Department of Computer Science

Hamilton, New Zealand

Continuous Typist Verification using Machine Learning

Kathryn Hempstalk

This thesis is submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy at The University of Waikato.

July 2009

Abstract

A keyboard is a simple input device. Its function is to send keystroke information to the computer (or other device) to which it is attached. Normally this information is employed solely to produce text, but it can also be utilized as part of an authentication system. Typist verification exploits a typist's patterns to check whether they are who they say they are, even after standard authentication schemes have confirmed their identity. This thesis investigates whether typists behave in a sufficiently unique yet consistent manner to enable an effective level of verification based on their typing patterns.

Typist verification depends on more than the typist's behaviour. The quality of the patterns and the algorithms used to compare them also determine how accurately verification is performed. This thesis sheds light on all technical aspects of the problem, including data collection, feature identification and extraction, and sample classification.

A dataset has been collected that is comparable in size, timing accuracy and content to others in the field, with one important exception: it is derived from real emails, rather than samples collected in an artificial setting. This dataset is used to gain insight into what features distinguish typists from one another. The features and dataset are used to train learning algorithms that make judgements on the origin of previously unseen typing samples. These algorithms use "one-class classification"; they make predictions for a particular user having been trained on only that user's patterns.

This thesis examines many one-class classification algorithms, including ones designed specifically for typist verification. New algorithms and features are proposed to increase speed and accuracy. The best method proposed performs at the state of the art in terms of classification accuracy, while decreasing the time taken for a prediction from minutes to seconds, and—more importantly—without requiring any negative data from other users. Also, it is general: it applies not only to typist verification, but to any other one-class classification problem.

Overall, this thesis concludes that typist verification can be considered a useful biometric technique.

Acknowledgments

This thesis could not have been completed without the assistance of a number of people, too many to mention them all individually. I am grateful to anyone who has helped me, supported me, and given me the strength necessary to complete such a substantial piece of work as this thesis. However, there are several groups who I would like to personally thank for their considerable input into my work.

First, I would like to thank the Tertiary Education Commission and the University of Waikato for the generous scholarships they have provided me with. This thesis would not have been possible without their financial support. Second, I would like to express my gratitude to the authors of "Learning to Identify a Typist" and "Keystroke Analysis of Free Text" who kindly provided me with their datasets. Your work catalyzed my interest in typist verification, and the datasets provided me with great insights - which are reflected in this thesis.

Third, I would like to thank my supervisors, Eibe and Ian, who have provided me with the guidance, encouragement and academic aid necessary to complete this thesis. Ian, your confidence in my work and optimism was constantly reassuring - I feel truly honoured to have you as a supervisor. Equally, Eibe your advice was always welcome, and your assistance with technical aspects of this thesis was much appreciated. Without either of you I am certain I would not have been able to complete this thesis.

Last but not least, I would like to acknowledge the support provided by my family, friends, lab co-inhabitants, Team xG, pets and partner. Mum, Dad and Paul - you have always been there for me and words cannot express how grateful I am of that. To all my friends and colleagues, thank you. To those of you still working on your PhDs - hang in there! To Lily and Oscar, for always brightening my day and showing that cat-hugs really are the best kind of hugs. And to Mark - your patience, faith in me and care of me will forever be cherished.

Contents

Abstract i

Acknowledgments iii

1 Introduction 1
- 1.1 Verification Systems . . . 2
- 1.2 Thesis Statement . . . 3
- 1.3 Motivation . . . 6
- 1.4 Contributions . . . 8
- 1.5 Thesis Structure . . . 9

2 Background 13
- 2.1 Biometrics . . . 15
  - 2.1.1 Biometric System Processes . . . 16
  - 2.1.2 Evaluating Biometric Measures . . . 17
- 2.2 Typing as a Biometric . . . 19
  - 2.2.1 Terminology . . . 20
- 2.3 Password Hardening . . . 22
- 2.4 Static Typist Verification . . . 28
- 2.5 Continuous Typist Verification . . . 29

3 Replicating the State of the Art 35
- 3.1 Statistical Methodology . . . 35
- 3.2 Learning To Identify A Typist . . . 37
  - 3.2.1 Algorithm . . . 38
  - 3.2.2 Experimental Setup . . . 39
  - 3.2.3 Results . . . 41
- 3.3 Keystroke Analysis of Free Text . . . 42
  - 3.3.1 Algorithm . . . 42
  - 3.3.2 Experimental Setup . . . 44
  - 3.3.3 Results . . . 45
- 3.4 Comparison of Techniques . . . 45
  - 3.4.1 Comparison Using the ROC Curve . . . 46

3.4.2 Comparison Using Datasets . . . 46
3.5 Summary . . . 49

4 Data Collection **51**
4.1 Collecting Data . . . 52
4.1.1 Recording with SquirrelMail . . . 53
4.1.2 Technical Issues . . . 55
4.1.3 Ethical Issues . . . 57
4.2 Final Dataset . . . 58
4.3 Re-evaluation of Existing Algorithms . . . 60

5 Preliminary Experiments with New Techniques **63**
5.1 Methodology . . . 64
5.2 PPM-Based Classifier . . . 65
5.2.1 Algorithm . . . 66
5.2.2 Results . . . 67
5.3 Spread Classifier . . . 68
5.3.1 Algorithm . . . 68
5.3.2 Results . . . 70
5.4 Context Classifier . . . 71
5.4.1 Algorithm . . . 71
5.4.2 Results . . . 72
5.5 Individual Digraph Classifier . . . 73
5.5.1 Algorithm . . . 74
5.5.2 Results . . . 75
5.6 Performance Comparison . . . 76
5.7 Summary . . . 77

6 Typist Behaviour **79**
6.1 The Digraph . . . 80
6.2 From Digraphs to Finger Movements . . . 86
6.3 Key Usage . . . 88
6.4 Pausing . . . 89
6.5 Ordering of Events . . . 91
6.6 Speed and Error Rate . . . 93
6.7 Summary . . . 94

7 One-Class Classification **97**
7.1 One-Class Classification versus Multi-Class Classification . . . 99

7.1.1 Comparing Classifiers Fairly . . . 100
7.1.2 Evaluation Method . . . 102
7.1.3 Results . . . 103
7.1.4 Summary . . . 106
7.2 Methods of One-Class Classification . . . 107
7.3 Combining Density and Class Probability Estimation . . . 109
7.3.1 Combining Classifiers Using Bayes' Rule . . . 110
7.3.2 Combined Classifier Performance . . . 112
7.3.3 Generating Different Proportions of Artificial Data . . . 114
7.3.4 Using Real Data . . . 115
7.3.5 Replacing Components . . . 116
7.4 Summary . . . 117

8 Typist Verification as One-Class Classification 119
8.1 Typing and One-Class Classification . . . 120
8.1.1 Methodology . . . 120
8.1.2 Features . . . 121
8.1.3 Results . . . 123
8.2 Adding Mouse Patterns . . . 125
8.2.1 Features . . . 126
8.2.2 Results . . . 128
8.3 Summary . . . 129

9 Conclusions 131
9.1 Collecting Data . . . 132
9.2 Identifying Channels of Information . . . 133
9.3 Requirements for System Training . . . 134
9.4 Revisiting the Hypothesis . . . 135
9.5 Future Work . . . 135

References 137

A Experiment Details 145
A.1 Purpose of Experiment . . . 145
A.1.1 Questionnaire . . . 145
A.1.2 Email Recording . . . 145
A.2 Mail Analysis . . . 150
A.3 Network Access . . . 154

A.4 Data Collection 154
A.5 Data Archiving/Destruction 155
A.6 Confidentiality 155
A.7 Example Email Recording 156
A.8 How to read an email recording 157

B Instructions For Participants **159**
B.1 Requirements 159
B.2 Links 159
B.3 CS Webmail 159
B.4 Mail Analysis 160
B.5 Recording Content 166
B.6 Final Review 166

C Questionnaire **167**

D List of Top 54 Predictive Digraphs **171**

List of Figures

2.1 Alphabetic layout (from [65]). 14
2.2 The original QWERTY keyboard layout (from [64]). 14
2.3 The enrollment process for biometrics. 17
2.4 The prediction process for biometrics. 18

3.1 The main workbench window 36
3.2 Partial ROC curves for KAOFT and LTIAT 47

6.1 Distribution of times for `Backspace Backspace` 81
6.2 Times for `Backspace Backspace` for Participants A and J (one sample) 82
6.3 Times for `Backspace Backspace` for Participants A and J (all samples) 82
6.4 Times `th` for Participants A and J (one sample) 83
6.5 Times `th` for Participants A and G (one sample) 84
6.6 Slur rate versus typist speed (WPM) 93

7.1 Cross-validation for biased two-class classification (with relabelling) 101
7.2 Cross-validation for one-class classification (with relabelling) . . 101
7.3 Cross-validation for unbiased two-class classification (with relabelling) 101
7.4 Classes removed versus weighted AUC for the audiology dataset. 107

A.1 Login screen—standard CS webmail 146
A.2 Inbox screen—standard CS webmail 146
A.3 Compose screen—standard CS webmail 147
A.4 Login screen—RT CS webmail 148
A.5 Inbox screen—RT CS webmail 148
A.6 Compose screen—RT CS webmail 149
A.7 Registration screen—Mail Analysis 150
A.8 Main screen—Mail Analysis 151
A.9 Reconstructed email screen—Mail Analysis 151
A.10 Reconstructed email screen showing raw recording—Mail Analysis 152

A.11 Summary screen—Mail Analysis 153
A.12 Options screen—Mail Analysis 153
A.13 Login screen—Mail Analysis 154

B.1 Login screen—RT CS webmail 160
B.2 Inbox screen—RT CS webmail 161
B.3 Compose screen—RT CS webmail 161
B.4 Registration screen—Mail Analysis 162
B.5 Main screen—Mail Analysis 163
B.6 Reconstructed email screen—Mail Analysis 163
B.7 Reconstructed email screen showing recording—Mail Analysis . 164
B.8 Summary screen—Mail Analysis 165
B.9 Options screen—Mail Analysis 165
B.10 Login screen—Mail Analysis 166

List of Tables

2.1 Password hardening techniques 22
2.2 Static typist verification techniques 28
2.3 Continuous typist verification techniques 30

3.1 Experimental results for the LTIAT algorithm 41
3.2 FRR/IPR results for the LTIAT algorithm 41
3.3 Experimental results for distance measures R and A 45
3.4 Experimental results for combined R and A measures 45
3.5 Summary of comparison experiment results 47

4.1 Emails recorded per participant 59
4.2 Re-evaluation results using new datasets 61

5.1 Effect of PPM model order on prediction 67
5.2 Final results for PPM using order = 1 67
5.3 Final results for the Spread Classifier 70
5.4 Effect of different context lengths on the Context Classifier . . . 73
5.5 Final results for the Context Classifier 73
5.6 Final results for the Individual Digraph Classifier 75
5.7 Average build and classification times 77

6.1 AUC values for popular digraphs 86
6.2 AUC values for movement types 87
6.3 AUC values for pausing between words 90
6.4 AUC values for key ordering measures 93
6.5 AUC values for WPM and backspace rate 94

7.1 Weighted AUC results for multi-class, two-class and one-class classifiers 104
7.2 Weighted AUC results for reduced numbers of non-target classes 106
7.3 Weighted AUC for two-class and one-class classifiers 113
7.4 Effect of $P(A)$ on the weighted AUC 114
7.5 Results of using real negative data in one-class classifiers 116

8.1 Final results for the Combined OCC typist classifier 124

8.2 Results using different sample lengths 125
8.3 AUC values for the Mouse Use feature 127
8.4 AUC values for the Mouse Backspace feature 128
8.5 Results for Mouse Use and the Combined OCC Typist Classifier 128

Chapter 1
Introduction

For over one hundred years, keyboards have been an essential tool for the production of any printed text. As technology has developed, the keyboard has evolved from the humble typewriter to become the standard input device for modern computers. Yet despite the proliferation of keyboards—especially over the last 30 years as the popularity of personal computers has exploded—little thought has been given to using a keyboard for anything other than entering text. This is not surprising: the keyboard was originally intended as an input device and to this day its primary purpose remains unchallenged.

Just like any other tool, we would expect that people regularly using a keyboard will become proficient at typing. "Typing" is simply "writing with a typewriter or computer" [74], but an extended definition states that it is the "process of inputting text into a device, such as a typewriter, computer, or a calculator, by pressing keys on a keyboard" [75]. Both these definitions cover every entry technique from hunt-and-peck input to skilled touch typing. Considering keyboards as mere entry devices overlooks a very important notion suggested by the last sentence; if different people have different techniques for entering text, perhaps the keyboard could also be used as an authentication or identification device.

It is this notion that forms the basis of this thesis. Is it possible to verify the identity of a keyboard user based on *how* they type? This is a different task to authorship verification, which focuses on *what* is written, rather than *how*. It is also different from identification or recognition: identification and recognition ask "who am I?" whereas verification and authentication ask "am I who I claim I am?" [39]. In summary, typist verification makes a judgment of whether that person is indeed who they say they are by examining the way they type.

1.1 Verification Systems

Just as doors only protect rooms from people who cannot turn a handle, authenticated computer sessions are only secure against people who cannot use a computer. If security is important, access must be controlled—by locking the door or computer so that only those with the key can gain entry. The "key" used depends on what is being controlled. It may be a physical object—in the case of a door, a metal key cut with grooves—or a piece of information that must be committed to memory, such as a PIN or password. The security of the system relies on the key being held by only the authorised user; security is forfeited if the key is lost, duplicated or passed on to anyone else.

Furthermore, access control systems typically verify a user only once—when they first present themselves to the system. Depending on what the system controls access to, the user may also be required to notify the system when they are finished. For example, at the border of a country you are required to present identification whenever you enter or leave. In a computer system, a login consisting of a username/password pair is often used to gain access to the system, and a simple "log off" command is used to indicate that the computing session has ended. The main difference between these two scenarios is that in the latter it is possible for another individual to use the system without needing to identify themselves first. If the user does not notify the computer system that they are finished, it will continue to operate under the assumption that the authorised user is still in control.

To combat this, the computer could be automatically locked after a certain period of inactivity. Additional hardware or software could be employed to detect when the user is no longer operating the computer; the software application BlueProximity locks a computer when a physical token is no longer sufficiently close that it is detected [27]. However, neither of these solutions are complete: an unauthorised individual can access the system if it is still active, or when they have the key.

Biometrics can provide a solution that ensures continuous security, without requiring the user to remember information or possess a particular token. Biometrics literally translates to "life measurement" [78], and it relies on measurements of the user themselves to perform authentication. If the recorded measurements do not match the user's profile, the person is refused access to the system. By placing biometric sensors directly in front of the user they can be continuously verified without interrupting their normal tasks.

Typist verification is an example of a biometric authentication system

where the keyboard serves as the biometric sensor. The system monitors a user's typing patterns in order to perform authentication. If the current pattern is sufficiently different from the reference one, the user is refused access to the system. However, unlike traditional access control systems, monitoring can occur continuously. This means that typist verification can act as both the access control key and as a policeman that constantly checks that the authorised user is the only one in control of the system.

These two modes of access control are often separated; systems that check a user's identity when they first present themselves are known as "static" or "fixed text" typist verification, and those that operate continuously are known as "continuous", "dynamic", or "free text" typist verification. In a continuous system users are free to enter whatever they please and the keystroke data is collected until the session is terminated. In contrast, static typist verification requires the user to type some known text and only records events that occur whilst this text is being entered. Continuous verification is more difficult because there is no restriction on what is typed; it is possible for a new sample to contain radically different sequences of key events to what has been entered previously.

1.2 Thesis Statement

This thesis argues that

> there is enough information in a user's typing input for continuous typist verification to be a useful form of biometric authentication.

Continuous typist verification is the scenario introduced in the previous section: verifying users' identities by examining the way they type, without placing any restrictions on what they type. The input is a stream of key events (key presses and releases) and the time that each occurs. This thesis considers only verification, not identification: success for the task of verification implies that identification is also possible. Also, verification is more appropriate in a computer system controlled with a login process because the expected identity of the user is known.

The question of whether the level of verification that is achieved is useful or not depends on several parameters:

1. The set of possible users.

2. Constraints on how quickly an impostor should be identified.

3. The different kinds of typing information that can be monitored.
4. How much information is available to the monitoring program.
5. Technical and ethical issues.
6. The amount of user variability between reference samples and test samples.
7. The desired level of accuracy.

The set of users should not affect the performance of a verification system because, in principle, there is no need to know about how other people type to confirm a given identity. However, this depends on the design of the system. For example, Gunetti and Picardi [33] compare the test sample to reference samples from every other user in their database, causing the system to perform well when there are many registered users and to fail completely when there is only one. An ideal system would not have this requirement, and this thesis investigates whether negative information (that is, data from other users) is necessary for successful verification.

Constraints on how quickly a behaviour-based verification system responds can greatly affect how well it performs. If it is required to respond quickly to an illegitimate user, the size of the sample being verified must be short enough that an impostor need only type a small amount before being refused access. Although passwords are often only 6 to 10 characters in length, the text is always identical and because it is well known by the user, the pattern of entry for the password is well defined. This means that a static verification algorithm can correctly verify a user with little input. With continuous verification, however, there may not be enough elements in common between the test sample and the reference profile to make a confident classification. Conversely, when samples are large enough to make a confident classification they may contain so much text that they are no longer useful for security purposes. Larger samples also take longer to classify than shorter ones. If the system is required to provide a response in a timely manner after the sample has been collected, it may be unable to do so if large samples of input are provided. Later chapters investigate the trade-off between system classification accuracy and sample length.

As well as size, the content of a typing sample also affects how well typist verification performs. If the recorded sample contains both key press and release events and their associated times, the following types of information can be obtained:

- Overall typing speed and accuracy.
- Genre of the task (e.g., email, instant messaging).
- Correction habits (e.g., whether backspace or delete is used for a given correction).
- Modifier key patterns (e.g., whether the left or right `shift` key is pressed to get a capital letter).
- Fine details of keystroke timings, including key press durations and key-to-key latencies.
- Key press and release ordering patterns (e.g., whether a given key is released before the next one is pressed).

One aim of this thesis is to quantify the amount of identity information carried by each of these channels, in typical situations. As shown in the next chapter, the most common typing patterns currently used for typist verification involve using key press durations and key-to-key latencies.

The amount and quality of the information available to the identification program recording the samples affects the difficulty of the task. For example, if a software timer is used, recorded times are less accurate than when using a hardware timer because the latter is unaffected by system load. However, a software timer is more appealing for typist verification because no extra equipment is required. When recording spans several tasks, the software application being used may affect the way the user types. Due to operating system security, it may not be possible for installed software to automatically determine the identity of the current application. This can be solved by implementing typist verification within each application, but this may be impractical. Ideally, a continuous verification system is agnostic regarding the source of keystrokes. In this thesis, only typing from within an email is collected, using a software timer. Chapter 4 discusses this process in detail.

The amount of information that can be recorded may also be affected by technical or ethical issues. Recording typing continuously may result in confidential data being captured, such as bank account details, passwords, or personal correspondence. If samples are stored, there is a risk that an attacker may be able to access any private data they contain. If the computer system monitoring key events is not strictly controlled, technical issues may arise from insufficient, incorrect or missing data. In these cases the system may choose

to ignore the samples; using them could damage the integrity or accuracy of the verification system.

The most difficult parameter to control when performing typist verification relates to user variability, that is, the degree to which a user's typing patterns naturally fluctuate. A typist's behaviour can change with illness, mood, stress and fatigue throughout the course of a day. If text is entered with one hand instead of two, perhaps because the typist is holding something or is injured, the pattern of entry will be different. Over longer periods of time—weeks, months and even years—a typist may become more (or less) proficient at typing. Because it is unlikely that any of these effects can be controlled, any potentially useful typing system must be capable of dealing with such variations.

Finally, the accuracy obtained by the system also determines how useful it will be. There is a trade-off between correctly identifying users and impostors: when constraints are relaxed so it is easier for a user to be identified as themselves, it is inevitably easier for an impostor to pass as that user too. If the system is designed to refuse access to all impostors, it may also reject legitimate users on a regular basis. There are no guidelines available to suggest what would be an appropriate level of accuracy, especially since the desired trade-off depends greatly on the cost of each type of error in a particular scenario.

It is considered rather unlikely that the hypothesis stated at the beginning of this section will be upheld except in rather restricted circumstances. Password-based static verification, better known as "password hardening," is one such circumstance, but does not fall within the hypothesis because it is not performed on a continuous basis. A somewhat less restricted circumstance is free typing within a particular program where users can type whatever they like. Previous research (discussed further in Chapter 2) suggests that successful identity verification can perhaps be accomplished in this rather unrealistic scenario, at least under certain conditions. The other end of the spectrum is when the user works naturally throughout a normal working day. This thesis focuses on verifying a user where each sample in the system is an email. This is similar to other continuous techniques described in Chapter 2, but extends them because it uses real emails rather than artificial ones.

1.3 Motivation

The extent to which typing patterns can be used on a continuous basis for verifying an individual's identity is currently unknown. It is known that in some restricted circumstances authentication can be performed using typing from

passwords and usernames, but there has been little investigation of continuous verification. Because of this, there are many unexplored avenues that could contain valuable information about a user's typing habits.

This information is not just valuable for typist verification. It can also be employed for accessibility, interface development and for testing the optimisation of keyboard layouts. For example, if a user "stutters" their typing, pressing a given key more than necessary, the bounce delay can be automatically increased to filter out the extra key presses. Fitts' Law [25] can be applied to an interface design to evaluate its effectiveness, but instead of using an estimate that considers each key press as an equal quantity, individual—and more accurate—key event times could be used. These same key event times could be used to determine whether one keyboard layout is better than another. In fact, the above scenarios have already been investigated in practice [73, 26, 35], but the full data was never released; only the results of the experiments were made available.

Studies of typing habits have traditionally focused on increasing a typist's speed. This is an admirable goal, and a large amount of work is devoted to it. Some of this research can be used to give insights into a typist's behaviour, and this is discussed in Chapter 6. Unfortunately no datasets have been found to be available because much of the relevant research happened over 50 years ago and is based on typewriters, not computers.

Continuous typist verification is arguably more desirable than any other scenario posed here: it allows the integrity of unattended computers to be protected without great expense. The problem differs from static verification or password hardening, but is applicable in a wider range of scenarios because there is no need to place any restrictions on what the user types. Continuous typist verification is designed to detect when a user does not appear to be themselves, usually when an impostor starts using the system. However, it may also be the case that the user themselves demonstrates abnormal behaviour. This may be just as important to detect. In situations where a high degree of risk accompanies abnormal user behaviour, such as operating heavy machinery, refusing access to the system may prevent injuries, or worse.

In order to discover how useful continuous typist verification really is, two concerns need to be addressed. The first is to gather a dataset of recorded typing from a range of individuals. The second is to use this dataset for an in-depth investigation into typist behaviour. This thesis addresses these concerns by studying current techniques, collecting a dataset, exploring the behaviour of the monitored typists, and presenting new algorithms that can perform typist

verification on a continuous basis.

1.4 Contributions

The discussion in Section 1.2 and the motivation from the previous section is summarised in the following objectives:

1. To discover whether negative information is necessary for successful verification.

2. To investigate the trade-off between sample size and classification accuracy.

3. To clarify the ethical and technical issues associated with typist recognition and find methods to address them.

4. To identify channels of information in typing input and the amount of identity information in each.

5. To investigate whether mouse button press and release information can be used to increase the accuracy of typist verification using keyboard patterns.

By addressing the above objectives, this thesis makes a number of contributions, including:

1. A new approach for performing one-class classification.

2. The definition of a domain where one-class classification is effective.

3. New algorithms for dealing with typist data.

4. A demonstration that continuous typing input can be described accurately by aggregate features rather than a raw stream.

5. Tools for logging key events, mouse events and their timings across a range of different platforms.

6. A dataset of anonymous users and their typing input in a real-world situation.

An important contribution is the new approach for performing one-class classification. The implications of this method extend beyond typist verification; it is completely general and can be applied to any one-class problem. Typist verification is one example of a one-class problem; detecting cancer is another. Another important contribution is an investigation showing when one-class classification should be used for two-class problems, even when negative data is available. The remaining contributions all relate to typist verification, and will be discussed in detail in later chapters.

1.5 Thesis Structure

Chapter 2 gives an overview of biometrics, and introduces the concepts that will be used to describe typist verification in subsequent chapters. It also surveys related work, covering existing techniques for password hardening, static typist verification and continuous typist verification. This survey is helpful in determining what might be an appropriate level of accuracy, in general, for typist verification. It provides benchmarks for both prediction time and accuracy, which are later used to evaluate new techniques introduced in this thesis.

Chapter 3 examines two of the techniques from Chapter 2 in detail. Each has been re-implemented using the Java programming language, and evaluated using the datasets from the original studies. The two techniques are then compared to each other using accuracy measures and by utilising the other's datasets. This chapter gives insight into the trade-offs between sample size and classification accuracy, and whether negative information is necessary to be able to verify a typist. Chapter 3 highlights the need for a better dataset of typing data: the selected algorithms perform poorly when presented with the other's datasets. This is due to a variety of reasons—including the datasets themselves—that are also discussed in this chapter.

The next chapter, Chapter 4, addresses the inadequacies of existing datasets for evaluating many different typist verification techniques. It begins by describing how email-based typing data was collected from 19 users over a period of 3 months. In total, almost 3000 emails were collected from the participants. However, several issues had to be addressed before these emails could be used for evaluating typist verification systems. Sections 4.1.2 and 4.1.3 discuss the technical and ethical issues that occurred during data collection. Finally, this chapter revisits the work from Chapter 3 and evaluates the re-implemented techniques with the new dataset.

Chapter 5 introduces four new approaches to typist verification, drawing on related work in Chapters 2 and 3. First, a similar approach to the first technique in Chapter 3 is proposed, substituting Prediction by Partial Matching (PPM) for the LZ78-based classifier. Second, a Gaussian-based approach is investigated, using digraph times to create a naïve Bayes style classifier. Third, a context-based classifier is proposed, combining ideas from the first two classifiers. The fourth uses a multi-class classifier to perform verification on individual digraphs. The four techniques are evaluated using the dataset from Chapter 4, and compared with each other and with the techniques from Chapter 3.

Chapter 6 uses one dataset introduced in Chapter 4 to explore how typists behave as they type an email. It begins by examining digraphs, that is, the time between two consecutive key press events. It then moves on to finger movements, drawing similarities between digraphs typed in a similar pattern of fingering. Section 6.4 studies when typists pause, and suggests reasons why they may hesitate—even fleetingly—as they type. Section 6.3 covers the usage of each key, particularly the invisible, modifier and non-alphabetic keys. The exploration in this chapter gives great insight into how typists behave, which of course can be exploited for verification. More importantly, the chapter identifies what channels of information are available in a typing sample.

Chapter 7 broadens the scope by investigating an important machine learning paradigm that can used for typist verification, and other problems. This paradigm is known as one-class classification. One-class classification is the use of machine learning algorithms for prediction, when it has been trained using only positive examples. It is analogous with novelty or outlier detection because it detects abnormal behaviour, except that it does not attempt to detect outliers in the training data. This chapter investigates when it is appropriate to use one-class classification, even when negative data is available. It also surveys existing one-class classifiers. Lastly, a new general technique for one-class classification is introduced, which is one of the main contributions of the thesis.

Chapter 8 relates the work in Chapter 7 to the problem of continuous typist verification. Features are identified using the channels of information suggested in Chapters 6 and 5, and these are used to train a general one-class classifier. The dataset from Chapter 4 is used for evaluation. This chapter addresses the argument of this thesis directly. It also covers the use of mouse patterns to improve classification accuracy.

Finally, Chapter 9 reflects on the work in earlier chapters, and summarises

them. The thesis ends with an outline of possible future work.

Chapter 2
Background

Around 140 years ago, in the late 1800s, two quite different events occurred that together underpin the field of typist verification: Christopher Sholes invented the modern typewriter [64] and Alphonse Bertillon formalised an early form of biometrics known as "anthropometrics"[78]. Typewriters had existed since the early 1700s [17], but until 1878 their keyboards were a single line of keys, often in a piano-like format and arranged alphabetically, as shown in Figure 2.1 [65]. Sholes tiered the keyboard into four rows of keys, in a format that we now call QWERTY (shown in Figure 2.2). The reason for this layout change was to prevent neighbouring typewriting bars from jamming by putting common letter pairs onto separate hands. At the time, touch typing was unheard of; typists used "hunt-and-peck"—first finding the appropriate keys, and then using the index fingers of either hand to strike them [17].

Although not the original intention, QWERTY had the by-product of slowing hunt-and-peck typists because the keyboard was no longer in an intuitive alphabetic format [10]. Touch typing was to be the solution to this problem. Typists learnt to memorise the keys, and type using four fingers of each hand (using the thumb for the space bar). Regimented lessons were eventually introduced to teach the skill of touch typing. The first book published as a study of typing behaviour describes how, despite these lessons, each typist had their own style that reflects their personality [23]. The book outlines how on one occasion a neatly dressed gentleman "turned in typed sheets which were decidedly not neat"[23]. On closer inspection it was revealed that the student was anything but neat in real life, and his appearance was due to the trim naval uniform he was required to wear. This is the first documented example of keyboards (and typewriters) being used to learn something about a typist. In other words, it is the first documented example of typing being used as a biometric.

However, biometrics were used for verification long before typing was invented. One of the first examples can be traced to Ancient Egypt: in the Nile

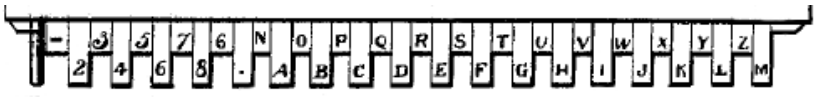

Figure 2.1: Alphabetic layout (from [65]).

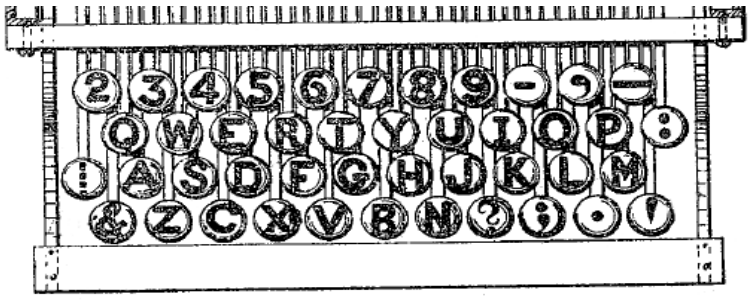

Figure 2.2: The original QWERTY keyboard layout (from [64]).

Valley traders were formally identified using physical attributes such as eye colour [17].

Despite a long history, the field of biometrics was not formally described until after the modern typewriter was invented. Alphonse Bertillon introduced "anthropometrics" (literally, "human measurements") in 1882, formally describing a biometric identification technique for the first time [78]. He took measurements of a person and noted any unusual features such as tattoos or scars, with the intention of using this to identify them [78]. In later years the same idea came to be known as biometrics ("life measurements"). Nowadays, biometrics are used to verify identity through physical characteristics such as fingerprints and behavioural ones such as typing [39].

This chapter provides background information relating to the use of keystroke timings for biometric authentication. The next section describes how biometrics can be used for authentication, how techniques are evaluated and the processes used at various stages. Section 2.2 explains how typing can be used as a biometric, and defines some terms that will be used later in this thesis. The next three sections discuss existing typist verification techniques, from password hardening and static verification in Sections 2.3 and 2.4 through to continuous verification in Section 2.5. The thesis focuses on continuous typist verification, but this chapter explores a wider area to give the necessary background.

2.1 Biometrics

Traditional authentication or identification systems use something *you have*, such as access cards, or something *you know*, such as a PIN or password, to check a user's identity [78]. In contrast, biometrics measure something *about you* [78]. The measurements of a user can be either physiological or behavioural [39]. Fingerprints are an example of a physiological biometric, and typing a behavioural one. There are many others, including voice, infrared facial and hand vein thermograms, face, iris, ear, gait, signature, DNA, odor, hand and finger geometry, and retinal scans [39]. Only four of these have a behavioural component: voice, gait, signature and typing.

Behavioural biometrics are generally considered weaker than physiological ones since they are more susceptible to fluctuations over time. For example, a user who feels tired may type more slowly than normal. Physical biometrics do no have this variability: a tired person may be more sloppy at placing their finger on a fingerprint scanner, but so long as it is aligned correctly their state of mind will have no affect on the measurement. Behavioural biometrics also change over time as the user becomes more practiced at the activity being monitored.

Jain *et al.* [39] state several ideal features of a biometric measure:

- **Universality** Everyone should have it.
- **Uniqueness** No two people should be the same.
- **Permanence** The measure should be invariant with time.
- **Collectability** It should be quantitatively measurable.
- **Performance** The overall system should be accurate.
- **Acceptability** People should be willing to accept the measure.
- **Circumvention** The system should not be easily fooled.

Fingerprinting scores well. Most people have fingerprints (medium universality), they do not change with time (high permanence), the chance of two fingerprints being the same is low (high uniqueness), fingerprints remain the same throughout time (high permanence), current systems are good (high performance), most people do not mind their fingerprints being taken (medium acceptability), and they are easily collected (medium collectability) [39]. The

major downfall is circumvention: advanced fingerprint locks have been defeated with nothing more than a photocopy of a valid print [62].

All these features are subjective: the values from Jain *et al.* [39] were determined from the opinions of three independent experts. According to them, typing has low permanence and performance, and medium collectability, acceptability and circumvention; no suggestions are made about the qualities of universality and uniqueness. In this thesis, Chapter 6 will tackle universality and uniqueness, Chapter 4 deals with collectability and acceptability, and Chapters 3, 5 and 8 cover permanence and performance. This thesis does not consider circumvention directly because no typist was given the opportunity to observe another's habits before attacking the system.

Although physiological biometrics rate better than behavioural ones on many of the above features, they usually require users to present themselves to the sensor, interrupting any task currently underway. On the other hand, behavioural biometrics can be used continuously without causing interruptions. This is where the power of a behaviour-based system lies: it can constantly perform authentication. This extra layer of security is desirable in situations where opportunities exist for an impostor to access an active system. This is especially true for computers because it is easy to leave an authenticated machine unattended.

2.1.1 Biometric System Processes

There are two stages to all biometric systems: enrollment and prediction. Both physiological and behavioural biometrics use the same processes, although the individual components differ depending on the type of biometric that is employed. Figure 2.3 shows the process for enrolling. Data is collected until it is considered acceptable, and then stashed in a data store. Each sample is annotated with the user's ID in the system, ensuring that its source is known. The process may be repeated several times depending on how many samples are required for the system to perform at a satisfactory level.

Once they are enrolled in the system, a user's new input can be checked using the process in Figure 2.4. Samples are collected in the same way as before. Some systems reject users if the samples are inadequate; others simply request them to try again, or to provide more data. An acceptable sample, the user ID and the stored samples known to belong to that user are provided to a matching process, which outputs an estimated probability that the new sample matches the stored samples for that user. If the probability reaches

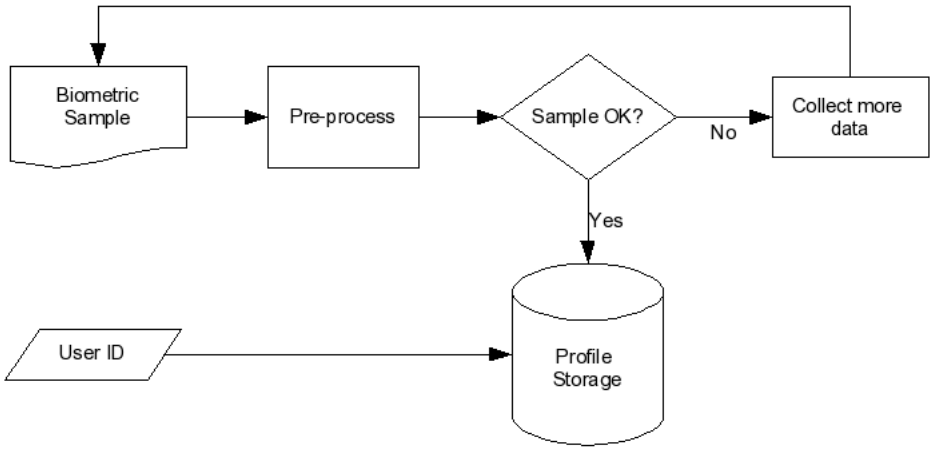

Figure 2.3: The enrollment process for biometrics.

the system threshold the sample is classified positively; otherwise it is rejected as belonging to an impostor.

Between the enrollment and prediction stages, most systems process the stored samples into some kind of model. In this case, the matching process need only compare the sample to the pre-built model—making predictions faster than if the process also had to build the model at this step. The threshold can be varied to change the trade-off between incorrectly rejecting users and allowing impostors through.

2.1.2 Evaluating Biometric Measures

Two types of error determine the effectiveness of a biometric system: Type I and Type II errors; false positives and false negatives respectively. False positives occur when impostors are wrongly classified as the user. False negatives occur when a user is wrongly refused access. In a perfect system, neither would be present. In reality, these errors happen regularly and exhibit a trade-off: increasing the number of false negatives makes it harder for an impostor to pass, so the number of false positives is reduced. The reverse is also true: if it is easier for an attacker to pass, legitimate users are less likely to be rejected. Where security is a concern, systems will attempt to minimise the number of false positives; it is more acceptable to annoy real users than to allow an attacker access.

In typist verification systems, these two types of errors are referred to as the Impostor Pass Rate (Type I error) and the False Rejection Rate (Type II error). The Impostor Pass Rate (IPR) measures the number of impostors that

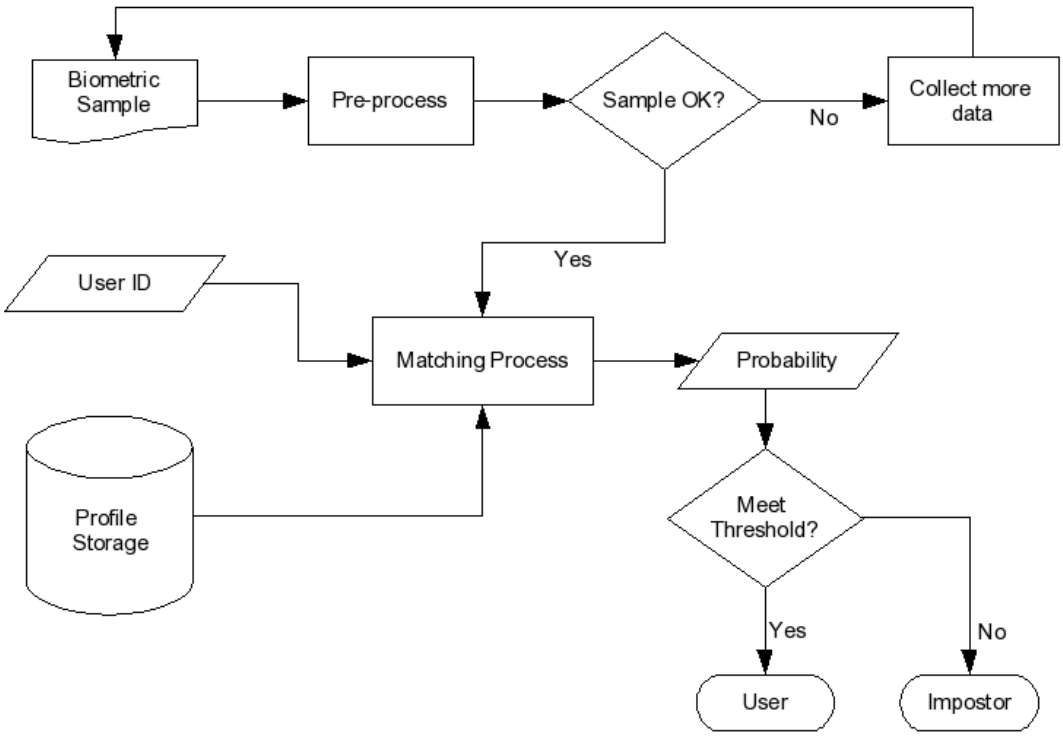

Figure 2.4: The prediction process for biometrics.

are allowed access. The False Rejection Rate (FRR) measures the number of times the system falsely rejects a legitimate user. In many other biometric systems, alternative names are used: the IPR is also known as the False Accept Rate (FAR) or False Match Rate (FMR); the FRR can also be known as the False Alarm Rate (FAR) or the False Non-Match Rate (FNMR). To prevent confusion, this thesis uses the terms IPR and FRR throughout.

The Equal Error Rate (EER) can also be used as a performance measure to compare biometric systems. This is where the FRR and IPR are equal. The Receiver Operating Characteristics (ROC) curve is a plot of the true positive rate versus the false positive rate [77] and the EER, FRR and IPR can be read off it. The advantage of reporting the EER over the FRR and IPR is that it provides a single measure of performance. However, in many situations—especially security—it is more desirable to have an IPR that is as low as possible, so reporting both figures is more useful in determining how useful the system will be. For example, Cho *et al.* [13, 79, 80] report the FRR when the IPR = 0. In the field of password hardening (discussed in Section 2.3) Monrose *et al.* set the FRR to be as low as possible because the typing patterns only have to strengthen the password, not replace it [47]. Existing biometric techniques typically report both the FRR and IPR, tuned to the

security requirements of their real world application.

Some systems report the accuracy or the error rate as measures of performance. These are the percentage of predictions the system got right and wrong respectively. Unfortunately, using these measures conceals the trade-off between false positives and false negatives, so it is difficult to tell whether a system will fit a given situation.

This chapter presents the FRR and IPR of all assessed systems, and elsewhere in the thesis the area under the ROC curve (AUC) is also reported. The AUC gives the probability that a randomly-chosen positive instance will rank higher than a randomly-chosen negative one, meaning that systems with an AUC value close to one are more likely to make correct classifications than systems with a lower AUC value. The AUC is independent of any thresholds, making it possible to assess two systems without needing to consider whether the threshold they utilize is the best one. Another advantage of using AUC is that by generating the ROC curve it is possible to read off the values for the FRR given IPR, or the IPR given FRR.

2.2 Typing as a Biometric

In the Second World War, Morse code operators could be identified by the length of their dots and dashes and their pauses between words and sentences [66]. A remote operator's patterns were often recorded by their home stations; in the event that the operator was captured, false messages sent from the enemy could be easily identified. A Morse key is the simplest possible keyboard: it has a single key that is either on or off. The operator presses and releases the key to generate the code, typing either a dot or a dash depending on how long the key is held down. The rhythm of the operator can be considered a behavioural biometric measure because measuring the lengths of dots, dashes and pauses is sufficient information to verify identity. There is no evidence to suggest that the individual styles of Morse operators proven to exist in the 1930's have not translated to the typewriter/computer keyboard. After all, the standard US computer keyboard is just 104 small Morse keys.

Using a computer keyboard for biometric authentication is achieved in a similar manner to a Morse key. In both cases, a keystroke is composed of two distinct events—a key press (key down) event, and a key release (key up) event. These events are recorded, and stamped with the current time. But because a computer keyboard has many keys whereas a Morse key has only one, the identity of the affected key must be recorded when any event occurs. The

result of recording is a continuous stream of key events and times. Capturing this stream, or "key logging," is infamous on the Internet for its use by hackers to obtain a victim's private data. It can be implemented on a computer using hardware or software, but the latter is more attractive for typist verification systems because no extra equipment is required.

In the 1930's the identity of Morse operators was verified manually. However, the process can be automated on a computer, although this is not always necessary, as shown later in Section 2.4. The computer records the event/time stream and divides it up into samples for processing. Each sample usually corresponds to an entire session of typing, although large samples may be broken up into smaller ones for easier processing.

The samples are either immediately provided to a machine learning algorithm for direct integration into a predictive model or are pre-processed into a set of features first. Both cases result in a model that can be used to make predictions on previously unseen samples of typing, with no manual input. Each approach has advantages and disadvantages, and these will be discussed in later chapters. The main idea of both techniques is the same: when given a new sample of recorded typing, they can predict whether or not it belongs to a given user. The system of recording, processing into samples, extracting features, learning, and prediction of new samples is typist verification.

2.2.1 Terminology

The input to a typist verification system is a stream of key events and the time that each one occurs. Each event is either a press or a release. The stream does not always alternate perfectly between presses and releases. For example, one key may be held down whilst the next is pressed, causing two presses to appear consecutively, followed by the releases. Most verification techniques make use of the time between pairs of events, typically the *digraph time* or *keystroke duration*. The following list explains terms used in typing research:

Key Event A *key event* is a single action with a key. It is a *key press* if the key is pressed down, or a *key release* if the key is being let go. *Key-down* and *key-up* events are alternative terms for presses and releases, respectively.

Key Logging The act of recording key events.

Keystroke A *keystroke* is the press and release of a single key.

Digraph Any sequence of two consecutive key press events, also referred to as a *digram* or *bigram*. Sequences with three characters are *trigraphs*, with four are *four-graphs*, and so on. Digraphs can additionally be classified into groups indicating which finger was responsible for each key, the type of movement being employed, what row the keys were typed on, or even which hand typed each key. For example, 1F, 2F, 1H and 2H digraphs refer to whether the digraph was typed with one finger (1F), two different fingers (2F), one hand (1H), or two (2H). [31]

Digraph Time The *digraph time* is the time between two consecutive key press events. It is sometimes called the *key-to-key latency*, *keystroke latency*, *interstroke time* or *interkeystroke interval*. [31]

Keystroke Duration The *keystroke duration* is the time between the press event and the related key release event for a single key. This is sometimes known as the *key-down time*, *dwell time* or *hold time*. [31]

Inter-key Time The *inter-key time* or the *key delay* is the time between the release of one key and the press of the next. This time may be negative if the key release for the earlier key occurs after the key press event for the later key.

Words-per-minute (WPM) Rate The *WPM Rate* is the number of words that a typist can type in a minute, on average. A "word" is standardised to 5 characters or keystrokes, including spaces. This thesis considers keystrokes: the final number of characters for a sample is not known because a mouse was available for editing in the considered datasets. The estimates of a typist's speed using keystrokes is higher than using characters because inevitably there will be corrected mistakes in a final text that account for extra keystrokes. Traditional calculations of speed on a typewriter utilize uncorrected text; mistakes are scored separately because they are notoriously difficult to correct.

Error Rate The *error rate* is how often a typist makes a mistake. Errors are difficult to automatically detect in a computer system; historically they were marked against a hard copy of copy-typed text. To recognise whether a particular keystroke is intended or not, the reader must have some understanding of the text. Computers do not yet adequately understand text, although they can spell-check for simple mistakes. Instead, the *correction rate*—how often the text is corrected—can be accurately

Algorithm	FRR (%)	IPR (%)	Sample Content	Classifier
Joyce and Gupta [40]	16.36	0.25	Username, password and names	Statistical
D'Souza [21]	24.00	0.00	Username, password and names	Statistical
Bleha *et al.* [9]	8.10	2.80	Name and fixed phrase	Bayes
Monrose and Rubin [49]	≈7.86%	≈7.86%	Username, password and names	Bayes
Ong and Lai [52]	≈15.00	≈15.00	Password	Clustering
Monrose *et al.* [47]	2.00	60.00	Password	Distance
Revett *et al.* [58]	5.60	5.60	Password	Distance
Hocquet *et al.* [38]	3.27	3.75	Username and password	Ensemble
Chang [12]	5.33	1.08	Password	Wavelets
Yu and Cho [79]	0.30	0.00	Password	One-class SVM
Yu and Cho [80]	3.69	0.00	Password	GA-SVM
Rodrigues *et al.* [60]	3.60	3.60	Numeric password	HMM

Table 2.1: Password hardening techniques

observed from a stream of keystrokes, but only if the mouse was not used for editing.

2.3 Password Hardening

Biometric authentication systems typically replace existing authentication systems. They can also extend them, requiring the user to perform authentication normally, and provide some biometric information about themselves as well. Password hardening is an unusual biometric system because it augments an existing authentication system: passwords. It is so named because it "hardens" the strength of a password by ensuring that it is not only typed correctly, but typed *in the right way* [53]. This is an easier task than static or continuous typist verification because the password content is known and it is expected that the patterns are well-defined because users should be familiar with their own passwords.

Unfortunately, it is well known that many users pick bad passwords that can be easily guessed, often choosing actual words or variations of them (e.g. scholar becomes sch0lar) [41]. It is not surprising that this occurs, because in many cases passwords are required to be changed regularly and must meet a minimum length, forcing users to choose something memorable and causing them to be more likely to disclose their password [2]. For example, the New Zealand Government Communications Security Bureau recommends that passwords should be "changed every 90 days", "checked for poor choices", "consist of at least seven characters" and "contain characters from at least three of the following sets: lowercase letters, uppercase letters, digits, punctuation and special characters"[32]. With such strict requirements on passwords, hardening becomes an attractive biometric system because it provides an added layer of security, even when users select bad passwords. Table 2.1 summarises the techniques discussed in this section.

Joyce and Gupta [40] were the first to use password hardening in a modified login environment. They obtained a set of reference samples by requiring each of 33 users to type his/her username, password, first and last names, eight times each. They suggest that those four well-known strings would have an identifiable pattern since the typing "does not involve difficulties like reading text from paper"[40]. In experiments, 975 classifications were attempted (of which 165 came from legitimate users) and the system achieved a FRR of 16.36% and an IPR of 0.25%. Their system used digraph times, where outliers greater than three standard deviations from the mean were discarded before the system began classifying new samples. The classifier was based on a statistical approach. For a sample to belong to the user each digraph was required to fall with 1.5 standard deviations of its reference mean, for all digraphs in the password.

D'Souza [21] used the same approach as Joyce and Gupta, with two small changes. Instead of throwing away digraph samples that were considered outliers, all digraphs were kept. Also, instead of requiring all digraphs to pass, only 80% were required to pass for a sample to be classified positively. The dataset consisted of 11 users with 51-60 login attempts each. The FRR was 24%, higher than that of [40]. The IPR was reported at 0%, but this should be taken with a grain of salt because it was determined from only two impostors attacking four users, and on two occasions the impostors did successfully gain access.

Another similar approach to Joyce and Gupta [40] was proposed by Bleha *et al.* [9]. Their system utilized names and fixed phrases, such as "University of Missouri Columbia". Digraph times were used to train a Bayes classifier, where a threshold was set on the overall probability that a sample belonged to the user. Their system was evaluated using 14 users and 25 impostors, resulting in a FRR of 8.1% and an IPR of 2.8% from 539 legitimate logins and 768 attacks.

Joyce and Gupta's work [40] was extended by Monrose and Rubin [49] in 1999. The same type of content was used for a sample: username, password, and first and last names for a single participant. 63 people provided samples over a period of 11 months, although it is unclear how many samples were provided by each person. In this work several different classification algorithms were investigated. In all cases the digraph times and keystroke durations were calculated for all possible digraphs, and factor analysis was applied to select a subset of features. The users were then partitioned into groups using k-means clustering. Three different classifiers were applied to the same task, all three

using a nearest neighbour approach after calculating a distance or probability for a test sample. The first classifier used a Euclidean distance between pattern vectors. The second used a non-weighted probability; it assumed each feature was distributed normally and independently—that is, a naïve Bayes approach. The third used a weighted probability; it was essentially the same as the second approach, except that features were weighted based on their discriminative power. Of the three methods, the weighted probability performed the best, obtaining an accuracy of 87.18%.

The same authors went on to test a "Bayesian-like classifier" that characterises the performance as a function of the number of classes being discriminated. Each feature vector was assumed to be "distributed according to a Gaussian distribution and an unknown vector is associated with the person who maximises the probability of the measurement vector"[49]. The accuracy of this technique was higher than the other three techniques, achieving 92.14% on the same dataset. Monrose and Rubin's approach is unique compared to other approaches surveyed in this section because they use identification for authentication. As mentioned in Chapter 1, identification systems find the user from a particular group of known users. The problem in this case is that the difficulty of verification depends on the set of users the system was trained on. Some continuous typist verification techniques also use this approach, as we will see later in Sections 2.4 and 2.5 and also in Chapter 3.

In 2000, Ong and Lai [52] asked 20 people to type three passwords 20 times, resulting in 60 samples. They repeated this on two other occasions, each on separate days and with the same users, over a period of two months. Their classifier used the digraph times and a modified k-means clustering algorithm. If a given password formed a cluster on its own, it was rejected as belonging to an impostor. Compared to other techniques, their system performed poorly. Their best results have a FRR and IPR of approximately 15% each.[1] However, they make an interesting observation: when impostors were allowed to watch users enter their passwords they had a higher chance of a successful attack.

Monrose *et al.* [47] performed password hardening using a system where all users had the same password. 481 successful logins were collected, for 20 users. Features were selected from digraph times and keystroke durations, forming a vector of distinguishing features for each user. The Hamming distance was used to determine how close a reference vector was to a test one. Their system had an unusually high IPR of 60%, and a FRR of 2%. This is explained

[1]These values were read off a graph and may not be exact, no tables or actual values were reported in the work.

by their focus on hardening logins, instead of replacing them. Rather than ensuring that it is as difficult as possible for an attacker to pass, they consider it more important that the FRR is low because the password patterns are not being used on their own.

More recently, Revett *et al.* [58] used 14 character passphrases from eight users for password hardening. They use the average, median, standard deviation, and coefficient of variation to form a feature vector for each digraph. The reference feature vector for each digraph was compared to the values of an unknown sample, using a custom distance measure that incorporated all four features. The distances for each digraph were summed, to obtain an overall distance between the reference profile and the user. If the overall distance reached the threshold for each user, the unknown sample was classified positively. Each password was entered 12 times to train the system, and attacked 16 times by each of 43 attackers. In this work, the authors evaluate the system using the EER, instead of separating the IPR and FRR. Their system obtained an EER of 5.6%, but using smaller (different) passphrases that were only eight characters long they managed to achieve an EER of 4.1%.

Another approach that used a four-feature vector for each pair of keys was introduced by Hocquet *et al.* [38]. In this case the feature vector consisted of four timing values for a single digraph: digraph time, keystroke duration, inter-key time and time between releases (i.e. both fingers up). These feature vectors were used to train a one-class classifier in combination with a handful of other features, such as the mean time and associated standard deviation of the entire sequence. In total, a set of 31 features was available. The classification algorithm was a fusion of three classifiers: a statistical classifier, a time discretizer and a ranked time method. The dataset used for evaluation contained 20–110 logins and 20–100 attacks for each of 38 users, and each sample was a username and password pair between eight and 30 characters long. When user-specific parameters were employed their system obtained a FRR of 3.27% and an IPR of 3.75%, the lowest of all the techniques discussed so far in this section.

Keystroke timing vectors are a popular feature set for password hardening. Chang [12] also used timing vectors. First, a vector of digraph and duration times for a password was stored in a keystroke timing vector. Second, discrete wavelet transformation was applied to the timing vector, producing a keystroke wavelet co-efficient vector in the relevant frequency domain. Finally, both the original timing vector and the wavelet vector were scored for consistency. For prediction, a sample belongs to a user if the score reaches the threshold.

Only 20 keystroke patterns were used to train the system; an additional 75 legitimate and 75 attacker samples were used to evaluate each password. In total, 21 different passwords were evaluated, achieving a FRR of 5.33% and an IPR of 1.08%.

The dataset used to evaluate Chang's work [12] was originally collected for work by Yu and Cho [79]. They hold the current record for password hardening, attaining a FRR of 0.3% and an IPR of 0% for the best linear support vector machine (SVM) classifier, where impostors are not allowed any practice. The FRR jumps to 0.8% when the impostors are allowed to practice the passwords first. In the full dataset, 21 users typed their password between 150 and 400 times and 15 imposters typed each password five times unpracticed, and also an additional five times after being allowed to observe how the user typed the password. Only the last 75 patterns typed by each user were used for testing; the rest were used to train the classifier. Each password was stored in a timing vector that contained the digraph and duration times for the entire password; however, if any of the vector elements were in the upper or lower 10% of times, the entire vector was discarded. Depending on the user's consistency, between 20% and 50% of their data may have been discarded. It is unclear whether consistency checking was performed only on the training data. Finally, a SVM set up for one-class classification was used to determine whether a given sample came from a user or an impostor. Two and four layer auto-associative neural networks were also tested, but ultimately the SVM produced the best results.

The same authors produced a similar method the following year [80]. They wrapped a genetic algorithm (GA) around a SVM. The GA is used to "cleanse" the data, selecting relevant features for the final SVM to use. They used the same dataset from their previous work, achieving a FRR of 3.69%—much higher than their previous technique. However, no timing vectors were thrown away, and only 50 (randomly selected) samples were used to train each GA-SVM model.

Although all the password hardening techniques discussed thus far operate on alphanumeric sequences, there are some situations where a password may be limited to a particular set of keys. All the techniques would continue to work in this scenario, but Rodrigues *et al.* [60] address this problem directly. They consider passwords that only contain numeric characters, such as PIN codes for an ATM. Each password was eight numeric characters long, and was typed 40 times by a user and 30 times by an attacker. Twenty users participated, using one password each. Each user's samples were collected at a rate of ten per session, in four separate sessions. Using a hidden Markov model (HMM),

their classifier accomplished an EER of 3.6%, similar to techniques that utilize full alphanumeric password content.

Regardless of which keys are pressed, the patterns associated with typing are not limited to timings between key events. Lau *et al.* [44] suggest that relative key orderings for events and shift key patterns could be used for authentication. For the former, they hypothesize that users press and release keys in a unique way. Considering the two keys A and B, three different ordering patterns can generate the phrase AB: $\{A_{down}, A_{up}, B_{down}, B_{up}\}$, $\{A_{down}, B_{down}, A_{up}, B_{up}\}$ and $\{A_{down}, B_{down}, B_{up}, A_{up}\}$. A shift key pattern is the case of which shift key is pressed for each letter. Lau *et al.* propose that there are four classes of users: strict left shift users, strict right shift users, users who use both the left and right and are consistent for a given letter, and users who use both but are inconsistent on some letters. Four categories are hardly enough to form a security measure on their own; it is possible for a lucky impostor to fall into the same class as the user. The idea of shift key patterns and relative key orderings could nevertheless be used to strengthen existing time-based approaches.

Another method of strengthening existing approaches was proposed by Cho and Hwang [14]. They focused on improving the quality of the timing vector patterns because, traditionally, inconsistent timings are either discarded or removed via feature selection. They used musical rhythms from popular tunes to attempt to create more unique entry patterns for passwords. They also attempted to increase entry consistency by providing audio and/or visual cues. They found that "the proposed artificial rhythms were found to be significantly more unique than those [...] obtained with a natural rhythm," and the samples typed with cues "were found to have decreased inconsistency in all cases"[14]. Unfortunately, this study was only a preliminary one with a single user, and the authors admit that further trials are needed. There is another concern: password hardening makes it more difficult for an impostor to successfully access a system even when the password is known, but increasing the consistency and providing a rhythm for entry could actually make it easier for an impostor to attack the system. For example, if the attacker knows the password and the rhythm, and the cue is provided as part of the entry process, they may find it easier to mimic the user's patterns than the case where no rhythm or cues were ever used.

Password hardening has arguably had great success. Including the techniques discussed here, commercial systems like BioPassword [8] are available. Not all use just the password; some use additional information such as the

Algorithm	FRR (%)	IPR (%)	Sample Content	Classifier
Gaines *et al.* [28]	0.00	0.00	6000 characters of prose	Manual
Umphress and Williams [76]	12.00	6.00	1400+300 characters of prose	Statistical
Bergandano *et al.* [7]	0.00	0.14	683 characters of prose	Nearest Neighbour

Table 2.2: Static typist verification techniques

user id, the user's name, or fixed phrases. However, they are all examples of password hardening because they attempt to strengthen an existing password-based authentication mechanism.

2.4 Static Typist Verification

Password hardening is a special case of static or "fixed text" typist verification. In both cases, the text entered by the typist is known in advance. The difference between the two is that password hardening strengthens an existing authentication system, but static typist verification can completely replace it. In this case, the text is presented to the typist to copy-type, and this is what is used for authentication. Although the content may be known in advance, the typist is not necessarily proficient at typing it. This is counteracted by using samples that are typically much longer than those used for password hardening, often an entire sentence in length, or longer. Table 2.2 summarise a few techniques that are intended to replace an existing authentication scheme.

Gaines *et al.* [28] was the first to formally introduce the idea of identifying a user by their typing patterns, and did so using static typist verification. In Gaines's experiments, six professional secretaries provided two samples each with a gap of four months between their collection. One sample contained three passages, varying in length but each with around 2000 characters. The digraph time was calculated for all keystrokes in the input. It was found there were 87 digraphs that appeared ten or more times in all samples for all typists, and after calculating significance statistics it was discovered that five of these digraphs together discriminated perfectly between the six secretaries. The five selected digraphs were `in`, `io`, `no`, `on`, `ul`. The last digraph, `ul`, could be substituted with `il` or `ly`, since the four core digraphs plus one of `ul`, `il` or `ly` gave perfect authentication. Even though the recordings were performed on a computer, no automated classification algorithm was used to distinguish between the typists—the typists could be perfectly classified by hand.

Gaines's success in static verification was later reinforced by several other papers in the area. Umphress and Williams [76] collected two typing samples from each of 17 users. Both samples were collected by asking the user to

copy-type some text as quickly as possible, presented in the form of a typing test. The reference sample used for training contained approximately 1400 characters of prose, while the test sample only contained around 300 characters. The keystrokes were grouped into words, and the first six digraph times for each word were calculated and stored in a matrix. The classifier was similar to Joyce and Gupta's [40]: each digraph must fall within 0.5 standard deviations of its mean to be considered valid. A threshold was placed on the proportion of valid digraphs, and samples that met or exceeded it were classified as belonging to the user. Their system obtained a FRR of 12% and an IPR of 6%.

The current state of the art in static typist verification was achieved by Bergandano *et al.* [7]. Instead of digraph times, they used trigraph times for features. Each trigraph was placed in an ordered list based on its average time (and then alphabetically if two or more trigraphs had the same average time). The idea is that a legitimate user's sample will be in approximately the same order as their reference sample, because anything that may affect a typist's pattern will do so by changing all average trigraph times in a roughly uniform way. For every user the lists of trigraphs from the test sample and all reference samples were compared. For a sample to be classified as belonging to a given user, the test sample must be closer to that user than any other person in the database. Evaluation was performed using 44 users and 110 attackers, the users typing five samples each but the attackers typing only one. The samples were 683 characters long. Their system accomplishes a FRR of 0% and an IPR of 0.14%, but has a FRR of 7.28% when the IPR is set to zero. This FRR is poorer than the best password hardening technique, [79], which achieves a FRR of less than 1%. However, this technique can be used to replace an existing password authentication system, so a different trade-off may be appropriate.

Much of the research on static typist verification is better classified as password hardening because it is designed to strengthen an existing system. There is at least one known example of a commercial use of static typist verification too, Psylock [56]. Psylock replaces a password with a sentence of text to be copy-typed. Its exact performance is not reported; however, the provider's website suggests that it may have an IPR of $<1\%$ and a FRR of $<5\%$ [57].

2.5 Continuous Typist Verification

Continuous typist verification using 'dynamic' or 'free text' is much closer to a real world situation than using static text. It is unrealistic to expect the

Algorithm	FRR (%)	IPR (%)	Sample Content	Classifier
Song *et al.* [67]	-	-	-	Markov Model
Monrose and Rubin [48]	80.00	80.00	Unknown	Weighted Probability
Dowland *et al.* [20]	-	-	Global logging	Statistical
Dowland and Furnell [19]	0.00	4.90	Global logging	Statistical
Nisenson *et al.* [50]	5.25	1.13	Task responses	LZ78
Gunetti and Picardi [33]	3.17	0.03	Artificial emails	Nearest Neighbour

Table 2.3: Continuous typist verification techniques

users to type the same document twice in a world where copy and paste are universal computer functions. Continuous typist verification allows the input to be completely different for each sample. It is also more desirable than password hardening or static typist verification because it continues to check that a legitimate user is operating the system even after a standard log-in procedure has occurred. Unfortunately, it is also more difficult: the user can enter whatever they please, so samples must be sufficiently large that they have enough common elements—otherwise a confident classification cannot be guaranteed. This section describes all known continuous typist verification techniques. They are summarised in Table 2.3.

Song *et al.* [67] investigated using all possible digraph combinations in a profile for a user. They calculated the digraph, duration and interkey times for pairs of keys, as well as the latencies for trigraphs and entire words. Their system can make predictions continuously by employing a Markov model. Given the current state of the system, the model can calculate the probability of a given event using normal distributions for each recorded time. No empirical results were presented in the paper.

Before attempting password hardening as reviewed in Section 2.3, Monrose and Rubin [48] attempted to use their three distance-based classifiers on free text. For evaluation purposes they recorded a combination of fixed and free text samples from 42 users over a period of seven weeks, but reduced their final dataset to 31 users due to timing issues on some machines. For fixed text samples, an accuracy of 90.7% was achieved using their weighted probability classifier. Accuracy diminished significantly when fixed text samples were compared to free text ones, and when free text samples were compared to each other—48.9% and 21.5% respectively. Despite their poor free text results, they claim that with future work "free text will perform comparably to that which can be obtained with structured text"[48].

A preliminary investigation by Dowland *et al.* [20] profiled ten users using digraph times collected using a custom-made high accuracy timer. Any times less than 40ms or greater than 750ms were immediately discarded. If the standard deviation for a given digraph was larger than its mean, the upper

and lower 10% of the values were removed and recalculated, as in Yu and Cho [79] discussed earlier. If there were less than 50 examples of a given digraph, then it was removed completely from the dataset. In order to classify a sample, a similar approach to that of Joyce and Gupta [40] was applied: a digraph was only valid if it came within x standard deviations of its mean. The value x was a parameter of the classification algorithm, and Dowland *et al.* tested the values 0.5, 1, 1.5 and 2. Although no final empirical results were reported, they found that for at least four users the digraph acceptance rate was the highest when a sample did belong to the given user. They concluded that keystroke analysis "had potential" [20].

A longer trial by Dowland and Furnell [19] collected around 3.5 million keystrokes from 35 participants over a period of three months. Keystrokes were logged globally across all applications on the user's computer. Global logging is unique to this work; in all other systems users were recorded when performing a set task. This meant that samples are likely to have a higher variation in typing. For example, a user typing in an instant messaging program is likely to provide short, sharp responses, whereas the same person typing in a text editor may have slower typing patterns because they are concentrating on composing their work. Global logging includes *all* the typist's behaviours, but does not allow the classification system to take advantage of task-specific typing habits.

Dowland and Furnell [19] used an identical method to Dowland *et al.* [20], except that the low pass filter was reduced from 40ms to 10ms to prevent potentially useful digraphs from being removed. The FRR was set to 0%, and using digraphs for classification they achieved an IPR of 4.9%. However, they noted that some users had inconsistent typing patterns, which dramatically affected the average IPR. By removing the five worst users their system achieved an IPR of 1.7%. They argue that a user's typing should be monitored only if "the method was shown to be a discriminating authentication technique for that user" [19].

Nisenson *et al.* [50] attempt to use a compression technique, LZ78, modified for prediction, to verify a typist. They collected free text samples from five users and 30 attackers. Each sample was either an open answer to a question, some copy-typing, or a block of free typing. All samples for each participant were collected in a single session. The samples were kept as a continuous stream of events and time differentials, each becoming a symbol for the classifier. The time differentials were quantized using clustering, preventing the possible number of symbols provided to their classifier from becoming too large. The LZ78-based classifier was trained on the stream of symbols. Given a new

symbol and the context in which it appears, the classifier predicts the likelihood of that symbol occurring—regardless of whether it is a key event or a time differential. Although their system is capable of providing predictions on a single event or time, they sum the log likelihood across all symbols and make predictions on an entire sample. With a user-specific threshold set on the probability, their system attains a FRR of 5.25% and an IPR of 1.13%.

Gunetti and Picardi [33] propose the most accurate continuous approach so far. Their technique is based on static typist verification work by Bergandano *et al.* [7], discussed in the previous section. However, their work differs in two ways. First, instead of considering only trigraphs, they do not limit the size of the n-graphs they rank—only making the restriction that the ranking must occur on lists made up of the same sized n-graphs. They term their relative rank-based measure the "R measure", with R_2 using digraphs, R_3 using trigraphs, and so on. Second, they add an absolute measure, the "A measure". This is used to ensure that attackers who type at a different overall speed to the user will not be classified as the user, even if relatively they appear the same. The two measures can be used separately, or combined. No matter which measures are used, the same nearest neighbour approach is taken as in [7], so that the performance is dependent on the set of users the system was trained on.

Evaluation was performed using a set of 40 users, each providing 15 samples of freely typed Italian text approximately 800 characters long. 165 attackers additionally provided one sample each. All samples were typed as if the participant was typing an email. Using the set of combined measures $R_{2,3,4}A_2$, and the aforementioned dataset, their system accomplishes a FRR of 3.17% and an IPR of 0.03%. Again, predictions are made periodically on entire samples rather than individual events. They argue that small samples can simply be merged together to form larger ones, so the need for large samples of text is not an issue. The main concern with this approach is that although it achieves an accuracy comparable to password hardening techniques, it requires a good database of users in order to do so.

In principle, the requirement of data from other users seems counter-intuitive. It should be possible to confirm identity without making comparisons with a large group of people. However, Gunetti and Picardi's [33] technique achieves a commendable accuracy, and does so utilizing an innovative relative approach. Nisenson's approach is also interesting: treating the event/time stream as a sequence allows the context of a digraph to be taken into account. This context is not considered by any other technique where empirical results

are available, even though there are many references in the literature that allude to it being a factor in typing.

The last two techniques discussed, Nisenson *et al.* [50] and Gunetti and Picardi [33], are reviewed in greater detail in the next chapter, where they are also re-implemented and compared to each other. Although numerous candidate techniques for replication have been discussed in this chapter, these two methods take unique approaches to typist verification. They also produce low FRR and IPR values, making them good choices for further investigation because they are already effective continuous typist verification techniques.

Chapter 3
Replicating the State of the Art

This thesis argues that there is enough information in a user's typing patterns to use them for biometric authentication, and as we have seen in the last chapter, many research projects have attempted to substantiate this same hypothesis. However, the broad overview in Chapter 2 does not describe the intricacies of typist verification—in many cases the system has been set up in such a way that the reported accuracy can only be achieved in restricted situations. To demonstrate this, this chapter evaluates two techniques introduced in Chapter 2: *Learning to Identify a Typist* [50] and *Keystroke Analysis of Free Text* [33]. These two techniques report by far the best results for continuous typist verification—around 96.8% (optimized parameters) and 98.4% (optimized parameters and using the best distance measure) accuracy respectively. None of the others report accuracy levels consistently above 95%.

The two selected techniques employ different approaches to the problem. The first sets a probability threshold for a single user, while the second finds the nearest neighbour between a given sample and all other users in the system. In each case, the authors were emailed to clarify small points of detail and obtain the original datasets. The techniques were then re-implemented (in the Java programming language) and evaluated using the original dataset from the published study. The process of reconstruction, the comparison of the two systems and the results of the experiments all provide fresh insight into difficulties associated with the problem of continuous typist verification.

The next section summarizes the statistical methodology and the test harness used for evaluation. The details of the experiments and the results obtained are described in Sections 3.2 and 3.3. Finally, the merits of the two methods are compared in Section 3.4, and Section 3.5 draws some conclusions.

3.1 Statistical Methodology

To facilitate testing under identical conditions, an interactive workbench for the algorithms and data was built. The workbench allows users to load pre-

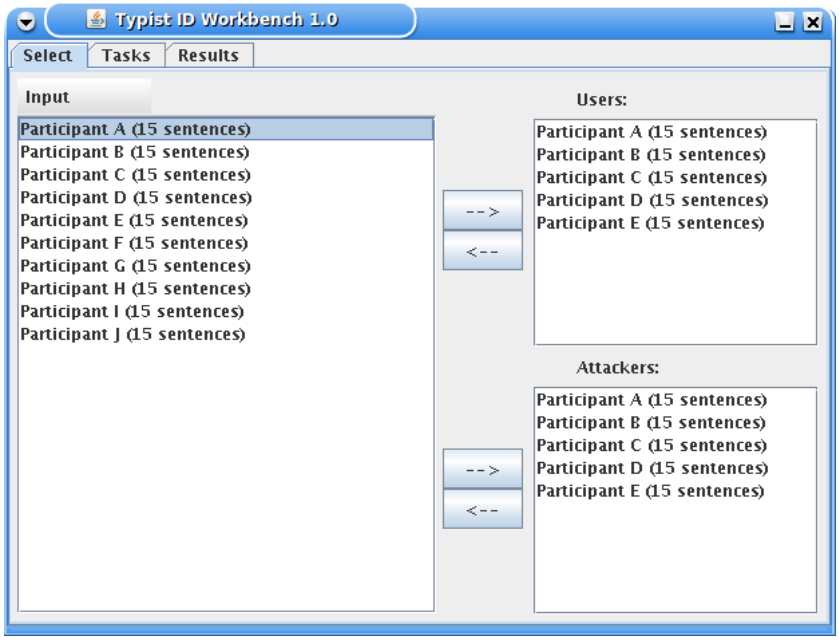

Figure 3.1: The main workbench window

recorded typist data, partition it into groups, and perform self-identification and user-vs-attacker tests based on a particular algorithm and partition. These algorithms are adaptive: they work by building a model for each user from that user's data and all other users' data. The purpose of the model is to distinguish between that user's data and the rest, and classify new data samples as belonging to that user or not.

In order to make the most of the available data, a "leave-one-out" test methodology is adopted. For a self-identification test, one sample belonging to a user is held out when the model is built. The held-out sample is then tested against the model and the success of the test is recorded. A full self-identification run for a particular user involves repeating this procedure for each of the user's typing samples, and the average result gives the overall self-identification accuracy for that user. Averaged over all users, self-identification determines the false rejection rate (FRR) of the system: the chance that a legitimate user will be erroneously identified as illegitimate.

For a user-vs-attacker test, a model is built for a given user from their entire set of samples minus one held-out sample. All the attacker samples are classified using the hold-one-out model. This is repeated for every possible hold-one-out model for the user. Since the technique in Section 3.2 requires no negative data for building a user's model, data from other users can be used for attacking the system without having to hold them out when building the model. The second technique (see Section 3.3) uses a nearest neighbour

approach among all users, so before utilizing a user's samples for attacking, their data must first be temporarily removed from the model. The overall accuracy of the user-vs-attacker test is the cumulative accuracy for all hold-one-out models tested with the attacker data. The user-vs-attacker test gives the impostor pass rate (IPR) of the system.

Ten-times ten-fold stratified cross-validation is also performed using each dataset. In this case, the data is randomly and equally distributed into ten groups, or "folds", and a model trained using nine of the ten folds. The outstanding fold is used for evaluation. This is repeated for every possible held-out fold, and the overall process repeated ten times to ensure that any results are not caused by a particular fold sampling.

The results from cross-validation can be used to obtain the area under the Receiver Operating Characteristics (ROC) curve, usually known at the AUC [77]. As discussed in Chapter 2, the AUC is a measure of classifier performance on a dataset that is independent of any arbitrary thresholds set by the classification algorithm: it is an estimate of the probability that a user's sample will rank higher than an attacker's one. The AUC results were not presented in the original studies, but are included here to allow a direct comparison between techniques. In all experiments the weighted AUC is used, to ensure that users with different numbers of training samples do not adversely affect the results. The weighted AUC is calculated by determining the AUC for each user, and weighting this value by the proportion of samples in the system belonging to that user.

3.2 Learning To Identify A Typist

The LTIAT (Learning To Identify A Typist) technique was proposed in Nisenson *et al*'s paper [50] *Towards Biometric Security Systems: Learning To Identify A Typist*. It uses both key-press and key-release events to build a profile of a user. LTIAT is based on the well known Lempel-Ziv (LZ78) lossless compression algorithm [82], modified to make more effective use of a limited amount of training data [50]. The method was originally tested using a dataset containing a single sample from each of 5 users and 30 attackers, which was kindly provided by Nisenson *et al.* for use in these experiments.

3.2.1 Algorithm

The original input is a continuous stream of keyboard events, both key-press and key-release, along with each event's time of occurrence in milliseconds (the actual timing accuracy is discussed below). The first step is to transform this stream into a sentence of events and quantized time differentials. Specifically, the input becomes:

$$e_1, \Delta_1, e_2, \Delta_2, \ldots, e_{n-1}, \Delta_{n-1}, e_n,$$

where e_1 and e_2 refer to the first and second keystroke events recorded, each either a *press* or a *release*, and Δ_1 is the time between the two. Next, the input stream is divided into "sentences," each of which is deemed to end when a maximum time differential Δ_{max} is encountered. This provides a convenient way of partitioning the input stream into independent samples, and of discarding long time delays because keystrokes that are minutes apart are unlikely to be related. Then the differentials Δ are modified by clustering the values using scalar quantization into a predetermined number Q of clusters and replacing each one by its cluster centroid. The actual quantization method used is unclear in [50]; however, k-means is used here after the author confirmed that this would be appropriate.[1] The purpose of this step is to reduce the size of the alphabet from which the items in the input sequence are drawn in a way that also smooths some of the outliers in the timing data. The value of Q was optimized as described below.

To build a model for a user u, the set of training sentences D_u belonging to u are used to build an LZms Tree. LZms is a variant of the LZ78 scheme [82] that is used for text compression. The general idea is to build a tree using the text as input. Each time a new symbol is seen in the current context, it is added as a branch to the tree and the algorithm returns to the root. If the symbol has been seen before at the given node, the algorithm follows the branch and awaits the next symbol. In order to use the LZ78 tree for prediction, each node keeps count of the number of distinct symbols it has seen [43]. Sentences are included into the prediction tree in the manner above, where each key-press, key-release and quantization interval is a symbol.

In order to build a larger tree with many contexts from a small amount of input, LZms makes two modifications to the standard LZ78 method. Such a tree will give an unseen sentence a greater chance of tracing to a leaf node. The first modification is known as "input shifting". The idea is simple: the sentence

[1] Nisenson, personal correspondence, 8 April 2006.

is learnt in its entirety, then the first symbol is removed and the sentence is learnt again. This is repeated s times, where the value of s is optimized as described below.

The second modification is called "backshift parsing" and is designed to provide some prior context when a new sentence is added. When the algorithm returns to the root after seeing a new symbol, m prior symbols are traced down the tree; then normal parsing resumes from the last trace node. The value m determines how many prior symbols are held in context when the algorithm returns to the root, and is optimized as described below in Section 3.2.2.

The probability that an unseen sentence x belongs to a particular user u's model is determined by tracing x through the fully-built tree and calculating the log-likelihood along the way. In order to translate this into a decision, a cutoff threshold is chosen below which x is deemed to belong to an impostor. A suitable threshold is determined when the model is built by repeatedly holding out a sentence from the user's data D_u and classifying the held-out sentence using the model built from the remaining sentences. At run time any sentence whose log-likelihood falls below the threshold is deemed to come from an attacker, while sentences whose log-likelihood exceed the threshold are classified as belonging to u.

3.2.2 Experimental Setup

The LTIAT algorithm was implemented as described by Nisenson *et al.* [50] and reviewed above, and incorporated into the workbench. In order to determine the best values for the parameters—the number of clusters Q, input shifts s and back shifts m—leave-one-out optimization was used. The particular values tested for each parameter were identical to those used in Nisenson's work: Q = 80, 90, 100, 110, 120; m = 0, 1, 2, 3, 4, 5; and s = 0, 1, 2, 3, 4, 5. A single leave-one-out optimization tests all 180 possible combinations of Q, s and m in order to find the settings that minimise the FRR. If more than one combination has the best settings then the combination with the smallest value of Q is used, with the smallest value of s then m being employed to break any ties. Results from this classifier will be optimistic since parameter tuning is performed on a per-user basis.

The accuracy of the times in this dataset appears to be 1–2 ms. The data obtained was already segmented into the sentences used in the original experiment and this same segmentation was used in the experiments here. The original paper reported that "the sentences, both before learning and before

testing, were split into segments of 100 keystrokes (arbitrarily set)", suggesting that the split sentences contained at most 200 keyboard events (key press and release for each keystroke). However, after attempting to reproduce this, the splitting required to get the same number of self-identification attacks was actually 50 events per sentence. This means that the sentences were split into segments with 100 symbols (including both events and their times as symbols), rather than 100 keystrokes.[2] The authors confirmed that no overlapping segments were used. Because the original sentences were not merged together before splitting took place, several blocks contained fewer than 50 symbols—since these smaller sentences were not removed in the original experiments, they were also included here. For the user-vs-attacker tests, the data was not split at all. Although the data for each user was originally obtained in a single typing session, each sentence was treated as an independent sample when calculating the system's FRR and IPR.

Notwithstanding the description in [50], correspondence with the authors confirmed that the original sentences were not segmented by using some Δ_{max} value. Instead, the data was obtained by asking subjects to type the fixed sentence *To be or not to be, that is the question*, and answer certain questions (e.g., *What is your favourite programming language and why?*), and each sentence was defined as a response to one of these tasks. All responses were typed in English and encoded using the keyboard driver codes. There are 164 sentences in total in the dataset; each of 30 attackers provided 4 sentences and the users provided between 5 and 19 sentences each.

Unfortunately the original data is dubious: six attacker sentences contain between four and eight events, corresponding to between two and four characters of text. On closer inspection all six sentences are the 4th sentence provided by each attacker, and correspond to the (assumed) question "Do you have anything else to add?"[3] In five of the six cases the response is "no" and the other is "ok". Extra characters are invisible keystrokes—shift and backspace. Since the smaller sentences are never used for training and are attacker sentences, including these small sentences will artificially inflate the IPR of the system. Ideally a minimum size should be imposed on all experiments to ensure that the size of the sentences does not affect the reported accuracy. This is also

[2]In personal correspondence, the authors conceded that 100 symbols may be correct.

[3]The authors did not clarify what all the questions were; however, after reconstructing the dataset from keyboard driver codes into readable text some of the questions are obvious. In the case of sentence four, one response, including a missing apostrophe, was "I dont anything else to add" and all responses made sense for the question "Do you have anything else to add".

	Classification Errors	Total Classifications
No. of passed impostors	63	6716
No. of false alarms	10	276

Table 3.1: Experimental results for the LTIAT algorithm

Evaluation Method	IPR (%)	FRR (%)	Weighted AUC
Original [50]	1.13	5.25	N/A
Re-implementation	0.94	3.62	N/A
CV	12.92	3.16	0.962
CV with cleansed dataset	8.38	3.79	0.965

Table 3.2: FRR/IPR results for the LTIAT algorithm

true of the self-identification tests—split sentences that have less than 100 symbols should not be included in the experiments. However, it does not appear that the data was preprocessed in the original experiments, so the replicated experiments below do not include any pre-processing either.

Results for ten-times ten-fold stratified cross-validation are also presented for the LTIAT algorithm. The AUC was computed by ranking based on the log likelihood. It has not been calculated for the self-identification and user-vs-attacker experiments—only the cross-validation experiments—because there are different models used for those two experiments (due to the sentence splitting). Results for performing cross-validation on a "cleansed" dataset are also reported in the next section. In this case, all sentences were split into blocks of 50 events and only full sentences that did not contain any less than 50 events were included in the dataset. All experiments were performed on a 2.4GHz machine with 1GB of RAM.

3.2.3 Results

The results from the experiments (shown in Tables 3.1 and 3.2) are approximately the same as those in the original paper, confirming the accuracy of the re-implementation. The non-significant variation can potentially be explained by the choice of k-means as a quantization algorithm; in the original work a different method may have been used. The cross-validation results have a substantially different IPR to the original experiments, although the FRR is similar to previous results. The difference in IPR is justified: for user-vs-attacker tests an entire impostor sample is used for attacking, in the cross-validation tests an impostor's sample is at most 50 events long—much smaller than the original sample lengths used to calculate the IPR and the same length as a user's samples in the self-attack experiments. The LTIAT algorithm appears to have difficulty correctly classifying smaller samples as belonging to an impostor, and so a substantial difference is observed between

the normal and cleansed datasets: the only difference is that the latter required that all samples contain exactly 50 events and the former used samples between 1 and 50 events long.

Each classification takes approximately 10 milliseconds, not including the offline time to build each user's model. The samples in this dataset have not been collected over a sufficiently long period of time to support any conclusions about whether this technique is appropriate for continuous use over periods of several hours, days or weeks. However, these results have verified that it does perform well when presented with accurately timed data collected in a single session.

3.3 Keystroke Analysis of Free Text

The KAOFT (Keystroke Analysis Of Free Text) technique was proposed in Gunetti and Picardi's paper [33] *Keystroke Analysis Of Free Text*. Several samples are collected from a user, and each is transformed into a list of n-graphs, sorted by their average times. To classify a new sample it is compared with each existing sample in terms of both relative and absolute timing. Only digraphs that appear in both the reference and unknown samples are used for classification.

3.3.1 Algorithm

KAOFT uses two measures to characterize the distance between typing samples. The first, introduced by Bergandano *et al.* [7], is known as the "R-Measure" and involves relative times. The second measure is known as the "A-Measure" and was created by the KAOFT authors, Gunetti and Picardi. The R-Measure is based on the durations of n-graphs, that is, the time between the first and last of n subsequent key-presses. First, these are extracted from each sample. Then the n-graphs that are common between two samples are determined, and the list for each sample is ordered by its average time (using alphabetical ordering as a tie-breaker). The degree of disorder of each list is found by taking the sum of the distances between the position of each n-graph in sample 1 and its position in sample 2.

We denote by $R_2(x, y)$ the degree of disorder of sample x with respect to sample y over n-graphs of size $n = 2$. Different R-Measures can be combined: $R_{2,3}(x, y)$ is defined as the sum of $R_2(x, y)$ and $R_3(x, y)$ weighted by the number of graphs used to compute each measure. Formally, R measures are combined

like this [7]:

$$R_{n,m}(x,y) = R_n(x,y) + R_m(x,y)M/N$$

where N and M are the number of n-graphs and m-graphs that the samples x and y have in common. It is required that $N > M$. This formula can be extended to arbitrary numbers of n-graphs by including further terms in the sum. This formula is not asymmetric: although in the above equation $R_{3,2}$ is a different calculation to $R_{2,3}$, the constraint $N > M$ is violated by $R_{3,2}$ because there will be more shared digraphs than trigraphs. There is only one possible way of combining the measures R_2 and R_3, and that is $R_{2,3}$.

The R-Measure copes with effects such as fatigue by assuming that all n-graphs are affected in a similar manner. If the user slows down, their n-graph durations increase but probably retain the same relative ordering. However, users who have different timings but the same relative orderings will be confused with each other. To combat this, an additional "A-Measure" compares the absolute times of n-graphs to ensure that the typing speed is similar enough between the two samples to have come from the same user. Two n-graph durations a and b for the same graph are defined to be *similar* if $1 < \max(a,b)/\min(a,b) \leq t$. We use $t = 1.25$ because this value was found to give the best results in [33]. The A-measure is defined as

$$A_n^t(x,y) = 1 - (\alpha/\beta),$$

where α is the number of similar n-graphs between x and y, and β is the total number of n-graphs shared by these two samples. A-Measures can be combined in the same way as R-Measures, and the two can be summed. For example, $R_2(x,y) + A_2(x,y)$ is the combined distance value for n-graphs of size 2.

To identify whether sample x comes from user A, we first settle on a given distance measure d. Then the mean distance between A's set of reference samples is determined, obtaining $m(A)$. Next we calculate the mean distance between the unknown sample x and all other registered users, B, defined as $md(B,x)$ for each user B. Sample x is deemed to belong to user A if the following conditions apply [33]:

1. $md(A,x) < md(B,x)$ for all registered users B different from A;

2. $md(A,x)$ is smaller than and closer to $m(A)$ than it is to any other $md(B,x)$ value. That is, the following two conditions hold:
   (2a). $md(A,x) < m(A)$

(2b). $md(A, x) < 0.5(md(B, x) + m(A))$

The idea is that to classify x as belonging to A it must resemble A more closely than it does any other registered user, and moreover must be close enough to A as well.

3.3.2 Experimental Setup

The KAOFT algorithm was implemented as described by Gunetti and Picardi [33] and reviewed above, and incorporated into the workbench, using the value for $t = 1.25$, as mentioned previously. The dataset originally used for evaluation, provided by Professor Gunetti, was gathered over a 6-month period and contains 15 samples from each of 40 different users. Users were asked to provide no more than one sample of 700–900 characters per day, but could provide it at any time of day. They were allowed to type whatever they liked, except that they should not type the same text for more than one sample. The sample was collected using a web-based form that recorded ASCII characters and associated key-press times. Attacker data was collected in the same way, except that 165 additional people provided a single typing sample each. Not all users gave permission for their samples to be released to a third party, so the dataset provided contained data for only 21 users, along with the 165 attackers.

Although the dataset used here is only half the size of the original, it still provides useful information for comparison purposes. Because the timer was implemented in user space[4] the resolution is approximately 10 ms. The samples were not broken up in any way for self-identification tests: each sample contains all the data recorded by a user in one session. The samples given to the system are on average 5 times longer than those for the previous technique.

The KAOFT data and algorithm was also tested using ten-times ten-fold stratified cross-validation in order to determine a weighted AUC value that can be compared to the previous technique. Unlike the LTIAT technique, KAOFT cannot produce a probability and is not easily modified to do so. Therefore, the results of the KAOFT experiments correspond to a single point on the ROC curve. The AUC can be estimated by plotting the point, then drawing the curve as two straight lines—from (0,0) to the point, and from the point to (100,100) (assuming a percentage point plot). The AUC is then the area under this "curve".

[4]In current versions of Linux, user space has a resolution of approximately 10 milliseconds, compared to kernel/system space which has a resolution of 1–2 milliseconds.

Table 3.3: Experimental results for distance measures R and A

Adopted Distance Measure	R_2	$R_{2,3}$	$R_{3,4}$	$R_{2,3,4}$	A_2	$A_{2,3}$	$A_{3,4}$	$A_{2,3,4}$
No. of passed impostors	406	572	779	1220	996	1367	1750	1634
No. of false alarms	24	42	47	83	65	101	146	155
IPR (%)	0.28	0.39	0.53	0.83	0.68	0.93	1.19	1.12
FRR (%)	7.62	13.33	14.92	26.35	20.64	32.06	46.35	50.79
Weighted AUC	0.962	0.931	0.918	0.866	0.896	0.837	0.762	0.742

Table 3.4: Experimental results for combined R and A measures

Adopted Distance Measure	$R_2 + A_2$	$R_{2,3} + A_{2,3}$	$R_{2,3,4} + A_2$	$R_{2,3,4} + A_{2,3,4}$
No. of passed impostors	576	784	689	1104
No. of false alarms	20	33	35	83
IPR (%)	0.39	0.54	0.5	0.75
FRR (%)	6.35	10.48	11.1	26.35
Weighted AUC	0.967	0.949	0.942	0.857

3.3.3 Results

Tables 3.3 and 3.4 show the results obtained by repeating the experiments of [33] with the new re-implementation and the data provided. For brevity, only those measures reported in Tables III and IV of [33] are given. There are 315 self-identifications and 146,475 attacks.

The results are significantly worse than those in the original paper. This is most likely because this technique relies on a large set of registered users—a small user database will inevitably make less accurate classifications than a large one—and only half the users were available for these experiments (21 out of 40). One advantage of the technique is that it is robust to fluctuations over time. However, it is affected by events such as distractions—a large pause will affect both the relative and absolute ordering of the n-graph it appears in. The first technique can cope with pauses because a long time differential would simply be quantized into a cluster with a similar value.

The time this method takes to make a classification increases linearly with the number of registered users. The system in [33] takes 140 seconds to make a classification on a 2.5 GHz Pentium IV, with 40 registered users. Like the original system, this re-implementation is far from optimized; it takes around 400 milliseconds on a similar machine to perform a single prediction with 21 users, and 20 seconds to build the model used for prediction.

3.4 Comparison of Techniques

The previous two sections have individually covered the LTIAT and KAOFT typist verification techniques. These two algorithms employ different strategies to obtain a commendable accuracy on their own datasets. However, it is difficult to determine which of the two is the most accurate based on a single

FRR/IPR pair because there is a trade-off between the FRR and IPR. For example, if the FRR is increased by selecting a threshold that makes it harder for a user to pass, the IPR of the system should lower because it also becomes more difficult for an attacker to gain access. Instead, the area under the ROC curve (AUC) can be used for comparison with the caveat that it is based on a single FRR/IPR pair in the case of KAOFT.

Using the AUC values reported so far is still an unfair way to compare the two techniques: each technique has so far only been tested using its own data, so the AUC values have been calculated using different test conditions. Ideally they should be tested using the same dataset, but the datasets collected for each technique do not have appropriate information to be directly used for comparison. Instead, each dataset must be processed in some way in order to be used for testing. Performing tests in this way gives insight into the performance of each technique under restricted conditions. The rest of this section discusses these comparisons.

3.4.1 Comparison Using the ROC Curve

Using the best results for each algorithm, the weighted AUCs are 0.962 for LTIAT and 0.967 using the measure R_2A_2 for KAOFT. This suggests that the two techniques have comparable accuracy, despite differing FRR and IPR scores for each. A finer comparison can be achieved by graphing the ROC curves for one user from each technique's dataset. Figure 3.2 shows partial ROC curves for LTIAT (user 5) and KAOFT (user 21). KAOFT's curve can be achieved in practice by mixing a random predictor in appropriate proportions. Both curves are only partial because in reality the curves continue along the top axis with a true positive rate of 1. Only one user is graphed for each technique because LTIAT produces probabilities that are specific to each user.

Figure 3.2 shows that LTIAT performs the best up until the IPR is 0.2%, having the highest true positive rate up until this point. Increasing the IPR further, it becomes easier for a user to successfully authenticate, and KAOFT becomes the most effective technique.

3.4.2 Comparison Using Datasets

In order to see the merits of LTIAT and KAOFT in greater depth, each was tested using the other's data. The datasets originally used for each method are too diverse to allow direct comparison between the two techniques. That for LTIAT contains both key-press and key-release events and has an accurate

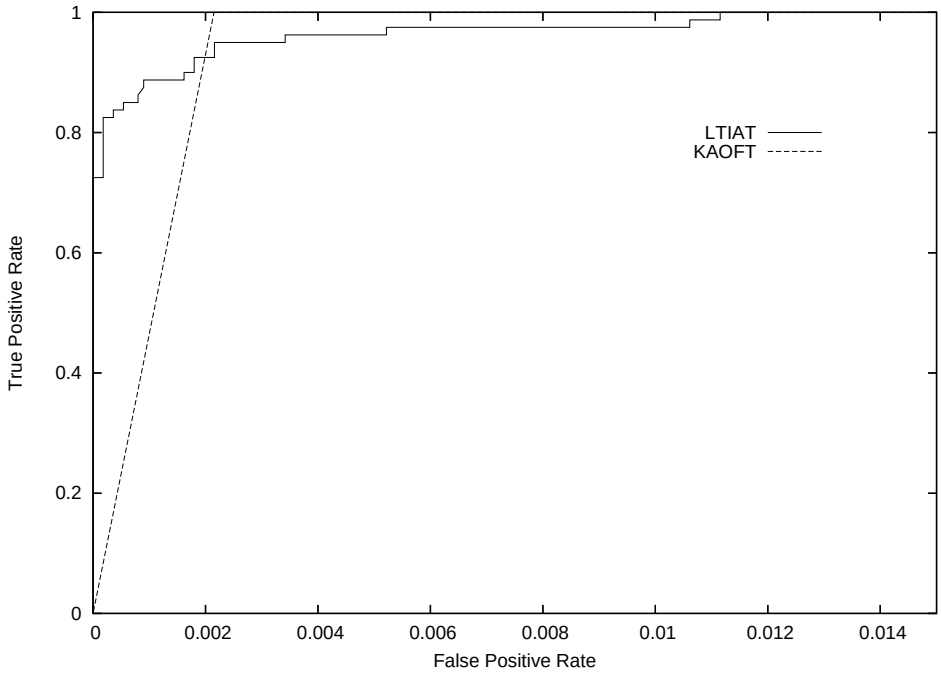

Figure 3.2: Partial ROC curves for KAOFT and LTIAT

Method/Experiment	FRR %	IPR %	Best Measure
LTIAT - Published	5.2	1.1	
LTIAT - Reproduced	3.6	0.9	
KAOFT - Published (40 users)	3.2	0.1	$R_{2,3,4}A_2$
KAOFT - Reproduced (21 users)	6.4	0.4	R_2A_2
LTIAT - Single Divided Sample (KAOFT data)	16.2	19.3	
LTIAT - Random Continuous Sentence (KAOFT data)	10.8	36.0	
LTIAT - 10ms Timing Resolution (LTIAT data)	2.9	13.5	
LTIAT - Key Presses Only (LTIAT data)	7.4	19.3	
KAOFT - Single Divided Sample (KAOFT Data)	35.9	0.5	R_2A_2
KAOFT - Random Continuous Sentence (KAOFT Data)	64.9	0.2	R_2A_2
KAOFT - Using LTIAT data	38.6	12.5	R_2A_2

Table 3.5: Summary of comparison experiment results

timing resolution, but only includes a single session of data for each user. That for KAOFT has 15 samples for each user but has a coarse timing resolution and only contains key-press events. Hence, processed versions of the datasets are considered. The results of the experiments are summarised in Table 3.5. The first block of the table repeats the published and reproduced results discussed in Sections 3.2 and 3.3. The remaining results are described further below.

Firstly, LTIAT was tested using KAOFT's data. Since it took excessively long to complete one self-identification classification using an optimized model (more than 24 hours of CPU time[5]), the data was broken down to resemble that of the LTIAT dataset. A single sample from each user was selected at

[5]For each single self-identification classification all 180 combinations of parameters were tested using hold-one-out models on the training data before classification takes place. The KAOFT data provides a much larger amount of training data and therefore the optimization step takes substantially longer.

random, and divided into sentences of 100 key-press events.[6] The procedure was repeated 5 times and the results averaged, yielding an FRR of 16% and an IPR of 19% (Single Divided Sample in Table 3.5). In a second experiment, one continuous sentence of size 100 was selected at random from each of the user's samples. Again, the procedure was repeated 5 times, yielding an FRR of 11% and an IPR of 36% (Random Continuous Sentence in Table 3.5). Here each sentence is from a different session, whereas all sentences in the first experiment were selected from the same session.

The LTIAT method clearly cannot cope with the KAOFT data. To ascertain whether the inaccuracy was caused by the low resolution timings or the lack of key-release events, it was re-tested twice using its own data. First, all key-release events and their associated times were removed from the data and the times relating to existing key-press events were altered to reflect the removal of the key-release events. Second, both key-press and key-release events were retained, but a 10ms resolution was simulated by adding a random number between 0 and 9 to each recorded time.

When the key-release events were removed, the IPR was identical to that found by dividing up a single sample of KAOFT data (19.3%). This is not surprising considering that both datasets are from a single continuous typing session and contain only key-press events. The FRR remained below 10% for both the removal of key-release events and the introduction of a 10ms resolution. However it is the IPR that is most interesting with regards to security and this was found to exceed 10% in both experiments. It is important to note that when the key-release events are removed, adding their time differentials to the appropriate key-press times introduces a slightly coarser timing resolution than had the data been recorded without the key-release events in the first place. The new resolution is estimated to be 2–4ms, because on average two times, each with a 1–2ms resolution, are used to replace a single one. The experiments show that LTIAT only performs well when provided with all keyboard events and with timing data that has an accuracy of 1–2ms.

The two experiments above were repeated using the KAOFT method. For the measure R_2A_2, the FRR was 36% and 65% and the IPR 0.5% and 0.2% for the first and second experiment respectively. The same pattern was seen through other measures—in the first experiment the FRR was half the size of the second experiment, and the reverse is true of the IPR. The KAOFT method was then tested using the LTIAT data, resulting in an FRR of 38%

[6]The average sentence in Nisenson's data for both users and attackers contains about 100 key press events.

and IPR of 12% (for R_2A_2), a similar FRR to that of the first experiment on the broken-down KAOFT data.

The results on the LTIAT data in particular indicate that the KAOFT method cannot cope with small sample sizes: the amount of information is insufficient to make correct classifications for valid users. Larger samples share more n-graphs, so the computed distance between them is more meaningful. Gunetti and Picardi suggest that short samples can simply be concatenated together to make larger ones [33]. In an attempt to test this, sentences from the LTIAT data were concatenated until the number of key-press events exceeded 300. However, this resulted in only 2 users and 2 attackers having more than 2 samples each, so the number of tests that could be performed was not informative.

3.5 Summary

This chapter has revisited two continuous typing recognition systems introduced in Chapter 2. They were re-implemented and tested with data that was used for evaluation in the original papers. The first, LTIAT, produced similar FRR and IPR rates to those in the original paper. The second, KAOFT, exhibited worse results, most likely due to a reduction in the number of registered users.

Neither can be viewed as the complete answer to continuous typist recognition, and both are restricted to limited situations—such as accurately recorded single sessions for the first method, and long recordings from users and impostors for the second. In both cases, the original data for each method performs poorly when applied to the other algorithm. For the LTIAT technique the poor performance on the KAOFT data can be attributed to the lack of key releases in the data. This means that some important patterns, relating to interleaving of press and release events, are lost. When considering KAOFT, it relies heavily on the availability of sufficiently large quantities of data from other users—a requirement that is not always easily satisfied in a real-world situation.

However, the two techniques do demonstrate that under ideal conditions it is possible to recognise a typist by how they type in a restricted situation where input has been collected for only one task. The results highlight the need for a better dataset—one that contains several paragraph sized samples per user, and has key-release events included. The next chapter covers the collection of such data.

Chapter 4
Data Collection

Collecting typing data for continuous typist verification sounds like an easy task, given that no extra hardware is required and keyboards are a standard input device for most computer systems. Recording simply requires some software to monitor the keyboard and log keystroke information. The user does not need to interact with this software in any special way; they can simply continue using the computer as they normally would. However, accurately collecting typing patterns is not an easy task: there are a range of technical and ethical issues that can affect the integrity of recording keystrokes.

The technical issues surrounding the recording of keystroke data are relatively minor compared to the ethical issues, but they can nonetheless play a crucial role in determining the accuracy of any typist verification system. The most obvious technical concern is what exactly is recorded. In Chapter 2 we saw that Gunetti and Picardi's [33] system requires only the key press events and their associated times—so this was all that was recorded in their dataset. However, when attempting to reuse this dataset in other experiments, including those in the previous chapter, the lack of key release events means that some algorithms are less accurate, and others cannot be used at all.

In evaluation of most typist verification problems, data is collected in an artificial situation. In the case of static verification, users might be required to copy-type some fixed text. For dynamic verification, users might be asked to respond to questions [50], type a paragraph as if it were an email [33] or perform set tasks on a given system [48]. In all of these cases many of the ethical issues present in a real-world scenario can be overcome. However, because users are required to use an artificial system, they are taking time away from their regular tasks to participate. This may restrict the amount of data that can be collected because users may have little incentive to use this system at length. This is evident in Nisenson *et al.*'s dataset [50], which was reconstructed from keycodes in the previous chapter: many of the users commented how bored they were with the task, and provided little input other than what was required to complete their session.

Data quality may also be an issue for artificial situations. Ideally the participants should be typing data that is meaningful to them in order to get representative samples of how they type. This is demonstrated by the success of password hardening, discussed in Chapter 2: users tend to know their user ids and passwords well and thus type them with a distinct rhythm. If the user is forced to type unfamiliar text there is a risk that their normal rhythms will be disturbed by the task of reading or comprehension. Of course, data quality cannot always be guaranteed in a real-world scenario either, although there is at least a greater chance that the user will be performing a familiar task.

It is obvious that both datasets used in the previous chapter are inadequate for further investigation into typist verification. One has only a small amount of data per user that has been recorded in a single session, the other does not contain key release events. Therefore, before experimentation on new algorithms could begin, it was necessary to collect some new typing data that extended over several sessions and contained key release and key press events, as well as their associated times.

This chapter covers the process of collecting new data and how the technical and ethical issues were addressed. Section 4.2 presents the properties of the final datasets, and Section 4.3 discusses the performance of the algorithms from Chapter 3 on these datasets. The collection of a reasonably-sized real-world dataset, which yields results on par with those obtained from restricted datasets that have been collected for use with a specific algorithm, is one of the main contributions of this thesis.

4.1 Collecting Data

In an ideal situation, data for typist verification would come from real-world recording where users were surreptitiously recorded as they perform their usual duties. Unfortunately the process of keylogging also records deleted keystrokes and keystrokes that are encrypted on screen (i.e. passwords). It is difficult to convince a user to allow their keystrokes to be monitored during usual computer use because there is a high probability that confidential information will be captured.

The advantage of being able to record real behaviour is that the user is unlikely to be affected by the constraints of an artificial task. As previously mentioned, users may be unwilling to use an artificial system, as is evident in Nisenson *et al.*'s dataset [50]. Furthermore, if they are required to enter unfamiliar data or use an unfamiliar workstation there may be subtle differences

between their usual patterns and the recorded ones. However, this may not be a problem for system evaluation because it would be unlikely that there would be a mix of keystrokes recorded in artificial and real-world situations. But it does raise the concern that a system evaluated on an artificial dataset may perform poorer than reported if used in practice.

Gunetti and Picardi [33] attempted to create a dataset that contained data similar to that of a real-world situation by asking their volunteers to "enter samples in the most natural way, more or less as if they were writing an email to someone." These samples were entered into a web-based form consisting of one textbox and a 'submit' button, with the amount of text roughly a paragraph in length for each sample. No restrictions were made on the operating system, Internet browser or keyboard used to access and enter information into the form. In total, 15 samples from each of 40 users and a further 165 attacker samples were obtained over a period of 6 months. The main limitation of this dataset is that it contains no key release events.

Instead of setting up an artificial situation, this thesis uses data collected from a real email system. This makes it easy to collect a large number of samples, but it is difficult to ensure that they meet requirements for length, content, or even authorship. However, the limitations of Gunetti and Picardi's dataset [33] can be overcome by ensuring that both key press and release events are recorded. After technical and ethical issues have been addressed, discussed here in Sections 4.1.2 and 4.1.3 respectively, the resulting dataset is comparable in nature to Gunetti and Picardi's dataset [33] except that it is recorded from real emails, not simulated ones. The next section discusses the process of recording the emails.

4.1.1 Recording with SquirrelMail

In early 2007, 19 participants from the University of Waikato's Department of Computer Science gave permission for their typing patterns to be recorded whilst they used their Computer Science email accounts. Their patterns were recorded over a period of just over three months, from mid-April 2007 through until the end of July 2007. The Computer Science Department accesses email through SquirrelMail [71], a web-based system that employs forms similar to the one used in Gunetti and Picardi's experiments. In order to use this email system to record typing patterns, an augmented version of SquirrelMail was set up, known as the recorded typing (RT) version. RT SquirrelMail is almost identical to the standard one, with the exception that it has the ability to

record typing patterns for email. The user interacts with the RT version in the same way as they would normally (it provides access to the participant's standard computer science email account).

RT SquirrelMail records the typing for each of the text entry boxes on the compose screen: *To*, *CC*, *BCC*, *Subject* and *Body*. Mouse press events (clicks) within the compose frame are also recorded. When the user saves or sends the email, the recorded patterns are saved. Full details of RT SquirrelMail, including the recording format, can be found in Appendices A and B, which respectively contain the details required for ethical consent and the instructions given to participants.

Because a real email system is used, the entry of confidential information cannot be prevented. However, controls were available to the participants to enable them to remove any private emails. The emails were only saved when the check box "If checked, all of your typing will be recorded for this email" was selected. If the participant forgot to uncheck the box, the email could still be removed from the dataset later. For the duration of the experiments, the participants were able to access a website, known here as mailAnalysis, that provided statistics on their own emails. MailAnalysis had two purposes: to encourage participants to use RT SquirrelMail over the standard SquirrelMail by providing interesting statistics, and to allow them to remove their own confidential emails from the dataset before evaluation began. Further information about mailAnalysis can be found in Appendix B.

Additionally, each participant filled in a questionnaire that included general questions relating to physiological attributes such as height, weight and handedness, and also information about their computing habits. This questionnaire provided information that was intended to be used to find out whether it is possible to identify key attributes of a typist—not just their identity—from their typing behaviour. However, there was an insufficient number of participants to be able to form any generalisations from this data and the responses are not used in any analysis in this thesis. For completeness, an example questionnaire is included in Appendix C.

Overall, 2897 emails were recorded from the 19 participants over the three month period. Although the most prolific participant recorded 1951 emails in the RT system, the majority of participants recorded less than 100 emails each. Of the 19 participants, 11 participated for the entire duration and eight were involved for the last two weeks. All participants, despite assurances, were understandably hesitant to participate. To ensure that the dataset contained a large number of users, some users (specifically, the last eight) were asked

to use the system for a limited time, and to use it only to type emails to the researcher. These eight participants therefore typed a smaller number of emails than the other 11 users and their email content was similar to the emails in Gunetti and Picardi's dataset [33]. Table 4.2 shows the number of emails recorded by each user and includes the number of emails that were finally used after technical and ethical issues were addressed. The next two sections detail these issues.

4.1.2 Technical Issues

Technical issues are rarely present in an artificial scenario: careful programming and supervision ensures that major issues are avoided. When collecting real emails however, supervision is impractical and even the most careful programming cannot compensate for users failing to follow instructions. For example, users were instructed to use the Opera browser if they were accessing their emails on a machine running Linux, and although Opera was installed on all relevant computers, many users forgot to use it and accessed RT SquirrelMail using Mozilla Firefox instead.

The occasional use of Mozilla Firefox on Linux-based machines presented the most difficult technical challenge, because the version of Firefox available for use during the data collection period contained a bug that caused certain key down events to fail to register [59]. Specifically, if a key was pressed and held down then any subsequent key presses would not be recorded until the held key was released. It is important to note that the missing key press events are actually lost, not delayed. Key releases are recorded as normal, so the end result is that there is the possibility of key release events recorded that have no associated key press event. This poses a huge problem for a typist verification system: characters appear on the screen as the key is pressed, so it is impossible to accurately calculate such features as the digraph time, or the duration of time a particular key is held down. Even attempting to reconstruct the input causes issues, as can be seen in the two example emails below. The first email shows the actual recording, and the second the actual input. Overall, 1217 of the 2897 emails were affected by this issue and were discarded from the dataset.

> Dear all,
>
> Jus watd to le yo kno ha the tying syste is nw upand running and yo can rcord yor tying by usin the special SquirrlMal.

> Chees,
>
> Dear all,
>
> Just wanted to let you know that the typing system is now up and running and you can record your typing by using the special SquirrelMail.
>
> Cheers,

Unfortunately the Firefox bug was not the only browser-related issue. It was discovered that Internet browsers are not uniform in the way they treat modifier keys. Modifier keys are the keyboard keys that modify the normal action of a key in some way: firing off a macro, capitalising a letter, providing access to alternate characters and so on. Some browsers treat modifier keys as independent key presses, recording the press and release events with their own key code. Others repeat the press event for a modifier until the key is released—alphanumeric keys will repeat in a similar fashion if held down long enough, but typically have matching releases for each key press. At the very least, all tested browsers (Internet Explorer, Firefox, Safari, Opera and Konqueror) record the current state of all modifier keys in the same structure that stores the keycode for each key event.

Ideally the emails should have a uniform format for ease of processing, but the modifier key event information differs between each browser. This information could be used later to reveal typing patterns. Given that the most common browser used during the recording process, Opera, only recorded the current state of the modifiers and all browsers had at least this behaviour, this was all that was recorded in RT SquirrelMail. Although some channels of information are lost by not recording modifier keys as individual events, no recorded samples have any more (or less) information recorded as a result of the browser that was used to access the system. In summary, the recordings are browser independent.

Finally, the use of a real email system over an artificial one presents a unique technical challenge: the length of any sample is not guaranteed to meet a minimum threshold. For example, if a user simply forwards an email, they only need to type the recipient's address. If the user replies to an email, they do not need to type anything—obviously it would be more helpful to the recipient if they did, but no new content needs to be provided in order to send the email. Compounding the problem further, shortcut keys, toolbar buttons and the mouse can be used to copy text from outside the browser window.

Therefore, legitimate emails can range in length from a few input events, to thousands.

It is unfair to impose a minimum limit on the emails before they are sent, especially since a lack of typing is often justified. Instead, after the recording phase ended, emails that did not contain at least 500 events were removed from the dataset. The minimum length was an arbitrarily chosen amount, corresponding to approximately two sentences of typed text (500 events equals 250 characters, or approximately 50 words). This size restriction guaranteed that the samples from RT SquirrelMail were larger than the smallest of the samples from the datasets in [33, 50], even after potentially sensitive data was removed. Of the remaining 1680 emails, 690 were removed as a result of this size restriction.

4.1.3 Ethical Issues

After technical issues had been addressed, only 990 emails remained in the dataset. These 990 emails were sourced from only 17 of the 19 participants—two participants had all of their samples removed from the dataset due to the technical issues described in the previous section. However, ethical issues must also be addressed before the dataset is used; the dataset cannot be distributed to anyone attempting to reproduce the results if it contains sensitive information. The ethical issues discussed here are typically not present in artificial scenarios.

The most obvious ethical issue is that people inevitably use email for personal correspondence, not just work-related discussions. RT SquirrelMail and mailAnalysis provided controls so that participants could remove confidential emails from the dataset. However, even with these measures in place there were a number of emails containing private information, and one even contained a password. To prevent such emails being present in the final dataset, a filter was run to remove any emails that might contain sensitive information. "Love" is an informal salutation and is unlikely to be used on a business-related email, so all emails that contained this word were removed. "Password" is an example of another filter word used to remove emails. After looking at some emails by hand, some other key words and names were discovered that were then used to remove emails from the dataset.

After some emails were manually checked, another issue became apparent: many of the participants were not native English speakers and often conducted email conversations in other languages. Whilst the presence of non-English

emails is not necessarily a bad thing, the dataset was collected by someone who only knew English and it was unclear whether any of the non-English emails contained sensitive information. Fortunately, only a small number of emails were not typed in English—instead typed in German, Swedish or Maori—and so these were simply removed from the dataset. Although these emails could have simply been referred back to the authors for double-checking, the relevant participants were keen users of the system and provided a sufficiently large number of English emails that this extra step was deemed unnecessary. Emails typed by the last eight participants were not subjected to the language filter; these last eight participants were typing to the researcher and despite emails being recorded in both Swedish and Maori—as well as English—they were guaranteed to be non-confidential so it was unnecessary to remove them.

At first blush the remaining emails may now appear to be anonymous. However, people tend to sign off their emails with their name and start by addressing the person they are sending to, so these names had to be manually removed from the data. The names could not be automatically removed by simply removing a number of characters from the start and end of the emails—some users were found to address a person or sign their name midway through authoring the email. To fully anonymise an email, the salutations and signatures were selected by hand and all events and times relating to the names were removed. The removed section is replaced with a single time differential that is the sum of all the time differentials in the removed section. Adding large time differentials should not prove to be an issue because the system should be robust to distractions. In Nisenson's system [50], time differentials greater than two seconds long were removed before learning. In Gunetti and Picardi's system [33], large time differentials were included into the average for each digraph. This last anonymisation step was only performed on a specific set of emails, as discussed in the next section.

4.2 Final Dataset

After ethical and technical issues were addressed, 607 emails remained. These emails had several assured qualities: they contained at least 500 events, were typed with a non-buggy browser, and contained no personal content. This dataset is comparable in size to Gunetti and Picardi's [33], which in the original paper contained 765 samples (their dataset used in the previous chapter contained only 480 samples). Gunetti and Picardi's [33] dataset contained data from a larger group of people, but at most 15 emails from any given

Participant	Total Emails	Issues Causing Removal			Final Emails (`sm-all`)	Final Emails (`sm-150`)
		Browser	Length	Ethical		
A	224		96	2	126	15
B	1951	1192	308	284	167	15
C	55		26	1	28	15
D	133		33	2	98	15
E	30		13	1	16	15
F	60		24	4	32	15
G	164		84	20	60	15
H	143		56	69	18	15
I	49	1	24		24	15
J	33		13		20	15
K	9		9		0	
L	9	7			2	
M	2				2	
N	4				4	
O	21	15	3		3	
P	4				4	
Q	1				1	
R	3		1		2	
S	2	2			0	
Total	2897	1217	690	383	607	150

Table 4.1: Emails recorded per participant

participant. In contrast, this dataset contains between one and 167 samples per user, with ten users typing 16 or more samples each. Table 4.2 shows the number of emails recorded, removed for technical and ethical reasons, and the final number of emails per user.

These 607 emails were separated into two datasets, the `sm-all` dataset that contains all 607 unaltered emails, and the `sm-150` dataset that contains 150 fully anonymised emails. The `sm-150` dataset contains the body text of 15 samples from each of ten users (Participants A–J), with salutations and signatures manually removed as described in Section 4.1.3. The `sm-150` dataset was created for two reasons: to ensure that no user had any more samples for training than any other, and so there was a fully anonymised dataset available for other researchers to use.[1] The `sm-all` dataset contains all possible information recorded for each email, but the email content may reveal the identify of the author or intended recipient.

The `sm-all` dataset is essentially a set of samples collected in a real system. As mentioned in Appendix A, each textbox on the email compose form was separately recorded, and mouse events occurring on this page were also recorded. At this stage only key events are of concern, so the mouse events are not included in the `sm-all` dataset. The separate textbox recordings are interleaved together using the absolute times, forming one sample of typing per email. Thus, a single sample includes all the typing for an email, in exactly the order the participant typed it.

[1]This dataset is available by request.

4.3 Re-evaluation of Existing Algorithms

In the previous chapter the two re-implemented techniques were compared by utilising the other technique's dataset. In both comparisons, the tests were biased. Even when the same dataset was used the algorithms were trained on data that was not suitable for them. Now there are two datasets, `sm-all` and `sm-150`, that address the shortcomings of the other datasets and can be used to fairly compare the two re-implemented techniques from Chapter 3. In this section the re-implemented methods are re-evaluated using the two new datasets.

Each of the two algorithms—namely LTIAT and KAOFT—was tested using fully optimised models and ten-times stratified ten-fold cross-validation. In the last chapter the best choice of Q, m and s for LTIAT was selected for each user. This results in a large number of models built per user. Since both new datasets contain significantly more data than was originally used, the LTIAT technique was tested using default parameters for this evaluation ($Q = 100$, $m = 0$, $s = 0$, i.e. the LZ78 algorithm with at most 100 time symbols). The m and s extensions should be unnecessary since they are designed to make the most of a small amount of data—and there is now a large amount of data. However, the main reason for testing in this way is that performing ten-times ten-fold cross-validation on optimised models would take an excessive amount of processing time for this algorithm (more than a month of processing time on a 2.4GHz machine). KAOFT was tested using the measure R_2A_2 because this was found to give the best results in the previous chapter.

Table 4.3 shows the results of evaluating the methods from the previous chapter with the two new datasets. These results reinforce the findings of the previous chapter: KAOFT is the most effective typist verification technique investigated so far. Note that `sm-all` has four times the volume of data and Table 4.3 shows there is an improvement when more data is added for either method. For the `sm-all` dataset, the LTIAT algorithm with default parameters outperforms the optimised algorithm on its own dataset (which reported an AUC of 0.962). For both datasets, KAOFT performs better than with its own data from the previous chapter, obtaining higher AUC values and lower FRR/IPR values as well. These datasets reinforce the findings of the previous chapter, but also indicate they themselves are of equivalent—if not better—quality to the original datasets.

It is important to remember that KAOFT requires data from other users to make a prediction about a target user, whereas LTIAT does not. Also, it is

Original datasets	FRR (%)	IPR (%)	Weighted AUC
LTIAT - optimised parameters	3.16	12.92	0.962
KAOFT - R_2A_2	6.35	0.39	0.967
`sm-150`			
LTIAT - $m = 0$, $s = 0$, $Q = 100$	3.53	29.33	0.950
KAOFT - R_2A_2	6.06	0.00	0.970
`sm-all`			
LTIAT - $m = 0$, $s = 0$, $Q = 100$	0.75	16.65	0.976
KAOFT - R_2A_2	3.60	0.00	0.982

Table 4.2: Re-evaluation results using new datasets

unclear just how much data is necessary for successful verification and whether other possible channels of information may be more informative than those used for the state of the art. The next few chapters attempt to answer these questions, using insights gleaned from the replication in the previous chapter, studies of related work, and thorough investigation of the new datasets.

Chapter 5
Preliminary Experiments with New Techniques

In Chapter 3, where the two leading continuous typist verification techniques and their results were replicated, a limiting factor for evaluation was the lack of a dataset that had sufficient information for it to be utilised for both techniques. The collection of new datasets, discussed in the previous chapter, resolved this issue and enabled the two techniques to be fairly tested. Using these same datasets from Chapter 4, some initial experiments were performed to investigate new ways to verify a typist.

This chapter presents this investigation by discussing new ways to verify typists using probability-based algorithms. Three of the algorithms can make judgements after being trained only on data from the target user. The other requires data from other users, but achieves good accuracy on individual digraphs. The FRR and IPR of all four algorithms can be varied at prediction time by changing the threshold that determines the yes/no verdict.

This trade-off allows the security of the system to be quickly adjusted to best suit the task at hand. Systems that simply predict yes or no usually require complete retraining in order to alter the trade-off between the FRR and IPR. Also, because an arbitrary threshold set automatically by an algorithm is not necessarily the best one, being able to rank numerical predictions allows a true assessment on the effectiveness of a typist verification system using the AUC.

The next section describes the experimental methodology that was used to evaluate each method. Each of the four methods is then discussed in turn, including explanations of the algorithms and the results obtained using the datasets from the previous chapter. The first method is similar to LTIAT from Chapter 3 except that it uses Prediction by Partial Matching (PPM) instead of LZms for the underlying model. The second constructs a probability density for each digraph, and makes predictions using the combined densities for all digraphs in a sample. The third combines ideas from the first two

methods in this chapter, integrating context with per-digraph models. The fourth classifies individual digraphs using C4.5 decision trees using absolute and relative timings as attributes. In Section 5.6 the methods are compared to each other and the algorithms from Chapter 3.

5.1 Methodology

Each algorithm was implemented in the Java programming language and integrated into the workbench described in Chapter 3. Although the workbench is capable of performing many different kinds of evaluation, all the results presented here use ten-times ten-fold stratified cross-validation. The same seed was used to initialise the pseudo-random number generator each time, ensuring that the algorithms were tested with exactly the same set of folds. This same seed was used for the evaluation at the end of Chapter 4.

Each fold contains 10% of the data that belongs to each user and attacker. Three of the algorithms in this chapter ignore any negative data that is present during training, ensuring that attackers are always new to the system. However, the last algorithm, the Individual Digraph Classifier, requires negative data in order to make a judgement. Sourcing negative data from the `sm` datasets means that the attackers are not novel, so in this case all negative data was deleted from the training folds before training began. This forced the algorithm to use negative data from elsewhere. In reality, this data could be automatically generated, or the real negative data could be used. In this thesis, the negative data was sourced from the LTIAT dataset, and each user or attacker was relabelled as if it belonged to a single impostor. This means that all algorithms in this chapter were evaluated after being trained on a single user's data from either the `sm-150` or `sm-all` datasets and attackers were always novel.

All algorithms utilise a threshold to separate a given user from any impostors. This threshold can be based on a probability, however, not all the methods in this chapter are able to produce these. This is not a problem for typist verification because the numerical predictions produced by the algorithms in this chapter are able to be sensibly ranked.

To determine the threshold for each model, each of the user's training samples is held out in turn during training, and a model built that includes all samples except the held out sample. The held out sample is tested against the model and the prediction recorded. After testing all possible held out samples, the threshold is set at the value that maximises the chance that the hardest

held out sample will pass as the user. That is, the threshold is the smallest predicted value calculated from the leave-one-out training. The final model for any fold is built using all of the data, and utilises the threshold calculated during the leave-one-out training. This is exactly the same method as LTIAT from Chapter 3 uses to determine a threshold. It is used for all but the last model presented in this chapter.

Because the thresholds set by the system—and therefore the FRR and IPR—are not necessarily ideal for a particular scenario, in all empirical results the weighted average AUC of the system is reported. As discussed in earlier chapters, the AUC gives the probability that a user's sample will rank higher than an attacker's one, and is independent of the arbitrary thresholds. The weighted average AUC is used so that users with varying amounts of data, such as those found in the `sm-all` dataset, do not adversely affect the results. The FRR and IPR are also reported for all algorithms.

5.2 PPM-Based Classifier

Prediction by Partial Matching (PPM) is a popular compression technique, but it can also function as a universal sequence prediction algorithm [6]. It is the latter application that is the most helpful for typist verification. PPM comes from the same class of prediction/compression algorithms as LZ78, the underlying prediction technique (with modifications) used for LTIAT in Chapter 3. In both cases the algorithms predict the likelihood of the next symbol, which is either a key event or time differential, depending on the current state of the system. The main difference between the two algorithms when they are used for prediction is that PPM uses a finite context model, basing its decisions on at most o previous events for some fixed number o, called the context length. In contrast, LZ78/LZms uses as many previous events as possible, until an event fails to match in the current context and the algorithm resets to the root of its tree.

The standard version of PPM deals with escape probabilities by using the probability of a novel character occurring [6]. Escape probabilities are the values used when the next character has not been seen before in the current context. For PPM, the method used to calculate escape probabilities was "Method C", which is generally regarded to be the best method for this classifier. Using Method C, the probability of an unseen character is based on the number of novel characters seen before in the current context. More precisely, the probability of an unseen character is $q/(C + q)$ and the probability of a

seen character is c_i/C, where q equals the number of distinct characters seen before, c_i the number of times a character i has been seen before, and C the total number of characters seen [6]. LZ78/LZms predicts $1/(C+1)$ for unseen symbols, and $c_i/(C+1)$ for all others.

5.2.1 Algorithm

PPM can be used for typist verification by utilising the LTIAT technique and simply substituting LZms with PPM. This new algorithm has two parameters: the number of quantization clusters q, and the order o of the PPM model. The process is otherwise the same as that of LTIAT. Given a number of training samples, the time differentials are first clustered using k-means and then the order-o PPM model is built using the key events and quantized times. A threshold is obtained by repeatedly building leave-one-out models during training, for all possible held out training samples, and using the probability that maximises the chance that the hardest held out training sentence will be classified correctly. Finally, the probability of a sample belonging to the user is calculated by predicting each symbol in a sample, and accumulating the log likelihood. This is translated into a decision by utilising the threshold obtained during training.

The PPM-based version of LTIAT was implemented in the Java programming language and incorporated into the workbench from Chapter 3. However, because the PPM algorithm was not modified in any way, an existing open-source implementation of PPM was utilised rather than a re-implementation. The PPM implementation used here was originally coded by Ron Begleiter for general evaluation of variable order Markov models [5]. The original LTIAT technique used a modified version of LZ78, and the entire LZms algorithm was implemented from scratch.

In general, LZ78 models are much faster to build and evaluate than PPM ones, but are less accurate. In the implementation here, PPM can build its model faster than LZms, because the two extensions—input shifting and back-shift parsing—increase the number of times each symbol is seen, and so build a much larger model than LZ78 would. On average, PPM ($o = 1$) takes one-fifth the time to build a model compared to LZms ($m = 5$, $s = 5$), and one-tenth the time to make a classification. Table 5.7 in Section 5.6 shows the average build and classification times for LZ78, LZms and PPM, along with other methods discussed in this chapter.

Order o	FRR	IPR	Weighted AUC
0	3.00	42.56	0.862
1	4.33	25.22	0.966
2	4.00	25.83	0.961
3	4.00	25.76	0.960
4	4.20	25.76	0.961
5	4.07	25.93	0.961
6	4.07	25.93	0.961
7	4.20	25.96	0.961
8	4.07	25.65	0.961

Table 5.1: Effect of PPM model order on prediction

Algorithm/Dataset	FRR	IPR	Weighted AUC
PPM / LTIAT	1.85	14.55	0.967
PPM / `sm-150`	4.33	25.22	0.966
PPM / `sm-all`	0.90	18.99	0.971
LZms / LTIAT	3.16	12.92	0.962
LZ78 / `sm-150`	3.53	29.33	0.950
LZ78 / `sm-all`	0.75	16.65	0.976

Table 5.2: Final results for PPM using order = 1

5.2.2 Results

The PPM-based classifier was evaluated using both the `sm-150` and `sm-all` datasets, using the methodology described in Section 5.1. In order to perform a fair comparison to LTIAT's evaluation in Chapter 4, the parameter q was set to 100. This parameter was optimised on a per-user basis in the original LTIAT implementation, but the LZms model took a substantial length of time to produce an optimised model for the `sm` datasets. Because of this, optimization was abandoned in favour of fixed parameters.

Optimisation on the order parameter, o, was attempted globally, with different values of o being tested for the `sm-150` dataset. The results of this evaluation is presented in Table 5.1. There was no substantial difference between the models after reaching $o = 1$, therefore this value was used globally for all users when using the `sm-all` dataset for evaluation. When the context length is one, the PPM algorithm uses at most one symbol for the prior context—the context for a time is a key event and vice versa. It is unusual that performance should peak with such a context length, and would usually indicate that something is wrong with the implementation. However, this effect might be caused by the interleaved symbol types; the stream alternates between key events and quantized times and it is possible that this forms a sequence where events that affect each other are out of order. For example, a time symbol may be affected by three preceding key events, but not the other time symbols that occur in between.

Table 5.2 presents the results for evaluating PPM and LZms/LZ78 with the LTIAT, `sm-150` and `sm-all` datasets. Each dataset was divided up using the

same cross-validation folds for each algorithm. Surprisingly, PPM performs no better than LZ78 for the `sm-all` dataset, despite PPM being generally regarded as a better classifier and more efficient encoder [6]. However, on the smaller `sm-150` and LTIAT datasets, PPM slightly outperforms LZ78.

5.3 Spread Classifier

Instead of considering the prior context for a particular digraph, the digraphs can be considered independent and each one evaluated separately. A naïve implementation would average the time for each digraph, and use this for prediction. Each feature is one digraph, and contains either the average time, or a value that denotes that this digraph is "missing" in the sample. One problem with this kind of approach is that it is impossible to control what is typed in a continuous scenario, so there is potentially a large number of missing digraphs for any given sample. In such a situation, the classifier may begin to make predictions based on what is missing, rather than any features that are present in a given sample.

The KAOFT technique discussed in Chapter 3 overcomes this issue by comparing two samples on only their common digraphs. Typical classifiers using subsets of features require retraining with each subset each time a prediction is required. However, retraining can be avoided by building a model for each digraph seen in the training data, but only using each model when the sample requiring verification contains that digraph. In the algorithm evaluated in this section, the spread of times for each digraph is used to form a per-digraph model. The predictions of each model are combined to make an overall prediction whether or not a given sample belongs to the user.

5.3.1 Algorithm

This algorithm, called the "Spread Classifier", utilises the spread of time values for each digraph in order to verify an identity. A model is built for each digraph seen during training, but only if it occurs at least x times, where x is a parameter. The algorithm has four other parameters: whether to remove time outliers, the value o above which times are considered outliers, the type of model to build, and the bandwidth b of spread. In this implementation, three different types of models were available:

- **Fixed Bandwidth** In this model digraph times are binned into discrete ranges during training. For prediction, the probability a time t belongs

to that user's digraph is the total number of training digraphs in the same bin, divided by the total number of digraphs D plus one. If no digraphs are in the same bin, the predicted probability is $1/(D+1)$. In this model, b is the size of the bin.

- **Gaussian** The Gaussian model fits a single Gaussian distribution to each digraph, based on the mean and standard deviation of the training times. A prediction for a time t is the value of the continuous probability density function for t. For this model, b is ignored and instead the standard deviation is used.

- **Mixture of Gaussians** This model is similar to the Gaussian model, but instead fits a mixture of Gaussians using the Expectation-Maximisation algorithm during training. The predicted value is the combined densities of the fitted Gaussian models. Again, b is ignored and the standard deviation is used to determine the spread.

In the case of the first model, Fixed Bandwidth, the value for b could be set automatically to the standard deviation of that particular digraph. Fixed values of b were applied globally to all digraphs, but the standard deviations were set per-digraph. In practice, fixed values of b performed worse than the per-digraph standard deviation, so in all results in the next section the standard deviation is used for the bandwidth.

The four parameters affected all digraphs; the parameters were not optimized on a per-digraph basis unless the bandwidth parameter utilised the standard deviation instead of a fixed value. Furthermore, no per-user optimization was employed—the parameters were identical for every user the system evaluated. Although per-user parametrization has been shown to improve results for other typist verification techniques [38], the results are not substantially different from those using global parameters, and so the lack of per-user optimization is not considered an impediment here. Also, all methods in this thesis, with the exception of LTIAT in Chapter 3, perform no per-user optimization.

A prediction on a sample to be verified is performed by considering only the models that both the trained system and the given sample have in common. Each digraph time is provided to the relevant model, and the probability/density is obtained. The densities are then accumulated together (using their log likelihood) and divided by the total number of digraphs evaluated. Thus, the length of a sample and/or the number of digraphs it has in common

Model	Minimum Digraphs x	FRR	IPR	Weighted AUC
`sm-150`				
Fixed Bandwidth	1	7.67	93.64	0.235
Gaussian	3	5.13	75.17	0.704
Mixture of Gaussians	3			0.855
`sm-all`				
Fixed Bandwidth	1			0.500
Gaussian	3			0.757
Mixture of Gaussians	3			0.923

Table 5.3: Final results for the Spread Classifier

with the training data should not affect the overall prediction. A threshold is obtained during training using the leave-one-out methodology adopted by the other techniques in this chapter. During prediction, any values falling below the threshold fail verification, those above the threshold pass.

5.3.2 Results

Table 5.3 presents the results for each of the `sm` datasets, and each type of model. In all cases the standard deviation was used as the bandwidth parameter b, and digraph times above 2000ms (2 seconds) were removed. When presented with the `sm-all` dataset the implementation was overwhelmed and took an excessive amount of time[1] to complete evaluation, so threshold optimization was abandoned and only the weighted AUC is presented for this dataset. This is also the case for the Mixture of Gaussians model for both datasets—the computation involved in optimizing the threshold took more than a week of computing time. For both `sm` datasets and the Mixture of Gaussians model, only the weighted AUC is presented.

It is possible to read off FRR and IPR values from the ROC curve for each user, but this would mean that the thresholds were obtained in a different manner—so it is difficult to make a fair comparison of the various models. However, the weighted AUC is available for all of the models and is independent of any thresholds, so this value forms the basis of all comparisons.

Not surprisingly, the most complicated per-digraph model—a mixture of Gaussians—performs the best on all datasets. Unfortunately this particular model takes a long time to process the `sm` datasets, making it unhelpful for typist verification because decisions cannot be made quickly enough to refuse an attacker access to the system in real time. The faster Fixed Bandwidth and Gaussian models perform poorly compared to other techniques such as LTIAT and KAOFT, and build and classification time increases to an unmanageable level when using larger datasets such as the `sm-all` one. Also, all AUC values

[1] More than one week of processing on a single 2.4GHz machine.

are worse than those previously observed for other techniques. Simply put, the per-digraph model representations of a typist are often large to store and time consuming to process, and do not challenge the current state of the art in any way.

5.4 Context Classifier

In Section 5.2.2 it was found that PPM worked best when the context was limited to a context length of one. The last section, 5.3, used more complicated metrics than PPM in order to verify identities, opting for a density-based approach. These two techniques can be combined, to form a single context-sensitive density-based classifier. This classifier, called the "Context Classifier", requires that a context matches before making a prediction using a digraph's one-class model. As a result of this requirement, a single digraph may have many different models—one for each context where the digraph appears. This is different to using trigraphs or larger n-graphs where the overall time of the entire n-graph is considered; only the digraph time is ever used.

5.4.1 Algorithm

The Context Classifier was implemented into the workbench from Chapter 3. Like the Spread Classifier in the last section, it removes outlier times above a given level, o. It has two other parameters, namely the prior context length and post context length for a character. If the post context length is set to one, the model is built on digraphs and is identical to the Spread Classifier when it uses a Gaussian model, except that no limitations are made on the number of observations before the system will attempt to use a model.

The prior context length determines how many characters are considered before the character in question and the post context determines the number of characters afterwards. A value of zero for the post-context length parameter means the second character in the digraph is anonymous. For example, considering the two sequences `the` and `thy` with a prior context length of one and a post context length of zero, the times for the digraphs `he` and `hy` would be incorporated into the model for `th` (where `t` is the prior context). When the post context length is increased to one, the times for `he` and `hy` are put into separate models `the` and `thy`. When the second character is anonymous the time value is still a digraph time—not a duration—because the only key press events are considered.

One Gaussian distribution is fitted to each {`prior context, character, post context`} triple, using the digraph times for the triple to build the Gaussian distribution. A sample is verified by considering triples that appear in both the training data and the given sample, and using their relevant distributions to produce densities that are combined in the same manner as in the Spread Classifier discussed previously.

This implementation of the Context Classifier uses only key press events, but it is possible to create a similar classifier using releases as well. Such a classifier would be similar to PPM in Section 5.2, but utilise a Gaussian distribution to make predictions instead. After considering the relevant performances of the classifiers in this section, evaluation of such a classifier was not conducted in favour of investigating other methods of typist verification.

5.4.2 Results

The results of ten-times ten-fold stratified cross-validation for different context lengths, up to a maximum of four characters, is presented in Table 5.4 using the `sm-150` dataset. In Table 5.4, c is used to represent context characters surrounding the main character C. When the context is at least two characters long, regardless of whether the characters are in the prior or post context, the system scores an AUC below 0.5. This suggests that a given key is affected by only one other key because context lengths smaller than two perform substantially better than others when used for verification. Gentner [30] asserted that "the inter-stroke interval for typing a given character is influenced by the neighbouring two characters to the left and one character to the right." Gentner's work was an investigation of keystroke timing in transcription typing, of which character context is one aspect. The analysis here does not support this assertion, however, key releases may influence timings but were not used when considering context in this classifier.

The results from Table 5.4 indicate that a post context of one and a prior context of zero—that is, digraphs (as used in the Spread Classifier)—is the most effective for verification, when using the AUC for comparison. This context was also tested using the `sm-all` dataset, which produced worse results. Unfortunately none of the results are good regardless of the dataset used. The AUC needs to be much greater in order to perform verification that would be practically useful. The Context Classifier and the related Spread Classifier are not good challengers to the current state of the art in typist verification.

Prior Context	Post Context	Context	FRR	IPR	Weighted AUC
0	0	C	7.47	89.54	0.619
0	1	Cc	8.07	97.40	0.644
1	0	cC	7.93	94.81	0.623
1	1	cCc	2.46	95.44	0.468
2	0	ccC	3.66	96.31	0.498
0	2	Ccc	4.33	95.04	0.489
2	1	ccCc	0.67	98.82	0.448
1	2	cCcc	0.80	96.79	0.429
2	2	ccCcc	0.20	99.76	0.413
0	3	Cccc	0.60	97.89	0.452
3	0	cccC	0.00	99.93	0.448
3	1	cccCc	0.07	99.93	0.423
1	3	cCccc	0.27	99.81	0.441

Table 5.4: Effect of different context lengths on the Context Classifier

Dataset	Prior Context	Post Context	FRR	IPR	Weighted AUC
`sm-150`	0	1	8.07	97.40	0.644
`sm-all`	0	1	1.14	99.07	0.622

Table 5.5: Final results for the Context Classifier

5.5 Individual Digraph Classifier

All classifiers introduced so far in this section are one-class classifiers: they require no data for training other than that belonging to the user being verified. In general, requiring negative data is a drawback because the system performance becomes dependent on it; poor quality data or low quantities of negative data will damage performance. On the other hand, the performance of one-class classifiers depends on only the target user's data, and their threshold can be adjusted to obtain a different trade-off between the FRR and IPR. Up until this point, only one-class classifiers have been proposed in this chapter. However, there is negative data available—both from other users in the `sm` datasets, and other datasets from Chapter 3—so it is possible to consider systems that make use of this data.

The "Individual Digraph Classifier" is one method of using the negative data, but retains the ability to change the trade-off between the FRR and IPR at prediction time. This is achieved by recording a set of features for every digraph that appears in the training samples and training a standard multi-class classifier with this data. The multi-class classifier can predict either "user" or "attacker" for previously unseen digraphs. A sample is verified by forming a threshold on the proportion of digraphs in it that are classified as "user" by the multi-class classifier. This threshold can be adjusted at prediction time without retraining any part of the system.

5.5.1 Algorithm

The Individual Digraph Classifier utilises C4.5 decision trees as the multi-class classifier, trained with a set of ten features. The implementation of the C4.5 decision trees used here was provided by the WEKA Machine Learning Workbench [77]. Each instance is derived from a single digraph in a sample with the features that were used to train the classifier being:

- **First Character** The first character in the digraph.
- **Second Character** The second character in the digraph.
- **Absolute Digraph Time** The digraph time, measured in milliseconds.
- **Absolute First Duration** The duration of time, measured in milliseconds, that the first key was held down.
- **Absolute Second Duration** The duration of time, measured in milliseconds, that the second key was held down.
- **Absolute Inter-Keystroke Time** The time in milliseconds between the release of the first key and the press of the second. This may be negative if the first key was still depressed when the second one was pressed.
- **Relative Digraph Time** The absolute digraph time divided by the WPM speed of the entire sample.
- **Relative First Duration** The absolute first duration divided by the WPM speed of the entire sample.
- **Relative Second Duration** The absolute second duration divided by the WPM speed of the entire sample.
- **Relative Inter-Keystroke Time** The absolute inter-keystroke time divided by the WPM speed of the entire sample.

Both absolute and relative times were used because either on their own had little discriminatory power; the learner was unable to perform better than predicting the majority class when trained with only one type of timing.

Just like LTIAT and the PPM Classifier, the C4.5 decision tree makes predictions on individual digraphs rather than samples. To translate this into a judgement for a sample, a threshold is set on the proportion of digraphs

Dataset	Threshold	FRR	IPR	Weighted AUC
`sm-150`	0.5	4.33	35.41	0.917
`sm-all`	0.5	0.70	46.48	0.927

Table 5.6: Final results for the Individual Digraph Classifier

that pass verification according to the decision tree. A digraph passes if it has a greater probability of belonging to the target user than an attacker. The attacker data is sourced from the LTIAT dataset in the evaluation below, ensuring that attackers are always novel to the system.

It is possible to set the threshold in the manner described in Section 5.1, however, finding the threshold in this manner takes an excessively long time for this classifier.[2] The classifier is trained on tens—often hundreds—of thousands of digraphs and takes a long time to build a model with so much data. Fortunately it is easy to set a sensible threshold on the proportion that achieves a similar FRR to other classifiers. If more than half of the digraphs pass verification, the entire sample passes too. The threshold is set by default at 0.5 for this classifier.

5.5.2 Results

The Individual Digraph Classifier was evaluated using ten-times ten-fold stratified cross-validation, in exactly the same manner as all other algorithms in this chapter. In both cases, the threshold was set to 0.5—no leave-one-out threshold optimization was performed. Table 5.6 presents the results of the evaluation.

This classifier did surprisingly well, considering that the Spread Classifier—which also evaluates individual digraphs—performs poorly. The Individual Digraph Classifier did not score a higher AUC than the state of the art, but came closer than both the Spread and Context Classifiers. The accuracy of the predictions could be improved by using bagging or boosting on the decision trees [55], but doing so would increase the time taken to build a model. Attempts at training bagged trees and boosted trees were abandoned because they did not complete in a reasonable amount of time.[3]

[2]More than a week of computing power on a single 2.4GHz machine to perform a full cross-validation evaluation on the `sm-150 dataset`

[3]It took more than 24 hours of processing time for evaluation of the `sm-150` dataset.

5.6 Performance Comparison

This chapter has presented four different methods of performing continuous typist verification, and evaluated their performance using the `sm` datasets. In terms of accuracy, the PPM classifier performed the best; it scored an AUC of 0.971 for the `sm-all` dataset. The PPM technique was based on LTIAT, changing only the machine learning component, and rivals the accuracy of LZ78—especially on small datasets. However, PPM is slower than LZ78 and the performance difference is not significant, so LTIAT using LZ78/LZms still trumps this technique.

In all analysis, accuracy has been a major factor in determining the effectiveness of the algorithms. Yet speed is also important. Ideally a system should be able to perform classifications in real time to ensure that attackers are quickly identified. Training time also needs to be considered because it would be advantageous to retrain the system over time, keeping the profiles up to date. Systems that take a long time to train may have problems when many users need to be verified—the training of profiles may overwhelm the machine being used.

Table 5.7 presents approximate timings for training and prediction for each of the systems from this chapter and Chapter 3, using the `sm-150` dataset. No threshold optimization was performed, the timings represent how long each algorithm takes to build a single model, and classify a single instance. The timings are rounded to the nearest five milliseconds and were calculated using ten-times ten-fold stratified cross-validation. Each technique is evaluated using its implementation in the Typist Workbench introduced in Chapter 3. Whilst none of the code was completely optimised each classifier was implemented in a similar code style and provides satisfactory performance, with the exception of PPM where a third-party implementation was used.

Of all the techniques, LTIAT using LZ78 is a clear winner on speed, but this should be taken with a grain of salt because the times were clustered before the system was trained. This means that the training time is artificially low, and in reality the Context Classifier is the fastest overall technique. Unfortunately, this classifier achieves an accuracy inferior to all other techniques, so the speed advantage is outweighed by its poor performance for verification.

All algorithms slow down—noticeably during training—when the larger `sm-all` dataset is used. The KAOFT technique and Individual Digraph Classifier both utilise negative data, and their training time is additionally influenced by the amount of negative data available. The results in Table 5.7 suggest that

Method	Training Time		Prediction Time	
	Average	Std Dev	Average	Std Dev
LTIAT - LZ78	50ms	40ms	10ms	10ms
LTIAT - LZms $m = 5$, $s = 5$	1000ms	500ms	150ms	20ms
PPM - $o = 1$	200ms	30ms	15ms	15ms
Spread - Fixed Bandwidth	50ms	10ms	100ms	50ms
Spread - Gaussian	50ms	10ms	100ms	50ms
Spread - Mixture of Gaussians	30000ms	200ms	10ms	10ms
Context - Prior/Post = 2	150ms	10ms	5ms	10ms
Individual Digraph	3000ms	200ms	25ms	20ms
KAOFT - R_2A_2	1300ms	100ms	100ms	40ms

Table 5.7: Average build and classification times

these two classifiers would be a poor choice for this dataset. Despite KAOFT scoring the highest accuracy it is also one of the slowest techniques.

5.7 Summary

This chapter has presented four different continuous typist verification techniques, the PPM Classifier, Spread Classifier, Context Classifier and Individual Digraph Classifier. Each classifier considers a different aspect of typing behaviour, from sequences to individual digraphs. None of these classifiers challenge the accuracy of LTIAT or KAOFT on the `sm` datasets, except for the PPM classifier, which slightly outperforms LTIAT on two of the datasets used. The classifiers have a varying speed performance on the `sm` datasets, but considering both accuracy and speed LTIAT is the best choice for typist verification seen thus far.

In order to challenge KAOFT's accuracy and LTIAT's speed and accuracy, a new approach is needed. To reduce the training and prediction times, smaller representations of typists are required. To achieve good accuracy, any approach must consider various aspects of a typist's behaviour—not just raw speed. The next chapter discusses typist behaviour, exploring what aspects of typing can be used to distinguish typists from one another. Chapter 7 introduces one-class classification and presents a general classifier that can outperform multi-class classifiers when attacking data is novel to the system. Chapter 8 draws on the insights from Chapters 6 and 7 to present a continuous typist verification technique that rivals all those explored thus far—considering both speed and accuracy.

Chapter 6
Typist Behaviour

There are many elements to consider in typing input, but ordinary typist verification techniques utilise only one: the time between key events. This is not necessarily a problem because methods that use time-based metrics, such as those in Chapters 3, 4 and 5, have been shown to achieve high accuracy. However, the performance comparison in the previous chapter showed that many of these techniques are slow; they utilize large numbers of features and because of this perform many comparisons before producing a judgement. In order to reduce the time taken, the dimensionality of typical typist verification systems needs to be addressed.

This chapter explores typist behaviour and considers what habits a typist has that might differ from another typist with a similar level of skill. The intention is to shed light on what features might be useful for verification, so a system can be created that utilises only these. Not all behaviours can be used to distinguish typists from one another—often typists perform a task in a similar manner. This isn't unhelpful: identifying uniform behaviours allows distinctive ones to be isolated, and it is these that are of interest for typist verification.

The digraph is one of the smallest units of data in a typing sample and one of the most commonly used structures for typist verification. Because of this it is the starting point for this chapter. First, different sorts of digraphs are considered in Section 6.1, including those that do not produce characters on the screen. Next, Section 6.2 abstracts from digraphs to finger movements, to examine whether there is any correlation between digraphs that are typed in the same way.

These features are still based on time, although several individual digraphs are combined when considering movements. Section 6.3 is the first analysis in this thesis of a non-time based metric, studying how often certain unprintable characters are used. Section 6.4 analyses pausing behaviour. Section 6.5 investigates whether typists may have different habits for releasing keys. Section 6.6 briefly discusses the potential of simple attributes such as typing speed

and error rate. All these channels of information can be exploited for typist verification.

6.1 The Digraph

In the English language, "the" is one of the most common words. Its component digraphs, `th` and `he`, are the most commonly typed alphabetical digraphs in the datasets collected in the previous chapter. However, neither are the most common digraph: that honour goes to a pair of backspaces, which is typed three times more often than any other digraph in the `sm-all` dataset. In fact, `th` is only the 6th most frequent digraph. The top ten frequent digraphs, in descending order, are: `Backspace Backspace`, `Right-Arrow Right-Arrow`, `e Space`, `Left-Arrow Left-Arrow`, `Space t`, `th`, `t Space`, `Space a`, `he`, and `Space Space`.

The top five digraphs are either pairs of unprintable characters, such as `Backspace`, or digraphs that include the `Space` character. Pairs of unprintable characters are not always helpful because they are often typed quickly, especially when the key is held down until it begins to repeat automatically. They can also be typed extremely slowly, such as when the user pauses to consider what they are editing. Figure 6.1 shows the spread of times for a pair of backspaces, rounding times to the nearest 20ms and grouping times that exceed 1000ms into the final bar. Although Figure 6.1 reveals a skewed normal distribution nestled next to a spike at 40ms where the backspace key is held down and the key repeat has engaged, and a long tail of values past 1000ms. One might naïvely assume that it would be safe to filter out times that are not within a given range—removing the spike and tail in the case of Figure 6.1. However, this is not the case. Applying a filter can have a negative effect: valid digraph times that discriminate one user from others may be discarded, and for some samples all occurrences of a digraph may be removed.

The negative effect can be demonstrated by considering a randomly selected sample for the digraph `th`, typed by Participant A from Chapter 4. Using a filter that removes times less than 40ms, three of the 40 values for Participant A's sample would be removed. All three values are perfectly legitimate: `t` and `h` are separate keys, and the typist is simply quick at typing them.

In general, it is unwise to use a filter that removes low times—especially for digraphs that are a pair of keys—because it is difficult to determine whether the digraph has been typed legitimately or is the result of held-down keys when considering a pair of keys. Filters that remove long times may be useful; it is

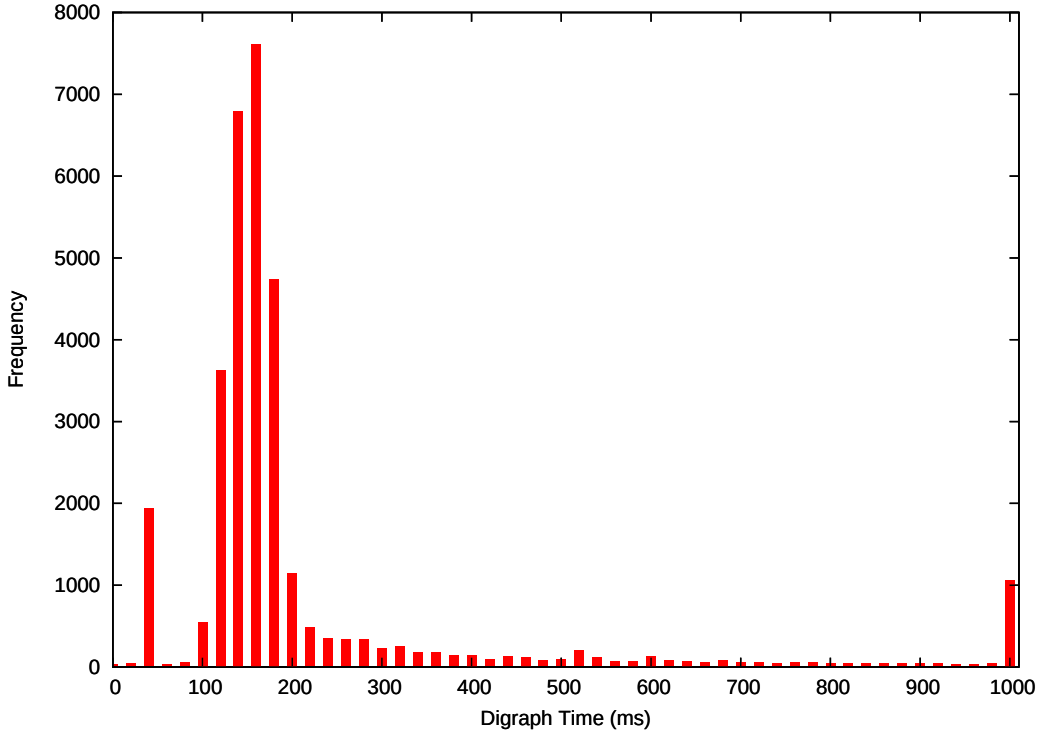

Figure 6.1: Distribution of times for `Backspace Backspace`

unlikely that a digraph that takes minutes to type occurs naturally in the flow of typing. The user is most likely distracted, typing with something in their hand, or absent from the terminal between press events. However, care must be taken to ensure that such a filter does not remove valid digraphs either. For example, 750ms seems too low—5% of all digraph times for `Backspace Backspace` would be removed using this value.

Ideally any filter should take into account the way that a digraph is typed, since a global filter will remove a different proportion of times for each. Some digraphs require the typist to move their hands across the keyboard, not just their fingers, and inevitably such digraphs will take a longer time to type. A more conservative filter could also be used, in the order of seconds rather than milliseconds. On the other hand, any well-designed learning scheme should be robust to outliers, so filtering out high digraph times should be unnecessary in order to perform verification.

Regardless of filtering, digraphs formed with a pair of identical letters, such as a pair of backspaces, are not always informative for discrimination between users. Participant A, the fastest typist averaging 70 WPM, has a strikingly similar spread for the digraph `Backspace Backspace` to Participant J, the slowest of the typists who averages only 39 WPM. Figure 6.2 displays the frequency of the digraph `Backspace Backspace` in a randomly selected sample

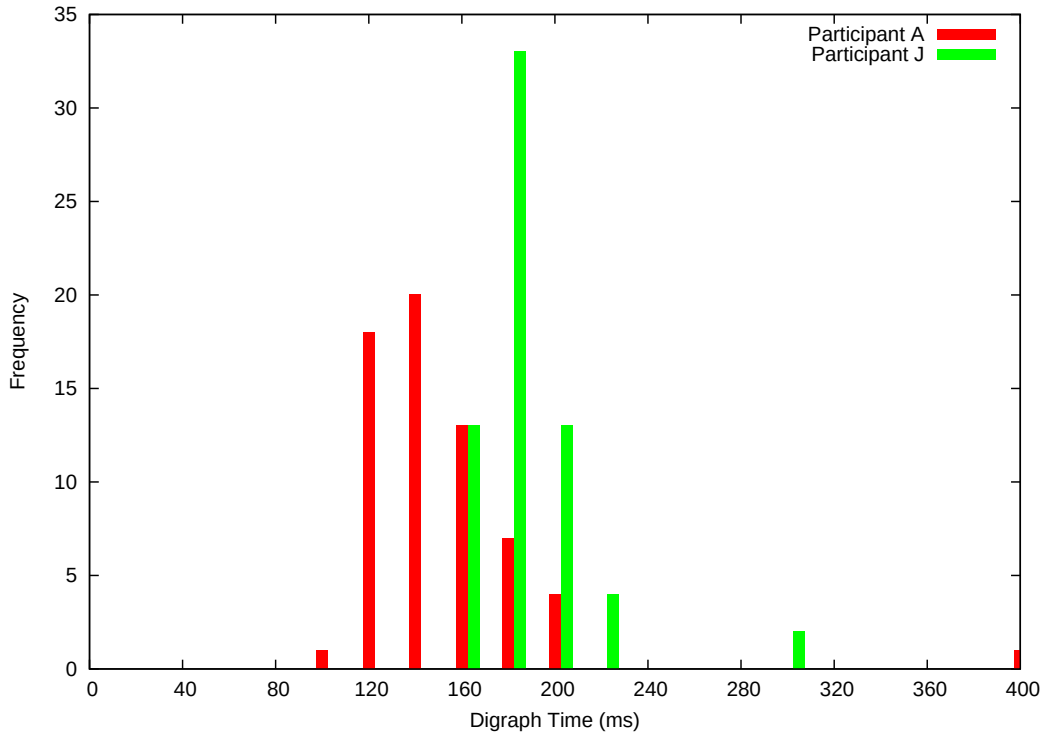

Figure 6.2: Times for `Backspace Backspace` for Participants A and J (one sample)

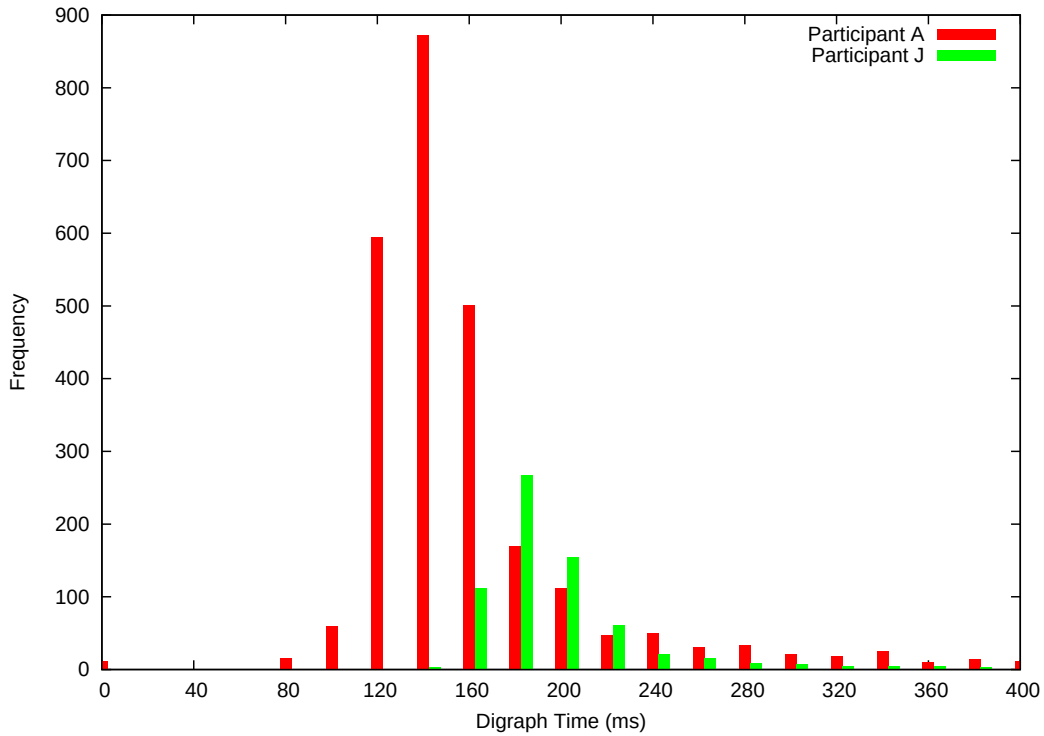

Figure 6.3: Times for `Backspace Backspace` for Participants A and J (all samples)

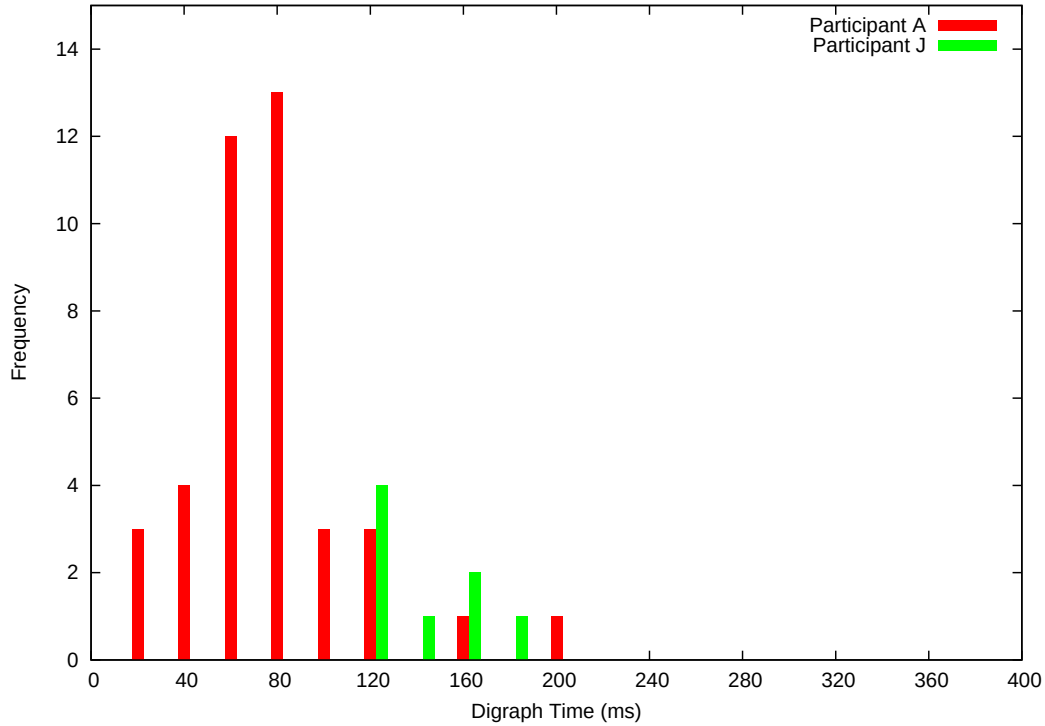

Figure 6.4: Times `th` for Participants A and J (one sample)

from each of the two users. Despite Participant A having an overall average speed almost twice as fast as Participant J, there is an average difference of only 35ms between the two samples. On average, Participant A had a digraph time of 200ms, Participant J had a time of 235ms. This is not just a result of unrepresentative sample selection: the same two users have the same overlap when considering the same digraph, `Backspace Backspace`, across all samples, as shown in Figure 6.3.

The digraph `th` does not suffer from the same issue as `Backspace Backspace` for these two users. Examining the same two users for `th`, the difference in overall typing speed is clearer. Figure 6.4 shows the spread of times, using the same two samples as before. Participant A typed `th` with an average time of 76ms, and Participant J managed only 143ms—almost half the speed of Participant A's efforts. Surprisingly, the second fastest typist, G, also types `th` at a distinctly different rate to Participant A, despite performing on average only 2 WPM slower overall. This is shown in Figure 6.5, which displays the same sample from Participant A against a random sample from Participant G. In this case G is more similar to J, with an average time of 141ms for `th` in the given sample. Yet G types nearly twice as fast as J overall.

This behaviour is not uncommon: a similar overall speed between typists

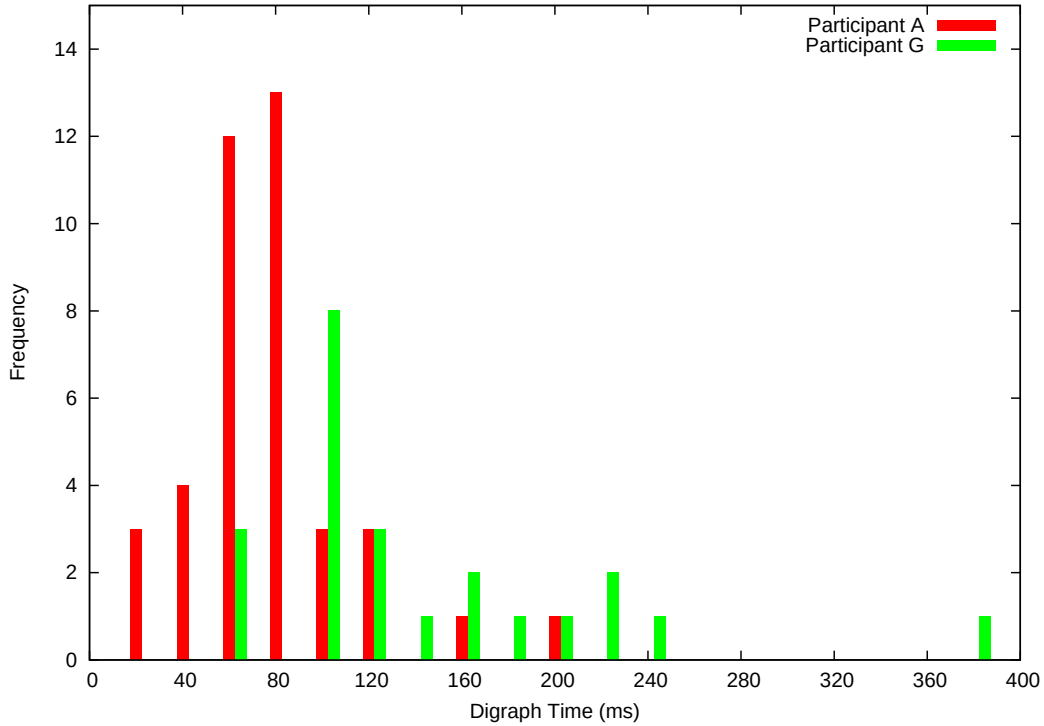

Figure 6.5: Times `th` for Participants A and G (one sample)

does not mean that their individual digraphs will be typed at the same rate. Ideally, for the purpose of verification, a user should type a digraph consistently, that is, with a low standard deviation, but uniquely—with a mean or median that distinguishes them from the rest. If all users are inconsistent the relevant digraph is unhelpful in discriminating between them. If some users are consistent, this quality can serve to separate them from the rest. If all users are consistent, the digraph must be typed uniquely in order for it to remain useful. It is difficult to determine in advance whether or not certain digraphs will have the required properties.

However, using the dataset from the previous chapter it can be shown that `Backspace Backspace` does not have the same discriminatory power as `th`. A number of techniques surveyed in Chapter 2 utilised the mean and standard deviation for each given digraph time, classifying a digraph as belonging to a user if it passed with x standard deviations of the mean ($x > 0, x \in \mathbb{R}$). Assuming a normal distribution, the probability density can be calculated for any digraph time, from the mean and standard deviation for a given digraph and user. This density is calculated using the continuous probability density function for a Gaussian distribution:

$$pdf = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(d-\mu)^2}{2\sigma^2}} \tag{6.1}$$

To evaluate the discriminating power of a single digraph, the probability density is first calculated for each digraph time, d. Next, the densities are ranked, noting whether each time belongs to the user, or an attacker. Finally, the ranked densities are used to calculate the area under the ROC curve (AUC), giving the probability that a user's time will have a higher density than an attacker's.

The higher the AUC for a user's digraph, the easier the user can be separated from the attackers and the more useful the digraph is for typist verification. Table 6.1 shows the AUC values for the top six most popular digraphs, and `in`, using the `sm-all` dataset and the first 10 users (the other users did not type enough to make their analysis useful). The last digraph, `in`, is included because Gaines *et al.* [28] found it to be one of five digraphs (`in`, `io`, `no`, `on`, `ul`) that together could be used to discriminate perfectly between six professional secretaries (see Chapter 2). Clearly, this digraph is the best one for verification of the selected digraphs in Table 6.1: it has a higher AUC value than any other digraph in the table, for all users. In contrast, the other digraphs perform poorly, rarely managing an AUC above 0.5—with the exception of `th`, which for some users rivals `in`. Three AUCs are missing from the table: `Right-Arrow Right-Arrow` for Participants A and B, and `Left-Arrow Left-Arrow` for Participant B. These digraphs were not ever typed by the relevant users, so it is impossible to calculate an AUC value for them.

Calculating the AUC for each digraph shows that a digraph's discriminatory performance is not only dependent on the digraph itself, but on the user as well. For example, Participant J scores an AUC of 0.784 for `th`, but Participant A only scores 0.577, even though earlier in this section it was shown that these two users were separable from each other using this digraph! Ranking all digraphs by their average AUC, the highest that was typed by all ten users is `in`, scoring 0.744, followed by `as`, `yo`, `er` and `on`. None of the digraphs perfectly discriminate any one user from the rest. The closest to do so is `tc`, which reports an AUC of 0.946 for Participant H.[1]

In general, digraphs that contain the space character and digraphs that contain unprintable characters such as `Right-Arrow` perform worse than the those formed from two alphabetical characters. However, there are exceptions

[1] Appendix D gives a ranked list of the top 54 digraphs typed by all ten users at least ten times each.

Participant	\b \b	→→	e Space	←←	Space t	th	in
A	0.490	-	0.556	0.461	0.539	0.577	0.737
B	0.562	-	0.441	-	0.522	0.484	0.756
C	0.431	0.411	0.595	0.468	0.585	0.633	0.668
D	0.489	0.609	0.590	0.639	0.522	0.585	0.789
E	0.530	0.432	0.696	0.451	0.482	0.699	0.706
F	0.466	0.580	0.574	0.651	0.515	0.592	0.719
G	0.466	0.493	0.576	0.469	0.549	0.535	0.781
H	0.621	0.600	0.652	0.623	0.522	0.616	0.726
I	0.530	0.454	0.516	0.512	0.521	0.500	0.730
J	0.539	0.571	0.558	0.528	0.471	0.784	0.831
Average	0.512	0.519	0.575	0.533	0.523	0.600	0.744

Table 6.1: AUC values for popular digraphs

for particular user/digraph combinations. This does not mean these digraphs cannot be used for verification; when treated as different behaviours from the rest they can help to discriminate between users. Sections 6.4 and 6.3 discuss ways to make use of them.

Notwithstanding the analysis in this section, it is impossible to determine in advance whether a particular digraph will be useful for verification unless the set of attackers is known. Most verification systems assume that this is not the case, and systems covered in Chapter 2 that use digraphs in the same manner as above do not select subsets of digraphs unless the technique utilises identification for authentication.

6.2 From Digraphs to Finger Movements

One of the major problems in a continuous typist verification system is that there are no guarantees that a user will type a particular digraph. If a single digraph differentiated a user from the rest, it would be tempting to try and force the user to type it. However, the advantage of a continuous system is that it can perform authentication without interrupting the user, which would be sacrificed by requiring the user to type something specific. One way of dealing with this is to abstract from the exact keys involved to the movements used. This reduces the number of possible features from thousands of digraphs to a handful of movements.

Abstracting to movements can be done in a number of ways. Dvorak *et al.* [23] describe six different classes of movements, as follows:

- **tap** Both letters in the digraph are the same, so the finger can simply 'tap' the same key (`ee`, `tt`).
- **reach** A digraph typed by the same finger on the same hand moving over a distance of only one key (`ft`, `ju`, `ed`).

Participant	Tap	Reach	Hurdle	Trill	Rock	Opposite
A	0.715	0.417	0.642	0.505	0.432	0.470
B	0.513	0.334	0.299	0.339	0.276	0.419
C	0.731	0.679	0.595	0.565	0.469	0.588
D	0.837	0.742	0.684	0.744	0.767	0.619
E	0.712	0.502	0.508	0.563	0.571	0.704
F	0.650	0.492	0.421	0.549	0.593	0.487
G	0.560	0.551	0.319	0.517	0.570	0.494
H	0.525	0.566	0.541	0.551	0.711	0.476
I	0.756	0.325	0.380	0.403	0.446	0.521
J	0.653	0.698	0.777	0.699	0.774	0.762
Average	0.665	0.531	0.517	0.544	0.561	0.554
% of Digraphs	7.59	2.56	1.16	14.94	25.20	48.55

Table 6.2: AUC values for movement types

- **hurdle** A digraph typed by the same finger on the same hand hurdling over the home row (`ce`, `un`).
- **trill** A digraph typed by adjacent fingers on the same hand (`fe`, `op`, `te`).
- **rock** A digraph typed by remote fingers on the same hand, in which the fingers often move in a rocking motion (`af`, `jp`, `on`).
- **opposite** A digraph typed by different hands (`if`, `od`, `ma`).

Of the top ten most discriminative digraphs typed by all ten users in Appendix D, all are trill or rock digraphs. This seems to indicate that the behaviours that can be used to verify users occur for digraphs typed with different fingers of the same hand. Table 6.2 shows the results of calculating the AUC for each movement type listed above, instead of for individual digraphs. A similar pattern is seen in Table 6.1: the performance of a movement for verification depends on both the movement and the user, and there is no movement that is better than the rest for all ten users. Surprisingly, taps are better discriminators on average than any others, despite only accounting for 7.59% of all digraphs.

A simpler abstraction is to consider only the hand that typed the digraph, giving three possible classes: left, right and both. A more complicated abstraction might use various combinations of hands, fingers and keyboard rows to form features. There are many possible abstractions—too many to cover here—but it is likely that the same pattern will occur for any abstraction. Each feature will have a different discriminating power for a particular user, but none will by itself be a panacea for typist verification, even though some features clearly perform significantly better than others.

The empirical evidence presented here indicates that digraphs formed from pairs of alphabetical letters are the most useful for typist verification. Never-

theless, it would be a mistake to attempt to use these alone. An ideal continuous system will not force a user to type particular digraphs, and because of this it cannot guarantee that a sample will contain any particular digraphs. Ideally, these results could be used to weight features, but doing this requires negative information. It is impossible to know in advance whether or not a digraph or movement will be helpful for verification, without understanding the way potential attackers type.

6.3 Key Usage

Although missing digraphs present a problem for verification, they can also be useful—the system may perform verification by considering *what* a user is typing. This is known as authorship verification. It is different to typist verification, which considers *how* a user is typing. Authorship verification systems typically work on a blocks of text, instead of the stream of events used to create them. Such systems are able to supplement typist verification because a stream of key events can easily be turned into a block of text.

Unfortunately, potentially helpful information will be lost when a typing stream is turned into text. Some keys on a computer keyboard produce unprintable characters that are important when creating text, but do not affect reading or understanding. These include cursor keys, `Backspace`, `Delete`, `Insert`, `Home`, `End`, `Page Up`, `Page Down`, `Shift`, `Meta`, `Control` and `Alt`. A typical authorship verification system would never see these keys because they do not produce characters directly, but are used for editing or modifying other keys. However, their occurrences are readily available for typist verification.

The simplest way of using these keys for verification is to calculate how often they occur in each sample and turn this into a probability. The absence of a particular key cannot confirm an identity because it might not have been typed simply by chance. On the other hand, the presence of a key might indicate that an attack is underway, especially if the user has never pressed it before. For example, Participant B never used the `Left-Arrow` or `Down-Arrow` key in the `sm-all` dataset, yet all others did use these keys in at least one of their samples. If this observation was used for verification, a sample containing the `Left-Arrow` or `Down-Arrow` keys would be considered unlikely to belong to Participant B. Such a sample should not be totally discounted, because there is always a slim chance that the user did indeed type with this key, perhaps by accident.

Navigational keys and keys used for editing are not the only ones whose

usage (or lack thereof) should be considered. After analyzing the most common digraphs from the previous sections, `88` stood out. It was the 15th most popular digraph, whereas no other numerical digraphs appeared in the top 250. Of the 5378 times that `88` appeared, only seven occurrences were not from Participant B. It was typed once by Participants A and G, and five times by Participant D. On closer inspection it was discovered that the digraph was actually `**`, that is, `88` modified by the `Shift` key. Participant B often used a line of asterisks to separate text within their emails, pressing and holding the key until a sufficiently long line was produced. This kind of behaviour would be caught by any competent authorship verification system because the repeated asterisks would be present in the printed text.

All of the digraphs in this chapter are evaluated in lowercase. Although the datasets collected earlier indicate whether `Shift` was down when another key was pressed, `Capslock` was not recorded, so it is difficult to determine whether some characters actually appear on screen in uppercase. Furthermore, all modifiers—`Shift`, `Meta`, `Control` and `Alt`—were recorded in the form of a mask for each key. Because of this, it is impossible to perform a similar analysis to Lau *et al.* [44] (reviewed in Chapter 2), checking to see which of the left or right versions of these four keys were pressed.

Navigational keys and those used for editing should be used with caution for typist verification. Their presence or absence cannot by itself confirm an identity. Whether they are typed depends on a number of factors, including whether a mouse was available to provide input. Just like the digraphs in the previous two sections, the discriminatory power of a particular key's usage is dependent on the user, the key, and the potential set of attackers.

6.4 Pausing

Studies of reading have found that "various levels of structure contribute to the process: letters, digraphs, syllables, words, phrases, sentences, and so on"[51]. Many typist verification techniques apply this idea by considering not just digraphs, but larger units such as words. One difficulty with any timing collected from typing samples is that it may be influenced by pauses. Inevitably typists pause as they prepare to type each structure, just as they would during reading. They may also pause for other reasons, such as being distracted, holding something that impairs their ability to type, tiredness, or leaving the terminal. Section 6.1 found that digraphs containing the `Space` key were poor discriminators compared to others. This is most likely due to

Participant	Before Space	After Space
A	0.531	0.310
B	0.565	0.745
C	0.786	0.584
D	0.507	0.515
E	0.619	0.590
F	0.660	0.521
G	0.486	0.518
H	0.689	0.391
I	0.569	0.705
J	0.548	0.570
Average	0.596	0.545

Table 6.3: AUC values for pausing between words

typists pausing while they prepare to type the next word. Unfortunately, it is impossible to confirm this because the typist would need to be observed, either in person or using a camera, to understand what causes them to pause.

Verification systems can utilise pauses explicitly to differentiate between typists, regardless of their cause. To do this, the system considers how long the typist takes to complete certain structures. Digraphs that contain the `Space` character can be useful here: it is likely some typists pause before pressing the space key, whereas others pause after doing so. After testing this with the `sm-all` dataset, it was found that most typists tended to pause after pressing the space key instead of beforehand. Just as we saw with other types of digraphs, these times were a useful discriminator for some of the users. The results of this evaluation are presented in Table 6.3.

Similar patterns to those in Table 6.3 are observed using other simple structures such as syllables, and larger ones such as sentences. Interestingly, digraphs that cross syllable boundaries are usually typed faster than elsewhere in the text. In this case, syllables were calculated using a hyphenation engine from the iText PDF generation library [45], and each digraph was split into one of two groups depending on whether or not it crossed a syllable boundary. On average, those that did were faster that those that did not 75% of the time, even when times greater than one second are removed from the analysis.[2]

Unfortunately, at higher levels of structure the effects of software timers are amplified; software timers have a resolution no finer than 10ms—higher if the system is under load. As consecutive events are accumulated into larger structures the overall timing error increases. For example, four events produce an overall error of between zero and 40ms. If this error is sufficiently large it may negatively impact a verification system's performance. This is a problem when considering pauses because it is difficult to determine how much of a

[2]Syllables are likely to produce only short pauses because the typist is moving between sections of text but ideally maintaining a flow, and so removing excessively large times should not affect the evaluation.

timing belongs to the user's natural patterns, how much is the result of pausing, and how much can be attributed to system error. In all the evaluation in this section no attempt was made to divide any timing into its component parts. Here, pausing is just the act of taking longer than usual to complete a task.

6.5 Ordering of Events

Typed language structure is made up of a number of key events. Characters need only two events; digraphs four; words and sentences need much more. Only half the events—the press events—cause text to appear on screen. Key releases are important for modifier keys, such as `Shift` or `Ctrl`, which transform a normal key in some way until they are released. Releases are also useful when a key is pressed for a long period: computer operating systems usually begin to automatically repeat characters after a key has been held down for a certain period of time, and continue to repeat at regular intervals until the key is released. Keystroke recognition systems that use audio recordings to determine what was typed [81] additionally define a *touch peak* and *hit peak*, referring to when the finger touches the key and when the key hits the keyboard backplate respectively. The keyboard itself can only provide information as to the state of a key, i.e. whether it is up or down.

When typing, press events must occur in the same order as the intended characters, but releases are not constrained. Studies have shown that touch typists move towards 3 different keys at any given time on average [16]. With so many movements going on at once, it would be unrealistic to expect events to alternate perfectly between presses and releases. Even hunt-and-peck typists can hold down overlapping keys if they use both index fingers to type.

The extent to which keys overlap can be quantified in a number of ways. In this thesis, four different measures are used for event ordering. Whilst there are many possible ways of evaluating this, these four metrics performed the best out of all that were tested. The metrics used are:

- **Slur Rate** The slur rate is the proportion of times a release event does not follow immediately after the press event for the same key, over the total number of events in the sample. It indicates how often a given user slurs events, instead of pressing and releasing keys cleanly. A typical user from the `sm` datasets has a slur rate around 0.7.

- **Slur Length** The slur length measures how many key events occur between a press event for a given key and its related release event. For

an entire sample, the slur length is the average length of all slurs. This measure quantifies how long a user slurs, without considering the overall time involved. The average slur length across all users in the `sm` datasets is 1.7.

- **Press Before Release** Following the same idea of slurring, this measure considers the proportion of times a press event for a new key occurs before the release event for the given key. It measures the number of times that $p_1p_2r_1r_2$ occurs as a pattern, as a proportion of the number of sequences with the pattern $p_1p_2r_xr_y$ where x and y may be any key. Most users in the `sm` datasets had a press before release value above 0.9, however the average value was 0.78.

- **Paired Perfect Order** The slur rate measures how often a given user does not press a key and immediately follow it with the release of the same key. This attribute is the inverse—it measures how often the user presses events in a "perfect" order but also requires that there is two consecutive "perfect" sets of key events. That is, it uses the event sequence $p_1r_1p_2r_2$. Like the Press Before Release, this is divided by the number of sequences with the same pattern $p_1r_xp_2r_y$ where x and y may be any key. A typical user from the `sm` datasets has a paired perfect order value between 0.8 and 1.

Table 6.4 shows the AUC for each of the four metrics, for each user. Overall, all four perform well compared to others considered here, and all but the Slur Length perform better than any other metric in this chapter. Even the Slur Length is only beaten by the digraph `in`, which achieves an average AUC of 0.744 across all ten users. On a per-user basis all four metrics perform well, with the exception of Participant D and the Slur Length and Press Before Release metrics. Surprisingly, Participant J is almost perfectly authenticated on Slur Rate alone! No other metric in this chapter comes so close to perfect authentication for any user.

The four metrics, Slur Rate, Slur Length, Press Before Release and Paired Perfect Order, are all good discriminators. They show that there is plenty of information available for authentication that would be lost if the system only considered press events. All four can be easily utilised as part of a typist verification system, but to date no system has used these attributes.

Participant	Slur Rate	Slur Length	Press Before Release	Paired Perfect Order
A	0.668	0.925	0.930	0.630
B	0.858	0.652	0.938	0.848
C	0.886	0.528	0.863	0.911
D	0.984	0.375	0.339	0.956
E	0.970	0.979	0.787	0.950
F	0.883	0.679	0.859	0.885
G	0.857	0.878	0.872	0.786
H	0.917	0.595	0.897	0.907
I	0.694	0.529	0.840	0.674
J	0.996	0.576	0.826	0.983
Average	0.871	0.671	0.815	0.853

Table 6.4: AUC values for key ordering measures

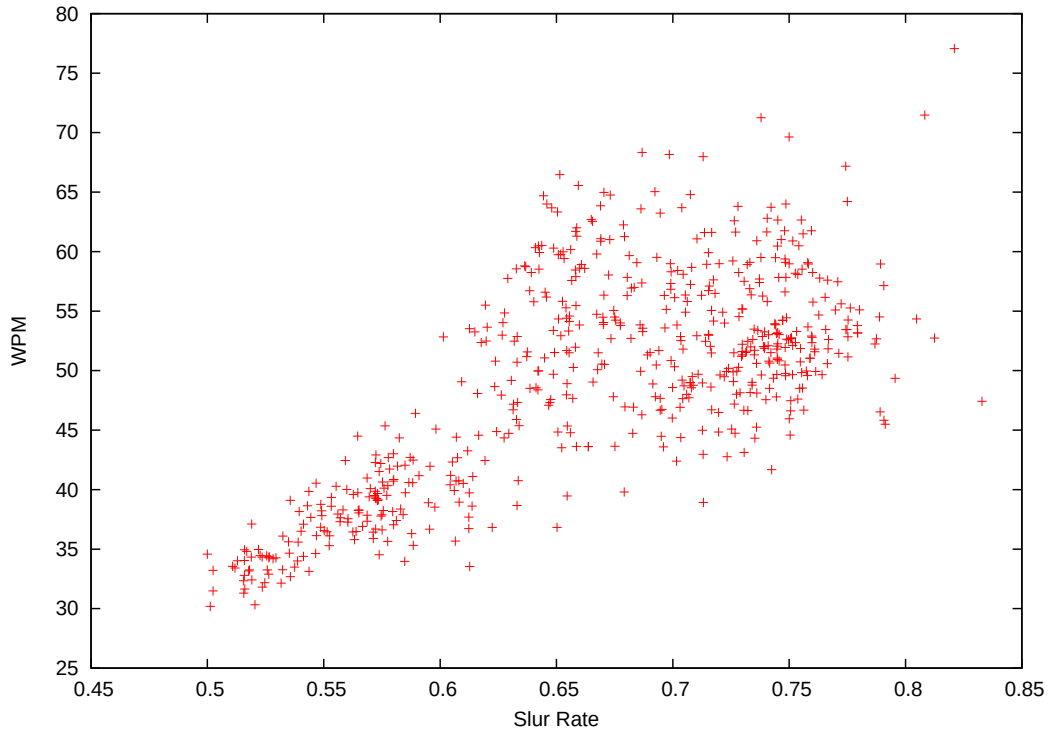

Figure 6.6: Slur rate versus typist speed (WPM)

6.6 Speed and Error Rate

When users are trained to touch type there are two important factors that are used to estimate their skill level: speed and error rate. All typing verification systems examined in Chapter 2 operate at a low level, using habits gleaned from examining presses, releases, digraphs and other small structures in typing. The findings in this chapter shows that although low level patterns exist, aggregates such as Slur Rate can be more informative for verification than individual digraphs.

The analysis in Section 6.5, implies that typing speed is useful for verification. The faster a user types, the closer together events occur and the more likely the typist is to slur presses and releases. This is demonstrated

Participant	WPM	Backspace
A	0.887	0.771
B	0.851	0.883
C	0.830	0.635
D	0.955	0.768
E	0.953	0.897
F	0.788	0.620
G	0.709	0.716
H	0.832	0.692
I	0.896	0.629
J	0.978	0.673
Average	0.868	0.728

Table 6.5: AUC values for WPM and backspace rate

in Figure 6.6, which shows the average WPM speed per sample against the Slur Rate for the same sample. Considering that these two attributes are so strongly correlated, it would be foolish to ignore typing speed in favour of lower level attributes such as digraph times.

Evaluation using the `sm-all` dataset shows that typing speed and error rate are just as useful for verification as the event ordering metrics in the previous section. Table 6.5 shows the AUC values for typing speed (measured in words per minute) and backspace rate (the proportion of backspaces in the sample). In some cases, such as Participant A, speed and backspace rate are better attributes than the Slur Rate, scoring higher AUC values. For others, the metrics in the previous section perform better. However, these two aggregates perform equally well as—usually better than—the best individual digraph, `in`. These results indicate these two simple metrics are viable attributes for verification.

6.7 Summary

On the face of it, typing appears simple: a user presses and releases keys to create text on the screen. When examined further, it is exposed as a complex task. Some keys are pressed before others are released. Some keys are used in preference to others when editing text. The time between certain digraphs does not always accurately reflect the typist's overall speed. And typists arbitrarily pause. Typist verification is a difficult task because none of these factors can be used alone to verify identity, and many have performance that depends on the user.

Unfortunately, it is impossible to determine in advance whether or not a particular attribute will be useful for identifying a given user. To do this, the set of possible attackers must be known. In some situations this may be available, but in general for verification this is assumed to not be the case.

Ideally, the performance of a typist verification system should not rest on knowledge of other users, but only on knowledge about the target user.

The success of typist verification using the attributes discussed in this section is currently unknown, even though the usefulness of each attribute is individually quantified here in terms of their AUC value. Chapter 7 discusses the sort of machine learning algorithms that can make predictions on the source of a sample using the attributes discussed in this chapter. Chapter 8 shows how to use these attributes and the algorithms from Chapter 7 to perform typist verification.

Chapter 7
One-Class Classification

All of the techniques for typist verification presented in earlier chapters, with the exception of Gunetti and Picardi's method [33] and the Individual Digraph Classifier, have one important feature in common: they are one-class classifiers. One-class classifiers are a type of machine learning algorithm that only requires information about a single 'target' class in order to build a model that can be used for prediction. They can make two possible judgements on an unknown instance. They predict *target* if the instance appears to belong to the same class the model was trained on, or *unknown* if the instance does not seem to come from the same class as the training data. Gunetti and Picardi's technique is not one-class classification because it requires data from many different people in order to build a predictive model for a given user.

One-class classification is often called *outlier detection* or *novelty detection* because it attempts to differentiate between data that appears normal and abnormal with respect to the training data. The literature can be confusing: outlier/novelty detection sometimes refers to problems where all instances are available at training time, and instead of building a model for prediction the learning algorithm must distinguish what—if any—instances from the dataset are outliers. However, the literature is consistent when referring to one-class classification. In this case, a training dataset is provided with information about a single class (which may or may not contain outliers) and a model is built that can then be used to predict whether previously unseen instances belong to the target class.

This approach to learning is typically used in situations where it is inappropriate to make use of non-target data. For example, consider the problem of password hardening discussed in Chapter 2.3. This is a biometric system used to strengthen logins, which it achieves by verifying that the provided password is typed with the correct rhythm. An important aspect of the system is that it is strengthening the password—not replacing it. This means that it is not possible to collect negative data for a given target password because to do so the password would have to be supplied to the attacker typist. This

constraint does not apply to typist verification because the patterns are mined from free text and it is possible to collect negative data without compromising the system.

It may seem sensible to suggest that two-class classification should be used in preference to one-class classification to verify a user since all the datasets presented here contain data from many different typists. Using two-class classification simplifies the learning problem—the algorithm need only to draw a boundary between the target class and all others. There are two reasons why one-class classification should be still considered. The first is practical: if typist verification is to be used on personal computers then acquiring typing rhythms for attacking users may be difficult. The second—and most important—reason for considering a one-class setup is that we cannot guarantee that the data that is available will cover the range of potential attackers. In some situations, including typist verification, new classes may appear at prediction time that are different from *all* classes available during training time.

Since there is only a limited number of attackers (20–30 in the datasets here), intuition suggests that one-class classification should be used in preference to two-class classification for typist verification because it seems unlikely that the existing attackers will provide comprehensive coverage of the potential attacker space. But it is not clear that this is the case: the small amount of negative data used with two-class classification may still form a better verification model for a user than one-class classification alone.

This chapter investigates when one-class classification should be applied to a given verification problem. It deals with general verification problems, rather than the specific problem of typist verification, so the techniques used can benefit other research areas. The next chapter relates the work discussed here to typist verification.

The next section describes how one-class and multi-class classifiers can be tested to discover their prediction accuracy for a given problem; it also describes how to compare the setups fairly. The approaches to classification are then compared and conclusions drawn about the use of one-class classification for verification. This comparison uses a density function for one-class classification; later sections will introduce a better one-class classifier. Section 7.2 covers methods of one-class classification, especially those used for verification problems. Finally, Section 7.3 explores how one-class and multi-class classification techniques can be combined to form a single one-class classifier, using an artificially generated second class alongside the target data for the multi-class component. This artificial data can be replaced with real negative data—if

such data is available—to improve on prediction accuracy.

7.1 One-Class Classification versus Multi-Class Classification

For verification problems such as typist verification, there are at least three possible ways to approach classification: multi-class classification, two-class classification and one-class classification. When negative (i.e. non-target) data is available, standard multi-class classification can be used. In this approach, the system learns to differentiate between all the classes in the training data, and the model is then used to predict the class of an unseen instance. If the predicted class matches the target class the verification succeeds, otherwise it fails. A potential disadvantage of this approach in the context of verification is that we are primarily interested in identifying occurrences of completely novel classes at prediction time and multi-class classifiers may not accurately discriminate against these; they match to the closest known class.

It is also possible to approach classification by reformulating the problem as a two-class classification problem. As a first step, all classes in the training data except the target class are relabelled to a single unified class: *outlier*. Then a multi-class classifier learns *target* versus *outlier* using the relabelled data. The classifier can make two possible predictions on an instance, *target* or *outlier*, corresponding to verification success and failure respectively. Two-class classification can potentially suffer from the same problem as standard multi-class classification when novel classes arise at prediction time: the classifier is only as good as the coverage of the outlier class.

Finally, the non-target classes present in the training data can be completely ignored and a one-class classifier used for verification. As discussed earlier, one-class classifiers do not use any negative data during their training phase. This means that a one-class classifier's accuracy is unaffected by the coverage of the non-target classes in the training data. However, it can be difficult to define an accurate boundary around a target class without knowing where non-target instances are likely to appear. The usual practice is to provide a parameter to the one-class classifier that is used to define what percentage of the training data should be considered outliers. This parameter is known as ν in the case of one-class support vector machines [63]. Using this parameter, the classifier can create decision boundaries by identifying outliers in the target class and effectively use these as a second class.

When negative data is available at training time, all three methods can be used for verification. However, choosing a classifier presents a problem: it is not obvious how to perform a fair comparison of multi-class and one-class classifiers in this context because each setup uses a slightly different set of test and training data. Therefore, before proposing that one method should be used over another for a given verification problem, the issue of comparing the classifiers must first be tackled.

7.1.1 Comparing Classifiers Fairly

A standard method for evaluating one-class classifiers is to split a multi-class dataset into a set of smaller one-class datasets, with one dataset per class containing all the instances for the corresponding class. The one-class classifier can then be trained on each dataset in turn, with a small amount of data held out from the training set and all the other datasets used for testing. Depending on the number of instances available for each class, this generally means that there is a large amount of negative (or 'attacker') data for testing, and a relatively small amount of positive data for both testing and training.

Multi-class classifiers are often evaluated using stratified 10-fold cross-validation, where the data is split into 10 equal-sized subsets, each with the same distribution of classes. The classifier is trained 10 times, using a different fold for testing and the other 9 folds combined for training. These two different evaluation methods are not comparable: in each one the classifier is trained on a different proportion of data for a given class, and is tested on different quantities of data.

In fact, it is not immediately obvious how to perform a fair comparison of one-class classifiers and multi-class ones: the former are designed to deal with classes that are unseen at training time, but the latter typically handle only classes that they have been trained on. There are two types of multi-class classification—biased, where the classifier has seen data from the attacker class during training, and unbiased, where attackers are always novel. Fortunately, there is a way to compare all three types of classification—multi-class biased, multi-class unbiased and one-class—to each other.[1] Let us consider the biased multi-class case first. A target and held-out 'attacker' class are identified. Then, a normal stratified cross-validation fold is performed, which maintains the class distributions. However, before relabelling the non-target classes,

[1]This technique is covered in further detail in "Discriminating Against New Classes: One-Class versus Multi-Class Classification" by Hempstalk and Frank [36].

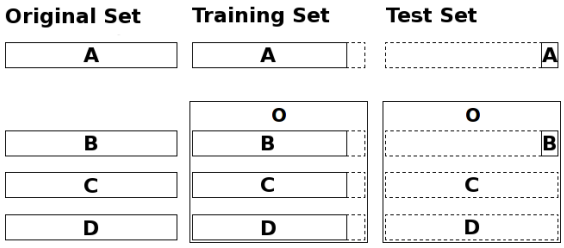

Figure 7.1: Cross-validation for biased two-class classification (with relabelling)

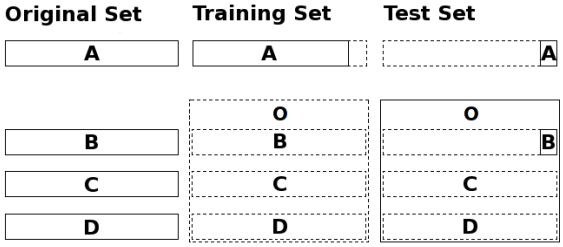

Figure 7.2: Cross-validation for one-class classification (with relabelling)

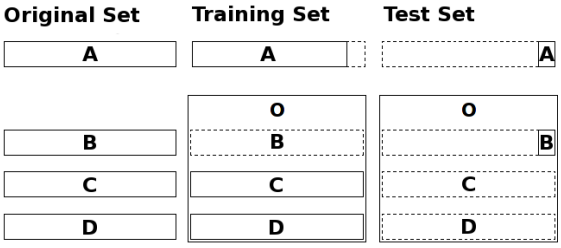

Figure 7.3: Cross-validation for unbiased two-class classification (with relabelling)

instances from the test set that do not belong to either the target or held-out class are deleted. Finally all non-target classes are relabelled to O, and the evaluation is performed. Figure 7.1 shows the resulting datasets used for two-class classification. Let us now consider the evaluation of a one-class classifier: it simply ignores all outlier training data, as shown in Figure 7.2. Lastly, let us consider unbiased two-class classification. In this case, before the final relabelling is performed, the heldout class is removed from the training set, as demonstrated in Figure 7.3.

The advantage of this approach is that the test set and the target data in the training set are identical for all three classification techniques, and it is now possible to compare results. Furthermore, as an additional benefit, it is also possible to compare true multi-class classification based on more than two classes with two-class and one-class classifiers by omitting the relabelling step where the non-target classes become class O.

Evaluation of a single target class using different classification techniques (multi-class, two-class and one-class) can be performed by accumulating all predictions for each possible combination of target and held-out classes. The area under the ROC curve (AUC) is then calculated for each target class. The AUC is used here for comparisons because it is independent of any threshold used by the learning algorithm. To compare classifier performance on an entire multi-class dataset, the weighted average AUC is used, where each target class c_i is weighted according to its prevalence $p(c_i)$:

$$AUC_{weighted} = \sum_{\forall c_i \in C} AUC(c_i) \times p(c_i) \tag{7.1}$$

Using a weighted average rather than an unweighted one prevents target classes with smaller instance counts from adversely affecting the results.

7.1.2 Evaluation Method

Five different classification techniques were each tested on UCI benchmark datasets [3] with nominal classes, providing an experimental comparison of the classification techniques. For each dataset ten-fold cross-validation was repeated ten times. The learning techniques used are:

1. Biased multi-class classification using the Naïve Bayes algorithm. No relabelling was performed, and data from the heldout class was not removed from the training dataset (similar to Figure 7.1, but without relabelling the non-target classes to O).

2. Unbiased multi-class classification using Naïve Bayes. No relabelling was performed, but data from the heldout class was removed from the training dataset (similar to Figure 7.3, but without relabelling the non-target classes to O).

3. Biased two-class classification using Naïve Bayes. All non-target classes were relabelled to 'outlier', and the test set contained only the target and (relabelled) heldout class, as in Figure 7.1.

4. Unbiased two-class classification using Naïve Bayes. All non-target classes were relabelled to 'outlier', and the test set contained only the target and (relabelled) heldout class and instances of the heldout class were removed from the training set, as in Figure 7.3.

5. One-class classification using a Gaussian density estimate for numeric attributes and a discrete distribution for each nominal one, assuming independence between attributes (i.e. 'Naïve Bayes' with only one class). All non-target classes were relabelled to 'outlier' and the test set contained only the target and (relabelled) heldout class, as in Figure 7.2.

The experiments utilise WEKA's [77] implementation of Naïve Bayes with default parameters for all multi-class and two-class tasks. This is the classifier in WEKA that is directly comparable to the one-class classifier used. The one-class classifier fits a single Gaussian to each numeric attribute and a discrete distribution to each nominal one. A prediction for an instance, X, is made by assuming the attributes are independent. The same happens in Naïve Bayes, but on a per-class basis. In both Naïve Bayes and the one-class classifier, missing attribute values are ignored.

7.1.3 Results

Using the methodology discussed previously, experimental results obtained on UCI datasets are presented in this section. First, results from comparing the five different classification techniques discussed above are shown. Then some of the results are examined in greater detail, focusing on unbiased two-class classification versus one-class classification.

Comparison on UCI Datasets

Table 7.1 provides empirical results for the five different classifiers on UCI datasets, compared using the weighted average AUC described in Section 7.1.1. Bold font indicates wins for two-class unbiased classification versus one-class classification and vice versa. Only UCI datasets with three or more class labels were used, because the evaluation technique requires at least three classes.

Table 7.1 has some noteworthy results, aside from the expected outcome that the biased classification techniques (columns 1 and 3) outperform the unbiased and one-class methods. Of the two biased techniques, one might naïvely expect the two-class approach to perform better: there are less labels and the outlier class contains many of them. However, on all but four of the 26 datasets (glass, lymphography, mfeat-factors and vehicle), the multi-class classifier either performs the same or better than the two-class classifier—and the difference for these four datasets is not significant. This can be explained by the fact the multi-class Naïve Bayes classifier is able to form a more complex

Datasets	Classes	Classification Techniques				
		Multi-class Biased (1)	Multi-class Unbiased (2)	Two-class Biased (3)	Two-class Unbiased (4)	One-class (5)
anneal	6	0.957	0.575	0.948	0.605	**0.788**
arrhythmia	16	0.801	0.724	0.775	**0.723**	0.576
audiology	24	0.960	0.883	0.946	**0.897**	0.881
autos	7	0.831	0.722	0.807	**0.736**	0.567
balance-scale	3	0.970	0.851	0.941	**0.851**	0.806
ecoli	8	0.958	0.855	0.947	0.889	**0.927**
glass	7	0.760	0.680	0.763	0.605	**0.702**
hypothyroid	4	0.931	0.576	0.915	0.587	**0.648**
iris	3	0.994	0.671	0.990	0.671	**0.977**
letter	26	0.957	0.932	0.941	**0.935**	0.887
lymphography	4	0.911	0.432	0.914	0.425	**0.739**
mfeat-factors	10	0.992	0.946	0.975	**0.964**	0.948
mfeat-fourier	10	0.966	0.917	0.949	**0.930**	0.909
mfeat-karhunen	10	0.996	0.969	0.983	**0.976**	0.955
mfeat-morph	10	0.952	0.890	0.948	0.928	**0.941**
mfeat-zernike	10	0.960	0.906	0.946	**0.912**	0.897
optdigits	10	0.986	0.948	0.978	**0.969**	0.959
pendigits	10	0.980	0.915	0.962	0.942	**0.953**
primary-tumor	22	0.839	0.778	0.834	**0.784**	0.732
segment	7	0.971	0.863	0.952	0.863	**0.937**
soybean	19	0.994	0.966	0.988	**0.973**	0.961
splice	3	0.993	0.831	0.983	**0.831**	0.720
vehicle	4	0.767	0.671	0.768	**0.696**	0.658
vowel	11	0.956	0.907	0.926	**0.909**	0.865
waveform	3	0.956	0.692	0.927	0.692	**0.864**
zoo	7	0.999	0.963	0.984	0.963	**0.984**

Table 7.1: Weighted AUC results for multi-class, two-class and one-class classifiers

model, with as many mixture components as there are classes. These results suggest that if one does not expect any novel class labels at testing time, one should not merge classes to form a two-class verification problem if Naïve Bayes is used as the classification method.

In situations where the attacker class is not present in the training set (i.e. considering the unbiased classifiers), the picture is not so clear. The multi-class classifier (column 2) scores three wins, five draws and 16 losses against its two-class counterpart (column 4). This result is consistent with intuition: when expecting novel classes during testing, it is safer to compare to a combined outlier class because the multi-class model may overfit the training data. By combining the non-target classes into one class we can provide a more general single boundary against the target class, and increase the chance that a novel class will be classified correctly.

As described earlier, one-class classification is intended to deal with novel classes, and learns only the target class during training. One would expect that the multi-class classifiers could potentially have an advantage because it has seen negative data during training. However, as highlighted in Table 7.1, when considering the unbiased two-class classifier (the most accurate classifier when novel classes are expected during testing), the one-class classifier loses

on fifteen datasets and wins on the other nine. On closer inspection, most of the datasets where the unbiased two-class classifier wins have a large number of class labels; at least half of the winning datasets have 10 or more original class labels. In contrast, the one-class classifier wins on only two datasets with many class labels—pendigits and mfeat-morph.

Exploring a Domain for One-Class Classification

In order to clarify in which situations one-class classification should be applied, it is instructive to investigate the relationship between the number of class labels available at training time and the accuracy of the two prospective classifiers: the one-class classifier and the unbiased two-class classifier. The number of instances for each class is also relevant; classes with a large number of instances will generally result in a more accurate classifier. However, here the primary concern is whether a classifier is capable of identifying novel classes. Hence, it is more appropriate to investigate the effect on accuracy obtained by reducing the dataset size by removing all instances for a particular class label, rather than by performing a random selection of instances.

For each of the datasets where the unbiased two-class classifier outperformed the one-class method, the experimental procedure from Section 7.1.2 was repeated, but on each run an additional single class label (and all associated instances) was removed from the training dataset. This was repeated until only two classes remained: the target class, and one original—but relabelled—class. Since the heldout attacker class is also being removed before training, reducing the datasets any further results in no attacker instances present in the training set. The process for producing the test set remains the same—ensuring that the dataset used to obtain the AUC is identical for each method. The results are presented in Table 7.2. For brevity the final column shows the number of classes that were removed before the one-class classifier became better than the two-class one.

For all but two of the datasets shown in Table 7.2, there exists a point where it is better to use the one-class classifier over the two-class one. This is not unexpected: as the number of non-target classes is reduced, their coverage diminishes until it is no longer worthwhile to use them to define a boundary around the target class. For the two datasets where this is not the case, arrhythmia and primary-tumor, the density estimate does not appear to form a good model of the data for either classifier and the AUC is relatively low in both cases.

Dataset	One-class	Original Two-class	Total Removals	Two-class Final AUC	One-class Wins After x Removals
arrhythmia	0.576	0.723	13	0.606	-
audiology	0.881	0.897	21	0.736	7
letter	0.887	0.935	23	0.876	23
mfeat-factors	0.948	0.975	8	0.736	5
mfeat-fourier	0.909	0.949	8	0.726	5
mfeat-karhunen	0.955	0.983	8	0.836	7
mfeat-zernike	0.897	0.946	8	0.728	4
optdigits	0.959	0.969	7	0.855	4
primary-tumor	0.732	0.784	19	0.740	-
soybean	0.961	0.973	16	0.954	10
vowel	0.865	0.909	8	0.827	8

Table 7.2: Weighted AUC results for reduced numbers of non-target classes

When plotting individual results, the weighted AUC continually decays as classes are removed from the training data. As a typical example, Figure 7.4 shows the results for the audiology dataset. Of course, the one-class classifier maintains a constant AUC because it does not use non-target data during training. The shape of decay shown in Figure 7.4 is typical of the datasets in Table 7.2.

From the results in this section we can say that where there are limited non-target classes available at training time, thus increasing the potential for a novel class to appear that is dissimilar from any existing non-target class, one-class classification should be used in preference to two-class classification.

7.1.4 Summary

In this section, comparisons of one-class and multi-class classification for verification problems have been explored. For verification problems—like typist verification—multi-class, two-class and one-class classification can be used to find a predictive model when negative data is present in the dataset. The effectiveness of the multi-class classifiers is highly dependent on the coverage of the negative data, which is considered here to be a function of the number of non-target classes that exist in the data. How many non-target classes are required for (unbiased) multi-class classification to be a more accurate solution than a one-class approach depends on the dataset involved. One-class classification is generally more effective when novel classes appear at prediction time that are different to *all* classes that appear at training time.

When dealing with verification problems based on behaviours, such as typist verification, these results indicate that it is preferable to use one-class classification methods because it is always possible for an impostor to differ from all the samples on which the system was trained. Furthermore, behavioural samples for impostors can be hard to obtain, so using one-class classification

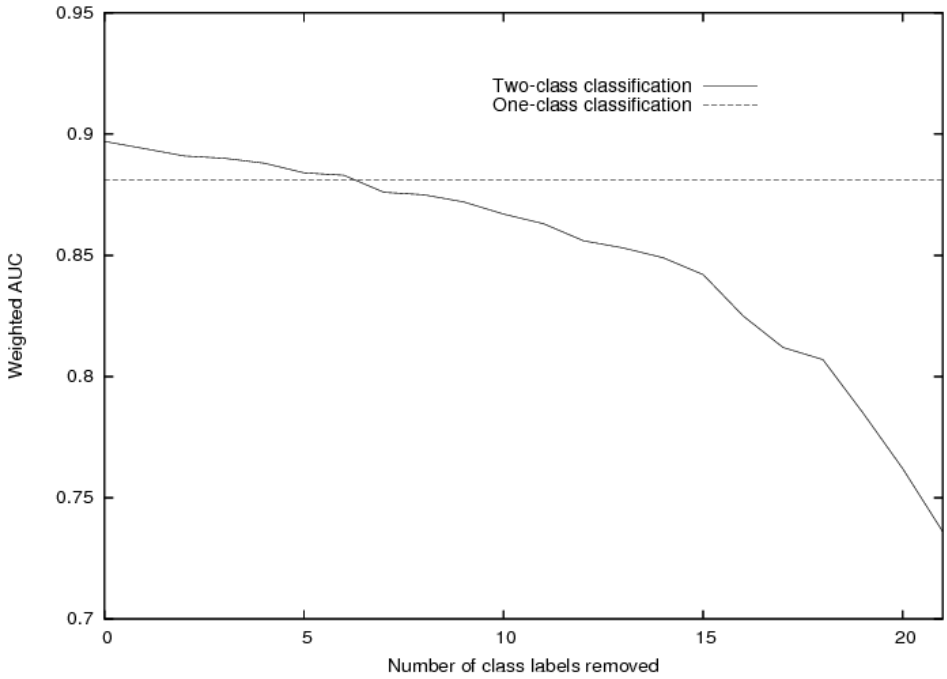

Figure 7.4: Classes removed versus weighted AUC for the audiology dataset.

simplifies the process of collecting data. It is still possible to make use of negative data using one-class classification techniques, as we will see in Section 7.3. The next section describes general methods of one-class classification.

7.2 Methods of One-Class Classification

Existing models for one-class classification either extend current methods for multi-class classification or are based on density estimation. In the latter approach, density estimation is performed by fitting a statistical distribution, such as a Gaussian, to the target data. Any instances with a low probability of appearing (more precisely, low density value) can be marked as outliers [54]. This is a sensible approach in cases where the target data follows the selected distribution very closely. The challenge is to identify an appropriate distribution for the data at hand. Alternatively one can use a non-parametric approach, such as kernel density estimation, but this can be problematic because of the curse of dimensionality and the resulting computational complexity.

Examples of applications of density estimation for one-class classification include: detecting masses in mammograms [69] using Parzen windows and

Gaussian mixture models, identifying outliers in sensor data [68] by utilising kernel density estimators, and predicting upcoming critical failures in jet engines [15] using Gaussian mixture models. Only Subramanium *et al.* [68] provides empirical results, stating that the "algorithms averaged 99% precision, and 93% recall for the engine measurements" [68]. In many cases the goal is to achieve a low IPR to ensure that real outliers are never missed: it is generally considered better to wrongly reject target data as an outlier than to let an outlier go undetected. In these examples failing to identify an outlier can be the difference between life and death. However, achieving a low IPR often results in a high FAR—Tarassenko *et al.* [69] notes this, commenting that all 40 masses were correctly identified as novel, but "a significant number of false positives were discovered ... just over one per image on average".

A common statistical approach to one-class classification is to identify outliers as instances that are greater than a distance, d, to a percentage, p, of the training data [4, 54]. This approach is analogous to clustering the data using a machine learning algorithm and determining a suitable boundary that encloses all the clusters [22]. The boundary can be generated by adapting the inner workings of an existing multi-class classifier [63], or by using artificial data as a second class, in conjunction with a standard multi-class learning technique [1, 24]. Methods in the former category generally rely heavily on a parameter that defines how much of the target data is likely to be classified as outlier [70]. This parameter defines how conservative the boundary around the target class will be. If it is chosen too liberally, then the model will overfit and we risk identifying too much legitimate target data as outliers. A drawback of these techniques is that an appropriate parameter value needs to be manually chosen at training time.

In contrast, when density estimation is used for one-class classification, a threshold on the density can be adjusted at prediction time to obtain a suitable rate of outliers. In some situations, where parametric density estimation fails, using classification-based methods may be favourable; these techniques are generally able to define boundaries on data that cannot be tightly modelled by a standard statistical distribution. In some cases there is a close link between classification-based techniques and density estimators. For example, it has been shown that one-class kernel Fisher discriminant classifiers can be used to perform non-parametric density estimation [61]. However, this only applies to very specific learning techniques.

Support vector machines (SVM) are a popular example of classification-based one-class classifiers. Some of their applications include: authorship

verification in combination with a technique referred to by the authors as "unmasking" [42], detecting anomalies in mass spectral data [72], analysis of patient seizures [29], and document classification (i.e. tagging) [46]. The implementation used by many of these examples—and others—was proposed by Schölkopf *et al.* [63] and has been implemented in a package called libSVM [11], an open-source SVM library. Their approach forms a hypersphere in feature space, and uses the manually provided parameter ν to decide how much of the training data to reject as outliers.

Of course, one-class classification has also been used for typist verification—the subject of this thesis. The methods employed vary and a full background is given in Chapter 2 and 3. Of the two typist verification methods reviewed in Chapter 3, Nisenson *et al.* [50] is the only example of one-class classification, and uses a LZ78-based classification model that employs a threshold on the probability. As discussed previously, Gunetti and Picardi's work [33] is not an example of one-class classification because it requires negative data in order to make a prediction.

7.3 Combining Density and Class Probability Estimation

Although there are many successful techniques for one-class classification,[2] a large number of multi-class classification algorithms have been developed and it would be useful to be able to utilize them for one-class problems. However, as we saw in Section 7.1, such methods rely heavily on the presence of negative data in order to perform an accurate predictive model. In cases where novel classes are expected at prediction time, or no negative data is available, one-class classification techniques become preferable. As discussed in the previous section, it is possible to customize existing multi-class classifiers by using a manually set threshold at training time, but doing so creates an inflexible classifier that must be re-trained in order to adjust the prediction threshold.

Fortunately there is a simple way to use multi-class classifiers for one-class classification: artificial data can be generated to take the role of the 'outlier' class. The easiest way to do this is to generate uniformly distributed data or some other distribution and train a classifier that can discriminate this artificial data from the real target instances. This approach has been used by Fan *et al.* [24] and Abe *et al.* [1]. In both cases, artificial instances

[2]For a wider review than that of the previous section, see Tax [70]

were generated dimension by dimension, using either Gaussian models or a uniform distribution. However, as the number of attributes grows, it quickly becomes infeasible to generate enough data to obtain sufficient coverage of the instance space, and the probability that a particular artificial instance occurs inside or close to the target class diminishes to a point that makes any kind of discrimination impossible.

Abe *et al.* [1] solved the dimensionality problem by employing active learning to identify artificial instances that could be considered outliers with respect to the target class. However, an automated system would be preferred because it creates a stable classifier (compared to the approach in Abe *et al.* [1], which is subjective because classifier accuracy varies based on the human input). A potential solution to this problem is to generate artificial data that is as close as possible to the target class.

Generating artificial data that is close to the target class can be performed by fitting a density function to the target data, then using this 'reference density' when randomly generating instances. But because the artificial data is no longer uniform it it necessary to take the reference density into account when calculating the probability that a given instance belongs to the target class. This can be achieved by using Bayes' Rule to combine the multi-class classifier with the density estimate into a single one-class classifier, as shown in the next section and covered in Hempstalk *et al.* [37]. One advantage of this approach is that no modifications need to be made to the multi-class classifier. Another advantage is that this technique essentially straddles the boundary between the two types of classifiers presented in the previous section, namely density functions and class probability estimation techniques.

7.3.1 Combining Classifiers Using Bayes' Rule

Let T denote the target class for which we want to build a one-class model. We have training data for this class. Let A be the artificial class, for which we generate artificial data using a known reference distribution. Let X denote an instance and let $P(X|A)$ denote the density function of the reference density.

What we would like to obtain is $P(X|T)$, the density function for the target class. If we had this density function, we could use it for one-class classification by imposing a threshold on its values. Let us assume for the moment that we know the true class probability function $P(T|X)$. In practice, we need to estimate this function using a class probability estimator learned from the training data. An example of a suitable inductive approach is bagging

of unpruned decision trees, which has been shown to yield good class probability estimators [55]. However, any multi-class classifier that can produce class probability estimates can be used. This is not an impediment in general because most multi-class classifiers do output these values, or can be easily modified to do so.

The following shows how we can compute the density function for T, namely $P(X|T)$, given the class probability function $P(T|X)$, the reference density $P(X|A)$, and $P(T)$, which is the prior probability of observing an instance of the target class. We start with Bayes' theorem:

$$P(T|X) = \frac{P(X|T)P(T)}{P(X)}$$

For a two-class situation, the probability of X is the probability of seeing an instance of X with either class label, so the equation becomes:

$$P(T|X) = \frac{P(X|T)P(T)}{P(X|T)P(T) + P(X|A)P(A)}$$

Now we solve for $P(X|T)$, and make use of the fact that $P(A) = 1 - P(T)$, because there are only two classes:

$$P(X|T) = \frac{(1 - P(T))P(T|X)}{P(T)(1 - P(T|X))}P(X|A) \tag{7.2}$$

This equation relates the density of the artificial class $P(X|A)$ to the density of the target class $P(X|T)$ via the class probability function $P(T|X)$ and the prior probability of the target class $P(T)$.

To use this equation in practice, we choose $P(X|A)$ and generate a user-specified amount of artificial data from it. Each instance in this data receives the class label A. Each instance in the training set for the target class receives class label T. Those two sets of labeled instances are then combined. The proportion of instances belonging to T in this combined dataset is an estimate of $P(T)$, and we can apply a learning algorithm to this two-class dataset to obtain a class probability estimator that takes the role of $P(T|X)$. Assuming we know how to compute the value for $P(X|A)$ given any particular instance X—and we can make sure that this is the case by choosing an appropriate function—we then have all the components to compute an estimate of the target density function $\hat{P}(X|T)$ for any instance X.

7.3.2 Combined Classifier Performance

Now we test the performance of Equation 7.2 for one-class classification, using the same datasets as in Section 7.1.3. In earlier sections comparing one-class and multi-class classification, a single Gaussian played the role of the one-class classifier; in this section a single Gaussian is used for the reference density, $P(X|A)$, and its parameters are estimated by fitting it to the target class to ensure that the artificial data is close to the target class. The role of the class probability estimator, $P(T|X)$, is filled by bagged unpruned C4.5 decision trees with Laplace smoothing, which as previously mentioned has been shown to provide good class probability estimates [55]. Finally the proportion of target instances, $P(T)$, is set to 0.5. Therefore, the data used to build the decision trees is exactly balanced. This setup allows us to simplify Equation 7.2 to:

$$P(X|T) = \frac{P(T|X)}{1 - P(T|X)} P(X|A) \tag{7.3}$$

This simplified equation has been used in the context of association rule learning by Hastie *et al.* [34].

Equation 7.3 is turned into a usable one-class classifier by choosing an appropriate threshold on $\hat{P}(X|T)$ and adjusting the threshold to tune the probability of an instance being identified as the target class. However, the results presented here use the weighted AUC, which is independent of any thresholds.

Table 7.3 presents the weighted AUC (Equation 7.1) for the one-class classifier from the previous section, its components, and two multi-class classifiers (in unbiased two-class configuration) equivalent to the components of the one-class classifier. The two-class classifiers Naïve Bayes and bagged unpruned C4.5 decision trees represent the components $P(X|A)$ and $P(T|X)$ respectively, but in two-class configuration (i.e. learning with negative data). These multi-class classifiers were evaluated using the same methodology as Section 7.1.2. The one-class classifier, One-Class Combined in Table 7.3, and its components $P(X|A)$ and $P(T|X)$, were also evaluated using the methodology developed in Section 7.1.2. In the case of each component, their values were used directly for ranking test cases (instead of the result of Equation 7.3). The component $P(T|X)$ uses the instances artificially generated from the reference distribution for the outlier class; no real negative data is provided.

Not surprisingly, the two-class classifiers outperform the one-class ones on twenty of the twenty-seven datasets. In these datasets there is sufficient nega-

Dataset	Two-Class Naïve Bayes	Two-Class Bagged Trees	One-Class Combined	One-Class $P(X\|A)$	One-Class $P(T\|X)$
anneal	0.605	0.624	0.779	**0.788**	0.531
arrhythmia	0.723	**0.775**	0.576	0.576	0.515
audiology	**0.897**	0.885	0.879	0.881	0.537
autos	0.736	**0.803**	0.565	0.567	0.569
balance-scale	0.851	**0.879**	0.865	0.806	0.708
ecoli	0.889	0.872	**0.927**	**0.927**	0.538
glass	0.605	**0.746**	0.696	0.702	0.615
hypothyroid	0.587	**0.860**	0.648	0.648	0.544
iris	0.671	0.708	**0.977**	**0.977**	0.510
letter	0.935	**0.983**	0.902	0.887	0.773
lymph	0.425	0.490	0.721	**0.739**	0.530
mfeat-factor	**0.964**	0.959	0.948	0.948	0.656
mfeat-fourier	**0.930**	0.914	0.910	0.909	0.534
mfeat-karhunen	**0.976**	0.944	0.956	0.955	0.516
mfeat-morph	0.928	0.906	**0.941**	**0.941**	0.829
mfeat-pixel	**0.965**	0.921	0.954	0.954	0.455
mfeat-zernike	**0.912**	**0.912**	0.898	0.897	0.526
optdigits	0.969	**0.975**	0.958	0.959	0.728
pendigits	0.942	**0.970**	0.958	0.953	0.860
primary-tumor	**0.784**	0.734	0.739	0.732	0.524
segment	0.863	**0.951**	0.937	0.937	0.513
soybean	**0.973**	0.961	0.962	0.961	0.556
splice	**0.831**	0.772	0.721	0.720	0.543
vehicle	0.696	**0.782**	0.684	0.658	0.660
vowel	0.909	**0.922**	0.873	0.865	0.645
waveform	0.692	0.727	0.863	**0.864**	0.342
zoo	0.963	0.913	**0.985**	0.984	0.601

Table 7.3: Weighted AUC for two-class and one-class classifiers

tive data to provide a good coverage for the outlier class. Of the seven datasets where the one-class methods win, only one (*zoo*) is a win for the combined model versus $P(X|A)$. The density function, $P(X|A)$, is often sufficiently accurate to describe the target data.

When considering only the one-class classifiers in Table 7.3, the results are more interesting. For every single dataset, the combined model outperforms the class probability estimator, $P(T|X)$. In contrast, the combined classifier only wins against the density function on twelve of the twenty-seven datasets and only two of the wins (balance-scale, letter) are statistically significant[3] at the 5% level. The vehicle dataset is significant at the 10% significance level.

The density function alone appears to be the best approach to one-class classification; however, none of its wins over the combined classifier are significant, whereas the combined classifier has at least two significant wins over the density function. Combining the classifiers using Equation 7.3 only results in an improvement over the individual components when both are able to add value; in many cases the probability estimator has a low AUC so the combined model can do no better than use the density function alone. Where the com-

[3]Based on using the formula $\frac{1}{\sqrt{4*N}} * 1.96$ for determining the bounds for 5% significance on the AUC. The value 1.96 represents the inverse standard normal distribution for 95% confidence, N represents the number of instances in the dataset. Values are considered significant if they differ by an amount larger than the calculated bounds.

Dataset	$P(A)$								
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
anneal	0.777	0.779	0.777	0.777	0.781	**0.782**	0.781	0.781	0.781
arrhythmia	0.576	0.578	0.575	0.575	0.575	**0.578**	0.575	0.575	0.576
audiology	0.880	0.880	0.880	0.879	0.879	0.880	0.879	0.878	**0.881**
autos	0.566	0.566	**0.567**	0.564	0.564	0.567	0.563	0.565	0.566
balance-scale	0.836	0.850	0.841	0.857	0.861	0.864	0.870	**0.872**	0.863
ecoli	0.927	0.928	0.929	**0.930**	0.928	0.926	0.926	0.929	0.927
glass	0.694	**0.708**	0.691	0.695	0.704	0.700	0.699	0.700	0.697
hypothyroid	0.648	0.649	0.648	0.649	0.649	0.650	0.647	0.651	0.647
iris	0.977	**0.978**	0.976	0.975	0.976	0.976	0.976	0.975	0.975
letter	0.899	0.900	0.900	0.902	0.902	0.903	0.905	0.905	**0.906**
lymph	0.727	**0.729**	0.727	0.724	0.723	0.716	0.705	0.711	0.725
mfeat-factors	0.948	0.948	0.948	0.948	0.948	0.948	0.948	0.948	**0.949**
mfeat-fourier	0.909	0.909	0.909	0.909	0.909	0.909	0.909	0.909	**0.910**
mfeat-karhunen	0.955	0.955	0.955	0.955	0.955	0.955	0.955	0.955	**0.956**
mfeat-morph	**0.941**	0.940	0.940	0.940	0.940	0.940	0.940	0.940	0.940
mfeat-pixel	0.953	0.953	**0.954**	0.953	0.953	0.953	0.953	0.953	0.953
mfeat-zernike	0.898	0.898	0.898	0.898	**0.899**	0.898	0.898	0.897	0.897
optdigits	0.958	0.958	0.958	0.958	0.958	0.958	0.958	0.958	**0.959**
pendigits	0.958	0.958	0.958	0.958	0.958	0.958	**0.959**	0.958	0.958
primary-tumor	0.737	0.733	0.732	0.729	0.737	0.737	0.731	**0.738**	0.737
segment	0.937	0.937	0.937	0.937	0.937	0.937	0.937	0.937	**0.938**
soybean	0.962	0.961	0.962	0.962	0.962	0.962	0.962	0.962	**0.963**
splice	0.720	0.722	0.720	0.719	0.721	0.720	0.720	**0.721**	0.720
vehicle	0.669	0.673	0.680	0.679	0.684	0.687	0.686	0.694	**0.702**
vowel	0.867	0.868	0.869	0.872	0.873	0.876	0.878	0.881	**0.889**
waveform-5000	0.862	**0.863**	0.8621	0.8622	**0.863**	0.862	0.862	0.862	0.861
zoo	0.985	0.984	0.984	0.984	0.984	0.982	**0.985**	0.984	0.984

Table 7.4: Effect of $P(A)$ on the weighted AUC

bined model scores significant wins, the probability estimator performs better than it does in many of the other datasets.

7.3.3 Generating Different Proportions of Artificial Data

Given that the combined model outperforms its components when both components contribute to the model and that it is often the probability estimator that performs poorly, it seems sensible to consider improving this component in order to increase the accuracy of the one-class classifier. One way to possibly improve the probability estimator is to vary the volume of artificial data provided to it.

Table 7.4 shows the results of varying the proportion of artificial data from 0.1 through to 0.9 using the combined one-class classification model from Section 7.3.1. For these experiments the instances were weighted to $1 - P(x)$, where $P(x)$ is the prior probability of the relevant class. For example, if $P(A) = 0.1$ then artificial instances are weighted to 0.9, and target instances are weighted to 0.1. The reason for weighting instances in this way is to balance the dataset.

In general we would expect the combined model to improve as more arti-

ficial data is provided because there is more information for the probability estimator to reason about. However, the model only consistently improves when provided with more data in cases where it is also able to improve over its individual components in the previous section (i.e. letter, vehicle, vowel). In most of the datasets, any improvement made by the probability estimation component is lost because the density function dominates the combined model.

7.3.4 Using Real Data

Clearly the act of adding more data does not improve the probability estimation component enough to make gains over the individual components. Another potential source of improvement is the negative data that we have been forced to ignore for pure one-class classification. The estimate of $P(T|X)$ now becomes the unbiased bagged decision tree estimator from Table 7.3.

Using negative data for training the probability estimator implies that we actually have a proper estimate of the outlier distribution, so instead of estimating $P(X|A)$ from the target data, we should estimate it from the negative data instead. However, the negative data is far from exhaustive—so whilst it might be able to add enough information to help the probability estimator, it may be detrimental to the density estimate and generally produces poor results.

Table 7.5 shows the original combined classifier results alongside those obtained by using negative data. Wins over the two-class classifiers from Table 7.3 are shown in bold. The components using negative data—$P(T|X)$ and $P(X|A)$, their combined model and the combined model for $P(T|X)$ and the original estimate $P(X|A)$ (using only target data) are all shown in the table. For brevity, results have been omitted for combining $P(X|A)$ built with negative data and $P(T|X)$ built with artificial data; this particular combination performs poorly since neither of the components are trained with useful data. Ideally both components should have an exhaustive amount of training data for the outlier class.

Including the negative data into just the probability estimator $P(T|X)$ has a much greater effect on all the results than varying the amount of artificial data as seen in the previous section. The combined model, using either artificial or negative data for the probability estimator and target data to build the density model, now scores ten wins over the two-class classifiers shown in Table 7.3. Although none of the wins by the one-class classifier are statistically significant, the wins scored by the two-class classifiers are not significant either.

Dataset	Standard Combined	Negative $P(T\|X)$ Combined	Negative $P(X\|A)$ & $P(T\|X)$ Combined	Negative $P(T\|X)$ Component	Negative $P(X\|A)$ Component
anneal	0.779	**0.786**	0.681	0.624	0.664
arrhythmia	0.576	0.576	0.565	0.775	0.565
audiology	0.879	0.893	0.603	0.885	0.536
autos	0.565	0.565	0.483	0.803	0.483
balance-scale	0.865	0.862	0.865	0.879	0.609
ecoli	0.927	**0.930**	0.836	0.872	0.738
glass	0.696	0.699	0.689	0.746	0.679
hypothyroid	0.648	0.648	0.546	0.860	0.546
iris	**0.977**	0.974	0.827	0.708	0.584
letter	0.902	0.940	0.658	0.983	0.652
lymphography	0.721	**0.733**	0.696	0.490	0.687
mfeat-factor	0.948	0.948	0.765	0.959	0.764
mfeat-fourier	0.910	0.919	0.632	0.914	0.628
mfeat-karhunen	0.956	0.959	0.691	0.944	0.684
mfeat-morph	0.941	**0.944**	0.651	0.906	0.591
mfeat-pixel	0.954	**0.955**	0.539	0.921	0.511
mfeat-zernike	0.898	0.909	0.643	0.912	0.640
optdigits	0.958	0.961	0.699	0.975	0.690
pendigits	0.958	**0.971**	0.780	0.970	0.717
primary-tumor	0.739	0.766	0.570	0.734	0.491
segment	0.937	0.941	0.772	0.951	0.734
soybean	0.962	0.968	0.609	0.961	0.515
splice	0.721	**0.836**	0.514	0.772	0.427
vehicle	0.684	0.695	0.584	0.782	0.550
vowel	0.873	0.896	0.606	0.922	0.589
waveform	**0.863**	0.861	0.704	0.727	0.565
zoo	**0.985**	**0.985**	0.551	0.913	0.472

Table 7.5: Results of using real negative data in one-class classifiers

When negative data is used to build the density model instead of estimating it from the target data, the combined classifier as well as $P(X|A)$ alone perform poorly. This can easily be explained: the negative data that is used to build the model is not complete and simply does not include enough information to be able to accurately discriminate between classes. It is possible that *some* negative data might be enough to improve the accuracy of $P(X|A)$ over estimating it from the target class. However, in all datasets tested here, estimating the density from the target data is more accurate (and usually significantly so). In fact, the results for at least half of the datasets show that using fake negative data for training the probability estimator performs significantly better than using real negative data to build the density model!

7.3.5 Replacing Components

The last section exchanged the one-class components for two-class ones by making use of negative data. However, it is not necessary to restrict such substitutions to types of components that only differ because they are trained with different datasets. For instance, the density function can be more complicated, such as a mixture of Gaussians, or simpler, such as a uniform distribution. Both of these functions were tested whilst investigating the combined model

one-class classification technique. In general the uniform distribution performs poorly. Using a mixture of Gaussians (EM algorithm from WEKA [77]) achieves more accurate density estimates for some datasets, but worse results on others. Full results of using other density functions can be found in Hempstalk *et al.* [37].

It is also possible to use probability estimation techniques other than bagged unpruned decision trees. Support vector machines (built with logistic models) are an example of a probability estimation model that can be used for $P(T|X)$. Incidentally, support vector machines are also a machine learning algorithm that can be modified to perform one-class classification, so it would be interesting to test the combined model against such a classifier. However, investigating this is outside the scope of this thesis—which focuses on typist verification. Hempstalk *et al.* [37] provides experimental results for using different components and also provides a preliminary comparison of the combined model to SVM-based one-class classification.

7.4 Summary

The results presented in this chapter support the conclusion that a general purpose one-class classifier can be used in place of a two-class classifier without compromising accuracy. Where negative data is available during the training phase, it can be provided to the probability estimation component in a combined model to potentially improve discriminatory power. Although using negative data means that a two-class model is actually being used, in most cases satisfactory results can be achieved with a one-class classifier that combines two different sorts of one-class classification—namely a density function and a probability estimator trained with additional artificial data.

One-class classification is an appropriate approach for typist verification because it means that no data needs to be collected for any user other than the target. It also solves the problem of having a non-exhaustive dataset—one-class classification can be employed to prevent novel users from accessing the system. Two-class classification can be used to increase accuracy, but is best reserved for situations where no novel users are expected, because there is always the potential for an unseen attacker to appear closer to the target user than any known attackers. The next chapter covers typist verification as a general one-class classification problem, using the method presented in Section 7.3 of this chapter as the underlying learning algorithm.

Chapter 8
Typist Verification as One-Class Classification

Continuous typist verification is typically performed by classification algorithms that are designed to handle typing data and nothing else. To use general algorithms instead of customized ones, the stream of key events and times must be transformed into a set of features: general machine learning algorithms—including the one-class classifier presented in Chapter 7—cannot handle an input that is a stream of symbols. It is hard to ensure that the features obtained from a typing stream are helpful for verification, but it is also hard to create a classifier customized to typist verification. It is unclear which of the two approaches—customized classifiers and general classifiers—is a better approach to typist verification, regardless of the effort involved. Section 8.1 addresses this by considering continuous typist verification as a general one-class classification problem, and comparing the results with the more traditional approaches that have been discussed in Chapters 2, 3 and 5.

One advantage of regarding typist verification as a general one-class problem is that it is easy to include additional features, such as mouse use, to increase the accuracy of the system. So far, mouse input has been ignored, even though it was recorded as part of the SquirrelMail data from Chapter 4. When the `sm-150` and `sm-all` datasets were produced the mouse events were removed, and the evaluation in Chapters 4, 5 and 6 relied only on keyboard events. The methods investigated in earlier chapters could not use this data. Mouse events are more complicated than key events—the action performed by the mouse depends on the pointer position—and this information is not easily converted into symbols, digraphs or other structures that are used with traditional typist verification approaches.

When typist verification is transformed into a general one-class classification problem mouse information can easily be included, and Section 8.2 proposes ways to do this. First, an analysis of mouse use is performed. Next, the results of this analysis are used to show that mouse features can be use-

ful for verification. Lastly, the features are added into the continuous typist verification algorithm presented in Section 8.1, and their effect evaluated.

8.1 Typing and One-Class Classification

Most of the techniques presented in the previous chapters use some kind of one-class classification. With the exception of KAOFT from Chapter 3 and the Individual Digraph Classifier from Chapter 5, none of the verification systems we have examined require negative data. In all cases the classifiers are customized to the task of typist verification, and cannot operate on other sorts of one-class problems without modification. In contrast, the general combined one-class classifier presented in the previous chapter can handle many different types of one-class problems. To utilise it for typist verification the stream of events and times must be converted into a set of features. This is not an impediment in general because most techniques examined in this thesis also require typing data to be transformed in some way.

The difficulty of using a general one-class classifier is that the set of defined features needs to describe aspects of a user's behaviour that distinguishes it from impostors. Chapter 6 provides a good starting point, but also indicates that some features perform well for some users and poorly for others. Determining in advance whether a particular feature is helpful for a specific user requires negative data, which ideally is avoided. However, the analysis in Chapter 6 showed that some features are effective for all users, regardless of their performance. These features are the most desirable because of their consistency; using features that are consistently good ensures the system will achieve similar results for all users.

Once a good set of features has been established, the combined one-class classifier expressed in Equation 7.2 from Chapter 7 (hereafter referred to as the Combined OCC) can be trained and used to verify previously unseen typing samples. The methodology used to train and test the classifier is described in the next section, followed by the set of features used during training. The results of evaluating the features and Combined OCC with the `sm` datasets are presented in Section 8.1.3.

8.1.1 Methodology

The Combined OCC was originally implemented in the Java language for the WEKA Machine Learning Workbench. Rather than attempting to modify the

classifier to work for the same typist workbench that has been used for all evaluation in this thesis, a wrapper was written that transformed typing input into a set of features in the WEKA Instances format and passed these onto the WEKA implementation of the Combined OCC. Any samples requiring verification were transformed into the same set of features that were used to train the Combined OCC model. The Combined OCC used bagged C4.5 decision trees for $P(T|X)$ and a Gaussian distribution for $P(X|A)$.

Although the Combined OCC is a one-class classifier, and can adjust its threshold at prediction time, the WEKA implementation has a manually set threshold parameter, known as the Target Rejection Rate. During training, instances are held out and tested against a leave-one-out model. After accruing a set of scores from the held out instances, the threshold is set to the score that ensured that the Target Rejection Rate would be met. For example, a Target Rejection Rate of 0.1 would cause at most 10% of the training instances to fail verification. Because of this, the FRR of any typist verification system utilising the Combined OCC is approximately the same as the Target Rejection Rate parameter.

It is possible to ignore the yes/no predictions produced by the Combined OCC and instead use the raw probabilities and set a threshold in some other fashion, but this was not attempted here. The Target Rejection Rate was left at the default parameter of 0.1. The main value used to evaluate the system was the AUC, which is independent of arbitrarily set thresholds. The raw probabilities were used to calculate the AUC, not the yes/no predictions because these would only produce a single point on the ROC curve. All evaluation of the Combined OCC is performed using ten-times ten-fold stratified cross-validation, with the same settings as in previous chapters. The folds are identical to those used to evaluate other algorithms in this thesis.

8.1.2 Features

Chapter 6 defines a number of attributes describing typist behaviour and investigates their discriminatory performance for users in the `sm-all` dataset. An initial set of features was produced using the highest performing of these attributes: those with overall AUC above 0.600, or per-user AUC values that were always above 0.500. The attributes meeting this criterion were:

- **Average Tap Time** The average absolute time taken to perform digraphs that are "taps", that is, formed from two identical keys.

- **Slur Rate** The proportion of press events that are not immediately followed by the release event for the same key.
- **Slur Length** The average number of events that occur between a press event and its related release event.
- **Press Before Release** The proportion of occurrences where a single press event occurs between the press and release event for another key: that is, $(P_1P_2R_1R_2/P_1P_2R_xR_y)$, where R_x and R_y are release events that are not necessarily from the same keys as the first two press events.
- **Paired Perfect Order** The proportion of occurrences of a "perfect" ordered pair of key events: that is, $(P_1R_1P_2R_2/P_1R_xP_2R_y)$, where R_x and R_y are release events that are not necessarily from the same keys as the first two press events.
- **WPM** The average speed of the entire sample, measured in words per minute.
- **Backspace Rate** The proportion of `Backspace` events in the sample.

The digraph `in` was excluded from this list of attributes, even though it met the above criteria. This digraph was not present in every sample, and would have a missing value for many occurrences. Missing values are not an issue for the Combined OCC, but this digraph was omitted anyway to ensure that classification was not dependent on the existence of a particular typed sequence. Of course, it is possible that a component from one of the selected features does not occur in a particular sample, but this was not the case for the `sm` datasets. None of the above features were missing from any sample.

A number of other features were also considered, including:

- **Average Backspace Block Length** The average length of sets of consecutive `Backspace` key presses.
- **Average Press Block Length** The average length of sets of consecutive press events.
- **Disorder** The disorder between this sample and a reference sample (see Chapter 3. The reference sample was obtained using the average digraph times from the KAOFT dataset.
- **Pause After Space** The average time of a digraph where the first character is the `Space` key, and the second character may be any key.

- **Pause Before Space** The average time of a digraph where the second character is the `Space` key, and the first character may be any key.

- **Peak WPM Rate** The top speed achieved over a window of 20 consecutive events. Twenty events correspond to approximately two words of typed text, using an average word length of 5 characters.

- **Use of x Navigation Key** The proportion of events in the sample that are from the x navigational key, where x is one of the following keys: `Left-Arrow`, `Right-Arrow`, `Up-Arrow`, `Down-Arrow`, `Home`, `End`, `Page-Up` and `Page-Down`. Each key x is a different feature.

- **Average Tap/Trill/Reach/Hurdle/Rock/Opposite Time** The average time for a given classification of finger movement. Each movement type is a different feature.

- **Average Left Hand/Right Hand Time** The average key duration for keys typed with the right hand or left hand. Each hand is a different feature.

- **Average Index/Middle/Ring/Little Finger Time** The average key duration for keys typed with a given finger. Each finger is a different feature.

- **Average H-F Hand-Finger Time** The average key duration for keys typed with a particular hand H and finger F combination. Each H-F combination is a different feature.

The features above had some discriminatory power, but did not significantly increase the performance of the Combined OCC over the first seven features in this section (Average Tap Time, Slur Rate, Slur Length, Press Before Release, Paired Perfect Order, WPM, Backspace Rate). Many other features were also investigated but discarded immediately because of poor results.

8.1.3 Results

Using the wrapper class in the typing workbench, the Combined OCC and feature set described in Section 8.1.2 were evaluated using ten-times ten-fold stratified cross-validation, in an identical manner to all other algorithms in this thesis. The results are presented in Table 8.1, along with the results on the same datasets for the state of the art—LTIAT and KAOFT—from Chapter 4.

Algorithm / Dataset	FRR	IPR	Weighted AUC
`sm-150`			
Combined OCC	36.06	2.70	0.945
LTIAT - $m = 0$, $s = 0$, $Q = 100$	3.53	29.33	0.950
KAOFT - R_2A_2	6.06	0.00	0.970
`sm-all`			
Combined OCC	12.39	4.13	0.975
LTIAT - $m = 0$, $s = 0$, $Q = 100$	0.75	16.65	0.976
KAOFT - R_2A_2	3.60	0.00	0.982

Table 8.1: Final results for the Combined OCC typist classifier

On the smaller `sm-150` dataset, the Combined OCC performs slightly worse than both KAOFT and LTIAT, but better than all but the PPM Classifier from Chapter 5. This is not surprising because these other algorithms always use a large representation containing often hundreds of features. In contrast, the Combined OCC uses just seven values to represent a sample. On the `sm-all` dataset the Combined OCC has a performance equalling the other methods, which is somewhat remarkable considering the small sample representation. A likely explanation for this performance increase is that the Combined OCC needs more than 15 training samples per user to make confident predictions because the users are not consistent enough that fewer samples can be used form a representative profile of their behaviour.

One advantage that the Combined OCC has over the other methods is speed. Both KAOFT and LTIAT take more than 24 hours of processing to complete a full evaluation[1] on the `sm-all` dataset, whereas the Combined OCC takes less than 10 minutes to perform the same evaluation, including feature extraction.[2] Despite this, the Combined OCC does not suffer from any loss of accuracy—it can build a model and accurately perform predictions in a fraction of the time of other algorithms. This ability can be attributed to the compact but efficient data representation; the chosen features describe a typist's behaviour sufficiently well that no more than seven features are needed to rival the LTIAT and KAOFT methods in terms of accuracy (measured with the weighted AUC) and surpass them in terms of speed.

Another advantage lies in the use of one-class classification. The Combined OCC does not need any negative data in order to verify a user, meaning that performance is not dependent on the set of users that the system is trained with. Although the use of negative data is not necessarily a drawback since in many cases it is available anyway, it is an unnecessary aspect of this algorithm, at least in this application.

[1] A full evaluation is one ten-times ten-fold cross-validation run for each user in the dataset.

[2] No parameter optimization was performed for any algorithm.

Sample Length	FRR	IPR	Weighted AUC
Full	12.39	4.13	0.975
500 Events	10.21	9.90	0.962
400 Events	9.46	13.82	0.954
300 Events	9.78	18.10	0.935
200 Events	10.01	24.43	0.905
100 Events	9.89	39.27	0.842
50 Events	9.79	51.24	0.770

Table 8.2: Results using different sample lengths

One drawback of other techniques is that many require a sample to contain a large number of events before they are able to make a confident prediction about identity. It is unclear whether the Combined OCC also suffers from this problem, so further evaluation was performed on the `sm-all` dataset. To test different sample lengths, each was divided up into a set number of events, in the same manner as for the LTIAT algorithm in Chapter 3. No sample was included that had fewer than the required number of events. Starting at a limit of exactly 500 events, the sample length was reduced in increments of 100 events. All samples in the `sm-all` dataset contain at least 500 events, and in most cases they are much larger than this.

Table 8.2 contains the results of evaluating the Combined OCC with six different sample lengths, from 500 to 50 events. As the length is reduced, the number of samples increases, but the calculated features become less meaningful and as a result the weighted AUC decreases. Nonetheless, until the sample length is reduced to 200 events the system still outperforms all algorithms from Chapter 5, except the PPM technique.

8.2 Adding Mouse Patterns

Most personal computers have at least two input devices, a keyboard and a mouse. For typist verification, only the former is considered. However, normal computer operation often involves input from the mouse as well, especially for highlighting and editing text. The SquirrelMail data described in Chapter 4 recorded mouse events as well as keyboard ones, but so far this data has been ignored. One reason is that most of the techniques were designed to work only with keyboard data, and could not make sense of the mouse events. A mouse event is any action occurring with the mouse—it can be press and release events on the buttons, scroll events on the centre button, or pointer movements. In the `sm` datasets, only mouse button press as well as mouse button release events were recorded, along with the on-screen position of the pointer when the event occurred.

Only some of the recorded mouse events may have an effect on the email samples from SquirrelMail. If a press event occurs outside the bounds of any of the text boxes, all of them are deselected and subsequent events have no effect on the email until at least one text box is selected again. When a left mouse click occurs inside a text box, the caret is repositioned to where the button is released. If the pointer is dragged across the screen whilst the left button is down, all text between the press and release positions is highlighted. Left mouse button presses followed by key press events can therefore have a varied effect on the email, depending on whether or not it has occurred within the bounds of a text box.

Unfortunately, the mouse data recorded in the `sm` datasets is flawed in a way that makes it difficult to use for mouse-user verification: the position of the pointer was recorded relative to the email web page, but how much the page was scrolled and the user's screen resolution were never recorded. Furthermore, each Internet browser used to access SquirrelMail displayed the page slightly differently. A browser identifier was recorded, but was of little help because the screen resolution also affects how the browser displays the page. Because of this, it is impossible to determine what the user was clicking on. This is a problem for verification because the mouse events are ambiguous: it is possible to tell that the mouse was used, but not what it was used for. In contrast, key events are explicit—they always produce the same character.

Another issue with the `sm` mouse data is that only button presses and releases were recorded, but mouse activity consists of many other actions. It is unclear how far the mouse moved between each event because the complete movement was never tracked. This information could have been recorded, but the data was originally intended for only keystroke-based verification, and recording all possible mouse events was considered unnecessary. In retrospect, the `sm` datasets would be more valuable had this information been included. Nevertheless, it is still possible to make use of the mouse events to some extent.

8.2.1 Features

Since the mouse pointer position is not accurately recorded, it seems pointless to attempt to use it for verification. Presses and releases were the only reliable mouse events in the `sm` data, along with a button identifier and the time of the event. A simple way of using this data for verification is to calculate the proportion of mouse events in the total number of events—including both key and mouse events. This "Mouse Use" feature can then be provided to the

Participant	Mouse Use
A	0.742
B	0.680
C	0.612
D	0.743
E	0.694
F	0.682
G	0.747
H	0.656
I	0.699
J	0.901
Average	0.716

Table 8.3: AUC values for the Mouse Use feature

same classifier used in the previous section. It gives an indication of how often a typist uses their mouse, although not an entirely accurate one because only clicks were recorded.

In Chapter 6 the usefulness of an attribute was analysed using the AUC. The same idea is applied here in Table 8.3 for the Mouse Use feature defined above. On average, this feature performs similarly to the best digraph feature (`in`) from Chapter 6. It is not more effective for verification than the typist's speed, but is nevertheless a helpful feature, and scores a higher AUC for all users than many individual digraphs do.

Since general use of the mouse is helpful, it might be worth considering when the mouse is used. However, the mouse is often used infrequently—sometimes not at all—so pairing occurrences of mouse events with specific key events in a mouse-keyboard digraph will result in missing values for most combinations. This does not mean that all combinations are useless: pairing mouse events with certain key events may be helpful for verification. An obvious choice is the mouse-keyboard digraph `Mouse Backspace` because in many cases it indicates that the user has repositioned the pointer before deleting some text. This digraph gives information about a user's editing behaviour, under the assumption that the pointer is indeed being repositioned. The `Backspace` key is also the most common key in the `sm` datasets, so occurrences of this feature will likely exist for most users.

One problem with the `Mouse Backspace` digraph is that the associated times are inconsistent. There are often large pauses between using the mouse and the `Backspace` key, perhaps caused by the two events being actioned by the same hand. This would be expected if the user is right-handed and operates the mouse with their right hand. As it happens, all of the users in the `sm` datasets were right handed and had their mouse positioned to the right of their keyboard—although it is impossible to know whether they used the same hand for these two events because the users were not actively monitored. Nonethe-

Participant	Mouse Backspace
A	0.650
B	0.536
C	0.508
D	0.531
E	0.649
F	0.640
G	0.588
H	0.568
I	0.583
J	0.538
Average	0.579

Table 8.4: AUC values for the Mouse Backspace feature

Algorithm / Dataset	FRR	IPR	Weighted AUC
`sm-all`			
Combined OCC—Mouse Use	13.28	4.28	0.975
Combined OCC—Original	12.39	4.13	0.975
LTIAT—$m = 0$, $s = 0$, $Q = 100$	0.75	16.65	0.976
KAOFT—R_2A_2	3.60	0.00	0.982

Table 8.5: Results for Mouse Use and the Combined OCC Typist Classifier

less, the varied pausing is unhelpful for verification, and instead the proportion of `Mouse Backspace` digraphs against the total number of `Backspace` events was used.

Table 8.4 shows the results for evaluating the Mouse Backspace feature. Although `Backspace` was the most common key, `Mouse Backspace` was absent from most samples—even for users who were prolific mouse users. As a result, the Mouse Backspace feature often had a value of zero, and was unhelpful in most cases. It is not completely without merit: in Chapter 6 it was found that the usefulness of any measure was often dependent on the user, and this is echoed in the results here. But as in Chapter 6, without negative data it is impossible to tell in advance if this feature is helpful for a particular user.

8.2.2 Results

Using the same seven features from Section 8.1.2, and the Mouse Use feature defined in the previous section, the Combined OCC was re-evaluated using the `sm-all` dataset. The `sm-150` could not be utilised for this evaluation; when this dataset was anonymised it was also reduced to contain only keyboard events. Unfortunately, the identities of the emails that were selected for the `sm-150` dataset were never kept, and it is difficult to determine which emails they are sourced from. However all the `sm-all` emails were kept, so the `sm-all` dataset was reproduced for this evaluation complete with mouse events.

Table 8.5 presents the results of evaluating the Combined OCC with the additional Mouse Use feature. The FRR and IPR of the system increased

slightly when the feature was added, but the AUC remained the same. This was a disappointing result considering that the average AUC rivalled that of the other seven features used to train the system. If the system were trained with more users, or a different range of users, this feature might be more helpful in increasing the accuracy of the system. Note that adding it did not noticeably impact running time; the Combined OCC is still many orders of magnitude faster than other algorithms.

8.3 Summary

This chapter has presented a new approach to continuous typist verification: representing a sample of text as a set of features and providing this to a general one-class classifier (rather than a customized one). This achieves a level of accuracy that compares to the state of the art, performing on par with LTIAT and KAOFT.[3] The Combined OCC has one clear advantage: it takes only 50ms to train the system and less than 5ms to make predictions on average—thoroughly eclipsing the other techniques in terms of speed without compromising accuracy.

Another advantage of the Combined OCC is that it is easy to add new features, as shown in Section 8.2. With little effort, mouse events are analysed, turned into a feature and used for classification. Although in this case the accuracy of the system is not increased, there is still opportunity for other features to be added that might boost the prediction power—even if only for particular users. Many features were used to test the Combined OCC, but in the end a mere seven were found to be sufficient to achieve an accuracy similar to LTIAT and KAOFT.

Finally, because the Combined OCC uses one-class classification, it is not dependent on negative data. This is yet another advantage over methods such as KAOFT, which can be more accurate, but requires a representative set of users in order to achieve this. In contrast, the Combined OCC is dependent only on data from the target user, yet obtains a similar accuracy to KAOFT on the data used for testing.

[3]Using a conservative estimate of AUC significance [18], the difference between these three algorithms is not significant at the 5% level for the `sm-all` dataset.

Chapter 9
Conclusions

This thesis has presented an investigation into continuous typist verification—the problem of continually checking that a computer user is who they claim to be by analyzing the way they type. This task is not easy: many factors influence the performance of a continuous typist verification system. Existing systems such as LTIAT and KAOFT determine identity by utilising timing patterns. These systems are able to achieve a commendable level of accuracy, but also suffer performance issues. In the case of KAOFT, the current state-of-the-art technique in terms of accuracy, negative data is required to build the verifier. Although in many cases this negative data is available, the accuracy of the system is dependent on it; the system performs well with this data—especially when it contains a representative set of possible impostors—but fails completely when trained only with data from the target user.

The published accuracy of LTIAT and KAOFT suggests that the hypothesis stated in Chapter 1 might be true, that there is enough information in a user's typing input for continuous typist verification to be a useful form of biometric authentication. However, when these techniques were re-implemented and re-evaluated in Chapter 3 it became clear that although they perform well on their own datasets, they are only able to achieve good results in restricted circumstances. For LTIAT, both key press and release events are essential, and KAOFT requires large samples and a representative set of attackers during training.

This thesis explores whether it is possible to perform typist verification in less restricted situations, such as those where no negative data is used during training. The aim of this exploration is to determine to what extent the information in typing input is useful for verification. There are several aspects that need to be considered as part of this, but the primary concern is the ability to identify channels of information that are available in typing input. Since the existing systems were evaluated with datasets that are deficient in some way, either in size or the type of events they contain, the first step towards achieving this was the collection of suitable data.

9.1 Collecting Data

Data collection is important for this thesis because it enables algorithms to be tested experimentally in a fair and uniform way. The datasets that were originally used to evaluate LTIAT and KAOFT cannot be used to fairly test algorithms because for the former the dataset is small, and for the latter key release events are missing. Such events are utilised for verification by LTIAT, and without them this method is unable achieve state-of-the-art accuracy. Other new methods may also be able to exploit key release events to perform verification.

Chapter 4 discusses the collection of typing input from real-world emails. Almost 3000 emails were collected over a period of 3 months, and processed into two final datasets with 150 email samples (`sm-150`) and 607 email samples (`sm-all`) respectively. The original emails are affected by some technical and ethical issues that meant that not all could be utilised in the final datasets. The most common problem was the result of a bug in the Internet browser used to access the email system, but other issues included confidential information being revealed, emails typed in a foreign language, sample length less than a stipulated minimum and inconsistencies in recording modifier keys uniformly across all Internet browsers utilised. It is likely that any typist verification system that operates on free text will endure similar issues, so explaining them is helpful to others.

The two datasets, `sm-150` and `sm-all`, are comparable in size to KAOFT's dataset and in content to LTIAT's dataset: they consist of at least 15 emails per user and contain key press events, key release events, and their associated timings. Both were used to re-evaluate LTIAT and KAOFT in Chapter 4, and similar results were achieved to the original algorithms matched with their original datasets from Chapter 3. The results indicate that the new datasets enable a fair comparison between new and existing methods of typist verification because they provide all the data necessary for evaluation.

Mouse events were also recorded for all the 3000 emails that were collected. These are not used for evaluation except for the combined one-class classifier (Combined OCC) technique in Chapter 8. The collected emails make several contributions to typist verification. They provide a fair way to evaluate different typist verification methods; they give insights into issues associated with collecting keyboard (and mouse) input; and they contain additional information that can be used to assist existing verification techniques.

9.2 Identifying Channels of Information

Now that two new datasets are available that address the shortcomings of the existing ones, it is possible to investigate what channels of information are available in typing input and quantify how useful each is for verification. Chapter 5 contains an exploratory investigation using probability-based algorithms. This gives insight into what channels of information exist. Algorithms that use small structures such as digraphs perform well but take a long time to produce predictions because they need to make a large number of comparisons.

Chapter 6 uses the `sm-all` dataset to investigate individual aspects of typing and quantify their usefulness for verification. Features that are considered include individual digraphs, finger movements, usage of editing and navigational keys, pausing behaviour, ordering of events and basic measures such as typing speed and error rate. These features provide different amounts of information that can be used to verify identity. Simple metrics like the typist's speed and backspace rate perform best, whereas individual digraphs are often poor at distinguishing between users. This might seem surprising given that individual digraphs are often used in typist verification systems. However, the samples in the `sm` datasets contain free text, so it is possible that the user is unfamiliar—and hence inconsistent—with many of the sequences they type. Using aggregates such as typing speed smoothes over noise, and renders the outcome less susceptible to small inconsistencies within and between samples.

The process of typing on a computer often involves other input devices, such as a mouse, which inevitably affects the patterns of entry. For example, the mouse can be used to reposition the caret in a block of text, causing a delay when the user moves their hand from the keyboard to the mouse and back again. Traditionally, mouse events are not considered in typist verification systems, but because they are something that affects typing behaviour, they can potentially be a viable channel of information. In Chapter 8 mouse events were investigated for verification by adding a single feature called Mouse Use to an existing system. Using the same analysis as in Chapter 6 it was found that this feature considered independently was useful for verification, but gave no noticeable change in accuracy when added to the Combined OCC.

One important insight gained from investigating the channels of information is that performance varies on a per-user basis. Although some features performed better than others for all users, those that performed poorly overall often achieved a reasonable level of discrimination for particular users. The difficulty with using these attributes for verification is that it is impossible to

determine in advance whether or not a particular feature will be useful without utilising negative data.

9.3 Requirements for System Training

In principle, negative data should be unnecessary for any verification system because there should be no need to compare to other people in order to confirm someone's identity. Chapter 7 addressed this directly by testing whether one-class classifiers can outperform multi-class ones. It was found that when negative data provided to a multi-class classifier was insufficiently representative of possible attackers, a one-class classifier yielded better predictive performance. This suggests that in cases where novel attackers are expected—and especially when they have the potential to differ from existing attackers—one-class classification should be used in preference to multi-class classification. In short, negative data is unnecessary for successful verification.

However, some systems are designed in a way that requires negative data, such as KAOFT in Chapter 3 and the Individual Digraph Classifier in Chapter 5. This is not necessarily a disadvantage because in many cases negative data is available, but performance suffers when only a limited amount is on hand. LTIAT and the Combined OCC show that it is possible to design algorithms that require no negative data and achieve comparable results to those that do. Additionally, when used for typist verification, the Combined OCC produces predictions many orders of magnitude faster than any other classifier discussed in this thesis.

Typist verification systems often require a large number of training samples in order to be able to confidently perform verification. In most cases this is the result of users typing inconsistently: typing data needs to be representative for each user, otherwise it may be falsely rejected for failing to match the training profile. The actual number of training samples depends on the length of each sample and the consistency of each user. For systems evaluated in this thesis the `sm-150` dataset provides reasonable results. This dataset has 15 samples for each of ten users, each sample having a minimum length of 500 key events. In each cross-validation run, between 13 and 14 samples were used to train the user's profile, the remaining samples being used for evaluation.

When the features provided to the Combined OCC were calculated on samples of reduced length it was found that the accuracy degraded. Overall, approximately the same volume of data was available for training; however, samples were divided up into smaller non-overlapping sequences. Many of

the features calculated become more meaningful as the length of a sample is increased, and the results reflect this. For continuous systems, where no restrictions are placed on the text being entered, sample sizes of at least 500 events produce the best results in this thesis.

9.4 Revisiting the Hypothesis

This thesis argues that

> there is enough information in a user's typing input for continuous typist verification to be a useful form of biometric authentication.

As discussed in Chapter 1, whether the level of authentication that is achieved is useful depends on a number of aspects. This thesis addresses them by collecting data, identifying channels of information and investigating the restrictions that must be imposed on verification systems. It also introduces a new method of performing one-class classification, which is then used to perform typist verification to a comparable level of accuracy to—but much faster than—the current state of the art.

The findings of this thesis are encouraging. With good data and well-designed algorithms continuous typist verification can be performed quickly and at a high level of accuracy. We can conclude that there is enough information for continuous typist verification to be used effectively for biometric authentication. However, it is impossible to tell whether these algorithms will be practically useful because no evaluation was performed with users who were interacting with an actual system. Whether or not the trade-off between false rejection rate and impostor pass rate is acceptable depends on what the system is protecting, and no assumptions have been made about what this might be.

9.5 Future Work

In Chapter 6 it was found that features had differing strengths of verification on a per-user basis. Since many techniques utilise only one type of feature, such as digraph time, existing techniques could be combined to form a system that covers several aspects of typing. One possible way of combining techniques is to train several different classifiers with the same data, and order them into a list. During prediction time, a sample must pass through all classifiers in the list to be successfully verified. If any classifier rejects the sample, verification fails and the remaining classifiers do not need to consider the sample. The

reason that this approach might be more appropriate than others such as voting is that impostors can be quickly identified by ordering the classifiers by their speed. Contrast the case of voting, where all classifiers must make a judgement before the system can verify a sample. Depending on the classifiers involved, this might take a long time.

Another avenue of future work would be to test the features from Chapters 6 and 8 with a different one-class classifier. For example, they could be used to train a one-class support vector machine (like that provided by libSVM [11]) instead of the Combined OCC. However, in Hempstalk and Frank [37] it was found that the Combined OCC achieves an accuracy comparable to libSVM's one-class SVM, so it is unlikely that there will be a substantial improvement in accuracy.

An aspect of typist verification that is more likely to improve on the current state of the art is the choice of attributes provided to the Combined OCC. For example, features could be selected on a per-user basis. Some of the features seen in Chapter 6 were not utilised due to poor performance overall, but these might be helpful for particular users. Also, the Combined OCC makes it possible to add new features with ease, so there is potential for accuracy to be increased if another channel of information is identified.

For the last one hundred years the humble keyboard has been used solely to produce printed text. This thesis has shown that it can also be successfully employed for authentication, verifying who people are by the way they type. Computers will begin to recognise us as individuals, just as we do with each other.

References

[1] N. Abe, B. Zadrozny, and J. Langford. Outlier detection by active learning. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 767–772, New York, NY, USA, 2006. ACM Press.

[2] A. Adams and M. Sasse. Users are not the enemy. *Communications of the ACM*, 42(12):41–46, December 1999.

[3] A. Asuncion and D. Newman. UCI machine learning repository, 2007.

[4] V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley & Sons, West Sussex, England, 1994.

[5] R. Begleiter, R. El-Yaniv, and G. Yona. On prediction using variable order Markov models. *Journal of Artificial Intelligence Research*, 22:385–421, 2004.

[6] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression*. Prentice Hall, New Jersey, USA, 1990.

[7] F. Bergadano, D. Gunetti, and C. Picardi. User authentication through keystroke dynamics. *ACM Transactions on Information and System Security*, 5(4):367–397, 2002.

[8] Biopassword, 2006. Publisher website available at: `http://www.biopassword.com` (2006).

[9] S. Bleha, C. Slivinsky, and B. Hussein. Computer-access security systems using keystroke dynamics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):1217–1222, 1990.

[10] P. Buzing. Comparing different keyboard layouts: Aspects of QWERTY, Dvorak and alphabetical keyboards, 2003. Article published online at: `http://pds.twi.tudelft.nl/~buzing/Articles/keyboards.pdf` (2009).

[11] C. Chang and C. Lin. *LIBSVM: A Library for Support Vector Machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[12] W. Chang. *Keystroke Biometric System Using Wavelets*, volume 3832, pages 647–653. Springer, Berlin, Germany, 2005.

[13] S. Cho, C. Han, D. H. Han, and H.-I. Kim. Web-based keystroke dynamics identity verification using neural network. *Journal of Organizational Computing and Electronic Commerce*, 10(4):295–307, 2000.

[14] S. Cho and S. Hwang. *Artificial Rhythms and Cues for Keystroke Dynamics Based Authentication*, volume 3832, pages 626–632. Springer, Berlin, Germany, 2005.

[15] D. Clifton, P. Bannister, and L. Tarassenko. A framework for novelty detection in jet engine vibration data. *Key Engineering Materials*, 347:305–312, 2007.

[16] W. E. Cooper. *Cognitive Aspects of Skilled Typewriting*. Springer-Verlag, New York, USA, 1982.

[17] W. E. Cooper. Introduction. In *Cognitive Aspects of Skilled Typewriting*, chapter 1, pages 1–38. Springer-Verlag, New York, USA, 1982.

[18] C. Cortes and M. Mohri. Confidence intervals for the area under the ROC curve. In *Neural Information Processing Systems 17*, pages 305–312, 2004.

[19] P. Dowland and S. Furnell. A long-term trial of keystroke profiling using digraph, trigraph and keyword latencies. In *Proceedings of the 19th Internation Information Security Conference, SEC 2004*, pages 275–290, 2004.

[20] P. Dowland, S. Furnell, and H. Singh. A preliminary investigation of user authentication using continuous keystroke analysis. In *Proceedings of the International Conference on Information Security Management and Small Systems Security*, pages 215–226, 2001.

[21] D. D'Souza. Typing dynamics biometric authentication. Master's thesis, University of Queensland, Australia, Queensland, Australia, 2002.

[22] P. Duke. *Cluster-Based Methods for Novelty Detection*, 2004.

[23] A. Dvorak, N. Merrick, W. Dealey, and G. Ford. *Typewriting Behavior*. American Book Company, New York, USA, 1936.

[24] W. Fan, M. Miller, S. J. Stolfo, W. Lee, and P. K. Chan. Using artificial anomalies to detect unknown and known network intrusions. In *Proceedings of the First International IEEE Conference on Data Mining*, pages 123–130, London, England, 2001. IEEE.

[25] P. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, June 1954.

[26] P. M. Fitts and M. I. Posner. *Human Performance*. Brooks/Cole Publishing Company, Belmont, CA, USA, 1967.

[27] L. Friedrichs. Blueproximity, 2008. Software available at: `http://sourceforge.net/projects/blueproximity/`.

[28] R. Gaines, W. Lisowski, S. Press, and N. Shapiro. Authentication by keystroke timing: Some preliminary results. Technical report, Rand Corporation, Santa Monica, USA, 1980.

[29] A. B. Gardner, A. M. Krieger, G. Vachtsevanos, and B. Litt. One-class novelty detection for seizure analysis from intracranial eeg. *Journal of Machine Learning Research*, 7:1025–1044, 2006.

[30] D. Gentner. Keystroke timing in transcription typing. In *Cognitive Aspects of Skilled Typewriting*, chapter 5, pages 95–120. Springer-Verlag, New York, USA, 1982.

[31] D. Gentner, J. Grudin, S. Laroschelle, D. Norman, and D. Rumelhart. A glossary of terms including a classification of typing errors. In *Cognitive Aspects of Skilled Typewriting*, chapter 2, pages 39–43. Springer-Verlag, New York, USA, 1982.

[32] Government Communications Security Bureau. New Zealand security of information technology publication 204: Authentication services and mechanisms. Technical report, New Zealand Government, Wellington, New Zealand, 2005.

[33] D. Gunetti and C. Picardi. Keystroke analysis of free text. *ACM Transactions on Information and System Security*, 8(3):312–347, 2005.

[34] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer-Verlag, New York, USA, 2001.

[35] K. Hempstalk. The great keyboard debate: QWERTY vs. Dvorak. In *Proceedings of the New Zealand Computer Science Research Students Conference*, page np, Hamilton, New Zealand, 2007. University of Waikato.

[36] K. Hempstalk and E. Frank. Discriminating against new classes: one-class vs. multi-class classification. In *Proceedings of Australian Joint Conference on Artificial Intelligence 2008*, pages 325–336, Berlin, 2008. Springer.

[37] K. Hempstalk, E. Frank, and I. H. Witten. One-class classification by combining density and class probability estimation. In *Proceedings of ECML PKDD 2008, Part I, LNAI 5211*, pages 505–519, Berlin, 2008. Springer.

[38] S. Hocquet, J. Ramel, and H. Carbot. Estimation of user specific parameters in one-class problems. In *Proceedings of the 18th International Conference on Pattern Recognition*, pages 449–452, Washington, DC, USA, 2006. IEEE Computer Society.

[39] A. K. Jain, R. Bolle, and S. Pankanti. *Biometrics: Personal Identification in a Networked Society*. Kluwer, Boston, USA, 1999.

[40] R. Joyce and G. Gupta. Identity authentication based on keystroke latencies. *Communications of the ACM*, 33(2):168–176, 1990.

[41] D. Klein. Foiling the cracker: A survey of, and improvements to, password security. In *Proceedings of the Second USENIX Security Workshop*, London, England, 1990.

[42] M. Koppel and J. Schler. Authorship verification as a one-class classification problem. In *Proceedings of the 21st International Conference on Machine Learning*, pages 489–495, New York, NY, USA, 2004. ACM Press.

[43] G. Langdon. A note on the Ziv-Lempel model for compressing individual sequences. *IEEE Transactions on Information Theory*, 29(2):284–287, 1983.

[44] E. Lau, X. Liu, C. Xiao, and X. Yu. Enhanced user authentication through keystroke biometrics, 2004.

[45] B. Lowagie and P. Soares. iText Java PDF library, 2009. Software available at: `http://www.lowagie.com/iText` (2009).

[46] L. M. Manevitz, M. Yousef, N. Cristianini, J. Shawe-Taylor, and B. Williamson. One-class SVMs for document classification. *Journal of Machine Learning Research*, 2:139–154, 2001.

[47] F. Monrose, M. Reiter, and S. Wetzel. Password hardening based on keystroke dynamics. *International Journal on Information Security*, pages 69–83, 2002.

[48] F. Monrose and A. Rubin. Authentication via keystroke dynamics. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 48–56, New York, USA, 1997. ACM.

[49] F. Monrose and A. Rubin. Keystroke dynamics as a biometric for authentication. *Future Generation Computer Systems*, 16(4):351–359, 2000.

[50] M. Nisenson, I. Yariv, R. El-Yaniv, and R. Meir. Towards behaviometric security systems: Learning to identify a typist. In *International Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 363–374, Berlin, Germany, 2003. Springer-Verlag.

[51] D. A. Norman and D. E. Rumelhart. Studies of typing from the LNR research group. In *Cognitive Aspects of Skilled Typewriting*, chapter 3, pages 45–66. Springer-Verlag, New York, USA, 1982.

[52] C. Ong and W. Lai. Enhanced password authentication through typing biometrics with the k-means clustering algorithm. In *Proceedings of the Seventh International Symposium on Manufacturing with Applications*, Hawaii, USA, 2000.

[53] A. Peacock, K. Ke, and M. Wilkerson. Typing patterns: A key to user identification. *IEEE Computer Society, Security and Privacy Journal*, pages 40–47, September/October 2004.

[54] R. Pearson. *Mining Imperfect Data*. Society for Industrial and Applied Mechanics, USA, 2005.

[55] F. Provost and P. Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.

[56] Psylock, 2009. Publisher website available at: http://psylock.com/ (2009).

[57] Psylock security, 2009. Electronic document available at: http://psylock.com/index.php/lang-en/technology/security (2009).

[58] K. Revett, S. Magalhaes, and H. Santos. *Enhancing Login Security Through the Use of Keystroke Input Dynamics*, volume 3832, pages 661–667. Springer, Berlin, Germany, 2005.

[59] M. Robson. Bug 301089—pressing a key while another key held down does not generate keydown event, 2005. Documented software bug available at: https://bugzilla.mozilla.org/show_bug.cgi?id=301089.

[60] R. Rodrigues, G. Yared, C. Costa, J. Yabu-Uti, F. Violaro, and L. Ling. *Biometric Access Control Through Numerical Keyboards Based on Keystroke Dynamics*, volume 3832, pages 640–646. Springer, Berlin, Germany, 2005.

[61] V. Roth. Kernel Fisher discriminants for outlier detection. *Neural Computing*, 18(4):942–960, 2006.

[62] A. Savage and J. Hyneman. Mythbusters episode 59, 2006. Television Series.

[63] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt. Support vector method for novelty detection. In *Advances in Neural Information Processing Systems 12*, pages 582–588, USA, 2000. MIT Press.

[64] C. L. Sholes. Typewriting machine. United States Patent 207,559, August 1878.

[65] C. L. Sholes, C. Glidden, and S. Soule. Typewriting machine. United States Patent 79,868, July 1868.

[66] T. Smith. Morse: The end of an era? *The UNESCO Courier*, 52(7):65–68, 1999.

[67] D. Song, P. Venable, and A. Perreg. User recognition by keystroke latency pattern analysis. Technical report, 1997.

[68] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *VLDB '06: Proceedings of the 32nd International Conference on Very Large Databases*, pages 187–198. VLDB Endowment, 2006.

[69] L. Tarassenko, P. Hayton, N. Cerneaz, and M. Brady. Novelty detection for the identification of masses in mammograms. In *Proceedings of the Fourth International IEEE Conference on Artificial Neural Networks*, pages 442–447, London, England, 1995. IEEE.

[70] D. M. Tax. *One-class Classification; Concept-learning in the Absence of Counter-examples*. PhD thesis, Delft University of Technology, Netherlands, Jun 2001.

[71] S. P. Team. Squirrelmail 1.4.7, 2006. Software and documentation available online at `http://www.squirrelmail.org/` (2006).

[72] C. Tong and V. Svetnik. Novelty detection in mass spectral data using support vector machine method. In *Computing Science and Statistics*, volume 34, Montreal, Canada, 2002.

[73] S. Trewin. Automating accessibility: the dynamic keyboard. In *Proceedings of the 6th International ACM SIGACCESS Conference on Computers and Accessibility*, pages 71–78, New York, NY, USA, 2004. ACM.

[74] type, 2009. Online reference available at: `http://www.askoxford.com/concise_oed/type?view=uk`.

[75] Typing, 2009. Online article available at: `http://en.wikipedia.org/wiki/Typing`.

[76] D. Umphress and G. Williams. Identity verification through keyboard characteristics. *Internation Journal of Man-Machine Studies*, 23(3):263–273, 1985.

[77] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, CA, USA, 2 edition, 2005.

[78] J. D. Woodward, N. M. Orlans, and P. T. Higgins. *Biometrics: Identity Assurance in the Information Age*. McGraw-Hill, New York, USA, 2003.

[79] E. Yu and S. Cho. *Novelty Detection Approach for Keystroke Dynamics Identity Verification*, volume 2690, pages 1016–1023. Springer, Berlin, Germany, 2003.

[80] E. Yu and S. Cho. Keystroke dynamics identity verification–its problems and practical solutions. *Computers and Security*, 23(5):428–440, 2004.

[81] L. Zhuang, F. Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. In *Proceedings of the 12th ACM conference on Computer and Communications Security*, pages 373–382, New York, NY, USA, 2005. ACM.

[82] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions of Information Theory*, 24(5):530–536, 1978.

Appendix A
Experiment Details

A.1 Purpose of Experiment

The purpose of these experiments is to record keystroke entry patterns whilst performing regular tasks on a computer. The recorded data can then be mined for biometric information using machine learning techniques.

A.1.1 Questionnaire

Participants will be asked to fill in a short questionnaire about their experience with computers and requesting some of their bio-data (such as height, weight, age, handedness, etc). The information will be used as attributes for machine learning algorithms, but will not identify the participant in any way. This information is used to help ascertain how much biometric information is contained in typing patterns.

A.1.2 Email Recording

In this experiment the participant uses an augmented version of SquirrelMail[1], rather than the standard Computer Science Department version. SquirrelMail is a web-based email system similar to that of Gmail, Hotmail and Yahoo! mail. SquirrelMail at:

`https://webmail.scms.waikato.ac.nz/cs/src/login.php`

gives standard access to the Computer Science Department email system. Figures A.1, A.2 and A.3 show the standard webmail client interface.

The recorded typing (RT) version is almost identical to the standard one, with the exception that it has the ability to record typing patterns for email. The user interacts with the RT version in the same way as they would normally (it provides access to the participant's standard CS email account). The differences are as follows:

[1]See www.squirrelmail.org for more information

Figure A.1: Login screen—standard CS webmail

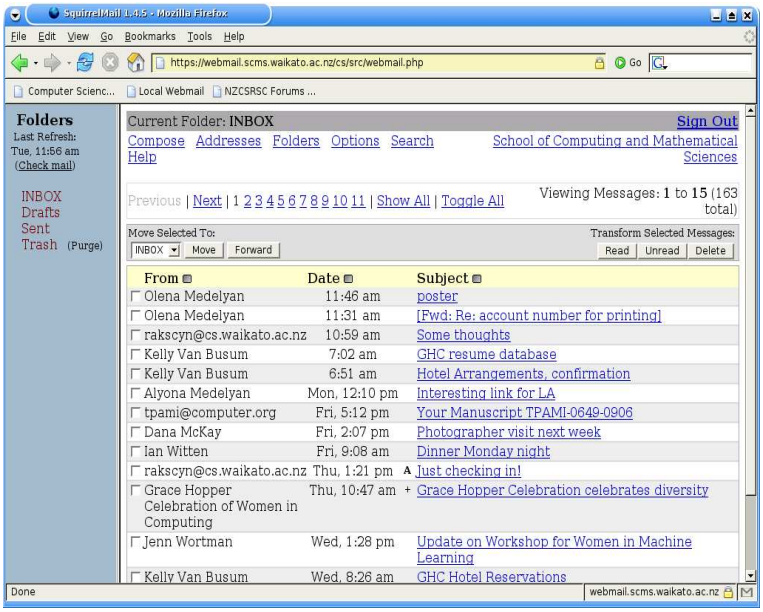

Figure A.2: Inbox screen—standard CS webmail

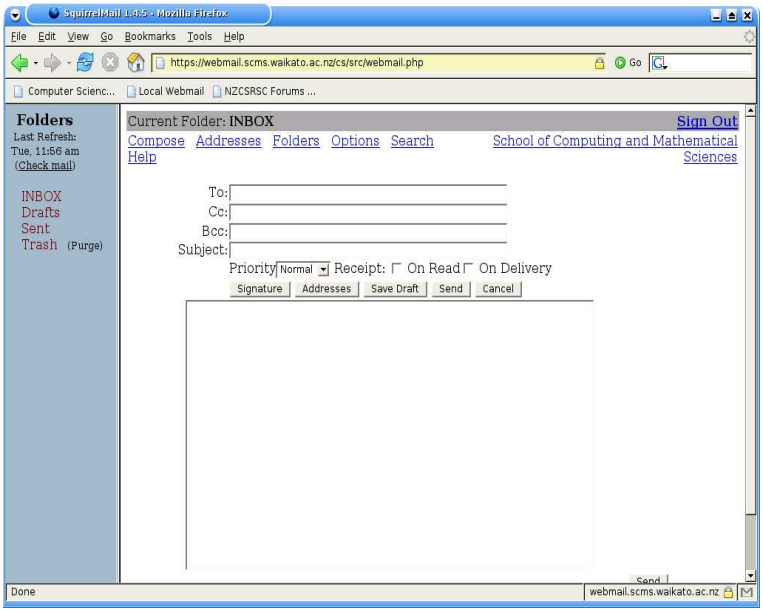

Figure A.3: Compose screen—standard CS webmail

1. The main login page carries a warning stating that this version of SquirrelMail is set to record typing patterns, and provides a link back to the standard version. It also provides a link giving the researcher's email address. The login area on the screen also has a colored background to indicate the difference between the standard and augmented versions. See Figure A.4 for a screenshot of the login screen.

2. The left hand panel (displaying folders) has a non-standard background color, and a notice saying "This version of SquirrelMail will record your typing." Figures A.5 and A.6 demonstrate the left hand panel messages.

3. The main window for composing an email (shown when a new email is created, or an email is forwarded/replied to) contains a check box under the subject entry box, that states: "If checked, all of your typing will be recorded for this email." If the check box is de-selected (it is selected by default) then the typing recorded for that email will not be saved for use in the experiment. If checked, when the user presses *Save Draft* or *Send* the recorded typing patterns will be saved to disk on the SquirrelMail server. An example screenshot is shown in Figure A.6.

The system records the typing for each of the text entry boxes on the compose screen: To, CC, BCC, Subject and Body. Mouse press events within the compose frame (only within that page of the browser) are also recorded. Press events can also be called clicks. Double clicks appear as two clicks very close

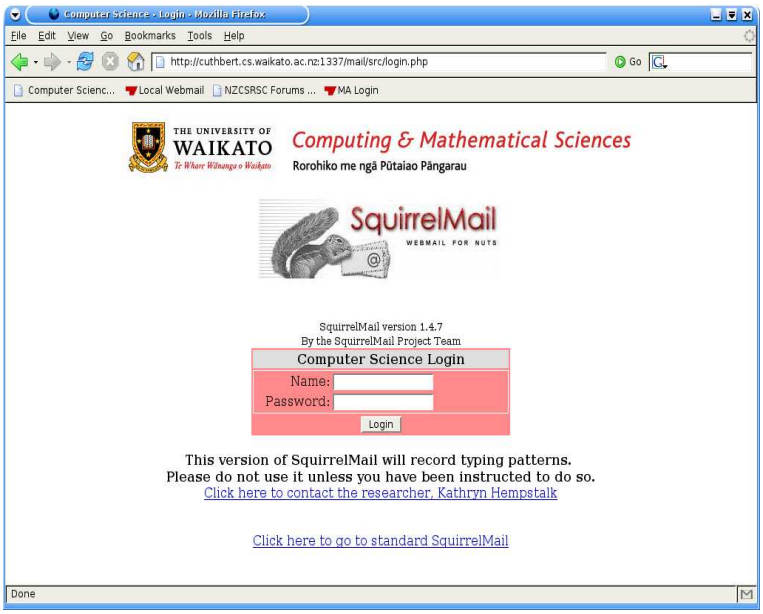

Figure A.4: Login screen—RT CS webmail

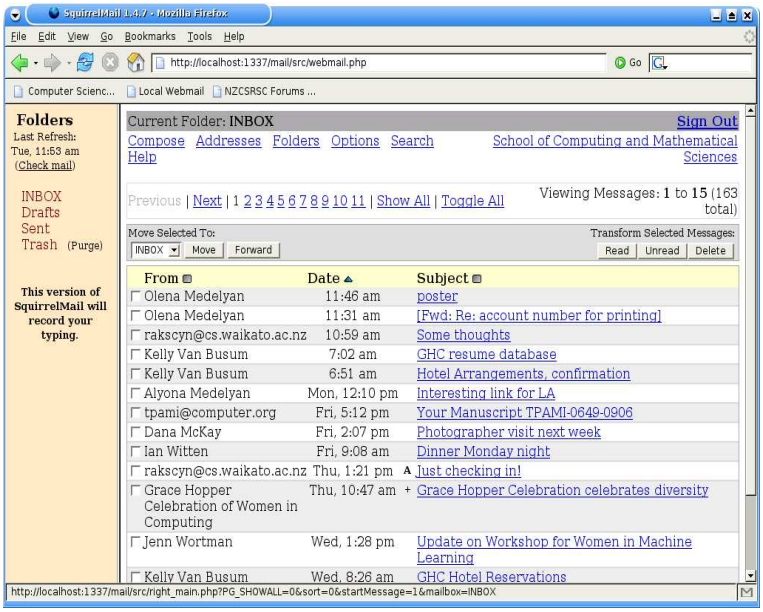

Figure A.5: Inbox screen—RT CS webmail

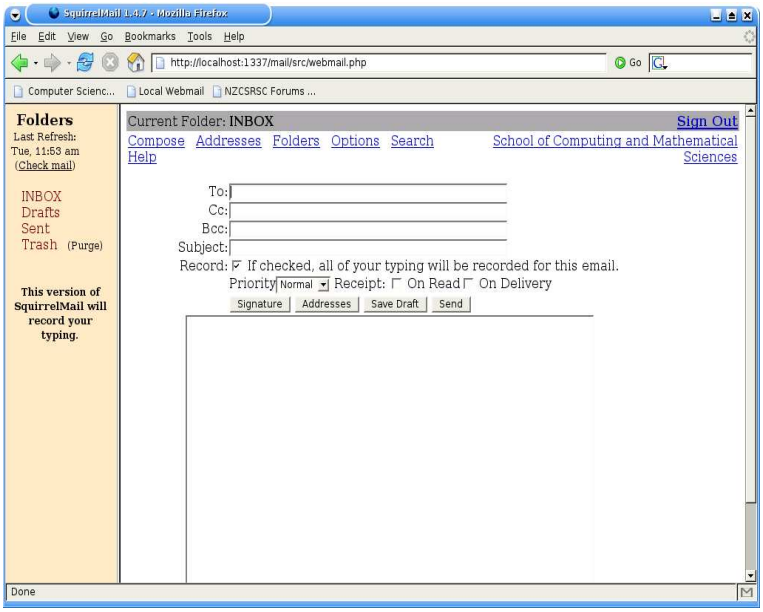

Figure A.6: Compose screen—RT CS webmail

together. When the user saves or sends the email, the recorded patterns are saved to disk on the server—but only if the check box on the compose window is checked. Each email recording is saved as a separate file on the server, in the format showing in Appendix A. The text already on the screen (such as from a saved draft, reply-to or forward) and login details (username/password) are not recorded. Only the typed keys and mouse events for the current email session are kept. Appendix A includes an example printout of an email recording, Appendix B describes what information is recorded.

It is safe to store the emails on the server in this plain format since only the computer administrator (Technical Support Group) will have read access on the server throughout the duration of the experiment. At the end of the experiment the files will be anonymised by a script, before being given to the researcher. Each SquirrelMail username will be changed to become a generated user ID. The master file associating assigned user ID with SquirrelMail username will be kept encrypted by a 128 bit Blowfish encoder, with the key known only by the researcher. The data stored on the server will be deleted in a "strong" way, that is, the recorded typing files (and mailAnalysis database files) will not only be removed, but the free space left behind will also be overwritten 7 times to prevent any data being retrieved.

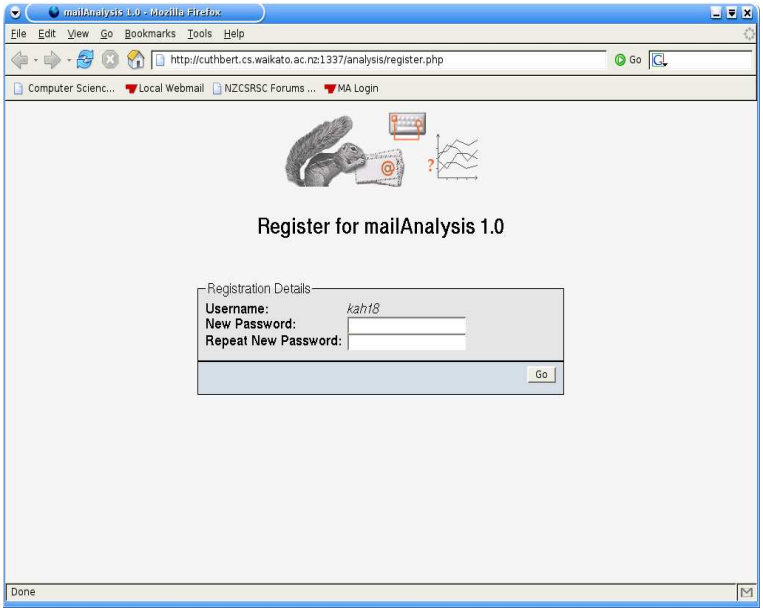

Figure A.7: Registration screen—Mail Analysis

A.2 Mail Analysis

To allow the participants to further control whether an email is included in the experiments, the mailAnalysis system is used. On the left hand pane of the main RT CS Webmail screen (see Figures A.5 and A.6) a link is provided titled "Analyse Mail". When pressed for the first time, the participant is taken to a registration screen (see Figure A.7), where they are asked to give a password to their mailAnalysis account. Upon registration, they are taken to the main analysis screen (see Figure A.8). This screen provides a list of all their emails that were recorded, sorted by date and time. As shown in Figure A.9, clicking the date of an email brings up a reconstruction of the email's typed input. The reconstruction may not match the exact layout intended due to the use of the mouse whilst the email was typed. It is possible to also see the raw recording of the email by clicking the button marked "Show Raw Recording", the original recording will appear in the same window as shown in Figure A.10 and described in the Appendix. The other columns provided in the main screen provide general statistics (like how many keys were pressed) for each email.

Clicking the "Summary Activity" link provides summaries of some statistics, including how many emails were typed and how much time was spent typing them (see Figure A.11). Clicking the "Options" link at the top right-hand corner of the page gives the user the ability to change their registered password (Figure A.12). Clicking "Logout", also at the top right-hand corner of the screen, returns the user to a login page where they must re-enter their

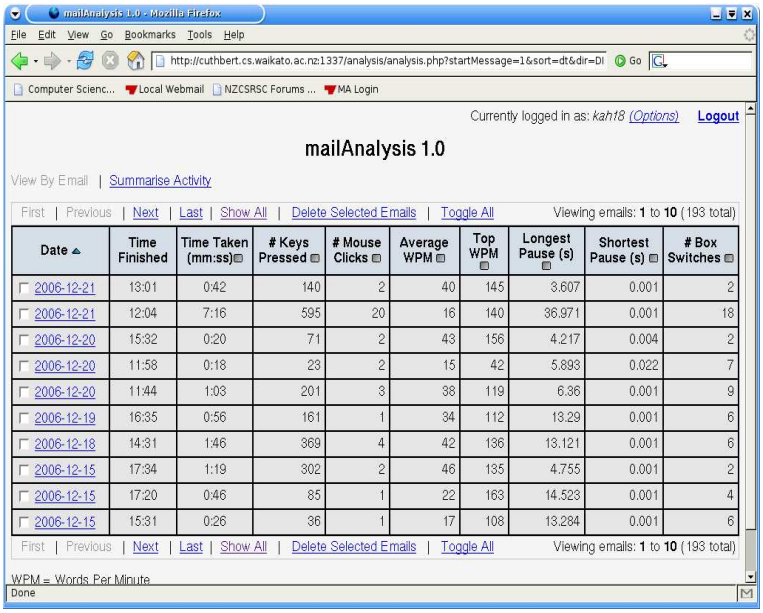

Figure A.8: Main screen—Mail Analysis

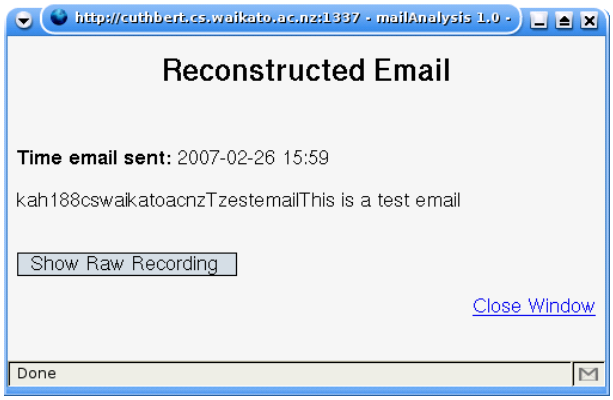

Figure A.9: Reconstructed email screen—Mail Analysis

Figure A.10: Reconstructed email screen showing raw recording—Mail Analysis

SquirrelMail username and mailAnalysis password before being granted access back to the mailAnalysis system (Figure A.13). Pressing the link in the main RT CS Webmail screen for a second time (after registering) will take the user directly into the mailAnalysis system. The system can also be accessed by entering login details directly into the mailAnalysis login screen.

The SquirrelMail username is encrypted as a session variable by PHP and passed to the mailAnalysis system, other users cannot access this variable. The session ID is used to gain access to the store of session variables, and is kept as a MD5 hash string in a cookie on the user's computer until either the cookie expires or is removed by some other process. The cookie is set to expire after 30 minutes (the default for SquirrelMail), but can be removed before it expires by clearing out the browser cache or by the user logging out of either SquirrelMail or mailAnalysis.

The mailAnalysis system is driven by a database stored on the same machine that runs the RT CS webmail install. A script runs over the pattern files stored on disk for each email and adds them to the database every 30 minutes. When a user deletes an email from mailAnalysis it is removed from all tables in the database and the file is flagged for deletion in another database table. On

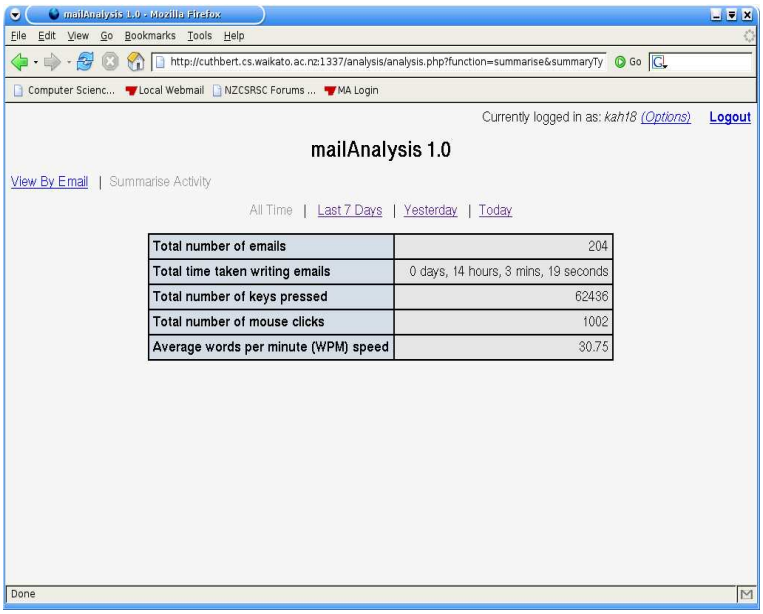

Figure A.11: Summary screen—Mail Analysis

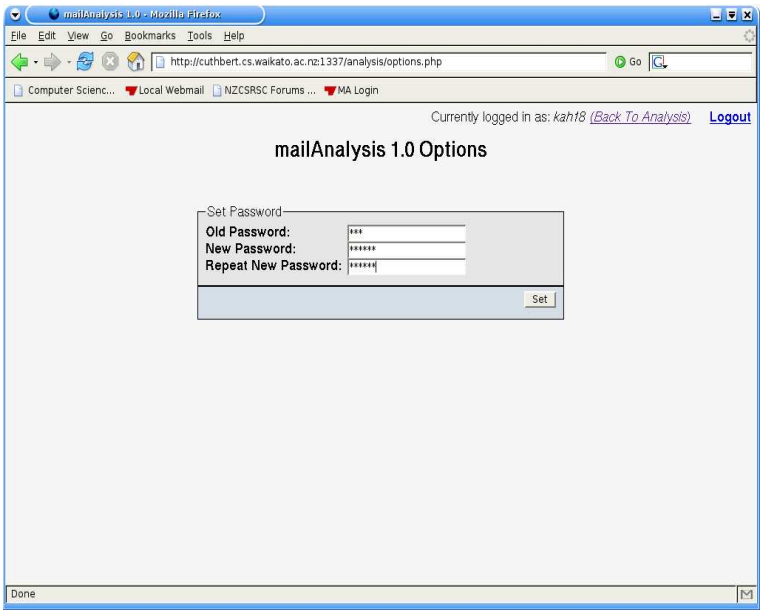

Figure A.12: Options screen—Mail Analysis

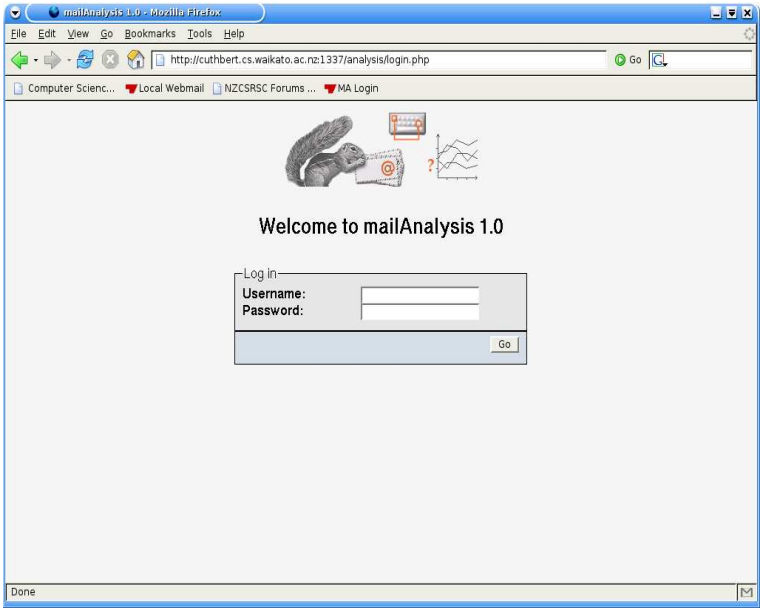

Figure A.13: Login screen—Mail Analysis

the next script pass, the pattern files flagged for deletion on the previous day are removed from disk and any new pattern files are added to the database. There is a small grace period between deleting emails in the database and deleting them off disk to ensure that accidental deletions do not result in the loss of useful data.

A.3 Network Access

Running the data collection on the University network poses no risk, there is no software to be installed and the participant is only instructed to use a different web-based version of SquirrelMail.

A.4 Data Collection

As detailed above, a questionnaire will be used to collect bio-data and computing experience for all research participants. The two experiments are themselves the process of data collection. No video, audio or notes will be taken throughout the observation period. Only keystroke and mouse click information will be recorded.

The data that is being collected may be sensitive information. Whilst the participant always has the opportunity to turn the recording system off, the content of what they type can be personal. It is possible to partially

reconstruct a document from typed input. It is difficult to do so, however, because the use of a mouse and lack of cursor location knowledge means that it is possible for input to be typed in non-consecutive order. Because a key recording system records everything, it is possible to see text that was removed from the document. Care will be taken to ensure the data will only ever be stored in such a way that only the researcher and the participant will have access to it in a way where this information can be accessed.

A.5 Data Archiving/Destruction

Data for the experiment may be temporarily kept on the participant's computer. The data will be moved to a secure store on the researcher's computer at the end of the observation period. This data will be destroyed at the end of the researcher's PhD, which will approximately be March 2009. All deletion methods will be "strong"—rather than just allowing the system to mark the deleted files as free space, each file will be overwritten by random data at least 7 times to ensure that the recorded information will not be recoverable after deletion. Some processed data may be kept for an indefinite period after this research is complete to allow others to replicate the results. However, the retained information will contain no content that would allow the identification of any research participant.

A.6 Confidentiality

Confidentiality and participant anonymity will be strictly maintained. All information gathered will be used for statistical analysis only and no names or other identifying characteristics will be stated in the final or any other reports.

A.7 Example Email Recording

Username: kah18
Date: 19-09-06-1602
Browser: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.9) Gecko/20061206 Firefox/1.5.0.9
Mouse: [2 8513937 0 (201,128) 3 8514033 0 (201,128) 2 8516617 0 (209,148) 3 8516681 0 (209,148) 2 8518361 0 (175,173) 3 8518425 0 (175,173) 2 8523089 0 (126,322) 3 8523161 0 (126,322) 2 8530689 0 (356,241) 3 8530753 0 (356,241)]
To typing: [168 8512062 0 169 8512226 0 138 8512253 0 139 8512454 0 167 8512492 0 169 8512534 0]
CC typing: [168 8514157 0 169 8514343 0 166 8514393 0 139 8514448 0 168 8514508 0 167 8514549 0 169 8514602 0 102 8515440 0 103 8515532 0 104 8515693 0 105 8515766 0 106 8515937 0 107 8516024 0]
BCC typing: [98 8517413 0 99 8517590 0 101 8517672 0 102 8517805 0 103 8517908 0]
Subject typing: [168 8519906 4 169 8519980 0 144 8520012 0 145 8520114 0 147 8520185 0 64 8520303 0 65 8520433 0 16 8520645 0 17 8520747 0 166 8520797 0 167 8520890 0 64 8521010 0 65 8521165 0 147 8521241 0 166 8521248 0 167 8521322 0 65 8521401 0 130 8521430 0 131 8521554 0 65 8521612 0 168 8521699 0 169 8521821 0 138 8521833 0 139 8522030 0 167 8522091 0 169 8522138 0 380 8522208 0 381 8522292 0]
Body typing: [168 8524311 4 138 8524448 0 169 8524454 0 166 8524487 0 139 8524540 0 168 8524577 0 167 8524609 0 146 8524659 0 169 8524683 0 147 8524741 0 156 8524801 0 157 8524884 0 143 8524950 0 376 8524987 0 377 8525098 0 64 8525140 0 65 8525262 0 138 8525347 0 169 8525422 0 139 8525446 0 168 8525523 0 167 8525547 0 169 8525618 0 146 8525671 0 147 8525742 0 156 8525809 0 157 8525896 0 143 8525961 0 376 8525996 0 377 8526102 0 64 8526177 0 65 8526282 0 158 8526314 0 159 8526396 0 156 8526451 0 157 8526552 0 64 8526630 0 139 8526641 0 168 8526722 0 65 8526730 0 169 8526822 0 174 8527039 0 175 8527131 0 159 8527242 0 64 8527661 0 65 8527764 0 168 8527791 0 169 8527888 0 145 8527914 0 138 8527961 0 139 8528100 0 138 8528166 0 165 8528201 0 139 8528293 0 380 8528343 0 381 8528420 0 16 8528669 0 17 8528737 0 16 8528790 0 17 8528855 0 16 8528910 0 17 8528952 0 16 8529001 0 17 8529078 0 164 8529085 0 165 8529225 0 139 8529279 0 138 8529370 0 139 8529477 0 381 8529548 0]

A.8 How to read an email recording

An email recording always begins with:

```
Username: userName
Date: dd-mm-yy-24hrTime
Browser: browserIdentificationString
```

The userName is the name of the user who typed the email, the date is the date and time that the email was finished (sent). The browserIdentificationString is the name and version of the web browser that was used to type this email (in the format in which the browser identifies itself). Following this, the section *Mouse* appears, which is recorded in the format:

$$clickButtonNumber, timeInMilliseconds, modifiers, (xPos, yPos), ... \tag{A.1}$$

The clickButtonNumber is $(button * 2) + (ifPress?0 : 1)$. The timeInMilliseconds is the absolute time this event was recorded. The modifiers is a four bit integer with each bit representing the state of the Ctrl, Alt, Shift and Windows keys at the time the button was pressed (these keys cannot be independently recorded in a web-based system). Finally the coordinates of the mouse pointer (relative to the document) are given.

For the remaining sections (To, CC, BCC, Subject, Body), the keystrokes are recorded in the following form:

$$keyStroke, timeInMilliseconds, modifiers, ... \tag{A.2}$$

The keyStroke is recorded as an ASCII character number and modified to be $(ASCIIcode * 2) + (ifPress?0 : 1)$. The timeInMilliseconds and modifiers are in the same form as the mouse events. Pointer coordinates are irrelevant and are not recorded.

Appendix B
Instructions For Participants

These instructions are for typists who have agreed for their emails to be recorded and used for evaluation in this thesis.

B.1 Requirements

This study requires you to have a Waikato Computer Science Department email account and access to a web browser. If you are using Linux as an operating system, you are requested to use Opera as a web browser because Mozilla based web browsers for Linux contain a bug that prevents your typing from being accurately recorded.

B.2 Links

The following URLs provide access to SquirrelMail and mailAnalysis for this study:

SquirrelMail: `http://chronicle.cs.waikato.ac.nz/mail`
mailAnalysis: `http://chronicle.cs.waikato.ac.nz/analysis`

B.3 CS Webmail

For the duration of the study we request you use a different copy of SquirrelMail for accessing your Computer Science emails. This copy, called RT SquirrelMail (RT for Recorded Typing), will record the typing you perform in the email compose screen. It does not record your password, or any text typed in a reply/forward. Only the text typed in the To, CC, BCC, Subject and Body text boxes, along with any mouse events that occur on that page are recorded.

If you do not wish your typing to be recorded for a particular email, simply uncheck the box marked "If checked, all of your typing will be recorded for

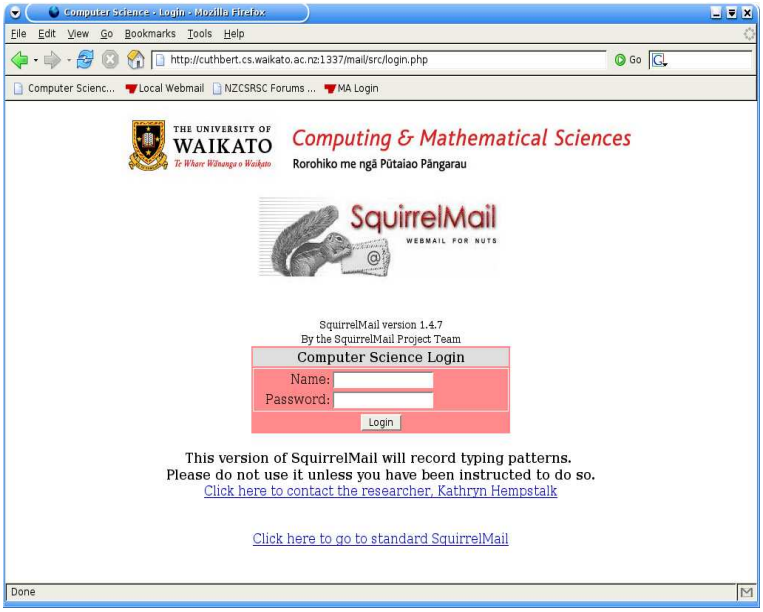

Figure B.1: Login screen—RT CS webmail

this email" before saving or sending the email (see Figure B.3). If you forget to uncheck the box, you are still able to remove the recording from inclusion in the experiment using the mailAnalysis system. Use of the mailAnalysis system is described further in the next section.

Standard SquirrelMail and RT SquirrelMail have no functional differences other than the obvious addition in RT SquirrelMail—the typing patterns for emails are recorded. Visual markers have been added to RT SquirrelMail to ensure you are always aware of which version of SquirrelMail you are using. The login screen, shown in Figure B.1, has a red box around the username and password entry, and a comment underneath providing a link back to the standard SquirrelMail install. The main webmail screens, shown in Figures B.2 and B.3, have been altered to have a yellow sidebar (instead of the default blue) and carry the warning "This version of SquirrelMail will record your typing". The sidebar also provides a link to mailAnalysis.

B.4 Mail Analysis

The mailAnalysis system provides you with the ability to review your own emails and delete them from the data store if you decide the content of the email is not appropriate for inclusion in machine learning experiments. As a participant you will only be given access to your own emails. You must register with mailAnalysis in order to access your recordings. You will be taken to

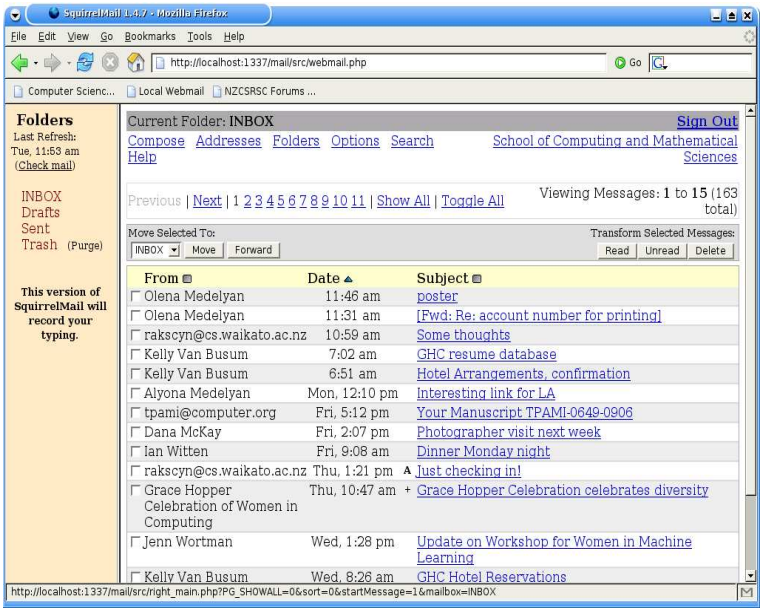

Figure B.2: Inbox screen—RT CS webmail

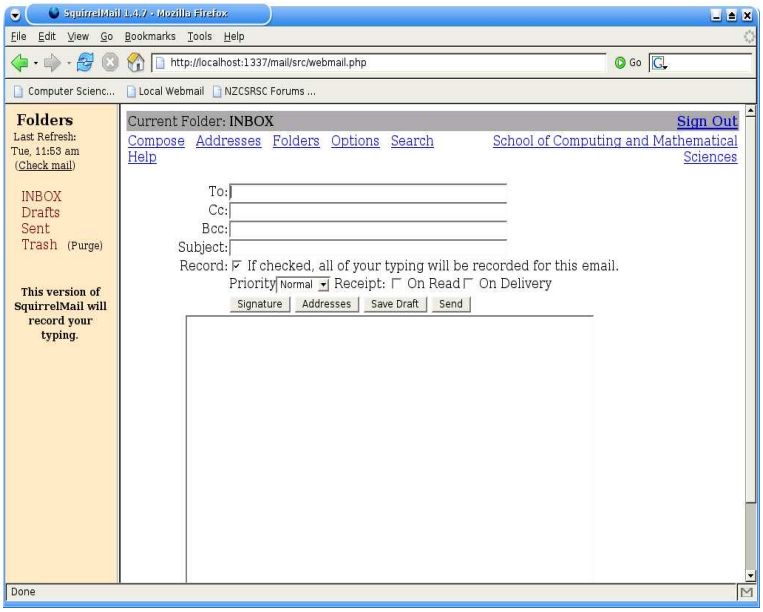

Figure B.3: Compose screen—RT CS webmail

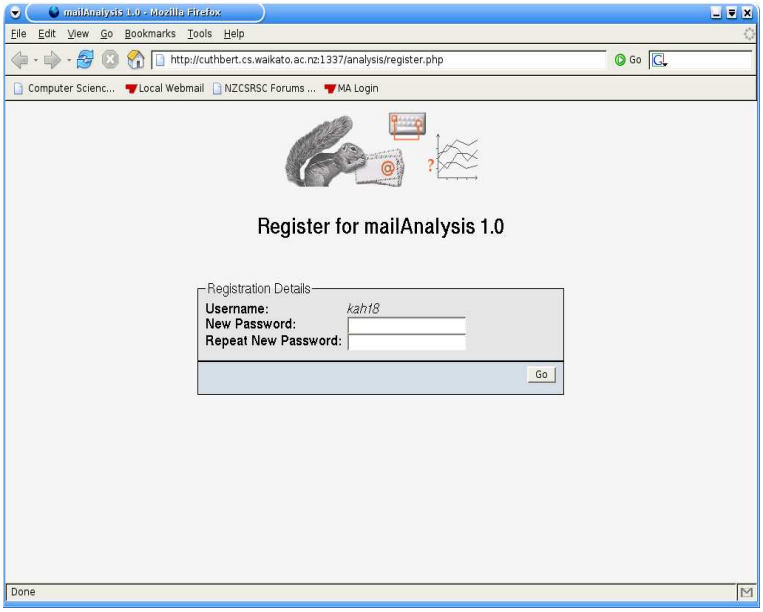

Figure B.4: Registration screen—Mail Analysis

the register page when you click "Analyse Mail" in the left hand pane of RT SquirrelMail if you have not previously registered (see Figure B.4). To register, simply enter a password in the boxes provided, and click "Go" to begin using mailAnalysis. After registering for the first time, clicking the "Analyse Mail" link in RT SquirrelMail will take you straight into the mailAnalysis system.

The main screen from mailAnalysis is shown in Figure B.5. It displays a list of all the emails recorded that will be used for machine learning, initially ordered by the date they were recorded. You can alter the ordering of the mailAnalysis system by clicking the grey boxes (or arrows) next to each column heading. Clicking the date of a particular email will pop up a window displaying a reconstruction of the typing pattern recorded for that email (see Figure B.6). In the reconstructed email window, it is possible to view the raw email recording by clicking the button marked "Show Raw Recording" (see Figure B.7). The reconstruction is not perfect—it shows only what could be retrieved from the recorded sequence of typing events. The purpose of the reconstruction in mailAnalysis is to allow viewing of an emails content, without subjecting you to the raw format of the recording.

You are encouraged to delete the email if the general content of the email is confidential. Please do not delete emails just because you have typed your name, email address or contact details inside it, because this sort of information will be removed before machine learning is performed. To delete an email, check the box next to that emails date and click the link marked "Delete Selected Emails." The emails are not instantly removed from disk, so if you

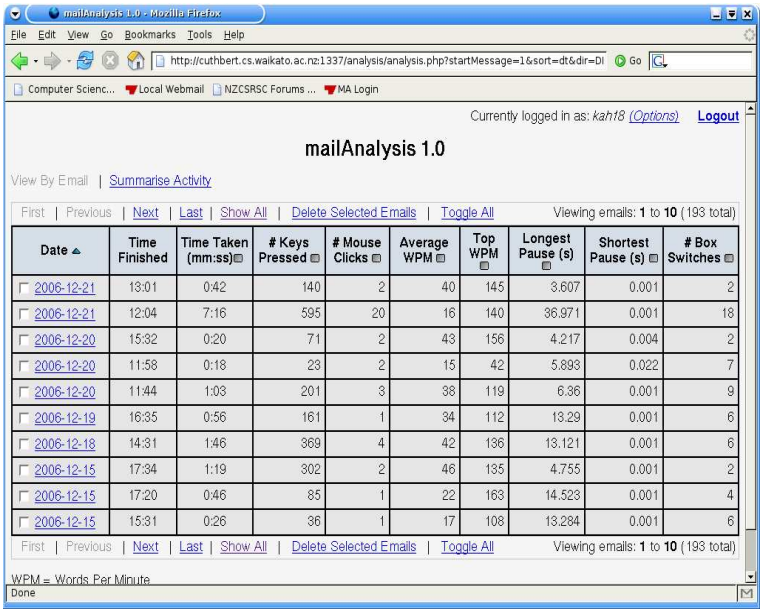

Figure B.5: Main screen—Mail Analysis

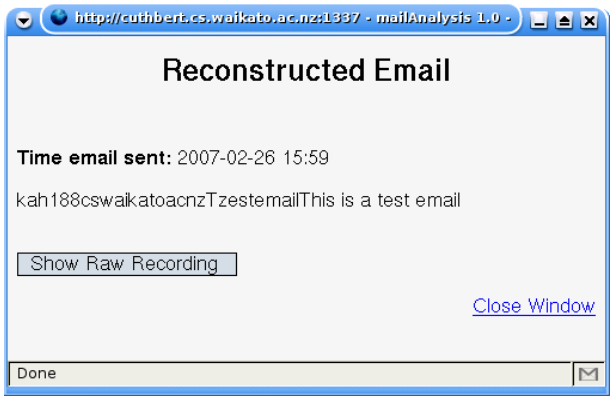

Figure B.6: Reconstructed email screen—Mail Analysis

Figure B.7: Reconstructed email screen showing recording—Mail Analysis

delete an email by accident please get in touch with the researcher as quickly as possible.

The mailAnalysis system also provides general summary statistics. These statistics can be accessed by clicking the link marked "Summarise Activity" on the main screen (Figure B.5). The summary statistics are provided for set periods of time (all time, last 7 days, yesterday and today) and each period can be accessed by clicking the appropriate link (see Figure B.8).

It is possible to change your registered password for mailAnalysis by selecting "Options" at the top right hand corner of the screen and entering your old password and a new one in the boxes provided. The password is used if you wish to use mailAnalysis without first entering SquirrelMail. At the login page (see Figure B.10), enter your SquirrelMail username and your current mailAnalysis password to be taken into mailAnalysis. Logging out of mailAnalysis by clicking "Logout" in the top right hand corner will take you back to the login page.

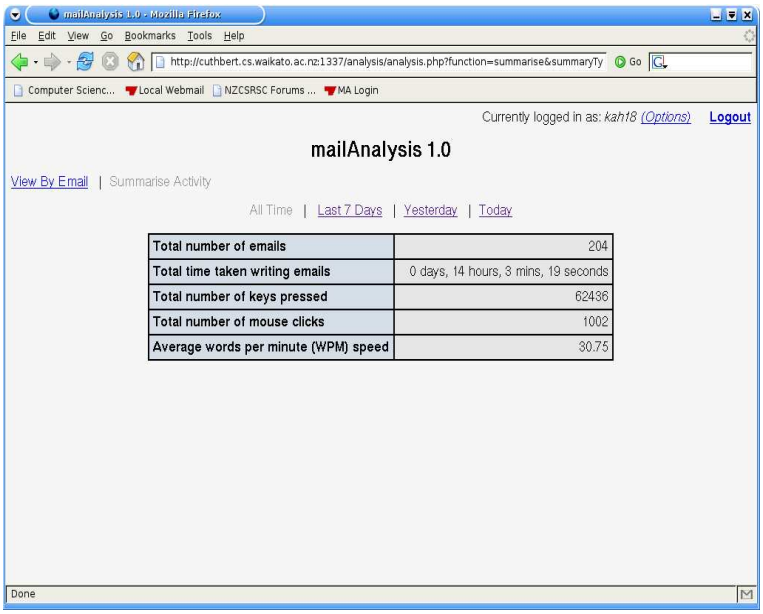

Figure B.8: Summary screen—Mail Analysis

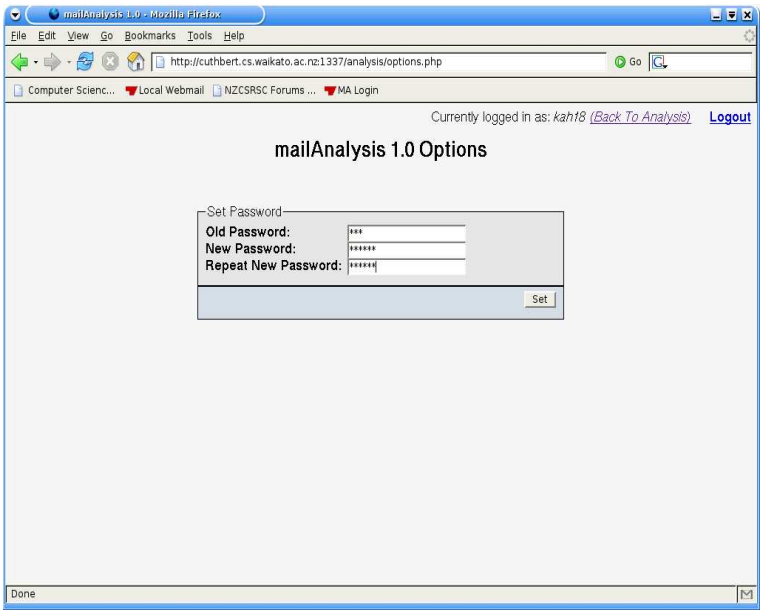

Figure B.9: Options screen—Mail Analysis

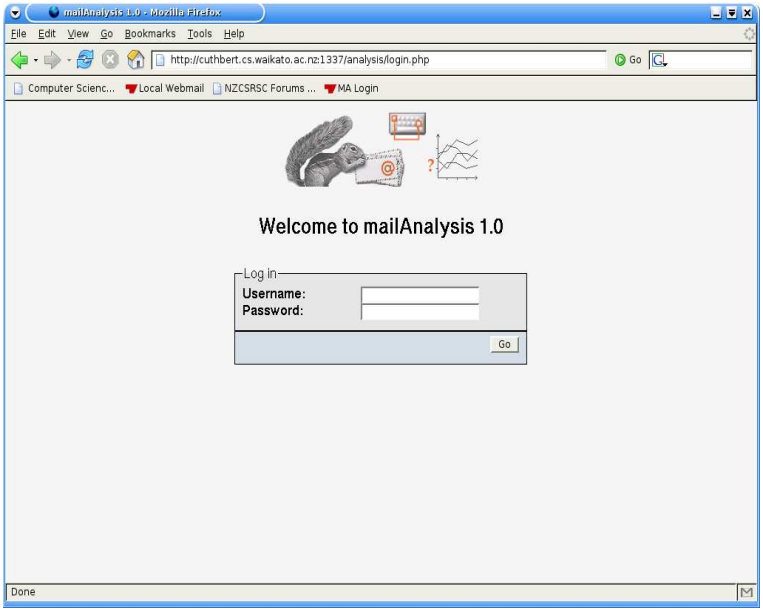

Figure B.10: Login screen—Mail Analysis

B.5 Recording Content

The data used during machine learning may be provided to a third party for reproduction of the results. Any email content which may identify a participant will be removed before this data is made available, and all stored usernames will be generalised to $userXX$, where XX refers to a random number assigned to each participant. The mailAnalysis system gives users access to their own reconstructions of the recorded typing patterns and the original raw formats of each email as well.

B.6 Final Review

At the end of the data collection, there will be a week-long grace period when no further emails will be recorded but the mailAnalysis system will still be available. This enables you to have a final review of the emails recorded in order to decide whether they contain material inappropriate for inclusion in this project, and delete them if necessary. You will be notified when the RT SquirrelMail system becomes unavailable and the final review period begins. The emails will not be accessed by the researcher until after this review period is up, at which time they will be anonymized and moved to a secure store on the researcher's computer.

Appendix C
Questionnaire

Introduction

The following questions are designed to provide bio-data for machine learning on the typing samples. This is not a test, this information is only used to evaluate the effectiveness of typist recognition. If at any stage you do not feel comfortable answering a question, leave the answer as blank.

Questions

Participant Number:

Question One: Age

Please circle the option that applies to you:

- 11–20
- 21–30
- 31–40
- 41–50
- 51–60
- 60 or older

Question Two: Height

What is your height in centimetres?

Question Three: Weight

What is your weight in kilograms?

Question Four: Handedness

In the table below indicate your preference for each hand in the following activities by placing a check mark in the appropriate column. Where the preference is so strong you would never try to use the other hand, put two check marks. In the case you are indifferent to the use of either hand, place a single check mark in both columns.

Some of the actions listed require the use of both hands. In these cases, the part of the task, or object, for which hand preference is wanted is indicated in brackets.

Please try and answer all questions, only leave a blank if you have no experience with the task.

Task	Left Hand	Right Hand
Writing		
Drawing		
Throwing		
Scissors		
Toothbrush		
Knife (without fork)		
Spoon		
Broom (upper hand)		
Striking Match (match)		
Opening Box (lid)		
Total Check Marks		

Question Five: Hand spans

On each of the following pages please place the hand indicated palm down on the page, with fingers stretched out as far as possible, and trace around it with a pen. The purpose of doing this is to enable hand span and finger lengths to be measured in a uniform way. [Space omitted in this appendix, actual questionnaires contained a single A4 page for each hand.]

Question Six: Keyboard

Are you a touch typist? That is, do you use all four fingers of both hands to type on letter keys on a keyboard?

If not, how many fingers of each hand do you use to type?

Do you have to look at the keyboard to type on letter keys?

Do you have to look at the keyboard to type on number keys?

Do you suffer from RSI (Repetitive Strain Injury), a wrist injury, or any other disability/injury that may impair your ability to type on a keyboard?

What type of keyboard do you type the majority of your emails with?

- Standard Keyboard
- Ergonomic (or Natural) Keyboard
- Laptop Keyboard
- Other (Please Specify)

Is there anything unusual about the keyboard that you use to type your work? For example, does it have some letters swapped (such as Z and Y for German users), do you use a layout other than QWERTY (e.g. Dvorak) or are there keys that do not work correctly?

Question Seven: Mouse

Please circle the option that best describes the mouse you use for the majority of your computer work.

- Ball Mouse
- Ball Mouse (Wireless)
- Optical/Laser Mouse
- Optical/Laser Mouse (Wireless)
- Trackball
- Graphics Tablet (Pen)
- Graphics Tablet (Mouse)
- Touchpad (On a laptop or as a pad integrated with a keyboard)
- Other (Please Specify)

Question Eight: Computer Use

How many hours do you spend on a computer in an average weekday?

How many of those hours would be spent on a computer at home?

How many hours do you spend on a computer in an average weekend day?

How many of those hours would be spent on a computer at home?

From the following options, which one best describes your main use for a computer?

- Email and Surfing the Internet
- Graphics (e.g. Drawing)
- Gaming
- Word Processing (e.g. Typing Reports) or Typing-Based Tasks (e.g. Programming)
- Work Related (e.g. Data Entry, Spreadsheets)
- Other (Please Specify)

Appendix D
List of Top 54 Predictive Digraphs

The table below lists the top 54 predictive digraphs. Each digraph was typed at least ten times by each of the first ten users in the `sm-all` dataset (Participants A to J). The AUC values for each user were weighted on the number of times each digraph was typed before being averaged and included in the table.

Digraph	Average Weighted AUC
in	0.744
as	0.726
yo	0.714
er	0.707
on	0.701
re	0.689
se	0.682
at	0.682
ou	0.681
ar	0.673
st	0.663
o Space	0.660
l Space	0.636
y Space	0.633
es	0.623
g Space	0.621
i Space	0.615
n Space	0.615
to	0.613
ha	0.609
he	0.604
th	0.600
t Space	0.599
a Space	0.599
Space y	0.599
r Space	0.595
d Space	0.595
s Space	0.594
ve	0.592
Space h	0.592
me	0.580
is	0.577
Delete Delete	0.576
e Space	0.575
le	0.575
an	0.571
ti	0.568
Space o	0.567
nt	0.566
Space s	0.562
or	0.559
en	0.559
Space f	0.555
ma	0.554
Space m	0.552
Enter Enter	0.548
Space w	0.548
Space i	0.547
e Backspace	0.540
Space c	0.536
Space a	0.536
Space t	0.523
Backspace Backspace	0.512
Space Backspace	0.507