where the tap had presumably taken place. By using a probability density function, the researchers were able to achieve an average accuracy of 70% for inferred PIN-pad digits.

Even though *Touchlogger* was a promising first step, due to its low granularity of only 10 distinguishable large screen areas, it remained unclear if the attack could be carried out on a full software keyboard. Furthermore, since the inference was performed on a single smartphone model, the question was left open whether other smartphones or tablet computers are similarly vulnerable.

ACCessory

In order to show the feasibility for a full software keyboard, Owusu et al. performed a second attempt to the problem by creating *ACCessory*. *ACCessory* is an Android application with functionalities similar to the previously mentioned *Touchlogger*, however differing in the granularity of distinguishable areas to tap. In an experiment, the researchers tested tap location inference in two separate modes. The first one, the *area mode*, consisted of a 60-cell tapping grid and the second one, the *character mode*, consisted of a standard QWERTY keyboard.

As a result, on the 60-cell grid in *area mode*, individual regions could be inferred with an accuracy up to 24% using a Random Forest classifier. The corresponding data set consisted of 1300 keystrokes from 4 distinct users. For the *character mode*, three subjects in total had to enter pangram passwords on the QWERTY keyboard. By combining the individual keystrokes into a sequence and assuming recognition errors in individual characters, the researchers could create a ranked list of candidate passwords by running a maximum likelihood search for the most probable classification errors. Here, 6 out of 99 password could be inferred under 4.5 median trials given that one trail refers to traversing down

[2] The pitch-angle corresponds to the x-axis of the accelerometer.
[3] The roll angle corresponds to the y-axis of the accelerometer.

one item of the candidate list. Furthermore, the majority of 59 out of 99 passwords could be inferred within 2^{15} median trials.

As a general result, even though the overall accuracies of the learning system scored rather low, the researchers could significantly reduce the search space for reconstructing a password.

TapPrints

The most comprehensive study to date regarding motion sensor emanations was conducted by Miluzzo et al. and differs from *ACCessory* and *Touchlogger* in many important ways. While the previous studies are both evaluated on Android smartphones, *TapPrints* investigates the tap inference on both iOS and Android platforms including tablets and smartphones alike.

Another important point of differentiation is the overall learning system. In order to raise the level of entropy, *TapPrints* combines readings from the accelerometer and gyroscope. For feature extraction, time-domain and frequency-domain features as well as the correlation and angles between individual sensor components are considered. The researchers then used an ensemble method for classification purposes combining decision trees, support vector machines, k-nearest neighbors and multinomial logistic regression in a winner-takes-it-all[4] voting fashion. The data set collected in this study contains over 40.000 individual taps collected from 10 different users. In addition, the researchers also requested users to use different input modalities while typing including the usage of the index finger and the thumb.

The *TapPrints* paper contains two experiments: The first is an icon tapping experiment where icons are arranged in a 20 cell grid. The second one is a letter tapping experiment involving the standard software keyboard offered by the operating system. In the icon tapping experiment, an average accuracy of 78% was achieved for the iPhone whereas 67% of icon taps could be correctly inferred on the Android device. The researchers could also show that taps could also be inferred from users that were not part of the training process.

In the letter tapping experiment users were asked to enter pangram[5] sentences on the OS soft-keyboard. Results showed that on both iPhone and Android an average of 43% of the letters could be correctly classified. Even though the average accuracy for individual

[4] This implies that all classifiers classify separately and the classifier with the highest prediction score wins.
[5] A pangram is a sentence using every letter of a given alphabet at least once.

letters were not particularly high, Miluzzo et al. could show that when pangrams were repeatedly entered, a majority vote could be applied to individual character recognitions allowing to recover the whole pangram in approximately 15 trials.

To sum up, *TapPrints* could demonstrate that motion sensor do indeed pose a latent security threat. As of the high accuracies on the 20-cell grid, the researchers could show that PINs could be obtained across unseen users and devices. Furthermore, it was also revealed that due to a majority voting scheme repeatedly entered phrases such as passwords, could also be obtained in a malicious attack.

Comparison to similar studies

Since the data used in *TapPrints* and in the other mentioned studies was collected in a controlled environment [44, 11, 35], it is not possible to tell if the feasibility of tap inference will also apply to data collected in a field environment. As this has not been investigated, this will form one of the central research questions in this work.

To draw the border between this study and the previous mentioned ones, this study will differ or relate as follows:

- **User interface**: For data acquisition, the user interface will cover tap area grids, as we have seen in *ACCessory*, *Touchlogger* and *TapPrints*, with 4, 12 and 20 distinguishable classes.
- **Devices**: Unlike *Touchlogger* and *ACCessory* and due to simplicity, the devices used in this study will be on the iOS platform. Apple iPhone 6, 6s and 7 will be considered.
- **Motion sensors**: *TapPrints* has shown that the gyroscope yields more information than the accelerometer [35], therefore both sensors will be monitored. Furthermore, sensors will be read at a frequency of 100Hz, since this had been proven to deliver best results [35, 44].
- **Data set**: In comparison to related studies, a data set containing data points collected in a laboratory environment and the field environment will be acquired. This data will contain the index finder and thumb as input modalities and standing and sitting as possible body postures.
- **Feature extraction**: *TapPrints* shows a deliberate list of features [35] that will be partially adopted.
- **Classification**: A feed-forward neural network as well as a support vector machine with RBF-kernel will be used for classification.

3 Machine Learning Fundamentals

As machine learning techniques will be used for the classification of individual tap locations on a smartphone touch screen, the following chapter will give an general overview of the machine learning concepts necessary to understand the classification aspects of this thesis.

3.1 Overview and Definition

Ever since computers were invented, there has been a desire to enable them to learn [49]. This desire has grown into the field of machine learning which seeks to answer questions on how to build systems that automatically improve with experience. Machine learning covers a set of methods and algorithms designed to accomplish tasks where conventional hard-coded routines have brought insufficient results [37].

The goal of a machine learning algorithm is to learn a function f that is able to predict sensible output values $y \in Y$ give input values $x \in X$:

$$f : X \to Y \tag{3.1}$$

Solving this problem is hard as the amount of input values used for learning this function is typically smaller in size than the unseen input values on to which f is applied to. Therefore, the challenge lies in finding a function that generalizes to unseen input values without simply remembering the seen inputs.

Mathematically, machine learning problems are formalized as optimization problems of an objective function which indicates the quality of the functional mapping between X and Y. This problem can either be a maximization or minimization problem. If we speak of the latter, the objective function is referred to as the *error function*.

There are four main categories of machine learning methods to be found in literature which all differ in their approach to learn the function f and in respect to the amount of

training samples available [14, 34]. These categories consist of *supervised learning*, *unsupervised learning*, *reinforcement learning* and *evolutionary learning*. As the task at hand refers to a supervised learning problem, supervised learning will be outlined in the following.

3.2 Supervised Learning

Supervised learning refers to the case where N samples are given from $X \times Y$, called the training data set $T = \{x^{(i)}, y^{(i)} | i \in \{i = 1 \dots N\}\}$. The training data is assumed to consist of approximate samples of a target function $F : X \to Y$, that is be learned by the learning algorithm. The data samples $x^{(i)} \in X$ are called input *features* whereas $y^{(i)} \in Y$ correspond to the so-called *labels*. If Y consists of discrete labels the learning problem corresponds to a classification task whereas if Y is on a continuous scale the problem refers to a regression task (see [34]).

The goal of a regression task is to predict a value of a dependant numeric variable based on the values of the independent variables. In order to make predictions, the regression model is learned from previously measured values of the dependent and independent variables [14]. The goal of a classification task, on the other hand, is to assign an input object, modeled through a set of independent features, to one of a set of known classes. Since the classification model is trained on known objects with corresponding labels, the model is able to predict belongings to a certain class [14]. For the task at hand, the smartphone touch screen will be split into non-overlapping areas each representing different classes. As features derived from the sensor recordings will be mapped to these area classes, the learning task in this thesis refers to a classification problem.

Practical applications of supervised learning are image recognition [51, 28], e-mail spam filtering [18] or network anomaly detection [30]. However, these are just a small subset of what can be accomplished so far.

Presumably the most widely known machine learning techniques belong to this category, such as the Support Vector Machines (SVMs), Support Vector Regression (SVR), Artificial Neural Networks, Bayesian Statistics, Random Forests and Decision Trees [14].

3.3 Bias-Variance Trade-off

The error of a supervised learning algorithm can be decomposed into three components being bias, variance and noise (see [15]):

$$\text{error} = (\text{bias})^2 + \text{variance} + \text{noise} \tag{3.2}$$

The *Variance* refers to the amount of which f adapts to the variations in the training data set. Models with a high variance have a tendency to learn minor relations irrespective of the real signal of the input values and are therefore prone to *overfitting* the training examples. This phenomenon applies to very flexible models such as a complex artificial neural network. On the other hand, *bias* is the model's tendency to consistently learn the wrong things from the training examples as it can not take all the information into account. This refers to a too simple model *underfitting* the training examples. An example of high bias would be a linear method such as linear regression trying to map a non-linear function. Finally, the *noise* is the irreducible error of the data distribution (see [22]).

As the goal is to minimize the error function, there is always a trade-off between bias and variance with very flexible models having low bias and high variance, and rigid models having high bias and low variance. Therefore, the model with the ideal predictive capability is the one that leads to the best balance between both properties [14].

3.4 Support Vector Machines

The SVM is a non-linear kernel based extension of the so-called maximum margin classifier. Originating from binary classification problems, were $y \in \{1, -1\}$, the general idea of a maximum margin classifier is to find a separating hyperplane in the p-dimensional feature space.

This hyperplane separates the training examples leading to a maximum distance between the observations of the two classes. This distance is referred to as margin M measuring the smallest distance of a training observation towards the defined hyperplane. Mathematically, the support vector classifier can be described as following optimization prob-

lem [22]:

$$\max_{\beta,\epsilon} M \tag{3.3}$$

$$\text{subject to} \sum_{s=1}^{p} \beta_s^2 = 1 \tag{3.4}$$

$$g(x_i) = y_i(\beta_0, \beta_1 x_1, \ldots \beta_p x_p) \geq M(1-\epsilon) \tag{3.5}$$

$$\epsilon \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C \tag{3.6}$$

The objective of this optimization problem is to maximize the margin M while choosing appropriate vector parameters β and ϵ. In this context, the parameter vector β contains the coefficients of the hyperplane whereas the vector ϵ includes so-called slack variables that account for instances which are located on the wrong side of the margin and the hyperplane. These can be expressed as follows assuming that M is positive [22]:

$$\epsilon_i = \begin{cases} 0 & g(x_i) \geq M \\ > 0 & M < g(x_i) < 0 \\ > 1 & g(x_i) > 0 \end{cases} \tag{3.7}$$

The hyperparameter C allows for a certain sum of ϵ_i observations to be on the wrong side of the margin or hyperplane, respectively [22]. C manages the bias-variance trade-off, since a low C tries to find a maximum margin hyperplane that separates the two classes, resulting in a low bias classifier for the available data set, but in a high variance classifier for test data. Subsequently, allowing a high C results in a high bias classifier that widens the margin which introduces more violations ϵ_i and reduces the variance of the classifier. Furthermore, C also controls the number of considered support vectors in dependence of the margin width (see [15]).

Extending the support vector classifier to non-linear decision boundaries brings us to the SVM. Instead of extending the predictor space using higher order polynomials and interactions, SVM uses the so-called “kernel trick” [15] resulting in the optimization problem to be rewritten as follows:

$$\hat{f}(x) = \beta_0 + \sum_{i \in S} \alpha_i \langle x, x_i \rangle \tag{3.8}$$

where $i \in S$ defines the subset of support vectors and $\langle x, x_i \rangle$ is the dot product of all pairs in the support vector. Thus, the parameters β_0 and $\sum_{i \in S}$ can be estimated with the help of least squares by computing the inner products of each pair in the support vector [15]. The

expression in $\langle x, x_i \rangle$ can be generalized by a kernel function

$$K(x_i, x_{i'}) = \sum_{s=1}^{p} x_{is} x_{i's} \tag{3.9}$$

indicating the linear kernel that quantifies the distance between each pair in the data set [22]. Accordingly, the equation above can be rewritten as

$$\hat{f}(x) = \beta_0 + \sum_{i \in S} \alpha_i K(x, x_i) \tag{3.10}$$

but in spite of restricting $K(\cdot, \cdot)$ to 3.9, an arbitrary kernel function can be chosen mapping the data into a high dimensional space where it is linearly separable. The "kernel trick" allows the SVM to work in an enlarged predictor space, by computing $\binom{n}{2}$ kernel functions $K(\cdot, \cdot)$, as opposed to an explicitly augmented predictor space, which is in fact computationally intractable [22]. In this work, we will be using a radial kernel function:

$$K(x_i, x_{i'}) = \exp(-\gamma(\sum_{s=1}^{p} x_{is} x_{i's})^2), \tag{3.11}$$

where γ is a tuning parameter. After the parameters are learned on the basis of the training set, a new observation with the feature vector x_0 is classified via the following decision rule

$$\hat{f}(x) = sign(\beta_0 + \sum_{i \in S} \alpha_i K(x_i, x_{i'})). \tag{3.12}$$

In view of the task at hand, an extension to multi-class classification of the SVM is utilized via one-versus-one classification. Given n classes, $\binom{n}{2}$ binary classifiers are learned.

3.5 Artificial Neural Networks

Artificial neural networks (ANNs) are computing systems inspired by the biological neural networks found in animal brains [19]. As these systems consist of several components, the first section will cover the artificial neuron which forms the fundamental processing unit of a network. Subsequently, individual activation functions of ANNs are discussed followed by the backpropagation algorithm which is used for training.

3.5.1 Artificial Neurons

A neuron is a fundamental processing unit of an artificial neural network. The diagram shows a model of a neuron which consists of following components (see [19]):

- A set of *connecting links* or *synapses* which have a certain weight defined as the vector $\vec{w}$. The signal, represented as $\vec{x}$, flows through the *synapse* and is multiplied by its weight w_i.
- An *adder* for summing the input signals and weights of the incoming synapses. These operations constitute a linear combiner.
- An *activation function* φ for limiting the amplitude of the output signal. This function is often referred to as the *squashing function* since it squashes the possible output range [19].
- An externally applied *bias* b which has the ability to lower or rise the net input to the activation function depending if the bias is negative or positive.

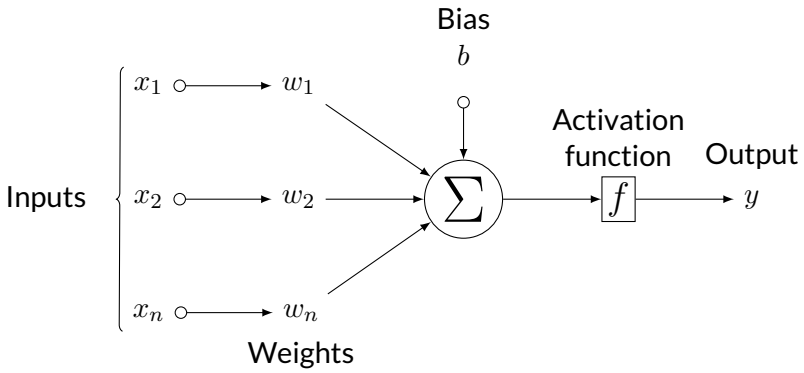


Figure 3.1: The diagram shows an artificial neuron's model with its individual components.

Mathematically, a neuron k can be expressed with the following equation [19]

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{3.13}$$

$$h_k = \varphi(u_k + b_k) \tag{3.14}$$

where $x_1, \ldots, x_n$ are the input signals; $w_{k1}, \ldots, w_{kn}$ refer to the synaptic weights of the neuron; u_k is the linear combiner output of the summation on which the bias b is added. The output of the neuron is expressed as h_k.

3.5.2 Activation Functions

The activation function $\varphi(u_k)$ denotes the output of the neuron k and forms the junction between the neuron's input x_k and output h_k. In order for the network to learn any complex non-linear function, each neuron in the networks requires a non-linear activation function [17]. In this context, one commonly used function is the s-shaped *sigmoid function* of which the *logistic function* [19] is an example:

$$\varphi(v) = \frac{1}{1 + \exp(-v)} \tag{3.15}$$

The logistic function, as in figure 3.2, is well suited for classification as it is a non-linear function and it transforms the output values to either side of the curve. This results in a clear distinction between classes. However, the function has a compact domain range, meaning that the logistic function squashes output values into ranges $(0, 1)$. Consequently, when the inputs of a neuron become large, the function saturates at 0 or 1 with a derivative in these points being close to 0. As the derivatives are used for training the network[1], the network trains slower if the weights or biases are high. This is referred to in literature as the *vanishing gradient problem* [38].

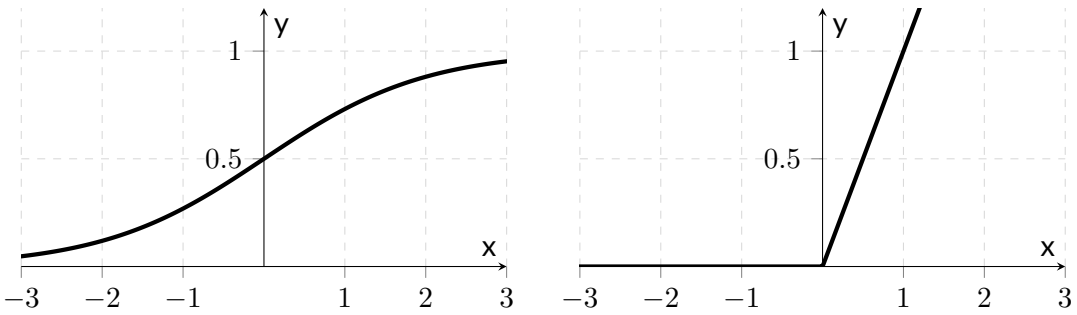


Figure 3.2: The figure shows a plot of the hyperbolic tangent activation function on the left and a plot of the ReLU activation function on the right.

In order to avoid saturation problems, a commonly used non-linear activation function is the Rectified Linear Unit (ReLU) function [38], which can be seen in figure 3.2:

$$\varphi(v) = \max(0, v) \tag{3.16}$$

[1] The training is performed via the backpropagation algorithm which is explained in section 3.5.4

ReLUs are non-saturating which results in a neuron always learning if the input is positive. Networks with ReLUs train several times faster and have become, as of 2015, the standard activation function for deep neural networks [29].

3.5.3 Feedforward neural networks

A feedforward neural network, or multilayer perceptron (MLP), is an ANN which consists of multiple layers L of artificial neurons. The goal of a feedforward network, as to other Ml algorithms, is to approximate some function $\hat{f}$ [17]. In terms of a classifier, $y = \hat{f}(x)$ maps an input x to a label y. Consequently, a neural network with m input nodes and 1 output node serves as a function with m inputs and 1 output.

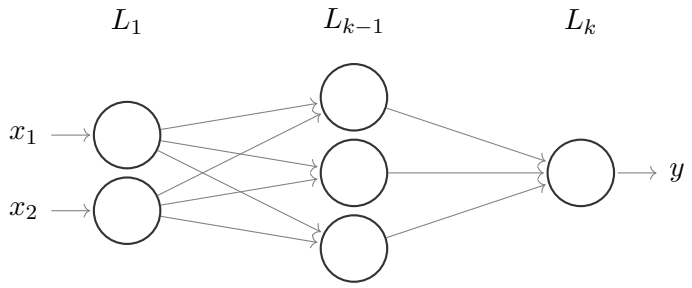


Figure 3.3: The diagram shows the typical structure of a feedforward network. The network has two input units in the input layer L_1 and one output unit in the output layer L_k. Layers $L_2 \ldots L_{k-1}$ the the so-called hidden layers.

The network is named feedforward as the information flowing through the network passes with the outermost input layer and and ends at the output unit of the output layer [17]. All units in a layer are fully connected to the succeeding layer, however, there is no interconnection between units in the same layer. Figure 3.3 shows a typical feedforward architecture with three layers including a single hidden layer L_{k-1}.

Considering equation 3.13, we can now compute the input to the output node, given by

$$\sum_{k=1}^{n} \alpha_k h_k \tag{3.17}$$

where h_k is the output of the kth hidden node and α_k being the weight from the kth hidden to the output node. The output unit's activation function is then applied to this value,

transforming the output to the given equation

$$y = \varphi\left(\sum_{k=1}^{n} \alpha_k \varphi\left(\sum_{i=1}^{m} w_{ik} x_i + b\right)\right). \tag{3.18}$$

.

3.5.4 Backpropagation

The purpose of the backpropagation algorithm [48] is to adjust the synaptic weights of the network to approximate the function $\hat{f}$ mapping the inputs x of the training data to the corresponding output labels y. In an iterative process the algorithm computes the overall error of the functional mapping and adjusts the weights of each neuron according to its individual error contribution. The result of this algorithm is a neural network configured to sensibly respond to unseen inputs for the specific supervised learning task.

As a first step, the weights of the network are initialized which is usually done in a random manner or based on a certain heuristic. Then each input pattern p, with features $p = \{x_0, x_1, x_2, \ldots, x_n\}$ and label y, is then sequentially processed, layer by layer, by the network in two phases. In the first phase, the *forward phase*, the ouput of the network is computed. The square error for the output nodes j is calculated as follows, where $\hat{y}_j$ denotes the output node's generated output and y_j is the desired ouput (see [19]):

$$E = \frac{1}{2} \sum_j (\hat{y}_j - y_j)^2. \tag{3.19}$$

Continuing in the *backward phase*, the network measures how much each neuron in the output layer L_k has contributed to each output neuron's error. Furthermore, as to measure the error contributions coming from each neuron in the previous layer, this step is repeated until the input layer is reached and all error contributions, the gradients, are computed. To put this in other words, the *backward phase* measures the error gradients across all connection weights in the network by propagating the error gradients back into the network (See [36]).

In this iterative process, the error gradients of the error function are calculated based on the partial derivative with respect to each connecting weight. If we define o_j as output of a neural unit with

$$o_j = \varphi(net_j) \tag{3.20}$$

and the units input as

$$net_j = \sum_{j=1}^{n} x_i w_{ij}, \tag{3.21}$$

the chain rule can be applied in order to compute the partial derivatives as follows:

$$\frac{\delta E}{\delta w_{ij}} = \frac{\delta E}{\delta o_j} \frac{\delta o_j}{\delta net_j} \frac{\delta net_j}{\delta w_{ij}} \tag{3.22}$$

Given the partial derivative in respect to the weight w_{ij}, the change of the weight Δw_{ij} can be determined. Here, the weight update equation (3.22) is computed depending on two cases. Either if the node j is in the hidden layer or in the output layer (See [36]):

$$\Delta w_{ij} = -\eta \frac{\delta E}{\delta w_{ij}} = -\eta \delta_j o_i \tag{3.23}$$

$$\delta_j = \begin{cases} \varphi'(net_j)(o_j - \hat{y}_j) & \text{node } j \in L_k \\ \varphi'(net_j) \sum_k \delta_k w_{jk} & \text{node } j \notin L_k \end{cases} \tag{3.24}$$

In equation (3.23), η denotes the *learning rate*, which defines to which amount the reverse gradients are applied to the weight update; k refers to a node in the successor layer of the node j.

Once the weights are updated, the global error of the network is calculated and the forward and the backward phase are repeated for the rest of the training patterns. One pass through all of the training patterns is called a *training epoch*. Several strategies exist to stopping the described training process. One condition is to stop after a fixed number of training epoches, a second can be a stop once the change in the weights reaches a certain low-end threshhold [19]. After the process, the final values of the weights are saved and can be used for predicting new incoming patterns.

4 Data Acquisition System

In this chapter, I will introduce TapSensing. TapSensing is a labeled data acquisition system designed to collect user generated taps with corresponding sensor readings. The chapter begins with a broader view on the system by explaining the overall system architecture and will then proceed into its individual components and implementation.

4.1 Overall System Architecture

The TapSensing application consists of two main components: the mobile and the server-side application. In brief, the mobile client provides the ability for a user to generate taps with corresponding motion sensor signals. For the data to be stored in a centralized manner, the server-side application provides HTTP endpoints as a gateway to the database. The architecture, as illustrated in figure 4.1, consists of various components which are outlined in the following:

- **Mobile application**: The iOS application provides interfaces for the user to generate taps and to label the data created. Furthermore, the application is capable of sending the acquired data to the server-side application. More features of the mobile application are presented in section 4.2.
- **NGINX**: For accepting and routing incoming HTTP requests, an NGINX reverse proxy/load balancer is used. The reverse proxy forwards the incoming requests to the server-side application and is capable of serving static content, such as images, HTML, CSS and JavaScript files of the form application.
- **Gunicorn**: Gunicorn[1] is a Python Web-server Gateway Interface (WSGI) HTTP server which runs the source code of the backend application.
- **Backend Application**: The backend application provides authentication and persistence functionalities which are accessible through HTTP endpoints. More information on the backend application is to be found in section 4.3

[1] For more information on Gunicorn, visit `http://gunicorn.org/`.

- **PostgreSQL**: TapSensing uses a PostgreSQL[2] database for storing the application state, user related information, the survey data and the retrieved tap and sensor information.
- **Form application**: The form application provides a web user interface where study participants can answer survey questions.

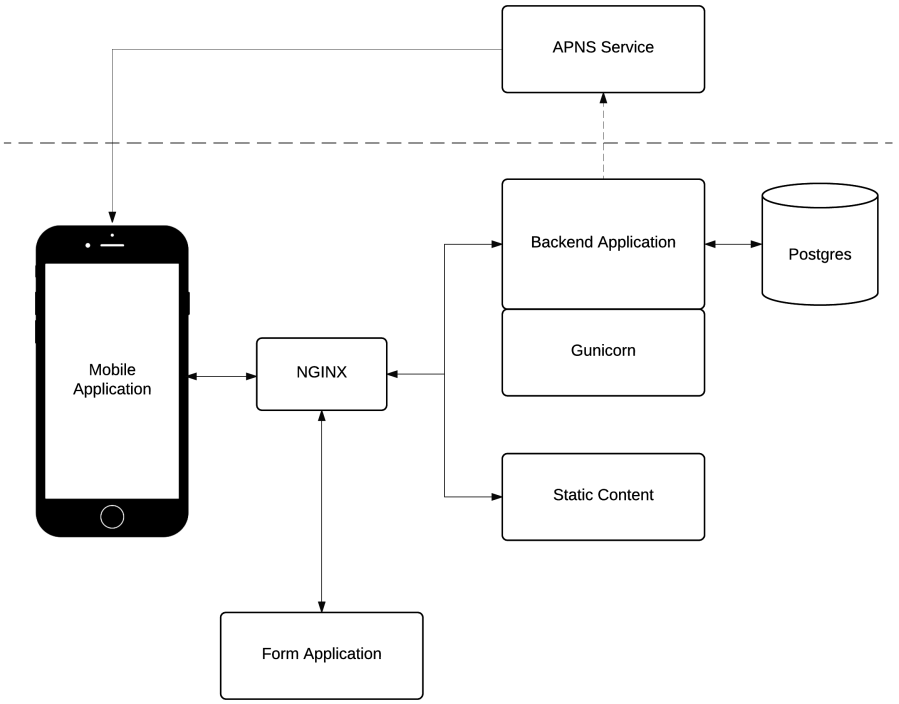


Figure 4.1: The diagram shows the overall architecture of TapSensing.

4.2 Mobile Application

The TapSensing mobile application is an iOS App written in the Swift programming language designed to run on devices with iOS 10 and above. The main purpose of the application is to provide a user interface for subjects to create taps with accelerometer and gyroscope readings for the labeled data acquisition phase of the experiment.

[2] PostgreSQL is a general purpose and object-relational database management system.

4.2.1 User Interface

Login Screen

When the application is opened for the first time, a login screen appears. As in standard login screens, the interface asks for credentials including username and password. Authenticating users, has the advantage, that the generated data can be mapped to individual users automatically.

Start Screen

During the trial the user is asked to generate taps once a day. In order to indicate if the user is eligible to perform a tap generation trial, the start screen shows a button that is either active or inactive. This switch depends on 4 distinct conditions: When data is col-

Figure 4.2: This figure shows the start screen in different configurations. On the left hand-side screenshot, the user has not performed a trial while the the middle image shows the screen where a trial has been performed. The right-hand side image shows the *lab mode* where trails can permanently be performed.

lected in the laboratory environment, the app is set to *lab mode*. In *lab mode* the button is always active and trials can be performed. When a user has not performed a trial today, the button is inactive. Consequently, when a user has already performed a trial on a specific day, the button is inactive and a further trial can only be performed on the following day. Once all field trials are performed, the app confirms that all data is collected and the button remains inactive.

Tap Input Screen

To acquire individual user taps the mobile application offers a user interface where buttons are aligned in a grid shape structure. The structure is calculated based on a specific configuration set where the amount of the vertical and horizontal buttons in the interface can be set. Figure 4.3 shows the interface, where 4, 12 and 20 distinguishable buttons are configured. To make it easier for the user to tap on every location of the screen

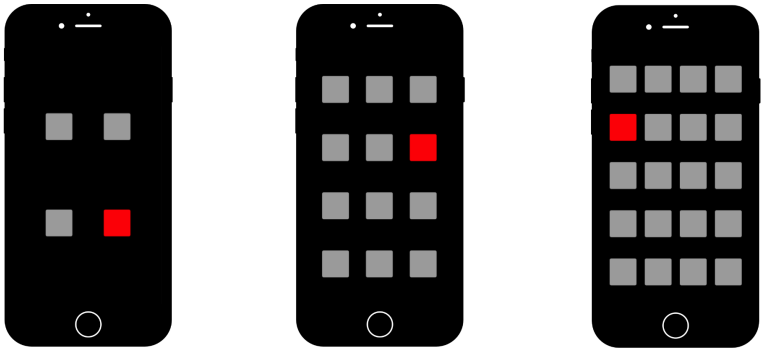


Figure 4.3: The figure shows the tap input user interface with buttons aligned in a grid shape structure. The leftmost structure offers 4 buttons, the middle offers 12 buttons whereas the right displays 20 distinguishable buttons.

exactly once, a red button indicating the next button to tap is highlighted guiding the user through the interaction process. While the user is tapping the grid, the gyroscope and accelerometer information is recorded. After all buttons have been tapped, either a new grid is loaded or the tap acquisition phase ends proceeding with the question interface.

Questions Screen

To label the data acquired in the tap input interface, the application provides screens for the user to answer several questions. These questions regard the body posture, input modality and the hand used to generate the taps. In the table 4.1 below the questions asked with corresponding answer choices are to be found.

To enable fast interactions, each answer possibility to a question is represented in the interface with an icon. Once an icon has been pressed, the application transits to the next question until all questions have been answered.

Figure 4.4: The figure displays question views with icons as answer possibilities.

Question	Answer Choices
Which body posture was used during the interaction ?	Standing, Sitting
Which finger did you use while tapping?	Index Finger, Thumb
Which hand did you use to tap?	Left, Right

Table 4.1: The table shows the questions asked in the question view.

Upload Screen

After all taps and questions are gathered, the acquired data is sent to the server. The interface at this point displays a spinning wheel for the user to acknowledge that the mobile phone is processing the data. In case an upload fails, the application provides a manual upload screen where past sessions can be uploaded.

4.2.2 Additional Features & Implementations

Obtaining Sensor Information

In order to access the gyroscope and the accelerometer, Apple provides a high-level API[3] for accessing the device's sensors: *Core Motion*. Core Motion provides motion and environmental related data from sensors including accelerometers, gyroscopes, pedometers, magnetometers, and barometers in easy-to-use manner.

[3] An Application Program Interface is a set of rules and subroutines provided by an application system for the developer to use. The following link leads to the Core Motion API documentation: `https://developer.apple.com/documentation/coremotion/`

Sensor values can either be accessed as proceeded including aggregations of the values or as raw version. For TapSensing, raw values are recorded to avoid any form of bias. The update interval can be configured at ranges from 10Hz - 100Hz. Higher update-rates are possible but not ensured to be processed in real-time by the device. For TapSensing, the update rate is configured with the highest (safe) value possible. This ensures that tap patterns are captured with high resolution in order to make a classification of the lap location easier.

Local Persistence

It is possible that a session upload fails due to lack of internet access or other reasons. Due to this, the application stores all session information and sensory data in a local SQLite database. For local persistence Apple provides its own framework called *Core Data* which is extensively used in TapSensing. Once the data has been successfully received by the backend, the data is deleted from the local database.

Ensuring data consistency

Sending all collected data in a single HTTP-request results in the sent package being too large for the server-side system to process. For this reason, the data is split up into packages of 300 objects per request. To send these requests in an asynchronous manner, the *Promise*[4][32] library *Hydra*[5] is used.

During an upload phase, packages are sent in a sliding window approach. Packets are sent three at a time until all packages have been acknowledged by the server-side application. For each transmitted package, the server responds with the amount of data objects contained within the request. Therefore, The mobile client can track the amount of packages transmitted to ensure that all data has been transmitted successfully. If one packet fails to arrive, the package is resent with an exponential back-off.

Push Notifications

TapSensing is registered with the Apple Push Notification Service allowing the application to receive push notifications. Notifications are used to remind the user during the field study that she/he has to take part in the study.

[4] Promises are a software abstracting for dealing with asynchronous computation. Promises are objects that may produce a value at some point in the future.

[5] `https://github.com/malcommac/Hydra`

Distribution

Installing iOS applications that are not uploaded to the Apple App Store, requires each device to be registered in the Apple Developer Portal with the smartphone's serial number. To avoid this, the application has been uploaded to the App Store enabling an easy distribution.

4.3 Backend application

The purpose of the backend application is to provide persistence functionalities for the acquired taps generated with the mobile application. The application is written on top of the Django[6] web application framework and the Django REST framework[7].

4.3.1 HTTP Endpoints

For the purpose of interoperability the network communication from the mobile application and the forms application to the server-side application is done via HTTP Requests in the JSON[8] format. The endpoints listed below represent tiny logic components that can be called from an external client.

4.3.2 Data Model

TapSensing's data model reflect the schemas that are used in the PostgreSQL database. As seen in figure 4.5, the data model consists of 4 data objects that are described in the following:

- **User**: The user model is inherited by the Django's user model[9]. The user model is used for authentication and persisting the authentication token. The other models described in this section are connected to the user model through a foreign key relation to identify the user associated with the data object.

[6] `https://www.djangoproject.com/`
[7] `http://www.django-rest-framework.org/`
[8] JSON (JavaScript Object Notation)[10] is a lightweight data-interchange format.
[9] For more information on the Django user model, the following URL leads to the model's documentation: `https://docs.djangoproject.com/en/1.11/ref/contrib/auth/`

HTTP Method	URL	Description
POST	/login	Provides login functionalities for the mobile client. This method returns an authentication, that is used for further requests to authenticate the user.
POST	/session	Provides upload functionality for a session data objects.
POST	/touchevent	Provides upload functionality for touchevent data objects.
POST	/sensordata	Provides upload functionality for a sensordata data objects.
POST	/apns	Retrieves the Device's APNS token. This is required to send push notifications to the users.
GET	/trial-settings	Provides the configuration for the tap input view. Here, the amount of buttons and the amount of grid repetitions that are to be performed in a single session can be defined.
POST	/survey	Provides upload functionalities for the survey form application.

Table 4.3: The table shows all HTTP endpoints of the server-side application.

- **Session**: The session data object holds all information associated with the user's trial. This includes the data collecting in the "labeling part"[10] of the application such as the hand used, the body posture and the typing modality. In addition, information is stored such as device info and if the session took place in a laboratory or field environment.
- **Touchevent**: The touchevent data object stores information regarding the "ground truth" of the user generated tap including the exact x and y coordinates, timestamp and an identifier of the specific grid rectangle tapped in the tap input view. Furthermore, it is noted if the user hit a specific rectangle and if the event is a touch-down or touch-up event.
- **Sensordata**: The sensordata data object captures all information obtained by the gyroscope and accelerometer of the mobile device. To differentiate between accelerometer and gyroscope values, the data object includes a type field. In addition,

[10] The "labeling part refers to the question views, where additional information on the data acquired is collected"'

the model captures the timestamp and the 3-sensor components (x, y, z) of the individual sensors.

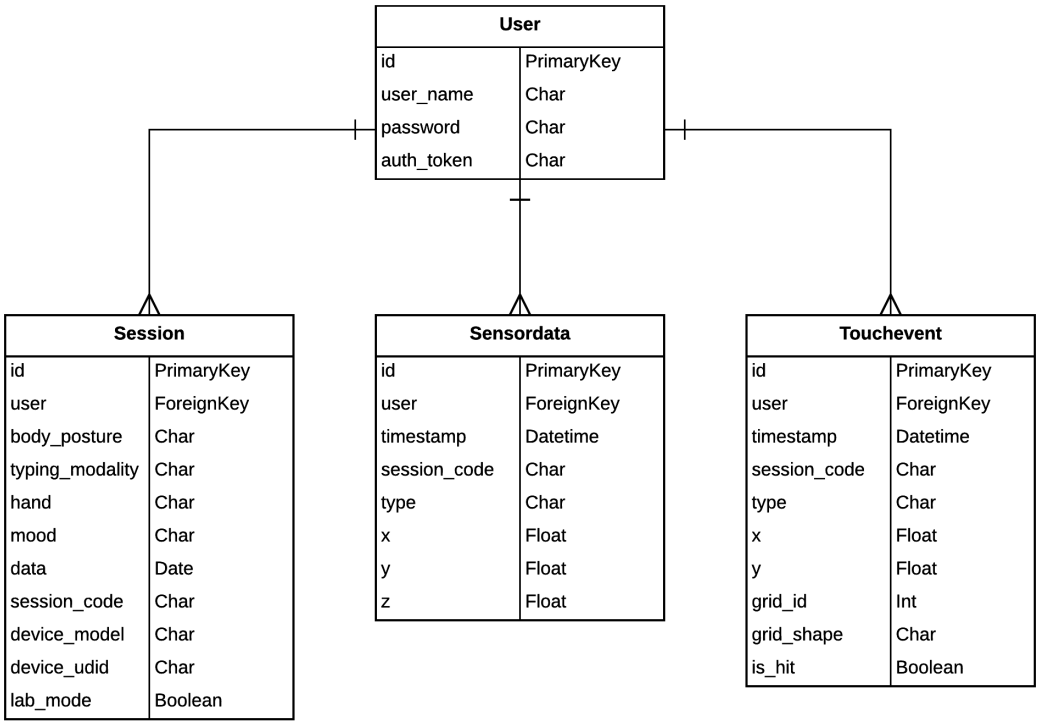


Figure 4.5: The diagram shows TapSensing's data model with the user, session, sensordata and touchevent data object.

4.3.3 Additional Features & Implementations

Authentication

The authentication mechanism implemented in TapSensing is based on a standard token authentication scheme. This allows the association of an incoming request with a set of identifying credentials, which is - in this context - the user the request initially came from. In order to obtain an authentication token, the mobile application sends the login credentials of a user including username and password to the login endpoint[11]. When the credentials have been checked for validity, the endpoint returns an authentication token in the following format:

[11] The login endpoint is described in section 4.3.1.

```
{ "token": "Token 3acc6c2a58723e2f1579d4526add2511f6a0a525" }
```

The token is then added to the HTTP-request in the "Authorization" header to authenticate the user.

Push Notifications

In order to remind participants to take part in the field study push notifications are sent daily. As a previous study has shown that push notifications greatly enhance user participation [7] during mobile field studies, notifications are sent on a daily basis reminding the participants to take action. In the same study [7], passive notifications - notifications without sound - have been seen to furthermore positively impact the participation. Following these assumptions, a push notification strategy has been designed combining notifications with and without sound. Figure 4.4 shows this implemented strategy with corresponding trigger times.

Time	Message	Sound
9:00 GTM+1	Good Morning. This is a friendly reminder to take part in the tapsensing study today.	inactive
12:00 GTM+1	Tapping is a lot of fun. Have you tapped the buttons today?	inactive
18:00 GTM+1	I know your day is busy, but don't forget to take part in the study.	inactive
21:00 GTM+1	You have not taken part in the study today. Please do it now.	active

Table 4.4: The table shows the push notifications strategy with individual notifications sent.

The interaction with the APNS service has been implemented using the Django package *django-push-notifications*[12]. It is used to provide mechanisms to register devices as well as to send notifications. To trigger the notifications at specific times unix cronjobs are executed.

Backups

To prevent data loss, the PostgreSQL database is backed up via a unix cronjob every evening. The database dumps are sent automatically to Amazon Web Services S3 Bucket.

[12] `https://github.com/jleclanche/django-push-notifications`

Deployment

The server-side application with the NGINX reverse-proxy and a gunicorn WSGI application server has been deployed on a Ubuntu 14.04 virtual environment. The server comprises of 1GHz of shared CPU and 1 GB RAM.

5 Method

5.1 Hypothesis & Research Questions

To recall, previous research has not shown that it is possible to predict tap location on smartphone screen for data collected in a field environment. Due to this reason, data will be collected from both the field and the laboratory environment to investigate how classifiers, when trained with data from these environments, perform. Initial assumptions are made that the environment has an effect on the classifier's prediction accuracy.

H.1 The environment of recorded sensor data has an effect on the prediction accuracy.

In order to test the hypothesis stated above, a further assumption is made assuming that when a subject interacts with a smartphone screen, for instance, while walking or during a public transportation ride, the recorded sensory data will contain distortions which will, most likely, have a negative impact on the prediction accuracies.

H1.1: The prediction accuracy for a classifier trained with the data in the laboratory environment will score higher than one trained with data collected in the field.

Moreover, assumptions are made on the way the user interacts with the device. A user can either use the thumb to touch or the index finger while holding the device in the other hand. Assuming that the input modality also has an effect on the behavior of the estimator, data sets for both hands will be evaluated.

H2: The input modality has an effect on the prediction accuracy.

It is plausible that tapping with the index finger, due to the force resistance of the supporting other hand, will cause less movement of the smartphone as typing with a thumb will. This is presumably to be reflected in the motion data.

H2.1: The prediction accuracy for a classifier trained with index finger tap data will score higher than one trained with thumb tap data.

Finally, assumptions are made based on the body posture a user has while tapping. Overall, a difference in classification results is assumed between standing and sitting.

H3: The body posture has an effect on the prediction accuracy.

To approve or reject this hypothesis, it is assumed that a user while sitting will move less compared to one who is standing which, as a result, will lead to a better predictability of the tap location.

H3.1: The prediction accuracy for a classifier trained with taps where a user sat will score higher than one trained with taps where a user stood.

As a more severe scenario arises if an attacker could scale out the attack to predict tap locations of unseen users, a cross user experiment will be performed as a further step. For this purpose, the question is posed if it is possible to infer tap location of unseen users in the field with a classifier trained on laboratory data from different set of users.

5.2 Experimental Approach

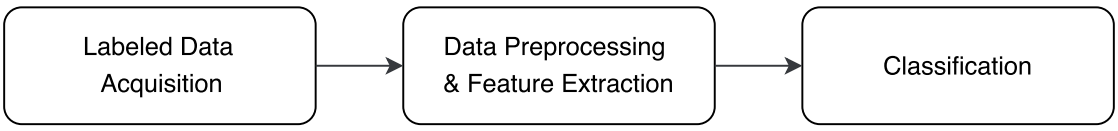


Figure 5.1: This figure shows the overall approach to the experiment.

The overall experiment consists of a three step process. In the first step, labeled data is acquired from subjects in the field and in the laboratory. For this purpose, the TapSensing application presented in the last chapter is used. The data acquisition part is a necessary step to collect sensor data with the corresponding ground truth of the tap locations for the supervised learning task.

After the data is successfully acquired, the continuos sensor recordings are pre-processed to obtain the portion of recording which represents each individual tap. To extract certain

characteristics of the sensor signature, feature are extracted in a further step. These features are then used to train a classifier.

In order to compare individual estimators, as for instance for the comparison between laboratory and field, a classifier will be evaluated using a k-fold cross-validation. The individual accuracy scores of the cross-validation are then compared by means of a Wilcoxon signed-rank test.

5.3 Labeled Data Acquisition

5.3.1 Participants

A total of 27 participants were invited to participate in the study, 12 (44%) females and 15 (56%) males. Participants had an average age of 26.4 years (Min=17, Max=53, SD=6.39) and all 27 were right-handed (100%). There were no restrictions concerning the demographics of individual subjects. However, each subjects had to be in possession of one of the permitted iOS devices.

5.3.2 Devices

The devices have been restricted to the Apple iPhone 6, 6s and 7 based on their mutual screen size. Furthermore, as the screen size has a large effect on the sensor signature created by a tap, the screen size is an important factor to enable the comparison of classification results among devices.

5.3.3 Environments & Conditions

As the study aims at collecting sensor reading both in the field as well as in a laboratory environment, the data acquisition is done in two distinct settings.

1. **Laboratory Evironment**: Participants were invited to a laboratory room which had a standard office ergonomics setup. Participants have been asked to either sit at the desk or stand in the room while tapping.
2. **Field Environment**: Participants were asked to generate taps using their smartphone at any place they are currently located. For example, this could be at home, at work or during leisure activity.

Besides the environment of the recorded sensor data, the collected data varies in the input modality and body posture. Participants are either allowed to use the index finger (while holding the device in the other hand) or the thumb to generate taps. Sitting and standing are allowed as body postures as these two represent the natural interactions with the smartphone.

Regarding the distribution of the input modalities and body postures, in the laboratory part of the data acquisition, the input modality and body posture will be instructed for an equal distribution of both variables. In the field acquisition, however, the body posture and input modality can be decided by the participant in order to gain a more natural distribution of both factors.

5.3.4 Survey

In a short survey participants are asked about their basic demographics. This includes the age, gender, current occupation and if they are left or right-handed. In a further step, questions are asked regarding their personal smartphone usage. It is asked which device model they own, which input modalities they generally use and which input modality they use most often. Tapping with the index finger, tapping with the thumb or typing with both thumbs are considered as answer possibilities for the question concerning the input modality.

5.3.5 Acquisition Procedure

In order for participants to generate tap data, the subjects were invited to come to the laboratory for the first part of the experiment. To avoid information asymmetry, each participant read an experiment instruction. Subjects should then confirm whether they have understood the previously read information. Additional questions regarding the instructions were answered. In the next step, subjects were asked to fill out an online questionnaire regarding their persona and personal smartphone usage.

To begin with the data acquisition, participants were asked to download the application from the Apple App Store and to login using provided credentials.

Subsequently, subjects were asked to perform 6 consecutive trials in the TapSensing application, whereas one trail includes tapping each grid four times in randomized order. Recall, the grid sizes defined consist of 4, 12 and 20 distinguishable buttons. For an equal

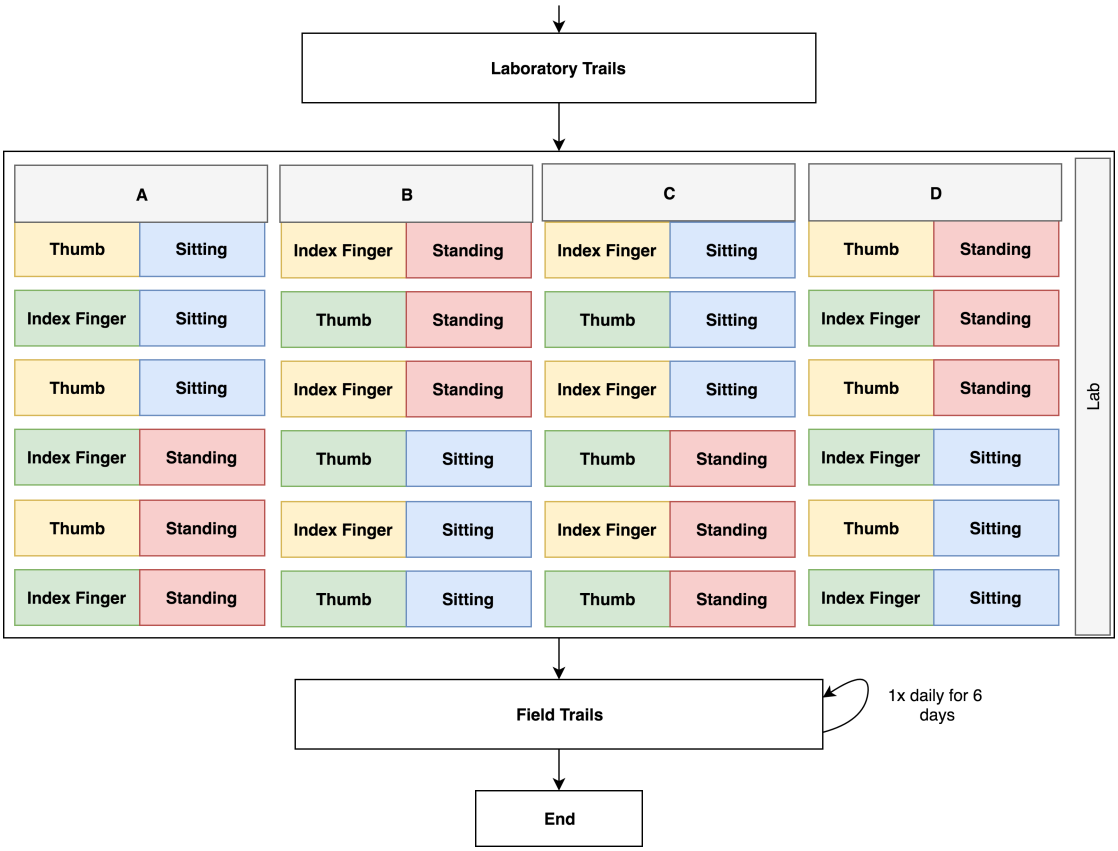


Figure 5.2: This figure shows the procedure every subject has to perform in the study.

distribution of the body posture and the input modality while tapping, each trial is dependent on one of four configurations onto which each subject is assigned. These configurations are to be seen in Figure 5.2 and have the advantage to minimize any learning effect the user has with the application. It is important to note that each subject is not allowed to alter the body posture or input modality during a trial. After all trials are performed, subjects were asked to continue with the field study.

During the field study, subjects performed one trial daily on 6 separate days. On each day push notifications were sent as a reminder to participate. Subjects were free to decide which input modality or body posture to use as the aim of the field study is to collect data that represents regular smartphone usage.

The whole data acquisition took approximately one hour for each participant, which is split into 30 minutes in the laboratory and another 30 minutes for the data acquisition on the 6 distinct days in the field.

5.4 Data Preprocessing

5.4.1 Preprocessing

Before features can be extracted from individual taps, the continuous sensor recording from each trial needs to be sliced to obtain the portion of the recordings which is relevant for the tap. The timestamp of the touchdown event is used to find an appropriate starting point. Here, 20ms are subtracted from the timestamp as due to the latency between the physical touchdown event and the recorded event. Based on the data collected in a pilot study, the average tap duration (being the duration between a touchdown and the corresponding touchup event) is between 70ms - 200ms. Taking this value into account, a window size of 150ms was chosen to enrich each slice with more information. Figure 5.3 shows the sensor components of a gyroscope reading with a slicing window marked with a grey background.

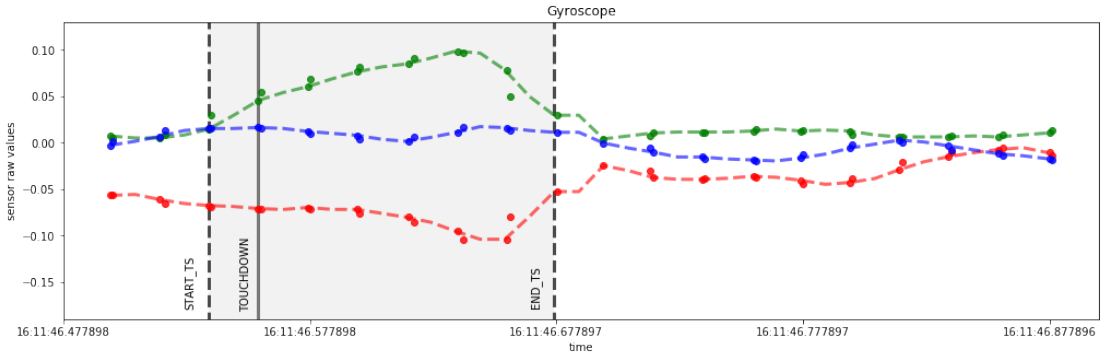


Figure 5.3: The figure shows a continuos gyroscope reading with the slicing window and corresponding timestamps. The timestamp of the touchdown event is used as an anchor point.

Unfortunately, due to different CPU loads on the mobile devices, sensor values are not pushed from the operating system to the mobile application in a constant manner leading to an uneven distribution of the sensor recordings on the time scale. To balance out each tap to a fixed amount of values, a cubic spline interpolation is used. Figure 5.3 shows the interpolated sensor components with dots representing the raw recordings. Furthermore, recordings that are below 75Hz are filtered out. After the slices are produced for each tap, features are extracted.

5.4.2 Feature Extraction

A feature is an individual measurable property or characteristic of a phenomenon being observed [14]. Therefore, choosing discriminant and independent features is a crucial step for the performance of the later classification. Fortunately, Miluzzo et al.'s preliminary work has shown a comprehensive list of possible sensor features of which the following features are partially adapted.

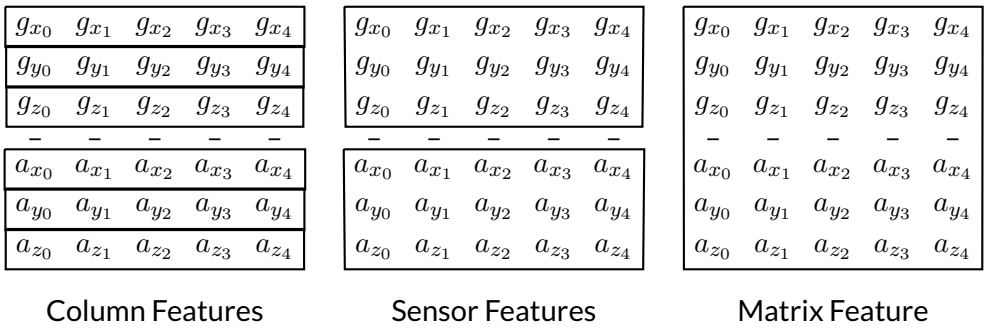


Figure 5.4: The figure shows how different features are extracted from the overall sensor matrices: Column features, sensor features and matrix features.

The features extracted are differentiated based on the set of axis where aggregational functions are applied. These can be categorized as column features, sensor features and matrix features, as illustrated in figure 5.4. Column features refer to features that are a product of a function being applied to each *x*, *y* and *z* sensor axis separately. Sensor features are defined as features where sensor axis are combined. An example for a sensor feature is the Pearson correlation which is calculated for each sensor component pair *xy*, *xz* and *yz*. With sensor features, relations between the components are captured. Lastly, matrix features are a product of a function being applied to both gyroscope and accelerometer time series.

Since taps on different locations of the screen generate different sensor signatures we design features that are able to capture the properties of the sensor readings generated by a tap. For this purpose, a total of 230 features have been extracted for each individual tap. The table 5.2 below shows the complete list of features ordered by feature type. Features have been extracted from both the time domain and the frequency domain. In addition, to standardize the range of the features extracted, a Min-Max scaling is applied to each feature.

Name	Description	Feature Type	Amount
peak	Amount of peaks in the time series	column	6
zero_crossing	Amount of zero crossings in the signal	column	6
energy	Energy of the signal	column	6
entropy	Entropy measure of the the signal	column	6
mad	Median absolute deviation of the signal	column	6
ir	Interquartile Range	column	6
rms	Root mean square of the signal	column	6
mean	Mean of the time series	column	6
std	Standard deviation of the time series	column	6
min	Minimum of the time series	column	6
median	Median of the time series	column	6
max	Maximal value of the time series	column	6
var	Variance of the time series	column	6
skew	Skewness of the time series	column	6
kurtosis	Kurtosis of the time series	column	6
sem	Standard error of the time series	column	6
moment	Moment in the time series	column	6
spline	Spline interpolation of the signal	column	6*12
fft	Fast Fourier Transform	column	6*5
cos_angle	Cosine Angle of sensor component pairs	sensor	6
pears_cor	Pearson Correlation of component pairs	sensor	6
fro_norm	Frobenius matrix norm	matrix	1
inf_norm	Infinity matrix norm	matrix	1
l2_norm	L2 matrix norm	matrix	1

Table 5.2: Table of features extracted from every tap.

5.5 Classification

After the features are extracted for all the tap data acquired, learning algorithms are applied in order to measure the classification accuracy. In this experiment, a SVM with radial basis kernel is used as well as a feedforward artifical neural network.

5.5.1 Evaluation

To evaluate the classifiers trained in the classification part of the experiment, a K-fold cross-validation is used. In K-fold cross-validation, the training set is divided into K subsets of equal sizes. Sequentially, every subset is tested using the classifier trained on the remaining K-1 subsets. During this process, each pattern in the data set is predicted once. After K classifiers are trained, the mean of the accuracy scores is computed to determine how well the classifier performs.

5.5.2 Grid Search

In order to optimize the hyperparameters of each learning algorithm, a grid search is performed. A grid search is an exhausting search through a defined subset of the hyperparameter space for each algorithm. The subset of parameters are combined using the cartesian product in order to configure the classifier. To evaluate each classifier with a set of hyperparameters, a 5-fold cross validation is performed.

The parameters C and γ of the SVM RBF kernel are tested during the grid search using exponential growing sequences, a recommended method to find suitable parameters [21].

$$C = \{2^k | k \in \{-3, -2, ..., 14, 15\}\} \tag{5.1}$$

$$\gamma = \{2^k | k \in \{-13, -12, ..., 2, 3\}\} \tag{5.2}$$

As for the SVM, hyperparameters are defined for the ANN. Presumably the most crucial parameter in an ANN is the amount of hidden layers defined. When a network is too large it tends to overfit the training data while a too small network can lead to high bias. Therefore, networks ranging from 2 to 12 hidden units are defined and evaluated during the grid search. To regularize the networks an L_2 penalty [41] is used. The following table 5.3 lists all configurations of the evaluated ANNs.

Hidden units	L2 Penalty	Learning Rate	Activation	Optimizer
12	0.001	0.01	RELu	Adam [23]
10	0.003	0.01		
8	0.0001	0.1		
4	0.0003			
2				

Table 5.3: ANN configurations during grid search.

After the grid search is computed, the best performing model with corresponding hyper-parameters is used for analysis.

5.5.3 Metrics

As the amount of taps are equal for each subject and grid cell, the amount of training examples for each class is balanced. Therefore, the standard accuracy score for the classifier is used.

$$A = \frac{TN + TP}{TN + FP + TP + FN} \tag{5.3}$$

, where TN is the number of true negative cases, FP is the number of false positive cases, FN is the number of false negative cases and TP is the number of true positive cases.

6 Results

6.1 Data Acquisition

The data acquisition was performed with 27 participants in total, 12 (44%) females and 15 (56%) males. Participants had an average age of 26.4 years (Min=17, Max=53, SD=6.39) and all 27 were right-handed (100%). 19 (70%) were students, 8 (30%) had other occupations. 18 (66%) subjects stated that their most used input modality is the their thumb while 4 (14%) preferred using their index finger and 5(18%) use both thumbs during interactions.

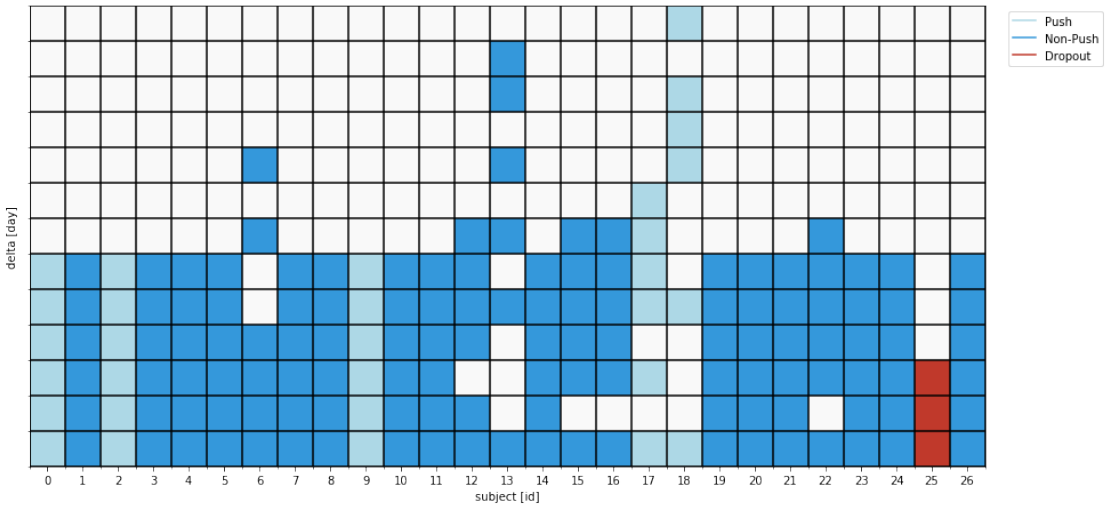


Figure 6.1: The figure shows on which days the participants took part in the field study. Light blue participants which did not receive push notifications while participants with dark blue marks had received push notifications. Red rectangles indicate a dropped out participant.

In regards to the participation during the study, 26 subjects managed to finish all laboratory and field study trials while 1 subject dropped out in the field study. Figure **??** shows on

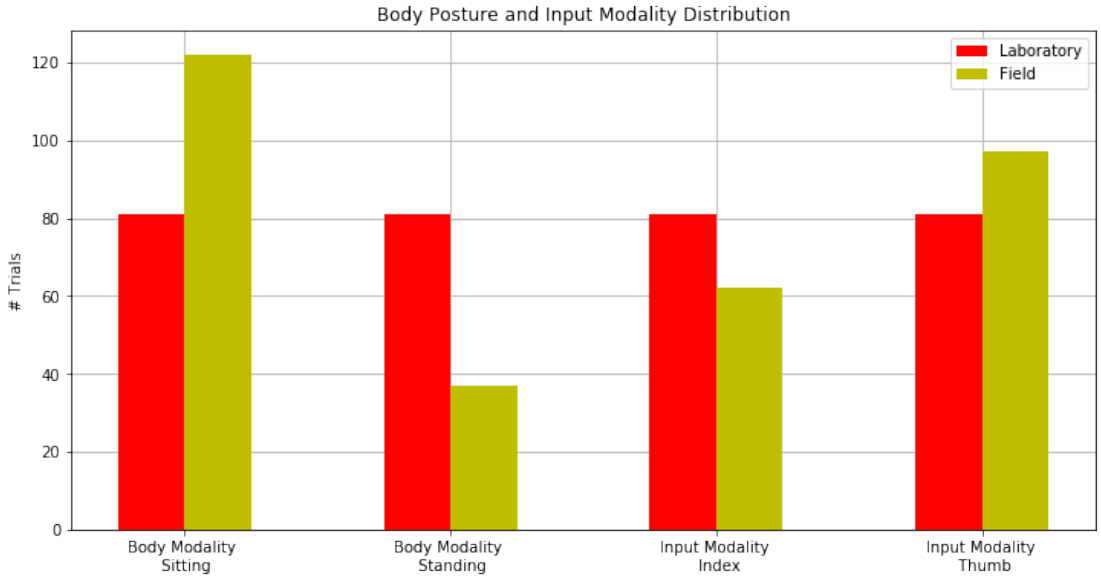


Figure 6.2: The bar chart shows the distribution of the input modalities and the body postures used in the field and the laboratory trials.

which day individual participants performed trails. 4 participants did not accept push notification and were therefore not reminded on a daily basis.

In total over 46.000 taps were generated in the whole data acquisition phase. To be more precise, approx. 25.000 taps were collected on the 5x4 grid, over 15.000 taps on the 4x3 grid and more than 5.000 taps on the 2x2 grid. Concerning the distribution of the body posture and the input modality, subjects used their thumb more frequently compared to the index finger and sat more often during the trails compared to the standing posture. The distributions are illustrated in figure 6.2.

The devices used to obtain the tap information were 12 (45%) Apple iPhone 6s, 10 (37%) iPhone 6 and the least common device was the iPhone 7 with 5 (18%).

In order to visualize the data collected, a t-Distributed Stochastic Neighbor Embedding (t-SNE) is shown in figure 6.3 where the 230-dimensional feature vector has been reduced to 2 dimensions. Furthermore, a plot showing the interpolated gyroscope and accelerometer signals acquired during a single tap generation trial is to be found in the Appendix.

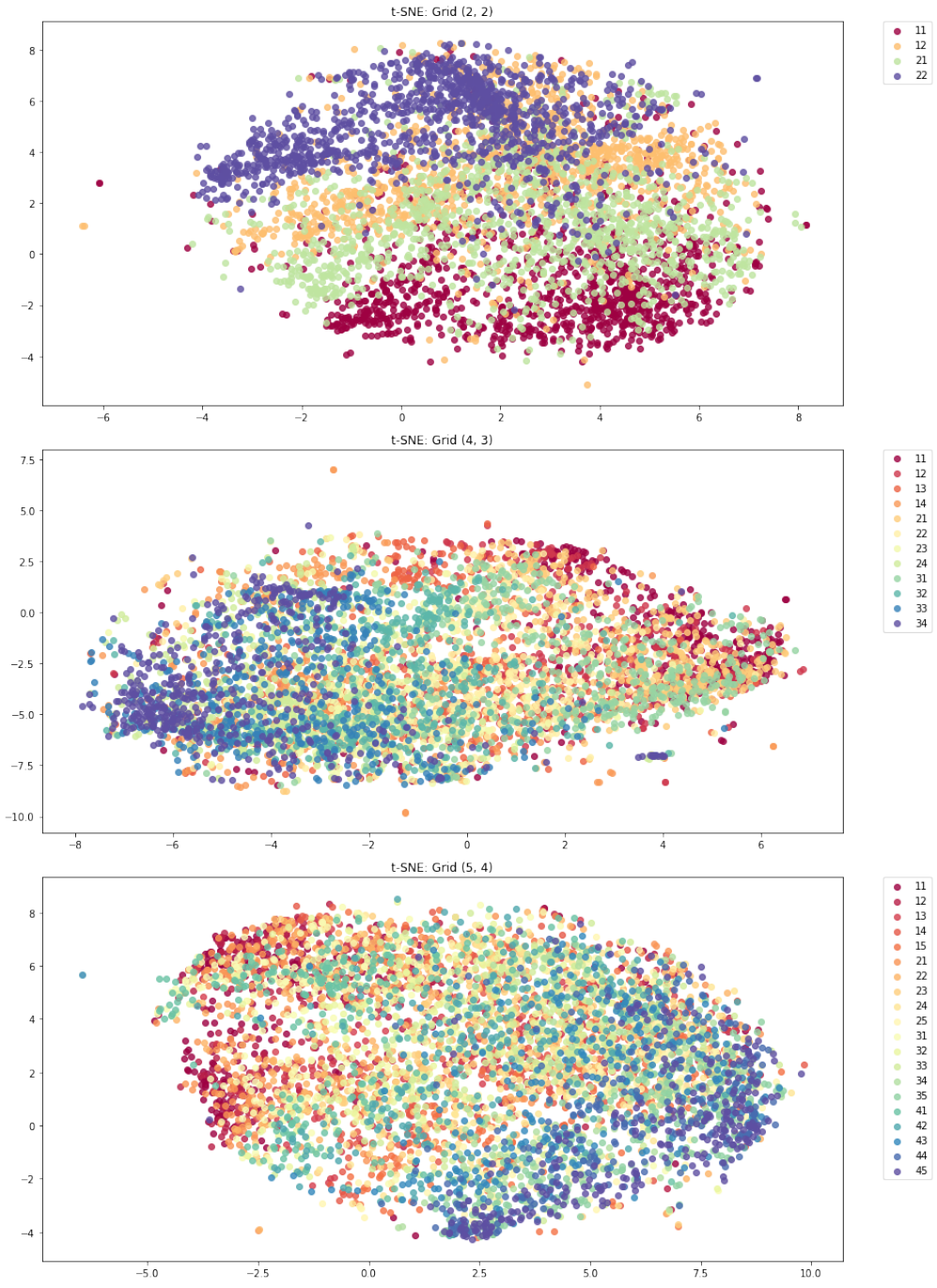


Figure 6.3: Visualization of 230-dimensional feature vectors reduced to 2 dimensions. The used t-SNE dimensionality reduction technique is unsupervised, thus it does not consider the labels during the optimization

6.2 Laboratory and Field Comparison

For the analysis between the controlled and the uncontrolled environment, a subset of the overall training material has been filtered based on the environment, grid size and the mobile device. After performing a grid search of the hyperparameter space for the SVM and, likewise, for the ANN, a 10-fold cross validation has been performed on the best of the two estimators yielding the highest mean accuracy.

2x2 Grid

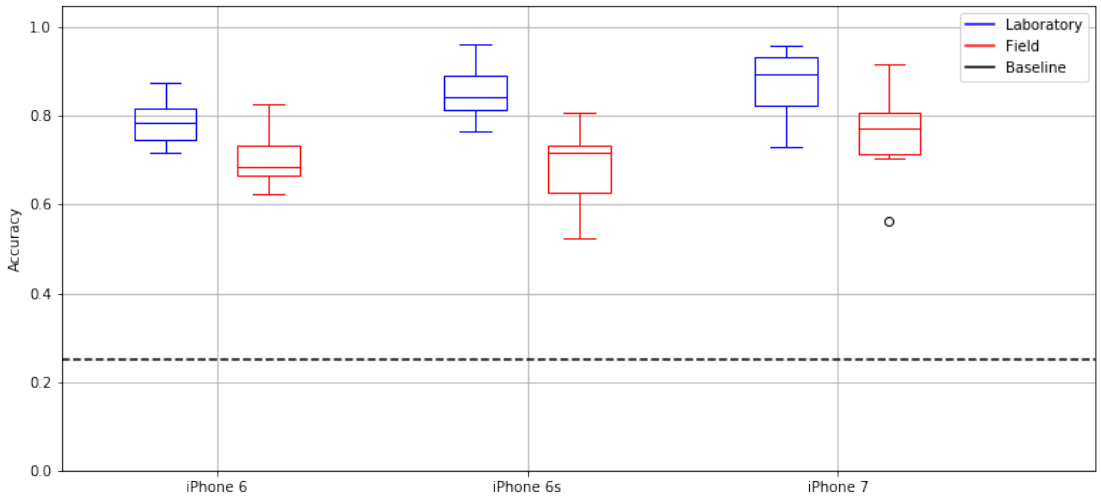


Figure 6.4: The figure shows the tap inference accuracies for the 2x2 grid of the 10-fold cross-validation. The measured classification accuracies are above the guessing probability of $\frac{1}{4} = 25\%$.

For the 2x2 grid, the results show mean accuracy measures in range 0.79 to 0.87 for the laboratory environment and ranges 0.68 to 0.76 for the field environment. Furthermore, the iPhone 7 data scores highest with a mean accuracy score of the 0.85 for this particular classification problem.

Moreover, across all devices, the results list that the mean accuracies for the field data are always lower compared to the laboratory data. The fact that the classification measures for both environments differ significantly is confirmed by a Wilcoxon signed-rank test yielding that the fold accuracies in the laboratory were significantly higher than the fold accuracies in the field environment $Z = 5$, $p < 0.05$.

		Accuracy				
Device	**Environment**	mean	min	max	std	**Classifier**
iPhone 6	Laboratory	0.79	0.72	0.87	0.05	SVM
	Field	0.71	0.62	0.83	0.06	SVM
iPhone 6s	Laboratory	0.85	0.77	0.96	0.06	SVM
	Field	0.68	0.52	0.81	0.09	SVM
iPhone 7	Laboratory	0.87	0.73	0.96	0.08	SVM
	Field	0.76	0.56	0.92	0.09	SVM

Table 6.1: Classification results for the 2x2 tapping grid. Notably, the SVM outperforms the ANN for this task.

4x3 Grid

For the 4x3 grid, the analysis shows mean accuracy scores in range 0.46 to 0.59 for the laboratory environment and ranges 0.40 to 0.47 for the field environment. Furthermore, the iPhone 7 data scores highest with a mean accuracy score of the 0.59 for this 12-class problem.

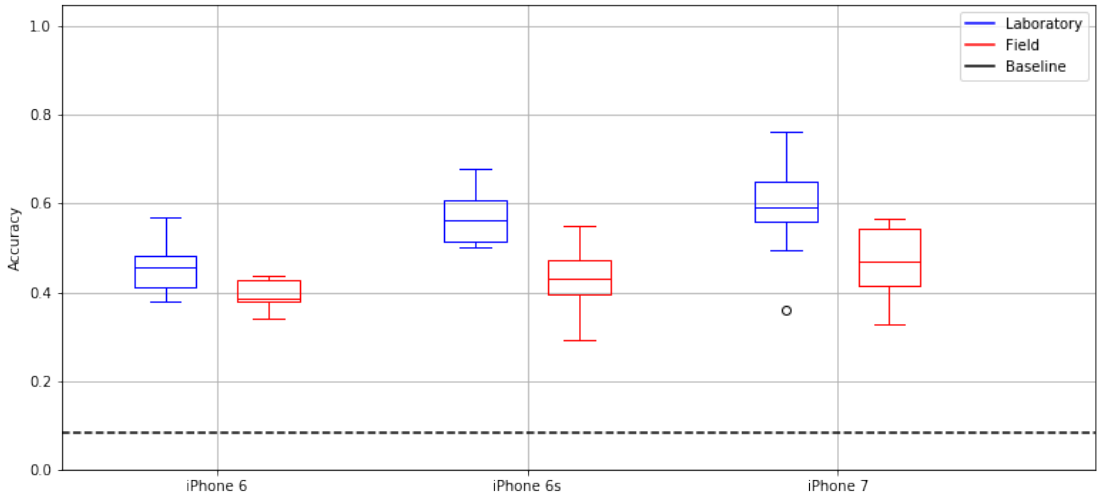


Figure 6.5: The figure shows the tap inference accuracies for the 4x3 grid of the 10-fold cross-validation. The measured inference accuracies are above the probability baseline of guessing ($\frac{1}{12} = 8.33\%$ for the 12 distinguishable buttons).

A performed Wilcoxon signed-rank test shows that the classification results for both environments differ significantly. The fold accuracies in the laboratory were statistically higher than the fold accuracies in the field environment $Z = 19$, $p < 0.05$.

		Accuracy				
Device	**Environment**	mean	min	max	std	**Classifier**
iPhone 6	Laboratory	0.46	0.38	0.57	0.06	SVM
	Field	0.40	0.34	0.44	0.03	SVM
iPhone 6s	Laboratory	0.57	0.50	0.68	0.06	SVM
	Field	0.43	0.29	0.55	0.07	ANN
iPhone 7	Laboratory	0.59	0.36	0.76	0.11	SVM
	Field	0.47	0.33	0.57	0.08	SVM

Table 6.2: Classification results for the 4x3 tapping grid.

5x4 Grid

For the grid with 20 distinguishable areas the inference accuracies measures range from 0.35 to 0.43 for the laboratory and 0.28 to 0.32 for the field data, respectively. Aligning with the previous results, the iPhone 7 shows highest mean accuracy scores of 0.43 for the data collected in the laboratory.

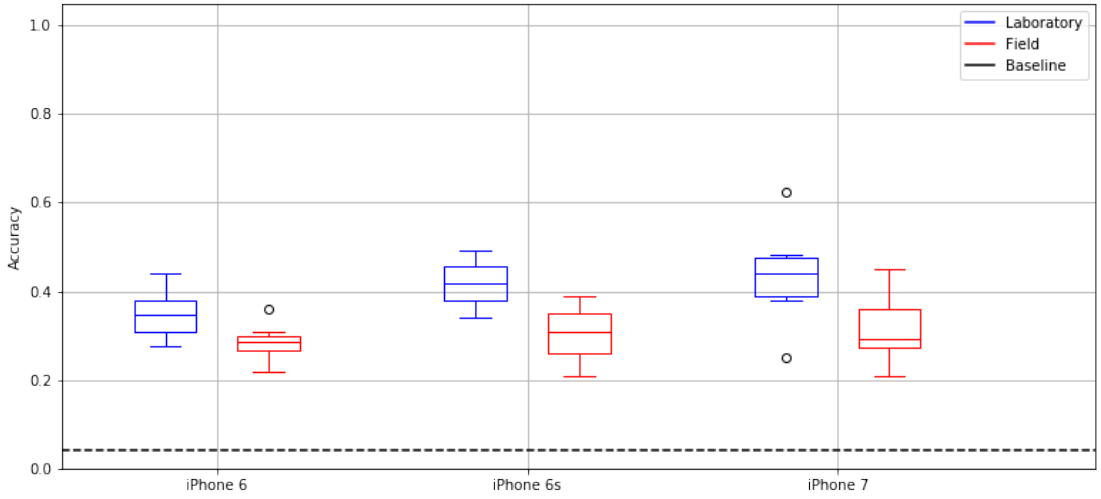


Figure 6.6: The figure shows the tap inference accuracies for the 5x4 grid of the 10-fold cross-validation. Results show that all inference accuracies are above the baseline of $\frac{1}{20} = 5\%$ for this classification problem.

Furthermore, the Wilcoxon signed-rank test shows that the classification results for both environments alter significantly. The fold accuracies in the laboratory were significantly higher than the fold accuracies in the field environment $Z = 12$, $p < 0.05$.

		Accuracy				
Device	**Environment**	mean	min	max	std	**Classifier**
iPhone 6	Lab	0.35	0.28	0.44	0.05	ANN
	Field	0.28	0.22	0.36	0.04	ANN
iPhone 6s	Lab	0.42	0.34	0.49	0.05	SVM
	Field	0.31	0.21	0.39	0.06	ANN
iPhone 7	Lab	0.43	0.25	0.62	0.09	SVM
	Field	0.32	0.21	0.45	0.08	SVM

Table 6.3: Classification results for the 5x4 tapping grid.

6.3 Input Modalities Comparison

As for the comparison between controlled and uncontrolled environments, the same classification experiment was performed to detect differences in the predictive models between the two input modalities: Index finger and thumb. As subjects were free to decide which input modality to use during the field study, the sample size has been adjusted in order to train each classifier with the same amount of training material.

Classifier trained for the individual grid sizes show similar results. For this reason, only the results for the 5x4 grid will be shown here. The results for the other grid sizes are listed in the Appendix.

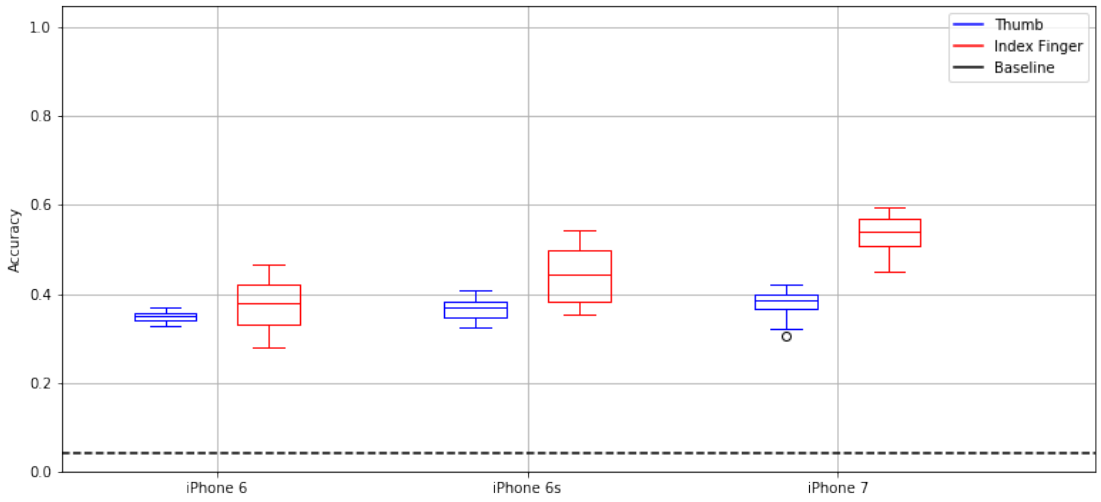


Figure 6.7: The figure shows the tap inference accuracies for the 5x4 grid of the 10-fold cross-validation.

For the 5x4 grid, the results show that across all devices the mean inference accuracies

for estimators trained with the thumb taps were lower compared to the classifiers trained with data containing index finger taps. For the iPhone 7, the estimator yields a mean accuracy of 0.54 for index finger samples compared to 0.38 for data representing the thumb as input modality. The same results apply for the other tested devices. A Wilcoxon signed-rank test shows that the classification results for both input modalities differ significantly. The fold accuracies on thumb data were statistically lower than the fold accuracies on index finger data $Z = 29$, $p < 0.05$.

		Accuracy				
Device	**Input Modality**	mean	min	max	std	**Classifier**
iPhone 6	Index	0.38	0.28	0.47	0.06	ANN
	Thumb	0.35	0.33	0.37	0.01	ANN
iPhone 6s	Index	0.44	0.35	0.54	0.06	SVM
	Thumb	0.37	0.33	0.41	0.02	ANN
iPhone 7	Index	0.54	0.45	0.59	0.04	SVM
	Thumb	0.38	0.30	0.42	0.04	ANN

Table 6.4: Classification results for the 5x4 tapping grid for both input modalities: thumb and index finger.

6.4 Body Posture Comparison

For the comparison between the two body postures (sitting and standing) the overall training material was filtered based on the device and body posture the user had while tapping. Furthermore, only index finger taps are considered in this experiment. As for the comparison of input modalities, the amount of training material was balanced.

		Accuracy				
Device	**Input Modality**	mean	min	max	std	**Classifier**
iPhone 6	Standing	0.30	0.19	0.45	0.08	SVM
	Sitting	0.35	0.29	0.41	0.03	ANN
iPhone 6s	Standing	0.37	0.34	0.42	0.03	SVM
	Sitting	0.47	0.34	0.61	0.08	SVM
iPhone 7	Sitting	0.58	0.44	0.82	0.11	SVM
	Standing	0.43	0.35	0.52	0.05	ANN

Table 6.5: Classification results for the 5x4 tapping grid for both body postures: sitting and standing.

Only the 5x4 grid is presented here, as similar results are found for the other grid sizes (see Appendix). The findings show that the mean accuracies between the two body modal-

ities differ (See figure 6.8). This is also indicated in a Wilcoxon signed-rank test showing that the classification results for both factors differed significantly Z = 31, p > 0.05.

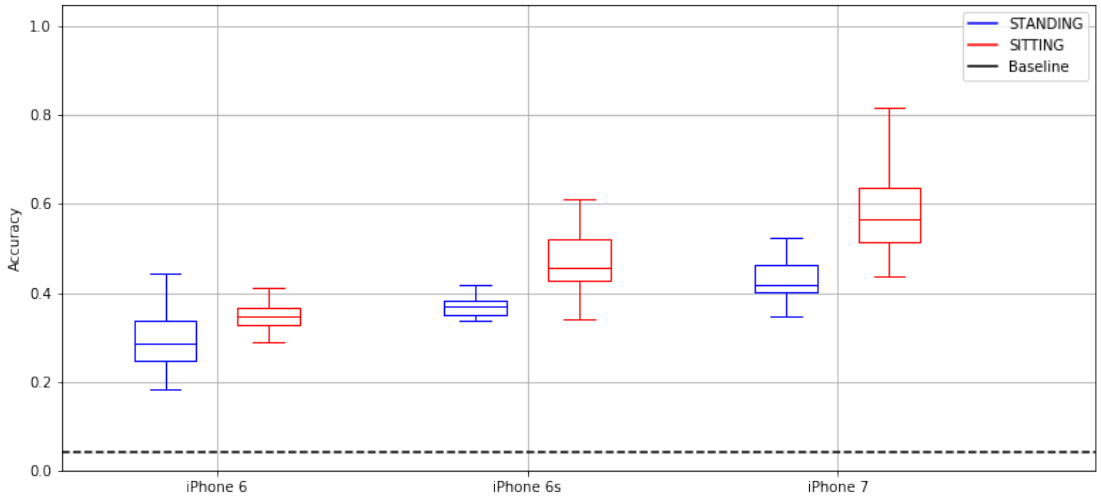


Figure 6.8: The figure shows the tap inference accuracies for the 5x4 grid of the 10-fold cross-validation.

6.5 Cross User Experiment

In order to determine if it is possible to train a classifier with data from a set of users to infer taps from people not involved in the training phase, a cross user experiment was performed. Here, a SVM classifier was trained for each subject with laboratory data from 4 randomly selected subjects sharing the same device. The subject's field data was then tested on the classifier and the accuracy was measured.

For each subject, the accuracy results for the 5x4 grid are displayed in the bar chart 6.9. It can be seen that for user 86 the prediction accuracy was at minimum value of 0.09 whereas for user 87 the overall maximal value of 0.34 could be measured. The mean score measured was 0.25.

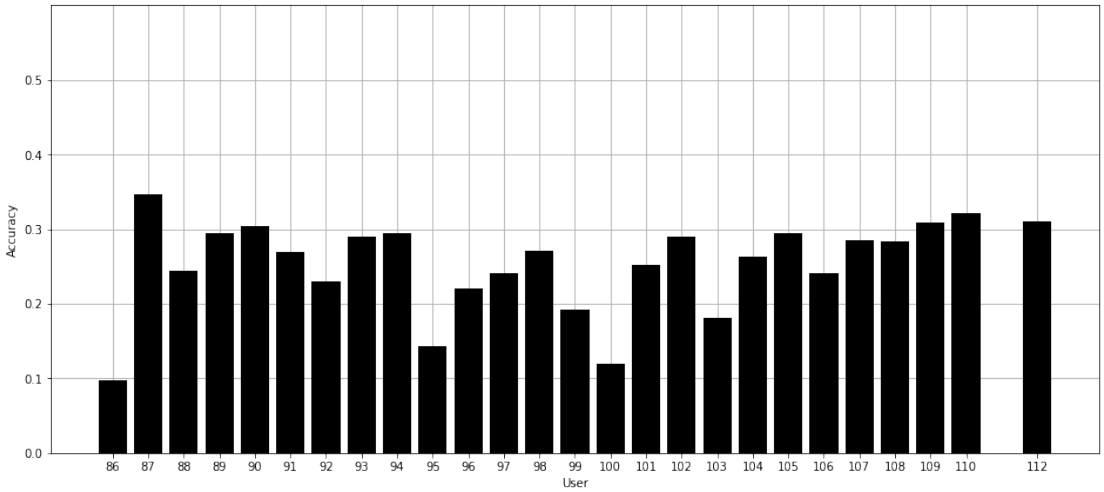


Figure 6.9: Results of the cross user experiment on the 5x4 grid.

7 Discussion

For the hypothesis tests, the assumptions made in section 5.1 will be either rejected or approved based on the results observed in the previous chapter.

H.1 The environment of recorded sensory data has an effect on the prediction accuracy.
H1.1: The prediction accuracy for a classifier trained with the data in the laboratory environment will score higher than one trained with data collected in the field.

Both assumptions can be approved as the results yield a significant difference between the performance measures of the estimators for both environments.

Across all available grid sizes the analysis shows that tap location inference is reasonably possible in the field as well as in the laboratory environment. Yet, for the field environment, an accuracy drop of approximately 20% was measured. Moreover, the results hint that PINs could be obtained in a real-world attack as the granularity of inferable locations is sufficient to project a PIN input mask on the 4x3 grid. Compared to previously proposed inference systems, the system presented in this work yields lower prediction accuracies than *TapPrints* [35]. However, as the scope of this work is to highlight the difference between both environments and not to display an upper bound to what is feasible, by interpreting the results, it is indicated that tap inference in the field is considerably more difficult.

Since device motion sensors are capable of capturing the slightest device vibrations, a vibrant environment or activity, one to which subjects where exposed during the field acquisition, is presumably prone to polluting the sensor signals with increased noise. This noise can distort the tap information encoded in the sensor signals aggravating clear predictions of the tap locations. Consequently, as subjects were free to perform tap generation trails where and how they wanted, this freedom is reflected in the recorded data sets with increased variability negatively impacting the classification accuracies.

When comparing the tested devices, the analysis has shown that the iPhone 7 taps could be predicted with higher accuracies compared to the iPhone 6 and iPhone 6s, respectively. During the feature extraction, it was observed that the iPhone 6 and the iPhone 6s have generated taps that were partially below the sampling rate of 100Hz, the rate initially defined in the TapSensing application. It is assumed that this is caused by high CPU loads on the devices. As a high CPU load causes the sampling rate to drop, due to lower resolution signals a decrease in estimator performance can be explained.

H2: The input modality has an effect on the prediction accuracy.
H2.1: The prediction accuracy for a classifier trained with index finger tap data will score higher than one trained with thumb tap data.

The analysis has shown that classification results of the computed models, when comparing the input modalities, differed significantly. As the index finger taps could be predicted at higher measures compared to the thumb taps, both hypothesis can be approved.

This outcome can be explained by comparing the motion of the individual input modalities. When a user taps the device with the index finger, the striking force of the finger hits the smartphone screen causing a shift towards the z-axis. When the other hand is used as a support, the applied force is partially resisted stopping the device from tilting. In contrast, when a user taps with the thumb, the striking force causes the device to rotate as the device is held in the same hand. This rotation causes a higher variance in the recorded data which results in an inferior predictability.

H3: The body posture has an effect on the prediction accuracy.
H3.1: The prediction accuracy for a classifier trained with taps where a user sat will score higher than one trained with taps where a user stood.

The results have shown that classification measures for both body postures, sitting and standing, differed significantly. The classification for sitting data yielded higher accuracies when compared to the standing data sets. Due to this finding, both assumptions can be approved.

The analysis indicates that the body posture poses an important influence factor on the variability in the motion data collected. This result can be explained based on two assumptions. Firstly, it is likely that subjects used their device while walking during the field study which poses a source for increased noise. Secondly, during the data acquisition in

the laboratory environment, it is known that subject did not walk while tapping the device. As this data was also contained in the training examples, it is assumed that standing on the spot also enables the user to make slight body movements which can effect the variability of the recorded samples.

Lastly, in the cross user experiment, it has been shown that it is possible to predict tap locations of users in the field that where not involved in the training process. This shows that in a real-world scenario an attacker could compute a model trained on several test users and, consequently, make predictions on unseen users in the field. However, as the accuracies measured ranged from 0.9 to a 0.34 for the individual users tested, the varying results indicate that the cross user inference can yield unreliable predictions. These findings align with previous results [35], as varying cross user accuracies were likewise to be found.

With the overall findings in this work, it has been shown that the performance of a tap inference system is strongly influenced by various sources of data variability. Consequently, if an inference system was to be deployed for a real-world attack, it would have to overcome the user switching input modalities, changing body postures and a potential increase in environmental noise from the user's current location in the field. As the impact of the environment was not modelled in related experiments [35, 11, 44], the proclaimed security threat of tap location inference has to be reassessed taking the variance of field data into account. However, as I believe that the performance gap between the field and laboratory environment could be bridged with appropriate filtering techniques or the design of more resilient features, the security threat of motion sensor emanation is yet prevalent.

8 Conclusion

In this thesis, *TapSensing* was presented, a data acquisition system that collects touchscreen tap event information with corresponding accelerometer and gyroscope readings. Having performing a data acquisition study with 27 subjects and 3 different iPhone models, a total of 45,000 labeled taps could be acquired from a laboratory and the field environment. After a feature extraction on the acquired sensor recordings, several machine learning classifiers have been trained and compared in order to firstly, determine if tap location inference is feasible for the field environment and secondly, to identify the sources of variability in the collected data. Furthermore, a real-world attack scenario has been evaluated where it has been tested if the user's field taps can be predicted based on a classifier trained on laboratory data from a different set of users.

The overall findings have shown that tap location inference is generally possible for data acquired in the field, however, with a performance reduction of approximately 20% when comparing both environments. Moreover, it has been shown that the performance of the inference is dependent on the body posture and the input modality used to perform taps as these pose sources for an increased variability in the motion data. Lastly, it has been shown that it is possible to predict tap locations of users in the field that were not involved in the training process which furthermore increases the threat posed by motion sensor emanations. As the tap inference has now been shown for a more realistic data set and by aligning with the previous experiments [11, 35, 44], this work yet again confirms that smartphone motion sensor could potentially be used to comprise the user's privacy. I hope that these findings raise the awareness of potential eavesdropping attacks due to the non-restricted motion sensor access by smartphone operating system providers.

8.1 Further Outlook

It this work, it was identified that the field environment bears a potential source of variability in the motion data resulting in a general decrease of the predictability of tap locations. It is assumed that this is due to an increase in noise originating from the user

activity or vibrant surrounding. For future studies, it could be investigated if applying appropriate filtering on the sensor data could mitigate this “field effect” whereas a second option would be to design resilient features. However, as hand-crafting such features requires high domain knowledge, convolution neural networks could be used to automatically extract features instead.

Convolution neural networks have shown to achieve high accuracies solving the Human Activity Recognition (HAR) problem [55] in which accelerometer signals are used to predict which activity the smartphone user currently has. As the gyroscope and accelerometer signals could be encoded as a single matrix, the convolution network is able to apply convolution filters on the input to automate the feature extraction process. This approach could not only be resilient against environmental noise but could also achieve higher accuracies than the currently proposed methods.

List of Acronyms

APNS	Apple Push Notification Service
ANN	Artificial Neural Network
CSS	Cascading Style Sheets
HAR	Human Activity Recognition
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
ML	Machine Learning
RBF-Kernel	Radial basis function kernel
REST	Representational State Transfer
SVM	Support Vector Machine
WSGI	Web Server Gateway Interface

Bibliography

[1] Deviceorientation event specification. URL https://www.w3.org/TR/orientation-event/.

[2] D. Asonov and R. Agrawal. Keyboard acoustic emanations. In *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, pages 3–11, May 2004. doi: 10.1109/SECPRI.2004.1301311.

[3] Adam J. Aviv, Katherine Gibson, Evan Mossop, Matt Blaze, and Jonathan M. Smith. Smudge attacks on smartphone touch screens. In *Proceedings of the 4th USENIX Conference on Offensive Technologies*, WOOT'10, pages 1–7, Berkeley, CA, USA, 2010. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1925004.1925009.

[4] M. Backes, M. Dürmuth, and D. Unruh. Compromising reflections-or-how to read lcd monitors around the corner. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 158–169, May 2008. doi: 10.1109/SP.2008.25.

[5] M. Backes, T. Chen, M. Duermuth, H. P. A. Lensch, and M. Welk. Tempest in a teapot: Compromising reflections revisited. In *2009 30th IEEE Symposium on Security and Privacy*, pages 315–327, May 2009. doi: 10.1109/SP.2009.20.

[6] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, and Caroline Sporleder. Acoustic side-channel attacks on printers. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, pages 20–20, Berkeley, CA, USA, 2010. USENIX Association. ISBN 888-7-6666-5555-4. URL http://dl.acm.org/citation.cfm?id=1929820.1929847.

[7] Frank Bentley and Konrad Tollmar. The power of mobile notifications to increase wellbeing logging behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 1095–1098, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1899-0. doi: 10.1145/2470654.2466140. URL http://doi.acm.org/10.1145/2470654.2466140.

[8] Yigael Berger, Avishai Wool, and Arie Yeredor. Dictionary attacks using keyboard acoustic emanations. In *Proceedings of the 13th ACM Conference on Computer and*

Communications Security, CCS '06, pages 245–254, New York, NY, USA, 2006. ACM. ISBN 1-59593-518-5. doi: 10.1145/1180405.1180436. URL http://doi.acm.org/10.1145/1180405.1180436.

[9] H.C. Black and J.R. Nolan. *Black's Law Dictionary: Definitions of the Terms and Phrases of American and English Jurisprudence, Ancient and Modern*. Black's Law Dictionary. West Publishing Company, 1990. ISBN 9780314762719. URL https://books.google.de/books?id=LqH24dwKIsUC.

[10] Tim Bray. The JavaScript Object Notation (JSON) Data Interchange Format. Internet-Draft draft-ietf-jsonbis-rfc7159bis-04, Internet Engineering Task Force, July 2017. URL https://datatracker.ietf.org/doc/html/draft-ietf-jsonbis-rfc7159bis-04. Work in Progress.

[11] Liang Cai and Hao Chen. Touchlogger: Inferring keystrokes on touch screen from smartphone motion. In *Proceedings of the 6th USENIX Conference on Hot Topics in Security*, HotSec'11, pages 9–9, Berkeley, CA, USA, 2011. USENIX Association. URL http://dl.acm.org/citation.cfm?id=2028040.2028049.

[12] Meredith A Case, Holland A Burwick, Kevin G Volpp, and Mitesh S Patel. Accuracy of smartphone applications and wearable devices for tracking physical activity data. *Jama*, 313(6):625–626, 2015.

[13] Patrick Colp, Jiawen Zhang, James Gleeson, Sahil Suneja, Eyal de Lara, Himanshu Raj, Stefan Saroiu, and Alec Wolman. Protecting data on smartphones and tablets from memory attacks. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, pages 177–189, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-2835-7. doi: 10.1145/2694344.2694380. URL http://doi.acm.org/10.1145/2694344.2694380.

[14] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2Nd Edition)*. Wiley-Interscience, 2000. ISBN 0471056693.

[15] Bradley Efron and Trevor Hastie. *Computer Age Statistical Inference: Algorithms, Evidence, and Data Science*. Institute of Mathematical Statistics Monographs. Cambridge University Press, 2016. doi: 10.1017/CBO9781316576533.

[16] Claudio Feijoo, José-Luis Gómez-Barroso, Juan-Miguel Aguado, and Sergio Ramos. Mobile gaming: Industry challenges and policy implications. *Telecommunications Policy*, 36(3):212–221, 2012.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[18] Thiago S Guzella and Walmir M Caminhas. A review of machine learning approaches to spam filtering. *Expert Systems with Applications*, 36(7):10206–10222, 2009.

[19] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998. ISBN 0132733501.

[20] Vanmala Hiranandani. Under-explored threats to privacy: See-through-wall technologies and electro-magnetic radiations. *Surveillance Society*, 8(1):93–98, 2010. ISSN 1477-7487. URL `https://ojs.library.queensu.ca/index.php/surveillance-and-society/article/view/3476`.

[21] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification. 2003.

[22] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014. ISBN 1461471370, 9781461471370.

[23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

[24] Robert Koch and Gabi Dreo Rodosek. The role of cots products for high security systems. In *Cyber Conflict (CYCON), 2012 4th International Conference on*, pages 1–14. IEEE, 2012.

[25] M. G. Kuhn. Optical time-domain eavesdropping risks of crt displays. In *Proceedings 2002 IEEE Symposium on Security and Privacy*, pages 3–18, 2002. doi: 10.1109/SECPRI.2002.1004358.

[26] Markus G Kuhn. Electromagnetic eavesdropping risks of flat-panel displays. In *Privacy Enhancing Technologies*, volume 3424, pages 88–107. Springer, 2004.

[27] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell. A survey of mobile phone sensing. *IEEE Communications Magazine*, 48(9):140–150, Sept 2010. ISSN 0163-6804. doi: 10.1109/MCOM.2010.5560598.

[28] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

[29] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521 (7553):436–444, 2015.

[30] Kyumin Lee, James Caverlee, and Steve Webb. Uncovering social spammers: social honeypots+ machine learning. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 435–442. ACM, 2010.

[31] Jó Agila Bitsch Link, Paul Smith, Nicolai Viol, and Klaus Wehrle. Footpath: Accurate map-based indoor navigation using smartphones. In *Indoor Positioning and Indoor Navigation (IPIN), 2011 International Conference on*, pages 1–8. IEEE, 2011.

[32] B. Liskov and L. Shrira. Promises: Linguistic support for efficient asynchronous procedure calls in distributed systems. In *Proceedings of the ACM SIGPLAN 1988 Conference on Programming Language Design and Implementation*, PLDI '88, pages 260–267, New York, NY, USA, 1988. ACM. ISBN 0-89791-269-1. doi: 10.1145/53990.54016. URL http://doi.acm.org/10.1145/53990.54016.

[33] Philip Marquardt, Arunabh Verma, Henry Carter, and Patrick Traynor. (sp)iphone: Decoding vibrations from nearby keyboards using mobile phone accelerometers. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 551–562, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0948-6. doi: 10.1145/2046707.2046771. URL http://doi.acm.org/10.1145/2046707.2046771.

[34] Stephen Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman & Hall/CRC, 1st edition, 2009. ISBN 1420067184, 9781420067187.

[35] Emiliano Miluzzo, Alexander Varshavsky, Suhrid Balakrishnan, and Romit Roy Choudhury. Tapprints: Your finger taps have fingerprints. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, pages 323–336, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1301-8. doi: 10.1145/2307636.2307666. URL http://doi.acm.org/10.1145/2307636.2307666.

[36] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 0070428077, 9780070428072.

[37] Tom Michael Mitchell. *The discipline of machine learning*, volume 3. Carnegie Mellon University, School of Computer Science, Machine Learning Department, 2006.

[38] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress. ISBN 978-1-60558-907-7. URL http://dl.acm.org/citation.cfm?id=3104322.3104425.

[39] Bernard C Nalty. The war against trucks aerial interdiction in southern laos 1968-1972. Technical report, OFFICE OF AIR FORCE HISTORY WASHINGTON DC, 2005.

[40] NATO. Tempest equipment selection process. URL http://www.ia.nato.int/niapc/tempest/certification-scheme.

[41] Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

[42] Leszek Nowosielski and Marian Wnuk. Compromising emanations from usb 2 interface. In *PIERS Proceedings*, 2014.

[43] NSA. Tempest: a signal problem, 2007. URL https://www.nsa.gov/news-features/declassified-documents/cryptologic-spectrum/assets/files/tempest.pdf.

[44] Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang. Accessory: Password inference using accelerometers on smartphones. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*, HotMobile '12, pages 9:1–9:6, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1207-3. doi: 10.1145/2162081.2162095. URL http://doi.acm.org/10.1145/2162081.2162095.

[45] Rafal Przesmycki. Measurement and analysis of compromising emanation for laser printer. In *PIERS Proceedings*, pages 2661–2665, 2014.

[46] Timo Pylvänäinen. Accelerometer based gesture recognition using continuous hmms. *Pattern Recognition and Image Analysis*, pages 413–430, 2005.

[47] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *Proceedings of the International Conference on Research in Smart Cards: Smart Card Programming and Security*, E-SMART '01, pages 200–210, London, UK, UK, 2001. Springer-Verlag. ISBN 3-540-42610-8. URL http://dl.acm.org/citation.cfm?id=646803.705980.

[48] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[49] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 44(1.2):206–226, 2000.

[50] Ben Shneiderman, Catherine Plaisant, Maxine S Cohen, Steven Jacobs, Niklas Elmqvist, and Nicholas Diakopoulos. *Designing the user interface: strategies for effective human-computer interaction*. Pearson, 2016.

[51] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[52] Peter Smulders. The threat of information theft by reception of electromagnetic radiation from rs232 cables. 9:53–58, 02 1990.

[53] Wim van Eck. Electromagnetic radiation from video display units: An eavesdropping risk? *Comput. Secur.*, 4(4):269–286, December 1985. ISSN 0167-4048. doi: 10.1016/0167-4048(85)90046-X. URL http://dx.doi.org/10.1016/0167-4048(85)90046-X.

[54] Martin Vuagnoux and Sylvain Pasini. Compromising electromagnetic emanations of wired and wireless keyboards. In *Proceedings of the 18th Conference on USENIX Security Symposium*, SSYM'09, pages 1–16, Berkeley, CA, USA, 2009. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1855768.1855769.

[55] Ming Zeng, Le T Nguyen, Bo Yu, Ole J Mengshoel, Jiang Zhu, Pang Wu, and Joy Zhang. Convolutional neural networks for human activity recognition using mobile sensors. In *Mobile Computing, Applications and Services (MobiCASE), 2014 6th International Conference on*, pages 197–205. IEEE, 2014.

[56] Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. *ACM Trans. Inf. Syst. Secur.*, 13(1):3:1–3:26, November 2009. ISSN 1094-9224. doi: 10.1145/1609956.1609959. URL http://doi.acm.org/10.1145/1609956.1609959.

Appendix

Results

Data Aquisition

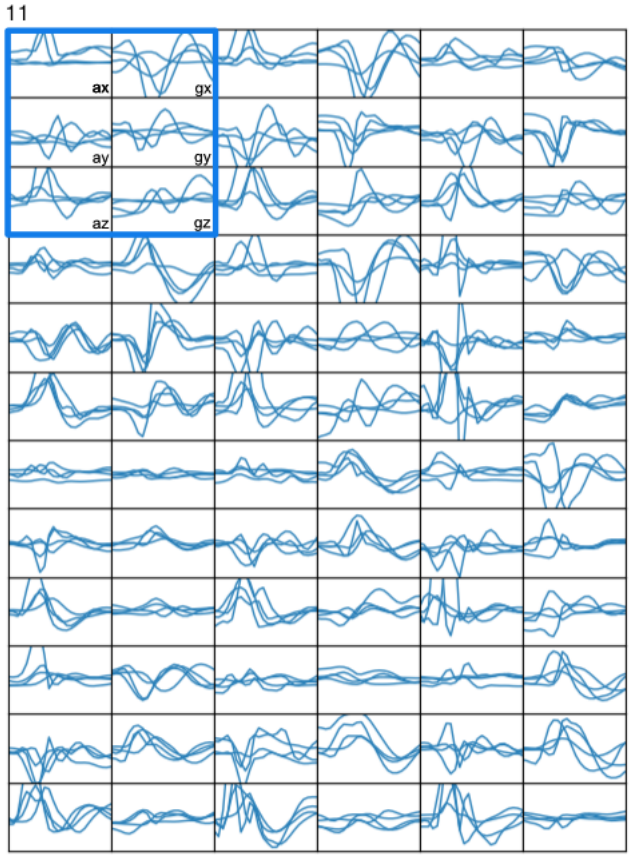


Figure .1: The visualization shows the collected gyroscope and accelerometer components for the 4x3 grid. In the top left corner the grid class 11 is shown which corresponds to the top left corner of the mobile device.

Input Modality

2x2 grid

For the comparison between input modalities, we find that for the 2x2 grid, all mean accuracies across devices were higher for index finger tap data compared to classifiers trained with the thumb tap data.

A Wilcoxon signed-rank test shows that the classification results for both input modalities differ significantly. The fold accuracies on thumb data were statistically lower than the fold accuracies on index finger data $Z = 27$, $p < 0.05$.

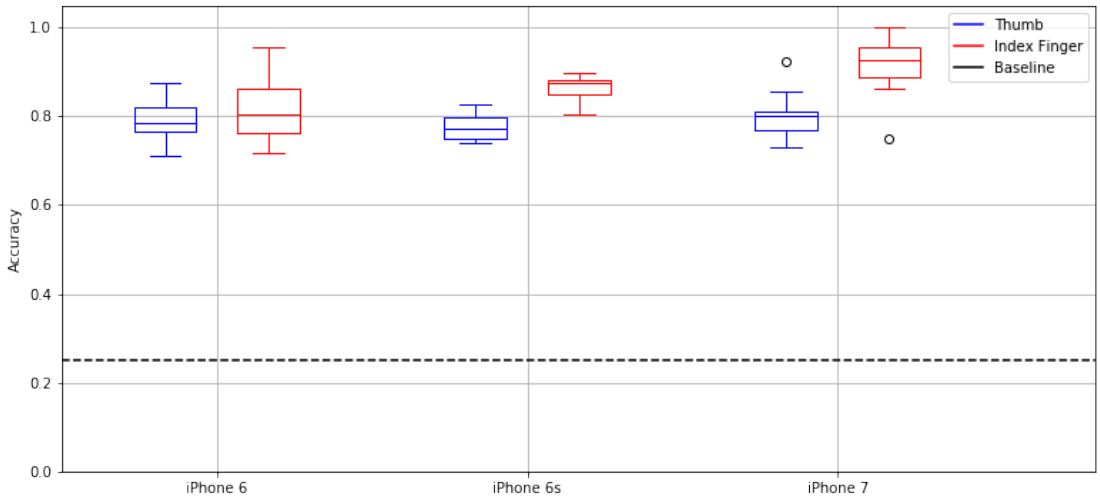


Figure .2: The figure shows the tap inference accuracies for the 2x2 grid of the 10-fold cross-validation.

		Accuracy				
Device	**Input Modality**	mean	min	max	std	**Classifier**
iPhone 6	Index	0.82	0.72	0.95	0.07	SVM
	Thumb	0.79	0.71	0.87	0.05	SVM
iPhone 6s	Index	0.86	0.80	0.90	0.03	SVM
	Thumb	0.78	0.74	0.83	0.03	SVM
iPhone 7	Index	0.91	0.75	1.00	0.07	SVM
	Thumb	0.80	0.73	0.92	0.05	SVM

Table .1: Classification results for the 2x2 tapping grid for both input modalities: thumb and index finger.

4x3 grid

For the 4x3, all mean accuracies across devices were higher for index finger tap data compared to classifiers trained with the thumb tap data. The greatest difference in inference accuracies are to found on the iPhone 7 where 0.7 (+/- 0.08) for index finger records and 0.52(+/- 0.03) for thumb taps.

A Wilcoxon signed-rank test shows that the classification results for both input modalities differ significantly. The fold accuracies on thumb data were statistically lower than the fold accuracies on index finger data $Z = 64.0$, $p < 0.05$.

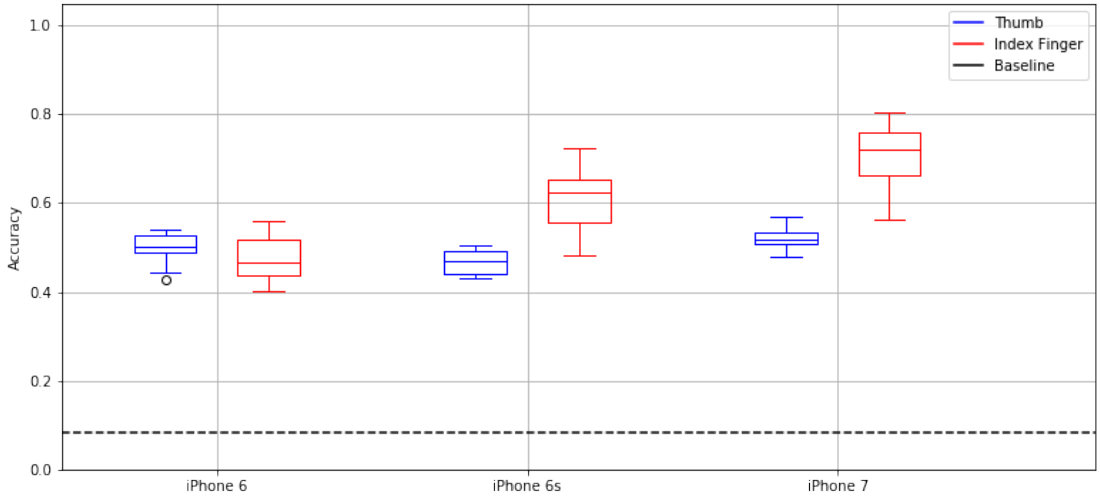


Figure .3: The figure shows the tap inference accuracies for the 4x3 grid of the 10-fold cross-validation.

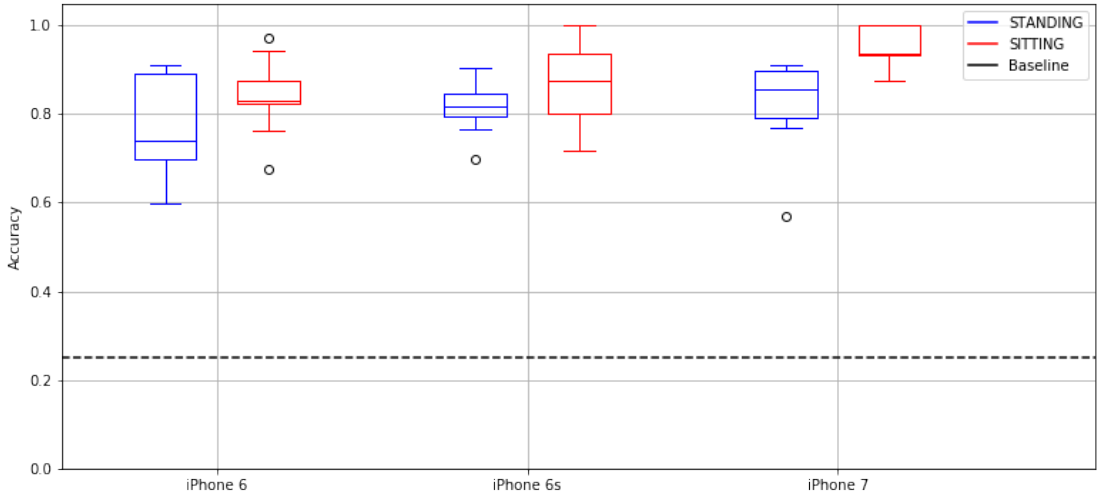
		Accuracy				
Device	**Input Modality**	mean	min	max	std	**Classifier**
iPhone 6	Index	0.48	0.40	0.56	0.06	SVM
	Thumb	0.50	0.43	0.54	0.04	SVM
iPhone 6s	Index	0.61	0.48	0.72	0.07	SVM
	Thumb	0.47	0.43	0.51	0.03	SVM
iPhone 7	Index	0.70	0.56	0.80	0.08	SVM
	Thumb	0.52	0.48	0.57	0.03	SVM

Table .2: Classification results for the 4x3 tapping grid for both input modalities: thumb and index finger.

Body Posture

2x2

For the 2x2, all mean accuracies across devices were higher for sitting data compared to classifiers trained with the standing data. A Wilcoxon signed-rank test shows that the classification results for both body postures differ significantly. The fold accuracies on standing data were statistically lower than the fold accuracies on sitting data $Z = 55.0$, $p < 0.05$.



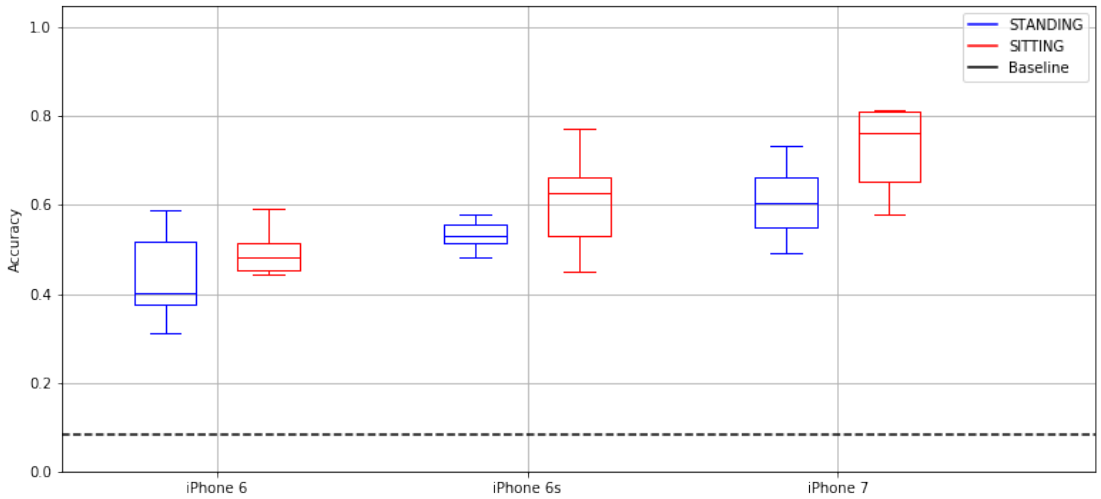
The figure shows the tap inference accuracies for the 4x3 grid of the 10-fold cross-validation.

		Accuracy				
Device	**Input Modality**	mean	min	max	std	**Classifier**
iPhone 6	Sitting	0.84	0.68	0.97	0.08	SVM
	Standing	0.80	0.63	0.94	0.10	SVM
iPhone 6s	Sitting	0.87	0.72	1.00	0.09	SVM
	Standing	0.85	0.73	0.93	0.05	SVM
iPhone 7	Sitting	0.95	0.88	1.00	0.04	SVM
	Standing	0.86	0.60	0.94	0.10	SVM

Classification results for the 2x2 tapping grid for both input modalities: thumb and index finger.

4x3

For the 2x2, all mean accuracies across devices were higher for sitting data compared to classifiers trained with the standing data. A Wilcoxon signed-rank test shows that the classification results for both body postures differ significantly. The fold accuracies on standing data were statistically lower than the fold accuracies on sitting data $Z = 71.0$, $p < 0.05$.



The figure shows the tap inference accuracies for the 4x3 grid of the 10-fold cross-validation.

		Accuracy				
Device	**Input Modality**	mean	min	max	std	**Classifier**
iPhone 6	Sitting	0.49	0.44	0.59	0.05	SVM
	Standing	0.47	0.34	0.62	0.08	SVM
iPhone 7	Sitting	0.73	0.58	0.81	0.08	SVM
	Standing	0.64	0.52	0.76	0.08	SVM
iPhone 6s	Sitting	0.61	0.45	0.77	0.11	SVM
	Standing	0.56	0.51	0.61	0.03	SVM

Classification results for the 4x3 tapping grid for both input modalities: thumb and index finger.

Laboratory Field Comparison

	Grid	Device	Environment	Sample Size	Classifier	Mean Accuracy	Accuracy STD	Mean Kappa	Kappa STD	ratio_index	ratio_sitting	ratio_standing	ratio_thumb
0	(2, 2)	iPhone 6	True	674	SVC(C=8, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.000244140625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.7893237719274141	0.04995206798713887	0.7191170528162798	0.06659407440644427	0.5039281705948373	0.4927048260381594	0.5072951739618407	0.4960718294051628
1	(2, 2)	iPhone 6	False	674	SVC(C=32, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.0009765625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.7069355341708989	0.06199558443322769	0.6090567151793655	0.08272526068636986	0.4885844748858447	0.7317351598173516	0.2682648401826484	0.5114155251141552
2	(4, 3)	iPhone 6	True	2066	SVC(C=32, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.000244140625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.459492516835635	0.05906333202626591	0.41030298296132817	0.06437132008655942	0.4983400959055699	0.5031353744005902	0.4968646255994098	0.5016599040944301
3	(4, 3)	iPhone 6	False	2066	SVC(C=8, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.001953125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.3975098094462521	0.030402507525282174	0.34275186287857473	0.03314680212361573	0.4857357357357358	0.7364864864864865	0.2635135135135135	0.5142642642642643
4	(5, 4)	iPhone 6	True	3403	MLPClassifier(activation='relu', alpha=0.001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(8,), learning_rate='constant', learning_rate_init=0.001, max_iter=10000, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)	0.3473751552649916	0.046858526838929694	0.3130178852513626	0.04932371151705335	0.501113089937667	0.4979964381121994	0.5020035618878005	0.498886910062333
5	(5, 4)	iPhone 6	False	3403	MLPClassifier(activation='relu', alpha=0.001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(8,), learning_rate='constant', learning_rate_init=0.001, max_iter=10000, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)	0.2832927394537601 6	0.037022592639406084	0.24555256556503846	0.03894952413295842	0.4817767653758542	0.7343963553530751	0.2656036446469248	0.5182232346241458
15	(4, 3)	iPhone 6s	True	287	MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(8,), learning_rate='constant', learning_rate_init=0.001, max_iter=10000, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)	0.4279788765388557	0.07247825789844138	0.3759930378385323	0.07905123238563586	0.31922955478370696	0.7590779917903379	0.2409220082096621	0.680770445216293
14	(4, 3)	iPhone 6s	False	287	SVC(C=4, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.0009765625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.5680145931702097	0.05617115018378952	0.5287595547578221	0.061263079747891164	0.5012594458438288	0.5003148614609572	0.4996851385390429	0.4987405541561713
13	(2, 2)	iPhone 6s	True	859	MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(8,), learning_rate='constant', learning_rate_init=0.001, max_iter=10000, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)	0.6793879680030309	0.08858258654637692	0.5723782475715515	0.1181495175914203	0.3170028818443804	0.7531219980787704	0.2468780019212296	0.6829971181556196
12	(2, 2)	iPhone 6s	False	859	SVC(C=64, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.001953125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.854592143590762	0.055489916858969375	0.8061258269201433	0.07400565386574398	0.5028571428571429	0.5	0.5	0.4971428571428572
17	(5, 4)	iPhone 6s	True	1432	MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(8,), learning_rate='constant', learning_rate_init=0.001, max_iter=10000, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)	0.30672963436022405	0.05752235651769441	0.27019799520713145	0.06059013413582241	0.3195266272189349	0.7575873258255392	0.2424126741744608	0.6804733727810651
16	(5, 4)	iPhone 6s	False	1432	SVC(C=8, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.001953125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.4185405876908631	0.04984086969775591	0.3879108350593651	0.05247428595701641	0.5008537279453614	0.5002845759817871	0.4997154240182128	0.4991462720546386
11	(5, 4)	iPhone 7	True	777	SVC(C=4, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.0009765625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.3176316526280808	0.0756708002459489	0.28168617259019924	0.07967305061938533	0.3668195081283868	0.8999583159649854	0.1000416840350146	0.6331804918716132
9	(4, 3)	iPhone 7	False	777	SVC(C=32, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.0001220703125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.4667957390352443	0.07970017425775612	0.4182895121622461	0.08700939609221156	0.3645251396648045	0.8994413407821229	0.1005586592178771	0.6354748603351955

	Grid	Device	Environment	Sample Size	Classifier	Mean Accuracy	Accuracy STD	Mean Kappa	Kappa STD	ratio_index	ratio_sitting	ratio_standing	ratio_thumb
7	(2, 2)	iPhone 7	True	2352	SVC(C=32, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.0001220703125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.7642045454545455	0.09342805141931956	0.6856060606060607	0.1245707352257594	0.3634453781512605	0.8991596638655462	0.10084033613445377	0.6365546218487395
6	(2, 2)	iPhone 7	False	2352	SVC(C=8, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.0009765625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.8729980822116081	0.07537586457156993	0.8306543777515294	0.10049574389918356	0.5	0.4979166666666667	0.5020833333333333	0.5
10	(5, 4)	iPhone 7	True	3905	SVC(C=2, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.001953125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.4346901823050677	0.09061917332256718	0.40493458006404337	0.0953960329649454 8	0.500418410041841	0.498744769874477	0.501255230125523	0.49958158995815904
8	(4, 3)	iPhone 7	False	3905	SVC(C=2, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.001953125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.5911476496006075	0.10545764900531608	0.5539430655507246	0.1150696504127730 8	0.5013966480446927	0.4993016759776536	0.5006983240223464	0.4986033519553073

Body Posture Experiment

	Grid	Device	Sample S	Classifier	Mean Accuracy	Accuracy STD	Mean Kappa	Kappa STD	ratio_index	ratio_sitting	ratio_standing	ratio_thumb
0	(2, 2)	iPhone 6	347	SVC(C=4, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.00390625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.7737649605296663	0.10410387034120573	0.7385014457458403	0.13866960594945776	1	0	1	0
1	(2, 2)	iPhone 6	347	SVC(C=8, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.0009765625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.8395650891935722	0.0791287018059373	0.7853117286727438	0.105911874222396	1	1	0	0
2	(4, 3)	iPhone 6	1026	SVC(C=4, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.001953125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.4370152395608118	0.08491928418087885	0.4184412939264161	0.09257891037753788	1	0	1	0
3	(4, 3)	iPhone 6	1026	SVC(C=64, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.0009765625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.491347372737481	0.04577321030842076	0.4446982374288451	0.05010796979957618	1	1	0	0
4	(5, 4)	iPhone 6	1710	SVC(C=16, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.000244140625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.30409147300141937	0.07905408284594044	0.2989958998138108	0.08319452699195405	1	0	1	0
5	(5, 4)	iPhone 6	1710	MLPClassifier(activation='relu', alpha=0.001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(8,), learning_rate='constant', learning_rate_init=0.001, max_iter=10000, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)	0.3492476482882101	0.03385418927301683	0.3147668174150482	0.03560063008050158	1	1	0	0
6	(2, 2)	iPhone 7	144	SVC(C=4, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.001953125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.9543910256410256	0.041113949928196204	0.9390873015873016	0.05488577260793058	1	1	0	0
7	(2, 2)	iPhone 7	144	SVC(C=8, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.000244140625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.8254042770954536	0.09912381635282216	0.807739387986248	0.13010574357515325	1	0	1	0
8	(4, 3)	iPhone 7	430	SVC(C=16, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.0001220703125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.7315953453453453	0.08494814254425473	0.7071976630779375	0.0927336495646166	1	1	0	0
9	(4, 3)	iPhone 7	430	SVC(C=4, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.001953125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.6059701470672147	0.07585040178312948	0.6021492937592853	0.08285216755286841	1	0	1	0
10	(5, 4)	iPhone 7	719	SVC(C=4, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.0009765625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.5831390532814583	0.10528529382754886	0.561164802405911 6	0.110878625768175	1	1	0	0
11	(5, 4)	iPhone 7	719	MLPClassifier(activation='relu', alpha=0.001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(10,), learning_rate='constant', learning_rate_init=0.001, max_iter=10000, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)	0.42591187959036925	0.05133691005548805	0.4268983299452625	0.05404864252647587	1	0	1	0
12	(2, 2)	iPhone 6s	321	SVC(C=2, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.001953125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.8725378787878789	0.08711421249878659	0.8301969143321154	0.116010744501473	1	1	0	0

13	(2, 2)	iPhone 6s	321	SVC(C=4, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.001953125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.8157075858202518	0.05496292370824566	0.7934607026501002	0.07426259838625503	1	0	1	0
14	(4, 3)	iPhone 6s	962	SVC(C=16, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.00048828125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.6092403504880213	0.10547935677519908	0.5736876904532384	0.11509373371085893	1	1	0	0
15	(4, 3)	iPhone 6s	962	SVC(C=16, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.001953125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.5338738251558857	0.028849795880474212	0.5238139322422672	0.031390650613314905	1	0	1	0
16	(5, 4)	iPhone 6s	1600	SVC(C=4, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.001953125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.4723580360275842	0.07548152978238902	0.44456073066582497	0.07945875095236536	1	1	0	0
17	(5, 4)	iPhone 6s	1600	SVC(C=16, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.00048828125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.370687150057806	0.02540504507306502	0.3687092783594823	0.02673075110156016	1	0	1	0

Input Modality Experiment

	Grid	Device	Sample S	Classifier	Mean Accuracy	Accuracy STD	Mean Kappa	Kappa STD	ratio_index	ratio_sitting	ratio_standing	ratio_thumb
0	(2, 2)	iPhone 6	877	SVC(C=4, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.0009765625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.8180825496342738	0.0741752792376117	0.7573812091589754	0.09889618092356801	1	0.6043329532497149	0.3956670467502851	0
1	(2, 2)	iPhone 6	877	SVC(C=32, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.00048828125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.7882362564036334	0.05060226365640569	0.7176055532952732	0.0675287811466476	0	0.6180159635119726	0.38198403648802737	1
2	(4, 3)	iPhone 6	2645	SVC(C=32, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.0001220703125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.475190766109911	0.05572766172205045	0.4273808777415645	0.060787851693725435	1	0.6120982986767486	0.3879017013232514	0
3	(4, 3)	iPhone 6	2645	SVC(C=8, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.00390625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.4973052731134006	0.035537003953800696	0.4515825820944143	0.03877587071588279	0	0.6260869565217392	0.3739130434782609	1
4	(5, 4)	iPhone 6	4366	MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(10,), learning_rate='constant', learning_rate_init=0.001, max_iter=10000, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)	0.3780874638326970 3	0.05954744510980747	0.3453293496229858	0.06265391712754917	1	0.608337150710032	0.3916628492899679	0
5	(5, 4)	iPhone 6	4366	MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(10,), learning_rate='constant', learning_rate_init=0.001, max_iter=10000, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)	0.3484548897581443	0.012238546052772732	0.31415300387092226	0.012887400414416056	0	0.6209344938158498	0.3790655061841503	1
6	(2, 2)	iPhone 7	413	SVC(C=4, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.001953125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.9148499458567524	0.06758189648711756	0.8864574314574314	0.09010624932191684	1	0.6513317191283293	0.3486682808716707	0
7	(2, 2)	iPhone 7	413	SVC(C=4, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.0009765625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.8045520404156251	0.05085004463707687	0.739088102657018	0.067599968640882	0	0.7433414043583535	0.2566585956416465	1
8	(4, 3)	iPhone 7	1240	SVC(C=8, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.001953125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.7021972833741505	0.07674519782238533	0.6751402746719375	0.08371953706713611	1	0.6532258064516129	0.3467741935483871	0
9	(4, 3)	iPhone 7	1240	SVC(C=4, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.0009765625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.5196531778162952	0.026147742366760908	0.4759770770736599	0.028529142892401323	0	0.7411290322580645	0.25887096774193546	1
10	(5, 4)	iPhone 7	2076	SVC(C=8, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.0009765625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.5353595370332939	0.04192495582087044	0.5109105221216261	0.044141694105773184	1	0.6536608863198459	0.3463391136801541	0
11	(5, 4)	iPhone 7	2076	MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(8,), learning_rate='constant', learning_rate_init=0.001, max_iter=10000, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)	0.3753634098622863	0.03517820472390807	0.34234966733206296	0.03699254981367977	0	0.7384393063583815	0.2615606936416185	1
12	(2, 2)	iPhone 6s	858	SVC(C=4, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.00048828125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.8649614828070181	0.027580892075838703	0.8199648515838748	0.03672314959125757	1	0.6258741258741258	0.3741258741258741	0

13	(2, 2)	iPhone 6s	858	SVC(C=16, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.0009765625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.7774436922104264	0.030274630302597843	0.7030648846583103	0.04044232389726461	0	0.6247086247086248	0.3752913752913753	1
14	(4, 3)	iPhone 6s	2603	SVC(C=64, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.0001220703125, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.6105936692073823	0.07227056920174561	0.5751836870383816	0.0788352179288022	1	0.6304264310411064	0.3695735689588936	0
15	(4, 3)	iPhone 6s	2603	SVC(C=32, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.0009765625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.4674343685422794	0.026918026410318933	0.4189107167534016	0.029397337935336786	0	0.6242796772954283	0.3757203227045717	1
16	(5, 4)	iPhone 6s	4314	SVC(C=32, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=0.000244140625, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)	0.44394128829801865	0.06367926463991158	0.4146606779838947	0.06702132742826558	1	0.6291145108947612	0.3708854891052388	0
17	(5, 4)	iPhone 6s	4314	MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(8,), learning_rate='constant', learning_rate_init=0.001, max_iter=10000, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)	0.3660334097332968	0.02399842882293271	0.3325735726681685	0.025281793977112936	0	0.6367640241075568	0.3632359758924432	1