Improving Intent Recognition in Mobile Payments using a Smartwatch

Trinity 2023

Candidate Number 1042754

A Project Report submitted for

Part C Mathematics and Computer Science

Abstract

This report extends upon the work of Sturgess et al. [1] by showing that a different method can achieve better results using the tap gesture, performed when a user taps a smartwatch on a NFC-enabled terminal to make a payment, as a biometric to recognise intent-to-pay. The method considered in this report splits the tap gesture into 3 component phases - reaching, alignment and withdrawal - uses models to recognise each phase and combines the results to form another prediction of intent-to-pay. This method was evaluated on the dataset acquired by Sturgess et al. [1] to validate this approach and compare the results. Using this data and method, systems were developed for intent recognition, and the optimal combination model architecture was identified which achieved an EER of 0.009 (an improvement over the EER of 0.04 achieved by Sturgess et al. [1]).

Acknowledgements

I would like to thank my supervisors for supervising this project, proofreading this report and providing the initial idea for this project.

I would also like to thank members of my family for identifying spelling and grammar issues in this report.

Word Count

The overall word count for this report is **9997 words**.

This figure was acquired by copying and pasting the relevant text of the latex document for this report into an online word counter (available at `https://wordcounter.net/`). The figure was then checked by repeating the process with a couple of different word counters (`https://thewordcounter.com/` and `https://wordcounter.io/`) getting the results 9997 and 9961, respectively.

Contents

Chapter 1

Introduction

Recent years have seen a dramatic increase in the use of cashless payments, particularly during the Covid-19 pandemic. This led to increased usage of mobile payment systems such as Google Pay that allow users to make a contactless payment through an NFC connection with a smartphone. In the UK in 2021, there were 11.2 million users of mobile payment systems with this expected to rise to 13.9 million by 2025 [2].

Smartwatches can be connected to smartphones and linked to mobile payment systems such that a smartwatch can be used in place of the smartphone during the NFC connection for a mobile payment. Consequently, due to convenience some mobile payment users have started to take advantage of this and many smartwatches and mobile payment systems support this functionality.

Unfortunately, it is common for NFC mobile payments, particularly those utilising smartwatches, to have little to no authentication or recognition of intent-to-pay. Consequently, NFC hacks are possible with malicious agents making an unintended payment or the user themselves accidentally making a payment [3]. Additionally, the problem of card clash - where 2 or more payment cards or devices are in the vicinity of a payment terminal resulting in a clash - can occur for NFC mobile payments, although the short-range nature of the technology makes this unlikely.

As a result, intent recognition systems are important to prevent unintended payments and, while smartphone payments can require a fingerprint scan, pin entry or other authentication action, it is common for smartwatches to be left unlocked for ease of use. To combat this, it is vital to have an automated system able to recognise intent-to-pay

accurately for mobile payments using smartwatches that does not require additional user action.

For this reason, authentication and intent recognition systems have been proposed for smartwatch payments using the payment tap gesture as a biometric [1]. This project aims to extend upon this work by proposing and evaluating a phase-based approach for intent recognition in mobile payments using smartwatches. This approach splits the tap gesture into 3 phases - reaching, alignment and withdrawal - which are recognised by separate models and then combines the results of these models to make an overall prediction of intent-to-pay.

1.1 Related Work

This section details some literature related to this project report.

Mobile Device Authentication

Authentication is a key issue for mobile devices such as smartphones and smartwatches. There are a wide range of methods that can be used for authentication including passwords, pins, fingerprints and facial recognition. In recent years, there has been a growing interest in using gestures as biometrics to authenticate mobile devices, particularly as part of a Multi-Factor Authentication system. Details on the gestures and devices used and results of a range of papers on the authentication of mobile devices using motion sensor data can be found in Table 1.1.

These papers demonstrate that using motion sensor data to authenticate a mobile device consistently performs well, including in tasks very similar to that considered in this project [1][8].

Gesture Recognition using a Smartwatch

Recognition of small gestures such as finger and hand gestures is an important task for gesture interface systems, some of which do not have access to visual data, and smartwatches are uniquely positioned to provide data for these fine gestures. As a result, there are numerous papers using data from smartwatch motion sensors (such as the accelerom-

Year	Author(s)	Gesture/Activity	Smart Device		EER
			Watch	Phone	
2011	Conti et al. [4]	Making or answering a phone call	✗	✓	0.069*
2015	Johnston and Weiss [5]	Walking	✓	✗	0.014
2015	Hong et al. [6]	Wave	✗	✓	0.055*
2016	Shrestha et al. [7]	Payment Tap	✗	✓	0.01*
2016	Lee and Lee [8]	Continuous	✓	✓	0.079*
2016	Lewis et al. [9]	User-chosen free-form gesture	✓	✗	0.149*
2016	Davidson et al. [10]	Range of natural gestures	✓	✗	0.065*
2017	Lee et al. [11]	Picking up a Phone	✗	✓	0.038*
2018	Al-Naffakh et al. [12]	Continuous	✓	✗	0.013
2021	Lee et al. [13]	Response to vibration	✓	✗	0.014
2022	Sturgess et al. [1]	Payment Tap	✓	✗	0.08

Table 1.1: Details of the gestures and activities used and the results of several papers related to the authentication of mobile devices using motion sensor data.
Some of the papers do not report an EER, in these cases the average of the FPR and FNR is given and marked with a *

eter and gyroscope) to recognise gestures. Details on the gesture or gesture class analysed and the results of a range of papers on gesture recognition using wrist-located motion sensors can be found in Table 1.2.

Year	Authors	Gesture	Results	
			Accuracy	F-measure
2014	Parate et al. [14]	Smoking	0.957	
2015	Xu et al. [15]	Range of Finger Gestures	0.98	
		Finger-writing	0.95	
2016	Wen et al. [16]	Range of finger gestures		0.87
2016	Sen et al. [17]	Playing Foosball	0.95	
2016	Ranjan and Whitehouse [18]	Range of person-object interactions	0.83-0.91	
2016	Davidson et al. [10]	Range of natural gestures	0.998	
2018	Zhu et al. [19]	Range of hand gestures	0.96	
2020	Tahir et al. [20]	Range of person-object interactions		0.879
2022	Sturgess et al. [1]	Payment Tap	0.96	0.88

Table 1.2: Details of the gestures and results from papers related to gesture recognition using motion sensor data from sensors in a watch or attached to a wristband.

These papers show that gesture recognition using smartwatches has been thoroughly investigated and shows good results in recognising a range of gestures including very small gestures such as finger movements[15][16].

Splitting Gestures into Phases

Segmenting a gesture into phases was first suggested in 1980 by Kendon et al. [21]. This proposed a gesture structure composed of 3 phases: Preparation (preparing to perform

the gesture, for instance, moving into the required position), Stroke (performing the main component of the gesture) and Recovery (returning to a rest state after the gesture).

Since this segmentation was proposed, this method has been used multiple times in gesture recognition, particularly when recognising gestures from visual data such as video.

In 1994, Davis and Shah [22] proposed a system to recognise hand gesture commands from visual data. This system splits the gesture into 4 phases: holding the start position, moving to the gesture position, holding the gesture and returning to the start position.

In 2003, Nickel and Stiefelhagen [23] proposed a system to recognise a 3D pointing gesture from visual data that separately recognised 3 stages of the gesture and combined the results.

In 2014, Yin and Davis [24] proposed a system for real-time continuous gesture recognition from visual data. This system split gestures into phases to provide more prompt responses to command gestures.

Phase segmentation is rarely applied to motion sensor data. Therefore, this work provides much needed evidence of the potential of this approach, which could be utilised in other gesture recognition systems to improve performance.

Intent Recognition

After a thorough literature search, the only paper found considering an intent recognition system is the work by Sturgess et al. [1] that this project extends upon. This demonstrates the importance of further work in this area to prevent accidental payments or drive-by hacks in mobile payments (and potentially other cashless payments).

1.2 Contributions

- This project shows that a phase-segmented approach can significantly improve the performance of an intent recognition system for mobile payments using smartwatches.
- It shows that this approach can be applied effectively for either in-store real-time usage or retrospectively.
- The system is terminal and user agnostic so does not need a specific terminal type or position and does not need training data from the user during enrolment.

- The code and data needed to reproduce the results is made available at `https://github.com/cn1042754/project`.

1.3 Outline

Chapter 2 describes the assumptions, machine learning models and evaluation metrics used in this project.

Chapter 3 describes the dataset [25], and discusses the method previously used for intent recognition on this dataset by Sturgess et al. [1] and the key results this method achieved.

Chapter 4 provides an overview of the method used in this project.

Chapter 5 contains information on how the method described in Chapter 4 was implemented for this project.

Chapter 6 describes the experiments performed for this project and the results achieved.

Chapter 7 discusses the key results from this project and explores some potential avenues for future work.

Chapter 2

Preliminaries

This chapter describes the threat and system models, the machine learning models and the evaluation metrics used in this project.

2.1 System and Threat Models

This section describes the assumptions made about the system and threats against it.

2.1.1 System Model

It is assumed that the user has an unlocked NFC-enabled smartwatch, attached to their wrist, that they use to make mobile payments on a NFC-enabled point-of-sale terminal. When making a payment, the user performs a tap gesture which involves distinctive movement patterns that are not involved in other gestures. Consequently, it is assumed that, when a tap gesture is performed, any payment made was intentional.

The system makes use of inertial motion sensors, such as an accelerometer and gyroscope, that are assumed to be installed on the smartwatch (this is reasonable as smartwatches have these sensors as standard). No other data is utilised by the system, so it is assumed that no other information, such as the location or user details, is available. Additionally, the presence of a paired smartphone, while necessary for current implementations of mobile payments, is not required by this system to allow for the case where, in the future, smartwatches can make payments independently.

Finally, the payment tap gesture is assumed to be triphasic, with Reaching, Alignment

and Withdrawal phases (see Section 3.1.3), and identifying these phases is sufficient to recognise the tap gesture.

2.1.2 Threat Model

This project considers that the user has the smartwatch unlocked and on their wrist in a public setting and an unintentional payment is made. This could be due to an attacker moving a NFC-enabled device near the watch or this could be accidental.

It is assumed that this occurs during a common activity within one of three public settings: walking, on a bus/train or in a store. These present the most likely cases for a malicious or accidental payment to be made. Consequently, despite other settings being possible (but unlikely) in the real world, the dataset and this project only consider these cases.

Additionally, it is assumed that the inertial sensors on the smartwatch are accurate, reliable and cannot be tampered with. Also, this intent recognition system and its execution are secure and cannot be bypassed or affected by an external agent.

The goal of the system is to detect that no tap gesture occurred at the NFC contact time and, therefore, that the payment was unintentional.

2.2 Classification Models

This section briefly describes the classification models used in this project: Decision Trees, Random Forests, Bayesian Models, Logistic Regression and AdaBoosted Decision Stumps. Detailed descriptions can be found in Appendices A.1, A, B, C and D, respectively.

2.2.1 Decision Tree

A Decision Tree classifier utilises a Binary Decision Tree to process its inputs. The nodes of the tree each test a condition on the input and then point to one of its child nodes. The leaves of the tree return a predicted class.

Decision Trees are simple models that are very prone to overfitting. However, they are highly interpretable, can provide useful insights into the task and can be used as

components of other models.

2.2.2 Random Forest

Random Forests are collections of Decision Trees. When given an input, they independently run the underlying trees and return the majority from the results of the trees.

They are designed to reduce overfitting and can fit any decision boundary with enough trees. Additionally, through the use of Gini Importances (see Appendix A.3), they can provide useful insights into the usefulness of the features to the task.

2.2.3 Bayesian Model

Bayesian models are classifiers that utilise Baye's rule (see Appendix B). They fit a probability distribution to the training data and then predict the probability that this distribution generates a given input for each possible class, returning the class with the highest likelihood.

These models are simple but are somewhat prone to overfitting. They provide some insight into the distributions of the features resulting from each class.

2.2.4 Logistic Regression

Logistic Regressors train a simple linear decision boundary for the training data and assign a predicted class to an input based on which side of the boundary it lies on.

They are simple models that can experience overfitting. This model is included to evaluate whether even a simple decision boundary can recognise intent-to-pay effectively.

2.2.5 AdaBoosted Decision Stump

Decision Stumps are Decision Trees with a maximum depth of 1. This results in a decision boundary that classifies inputs based on a simple inequality of a single feature.

AdaBoost is a boosting algorithm that can boost the performance of a weak learner (such as a Decision Stump) by training a collection of models on different distributions of the training data.

The resulting boosted Decision Stump produces a linear decision boundary similar to Logistic Regression but using a different training method.

2.3 Evaluation Metrics

This section describes a range of statistics used to evaluate classification models.

In this section, the following shorthands are used:

- TP - the number of True Positives (correctly classified positive data)
- TN - the number of True Negatives (correctly classified negative data)
- FN - the number of False Negatives (incorrectly classified positive data)
- FP - the number of False Positives (incorrectly classified negative data)

2.3.1 Definitions

Error Rate

The Error Rate is the proportion of data that are incorrectly classified.

$$Error = \frac{FP + FN}{TP + TN + FP + FN}$$

Accuracy

The Accuracy is the proportion of data that are correctly classified.

$$Accuracy = 1 - Error = \frac{TP + TN}{TP + TN + FP + FN}$$

False Positive Rate (FPR)

The False Positive Rate is the proportion of negative data that are incorrectly classified.

$$FPR = \frac{FP}{TN + FP}$$

False Negative Rate (FNR)

The False Negative Rate is the proportion of positive data that are incorrectly classified.

$$FNR = \frac{FN}{TP + FN}$$

Equal Error Rate (EER)

The Equal Error Rate is the value of FPR and FNR when the model's decision threshold[1] is chosen so that these 2 statistics are equal. This value exists since varying the decision threshold leads to a trade-off in the FPR and FNR which will intersect at some point.

Precision

The Precision is the proportion of the data that are predicted to be positive that are actually positive.

$$Precision = \frac{TP}{TP + FP}$$

Recall

The Recall (AKA True Positive Rate or Sensitivity) is the proportion of positive data that are correctly classified.

$$Recall = \frac{TP}{TP + FN} = 1 - FNR$$

Specificity

The Specificity (AKA True Negative Rate) is the proportion of negative data that are correctly classified.

$$Specificity = \frac{TN}{TN + FP} = 1 - FPR$$

F-Measure

The F-measure (AKA F_1 Score or Dice Coefficient) is the harmonic mean of the Precision and Recall.

$$F_1 = \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2TP}{2TP + FP + FN}$$

[1] Binary Classification Models can produce a certainty in their class production. The decision threshold is the minimum certainty required for the model to predict the input as positive.

2.3.2 Limitations

Unfortunately, there are issues with using certain metrics, particularly in a security context, as explored by Eberz et al. [26].

The Error Rate and Accuracy make no distinction between False Positives and False Negatives. Consequently, unless the dataset contains equal proportions of positive and negative data, metrics are skewed towards the error on the more populous class.

Additionally, the different errors have different consequences which need to be considered. In this context, False Positives indicate that an intended payment is accepted potentially allowing theft. False negatives indicate that an intended payment was rejected resulting in a significant inconvenience to the user. Therefore, the system needs to be optimised according to the needs of the system and the potential results of the different error types.

This leads to Error Rate and Accuracy rarely being utilised in a security context since there are other metrics such as EER that provide insight into the different error types.

Chapter 3

Previous Work

This chapter describes the dataset, presents an overview of the method previously used for intent recognition by Sturgess et al. [1] and discusses the key results achieved.

3.1 Dataset

This section describes the dataset [25] used for this project, detailing the acquisition and discussing an example.[1]

3.1.1 Data Acquisition

This subsection is paraphrased from [1].

The data was produced by several sensors of a smartwatch during a payment tap gesture as well as data collected during other activities. Data was collected using 4 inertial sensors that are derived from the onboard accelerometer and gyroscope. The accelerometer measures the change in velocity and the gyroscope measures angular velocity. The linear accelerometer is derived from the accelerometer by excluding gravitational effects. The gyroscope rotation vector (GRV) combines information from the accelerometer and gyroscope to calculate the orientation of the device.

The payment tap gesture data was collected from 16 participants in a lab experiment; 9 of whom also provided data from other activities.

The payment tap gesture data was collected from the users triggering a NFC reader

[1]The dataset was acquired by Sturgess et al. [1].

attached to one of 7 terminals. There were 6 fixed terminals and a freestyle terminal that could be picked up. Details are in Figure 3.1 and Table 3.1.

Figure 3.1: Image of the six fixed terminals (labelled, see Table 3.1).
This figure is taken from [1].

Terminal	Height(cm)	Tilt(°)	Distance(cm)
1	100	0	5
2	120	60	25
3	95	45	-10
4	105	30	15
5	110	15	10
6	115	90	30
F	picked up from centre of platform		

Table 3.1: Terminal details labelled as in Figure 3.1; F is the freestyle terminal.
This table is taken from [1].

The sensor data was collected along dimensions x,y,z as shown in Figure 3.2.

3.1.2 Pre-processing

Before any use the dataset underwent pre-processing. This involved segmenting the payment tap gesture data into windows starting 4 seconds before and ending 2 seconds after the NFC contact time. The data from other activities was split into 4-second windows with a 50% overlap.

Finally, data was processed by a low pass butter filter to remove some of the noise.[2]

[2]This has not been applied to the public dataset.

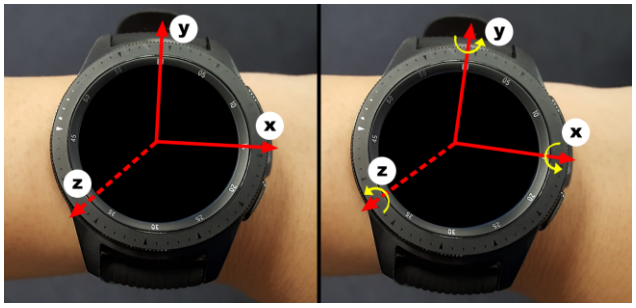

Figure 3.2: Diagram of the sensor axes.
This figure is taken from [1].

3.1.3 Example

An example of the data from a payment tap gesture is shown in Figure 3.3. The data is split into 3 sections based on the 3 phases of the gesture (shown by the vertical lines in the graphs). These gesture phases are reaching, alignment and withdrawal.

The reaching phase is when the smartwatch is moved from a rest position to the payment terminal. In the example, this is shown by the spike in the acceleration and gyroscope data during this phase.

The alignment phase is when the user tentatively moves the watch to achieve NFC contact or waits for the NFC contact to be registered. In the example, during this phase, the movement of the watch is very small as the user waits.

The withdrawal phase is when the watch is moved back into a rest position. In the example, this is shown by a spike in the accelerometer and gyroscope data.

3.2 Method Overview

The method previously used [1] selected a fixed window from each gesture, extracted features from this window, and trained and evaluated a model on these features.

Each model was trained on 90% of the dataset and evaluated on the other 10%. Ten copies of each model were trained and evaluated with the evaluation metrics averaged to produce more representative results.

3.2.1 Windows

Parts of this subsection are paraphrased from [1].

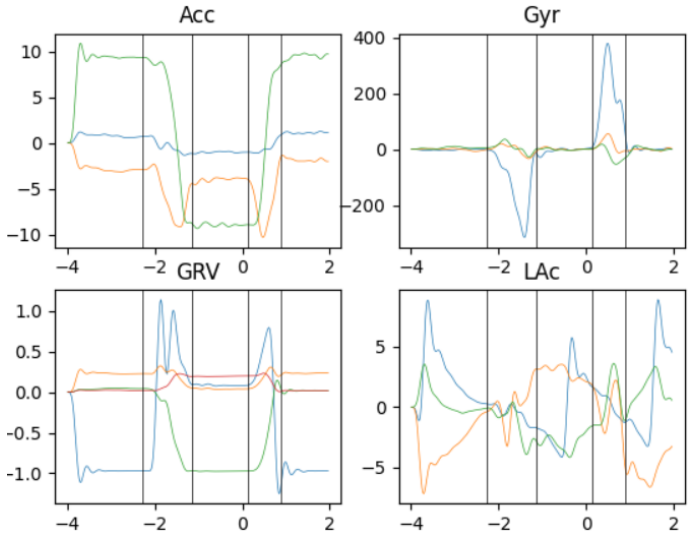

Figure 3.3: Data from a payment tap gesture; the vertical black lines correspond to the phase transitions.

The windows evaluated were fixed in time relative to the NFC contact time for the gesture (or relative to the 0 time for the data collected from other activities - see Section 3.1). A selection of windows were evaluated and were parameterised by the offset of the end of the window and its length. For an NFC contact timestamp T_0, window size s and offset o, the window retrieved ends at timestamp $T_E = T_0 - o$ and starts at timestamp $T_S = T_E - s$.

The offsets considered were $-2, -1.5, -1, ..., 1.5, 2$ seconds, and the window sizes were $0.5, 1, 1.5, ..., 3.5, 4$ seconds.

3.2.2 Features

This subsection is paraphrased from [1].

Whenever a window is retrieved, a low pass butter filter was applied to reduce noise and then the following five dimensions for each accelerometer, gyroscope, or linear accelerometer sample were processed: the filtered x-, y-, and z-values, and the energy of the filtered and unfiltered values, where the energy of $\{x, y, z\}$ is $\sqrt{x^2 + y^2 + z^2}$. GRV samples are expressed as quaternions so only the four filtered values were processed (since the Euclidean norm of a quaternion is always 1). In total, each window was processed in 19 dimensions.

Ten statistical features were extracted in each dimension: minimum, maximum, mean,

median, standard deviation, variance, interquartile range, kurtosis, skewness, and peak count. Additionally, the mean and maximum velocities along each axis, and the displacement along each axis and overall were calculated from each of the accelerometer, gyroscope, and linear accelerometer vectors. Ultimately, each window was reduced to 220 features.

3.2.3 Model

The models used were Random Forest classifiers (see Appendix A) with 100 Decision Trees each. These were trained on the features from the training windows and evaluated on the features from the test windows.

3.3 Key Results

This section summarises the key results of Sturgess et al. [1], which are relevant to this project.[3]

3.3.1 Optimal Window Parameters

This subsection is paraphrased from [1].

Figure 3.4 shows the average F-measure for each of the valid windows (where the window lies within the $[-4, 2]$ second range of the gesture).

In-store Usage

For in-store usage, only the windows that end by the NFC contact time (where the offset $o \geq 0$) can be considered.

The results were determined mainly by the offset with the later ending windows scoring much higher. This is as expected since it is common for the significant parts of the gesture (such as the alignment phase) to occur shortly before the NFC contact time (as shown in Section 3.3) so the earlier windows might not contain relevant data.

Overall, the optimal offset is $o = 0$, and when the window size $s = 2.5$ (favouring the authentication models also trained by Sturgess et al. [1]) the average F-measure score is 0.86 with precision 0.93, recall 0.82 and EER 0.04.

[3]This section makes repeated reference to evaluation metrics described in Section 2.3.

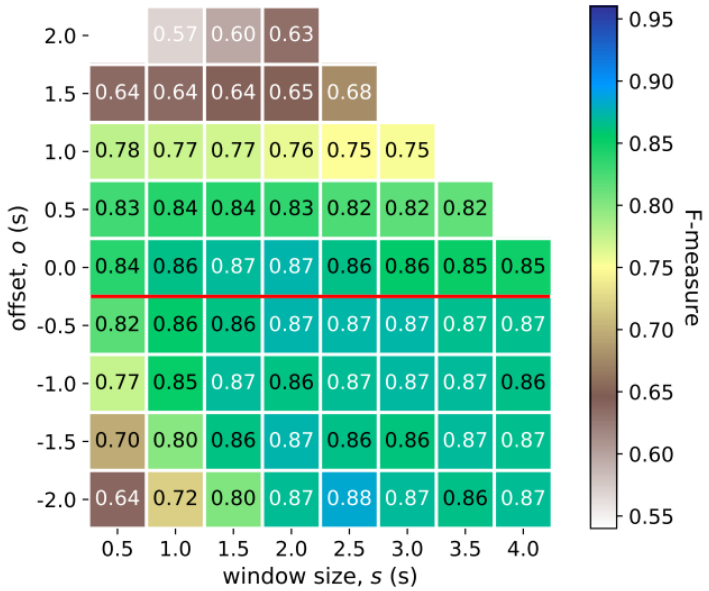

Figure 3.4: Average F-measure (AKA F1 or Dice) scores for each window by offset and size (see Section 3.2.1). The results for windows that end by the NFC contact timestamp, and are applicable to in-store usage, are above the red line.
This figure is taken from [1].

Retrospective Usage

For Retrospective historic usage, all valid windows can be considered. There is a slight improvement for windows including data shortly after the NFC contact time, particularly for windows overlapping this point (i.e. where $o \leq 0$ and $s > |o|$). This suggests that the alignment phase is the most distinctive part of the tap gesture and that the withdrawal phase is as useful as the reaching phase in recognising intent-to-pay. The strongest results were seen in windows spanning all 3 gesture phases, particularly for $2 \leq s \leq 3$ and $o \leq -0.5$, when the average F- measure was 0.87.

3.3.2 Limited Sensors

This subsection is paraphrased from [1].

As shown in Figure 3.5, the results for limiting to only the Accelerometer and Gyroscope sensors are very similar and slightly better than using all of the sensors (see Figure 3.4). This is likely because the other sensor values are derived from these (so are redundant) and are less useful in recognising intent-to-pay (possibly due to extra noise in these sensors).

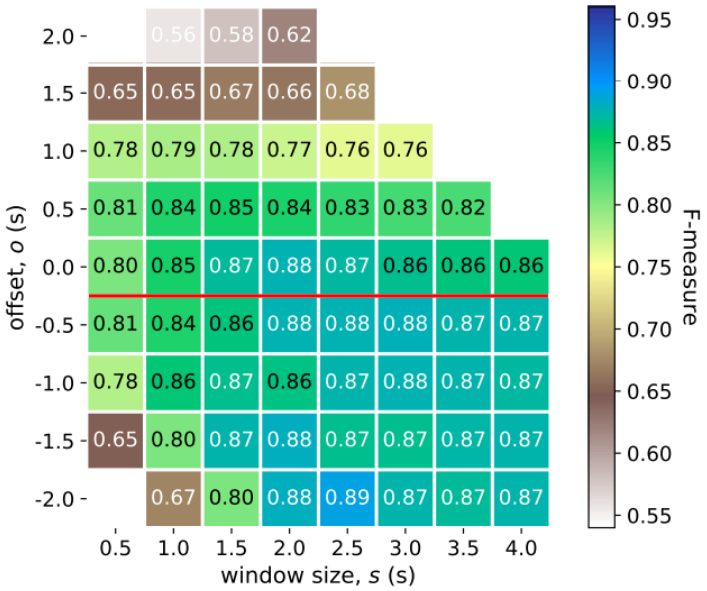

Figure 3.5: Average F-measure scores for each window by offset and size (see Section 3.2.1) for models only using the Accelerometer and Gyroscope sensors. The results for windows that end by the NFC contact timestamp, and are applicable to in-store usage, are above the red line.
This figure is taken from [1].

3.3.3 Feature Importance

This subsection is paraphrased from [1].

As shown in Table 3.2a, the number of peaks in the magnitude of the Linear Accelerometer data is very important. This is likely because the alignment phase can be characterised by a large number of small rapid movements as the user moves the watch around to locate the NFC contact point. Consequently, due to the distinctness of the alignment phase, features that can indicate the presence of this behaviour are more important.

Additionally, it can be seen that no GRV features are present in the table. This suggests that the GRV data is not very useful in recognising intent-to-pay or the alignment phase of a tap gesture.

Furthermore, features from the x-axis of the Gyroscope (rotation around the wrist) and the z-axis of the accelerometer (acceleration perpendicular to the watch face) appear frequently, suggesting that, during the alignment phase or tap gesture, the movement in these directions is peculiar.

Table 3.2b shows the feature importances when limited to accelerometer and gyroscope features. There are some minor changes, the most significant being the increased impor-

$o = 0, s = 2.5$		**All Windows**	
Feature	**Count**	**Feature**	**Count**
Acc-z-iqr	79	LAc-unf-pkcount	3397
Acc-z-kurt	65	Gyr-x-mean	1985
Gyr-x-mean	57	Acc-z-var	1789
LAc-unf-pkcount	55	Acc-z-stdev	1706
Acc-disptotal	44	Acc-z-iqr	1696
Acc-unf-iqr	43	Acc-z-med	1316
Gyr-unf-min	43	Acc-unf-iqr	1118
Acc-z-var	33	Gyr-x-velomean	1011
Acc-z-stdev	20	Gyr-unf-min	971
Acc-x-pkcount	11	Acc-disptotal	907

(a) All Sensors

$o = 0, s = 2.5$		**All Windows**	
Feature	**Count**	**Feature**	**Count**
Acc-z-kurt	73	Acc-z-var	2109
Gyr-unf-min	73	Acc-unf-iqr	2049
Acc-z-iqr	70	Gyr-x-mean	1841
Acc-unf-iqr	65	Acc-z-stdev	1716
Gyr-x-mean	52	Acc-z-iqr	1485
Acc-z-var	47	Acc-unf-pkcount	1412
Acc-disptotal	35	Acc-z-med	1368
Acc-z-stdev	30	Gyr-unf-min	1239
Acc-x-pkcount	3	Gyr-x-disp	1117
Gyr-x-disp	2	Gyr-x-velomean	1043

(b) Limited to accelerometer and gyroscope

Table 3.2: Modal top-five features by Gini importance summed over classifiers in optimum window s = 2.5, o = 0 and across all windows. Features are in the format sensor- axis-statistic; unf is the magnitude of the unfiltered {x, y, z} values and disptotal is the Euclidean displacement.
These tables are from [1].

tance of the peak count of the acceleration magnitude due to the absence of the Linear Accelerometer.

Chapter 4

Method

The proposed method is to train separate phase classifiers for each phase of a tap gesture and a combination model on the prediction probabilities produced by these phase models.

The flow of training data is shown in Figure 4.1.

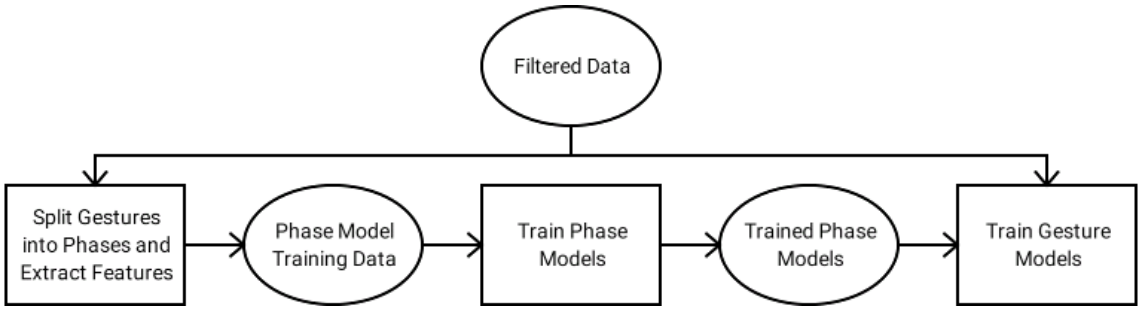

Figure 4.1: Diagram of the training data flow.

4.1 Phase Segmentation

To produce the phase model training data, the tap gestures were segmented into phases. This was achieved by manually segmenting each gesture using graphs of the gesture data. An example of this can be seen in Section 3.1.3.

4.2 Phase Model Training Data

Using the phase transition times, windows from each gesture were selected. Additionally, windows were taken at regular intervals from the non-gesture data (the data collected

during other activities) with a 50% overlap. From each of these windows, features were extracted to produce the phase model training data.

For each phase and gesture, positive windows were selected by taking windows of a specified length from the gesture overlapping the phase. Three such positive windows were chosen for each phase and gesture by taking windows with the same start, middle or end timestamp as the phase.

Additionally, negative windows were identified by taking windows with a 50% overlap from each end of the gesture only including windows not overlapping the phase.

The features extracted are the same as used by Sturgess et al. [1] and are described in Section 3.2.2.

4.3 Phase Model Architecture

The phase models are trained on the data described in Section 4.2 only using data from the gestures specified in the training mask for the model. The training mask contains the gesture ids for a stratified random subset of the data and contains around 60% of the possible ids.

When being run after training, the phase models iterate across windows of a specified length with a 50% overlap, extract features (see Section 3.2.2) from each window, and run the trained model on these features. The resulting prediction probabilities (the certainty with which the model identifies a valid phase) from each window are aggregated by taking the maximum value. The result is close to 1 for a gesture containing a valid phase. Otherwise, the result is close to 0.

4.4 Gesture Model Architecture

The combination models are trained on the results of the phase models on their corresponding validation dataset. The overall training dataset for the combination model is created by joining the results of each set of phase models using the same test data split as the combination model in a cross fold format. The validation sets each contain a stratified subset containing about 20% of the total dataset. Consequently, each combination model is trained over 80% of the dataset, where each phase model is only run on gestures it was

not trained on.

After training, the gesture model will run its phase models on an input gesture, average the results over each phase and window size, and run the combination model on these averaged results to predict whether the data forms a valid payment tap gesture.

Chapter 5

Implementation

This Chapter discusses the implementation of the method described in Chapter 4.[1]

5.1 Programming Language and Libraries

This project was implemented in Python and makes use of the scikit-learn [27], NumPy [28] and SciPy [29] libraries. The scikit-learn library implements a range of machine learning models. The NumPy library implements array operations and the SciPy library implements the multivariate Gaussian distribution's density function.

5.2 Data Storage

The gesture data is stored in a csv (Comma Separated Value) format with gesture and non-gesture files for each user. The files have the following columns:

- User ID - an identification string for the user that produced the datum (of the form "User<ID>", for instance "User001")
- Gesture ID - an identification string for the gesture that the datum belongs to (of the form "<activity>-<number>", for instance "TAP1-001")
- Sensor ID - an identification string for the sensor that produced the datum ("Acc", "Gyr", "LAc" or "GRV")

[1] The code from this project is available at `https://github.com/cn1042754/project`.

- Timestamp - the time, in seconds, that the datum was collected, relative to the NFC contact time for the gesture
- X - the value in the x-axis
- Y - the value in the y-axis
- Z - the value in the z-axis item W - the value in the w-axis (empty except for GRV data)
- Energy Unfiltered - the energy of the datum before being filtered (empty for GRV data)

The phase transition times are stored in csv format with the following columns:

- User ID - an identification string of the user that produced the gesture
- Gesture ID - an identification string of the gesture the transition times are for
- Time1 - the timestamp when the reaching phase started
- Time2 - the timestamp of the transition between the reaching and alignment phases
- Time3 - the timestamp of the transition between the alignment and withdrawal phases
- Time4 - the timestamp when the withdrawal phase ended

Lists of the gesture IDs for each user were stored in csv format with the user ID in the first column followed by a sequence of gesture IDs. Also, the window IDs for the phase model training data were stored in csv format with columns for the user ID and window ID.

Finally, the phase model training data is stored in csv format with separate files for each user and stored in rows with a window ID followed by a list of feature values.

All other objects or models stored are saved and loaded using the Python joblib library.

5.3 Modules

This project was implemented in a module format and contains the following Python modules/scripts:

- Phase Splitter
- Phase Time Graphs
- Gesture IDs
- Utils
- Feature Extractor
- Data Splitter
- Data Splitter Limited Sensors
- Mask Generator
- Aggregation Function
- Phase Model
- Combination Model
- Stats Function
- Gesture Model

The execution flow and import connections are shown in Figure 5.1. These modules are explained further in the following sections.

5.4 Support Modules

This Section describes the modules that perform operations that support the main execution flow. The support modules are:

- Gesture IDs[2]

[2]The Gesture IDs module is part of the main execution flow (see Figure 5.1) but is classed as a support module.

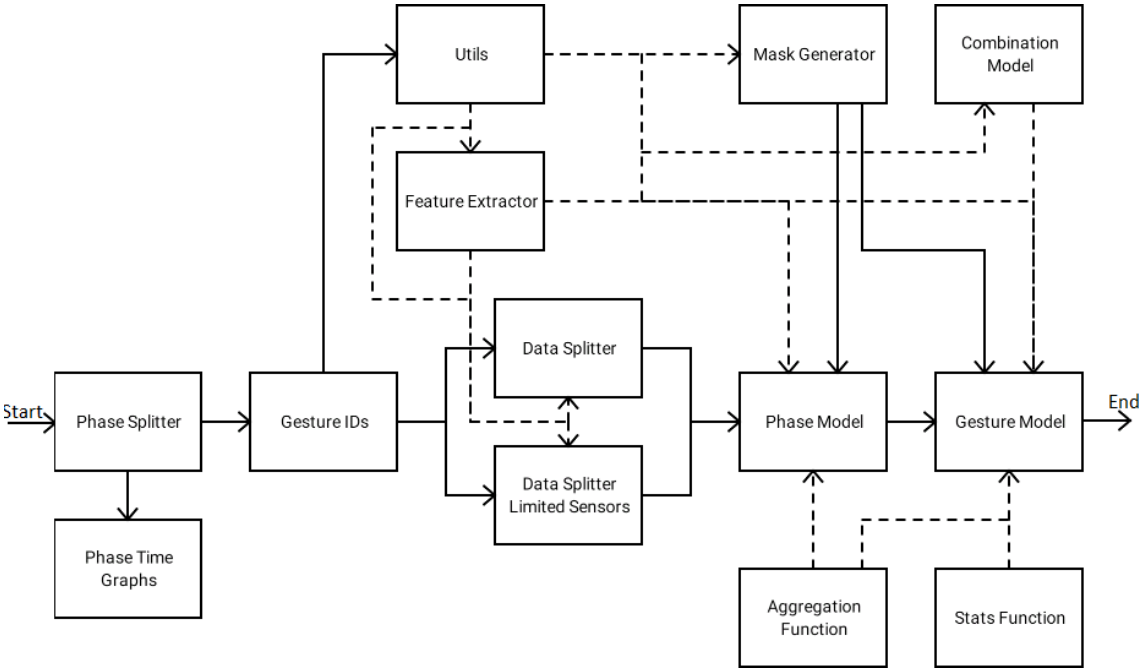

Figure 5.1: Connections between modules. The solid arrows indicate that a module must be run before another can be run or imported. The dashed arrows indicate that a module is imported by another.

- Utils
- Feature Extractor
- Mask Generator
- Aggregation Function
- Combination Model
- Stats Function

5.4.1 Gesture IDs

This module reads the phase transition times file to compile a list of the valid gesture IDs for each user and stores these as described in Section 5.2.

5.4.2 Utils

This module contains constants and methods used throughout the implementation. It contains lists of the valid user IDs, gesture IDs, sensor names, terminal IDs, phase names

and window sizes for the phase models. Additionally, it contains constants about the different column indices for the csv files (described in Section 5.2). Finally, it contains methods to retrieve data.

5.4.3 Feature Extractor

This model contains an object that extracts the features (described in Section 3.2.2) from the gesture data passed to it.

It is implemented by taking a list of feature functions and running them on the data given or loaded. The feature functions are each a function with a list of sensors and a name. The feature extractor compiles the names of the features into a single list.

This module also contains a feature extractor object that only returns features from the accelerometer and gyroscope data.

5.4.4 Mask Generator

This module contains a range of Mask Generator objects. These mask generators produce a random stratified split of the dataset into 3 sets: a training set (to train the phase models), a validation set (to train the combination models) and a test split (to evaluate the gesture model). It returns a list of the user and gesture ids for each subset of the data. These masks are produced according to a given test split index and validation split index that are specified to allow for the models to use a cross fold method for training and evaluation.

There are several variants of the mask generator instantiated within the module.

There is a general mask generator that produces an even split of the data into 5 subsets.

There are also user masks which fix the data from a particular user as the test data and produce an even split of the other data into 4 subsets used for the training and validation sets. In cases where a user has no non-gesture data, this part of the test set is assigned randomly from the non-gesture data of the other users.

Finally, there are terminal masks that fix the data from a particular terminal as the test data and assign the negative part of the test data from the non-gesture data.

5.4.5 Aggregation Function

This module defines a class of functions with names that run on lists. There are instances of this class to take the maximum, minimum, mean and sum of the list provided.

5.4.6 Combination Model

This module contains the combination models that combine the phase model results.

It defines a Combination class that has a name and underlying model and when run on given data returns the list of prediction probabilities for the positive class (valid payment tap) from the underlying model.

This module defines a Bayesian Classifier (see Appendix B) that assumes the independence of the prediction probabilities produced by the phase models for different phases, conditional on class.

There are Combination objects for this Bayesian Classifier, a Decision Tree, a Random Forest, Logistic Regression and an AdaBoosted Decision Stump (see Appendices A.1, A, C and D, respectively). This range of models is implemented to compare combination methods and to gain insights into recognising intent-to-pay.

5.4.7 Stats Function

This module defines a class of objects for calculating various statistics used to evaluate and optimise the gesture models. These objects each have a name and a run on a list of 3-tuples containing the predicted and target class labels and a count of the number of occurrences of this pair of labels.

This Stats Function objects calculating the FPR, FNR, Error Rate, Precision, Recall, F_1 score and Specificity (see Section 2.3).

5.5 Main Modules

This Section describes the modules in the main execution flow:

- Phase Splitter

- Phase Time Graphs[3]
- Data Splitter
- Data Splitter Limited Sensors
- Phase Model
- Gesture Model

5.5.1 Phase Splitter

This module implements a Graphical User Interface to manually segment the gestures into phases. A screenshot of the GUI is in Figure 5.2.

There are buttons allowing navigation to the next or previous user or gesture. The currently selected transition times are displayed below the navigation buttons. The left-hand side of the interface displays the gesture data using graphs for each sensor. The currently selected timestamps are displayed on the graphs as vertical black lines. The time the mouse is hovering over is shown as a vertical red line on the graphs. The time value currently being updated is shown by a border around the time label.

The interface is initialised with the currently stored times for the gesture when the gesture is loaded. The stored values are updated once all 4 times have been altered through the interface.

When no times have been stored previously, the module compiles a list of all user and gesture IDs, initialises all of the times to 0, and creates a csv file storing these values as described in Section 5.2.

5.5.2 Phase Time Graphs

This module generates histograms of the phase durations, ignoring phases that are less than half a second in length, as these are unlikely to be significant enough for a phase model to recognise them.

These histograms are created to allow the identification of appropriate window size(s) for the phase models.

[3]The Phase Time Graphs module is not required by later modules but is essential for the identification of ideal window sizes for the phase models.

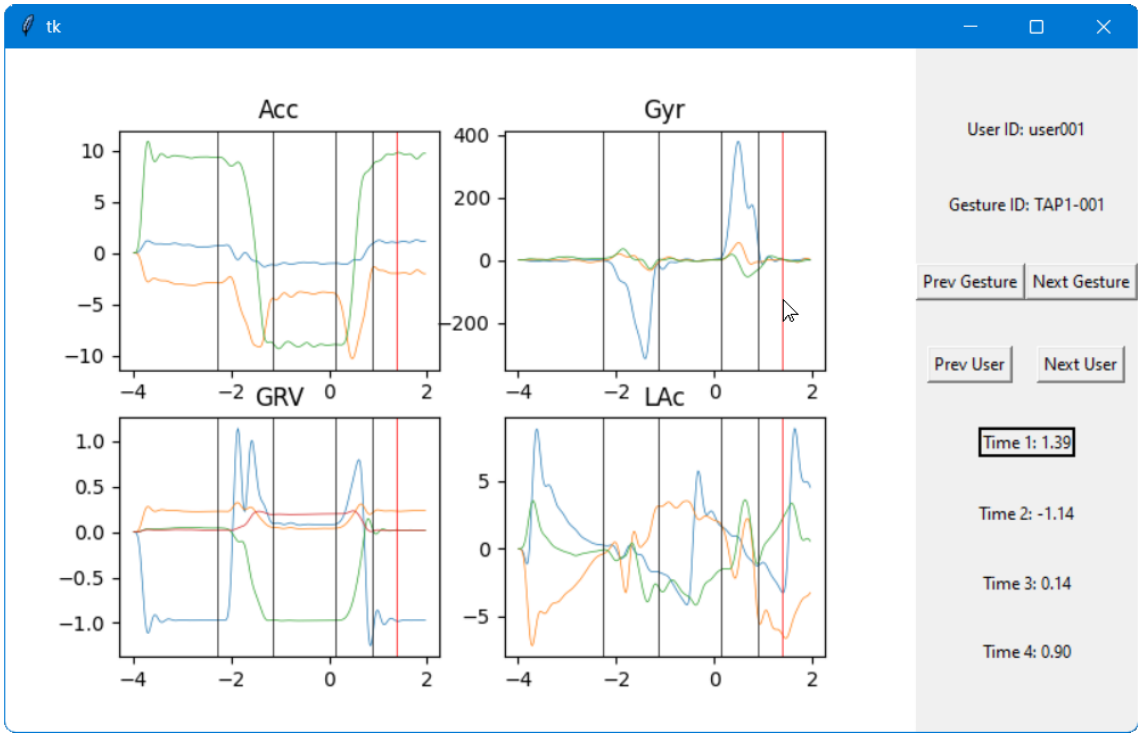

Figure 5.2: GUI for the Phase Splitter.

5.5.3 Data Splitter

This module generates the windows, as described in Section 4.2, from the filtered gesture data. For each window, it uses the feature extractor module (see Section 5.4.3) to extract features for the window, which are then stored as described in Section 5.2 along with lists of the window identifiers (in the format "<gestureID>-<counter>", for instance "TAP1-001-1").

5.5.4 Data Splitter Limited Sensors

This module is identical to the Data Splitter module (see Section 5.5.3) but uses the feature extractor that works on a limited subset of sensors (see Section 5.4.3).

5.5.5 Phase Model

This module creates the Phase Model objects (described in Section 4.3) using Random Forest classifiers with 100 Decision Trees. The underlying models are trained and stored when first instantiated and then loaded in the future.

It creates Phase Model objects for each mask generator and each test and validation

index for that generator (see Section 5.4.4).

5.5.6 Gesture Model

This module defines the Gesture Model class, as described in Section 4.4, using the combination models created in the Combination Model module (see Section 5.4.6).

On instantiation, the underlying phase models are run on their validation sets and the models for the same phase and test index are combined and run on the corresponding test data. The resulting prediction probabilities are then compiled and stored.

Once instantiated, the model can be assigned a combination model (the default is a Bayesian classifier) which is then instantiated, trained on the validation data, and stored. The trained combination models are then run on the corresponding test data with the predicted classes stored.

Finally, a gesture model can be evaluated by any Stats Function object (see Section 5.4.7) and can be optimised to acquire the EER (see Section 2.3) - for implementation purposes, the average of the FPR and FNR is taken when these statistics are as close as possible for a set of candidate decision thresholds.

All data stored by these objects is reloaded when the gesture model is reinstantiated.

This module also runs the gesture model experiments detailed in Chapter 6.

Chapter 6

Experiments

This chapter describes the experiments performed to evaluate the method described in Section 4 and discusses the results.[1]

6.1 Phase Durations

Histograms of the durations of each phase, as produced by the Phase Time Graphs module (Section 5.5.2), can be found in Figure 6.1.

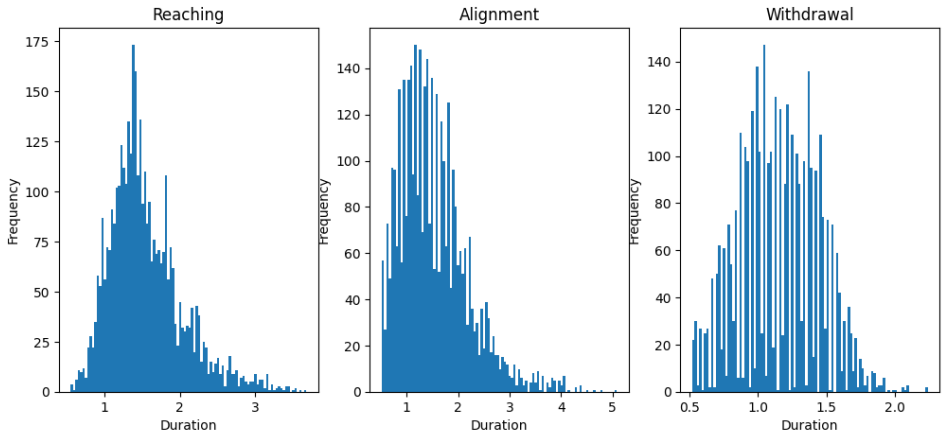

Figure 6.1: Histograms of the Phase durations.

This shows a large variation in the durations of the phases so, to compensate for this, multiple window sizes were selected for the phase models. Specifically, window sizes of 1 and 2 seconds were chosen.

[1]This section makes repeated reference to evaluation metrics described in Section 2.3.

This was to ensure that the phase models can reliably recognise the distinctive patterns in phases close to their window size due to the variations in the patterns that occur for different durations.

6.2 Overall Effectiveness

The overall results for each combination model are shown in Table 6.1 and the EER graph for the optimal combination can be seen in Figure 6.2.

Combination Model	EER	F_1	Precision	Recall	Specificity
Bayesian Model	0.018	0.922	0.869	0.982	0.982
Decision Tree	0.020	0.966	0.969	0.963	0.996
Random Forest	0.009	0.966	0.944	0.989	0.993
Logistic Regression	0.012	0.949	0.914	0.987	0.989
Boosted Decision Stump	0.013	0.941	0.899	0.986	0.987

Table 6.1: Evaluation statistics for each combination model.

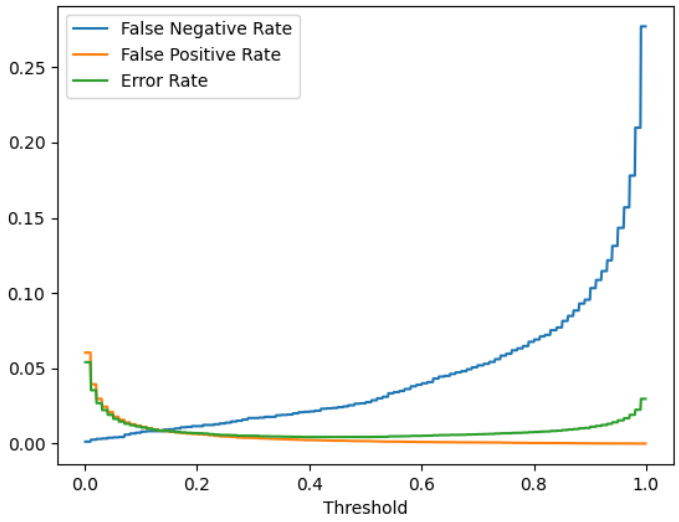

Figure 6.2: The EER Graph for the Random Forest Combination.

These results show that the Random Forest combination is optimal in all of the evaluation metrics. This would imply that the other combinations are overfitting[2] since Random Forests are designed to minimise this issue (see Appendix A).

Additionally, the Logistic Regression and Boosted Decision Stump models attain similar results as they both train a linear decision boundary (this trend is also shown in the rest

[2]Where the model learns the noise on the training data instead of just the target relationship.

of the results in this chapter but will not be discussed again).

6.3 Phase Importances

This section considers and evaluates the importance and informativeness of each phase of the tap gesture using a range of methods.

6.3.1 Removing Each Phase

This subsection evaluates the impact that removing each phase has on some of the evaluation statistics (EER and F-measure) of each combination model comparing the results to those in Table 6.1. The results are in Table 6.2 and, in all cases, the statistics are worse than in Section 6.2.

Combination Model	**Reaching Removed**		**Alignment Removed**		**Withdrawal Removed**	
	EER	**F_1**	**EER**	**F_1**	**EER**	**F_1**
Bayesian Model	0.024	0.896	0.029	0.877	0.022	0.905
Decision Tree	0.041	0.926	0.041	0.926	0.036	0.939
Random Forest	0.017	0.933	0.021	0.911	0.017	0.924
Logistic Regression	0.019	0.919	0.022	0.907	0.019	0.924
Boosted Decision Stump	0.020	0.913	0.023	0.906	0.018	0.915
Average Change	+68%	-3.3%	+92%	-4.6%	+58%	-2.9%

Table 6.2: EER and F-measure when each phase is removed relative to Table 6.1.

6.3.2 Gini Importance

Table 6.3 shows (values proportional to the) Gini importances (see Appendix A.3) for the Random Forest Combination. These results are proportional to the number of training data each phase is used to split.

Phase	**Gini Importance**
Reaching	0.277
Alignment	0.142
Withdrawal	0.081

Table 6.3: Gini importances (see Appendix A.3) for each phase.

6.3.3 Discussion

The results of removing each phase indicate that the alignment phase is the most important. However, the Gini importances indicate that the Reaching phase is the most

important (splitting almost twice as much data as the Alignment phase). This suggests that, although the Alignment phase is the most distinctive part of the tap gesture, the Reaching phase is still very important to recognising intent-to-pay.

Both experiments support the Withdrawal phase being the least useful. This is likely because the Withdrawal phase is similar in other gestures, so is not very distinctive to tap gestures. Additionally, the Dataset contained a significant portion of data with a partial or missing Withdrawal phase - due to how it was segmented into separate tap gestures - which could have reduced the performance of the Withdrawal phase models.

6.4 In-store Usage

In this section, the ability of the method to be used in-store is evaluated by considering a variant of the gesture model that only looks at data before the NFC contact time and has no withdrawal phase. The results of this can be seen in Table 6.4.

Combination Model	**EER**	**F_1**	**Precision**	**Recall**	**Specificity**
Bayesian Model	0.033	0.868	0.789	0.966	0.969
Decision Tree	0.062	0.894	0.901	0.887	0.988
Random Forest	0.025	0.894	0.826	0.975	0.975
Logistic Regression	0.029	0.882	0.809	0.970	0.972
Boosted Decision Stump	0.031	0.874	0.797	0.969	0.970

Table 6.4: Evaluation statistics for the real-time gesture model.

These results show that this method is not as effective for real-time usage as it is retrospectively (evaluated for the overall performance - see Section 6.2). However, this method still performs well in these circumstances and significantly improves upon the results achieved by Sturgess et al. [1].

6.5 Limited Sensors

This section evaluates the ability of the model to perform when the sensors are limited to just the accelerometer and gyroscope (with the linear accelerometer and GRV excluded). The results can be found in Table 6.5.

These results show only minor changes from the results in Section 6.2. Consequently, the linear accelerometer and GRV sensors are contributing little to no additional information

Combination Model	EER	F_1	Precision	Recall	Specificity
Bayesian Model	0.018	0.926	0.878	0.980	0.984
Decision Tree	0.021	0.965	0.968	0.963	0.996
Random Forest	0.011	0.955	0.925	0.988	0.990
Logistic Regression	0.013	0.942	0.900	0.987	0.987
Boosted Decision Stump	0.016	0.930	0.880	0.985	0.984

Table 6.5: Evaluation statistics for the limited sensor gesture model.

and could obscure the target relationship. As a result, for practical use of this method, these sensors need not be included.

6.6 Unseen Data

This section evaluates the ability of the model to cope with new users or Terminal Configurations.

6.6.1 Unseen User

In this section, gesture models were trained for each user by using that user's data as the test set and using the rest to train the models. The evaluation statistics were then averaged over each of these gesture models to approximate the behaviour of this method for a new user. The results are in Table 6.6.

Combination Model	EER	F_1	Precision	Recall	Specificity
Bayesian Model	0.006	0.898	0.817	0.996	0.992
Decision Tree	0.002	0.954	0.912	1.000	0.996
Random Forest	0.001	0.984	0.969	1.000	0.999
Logistic Regression	0.004	0.948	0.905	0.996	0.996
Boosted Decision Stump	0.004	0.946	0.901	0.996	0.996

Table 6.6: Average evaluation statistics for the unseen user gesture models.

These results are vastly improved over Section 6.2 because of the smaller test set and larger training set. However, the results do still indicate that the method performs well for unseen users.

6.6.2 Unseen Terminal Configuration

This section averages the results for gesture models trained for each Terminal Configuration (as shown in Figure 3.1 and Table 3.1) using that terminal's data for a test set and the rest of the data for training. The results are in Table 6.7.

Combination Model	EER	F_1	Precision	Recall	Specificity
Bayesian Model	0.025	0.887	0.811	0.975	0.975
Decision Tree	0.043	0.941	0.968	0.917	0.997
Random Forest	0.019	0.912	0.852	0.982	0.981
Logistic Regression	0.020	0.906	0.842	0.981	0.980
Boosted Decision Stump	0.019	0.909	0.847	0.981	0.981

Table 6.7: Average evaluation statistics for the unseen terminal gesture models.

These results show that the models perform slightly worse for satisfactory, which suggests that the method can still be applied to tap gestures on terminals that are not present in the training data.

6.7 Feature Importances

This section evaluates which features are most important to recognising each phase.

6.7.1 Overall

The importance of the features to each phase was evaluated by counting the number of times each feature occurred in the modal top 5 features for a phase model by Gini Importance (see Appendix A.3). The results for the top 10 features for each phase can be seen in Table 6.8.

Reaching Phase		Alignment Phase		Withdrawal Phase	
Feature	Count	Feature	Count	Feature	Count
Gyr-x-mean	960	Acc-z-velomax	952	Gyr-x-velomean	960
Gyr-x-median	672	Acc-z-velomean	929	Gyr-x-velomax	960
Gyr-x-velomax	495	Acc-z-disp	839	Gyr-x-mean	944
Gyr-z-mean	483	Acc-z-median	820	Gyr-x-disp	896
Acc-z-iqr	440	Acc-z-max	638	Acc-disp	480
Acc-e_fil-var	397	Acc-z-mean	531	Acc-z-iqr	229
Acc-e_fil-stdev	360	Acc-e_fil-var	47	Gyr-z-mean	226
Acc-e_unf-mean	338	Acc-e_fil-stdev	42	Gyr-x-median	70
Gyr-x-mean	216	Gyr-e_unf-min	2	Acc-e_fil-stdev	17
Acc-z-stdev	141			Acc-e_fil-var	9

Table 6.8: Frequencies with which each feature occurs in the modal top 5 features of a Phase model by Gini Importance. The feature names are in the format sensor-axis-statistic; e_fil and e_unf are the Euclidean norms of the filtered and unfiltered x,y,z values.

These results demonstrate that the Reaching and Withdrawal phases look at similar features, likely due to the similarity of these phases.

In particular, both phases make considerable use of the x-axis of the gyroscope data. As shown in Figure 3.2, this measures the rotation of the watch face around the axis parallel to the wrist. This suggests that the way the wrist is rotated during the Reaching and Withdrawal phases is distinctive to the tap gesture. This is to be expected as other gestures involving a similar movement (for instance picking something up) are usually focused around the hand and not the wrist (in particular there is no target orientation of the wrist) unlike the tap gesture.

Furthermore, both of these phases make use of the variation in the Euclidean norms and z-axis of the accelerometer. This is likely because, during the Reaching and Withdrawal phases, the watch moves a significant distance in a short time accelerating and then stopping. This results in a large variation in the stated features which would be distinctive to this movement.

The Alignment phase looks at different features to the other phases due to its distinctness. Specifically, it looks at the scale of the acceleration in the z-axis (as shown in Figure 3.2, this is the movement perpendicular to the watch face). This is because, during the Alignment phase, the watch is held relatively still against or near the NFC terminal resulting in little movement in this dimension.

Furthermore, the variation in the Euclidean norms of the accelerometer data is used by this phase. This is a result of the limited movement of the watch, during the Alignment phase, leading to a small variation in the norm of the acceleration (the scale is not used as this would vary depending on the effect of gravity).

Interestingly, the features important to identifying the Alignment phase found here disagree with those found by Sturgess et al. [1] to be important to recognising intent-to-pay (see Section 3.3.3). In particular, the previously used method made significant use of the number of peaks in the norm and the variation in the z-axis of the accelerometer. This suggests that, unlike previously concluded, the optimal method suggested by Sturgess et al. [1] does not look for the Alignment phase. It seems probable that the previously suggested method instead identifies distinctive patterns in the phase transitions which cannot be identified by the method proposed in this project.

6.7.2 Limited Sensors

In this section, the feature importances are analysed for the phase models using only the accelerometer and gyroscope data. The results are found in Table 6.9.

Reaching Phase		**Alignment Phase**		**Withdrawal Phase**	
Feature	**Count**	**Feature**	**Count**	**Feature**	**Count**
Gyr-x-mean	40	Acc-z-disp	40	Gyr-x-velomean	40
Gyr-x-median	28	Acc-z-velomax	40	Gyr-x-disp	40
Gyr-z-mean	22	Acc-z-mean	38	Gyr-x-velomax	40
Acc-e_fil-var	20	Acc-z-velomean	34	Gyr-x-mean	40
Acc-z-stdev	20	Acc-z-max	24	Acc-disp	20
Acc-e_unf-mean	18	Acc-z-median	24	Gyr-z-mean	18
Acc-e_fil-stdev	14			Acc-z-stdev	2
Gyr-x-max	8				
Gyr-x-velomax	8				
Acc-z-iqr	8				

Table 6.9: Frequencies with which each feature occurs in the modal top 5 features of a limited sensor phase model by Gini Importance. The feature names are in the format sensor-axis-statistic; e_fil and e_unf are the Euclidean norms of the filtered and unfiltered x,y,z values.

These results show the same trends described in Section 6.7.1. Additionally, neither set of feature importance results contains any features derived from the linear accelerometer or GRV sensors. This supports the conclusion reached in Section 6.5 that these sensors do not contribute to the performance of the model.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This project has successfully achieved its goal of improving upon the intent recognition system proposed by Sturgess et al. [1]. The system in this project achieves an EER of 0.009 and F-measure of 0.966, for the optimal combination, compared with the previously achieved values of 0.04 and 0.89.

Consequently, this project has shown that this system is secure in settings where the system and threat model (see Section 2.1) apply. Also, this system could be used in conjunction with others, such as authentication systems, to protect against unintentional payments with little cost to usability.

Additionally, this project has evaluated the importance of each phase in recognising intent-to-pay. This resulted in the conclusion that although the Alignment phase is most distinctive to the tap gesture the Reaching and Withdrawal phases are still important to creating a good model.

The new system for intent recognition is user and terminal agnostic; its ability to recognise the gestures of unseen users and terminal configurations was analysed and was found to perform better or only slightly worse in this context. Consequently, users don't have to have an enrolment phase in which they perform tap gestures, nor do they have to provide examples when using an unseen Terminal configuration. As a result, the system is easily usable and convenient for the user.

Furthermore, the new system can be adapted for real-time use in stores or for smart-

watches with only an accelerometer and gyroscope with little to no effect on the performance.

Finally, this project provides evidence of the potential of phase segmentation in improving the performance of gesture recognition systems using motion sensor data.

7.2 Limitations and Future Work

The main limitation of this project is the small size of the dataset which only includes 16 users. While this dataset was sufficient to show the feasibility of this method, a larger dataset should be collected including natural tap gestures (and not just those produced artificially in a lab setting) and the method reanalysed on this new dataset.

Another limitation is the manual segmentation of the gesture data into phases for training the phase models. This makes incorporating new training data into the system inefficient since all of the new data would need to be manually segmented. Furthermore, the segmentation could include error and bias due to segmenting only on the motion sensor data. To counteract these limitations, in future work, automated methods for phase segmentation should be explored and evaluated and a different approach for manual segmentation could be considered (for instance using videos of the gestures as well or averaging the transition times produced by multiple people).

Additionally, other model architectures could be considered for the phase models instead of just Random Forests. This would allow a comparison of the different models which could improve the system performance and provide more insight into the distinctive anatomy of a tap gesture.

Another possible avenue for future work would be evaluating other outputs from the phase models. For instance, the timestamp for the optimal window could be included or all of the prediction probabilities. This could allow improved performance as the phases of the tap gesture occur in a particular order.

A variant of this method could be trained and evaluated for a particular terminal configuration. This would result in a more accurate system (for the particular terminal) at the cost of usability (separate models would be needed for each terminal configuration the system is used for).

Finally, the effect of this phase-segmentation approach for gesture recognition on motion sensor data could be further evaluated for other tasks. This could provide a method to improve the performance of gesture recognition systems without a substantial increase to the computational requirements.

Appendix A

Random Forests

A.1 Decision Tree

Random Forests are collections of Decision Trees. This section describes Decision Tree classifiers and how they are trained. An example of a simple Decision Tree classifier can be seen in Figure A.1.

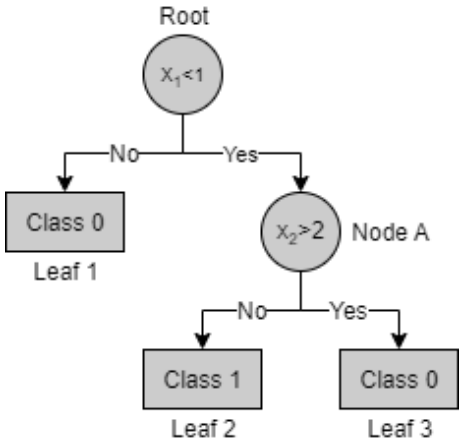

Figure A.1: An example of a Decision Tree classifier.

Decision Trees are formed by a collection of 2 types of nodes - decision and leaf - and the connections between these nodes. The decision nodes each contain a binary condition for the model's input and have connections to 2 child nodes which are visited next according to the state of the condition. For instance, in Figure A.1, there are 2 decision nodes, labelled Root and Node A, with conditions "$x_1 < 1$" and "$x_2 > 2$". The leaf nodes specify a predicted class for the model to return. For instance, in Figure A.1, there are 3 leaf

nodes, labelled Leaf 1,2 and 3, with classes 0,1 and 0 respectively.

The traversal of the tree for any input starts at the root node and passes through a series of decision nodes before arriving at a leaf at which point the specified class is returned. At a decision node, the state of the condition determines which child node is visited.

For example, following the execution through the tree in Figure A.1, for the input $x = (0, 1)$ the condition at the Root node holds so Node A is visited. Then, the condition of Node A is False so Leaf 2 is visited and class 0 is returned.

Decision Trees are trained using Algorithm 1. The **score** is often taken to be entropy or Gini impurity (see Section A.3) and the termination condition is usually all of a node's training data having the same label or a maximum depth.

Algorithm 1: Training Procedure for a Decision Tree

Input: Training data $\mathbf{X}, \mathbf{y}$ with d feature dimensions

Condition Score **score**

Termination Condition **terminate**

Initialise the tree T with an empty root node N

Set *untrained_nodes* $= [N]$

while *untrained_nodes is not empty* **do**

  Remove a node N from *untrained_nodes*

  Let $\mathbf{X}', \mathbf{y}'$ be the training data for node N

  if *not* ***terminate****(N)* **then**

    Randomly select a subset of the d features

    Sort $\mathbf{X}', \mathbf{y}'$ in each of these features and generate candidate conditions by taking the midpoints of adjacent feature values

    Choose the optimal condition according to **score** and assign it to N

    Create 2 child nodes of N and add them to *untrained_nodes*

  else

    Assign node N the most populous class in $\mathbf{y}'$

  end

end

Output: Tree T

A.2 Random Forest

Random Forests are formed by training a collection of Decision Trees. Any input is run through each of these decision trees and the most commonly returned class is given as the class prediction.

Each tree is trained on its own training dataset. The training dataset for each tree is produced by sampling uniformly at random, with replacement from the data provided to train the random forest until a new dataset of the same size has been created.

Training the model in this fashion results in each tree being trained on approximately 63% of the training data (see Appendix E). Consequently, each tree observes a different selection of noise which results in the overall model being more resilient to overfitting (where a model fits too closely to the noise of the training data).

A.3 Gini Impurity/Importance

The Gini Impurity Index is a measure of probability distributions that is often used for training Decision Trees [30]. When the probabilities (approximated by proportions) for each class label in a dataset are $p_1, ..., p_N$, the Gini Impurity Index is defined as

$$\sum_{i=1}^{N} \sum_{j \neq i} p_i p_j = 1 - \sum_{i=1}^{N} p_i^2$$

Gini Importance is a frequently used measure of feature importance within Random Forests and Decision Trees. It is defined as the total number of data split by conditions involving a particular feature over all trees in the forest.

Appendix B

Bayesian Models

The main component of Bayesian classifiers is the use of Bayes' Rule [31]

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B)}$$

In this context, B is the event that a given feature vector $\mathbf{x}$ was produced by the target relationship and A is the event that the datum has a particular class y. Rewritten in this context gives

$$\mathbb{P}(y|\mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}|y)\mathbb{P}(y)}{\mathbb{P}(\mathbf{x})}$$

Typically, Bayesian classifiers ignore the denominator in this equation and just use the numerator as for each possible class for the feature vector the denominator is the same. This results in the calculation of the values $\mathbb{P}(\mathbf{x}|y)\mathbb{P}(y)$.

This demonstrates that Bayesian classifiers assume a distribution for the feature vectors conditional on the class and another distribution for the classes.

Typically, the class distribution is approximated by calculating the proportion of each class that occurs in the training data or, occasionally, by a uniform distribution over the classes.

The conditional distribution of the feature vectors is usually assumed to be the same type for each class. A common architecture is to assume the independence of the binary and real-valued features, and use Bernoulli distributions (see Appendix B.1) for the binary features and (multivariate) Gaussian distributions (see Appendix B.2) for the real-valued features.

B.1 Bernoulli Distribution

A Bernoulli distribution is a distribution over the set $\{0, 1\}$ that is parameterised by a value $p \in [0, 1]$.

For a random variable $X \sim Bernoulli(p)$,

$$\mathbb{P}(X = 1) = p$$

$$\mathbb{P}(X = 0) = 1 - p$$

B.2 Gaussian Distribution

A Gaussian (or Normal) distribution [32] is a distribution over real values and is parameterised by a mean $\mu \in \mathbb{R}$ and a standard deviation $\sigma \in \mathbb{R}$.

For random variable $X \sim Normal(\mu, \sigma^2)$ the probability density function is

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The Gaussian distribution can also be extended to a multivariate form that is parameterised by a number of dimensions d, a mean vector $\boldsymbol{\mu} \in \mathbb{R}^d$ and a covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$. For random variable $\mathbf{X} \sim Normal(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ of d dimensions, the probability density function is

$$f_{\mathbf{X}}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d det(\boldsymbol{\Sigma})}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})}$$

Appendix C

Logistic Regression

The Logistic Regression model has a long history [33] and had great success in early machine learning. It led to the development of the perceptron [34] which was soon used as a building block for early Neural Network models. The model itself is relatively simple and consists only of a linear map followed by a sigmoid non-linearity.

The sigmoid function converts any real number into a value in the range $(0, 1)$ and is often used in machine learning to produce a probability or measure of certainty for classification models. It is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

This creates a logistic regression model with weight vector $\mathbf{w}$ and bias scalar b that on input vector $\mathbf{x}$ produces a prediction

$$\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x} - b}}$$

This type of model is often trained using a gradient descent based optimisation method and a classification loss function such as cross-entropy.

Appendix D

Ada-Boost Algorithm

The AdaBoost (or Adaptive Boosting) Algorithm was the first practical boosting algorithm proposed [35]. It is designed to boost the accuracy of a weak-learning algorithm (i.e. an algorithm that can do slightly better than random guessing) and has theoretical guarantees that, for a large enough number of iterations, the error can be made arbitrarily small, as shown by Freund and Schapire [35].

Algorithm 2: A formulation of the AdaBoost Algorithm [36] using target labels from $\{-1, 1\}$.

Input: sequence of samples $\{(x_1, y_1), ..., (x_N, y_N)\}$,

weak learning algorithm **WeakLearn**,

number of iterations T

Initialise $p_i^1 = \frac{1}{n}$ for $i = 1, ..., N$

for *t=1,...,N* **do**

Call **Weaklearn** with target distribution p^t, getting h_t

$\epsilon_t = \sum_{i=1}^{n} p_i^t |h_t(x_i) - y_i|$

$\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$

$p_i^{t+1} = \frac{p_i^t e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$ where Z_t is a normalisation coefficient

end

Set $h(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$

Output: hypothesis h

Theorem D.0.1. *For any $t = 1, ..., T$, the error of h_t according to p^{t+1} is $\frac{1}{2}$, in Algorithm 2.*

Proof. The error of h_t with respect to p^{t+1} is

$\mathbb{P}_{(x,y)\sim p^{t+1}}(h_t(x) \neq y)$
$= \sum_{i:h_t(x_i)\neq y_i} p_i^{t+1}$
$= \frac{1}{Z_{t+1}} \sum_{i:h_t(x_i)\neq y_i} p_i^t e^{\alpha_t}$
$= \frac{1}{Z_{t+1}} \sum_{i:h_t(x_i)\neq y_i} p_i^t (\frac{1-\epsilon_t}{\epsilon_t})^{\frac{1}{2}}$
$= (\frac{1-\epsilon_t}{\epsilon_t})^{\frac{1}{2}} \frac{1}{Z_{t+1}} \sum_{i:h_t(x_i)\neq y_i} p_i^t$
$= \frac{(\epsilon_t(1-\epsilon_t))^{\frac{1}{2}}}{Z_{t+1}}$

$Z_{t+1} = \sum_{i:h_t(x_i)\neq y_i} p_i^t e_t^{\alpha} + \sum_{i:h_t(x_i)=y_i} p_i^t e^{-\alpha_t}$
$= \sum_{i:h_t(x_i)\neq y_i} p_i^t (\frac{1-\epsilon_t}{\epsilon_t})^{\frac{1}{2}} + \sum_{i:h_t(x_i)=y_i} p_i^t (\frac{1-\epsilon_t}{\epsilon_t})^{-\frac{1}{2}}$
$= 2(\epsilon_t(1-\epsilon_t))^{\frac{1}{2}}$

So the error of h_t with respect to p^{t+1} is

$\mathbb{P}_{(x,y)\sim p^{t+1}}(h_t(x_i) \neq y_i)$
$= \frac{(\epsilon_t(1-\epsilon_t))^{\frac{1}{2}}}{Z_{t+1}}$
$= \frac{(\epsilon_t(1-\epsilon_t))^{\frac{1}{2}}}{2(\epsilon_t(1-\epsilon_t))^{\frac{1}{2}}} = \frac{1}{2}$ □

Assuming that the weak learning algorithm returns a hypothesis that achieved an error of less than $\frac{1}{2}$ (a requirement for weak learning algorithms), Theorem D.0.1 states that the error of this hypothesis on the new distribution is exactly $\frac{1}{2}$. Consequently, the probabilities of the data that the hypothesis misclassified must have increased.

The result is the frequently misclassified data become more prevalent in the distribution, focusing the weak learning algorithm on these more difficult data and eventually correctly classify them.

This ends in a set of hypotheses that are combined according to how accurate they were on their respective target distribution. The result is the output hypothesis that typically has much better accuracy on the data than a hypothesis produced directly from the weak learning algorithm.

Appendix E

Approximating the proportion of training data seen by a tree in a Random Forest

Lemma E.0.1. $\forall x \in \mathbb{R}$, $lim_{n\to\infty}(1+\frac{x}{n})^n = e^x$

Theorem E.0.2. *The expected proportion of objects that occurs in a sample of n objects which are drawn uniformly at random with replacement from a set of n distinct objects tends to* $1-\frac{1}{e}$ *as n tends to* ∞.

Proof. Let X_i be an indicator variable for the inclusion of object i in the sample of size n as described in the statement above [i.e. $X_i = 1$ when the object is included and 0 otherwise].

The probability that object i is drawn at least once is $1-(\frac{n-1}{n})^n$

Consequently, $\mathbb{E}(X_i) = 1-(\frac{n-1}{n})^n$

By linearity of expectation, the expected proportion of objects included within the sample is

$\mathbb{E}(\frac{1}{n}\sum_{i=1}^{n} X_i) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}(X_i) = 1-(\frac{n-1}{n})^n$

By lemma E.0.1,

$lim_{n\to\infty}\mathbb{E}(\frac{1}{n}\sum_{i=1}^{n} X_i) = lim_{n\to\infty}1-(1-\frac{1}{n})^n = 1-e^{-1} = 1-\frac{1}{e}$. □

Consequently, for sufficiently large n, the expected proportion of objects that occur in

a sample of n objects which are drawn uniformly at random with replacement from a set of n distinct objects is close to $1 - \frac{1}{e}$. Since these are the conditions that the training data for a single tree in a Random Forest is produced (see Appendix A.2), the expected proportion of the dataset seen by the individual tree is around $1 - \frac{1}{e} \approx 0.63$ or 63%.

Bibliography

[1] J. Sturgess, S. Eberz, I. Sluganovic, and I. Martinovic, "Watchauth: User authentication and intent recognition in mobile payments using a smartwatch," in *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, June 2022, pp. 377–391.

[2] R. de Best, "Uk: Contactless mobile payment users 2025," Feb 2023. [Online]. Available: https://www.statista.com/statistics/1031085/number-of-mobile-payment-users-in-the-uk/

[3] G. Torbet, "How does a drive-by nfc hack work?" Jun 2022. [Online]. Available: https://www.makeuseof.com/tag/drive-nfc-hack-work/

[4] M. Conti, I. Zachia-Zlatea, and B. Crispo, "Mind how you answer me! transparently authenticating the user of a smartphone when answering or placing a call," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, 2011, pp. 249–259.

[5] A. H. Johnston and G. M. Weiss, "Smartwatch-based biometric gait recognition," in *2015 IEEE 7th International Conference on Biometrics Theory, Applications and Systems (BTAS)*. IEEE, 2015, pp. 1–6.

[6] F. Hong, M. Wei, S. You, Y. Feng, and Z. Guo, "Waving authentication: your smartphone authenticate you on motion gesture," in *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, 2015, pp. 263–266.

[7] B. Shrestha, M. Mohamed, S. Tamrakar, and N. Saxena, "Theft-resilient mobile wallets: Transparently authenticating nfc users with tapping gesture biometrics," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, 2016, pp. 265–276.

[8] W.-H. Lee and R. Lee, "Implicit sensor-based authentication of smartphone users with smartwatch," in *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, 2016, pp. 1–8.

[9] A. Lewis, Y. Li, and M. Xie, "Real time motion-based authentication for smartwatch," in *2016 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2016, pp. 380–381.

[10] S. Davidson, D. Smith, C. Yang, and S. Cheah, "Smartwatch user identification as a means of authentication," *Department of Computer Science and Engineering Std*, vol. 6, pp. 17–26, 2016.

[11] W.-H. Lee, X. Liu, Y. Shen, H. Jin, and R. B. Lee, "Secure pick up: Implicit authentication when you start using the smartphone," in *Proceedings of the 22nd ACM on symposium on access control models and technologies*, 2017, pp. 67–78.

[12] N. Al-Naffakh, N. Clarke, and F. Li, "Continuous user authentication using smartwatch motion sensor data," in *Trust Management XII: 12th IFIP WG 11.11 International Conference, IFIPTM 2018, Toronto, ON, Canada, July 10–13, 2018, Proceedings 12*. Springer, 2018, pp. 15–28.

[13] S. Lee, W. Choi, and D. H. Lee, "Usable user authentication on a smartwatch using vibration," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 304–319.

[14] A. Parate, M.-C. Chiu, C. Chadowitz, D. Ganesan, and E. Kalogerakis, "Risq: Recognizing smoking gestures with inertial sensors on a wristband," in *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, 2014, pp. 149–161.

[15] C. Xu, P. H. Pathak, and P. Mohapatra, "Finger-writing with smartwatch: A case for finger and hand gesture recognition using smartwatch," in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, 2015, pp. 9–14.

[16] H. Wen, J. Ramos Rojas, and A. K. Dey, "Serendipity: Finger gesture recognition using an off-the-shelf smartwatch," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 2016, pp. 3847–3851.

[17] S. Sen, K. K. Rachuri, A. Mukherji, and A. Misra, "Did you take a break today? detecting playing foosball using your smartwatch," in *2016 IEEE international conference on pervasive computing and communication workshops (PerCom Workshops)*. IEEE, 2016, pp. 1–6.

[18] J. Ranjan and K. Whitehouse, "Towards recognizing person-object interactions using a single wrist wearable device," in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, 2016, pp. 722–731.

[19] P. Zhu, H. Zhou, S. Cao, P. Yang, and S. Xue, "Control with gestures: A hand gesture recognition system using off-the-shelf smartwatch," in *2018 4th International Conference on Big Data Computing and Communications (BIGCOM)*. IEEE, 2018, pp. 72–77.

[20] S. Tahir, A. Raheel, M. Ehatisham-Ul-Haq, and A. Arsalan, "Recognizing human-object interaction (hoi) using wrist-mounted inertial sensors," *IEEE Sensors Journal*, vol. 21, no. 6, pp. 7899–7907, 2020.

[21] A. Kendon *et al.*, "Gesticulation and speech: Two aspects of the process of utterance," *The relationship of verbal and nonverbal communication*, vol. 25, no. 1980, pp. 207–227, 1980.

[22] J. Davis and M. Shah, "Visual gesture recognition," *IEE Proceedings-Vision, Image and Signal Processing*, vol. 141, no. 2, pp. 101–106, 1994.

[23] K. Nickel and R. Stiefelhagen, "Pointing gesture recognition based on 3d-tracking of face, hands and head orientation," in *Proceedings of the 5th international conference on Multimodal interfaces*, 2003, pp. 140–146.

[24] Y. Yin and R. Davis, "Real-time continuous gesture recognition for natural human-computer interaction," in *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2014, pp. 113–120.

[25] J. Sturgess, S. Eberz, I. Sluganovic, and I. Martinovic, "Watchauth dataset (original)," 2022. [Online]. Available: https://ora.ox.ac.uk/objects/uuid:8c7cbd89-7953-40ff-b28f-3aa3f8c59a01

[26] S. Eberz, K. B. Rasmussen, V. Lenders, and I. Martinovic, "Evaluating behavioral biometrics for continuous authentication: Challenges and metrics," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 386–399.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[28] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array

programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[29] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[30] J. Mingers, "An empirical comparison of selection measures for decision-tree induction," *Machine learning*, vol. 3, pp. 319–342, 1989.

[31] J. Joyce, "Bayes' Theorem," in *The Stanford Encyclopedia of Philosophy*, Fall 2021 ed., E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University, 2021.

[32] P. Ahrendt, "The multivariate gaussian probability distribution," *Technical University of Denmark, Tech. Rep*, p. 203, 2005.

[33] J. S. Cramer, "The origins of logistic regression," 2002.

[34] H. D. Block, "The perceptron: A model for brain functioning. i," *Rev. Mod. Phys.*, vol. 34, pp. 123–135, Jan 1962. [Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.34.123

[35] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002200009791504X

[36] R. Schapire, *Explaining adaboost*. Springer Berlin Heidelberg, Jan. 2013, pp. 37–52, publisher Copyright: © Springer-Verlag Berlin Heidelberg 2013.

[37] S. Butterworth, "On the Theory of Filter Amplifiers," *Experimental Wireless & the Wireless Engineer*, vol. 7, pp. 536–541, Oct. 1930.

[38] Wikipedia contributors, "Butterworth filter — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Butterworth_filter&oldid=1142938039, 2023, [Online; accessed 5-April-2023].

[39] P. H. Swain and H. Hauska, "The decision tree classifier: Design and potential," *IEEE Transactions on Geoscience Electronics*, vol. 15, no. 3, pp. 142–147, 1977.

[40] T. K. Ho, "Random decision forests," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, 1995, pp. 278–282 vol.1.

[41] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 161–168. [Online]. Available: https://doi.org/10.1145/1143844.1143865