

AUTHENTICATION OF SWIPE GESTURES IN SMARTPHONES USING SENSOR DATA

OSKAR CERVIN

Master's thesis
2021:E4



LUND INSTITUTE OF TECHNOLOGY
Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematical Statistics

Authentication of Swipe Gestures in Smartphones using Sensor Data

Oskar Cervin

Supervisors: Dr. Johan Swärd & Prof. Andreas Jakobsson

Abstract

Smartphones are an increasingly important part of our lives. As services that treat sensitive information are found on most modern phones, there is a growing need for security. The common approaches of entering a PIN-code or some pattern when accessing an application on the phone is not necessarily sufficient, as they are vulnerable to different types of attacks. The embedded sensors of modern smartphones enable new types of classification based on behavioural patterns. The built in accelerometer and gyroscope measures the acceleration and rotation of the phone in three different directions, respectively (see Figure 1). This work investigates if the data collected from these sensors during a swipe-gesture contains any information about the individual conducting it, such that the individual can be distinguished from others. The data used, as opposed to much of the work that has been conducted in the field, is collected from individuals in their everyday-life. This proves to be a more challenging problem and a method, the Modified Hausdorff distance, performing well when touch-gestures are collected in a controlled environment is refuted. Furthermore, it is shown that there are differences between the different phone-models and operative systems for many features. In an attempt to capture individual information, spectrograms of the signals from the sensors are computed. Different ways of computing these are discussed and tested. A convolutional neural network is used to classify the images in a supervised setting, achieving a test accuracy of 36 % for 42 Android-users and 55 % for 18 iOS-users. Also, using autoencoders for feature representation of the spectrograms in an unsupervised setting is tested. When using kernel density estimation, equal error rates of $\sim 40\%$ is achieved.

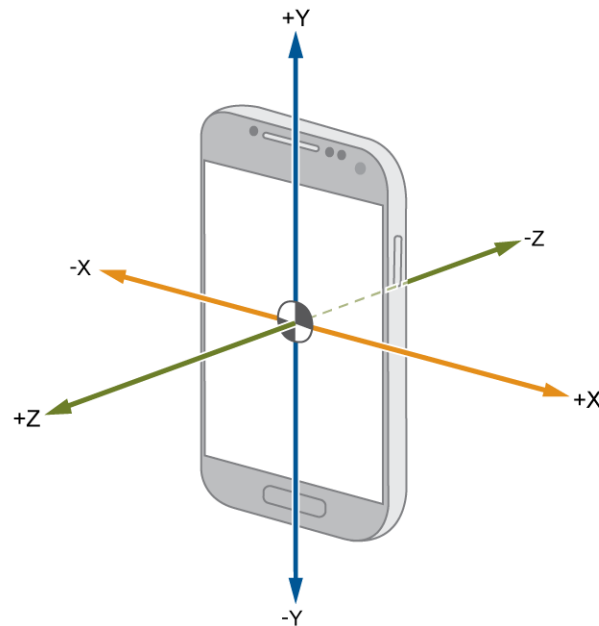


Figure 1: The coordinate system of a phone. [1]

Acknowledgements

I would like to express my gratitude for my supervisor Doctor Johan Swård, who's support, ideas and feedback have been of invaluable importance for this thesis. I would like to thank Professor Andreas Jakobsson for introducing me and Johan and being helpful whenever given the opportunity.

Furthermore, I would also like to thank internet programming- and data science/machine learning community, whose richness and helpfulness never ceases to amaze. All programming have been done in the programming language `Python` and the report is typeset in `LATEX`.

Also, thank you Hanna.

Contents

| | |
|--|------------|
| Abstract | iii |
| Acknowledgements | iv |
| List of Figures | vii |
| List of Tables | ix |
| 1 Introduction | 1 |
| 1.1 Background | 1 |
| 1.2 Purpose | 1 |
| 1.3 Previous work | 2 |
| 2 Theory | 4 |
| 2.1 Modified Hausdorff distance | 4 |
| 2.2 The Fourier Transform | 5 |
| 2.3 Ridge Regression | 6 |
| 2.3.1 Spectrum with Ridge Regression | 7 |
| 2.4 Spectrograms | 8 |
| 2.5 Machine Learning | 8 |
| 2.6 Artificial Neural Networks | 9 |
| 2.6.1 Convolutional Neural Networks | 11 |
| 2.6.2 Autoencoders | 13 |
| 2.7 Novelty and Outlier Detection | 13 |
| 2.7.1 One-Class Support Vector Machines | 13 |
| 2.7.2 Local Outlier Factor | 15 |
| 2.7.3 Kernel Density Estimation | 15 |
| 3 Method | 17 |
| 3.1 The data set | 17 |
| 3.2 Anomaly Detection | 21 |
| 3.3 Feature Extraction | 21 |
| 3.4 Modified Hausdorff Distance | 22 |
| 3.5 Computing Spectrograms | 23 |
| 3.6 Supervised Learning with Spectrograms | 25 |
| 3.7 Semi-supervised Learning with Spectrograms | 25 |
| 3.8 Unsupervised Learning with Spectrograms | 26 |
| 4 Results | 27 |
| 4.1 Feature Extraction | 27 |
| 4.2 Modified Hausdorff Distance | 33 |
| 4.3 Supervised Learning with Spectrograms | 33 |
| 4.4 Semi-supervised learning with Spectrograms | 35 |
| 4.5 Unsupervised Learning with Spectrograms | 36 |

| | |
|---|-----------|
| 5 Discussion | 42 |
| 5.1 Features | 42 |
| 5.2 Modified Hausdorff Distance | 43 |
| 5.3 Spectrograms and Spectrogram-based Classification | 43 |
| 6 Conclusions | 45 |
| References | 46 |
| Appendix I | 48 |
| Appendix II | 52 |
| Appendix III | 55 |

List of Figures

| | | |
|----|---|-----|
| 1 | Coordinate system of a phone | iii |
| 2 | The Hausdorff distance between two sets | 4 |
| 3 | Example of an ANN. | 10 |
| 4 | Example of a convolution | 12 |
| 5 | Example of a CNN | 12 |
| 6 | Example of the structure of an autoencoder | 13 |
| 7 | Example of kernel density estimation. | 16 |
| 8 | Distribution of number of user that made a certain number of swipes | 18 |
| 9 | Example of a measured swipe | 18 |
| 10 | Distributions of durations | 19 |
| 11 | Distributions of the average sampling frequency | 20 |
| 12 | Distribution of number of samples | 20 |
| 13 | PCA:s of the mean-feature | 28 |
| 14 | PCA:s of the variance-feature | 28 |
| 15 | The mean-RMS-feature | 29 |
| 16 | The variance of the RMS | 29 |
| 17 | PCA:s of the mean spectra computed with ridge regression | 30 |
| 18 | PCA:s of the varaince of the spectra computed with ridge regression | 30 |
| 19 | PCA:s of the mean spectra computed with FFT | 31 |
| 20 | PCA:s of the variance of the spectra computed with FFT | 31 |
| 21 | PCA:s of all features | 32 |
| 22 | PCA:s of a variant of all features | 32 |
| 23 | EER:s for a semi-supervised approach | 35 |
| 24 | AUC:s for a semi-supervised approach | 36 |
| 25 | Distributions of scores for a semi-supervised approach | 36 |
| 26 | Examples of spectrograms and their reconstructions | 37 |
| 27 | PCA:s of spectrograms in latent space | 38 |
| 28 | EER:s in for an unsupervised approach | 39 |
| 29 | AUC:s in for an unsupervised approach | 39 |
| 30 | Distributions of scores in an unsupervised setting | 39 |
| 31 | PCA of spectrograms in latent space for test data | 40 |
| 32 | Results for test data | 41 |
| 33 | PCA:s of the mean-feature | 48 |
| 34 | PCA:s of the variance-feature | 48 |
| 35 | PCA:s of the mean-RMS-feature | 49 |
| 36 | PCA:s of the variance of the RMS | 49 |
| 37 | PCA:s of the mean spectra computed with ridge regression | 50 |
| 38 | PCA:s of the variance of the spectra computed with ridge regression | 50 |
| 39 | PCA:s of the variance of the spectra computed with FFT | 51 |
| 40 | PCA:s of the variance of the spectra computed with FFT | 51 |
| 41 | Distributions of scores for different ways of computing the MHD | 52 |
| 42 | Distributions of scores for different ways of computing MHD | 53 |
| 43 | ROC-corves for different ways of computing MHD | 53 |

| | | |
|----|---|----|
| 44 | ROC-curves for different ways of computing MHD | 54 |
| 45 | Example of spectrograms and their reconstructions | 55 |
| 46 | PCA of spectrograms in latent space | 55 |
| 47 | Results for an alternative approach | 56 |

List of Tables

| | | |
|----|---|----|
| 1 | Common choices of activation function | 10 |
| 2 | Common choices of output function | 10 |
| 3 | Common choices of loss function | 11 |
| 4 | Common choices of kernel function | 14 |
| 5 | Common choices of kernel function | 16 |
| 6 | Scheme of data usage | 21 |
| 7 | Extracted features | 22 |
| 8 | EER:s for MHD | 33 |
| 9 | Results for different ways of computing the spectrograms | 34 |
| 10 | Additional spectrogram results | 34 |
| 11 | More additional spectrogram results | 34 |
| 12 | Test results for the best performing spectrogram-approach | 35 |
| 13 | EER:s for autoencoders | 37 |

1 Introduction

1.1 Background

Over the course of the past two decades, smartphones have evolved from being mobile devices dedicated for phone calls and text messages to devices that provides entertainment as well as a wide range of services. As these services may treat sensitive and private information, the importance of protection of such information has become increasingly important. Traditionally, a common method of user authentication has been to enter a PIN-code or some pattern when entering the smartphone. However, these kinds of methods are vulnerable to attacks, such as over-the-shoulder attacks or smudge attacks. Also, it is common that people use the same PIN-code for several different services.

Two important insights that are exploited in this thesis, are that users develop certain behaviors when interacting with their phones and that these behaviors may be used for authentication. If these measured behaviors seem to follow a regular pattern, the need of authentication might not be as extensive. Deviations, on the other hand, may cause alert and call for further authentication requests.

The embedded sensors of modern smartphones enables such new types of authentication. The data collected from different sensors can be used to model behavior based on biometrical behaviors or features. Biometrics may be divided into physiological and behavioural biometrics. Authentication based on physiological biometrics would include fingerprint and face recognition. Typing rhythm, gait, signature, different gestures, and touch gestures would, on the other hand, be features for behavioral biometrics based authentication.

A proposed method for authentication that falls inside the domain of behavioural biometrics is authentication through swiping. Upon entering some service, the user swipes over the screen of the phone. During the swipe, information about the person conducting the swipe can be retrieved from several sensors, which can be used for authentication.

1.2 Purpose

Callsign is an internet-security company that describes itself by:

"Founded in 2012, Callsign's mission is to seamlessly power the identification of every web, mobile and physical interaction. To do this we hire the brightest, most inquisitive minds who want to change the rules of identity and make this mission a reality. Callsign products use deep learning techniques to combine event, threat, and behavioural analytics with multi-factor authentication, securing access to services whilst uniquely ensuring the most frictionless and transparent user experience. We provide risk intelligence in real time, enabling organizations to intelligently adjust authentication journeys, also in real time. By pinpointing suspicious access attempts, we can step up and step down the authentication requirement, catching fraudulent activity more effectively while simultaneously removing friction for legitimate users.

Apart from looking to keep people safe from bad guys, we are also looking to make their digital lives better with ubiquitous technology that works for all, regardless of circumstance."[2]

A large experimental data set containing swipes from individuals in their everyday-life has been collected by Callsign, containing accelerometer data, gyroscope data, orientation data as well as the position, area and pressure of the finger among other data sources. There are, of course, potential for information about the user to be contained in the data from any of the sensors. So, in purpose of finding the most robust model for authentication it would be reasonable to consider data from all sensors. However, it might be of interest to know what information an individual sensor, or a combination of a few, will carry.

The scope of this master thesis is to investigate if the time series from the accelerometer and the gyroscope, collected during a swipe, carry any information about the person swiping in such a way that he or she can be distinguished from a perpetrator solely based on this information. This will be done taking applicability on real-world scenarios into consideration. For instance, ideally, when learning a model for a specific user the learning will be based on the first couple of swipes collected from the user. Also, an unsupervised model is to be preferred over a supervised model. A supervised model would have to be retrained each time a new user would be added to the system which would render the maintenance of such a model to be very cumbersome. However, supervised models might prove helpful in determining if there is any information in certain data or features.

Trying to answer how much information the sensor data contains might prove difficult. The data collected is not from a controlled environment, in contrast to most of the previous work done. The individuals swiping might very well be on the move or sitting by their desks, causing very different conditions for the swipes. Also, the subjects uses different phone-models that are of different sizes and hardware may be of varying quality. Furthermore, to limit the scope to only use accerometer and gyroscope data does also make the problem more difficult.

The approach will be to

1. explore the data,
2. try available models,
3. try to classify the users in a supervised setting
4. try to classify the users in an unsupervised setting

1.3 Previous work

An extensive overview of what have been done in the field of authentication through behavioural biometrics is offered by Abuhamad et. al. in [3]. Amongst several overviews, it includes an overview of different touch gestures, in which swipes, flicks, slides, and hand-writing are included.

A common approach for any kind of problem within data science and or machine learning is to extract features from the data. Classification or analysis can then be based on the features rather than the raw data. Several such approaches is presented in [3]. All of [4]-[10] extract some features from different sensor, which then are fed to different types of classification algorithms, including naïve Bayes classifier, k-nearest neighbours, and random forests. In all of these, some features are based on data from the touch screen, whilst the purpose of this thesis is limited to explore data from the accelerometer and the gyroscope only.

In [11], Jain and Kanhangad uses the modified Hausdorff distance (MHD) as a method to compare similarity between two touch gestures. Several different gestures are investigated: left to right swipe, right to left swipe, scroll up, scroll down, zoom in, zoom out, and single tap. For each gesture, there are several features: accelerometer and orientation data in three directions, respectively, and features related to the position of the finger on the screen and the area the finger occupies. When comparing two gestures of the same type, the MHD was used to compute a score for each feature and direction. The different scores was then normalized and fused in several different ways to generate a matching score. The best result when all gestures are combined is an equal error rate (EER) (the error rate achieved when equally many imposters are undetected as genuine users are denied access) of 0.31 %. A perhaps more relevant results for this thesis are the ones from a single swiping like movement. An EER of 3.05 % is achieved for the sliding up movement. This result is for when both sensor data and touch-screen data are used. The accelerometer alone achieves an EER of 3.35 % and the orientation achieves an EER of 0.56%. This is however when scores from all of the gestures are combined.

Throughout [11], the MHD is compared with dynamic time warping (DTW). Generally, MHD outperforms DTW. This is true for both of the data sets on which the experiments are performed, where the main thing differing between the two sets is the device used. The authors argue that this suggests that their method does not seem to be phone-model specific.

2 Theory

This chapter describes the methods to be used in this work and also give some theoretical background to problems that will be encountered. First, the Modified Hausdorff distance will be presented whereafter various methods for frequency analysis will be introduced. Following this, methods for supervised and unsupervised learning will be presented as well as an introduction to these subjects.

2.1 Modified Hausdorff distance

The Hausdorff distance (or Pompeiu-Hausdorff distance) d_H , is a measure for the distance between two sub-sets of a metric space. For two non-empty sets X and Y it is defined by

$$d_H(X, Y) = \max\left\{\sup_{x \in X} \inf_{y \in Y} d(x, y), \sup_{y \in Y} \inf_{x \in X} d(y, x)\right\}, \quad (1)$$

where d is some measure for the distance between two points. An example is found in Figure 2.

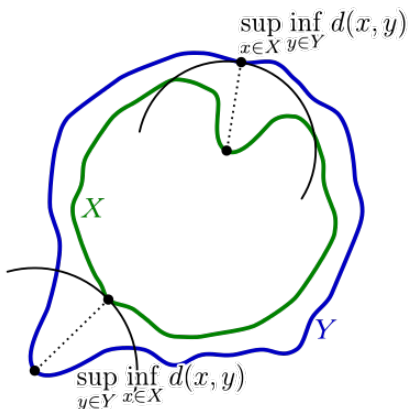


Figure 2: An example that illustrates the different parts of the Hausdorff distance [12].

In [13], Dubuisson and Jain compares different types of measures based on the Hausdorff distance for object matching. The authors' experiments are based on edge points of objects in synthetic images. The measure that performs best is referred to as the modified Hausdorff distance (MHD), which in the discrete case, for two sets $X = \{x_1, x_2, \dots, x_N\}$ and $Y = \{y_1, y_2, \dots, y_M\}$ is defined by

$$d_{MH} = \max(d_{mh}(X, Y), d_{mh}(Y, X)), \quad (2)$$

where

$$d_{mh}(X, Y) = \frac{1}{N} \sum_{x \in X} \min_{y \in Y} \|x - y\|_2 \quad (3)$$

and consequently

$$d_{mh}(Y, X) = \frac{1}{M} \sum_{y \in Y} \min_{x \in X} \|y - x\|_2. \quad (4)$$

The function d_{mh} can be said to compute the average of the minimum distances between the points in one set and the points in the other. In the MHD, both averages are computed and the greater is chosen as measure.

2.2 The Fourier Transform

The Fourier transform is an often used tool within signal analysis. It transforms a signal in time into the frequency plane. For a continuous signal, $x(t)$, it is defined as

$$\mathcal{F}x(f) = X(f) = \int_{-\infty}^{\infty} e^{-i2\pi ft} x(t) dt, \quad (5)$$

and the discrete counterpart, for the discrete signal x_k , as

$$\mathcal{F}x(f) = X(f) = \sum_{k=-\infty}^{\infty} x_k e^{-i2\pi fk}. \quad (6)$$

The result of the transformation is a complex-valued function. The magnitude of the complex number $\mathcal{F}(f)$, will represent the amount of the presence of the frequency f in x and the argument of $\mathcal{F}(f)$ will represent phase shift relative to the sinusoid with frequency f . The plot of $|\mathcal{F}(f)|$, the first case, is called the magnitude spectrum (and will be referred to as the spectrum for the rest of this work).

In reality, a sampled frequency is not defined on the whole real axis, but rather on the interval defined by the time of recording. This corresponds to multiplying the signal $x(t)$, defined on whole of the real axis, by a rectangular window (in this case with the length $2a$):

$$(\theta(t+a) - \theta(t-a))x(t) = w_a(t)x(t), \quad (7)$$

where $\theta(t)$ represent the Heaviside function defined by

$$\theta(t) = \begin{cases} 1, & t > 0, \\ 0, & t < 0. \end{cases} \quad (8)$$

The Fourier transform of the product of two functions f and g , with the Fourier transforms F and G , respectively, is the convolution $F * G$. Thus, the transform of (7) is the convolution of the transforms of w_a and x . As

$$\mathcal{F}w_a(f) = \frac{\sin 2\pi a f}{\pi f}, \quad (9)$$

the sinc-function, this will clearly affect the function and will appear as lobes in the spectrum.

The limited time of the recording of a signal is not the only problem when applying the

transform on measured data. Most recordings of signals are digital, i.e., the signal is sampled. The measurements of the signal will be discrete even though the signal itself may be continuous. The sampling process can be represented by the so-called Dirac-comb

$$\mathfrak{III}_T = \sum_{k=-\infty}^{\infty} \delta(t - kT), \quad (10)$$

where δ is the Dirac delta function and T is the time between each sample, so the sampling rate f_s is $f_s = 1/T$. The sampling of $x(t)$ is thus represented by $\mathfrak{III}_T(t)x(t)$. Transforming \mathfrak{III}_T gives a new Dirac comb:

$$\mathfrak{III}_T(t) \xrightarrow{\mathcal{F}} T\mathfrak{III}_{1/T}(f). \quad (11)$$

Hence,

$$\mathfrak{III}_T(t)x(t) \xrightarrow{\mathcal{F}} T(\mathfrak{III}_{1/T} * X)(f) = T \sum_{k=-\infty}^{\infty} X(f - k/T). \quad (12)$$

Here it can be seen that the spectrum will be periodic with the period $1/T = f_s$. If X is zero for $f \notin [-0.5f_s, 0.5f_s)$, the terms will not effect each other. However, if X is non-zero outside the interval, they will and problematic effects will occur.

If the the signal x is real-valued, the part of the spectrum with negative frequencies will be a mirrored image of the part with positive frequencies. This can easily be seen by applying Euler's formula:

$$X(f) = \int_{-\infty}^{\infty} e^{-i2\pi ft} x(t) dt = \int_{-\infty}^{\infty} x(t)(\cos(2\pi ft) - i\sin(2\pi ft)) dt. \quad (13)$$

For $-f$ the expression instead becomes,

$$X(-f) = \int_{-\infty}^{\infty} x(t)(\cos(2\pi ft) + i\sin(2\pi ft)) dt \quad (14)$$

and it can be seen that even though the phase has changed $|X(-f)| = |X(f)|$ will hold, as $x \in \mathbb{R}$. (If $x \in \mathbb{C}$, this is not necessarily the case). So, only frequencies smaller than half the sampling frequency f_s can be distinguished safely. This limit is called the Nyquist frequency and is commonly denoted with f_N .

There are many algorithms for computing the Fourier transform. One that is frequently used, is the fast Fourier transform (FFT). It exploits the evenly sampled signal to achieve an algorithm of the speed

$$\mathcal{O}(n \log n). \quad (15)$$

2.3 Ridge Regression

For a data vector y of size $n \times 1$, a feature matrix A of size $n \times m$ and full rank, and the weight vector x the ordinary least squares is given by

$$\operatorname{argmin}_x \|y - Ax\|_2^2. \quad (16)$$

Since $\|\cdot\|_2^2$ is convex, the solution to (16) is given by differentiating and putting the derivative equal to zero, yielding

$$x = (A^T A)^{-1} A^T y. \quad (17)$$

Commonly, to allow x to generalize well and avoid overfitting, a regularization term that penalizes the size of x is added to (16). Regular choices are the L^1 -norm, the L^2 -norm (squared) or even L^∞ -norm. Choosing the L^2 -norm as regularizer gives the ridge regression method:

$$\operatorname{argmin}_x (\|y - Ax\|_2^2 + \lambda \|x\|_2^2), \quad (18)$$

where $\lambda > 0$ is called the regularization factor and A no longer needs to be of full rank. Again, the solution is given by differentiating and putting the derivative to zero:

$$x = ((A^T A + \lambda I))^{-1} A^T y, \quad (19)$$

where I is the identity matrix of size $m \times m$. Comparing (17) and (19), the difference is that λ is added to the main diagonal of the inverted matrix. This hinders x from growing too large.

2.3.1 Spectrum with Ridge Regression

The usual approach for computing the spectrum of a signal is to use FFT. This, however, requires the data to be evenly sampled. One option is to resample the signal, but other options are offered by for instance different types of regression.

Suppose $y = (y_1, y_2, \dots, y_n)^T$ is the signal sampled at the time points $t = (t_1, t_2, \dots, t_n)^T$ and the spectrum is to be computed with ridge regression. The vector x and the feature matrix A from (18) should represent the Fourier coefficients and different frequencies, respectively, or more specifically: the columns of A should represent signals with different frequencies. This is achieved by letting an element a_{jk} of A be defined by

$$a_{jk} = \frac{1}{\sqrt{n}} e^{i2\pi f_k t_j}, \quad (20)$$

where f_k is one of m chosen frequencies. The phases of y in relation to the frequencies in A are unknown. This is dealt with by applying the Hilbert transform on y : $\hat{y} = H(y)$ and performing ridge regression for \hat{y} rather than y . Note, since \hat{y} is complex, f_N now is $f_N = f_s$. However, as the transform cancels the negative part of the spectrum this does not matter much in practice, but it should be noted the mirrored image is instead replaced with a part that is all zero.

Suppose that the time vector t is sampled with the intervals $\tau_i = t_{i+1} - t_i$, $i = 1, 2, \dots, n - 1$ and that $\tau_{min} = \min(\tau_1, \tau_2, \dots, \tau_{n-1})$. If

$$\tau_i = \frac{p_i}{q} \tau_{min}, \quad (21)$$

$\forall i$ and $p_i, q \in \mathbb{N}$, the spectrum will be periodic with the period q/τ_{min} . Notice, that if t is evenly sampled, i.e., $\tau_i = \tau_{min} = \tau$ and $p_i = q = 1$, this becomes the same periodicity as in

(12), i.e., the sampling frequency. Equation (21) tells when an exact repetition of spectrum will occur. If the equality sign is changed so that instead

$$\tau_i \approx \frac{p_i}{q} \tau_{min} \quad (22)$$

the spectrum will get a quasi-period and repetition-like sequences for frequencies that may be much lower than the frequency when exact repetition occur. This is of course important to be aware of when studying spectra, cf. [14].

2.4 Spectrograms

The spectrogram of a signal is a visualisation of how the frequency of the signal changes over time. It is computed by sliding some window function, typically a rectangular window, over the signal and computing the spectrum for each window. When a rectangular window is applied to an evenly sampled signal, there are two degrees of freedom: the window size, i.e., how many samples to include in one window, and stride which decides number of samples the window moves forward each step. If the signal is unevenly sampled, these choices are no longer as unproblematic. When the signal is evenly sampled, windows that span the same number of samples also span time intervals of the same size. This is no longer the case when the time between two samples varies - either the windows have to vary in number of samples or in the time they span or the signal has to be resampled.

It may be of interest to have a uniform size for spectrograms computed for signals of different lengths, both in time and in number of samples, and with varied non-uniform sampling. Then yet further degrees of freedom are introduced, as how to keep the number of windows constant must be chosen. If the window length is to be constant, either the stride has to vary or the number of samples needs to be constant, thus requiring the need for the signal to be cut. If instead all of the samples is to be used then the stride or window size needs to vary. Of course, the stride can also be constant if varying the window size or cutting the signal.

To have even more comparable spectrograms it would be reasonable to consider the time-lapse of the signal as well as the window, rather than just the number of samples, cutting the signal such that each signal stretches the same amount of time and choosing the window length as constant as possible.

2.5 Machine Learning

Machine learning is a subarea of the now widely used term artificial intelligence. It is the study of methods and algorithms that improve automatically by experience or by using data. The process of adapting a certain algorithm or method to specific task is called training. The task of a trained machine learning model is often quite specific, but the specific task can be in for wide range of practices. As machine learning is applicable wherever there is data it can be used for practically any subject.

Machine learning is often divided into three subcategories. Which domain a task falls under is often depending on if and what data is available. The categories are supervised learning, unsupervised learning and reinforcement learning. Reinforcement learning (which is not going to be used in this work) is the process of learning an agent to achieve a goal in an environment. The agent learns while exploring the environment, being rewarded when achieving, or getting close to achieving, its goal and being punished when failing. A famous result of reinforcement learning is the chess-computer Deep Blue which in 1997 beat the then reigning chess world champion Garry Kasparov. Another example of where reinforcement learning is used, is for self-driving cars.

Supervised learning is when an algorithm is supposed to learn from a data set where each data point in the data set is associated with some label or value. The purpose is for the model to learn how to associate the data with the corresponding label or value (the first is classification and the latter is regression). It does so by iteratively learning the patterns of the data, taking the data points as input trying to create a correct output (label or value). A typical example of supervised learning is image classification, where the data set consists of labeled images depicting different categories of objects and the model learns how to correctly label the image depending on the content of the image.

In the third category mentioned, unsupervised learning, the algorithms treats data that is not labeled. The purpose of many methods is to bring order by clustering the data or finding an appropriate distribution that explains the data well. Also, dimensional reduction of the data can be categorized as unsupervised learning. The in-between of supervised and unsupervised learning is called semi-supervised learning.

2.6 Artificial Neural Networks

Artificial neural networks (ANN:s) is a category of machine learning methods. The name stems from that, originally, the idea was to mimic the human brain. (If there is any similarities between the brain and the mathematical models presented below is left to the reader to decide.) In an ANN, a data vector, or some feature vector, together with an additional one is called the input layer and is fed to a network consisting of one or several layers of nodes (or neurons) called hidden layers. The last layer of the network, which produces the output, is called the output layer. Figure 3 shows a simple single layered ANN, with an input layer consisting of three features (two data features and one for bias adjustment) which is fed to a hidden layer of two nodes, which in Figure 3 is labeled a_1 and a_2 . Each feature is fed to each node – the layer is then called fully connected. The features are weighted when passed to the nodes, such that $a_1 = \varphi(w_{04}x_2 + w_{02}x_1 + w_{00})$ and $a_2 = \varphi(w_{05}x_2 + w_{03}x_1 + w_{01})$, where the function φ is called the activation function and is chosen to be the same for every node in a layer. Common choices for φ are displayed in Table 1. The nodes are in turn, together with a node for bias adjustment, fed forward to the next layer where the procedure is repeated. In Figure 3, the next layer is the last, i.e., the output layer, consisting of a single node. The output of the network becomes $\hat{\varphi}(w_{12}a_2 + w_{11}a_1 + w_{10})$, where $\hat{\varphi}$ is the activation function of the output layer, called the output function. The choice of $\hat{\varphi}$ is different than that of the activation function for a hidden layer as the network now is producing its result.

The choice of output function $\hat{\varphi}$ for the network is dependent on the task. A summary of the most common is displayed in Table 2.

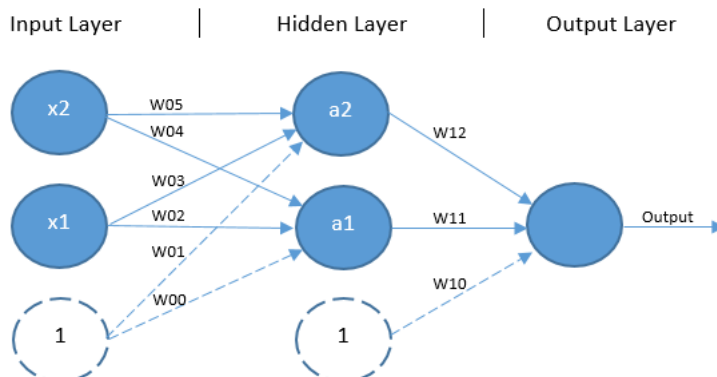


Figure 3: The structure of a very simple ANN [15].

Table 1: Common choices of activation function. For α , it holds that $\alpha > 0$.

| Function | Abbreviation | Definition |
|-----------------------------|--------------|---|
| Rectified Linear Unit | ReLU | $\max(0, x)$ |
| Leaky Rectified Linear Unit | Leaky ReLU | $\max(0.1x, x)$ |
| Hyperbolic tangent | tanh | $(e^x - e^{-x}) / (e^x + e^{-x})$ |
| Softsign | – | $x / (x + 1)$ |
| Softplus | – | $\log(1 + e^x)$ |
| Exponential Linear Unit | ELU | $\begin{cases} \alpha(e^x - 1), & x < 0, \\ x, & x > 0 \end{cases}$ |

Table 2: Common choices of output function. The softmax differs from the other in the way that it takes a vector as input and produces a vector of the same size as output. The elements of the output vector can be interpreted as probabilities as the elements will be in the range $(0, 1)$ and sum to 1.

| Function | Definition | Usage |
|----------|----------------------------|-----------------------|
| Linear | x | Regression |
| Sigmoid | $1 / (1 + e^{-x})$ | Binary classification |
| Softmax | $e^{x_i} / \sum_j e^{x_j}$ | Classification |

The ANN displayed in Figure 3 is of course nothing more than a mathematical function that for an input vector x produces an output y . To repeat, the function for the ANN in Figure

3 is

$$\begin{cases} y = \hat{\varphi}(w_1^T a), \\ a_i = \varphi(w_0^{iT} x), \end{cases} \quad (23)$$

where the inputs, the nodes and the weights have been represented with vectors and the bias is included in the first two. The output of the ANN is compared with the label or value paired with the data vector-input, making ANN a supervised method (however, as will be seen in later parts, there are ways to work around this so that ANN:s can be used in an unsupervised settings as well). The function that compares the output y_i with the label or value, d_i of the input is called the loss function and is also dependent on the task at hand. Table 3 displays the choices for different situations.

Table 3: Common choices of loss functions for different usages. In the definition of the categorical cross-entropy \bar{d}_i and \bar{y}_i are vectors – \bar{y}_i is the probability vector that is the output of the softmax function \bar{d}_i is a vector where the element that represents the class of the i :th sample is one and the rest is zero.

| Function | Definition | Usage |
|---------------------------|--|-----------------------|
| Mean Squared Error | $\frac{1}{2N} \sum_{i=1}^N (d_i - y_i)^2$ | Regression |
| Binary Cross-Entropy | $-\sum_{i=1}^N (d_i \log y_i + ((1 - d_i) \log(1 - y_i)))$ | Binary classification |
| Categorical Cross-Entropy | $-\sum_{i=1}^N \bar{d}_i \log(\bar{y}_i)$ | Classification |

It is very natural to formulate the wanted network, \hat{y} , as the result of an optimization problem with respect to the weights w :

$$\hat{y} = \min_w L(y, d), \quad (24)$$

where L represents the relevant loss function. This can be solved with some gradient descent based method, most often stochastic gradient descent (SGD) or some variant of it like ADAM (the name is derived from adaptive moment estimation) or root mean square propagation (RMSprop). The details of how to calculate the derivatives of the weights with respect to the loss function will be omitted in this report. The reader is instead referred to, for instance, the literature of practically any introductory machine learning course.

2.6.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN:s) are a type of artificial neural network that have become very popular and proven very useful when processing images. They can be represented as a sparsely connected "regular" ANN with repeating weights, but this is not the common way to think about CNN:s. The more intuitive way, as the name suggests, is to think about the convolutions being performed. When a convolution of an image of size N is done, a window, or kernel, of size $n \times m$ (common choices for CNN:s are $n = m = 3$ or $n = m = 5$) is slid over the image producing a new image of size $((N - n)/s + 1) \times ((M - m)/s + 1)$, where s is the stride and no padding is done. An example of a convolution (actually two) is shown in Figure 4.

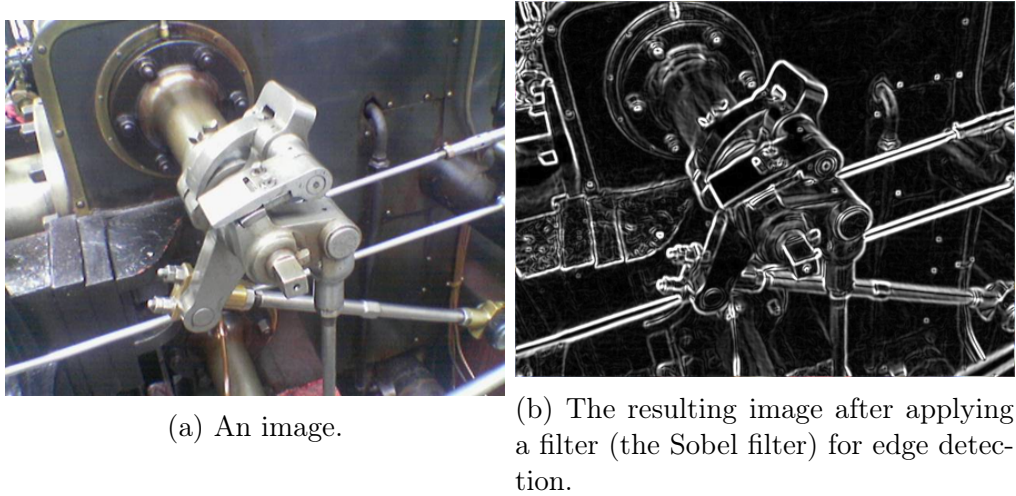


Figure 4: An example of what happens when a convolution is done, in this case when the Sobel filter is applied. [16]

In a CNN, several kernels are convolved with the input image which after being passed through an activation function (most often ReLU) creates several new images. Commonly, the new images are then passed through a max-pooling filter. A window of size $n_{MP} \times m_{MP}$ with stride s_{MP} (often $n_{MP} = m_{MP} = s_{MP} = 2$) is swept over the new image in each location choosing the largest element, reducing the total size of the image with a factor of four (assuming even sizes in both directions). This duality, of a convolutional layer followed by a max-pooling layer can be repeated for several times, after which a number of images with reduced size is left. Before being passed through a fully connected dense layer with an appropriate output function, the image structure is dissolved and all remaining node, or pixels, are rearranged to a single layer of nodes. An example of a CNN is displayed in Figure 5.

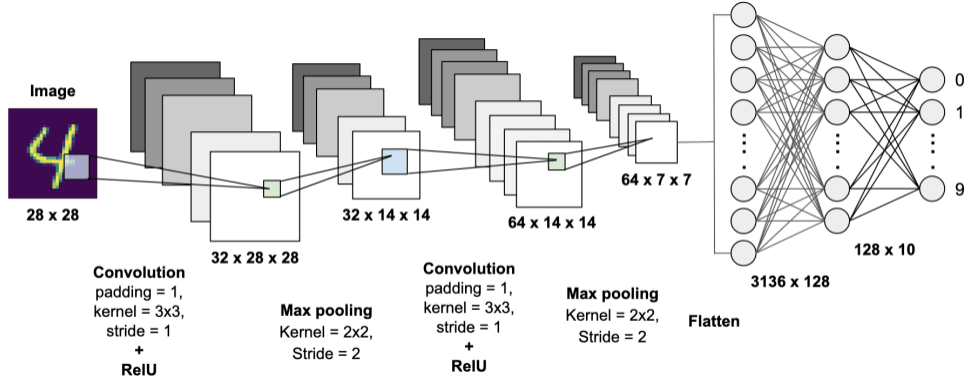


Figure 5: An example of a CNN [17].

2.6.2 Autoencoders

The purpose of an autoencoders is to get a low dimensional representation of something of a high dimension. They can be used for downsampling, noise reduction, feature learning and to generate synthetic data. The first part of an autoencoder is called the encoder. It takes the input and produces a low dimensional code – a representation in the so called latent space. The second part of the network is called the decoder and does the opposite – it produces an output (hopefully) similar to the input. The design of the network as well as the code representation can vary quite a bit, but a common choice is to make the decoder a mirrored image of the encoder. A sketch of the structure of an autoencoder is displayed in Figure 6.

The training of an auto-encoder is a regression problem and is unsupervised since it is not required for the data to be labeled. As the purpose is to recreate the input, the input poses as comparison in the loss function. (In order to minimize the loss, an obvious solution would just be to put the output equal to the input. However, the purpose of an autoencoder is the latent space representation, which would not be very interesting if doing so).

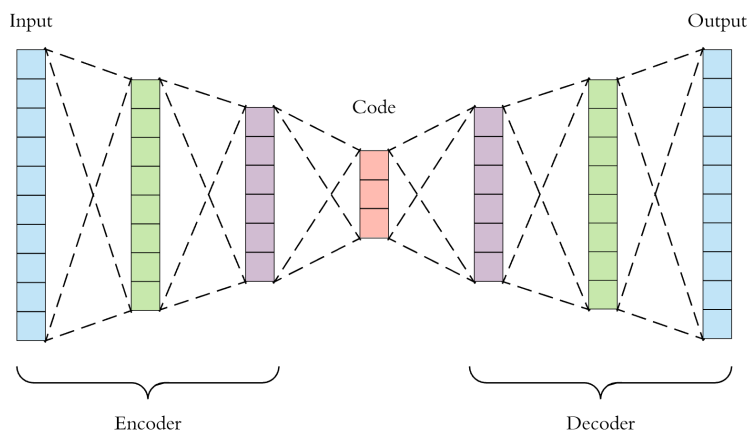


Figure 6: An example of the structure of an autoencoder [18].

2.7 Novelty and Outlier Detection

Novelty detection and outlier detection are closely related. In the former, the aim is from a given one-class data set determine if new samples belong to the class or not. In the latter, the starting data set is polluted and the aim is to distinguish outliers, deviant data points, in an existing data set. This can also be used on new data.

2.7.1 One-Class Support Vector Machines

Support vector machines (SVM:s) is a machine learning technique that is often used in a supervised setting. The most simple, or at least the most illustrative, is the two-class SVM. In the simplest case, the SVM during training finds the hyperplane that separates the two classes with the largest margin. If the data is linearly separable, the so called hard margin can be used to completely separate the two classes. Using the soft margin instead introduces

a slack variable that gives a higher tolerance for outliers and missclassifications in the training data and allows for data sets that are not linearly separable. Mathematically put, the SVM can for n data vectors x_i of size p with labels $y_i \in \{-1, 1\}$, $i \in \{1, 2, \dots, n\}$ be formulated as

$$\min_{w, \rho, \zeta} \frac{1}{2} w^T w + C \sum_{i=1}^n \zeta_i \quad (25)$$

$$\text{subject to } y_i(w^T \phi(x_i) + \rho) \geq 1 - \zeta_i, \zeta_i \geq 0, \quad (26)$$

where w is the weights of the hyperplane, ζ_i represents the slack variables and C determines the strength of this penalty. Intuitively, if $\zeta_i = 0, \forall i$, the first part, $\min 1/2 \|w\|^2$, maximizes the decision margin while constrained to still classify correctly, which is formulated in (26). If a linear approach is not sufficient, the data points can be (implicitly) projected into a feature space, ϕ , where the data can be separated. In the solution to (25)-(26), which will not be carried out here, it shows that that the data points are not explicitly projected to the feature space, but it is only an implicit projection, by a kernel function $K(x, x') = \phi(x)^T \phi(x')$. Common choices of kernel functions are displayed in Table 4.

Table 4: Common choices for kernel function. The parameters r_0 , γ and θ are chosen constants and $\gamma > 0$.

| Kernel Function | Definition |
|-----------------------|---|
| Linear | $\langle x, x' \rangle$ |
| Polynomial | $(\langle x, x' \rangle + r_0)^n$ |
| Radial Basis Function | $e^{-\gamma \ x - x'\ ^2}$ |
| Sigmoid | $\tanh(\theta \langle x, x' \rangle + r_0)$ |

One-class SVM is a natural extension of the two-class SVM to be used in an unsupervised setting. Just as in the two-class SVM, the goal is to separate the data with a hyperplane (possibly in some feature space), but now a majority of the data will be separated from a minority considered outliers. In [19] by Schölkopf et. al, where the one-class SVM was originally introduced, it is formulated as

$$\min_{w, \zeta, \rho} \frac{1}{2} w^T w + \frac{1}{\nu n} \sum_{i=1}^n \zeta_i - \rho \quad (27)$$

$$\text{subject to } (w^T \phi(x_i)) \geq \rho - \zeta_i, \zeta_i \geq 0, \quad (28)$$

where $\nu \in (0, 1]$ and represents probability of finding an outlier in the set and regularizes the penalty term. Notice, the only thing differing (except for ρ being moved around and C being replaced with $1/(\nu n)$) is that factor y_i is missing in (28) as compared to (26). For further details, the reader is referred to [19].

2.7.2 Local Outlier Factor

Local Outlier Factor (LOF) is a way of determining if a sample is an outlier or not by using its neighbours. It tries to capture the deviancy of a data point with respect to its surrounding points. Points dissimilar to its neighbours will be considered outliers.

LOF estimates a density from the k nearest neighbours of the sample. Let the set of the k nearest neighbours of the point x be denoted $N_k(x)$ and the distance, according to some distance measure, to the k :th nearest neighbour $d_k(x)$. Furthermore, let the reachability distance, d_R , for two points x and y be defined by

$$d_R^k(x, y) = \max \{d_k(y), d(x, y)\}. \quad (29)$$

Then, $d_R(x, y)$ is the actual distance between x and y , but with the threshold $d_k(y)$. The local reachability, d_{LR} is defined by

$$d_{LR}^k(x) = \left(\frac{1}{|N_k(x)|} \sum_{y \in N_k(x)} d_R(x, y) \right)^{-1} \quad (30)$$

and represents the inverted average reachability distance from neighbours of x , to x . With this, the LOF-score for the point x is defined by

$$\frac{\sum_{y \in N_k(x)} d_{LR}^k(y)}{|N_k(x)| \cdot d_{LR}^k(x)}. \quad (31)$$

Here, the reachability of x is compared with its neighbours, generating a low score if the reachability of x is large compared with its neighbours and a high score if the relation is the opposite, representing the cases when x is considered an inlier and an outlier, respectively. A strength of LOF is that it can perform classification for both more and less dense areas.

A reasonable choice for k is 20 and the method is also equipped with a parameter representing the expected ratio of outliers. For full details, see [20].

2.7.3 Kernel Density Estimation

The perhaps most intuitive and easiest way of estimating a density is by using a histogram, where each sample or value falls inside a bin, some interval, and the height of the bin is then proportional to the number of sample in that bin. The histogram is however quite clumsy and is also sensitive to how the bins are chosen and illustrated. A correction for this can be made with so called tophat kernel. In a histogram, each value that falls inside a bin contributes to add some height to the rectangular rising from the interval the bin covers, no matter where in the interval the point is located. In the tophat kernel, a rectangular with some height and width is centered around the location of the value. The result is perhaps not as clean, but represents the underlying data better.

Instead of adding rectangles for each data point, some other kernel density can be chosen. Common choices are displayed in Table 5. If the kernel is represented with $K(x, h)$,

where h is a parameter that controls the width of the kernel, the density estimation $\rho(y)$ for the value y can then be formed as

$$\rho(y) = \sum_{i=1}^n K(y - x_i, h), \quad (32)$$

where $x_i, i \in 1, 2, \dots, N$, are training samples. Examples of estimations of distribution for different kernels are displayed in Figure 7. Performing classification (or outlier detection), some threshold value for acceptance and rejection is chosen.

Table 5: Common choices for kernel function. The parameter $h > 0$.

| Kernel Function | Proportional to |
|-----------------|--|
| Tophat | $\begin{cases} 1, & x < h, \\ 0, & x > h \end{cases}$ |
| Exponential | $e^{- x /h}$ |
| Gaussian | $e^{-x^2/(2h^2)}$ |
| Linear | $\begin{cases} 1 - x /h, & x < h, \\ 0, & x > h \end{cases}$ |
| Epanechnikov | $1 - \frac{x^2}{h^2}$ |
| Cosine | $\begin{cases} \cos(\frac{\pi x}{2h}), & x < h, \\ 0, & x > h \end{cases}$ |

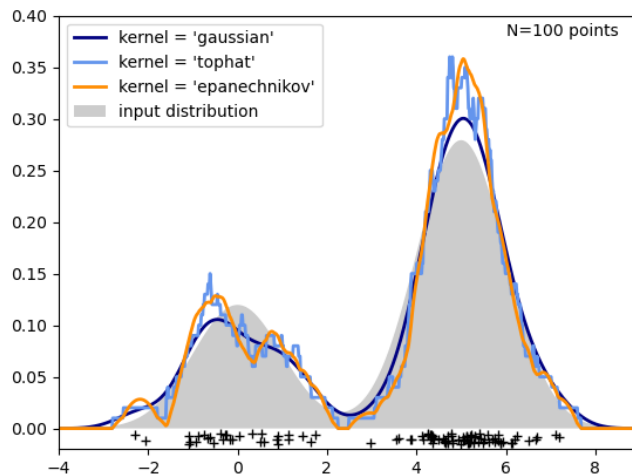


Figure 7: Examples of kernel density estimations for different choices of kernels [21]. The crosses represents samples drawn from the input distribution.

3 Method

This chapter describes how the methods presented in the previous chapter was used. The overall structure of the work was:

- anomaly detection
- feature extraction
- trying the modified Hausdorff distance
- computing spectrograms
- using the spectrograms trying to classify users in a supervised setting
- using the models from the previous step for classification in an unsupervised setting, making the approach semi-supervised all-in-all
- training an autoencoder for dense spectrogram representation to be used for classification, making the approach unsupervised all-in-all

Before describing these steps further, an introduction of the data set to be used is in order.

3.1 The data set

The data set provided by Callsign consists of 28193 swipes, conducted by 90 different users, on 20 different phone models of six different brands which either had the operative system iOS or Android. There are more users using Android than iOS: 62 respectively 28. The width and height of each model is given. For each swipe several timeseries are provided: accelerometer and gyroscope readings in x -, y - and z -direction as well as orientation in three different directions (orientation tells how the phone is positioned in a global coordinate system and is calculated with readings from the accelerometer and the magnetic sensor). Also, the position and the pressure of the finger on the screen and the area it occupies are provided.

The number of swipes a user has performed is not equal and varies quite a lot. Figure 8 shows a histogram of the distribution of the number of users that have done a certain number of swipes.

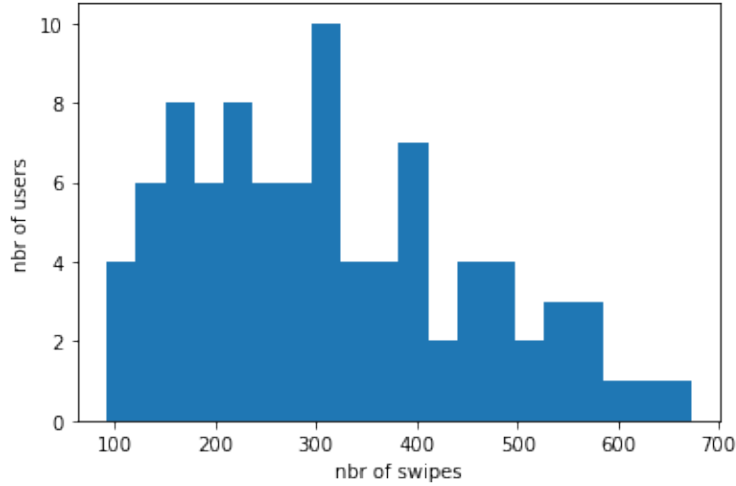


Figure 8: Histogram of how many users that have made a certain number of swipes.

The swipes themselves varies quite a lot too. An example of the readings from the accelerometer and the gyroscope during a swipe is provided in Figure 9. The time is measured from when the authentication application is started and ends when the swipe has been performed successfully.

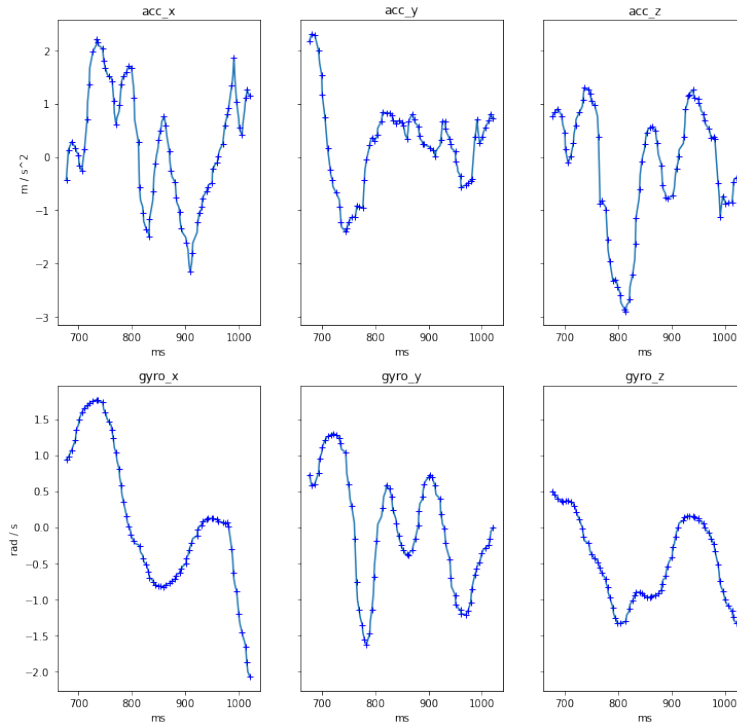


Figure 9: An example of the measured acceleration and angular velocity during a swipe. The blue crosses represents the measured points and line is the linear interpolation of the points

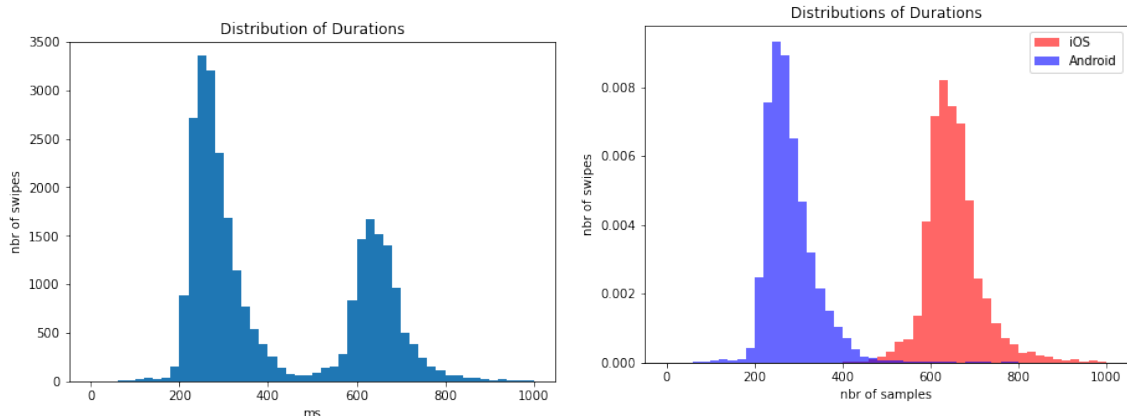
The duration of the swipes varies quite a bit as well, which is displayed in Figure 10a. If the

swipes coming from an iOS-phone and an Android-phone are plotted independently, which is done in Figure 10b, it seems clear that the two distributions are different for the operative systems.

Apart from the durations differing, the sampling rates also differ. The signals from the accelerometer and the gyroscope are not uniformly sampled and they are actually not sampled at the same exact times. However, if the average sampling rate of a signal with sampling points $t = (t_0, \dots, t_{n-1})$ is defined by

$$\begin{cases} \bar{f}_s = \frac{1}{\bar{\Delta}t}, \\ \bar{\Delta}t = \frac{t_{n-1} - t_0}{n-1}, \end{cases} \quad (33)$$

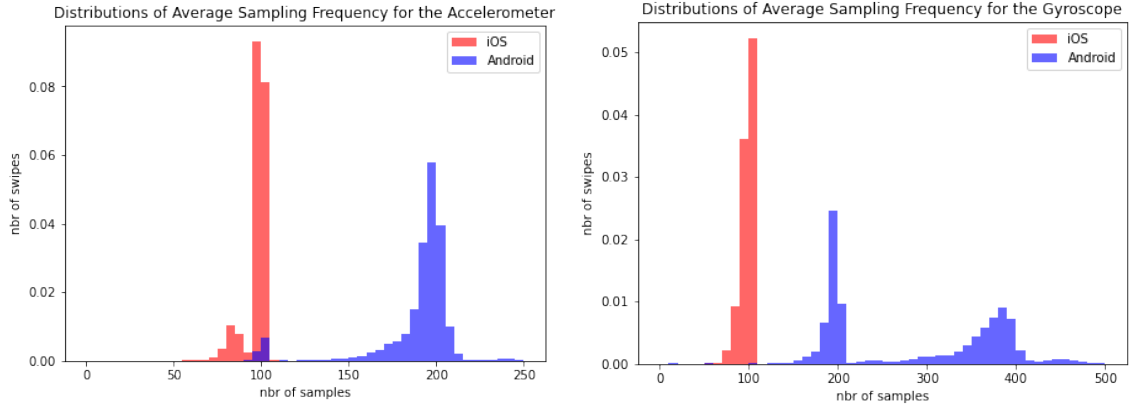
some comparison between sampling rates can be made. The comparisons for the accelerometer and the gyroscope are shown in Figure 11a and 11b, respectively. The sampling rates from Android-phones have a higher variance, especially the rates from the gyroscope-signals. This is not surprising as there are five different brands with the Android-operative system while, of course, only one with iOS. Also the distribution of number of samples in the swipes are displayed, in Figure 12.



(a) Distributions of durations.

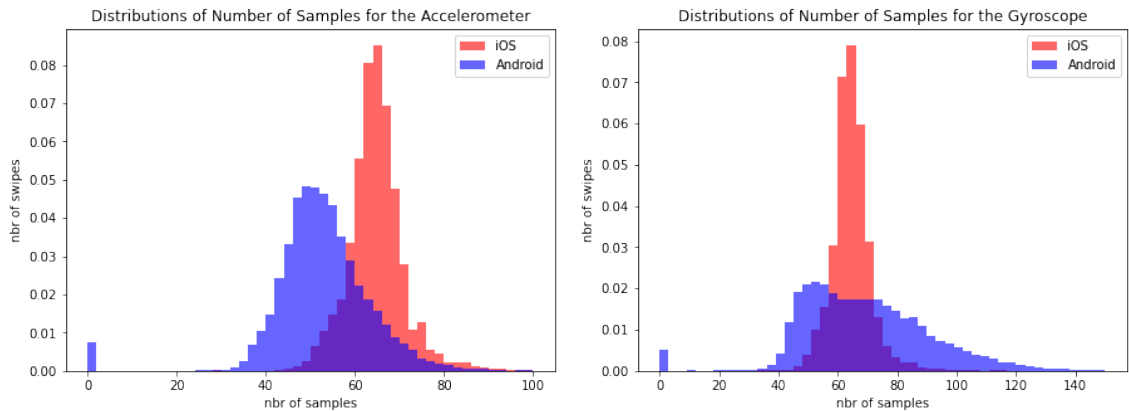
(b) Distribution of durations for Android and iOS.

Figure 10: Distribution of durations, where the differences between Android and iOS shows. Note that (a) uses absolute frequencies, while (b) uses relative frequencies.



(a) Distributions of the average sampling frequency for the accelerometer. (b) Distributions of the average sampling frequency for the gyroscope.

Figure 11: The distributions of the average sampling frequencies. There is a clear difference between **Android** and **iOS** in both cases. For the gyroscope, the **Android**-distribution is divided in two parts.



(a) Distributions of the number of samples for the accelerometer. (b) Distributions of the number of samples for the gyroscope.

Figure 12: The distributions of the number of samples in a signal. There is a clear difference between **Android** and **iOS** in both cases.

The data was split in different parts for training, validation and testing. As the work consists of several parts the data usage scheme is bit intricate. It can be found in Table 6.

Table 6: Scheme of data usage. An X marks the usage of data and X* marks that the data was split further. These splits was however not made on a user-basis.

| | Users (Android- + iOS-users) | | |
|-------------------------------|------------------------------|------------|----------|
| | 90 (62+28) | | |
| | Training | Validation | Test |
| | 60 (42+18) | 20 (15+5) | 10 (5+5) |
| Feature Extraction | X | - | - |
| Modified Hausdorff Distance | X | - | - |
| Convolutional Neural Networks | X* | - | - |
| Classification based on CNN | - | X* | - |
| Autoencoders | X* | - | - |
| Classification based on AE:s | - | X* | X |

3.2 Anomaly Detection

Anomalies, in this case swipes that are deviant, are often quite easy to detect for a human by just looking at them. With just above 28000 swipes, however, this is not a possibility. Removing swipes with fewer than a certain number of samples is a quite obvious start as a swipe containing just a few samples is not likely to have been conducted or measured properly. For the two parts, the feature extraction and the MHD-part this limit was set to eight. When computing the spectrograms, it was set to 40, not so much for treating more swipes as outliers but for being able to create spectrograms of meaningful and uniform size (more on this later).

Also, swipes that had too deviant durations was considered outlier. If the duration was less than half or more than twice the mean duration of the swipes from the same user, the swipe was considered an outlier.

3.3 Feature Extraction

There are of course a lot of features to extract from the in total six signals from a swipe (accelerometer and gyroscope readings in three dimensions each). The features found in Table 7 were extracted from the signals of the form $\mathbf{x}_{acc} = (\mathbf{x}_0^{acc}, \mathbf{x}_1^{acc}, \dots, \mathbf{x}_{n-1}^{acc})$ and $\mathbf{x}_{gyro} = (\mathbf{x}_0^{gyro}, \mathbf{x}_1^{gyro}, \dots, \mathbf{x}_{m-1}^{gyro})$ sampled at the times $t_{acc} = (t_0, t_1, \dots, t_{n-1})$ and $t_{gyro} = (t'_0, t'_1, \dots, t'_{m-1})$, respectively.

Table 7: Features extracted and a short description. Each feature was normalized with standard normalization.

| Feature | Description |
|--|---|
| Mean | The mean of each signal |
| Variance | The variance of each signal |
| Mean of Root Mean Square | The mean of the root mean square, which is $\mathbf{x}_i^{rms} = \ \mathbf{x}_i\ _2/3$ |
| Variance of Root Mean Square | The variance of the root mean square |
| The Mean of Spectrum with FFT | Spectrum computed via resampling and then FFT. The mean is computed by $a^T f / \sum_i a_i$, where f represents the frequencies and a the corresponding vector of the strength of the frequencies. |
| Variance of Spectrum with FFT | The variance of the spectrum computed via resampling and FFT |
| Mean of Spectrum with Ridge Regrsson | Mean of the spectrum computed with ridge regression |
| Variance of Spectrum with Ridge Regrsson | Variance of the spectrum computed with ridge regression |

When resampling was done, the number of samples in the output signal was kept equal to the number of samples in the input signal. For interpolation a cubic spline was used.

A choice of how much or which parts of the signal to consider is also available. The finger does not touch the screen during the whole signal, but only parts of it, so in order to only treat the movements during the actual swipe the rest of the signal should be discarded. However, there might be interesting information in the movements the moments just before and just after the finger touches the screen so it would be reasonable to include a few samples both before and after the finger touches the screen. Extending this argument to its maximum, the whole signal should be considered as there might be information stored even though it is before the actual swipe is being conducted. Both cases were tested.

3.4 Modified Hausdorff Distance

The procedure in this section roughly followed the approach taken in [11]. As [11] handles data from accelerometer and orientation data, the orientation data was also included in this part in order to get comparable results.

The MHD between two signals was computed in accordance with Section 2.1, using the L^2 -norm as distance measure. A pair of signals produces nine scores, if each channel is treated separately, and three scores if the samples from each sensor is treated as a three dimensional point (which is not done in [11]). When all scores were computed (more on this below), they were normalized in three different ways. If s is the total set of scores for one channel the following normalization techniques were used:

1. Standard:

$$s'_i = \frac{s_i - \mu}{\sigma}, \quad (34)$$

where μ and σ represents the mean and standard deviation of s , respectively.

2. Min-max:

$$s'_i = \frac{s_i - s_{min}}{s_{max} - s_{min}}, \quad (35)$$

where s_{min} and s_{max} are the elements of the lowest and highest value in s , respectively.

3. tanh-estimator:

$$s'_i = \frac{1}{2} \left(\tanh \left(\frac{s_i - \mu}{100\sigma} \right) + 1 \right). \quad (36)$$

After the normalization-step, the score was normalized but it yet remains to fuse the nine or three different scores. If s_i represents the score from the i :th of the n channels, the following fusing techniques were used:

1. Minimum:

$$S = \min(s_1, s_2, \dots, s_n) \quad (37)$$

2. Maximum:

$$S = \max(s_1, s_2, \dots, s_n) \quad (38)$$

3. Sum:

$$S = \sum_{i=1}^n s_i \quad (39)$$

4. Product:

$$S = \prod_{i=1}^n s_i \quad (40)$$

In total, this creates 24 different approaches for computing the MHD-score.

In order to compare scores the following approach was used: the first five swipes of a user was used as ground for comparing. This set can, in lack of better terms, be labeled as training set. The rest of the swipes was put in (again, in lack of better terms) a validation set together with equally many randomly drawn swipes from other users with phones with the same operative system. The MHD-score between each member of the training set and validation set was then computed, in all the 24 different ways.

3.5 Computing Spectrograms

As presented in Section 2.4 there are a lot of choices to be made when computing spectrograms. Two main approaches were used to get uniform sizes: to vary the window size in order to get the same number of windows for different swipes and to keep the window size constant (in number of samples) but only considering a certain number of samples (these

approaches will be referred to as approach 1 and approach 2). The number of samples was chosen to 40, which also sets a limit for the minimum number of samples the signal can consist of. Approach 1 also has a limit for the lowest number of samples as the least has to be equal to number wanted windows. If also requiring a minimum length of the window, the limit instead becomes this minimum length added to the number of wanted windows. Throughout this part, this limit was 40 for approach 1 as well approach 2 (40 is chosen based on Figure 12). In both cases the stride was set to one.

Initially, the number of windows was set to 35, giving a minimum window length of five for approach 1 and a window length of five to approach 2. As now familiar, there are six different signals for one swipe. These were both treated separately, such that each signal produced one spectrogram, but also the L^2 -norm for the signals of each sensor was treated, such that each sensor produced a spectrogram each.

The spectra was computed in three different ways: with ridge regression, with resampling followed by FFT and with resampling followed by ridge regression. It is of importance that, when doing ridge regression, the frequencies in the feature matrix A covers a sufficient interval. To cover the range up to the first significant quasi repetition occurred, as discussed in Section 2.3.1, was practically proved sufficient. For each signal, the frequencies in A were $\{0, 1, 2, \dots, \lceil 1.05\bar{f}_s \rceil\}$ Hz, where \bar{f}_s is the mean sampling frequency, was enough to cover this interval. To maintain spectrograms of matching sizes, only the first 50 frequencies were saved, corresponding to frequencies up to 50 Hz.

When using the FFT, choosing an appropriate maximum frequency is not a concern. However, the mesh of the frequency grid should still match between different swipes and correspond to $0, 1, 2, \dots, 49$ Hz. This was done similarly as for when using ridge regression – by letting the number of points of the output from the FFT be $\lfloor f_s \rfloor$. Notice that the i :th frequency, f_i , will only be $f_i = i$ (as desired) when $\lfloor f_s \rfloor = f_s$. However, as

$$f_i = i \frac{f_s}{\lfloor f_s \rfloor} \iff \frac{f_i}{i} = \frac{f_s}{\lfloor f_s \rfloor} \quad (41)$$

and

$$\frac{\lfloor f_s \rfloor - 1/2}{\lfloor f_s \rfloor} < \frac{f_s}{\lfloor f_s \rfloor} \leq \frac{\lfloor f_s \rfloor + 1/2}{\lfloor f_s \rfloor}, \quad (42)$$

the difference will be quite small since f_s at least is $f_s \approx 100$.

A few more variants of spectrograms was computed: the number of windows was set to 30 instead of 35 for both approaches, giving a minimum window length and window length, respectively, of ten instead of five as 40 was kept as the minimum number of samples. These settings were only tried together with FFT-computation of the spectra and treating each channel separately. Also, a different frequency grid was tried: $\{0, 1/2, 1, 3/2, 2, \dots, 39/2\}$ (approximately, since FFT was used). Again, only for treating the channels separately. Justification for these choices will be clear in the results section.

Approaches keeping the time instead of the numbers of samples constant was also discussed

in **Section 2.4**. When using resampling, this is equivalent as the interval between two samples will be constant. When not resampling, a desired window-length d needs to be chosen and thus the samples best matching d best found. For the i :th window, with starting point at the time t_i , this can be formulated as

$$\operatorname{argmin}_{t_j} |d - (t_j - t_i)|, \quad (43)$$

, where t_j also is a time of sampling, reasonably, with the constraint $j > i$. Imagine $t_{i+1} - t_i \gg d$, which very well might be the case, the solution to (43) will be t_{i+1} . Taking t_{i+1} as solution, the spectrum will be computed from only two samples, which of course is unreasonable. Either, some limit k would have to be introduced such that $j > i + k$ or some interpolation has to be performed, which is done in full extent by resampling.

3.6 Supervised Learning with Spectrograms

The purpose of this part is dual – it is both to see how classification can be made in a supervised setting but also to evaluate the different methods for computing the spectrograms presented in the previous section. A reasonable choice of model is a CNN, as the spectrograms of a swipe can be treated as two and six channeled images, respectively. After some testing, a network structure of three convolutional layers interchanged with max-pooling layer followed by a dense layer of 512 nodes and the output layer was decided upon. As activation function in the convolutional layers ReLU was used.

Users with iOS-phones and Android-phones were treated separately, due to the differences between the operative systems. One network was trained for each group and consequently trained for distinguishing Android-users from other Android-users and iOS-users from other iOS-users. The data used (see Table 6) was split in three sets. Twenty random samples from each user was used for validation and equally many for testing. The rest was used as training data. As this set is rather unbalanced with respect to samples per class, the samples were weighted to even this out.

3.7 Semi-supervised Learning with Spectrograms

The output of the networks in the previous section is a vector of probabilities where the i :th element represents the probability for the spectrogram being from the i :th user. In this part, the networks was fed with new spectrograms from previously unseen users. These new spectrograms will also produce outputs – a vector of the size of the number of classes the network was trained on (42 for Android and 18 for iOS). The semantic interpretation of this vector is perhaps a bit dubious, as the vector consist of probabilities for classes the input with certainty does not belong to. These outputs can, nevertheless, be used as feature representation. The spectrograms of the validation set (see Table 6) was fed the networks (still separated by operative system), producing features. Based on the features, classification was carried out in an unsupervised manner, using one-class SVM, LOF and KDE.

Two types of approaches were used: one where the models got a large number of samples

for training after which validation was performed on another set. The second set consisted of the 40 last swipes of each user. The other approach was one-step-ahead prediction, where the smallest training size was 5.

One-step-ahead prediction In one-step-ahead prediction, the model is first trained with the n first samples of a class. The model then predicts the $n + 1$:th sample and a sample of a negative class. The model is then retrained with first $n + 1$ samples and predicts the $n + 2$:th sample and another sample of a negative class and so on. This illustrates how well the model can perform based on how many samples it has trained on.

3.8 Unsupervised Learning with Spectrograms

The idea in this part is actually quite similar to the one in the previous section. The difference is, as the title implies, that the setting now is unsupervised. This is a more realistic approach as the maintenance of a supervised model, which would have to be changed every time a new user was added, would be cumbersome. It would be more realistic to let a new user perform a number of test swipes. The task is then to determine if future swipes are similar to these or not.

So, instead of using the networks trained for supervised learning, a set of autoencoders with convolutional layers was trained. The purpose of the autoencoders are, just as that of the CNN:s in the previous section, to get a feature representation of the spectrograms. One-class SVM, local outlier factor and kernel density estimation were again used to perform the unsupervised classification.

In order to determine the best performing autoencoder, one-class SVM, LOF and KDE with a Gaussian kernel were applied to the features produced by each of the encoders. The models got a large amount of data to train on and was validated on on another set (the last 20 swipes of each user). The encoder producing the best result was chosen. In the next step, the best prediction model (i.e, one-class SVM, LOF or KDE) was chosen. This was done by choosing the model performing best when using one-step-ahead prediction. Finally, the best performing encoder was applied to the spectrograms of the test set, after which the best performing classification method was applied with one-step-ahead prediction.

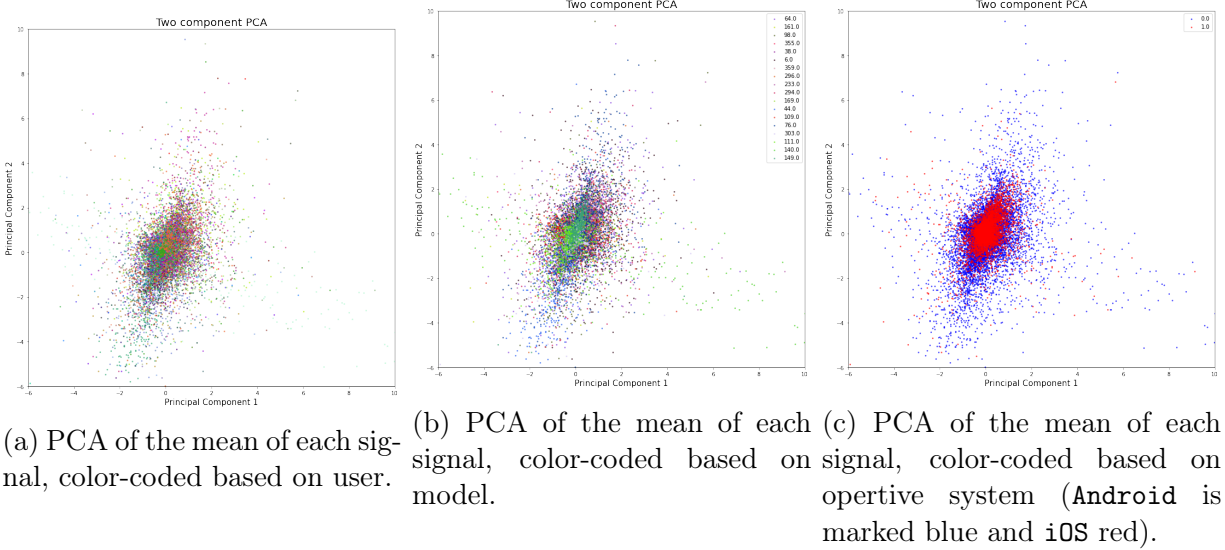
4 Results

The results are presented in this chapter. First, the results of the feature extraction is presented and then the results of the MHD. Following this, the results comparing different ways of computing the spectrograms are presented together with the results in a supervised setting. Second to last, the result in the semi-supervised setting are displayed and lastly, the results in a unsupervised setting are presented.

4.1 Feature Extraction

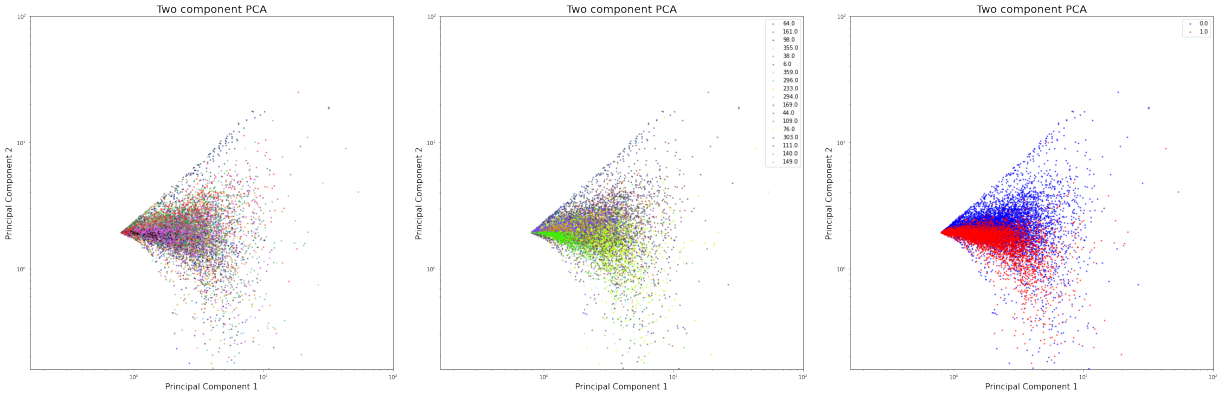
The features described in Table 7 are displayed in Figures 13-20. As all feature spaces are six-dimensional, with exception for the RMS-features that are two dimensional, principal component analysis (PCA) has been performed in order to illustrate the features in two dimensions. How the users are distributed as well as how the models and operative systems are distributed are displayed. As the goal is to find features that discriminate as well possible between different users it is important to know if a potential clustering is caused by user differences or differences between the phone-models. In Figure 21 all features concatenated are shown.

The mean feature does not discriminate much in any instance. Figure 13c shows that the distribution of the **Android**-phones are larger than the **iOS**. As the samples from a certain model typically is centered in one location, as Figure 13b suggests, it is hard to distinguish if the feature finds user-difference. In Figure 14 the differences are more clear. The samples from a user, model or operative system are typically cone-shaped with different directions out from the origin. Potential differences between users seems to be well explained by underlying model-differences.



(a) PCA of the mean of each signal, color-coded based on user. (b) PCA of the mean of each signal, color-coded based on phone-model. (c) PCA of the mean of each signal, color-coded based on operative system (Android is marked blue and iOS red).

Figure 13: PCA:s of the mean of each signal color-coded for different instances: user, phone-model and operative system.

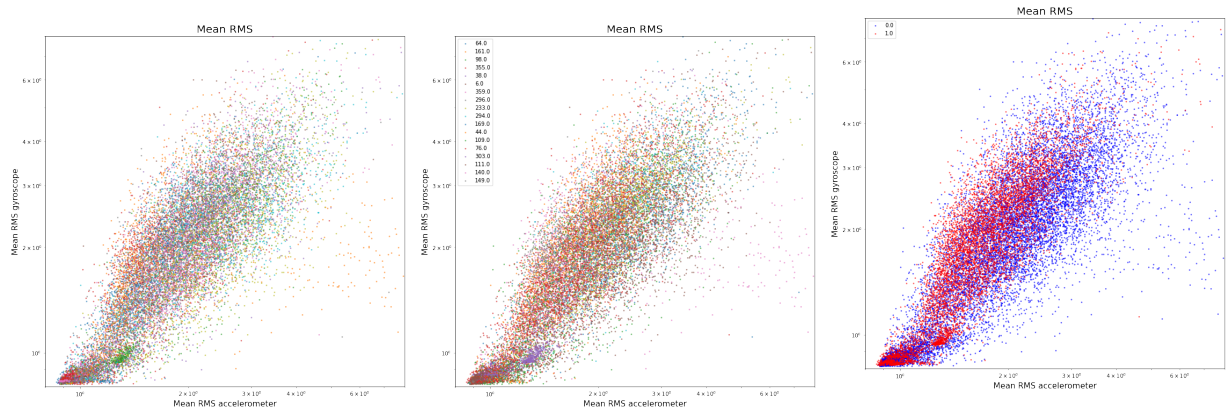


(a) PCA of the variance of each signal, color-coded based on user. (b) PCA of the variance of each signal, color-coded based on phone-model. (c) PCA of the mean of each signal, color-coded based on operative system.

Figure 14: PCA:s of the variance of each signal color-coded for different instances: user, phone-model and operative system. In all instances there are differences between the classes. Notice that the figure-axis are in log-scale.

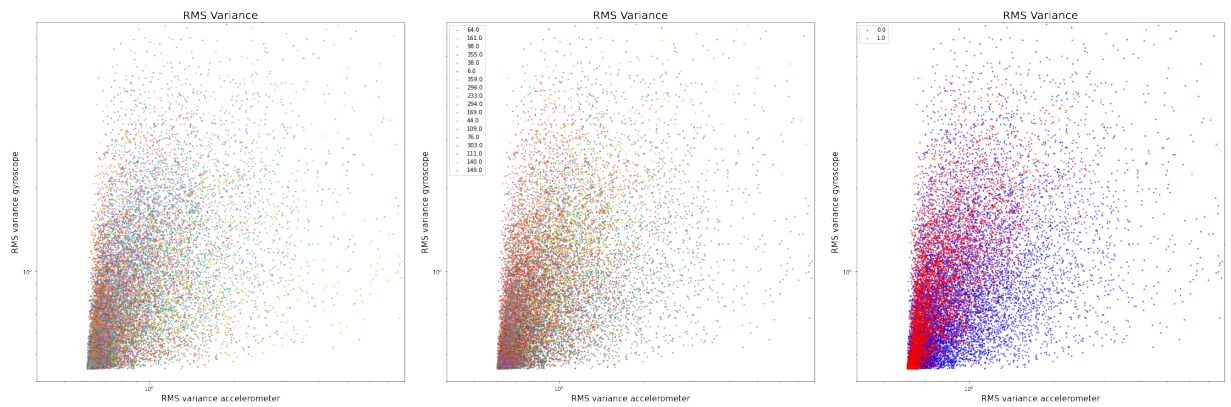
The mean RMS, seen in Figure 15, seems to be the feature that shows the least difference between Android and iOS, as they seem to be about equally distributed throughout Figure 15c. Many users are distributed over a quite large area, indicating large intraclass variation. There are however a number of classes that form quite dense clusters in the bottom left corners of the figures. About the same goes for the variance of the RMS, displayed in Figure 16, in that, that both Android and iOS are distributed over roughly the same areas (eventhough iOS is shifted a bit upwards) and that many classes are distributed over an quite large area. Looking closely at Figure 16a some classes have a quite narrow cone-like shape

with different angles out from the origin. These different angles suggest different relations between the acceleration and the rotation of the device.



(a) The mean RMS for the accelerometer and the gyroscope, color-coded based on user. (b) The mean RMS for the accelerometer and the gyroscope, color-coded based on model. (c) The mean RMS for the accelerometer and the gyroscope, color-coded based on operative system.

Figure 15: The mean RMS for each sensor, color-coded for different instances: user, model and operative system. Notice that the figure-axis are in log-scale.

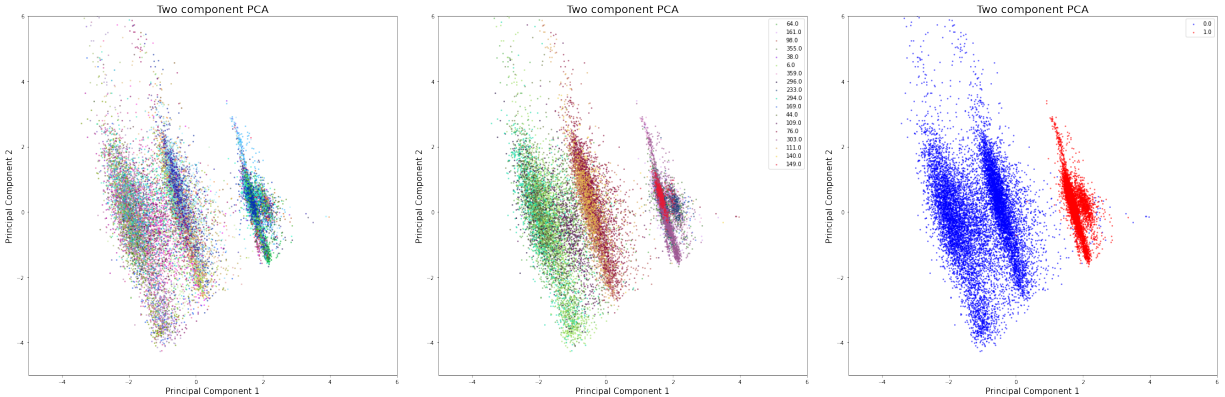


(a) The variance of the RMS of the accelerometer and the gyroscope, color-coded based on user. (b) The variance of the RMS of the accelerometer and the gyroscope, color-coded based on model. (c) The variance of the RMS of the accelerometer and the gyroscope, color-coded based on model.

Figure 16: The variance of the RMS for each sensor, color-coded for different instances: user, model and operative system.

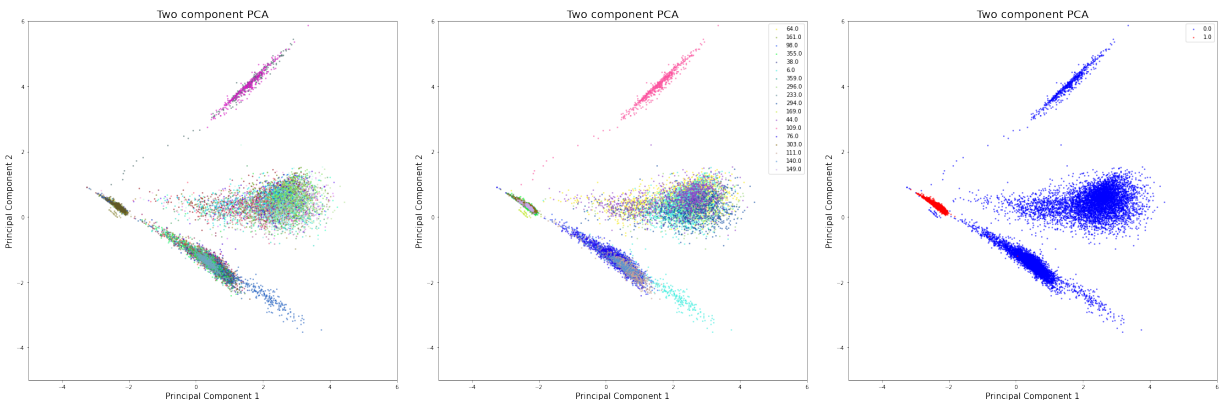
The PCA:s of the mean and variance of the spectra computed with ridge regression are displayed in Figure 17 and Figure 18, respectively. Both have rather peculiar appearances. In Figure 17, the samples are separated in four different planes. The operative systems are, with a few exceptions, completely separated. The **Android**-samples are split up in two different clusters and so is the **iOS**-samples, even though the distance apart is not as large;

in both cases these clusters are well explained by model-differences. In Figure 20 a circular shaped cluster is neighboured by two planes of samples. Also here, the samples seems to separate well after operative system and model. Noticeable is that the iOS models in Figure 20c occupy a very small area.



(a) PCA of the mean of the spectra computed with ridge regression, color-coded based on user. (b) PCA of the mean of the spectra computed with ridge regression, color-coded based on model. (c) PCA of the mean of the spectra computed with ridge regression, color-coded based on operative system.

Figure 17: PCA:s of the mean of the spectra computed with ridge regression for different instances: user, model and operative system.

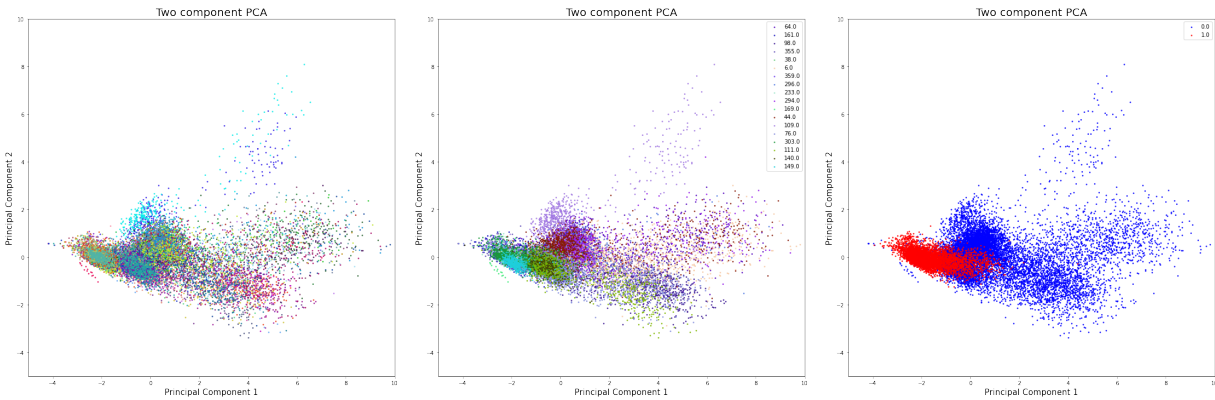


(a) PCA of the variance of the spectra computed with ridge regression, color-coded based on user. (b) PCA of the variance of the spectra computed with ridge regression, color-coded based on model. (c) PCA of the variance of the spectra computed with ridge regression, color-coded based on operative system.

Figure 18: PCA:s of the variance of the spectra computed with ridge regression color-coded for different instances: user, model an operative system.

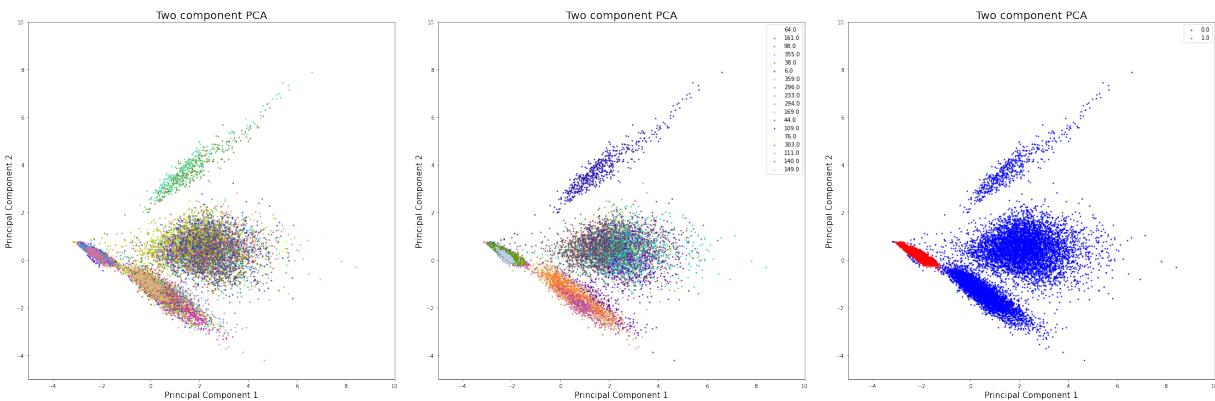
The PCA of the mean and variance of the spectra computed via resampling and FFT are shown in Figures 19 and 20. There is not the same clear separation as in Figure 17, but there is still clear differences between Android and iOS. The Android-samples are distributed over

a larger area, which, again, seems to be explained by model-differences. The same things said about Figure 18 can be said about Figure 20 as they are remarkably similar.



(a) PCA of the mean of the spectra computed via resampling and FFT, color-coded based on user. (b) PCA of the mean of the spectra computed via resampling and FFT, color-coded based on model. (c) PCA of the mean of the spectra computed via resampling and FFT, color-coded based on operative system.

Figure 19: PCA of the mean of the spectra computed via resampling and FFT, color-coded for different instances: user, model and operative system.

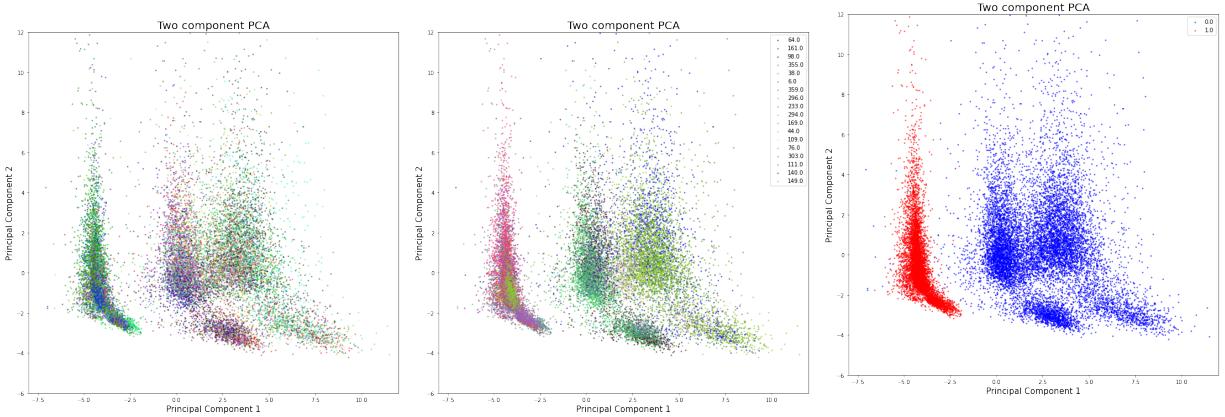


(a) PCA of the variance of the spectra computed via resampling and FFT, color-coded based on user. (b) PCA of the variance of the spectra computed via resampling and FFT, color-coded based on model. (c) PCA of the variance of the spectra computed via resampling and FFT, color-coded based on operative system.

Figure 20: PCA of the mean of the spectra computed via resampling and FFT, color-coded for different instances: user, model and operative system.

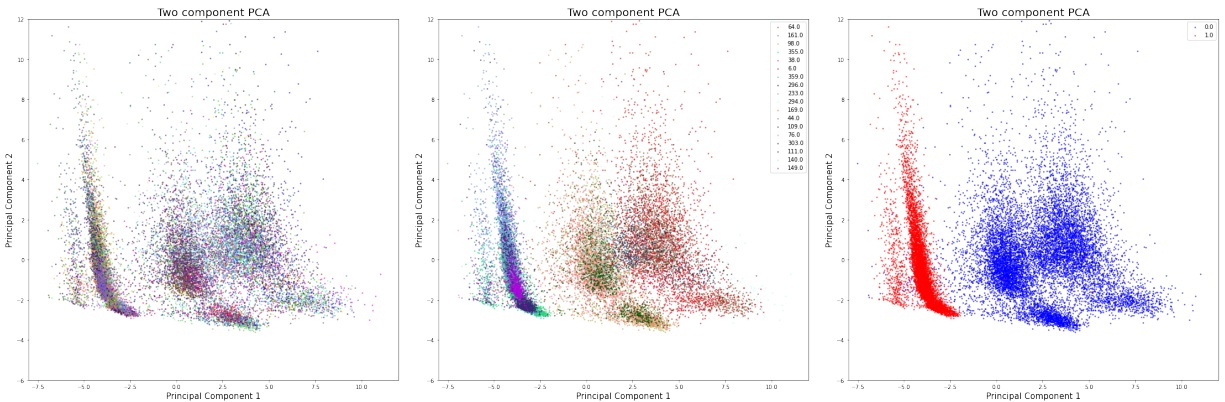
Studying Figure 21 it can be seen that the samples separates after operative system very successfully – they are (almost) completely apart. Also, it seems that the samples clusters after model, even though some models are quite widely distributed. Comparing with the corresponding samples in in Figure 21a, it seems like these spread out samples come from one and the same user.

As pointed out in **Section 3.3** it is not necessarily natural to consider the whole signal – to consider only the samples where a finger touches the screen (with some margin on each side) would also be reasonable. The PCA:s of all features, computed from these parts of the signal, are displayed in Figure 22. The figures are very similar to the figures in Figure 21 (with a different color-coding in Figure 22a and 22b), indicating that doing one way or the other does not seem to matter too much. For all features computed with a limited part of the signals, see **Appendix I**.



(a) PCA of all features combined, color-coded after user. (b) PCA of all features combined, color-coded after model. (c) PCA of all features combined, color-coded after operative system.

Figure 21: PCA of the all features combined, color-coded for three instances: user, model and operative system.



(a) PCA of all features combined, color-coded after user. (b) PCA of all features combined, color-coded after model. (c) PCA of all features combined, color-coded after model

Figure 22: PCA of all features combined, where the features are computed from only the parts of the signals where the finger touches the screen (with some margin). The figures are color-coded for three different instances: user, model and operative system.

4.2 Modified Hausdorff Distance

The EER of all of the 24 different ways of computing the MHD-matching scores are found in Table 8. It does not seem to matter much which approach is taken as the EER:s does not differ much – the largest absolute difference is 0.037. The results are not very impressive either and are well above the EER:s achieved in [11]. Due to the quite poor results, exploring the performance of MHD for only accelerometer and gyroscope data was not explored. For further results, see **Appendix II**.

Table 8: The EER:s for the 24 different ways of computing matching scores.

| | Channel-wise | | | Sensor-wise | | |
|---------|--------------|---------|-------|-------------|---------|-------|
| | Standard | Min-Max | tanh | Standard | Min-Max | tanh |
| Min | 0.452 | 0.456 | 0.452 | 0.454 | 0.452 | 0.454 |
| Max | 0.441 | 0.448 | 0.441 | 0.451 | 0.46 | 0.451 |
| Sum | 0.444 | 0.445 | 0.444 | 0.45 | 0.455 | 0.45 |
| Product | 0.462 | 0.447 | 0.444 | 0.478 | 0.45 | 0.45 |

4.3 Supervised Learning with Spectrograms

In order to measure the usefulness of spectrograms computed in different ways, the accuracy, F1-score, and loss of the of the networks described in **Section 3.6** predicting on a validation set was used. The results for various ways of computing the spectrograms are shown in Table 9. There is a clear trend, that to treat each signal separately, gives better performance than treating the norm of the signals from each sensor. There is not the same clear trend when comparing ridge regression and FFT. The scores from these are distributed in the same range, with ridge regression in both ends of the interval (noticeably, the approach that performs best on **iOS**-user is the one performing worst on **Android**-users (not taking the approaches where the signals treated are the norm of the signals of each sensor)). It was noticed that computing spectrograms with the approaches using FFT was considerably faster than ridge regression, which perhaps is not very surprising. This of course favors the FFT approaches. Based on these results a few other variations was tried as well (it was not considered necessary to try the new variations for the variants with less promising results). The results of these are displayed in Tables 10 and 11.

The approach deemed best is marked and found in Table 10. The result for this approach on the test-data is found in Table 12. The results are similar to those from validation data, and are for both **Android** and **iOS** well above the expected result from a naive classifier making predictions at random ($1/42 \approx 0.024$ and $1/18 \approx 0.056$).

Table 9: Results for different ways of computing the spectrograms. The numbers of windows are 35 and the (maximum) window size is 5. The frequencies are 0, 1, 2, ..., 49. X represents a 'yes' and - a 'no'. Ridge regression is shortened RR. Approach refers to approaches described in **Section 3.5**.

| Spectrograms | | | | Android | | | iOS | | |
|--------------|-------|-----------|---------------|----------|----------|------|----------|----------|------|
| Approach | L^2 | Resampled | Spectrum with | Accuracy | F1-score | Loss | Accuracy | F1-score | Loss |
| 1 | X | - | RR | 0.28 | 0.25 | 2.63 | 0.37 | 0.35 | 2.21 |
| 2 | X | - | RR | 0.26 | 0.24 | 2.77 | 0.41 | 0.4 | 1.98 |
| 1 | X | X | RR | 0.34 | 0.33 | 2.48 | 0.36 | 0.34 | 2.26 |
| 2 | X | X | RR | 0.29 | 0.27 | 2.69 | 0.38 | 0.35 | 2.05 |
| 1 | - | - | RR | 0.39 | 0.38 | 2.51 | 0.52 | 0.5 | 2.64 |
| 2 | - | - | RR | 0.29 | 0.27 | 2.68 | 0.61 | 0.6 | 1.68 |
| 1 | - | X | RR | 0.37 | 0.36 | 2.44 | 0.53 | 0.52 | 1.67 |
| 2 | - | X | RR | 0.33 | 0.32 | 2.43 | 0.59 | 0.58 | 1.71 |
| 1 | X | X | FFT | 0.33 | 0.31 | 2.37 | 0.38 | 0.33 | 2.1 |
| 2 | X | X | FFT | 0.28 | 0.24 | 2.53 | 0.40 | 0.39 | 2.01 |
| 1 | - | X | FFT | 0.38 | 0.36 | 2.61 | 0.53 | 0.52 | 1.61 |
| 2 | - | X | FFT | 0.31 | 0.29 | 2.6 | 0.56 | 0.55 | 1.54 |

Table 10: Additional results for another way of computing the spectrograms. The number of windows is 30 and the (maximum) window length is 10 and the frequencies are 0,1,2,...,49.

| Spectrograms | | | | Android | | | iOS | | |
|--------------|-------|-----------|---------------|-------------|-------------|-------------|-------------|-------------|-------------|
| Approach | L^2 | Resampled | Spectrum with | Accuracy | F1-score | Loss | Accuracy | F1-score | Loss |
| 1 | - | X | FFT | 0.37 | 0.36 | 2.59 | 0.54 | 0.53 | 1.59 |
| 2 | - | X | FFT | 0.37 | 0.36 | 2.35 | 0.58 | 0.57 | 1.52 |

Table 11: Additional results for yet another way of computing the spectrograms. The number of windows is 30 and the (maximum) window length is 10 and the frequencies are 0,1/2,1,...,39/2.

| Spectrograms | | | | Android | | | iOS | | |
|--------------|-------|-----------|---------------|----------|----------|------|----------|----------|------|
| Approach | L^2 | Resampled | Spectrum with | Accuracy | F1-score | Loss | Accuracy | F1-score | Loss |
| 1 | - | - | FFT | 0.27 | 0.27 | 3.1 | 0.48 | 0.46 | 1.78 |
| 2 | - | - | FFT | 0.26 | 0.23 | 2.9 | 0.52 | 0.51 | 1.7 |

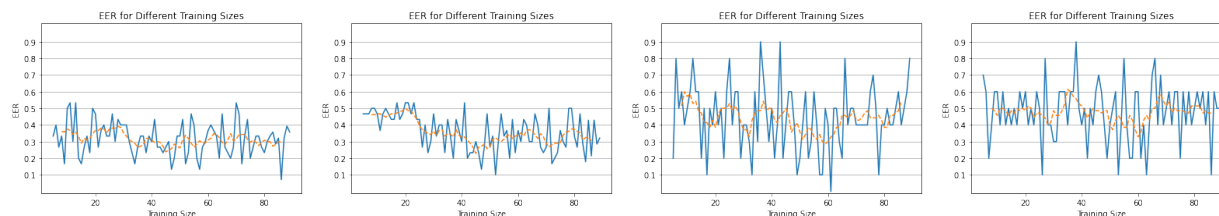
Table 12: The results of the chosen approach on test-data (the number of windows is 30 and the (maximum) window length is 10 and the frequencies are 0,1,2,...,49).

| Spectrograms | | | | Android | | | iOS | | |
|--------------|-------|-----------|---------------|----------|----------|------|----------|----------|------|
| Approach | L^2 | Resampled | Spectrum with | Accuracy | F1-score | Loss | Accuracy | F1-score | Loss |
| 2 | - | X | FFT | 0.36 | 0.34 | 2.16 | 0.55 | 0.54 | 1.6 |

4.4 Semi-supervised learning with Spectrograms

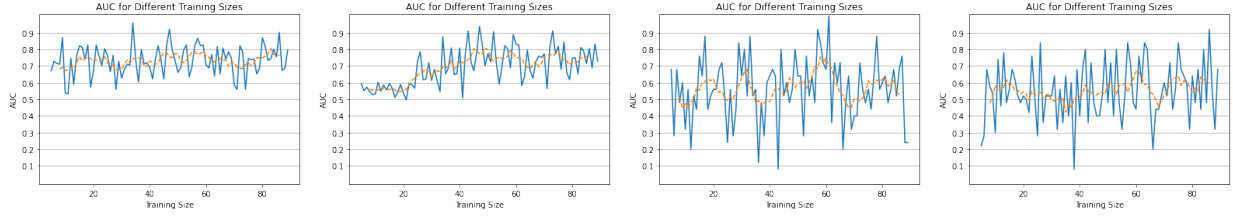
The results of applying KDE and LOF (due to poor performance one-class SVM was not granted space in this part of the report) to outputs of the networks derived in the previous section are displayed in Figures 23-25. The input of the networks are the validation set, see Table 6. Figure 23 shows the EER for one-step-ahead prediction. There is a clear difference between **Android** and **iOS**, the **iOS**-signals are much more noisy, due to a lower support, but also the general performance is better for **Android**. The same pattern is present in Figure 24, which shows the area-under-the-curve score (AUC-score) for the different combinations of operative system and model.

Comparing Figure 25a and 25c it also becomes clear that the procedure for **Android**-users works better. The histograms in 25a has the differences that would be expected for a somewhat successful model. The differences between the histograms in 25c are very small. The differences between 25b and 25d can formulated in a similar way.



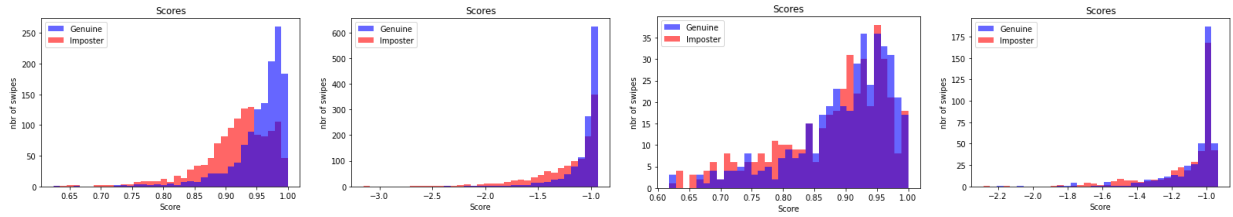
(a) EER and its trend for **Android** using KDE. (b) EER and its trend for **Android** using LOF. (c) EER and its trend for **iOS** using KDE. (d) EER and its trend for **iOS** using LOF.

Figure 23: EER for one-step-ahead prediction using KDE and LOF for **Android** and **iOS**, separately. The support for **Android** is 15 and for **iOS** is 5, giving a more noisy signal.



(a) AUC and its trend for **Android** using KDE. (b) AUC and its trend for **Android** using LOF. (c) AUC and its trend for **iOS** using KDE. (d) AUC and its trend for **iOS** using LOF.

Figure 24: AUC for one-step-ahead prediction using KDE and LOF for **Android** and **iOS**, separately. The support for **Android** is 15 and for **iOS** is 5, giving a more noisy signal.



(a) Distribution of score for **Android** using KDE. (b) Distribution of score for **Android** using LOF. (c) Distribution of score for **iOS** using KDE. (d) Distribution of score for **iOS** using LOF.

Figure 25: Histograms showing the distributions of all scores from all iterations using KDE and LOF. **Android** and **iOS** are treated separately.

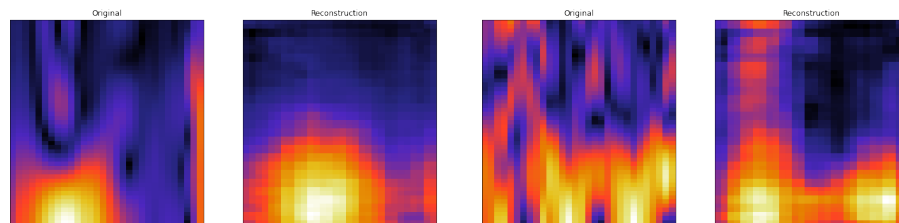
4.5 Unsupervised Learning with Spectrograms

A number of different autoencoders were designed and trained. After compressing the spectrograms to feature space the three methods for unsupervised classification described in **Section 2.7** were applied. The models were given quite large sets for training. The results are displayed in Table 13.

Table 13: Three different EER-scores for different unsupervised classification models for a number of autoencoders with different design. The one with the best performance is marked. 'Latent Space', in the column names, refers to the size of the latent space. The different 'Layer' headings refer to the number of kernels for the convolutional layer and λ_{conv} and λ_{dense} refers to the size of the regularisation for the L^2 -penalty term added to the convolutional layer and the dense layer, respectively.

| Autoencoder | | | | | | | EER | | | |
|--------------|-----------|-----------|---------|-------------|------------------|-------------------|------------|--------------|--------------|--------------|
| Latent Space | Layer 1 | Layer 2 | Layer 3 | Activation | λ_{conv} | λ_{dense} | Loss | SVM | LOF | KDE |
| 128 | 64 | 32 | - | tanh | 10^{-3} | 0 | 0.13 | 0.496 | 0.378 | 0.378 |
| 128 | 64 | 32 | - | tanh | 10^{-3} | 10^{-3} | 0.28 | 0.495 | 0.374 | 0.36 |
| 128 | 64 | 32 | 32 | tanh | 10^{-3} | 0 | 0.22 | 0.494 | 0.359 | 0.356 |
| 128 | 64 | 32 | 32 | ReLU | 10^{-3} | 0 | 0.27 | 0.486 | 0.361 | 0.394 |
| 64 | 32 | 64 | 64 | tanh | 10^{-3} | 0 | 0.22 | 0.496 | 0.361 | 0.341 |
| 64 | 64 | 32 | - | tanh | 10^{-4} | 10^{-4} | 0.18 | 0.489 | 0.34 | 0.326 |
| 64 | 32 | 16 | - | tanh | 10^{-4} | 0 | 0.16 | 0.493 | 0.361 | 0.354 |
| 64 | 64 | 32 | 32 | tanh | 10^{-3} | 0 | 0.26 | 0.502 | 0.368 | 0.363 |
| 64 | 32 | 64 | 64 | ReLU | 10^{-3} | 0 | 0.22 | 0.406 | 0.353 | 0.404 |
| 32 | 64 | 32 | - | tanh | 10^{-4} | 0 | 0.2 | 0.496 | 0.347 | 0.323 |
| 16 | 32 | 16 | - | tanh | 10^{-4} | 0 | 0.3 | 0.503 | 0.309 | 0.344 |
| 16 | 32 | 64 | - | tanh | 10^{-4} | 0 | 0.29 | 0.499 | 0.345 | 0.362 |
| 8 | 32 | 16 | - | tanh | 10^{-3} | 0 | 0.42 | 0.489 | 0.321 | 0.359 |
| 8 | 32 | 64 | - | tanh | 10^{-3} | 0 | 0.4 | 0.494 | 0.364 | 0.371 |

The best performing autoencoder is one with a 16-dimensional latent space. A couple of examples of the original spectrograms and their reconstruction can be seen in Figure 26. The PCA:s of the feature space of the spectrograms for the validation data (see Table 6) is displayed in Figure 27. It looks quite pleasing as the users are quite spread all-in-all, but in most cases the spread of one user is quite small. However, as 27c shows, the operative systems separate quite much, implying that there is an underlying phone-model dependency. The samples are centered somewhat more to the left than the right.



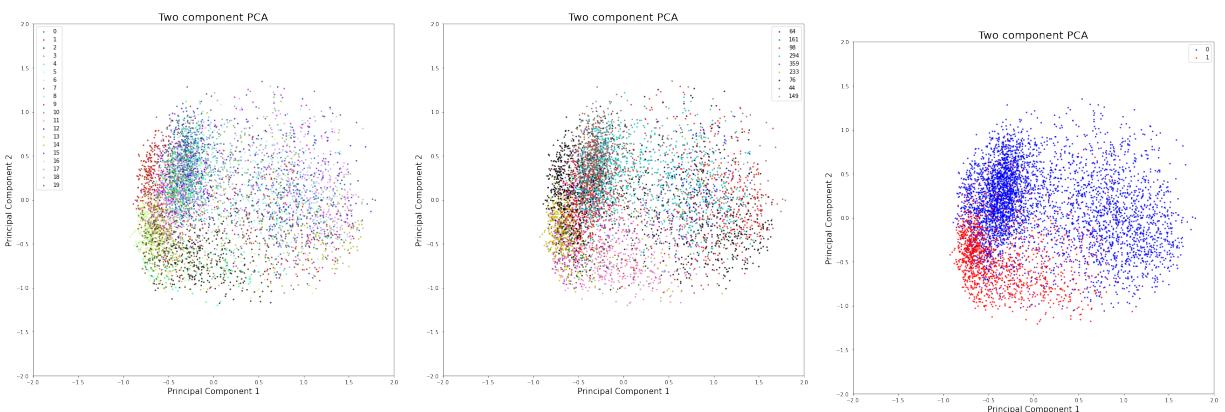
(a) An original spectrogram and its reconstruction.

(b) Another original spectrogram and its reconstruction.

Figure 26: Examples of original spectrograms and their reconstruction. The reconstructions do not capture the fine structure of the original image but overall has the same appearance.

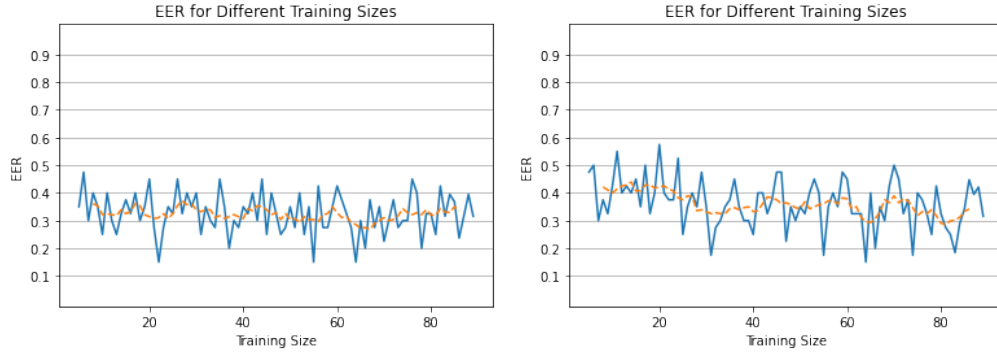
The EER and AUC-score for one-step-ahead prediction for KDE and LOF are displayed in Figures 28 and 29 (due to its poor performance during the previous step, one-class SVM was not considered for this part). Also, the distribution of all scores from all iterations are displayed in Figure 30. Surprisingly, the EER:s are now lower for KDE than it is for LOF, which is the opposite result from Table 13! KDE performs better when it comes to AUC as well. It is especially for the first 25-30 iterations KDE outperforms LOF. It is unexpected that there is not a trend of improving with an increasing training size for the KDE. For LOF, however, there is an tendency of improvement for the 25-30 first iterations.

There is a clear difference between the genuine scores and the scores of the simulated imposter. The histogram in Figure 30a looks quite pleasing with the central mass of the genuine scores located to the right and to the left for the scores of the imposters. In Figure 30b the genuine scores still are somewhat more centered to the right than the imposter ones, with the blue staple at -1 being significantly higher than the red. All-in-all, the KDE is deemed to perform better and thus chosen for the test data. To see the corresponding results when using a different approach for computing the spectrogram, the other approach in Table 10, see **Appendix III**.



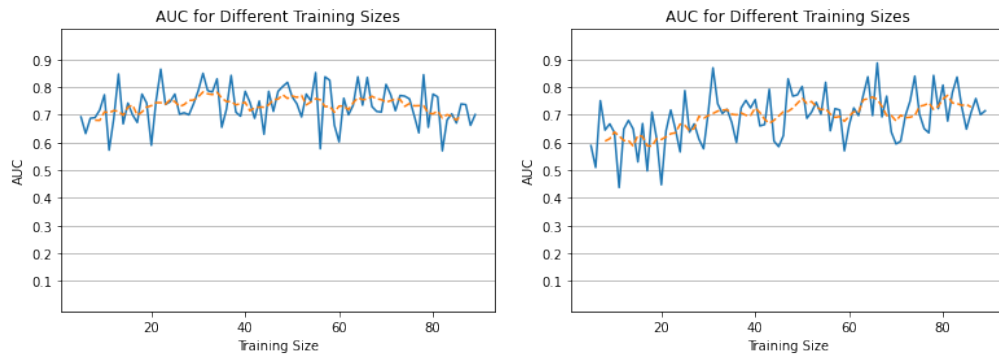
(a) PCA of the spectrograms in latent space, colorcoded after user. (b) PCA of the spectrograms in latent space, colorcoded after phone-model. (c) PCA of the spectrograms in latent space, colorcoded after operative system.

Figure 27: PCA of the latent space of the spectrograms in the validation set for three different instances: user, phone-model and operative system. The samples are spread in a circle that is a bit heavier to the left.



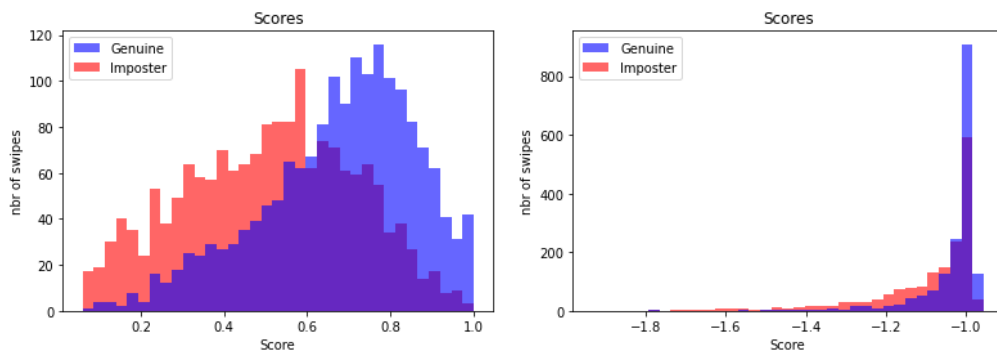
(a) EER and its trend for different training sizes when using KDE. (b) EER and its trend for different training sizes when using LOF.

Figure 28: EER:s depending on training size for the different models.



(a) AUC and its trend depending on training size when using KDE. (b) AUC and its trend depending on training size when using LOF.

Figure 29: AUC:s depending on training size for the different models.

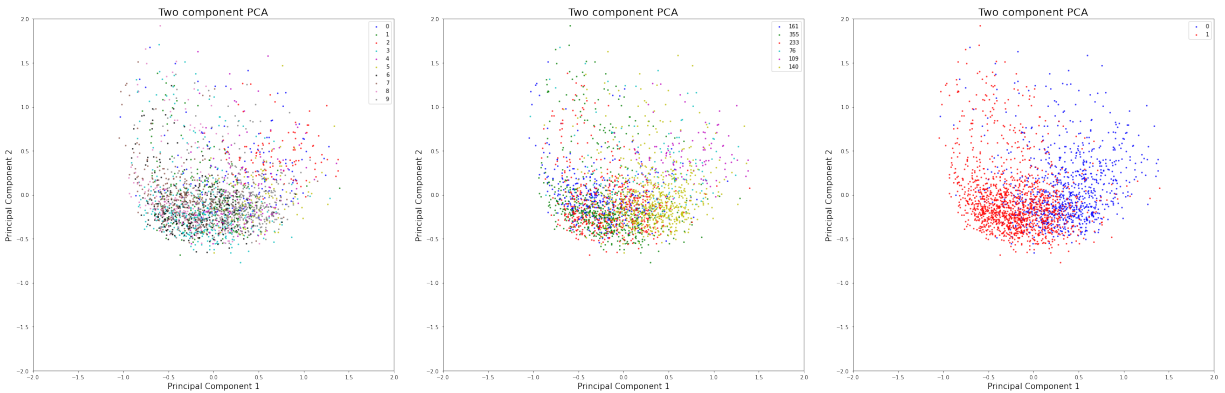


(a) Distributions of scores from all iterations using KDE. (b) Distributions of scores from all iterations using LOF.

Figure 30: Histograms showing the distributions of scores from genuine users and simulated imposters. The genuine-scores are centered further to the right in both cases.

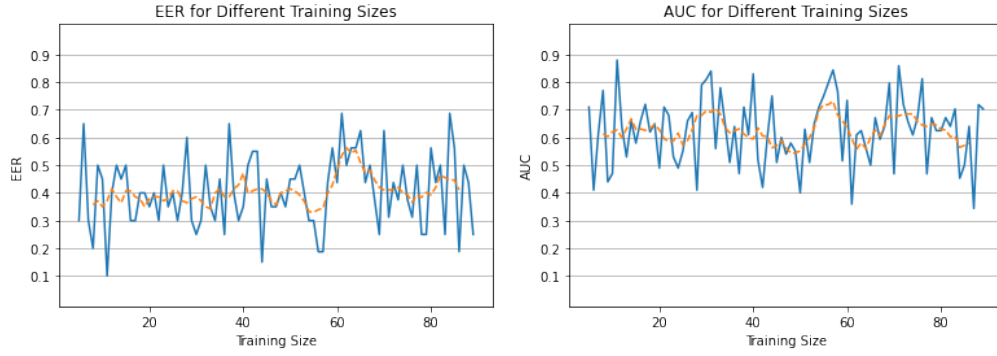
The PCA of the feature representation of the spectrograms of the test data are displayed in Figure 31. There is a larger proportion of the samples in the dense area of the large cluster than in the PCA in Figure 27.

The result of the predictions of the KDE can be seen in Figure 32. The results are clearly worse than for the evaluation data. The trend of the EER is even above 0.5 for a few training sizes. Both the EER and AUC are also generally higher and lower, respectively, than their validation counterparts. Comparing the histograms, the imposter-scores are clearly shifted to the right, making the unwanted overlap larger.

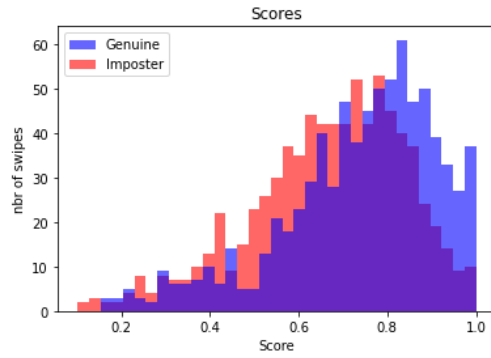


(a) PCA of the spectrograms in latent space, colorcoded after user. (b) PCA of the spectrograms in latent space, colorcoded after phone-model. (c) PCA of the spectrograms in latent space, colorcoded after operative system.

Figure 31: PCA of the latent space of the spectrograms in the test set for three different instances: user, phone-model and operative system. The samples are distributed in a circle that is a quite heavy at the bottom.



(a) EER and its trend depending training size for the test set, using KDE. (b) AUC and its trend depending training size for the test set, using KDE.



(c) Distributions of scores from all iterations for the test-set, using KDE.

Figure 32: The EER, AUC and distributions of scores for the test set.

5 Discussion

5.1 Features

The means of the signals does not tell to much. It can be seen that the swipes from **Android**-phones are a bit more dispersed than those from **iOS**, when considering the mean-feature. For the variance-feature, the classes in the different instances are cone-shaped, pointing out from the origin in different directions. It seems that the apparent discriminating tendencies for the users are due to underlying phone-model-differences. When the models are bundled together in their operative systems this is even more clear.

Two of the features that show the least tendency for clustering for any instance are the RMS-features. RMS, that is proportional to the L^2 -norm (the L^2 -norm would have made an equivalently reasonable choice), measures the the size of the signals from each sensor. It is a measure of the size of the phones acceleration and its rotation. A high mean RMS thus corresponds to a swipe where the phone has been through a lot of motion. As many users are distributed all over Figure 15, it implies that the intraclass variation is quite large. This is very natural, as the samples comes from real conditions – the conditions and environment for when a swipe is being performed may vary quite a lot which naturally would effect the movements of the phone. The same picture is painted by the variance of the RMS. Even though this rather measures the variability of the swipe, high variance would correspond to vivid, and changing, movement.

The features that shows the most clustering are the frequency related features. The similarity between Figures 18 and 20 are quite remarkable, as well as reasonable as they are supposed to represent the same things. Why there is such similarity between the variances of the spectra and not between the means is quite hard to tell. Small differences, however, can cause changes of the principal components which would potentially give different representation in two dimensions, even though being similar in six dimensions. This is perhaps not a satisfactory explanation as there might be more plausible causes of the dissimilarity.

The feature extraction part really shows the importance of understanding what causes the differences between users, as there are clear and significant differences between users. The differences are in most cases caused by that the swipes are performed on different devices, that are of different brands and models. As intruders are simulated by other users it would for certain be easy to separate these in most cases and thus achieve good results. Such results would of course be false and faulty as they do not in a good way represent the security against a real attack. The measure taken to deal with this is to only use users that have phones with the same operative system as simulated intruders. It can be questioned if this is sufficient. Especially **Android**-models show great diversity and the same points made generally could be made for just **Android**. To divide further, based on model or branch, would be possible but cumbersome and not necessarily more robust. It would create six or 20 different pools instead of two, where the number of users of each brand or model varies. Some models and brands are used by only a single or a few users (actually, only two of the ten brands are used by more than six users). It would hence be necessary to cross the brand- or model-boundary

anyway in some cases. Nevertheless, some separation based on brand would be reasonable. Luckily, iOS-phones are of only one brand (they are all iPhones, of various models) and separating this brand out from the others leaves the same set when separating iOS from Android. The different iOS-models have a small spread, meaning that users with another iOS-model can pose as credible intruders. That iOS has a very small spread compared with Android is not surprising. Android-phones comes from many different manufacturers with, one can expect, varying hardware quality whilst iOS, more or less, are versions of the same phone.

5.2 Modified Hausdorff Distance

The MHD did not work nearly as well in this work as it did in [11] and it did not work as well as when using spectrogram-based approaches, even though an additional data source was used for MHD. The setting in this work is of course very different from that in [11], where the swipes are collected in a controlled environment and from only one device at the time. Also, the 3% EER for the accelerometer was achieved when considering several movements together which reasonably would boost the performance, while in this work, only one swipe at the time is considered. Furthermore, information from the touchscreen was used in [11].

MHD intuitively seems like a quite blunt tool. It takes no consideration of the time data and thus no consideration of the order of the points. For the first sample in one signal, it may be that the last sample, of the signal it is compared with, is the closest. It is unable to find, and for that matter compare, structures that are dependent of order. MHD is, in this case, a comparison of how much the ranges of the signals differ. The capability of such a tool for a problem of this complexity seems limited, even if the results of [11] would provide arguments for the opposite.

Also, orientation might be a dubious feature to base authentication on. One of the three numbers for a orientation measurement represents the angle between the direction the phone points in and the longitudes of the earth. If a user, for instance, sits by the same desk when swiping it would be reasonable to assume that the phone points in roughly the same direction. This information does not really tell much about the user. Information that have global bearing, that can recognize the user away from the desk as well, is desirable.

5.3 Spectrograms and Spectrogram-based Classification

As long as all signals are compared separately it does not seem to matter too much which method for computing spectrograms that is chosen. There is no clear tendency that there would be a significant difference between computing the spectrograms with ridge regression or via resampling followed by FFT or ridge regression. The resampled approaches have the intuitively appealing property that each spectrum of a spectrogram represents the same amount of time, which is not the case when using ridge regression solely. The approaches treats large gaps in the signal differently – with ridge regression, the time intervals are distorted and when resampling, the gap is filled with new points via interpolation. The latter is not problem-free either. Adding information where there is not any may introduce false

structures. In that sense, ridge regression is preferable as it does not alter the original data. Again, these differences does not seem to matter much.

What is noticeable is that the trained CNN:s clearly performs better for `iOS` than for `Android`. Intuitively, the opposite would have been expected. As the `Android` samples throughout **Section 4.1** shows greater diversity and clusters after model, it would seem like this would grant easier classification. It might be that the diversity causes the problem. For a data set of greater diversity, the network needs to be able to generalize better and may thus lose capability of being more specific. The `iOS`-set, that seemingly are of lower diversity, may be able to find the more specific differences between users. On the other hand, there are more users, i.e. classes, in the `Android`-set. Comparing the ratios of the achieved test-accuracies with the expected results of a naive classifier (15.21 and 9.9, respectively) the network for `Android` overperforms more than the network for `iOS`. The former theory, however, goes well with the results in **Section 4.4**, where spectrograms of new users are presented to the networks. Here, the results for `Android` are clearly better than the results for `iOS`. If the generalization properties of the `Android`-network are better, it may be better suited for predicting new samples. Actually, the diversity of the users the network is trained on may in itself be the reason for the (relative) better performance. As the output of the network, i.e the feature space, are the probabilities of the input being from a certain user, more diversity among the user will imply a feature space that is closer to being orthogonal.

The autoencoder chosen in **Section 4.5** can decently reconstruct a spectrogram from a 16-dimensional latent space. As indicated by Table 13, there are autoencoders that probably could do a better reconstruction as their loss is considerably lower, but this is of course not the purpose of the autoencoder. The purpose is to get a good low dimensional representation. In the PCA of the outputs of the autoencoder, there is a difference between `Android` and `iOS`. Again, it is hard to tell if the clusters of the lower instances depends on model- or user-differences.

The histograms in Figure 30a satisfactory shows that the scores the genuine users and the scores of an imposter, for the validation set, comes from two different distributions. The corresponding Figure 32c is not as satisfactory. There are two distributions, but they do not differ as much and the results of the other measures, the EER and AUC, are clearly worse too. A poorer result for the test data compared with validation data is expected – some overfitting is inevitable. The difference is, however, larger than what could be expected. Comparing Figure 27 and 31, the latter seems more dense and would thus be harder to predict correctly, which may pose as an additional explanation.

6 Conclusions

The most important insight from this work is the importance of the role phone-model and operative system plays. The differences of these causes apparent differences on user level. It is of great importance to be aware of this, as it might be very easy differing two users swiping on phones of different models, but hard to distinguish them if swipes are conducted on the same phone.

In difference from many previous studies of authentication through behavioural biometrics, the data in this work was not from a controlled environment. Methods successfully deployed for touch-screen gestures in such environments have been refuted, implying that the real problem is much harder than the results from such studies might suggest.

The approach taken with spectrograms is one way of trying to extract potential information from the swipes and succeeds of doing this to some extent. Especially the supervised learning part shows that there ought to be much information stored in spectrograms, even though it is harder to exploit in an unsupervised setting. The difficult question to answer is, if it is user information that has been extracted.

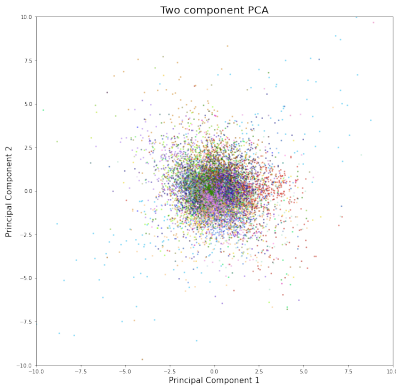
References

- [1] MathWorks, *Accelerometer*, image, MathWorks Help Center, Viewed May 31 2021, <https://www.mathworks.com/help/supportpkg/android/ref/accelerometer.html>
- [2] Callsign, *About Us*, Callsign Webpage, Viewed May 31 2021, <https://callsign.com/about/>
- [3] M. Abuhamad, A. Abusnaina, D.n Nyang, and D. Mohaisen, *Sensor-based Continuous Authentication of Smartphones' Users Using Behavioral Biometrics: A Contemporary Survey*, ArXiv, vol. abs/2001.08578, 2020.
- [4] T. Nohara and R. Uda, *Personal identification by flick input using selforganizing maps with acceleration sensor and gyroscope*, in Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication. ACM, 2016, p. 58.
- [5] H. Saevanee and P. Bhatarakosol, *User authentication using combination of behavioral biometrics over the touchpad acting like touch screen of mobile device*, in 2008 International Conference on Computer and Electrical Engineering. IEEE, 2008, pp. 82–86.
- [6] . Xu, Y. Zhou, and M. R. Lyu, *Towards continuous and passive authentication via touch biometrics: An experimental study on smartphones*, in 10th Symposium On Usable Privacy and Security (SOUPS 2014), 2014, pp. 187–198.
- [7] K. W. Nixon, X. Chen, Z.-H. Mao, and Y. Chen, *Slowmo-enhancing mobile gesture-based authentication schemes via sampling rate optimization*, in 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, 2016, pp. 462–467.
- [8] J. Nader, A. Alsadoon, P. Prasad, A. Singh, and A. Elchouemi, *Designing touch-based hybrid authentication method for smartphones*, Procedia Computer Science, vol. 70, pp. 198–204, 2015
- [9] M. Antal and L. Z. Szabo, “Biometric authentication based on touch- ´ screen swipe patterns,” Procedia Technology, vol. 22, pp. 862–869, 2016.
- [10] S. Mondal and P. Bours, *Continuous authentication and identification for mobile devices: Combining security and forensics*, in 2015 IEEE International Workshop on Information Forensics and Security (WIFS). IEEE, 2015, pp. 1–6
- [11] A. Jain and V. Kanhangad, *Exploring orientation and accelerometer sensor data for personal authentication in smartphones using touchscreen gestures*, Pattern recognition letters, vol. 68, pp. 351–360, 2015
- [12] Wikipedia, *Hausdorff distance*, image, Wikipedia - The Free Encyclopedia, Viewed June 1 2021, https://en.wikipedia.org/wiki/Hausdorff_distance
- [13] M-P. Dubuisson and A. Jain, *A Modified Hausdorff Distance for Object Matching*, Proceedings of 12th International Conference on Pattern Recognition, vol. 1, pp. 566-568, 1994.

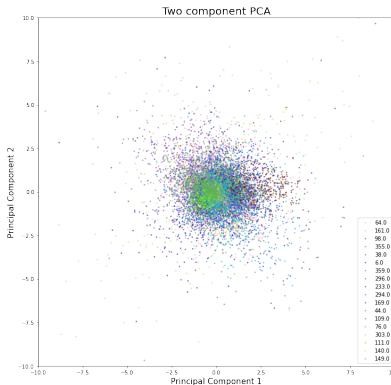
- [14] F. Mignard, *About the Nyquist Frequency*, Gaia-FM-022, 2005.
- [15] J. Burkill, *Artificial Neural Networks – Part 2: MLP Implementation for XOR*, image, Machine Learning and Optimization, Viewed 30 May 2021, <https://www.mlopt.com/?tag=multilayer-perceptron>
- [16] Wikipedia, *Sobel operator*, image, Wikipedia - The Free Encyclopedia, Viewed May 30 2021, https://en.wikipedia.org/wiki/Sobel_operator
- [17] A. Chakure, *Convolutional Neural Networks (CNN) in a Brief*, image, DEV Community, Viewed May 30 2021, <https://dev.to/afrozchakure/cnn-in-a-brief-27gg>
- [18] A. Dertat *Applied Deep Learning - Part 3: Autoencoders*, image, towards data science, Viewed May 30 2021, <https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798>
- [19] B. Schölkopf, J Platt, J. Shawe-Taylor and A. Smola, *Estimating Support of a High-Dimensional Distribution*, Neural Computation, vol. 13, pp. 1443-1471, 2001.
- [20] M. Breunig, H-P. Kriegel, R. Ng, J. Sander, *LOF: Identifying Density-Based Local Outliers*, SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data, pp. 93-104, 2000.
- [21] Jake Vanderplas, *Density Estimation*, image, scikit-learn, Viewed June 20 2021, <https://scikit-learn.org/stable/modules/density.html>

Appendix I

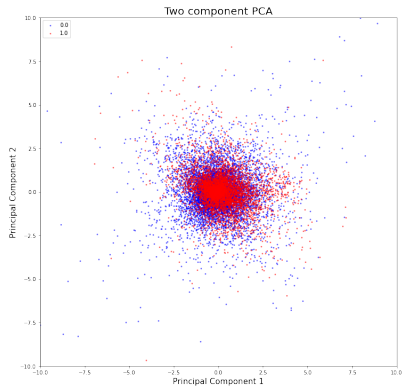
Figures 33-40 shows the PCA:s of the features computed with the parts when the finger touches the screen, with some margin in each side.



(a) PCA:s of the mean of each signal, colorcoded based on user.

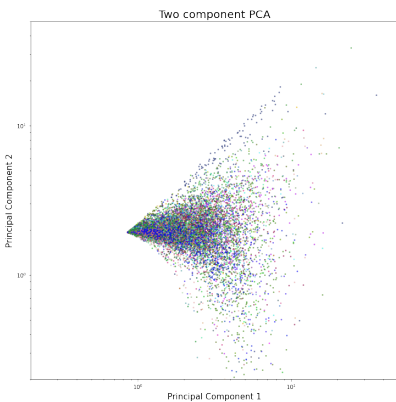


(b) PCA of the mean of each signal, colorcoded based on model.

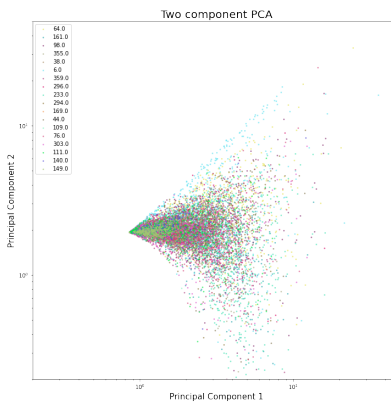


(c) PCA of the mean of each signal, colorcoded based on operative system (Android is marked blue and iOS red).

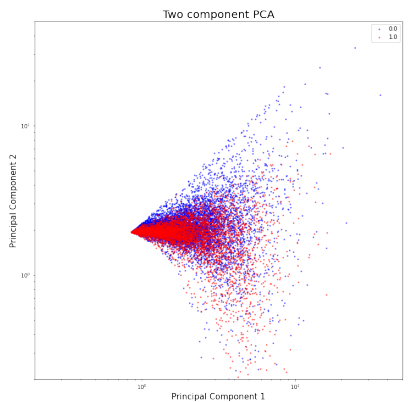
Figure 33: PCA:s of the mean of each signal colorcoded for different instances: user, phone-model and operative system.



(a) PCA of the variance of each signal, colorcoded based on user.

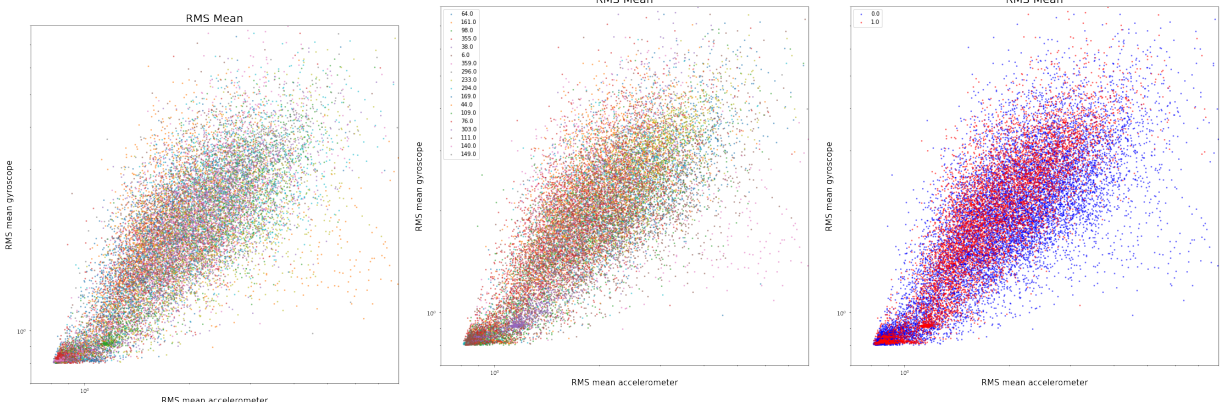


(b) PCA of the variance of each signal, colorcoded based on model.



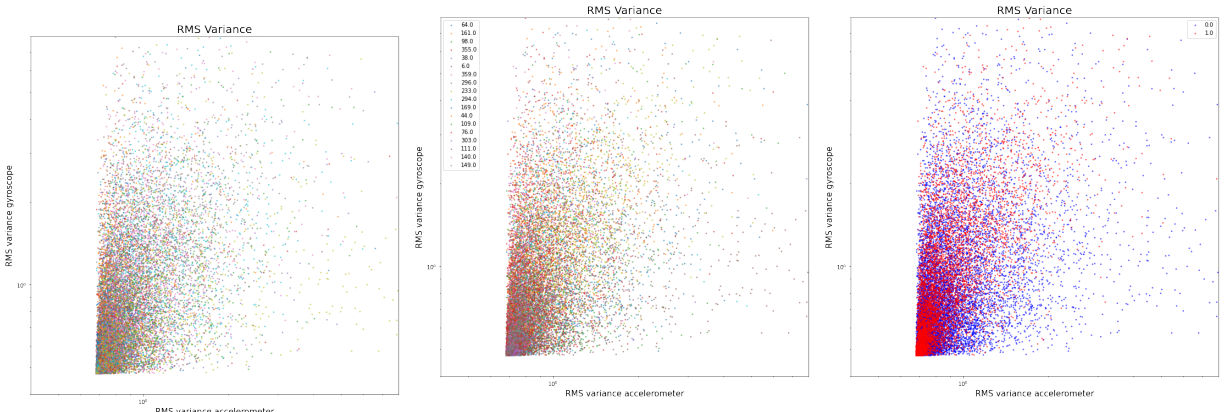
(c) PCA of the mean of each signal, colorcoded based on operative system.

Figure 34: PCA:s of the variance of each signal colorcoded for different instances: user, phone-model and operative system. In all instances there are differences between the classes. Notice that the figure-axis are in log-scale.



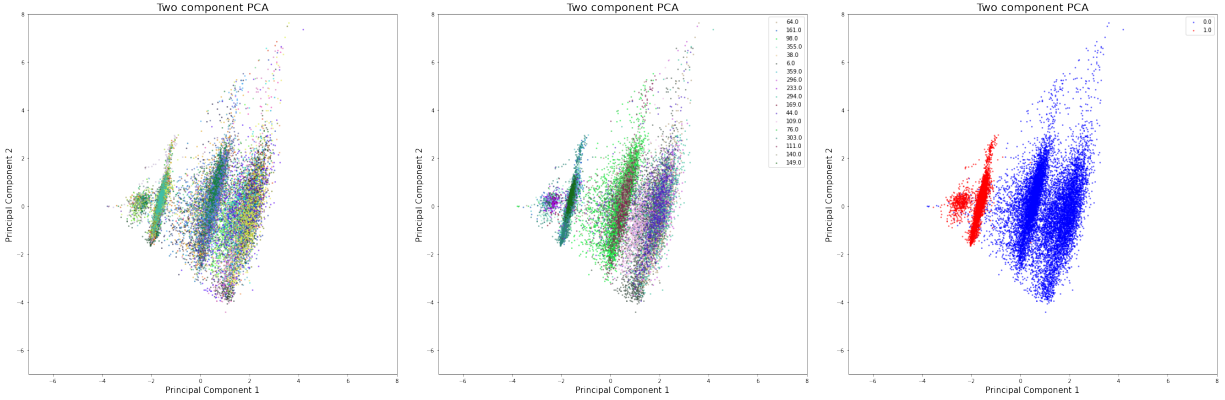
(a) The mean of the RMS for the accelerometer and the gyroscope, colorcoded based on user. (b) The mean of the RMS for the accelerometer and the gyroscope, colorcoded based on model. (c) The mean of the RMS for the accelerometer and the gyroscope, colorcoded based on operative system.

Figure 35: The mean of the RMS for each sensor, colorcoded for different instances: user, model and operative system. Notice that the figure-axis are in log-scale.



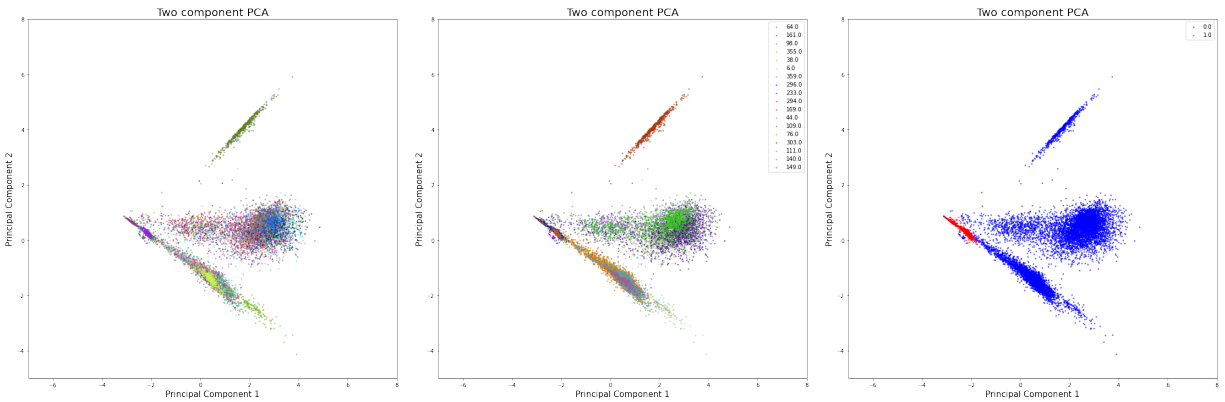
(a) The variance of the RMS of the accelerometer and the gyroscope, colorcoded based on user. (b) The variance of the RMS of the accelerometer and the gyroscope, colorcoded based on model. (c) The variance of the RMS of the accelerometer and the gyroscope, colorcoded based on model.

Figure 36: The mean of the RMS for each sensor, colorcoded for different instances: user, model and operative system.



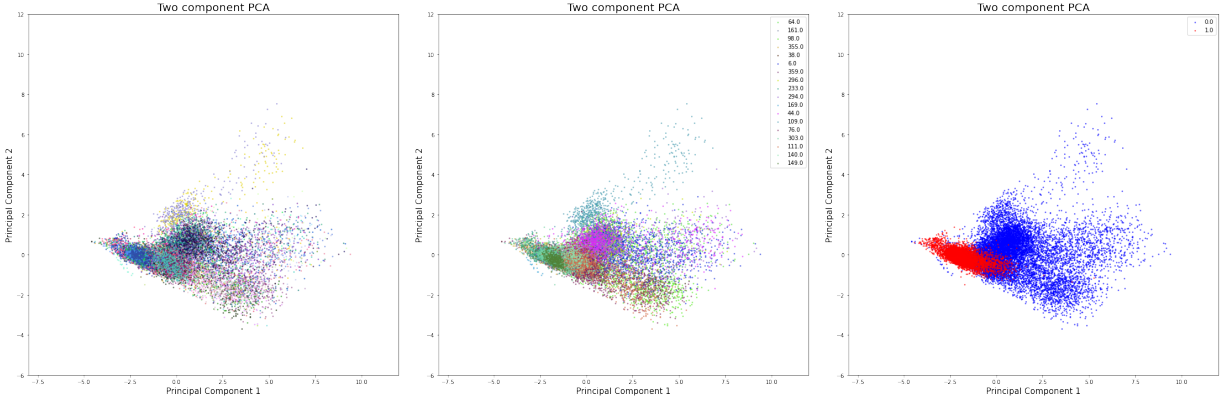
(a) PCA of the mean of the spectra computed with ridge regression, colorcoded based on user. (b) PCA of the mean of the spectra computed with ridge regression, colorcoded based on model. (c) PCA of the mean of the spectra computed with ridge regression, colorcoded based on operative system.

Figure 37: PCA:s of the mean of the spectra computed with ridge regression for different instances: user, model and operative system.



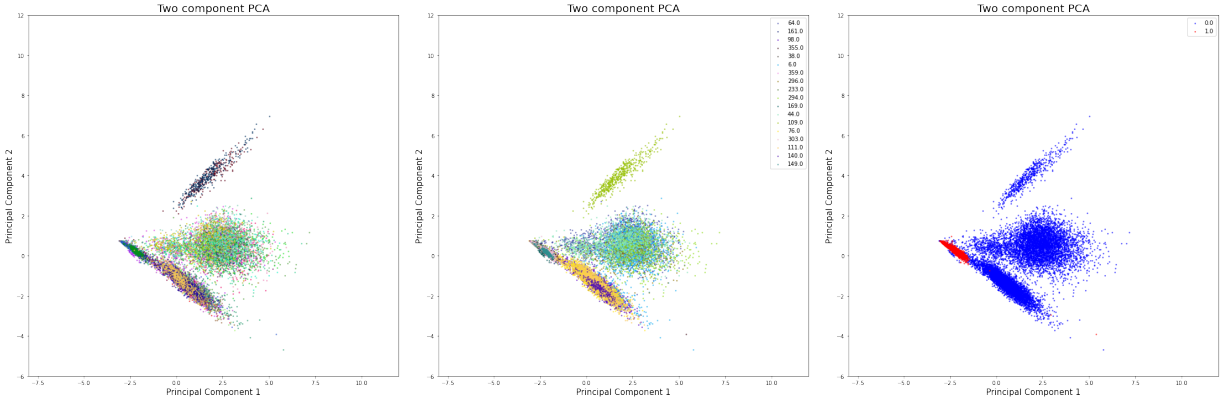
(a) PCA of the variance of the spectra computed with ridge regression, colorcoded based on user. (b) PCA of the variance of the spectra computed with ridge regression, colorcoded based on model. (c) PCA of the variance of the spectra computed with ridge regression, colorcoded based on operative system.

Figure 38: PCA:s of the variance of the spectra computed with ridge regression colorcoded for different instances: user, model an operative system.



(a) PCA of the mean of the spectra computed via re-sampling and FFT, colorcoded based on user. (b) PCA of the mean of the spectra computed via re-sampling and FFT, colorcoded based on model. (c) PCA of the mean of the spectra computed via re-sampling and FFT, colorcoded based on operative system.

Figure 39: PCA of the mean of the spectra computed via resampling and FFT, colorcoded for different instances: user, model and operative system.



(a) PCA of the variance of the spectra computed via re-sampling and FFT, colorcoded based on user. (b) PCA of the variance of the spectra computed via re-sampling and FFT, colorcoded based on model. (c) PCA of the variance of the spectra computed via re-sampling and FFT, colorcoded based on operative system.

Figure 40: PCA of the mean of the spectra computed via resampling and FFT, colorcoded for different instances: user, model and operative system.

Appendix II

Some more results regarding the tests with Modified Hausdorff distance.

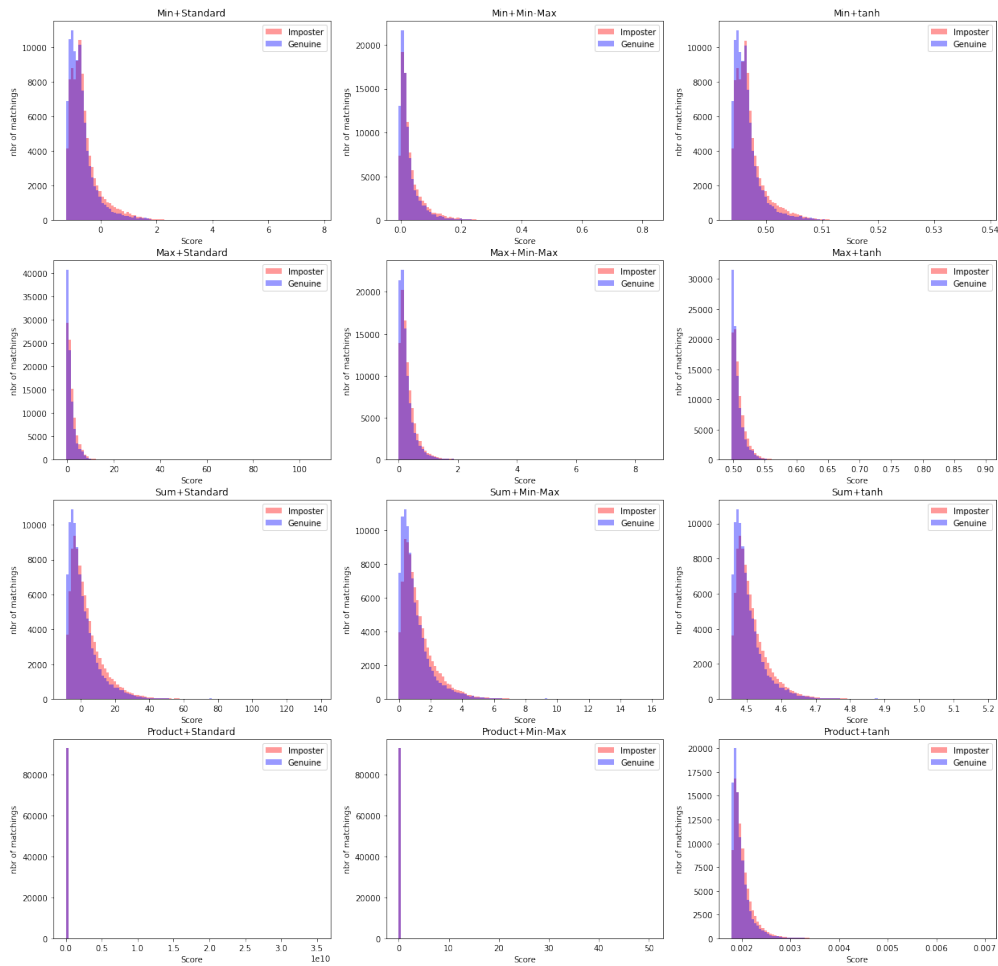


Figure 41: Histograms of the scores for different ways of normalizing and fusing when each signal is treated separately. The differences are small. A thin red edge can be detected in some of the figures.

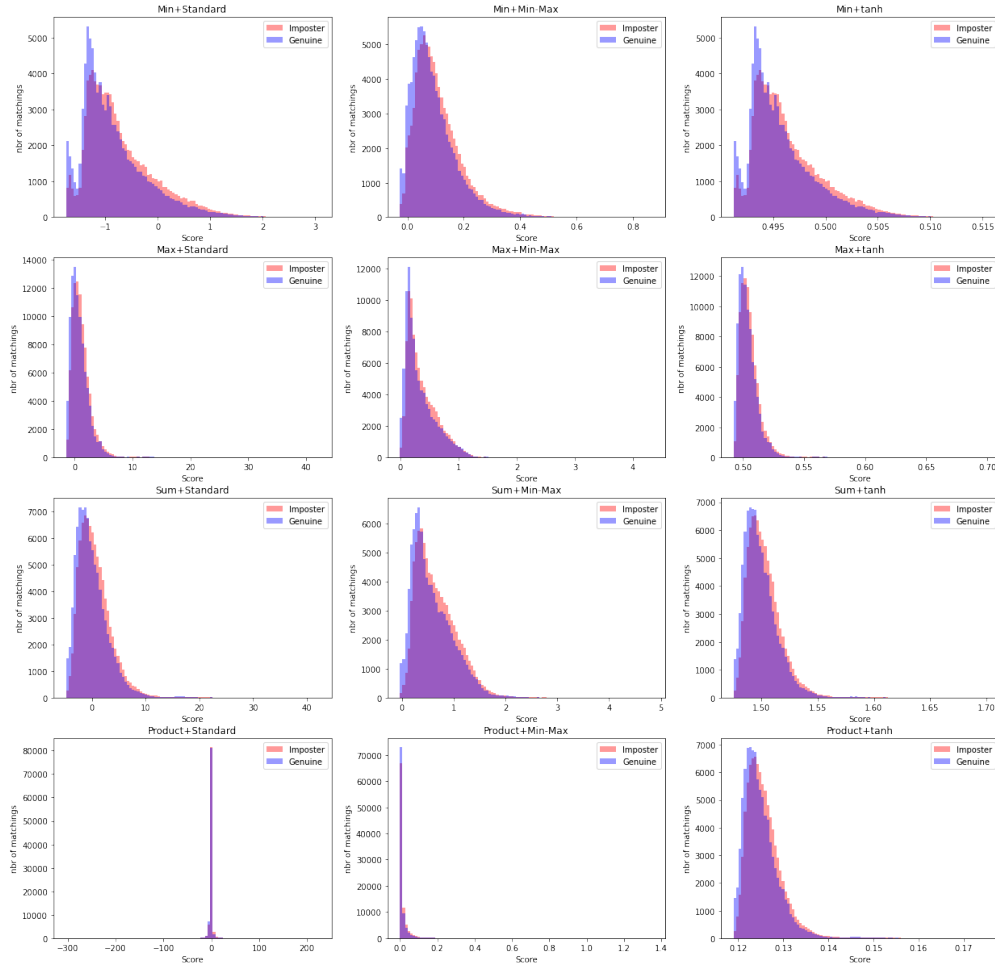


Figure 42: Histograms of the scores for different ways of normalizing and fusing for when the signals from each sensor is compared. The differences are small. A thin red edge can be detected in some of the figures.

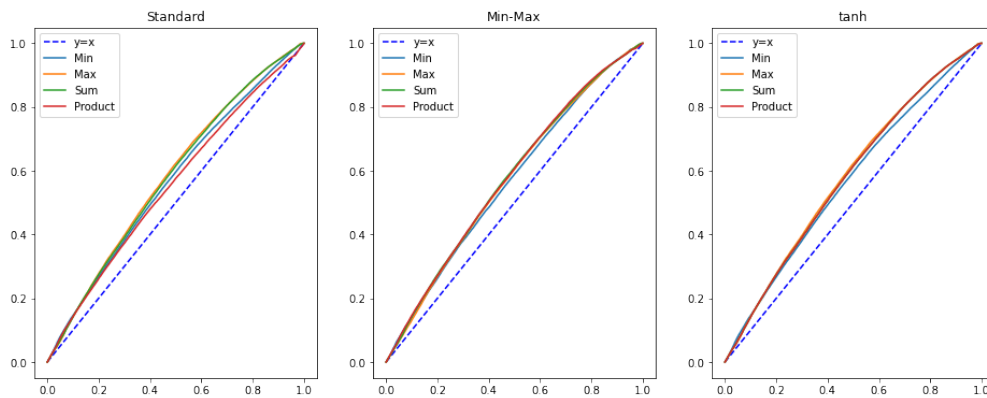


Figure 43: ROC-curves for the different ways of normalizing and fusing the scores when the signals are treated separately. All curves at least stays over the dreadful line $y = x$, even though the margin is not very large.

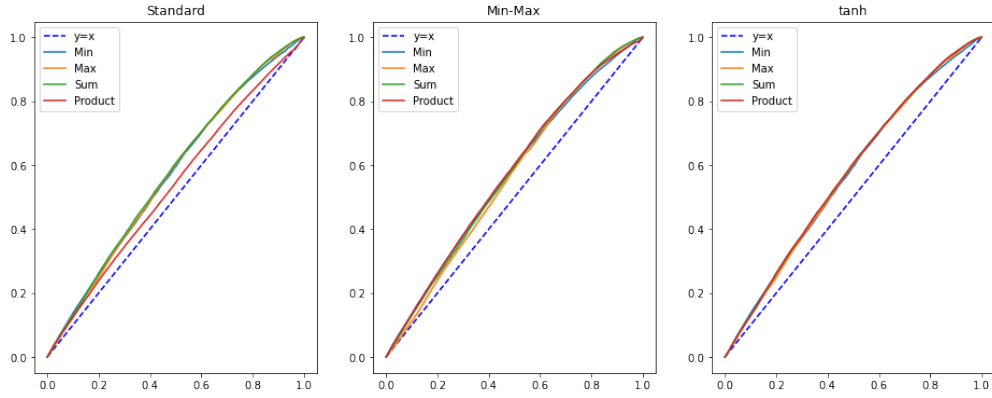
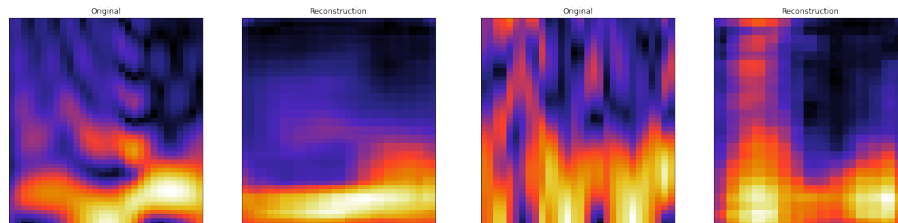


Figure 44: ROC-curves for the different ways of normalizing and fusing the scores when the signals from each are not separated. All curves at least stays over the dreadful line $y = x$, even though the margin is not very large.

Appendix III

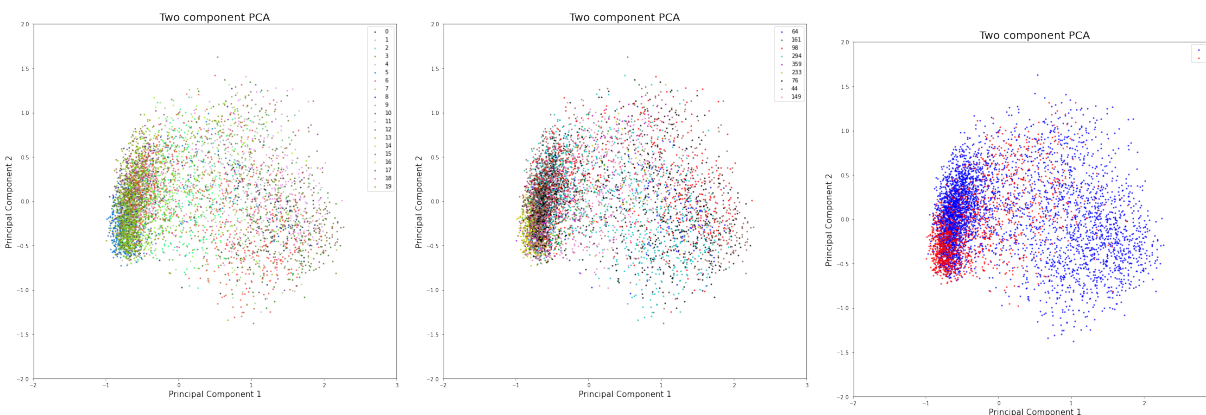
Results when computing the spectrograms in the validation set with the unmarked approach in Table 10 are shown in Figures 45-47.



(a) An original spectrogram and its reconstruction.

(b) Another original spectrogram and its reconstruction.

Figure 45: Examples of original spectrograms and their reconstruction. The reconstructions do not capture the fine structure of the original image but overall has the same appearance.

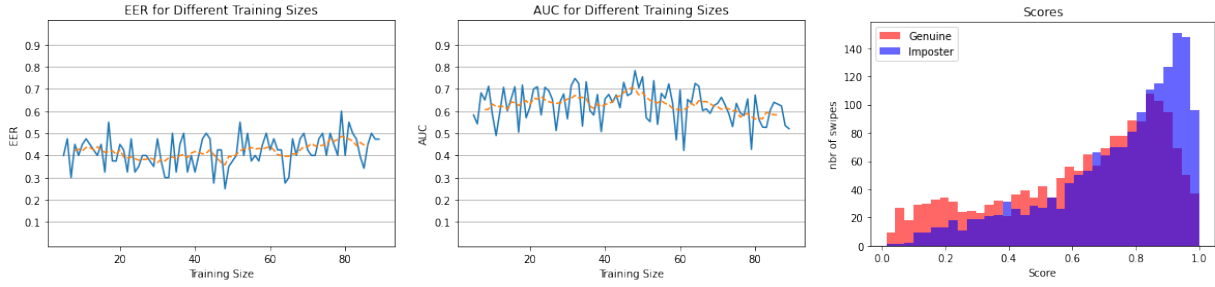


(a) PCA of the spectrograms in latent space, colorcoded after user.

(b) PCA of the spectrograms in latent space, colorcoded after phone-model.

(c) PCA of the spectrograms in latent space, colorcoded after operative system.

Figure 46: PCA of the spectrograms in the latent space for three different instances: user, phone-model and operative system. The samples are spread in a circle that is a bit heavier to the left.



(a) EER and its trend for different training sizes, using KDE. (b) AUC and its trend for different training sizes, using KDE. (c) Distributions of scores from all iterations, using KDE.

Figure 47: The EER, AUC and distributions of scores for the validation set.

Master's Theses in Mathematical Sciences 2021:E4
ISSN 1404-6342
LUTFMS-3421-2021
Mathematical Statistics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden
<http://www.maths.lth.se/>