Faculty of Fundamental Problems of Technology
Wrocław University of Science and Technology

User authentication with keystroke dynamics

Gabriel Tański
Student no. 236346

Master Thesis
under the supervision of
Piotr Syga, PhD

Politechnika
Wrocławska

Wrocław 2021

Contents

Abstract

With multifactor authentication becoming increasingly popular, keystroke dynamics is frequently referred to as a method that improves user security without increasing the system's complexity. It is even more usable in the context of continuous authentication, because specific user interaction is not required for it to work.

In this thesis, keystroke dynamics authentication models are evaluated in the context of continuous authentication. Similar models are created for the purpose of demographic feature inference, which can act as a supportive mechanism for the authentication model. Additionally, various methods of data processing and their effects on the models' accuracy are proposed and discussed.

The tests show that an authentication accuracy of at least 86.6% is achievable using Support Vector Machines or k-nearest neighbors. It is also shown that effective authentication can be achieved in under 1 second, making it feasible for continuous authentication. It is also briefly discussed that the size of the dataset has a non-negligible impact on the models' performance.

Demographic feature inference is shown to be feasible in the case of one's gender and dominant hand, top confirmed accuracy being 73.3% and 85.8% respectively. An additional analysis of the latter is performed.

Introduction

Every day, humans encounter countless situations in which their identity has to be confirmed. The vast majority of these verifications are performed subconsciously - our acquaintances recognize us by our looks, our behavior, our voice. In some cases we need to present documents confirming our identity, or some knowledge that only we are supposed to have, like a reservation number or referring to a common acquaintance.

This is not limited to human interactions, because the same applies to computer systems, where the process of authentication (i.e., the process of verifying the identity of a user) is usually far less subtle. The methods of verifying one's identity are categorized into three factors of authentication.

The first one is knowledge, for example a passphrase. The second one is possession, usually in the form of a mobile device or a token. The last one concerns inherent characteristics of one's body, usually fingerprints or external appearance. It also encompasses all the behavioral patterns and mannerisms, such as the manner of walking or the way someone types on a keyboard.

In the case of computer systems, the most commonly used factor is knowledge in the form of a password. It has several weaknesses, most notably it can easily be exposed and used by unauthorized third parties. As for the human-based interactions, it seems clear that we recognize other people mostly based on their physical features. The method is prone to errors, but is reliable most of the time, and is also passive – does not require the cooperation of another person and can be done "in the background".

Using factors of authentication other than knowledge is becoming increasingly popular online, with multiple services offering Two-Factor Authentication (2FA), meaning that more than one factor is used in the process of authenticating the user, usually password and a mobile token. The Revised Payment Services Directive (EU Directive 2015/2366)[12], also known as PSD2, introduced legislation which obliged banks in the European Union to enable 2FA for all their customers.

In the banking sector, where high security is a fundamental requirement, there are also pilot programs that use inherent biological features of their users to determine whether the person that is using the online account is the owner of the account. Not every device is equipped with cameras or fingerprint scanners, but every device is interacted with in some specific way, and therefore behavioral patterns can be utilized instead.

Using the user's keystrokes is an effective and low-cost (both financially and computationally) way of authenticating the user. The method is also capable of performing these actions continuously. This kind of authentication is relatively simple to set up and use, it can greatly increase the security of online accounts (not only on banking sites) and physical devices. While usually insufficient on its own, it is a reliable way to mitigate potential attacks requiring little user involvement.

In this thesis, some commonly used techniques for keystroke-based authentication are presented and discussed. The overall accuracy of a model is an important aspect thereof, however in the context of continuous authentication the time factor is of utmost importance as well. Therefore, the methods are also evaluated in terms of preparation and run time required to perform a verification. The dataset used for training and testing was created specifically for this purpose and, as of the submission of this thesis, has already been deleted per the obligation to subjects who submitted their keystroke data.

In addition to discussing the commonly used authentication techniques, some data cleansing methods are presented, discussed, and evaluated. In the case of data collected in non-laboratory environment, there is always noise in the keystroke data. Example of such noise include typographical errors, e.g., missing a letter, adding an extra character, or pressing the keys in the wrong order. Another source of noise in the data is in the timing information, for example due to prolonged breaks between successive keystrokes. Since the data submitted by the subjects contains only keystrokes, it is possible that their own error-correction attempts leave a mark in the keystroke data, for example in the form of the Backspace key, or even arrow keys. These issues require the data to be properly cleansed in order to reduce the

noise's influence on authentication. As part of the authentication pipeline, data cleansing also needs to be performed in a way that minimizes the time required for the authentication. On the one hand, the process takes time, but on the other, removing unnecessary data makes the processing quicker.

The chapter named Terminology contains the basic terms related to the subject covered in this thesis. The chapter Previous Work discusses previous work in the field. Data Acquisition outlines the process of gathering keystroke data. Data Processing follows the preprocessing and cleansing of the data, namely how the data is converted from its raw form stored in the database to something that can be analyzed. The chapter Models covers the statistical and machine learning models used for authentication. In the chapter Results the results for raw and sanitized data are presented. The last chapter, Summary, summarizes the thesis, addressing the limitation and presenting conclusions.

Appendix Summary in Polish briefly summarizes the entire thesis in Polish. Appendix Rewritten text contains the full text that was rewritten by the volunteers. The appendix Data Samples showcases the sample raw data that was collected. The appendix Contents of the DVD explains the contents of the DVD disc submitted along with this thesis.

Terminology

This chapter explains the terminology that is used throughout the thesis to recall the definitions that are used. Theoretical introduction contains abstract concepts, whereas Frequently used terms covers definitions directly related to the topic, which are referred to in the subsequent chapters.

3.1 Theoretical introduction

3.1.1 Biometrics

Derived from the Greek words *bio*, meaning life, and *metric*, meaning to measure, it is the measurement and statistical analysis of people's intrinsic physical and behavioral characteristics. Usually it is used due to the characteristics being unique or nearly unique, meaning that the person in question can be recognized with a significant degree of certainty. Biometric features include fingerprints, facial patterns, iris patterns, voice, hand or forearm vein patterns, and more.

3.1.2 Behavioral biometrics

A subcategory of biometrics, behavioral biometrics deals with measurable patterns in human behaviors. Usually it does not require specialized equipment to capture. Examples of behavioral patterns include, among others, keystroke patterns, patterns of computer mouse movements, and gait (manner of walking).

3.1.3 Keystroke dynamics

Keystroke dynamics, also called keystroke patterns, is the detailed information about the time when each key was pressed and when it was released by a person typing on a keyboard. Usually, regular computer keyboards are considered, but some research also focuses on virtual keyboards on mobile devices or specialized keyboards equipped with pressure sensors.

This technique has numerous advantages. The most important are: low cost compared to other biometric-based authentication methods, usually no special hardware requirements in, and non-invasiveness, i.e., no need for active user participation past the enrollment phase.

3.1.4 Enrollment

Biometric authentication relies on some model that knows the users' relevant features – be it facial characteristics or typing patterns. Enrollment is the phase where such a model is taught about the users using the training data. Depending on the technique, this may simply be calculating the mean of some feature or training a neural network. In other words, the enrollment phase is the first stage in the existence of a model. Only after enrollment can the model be tested or deployed.

3.1.5 Continuous authentication

One of the applications of behavioral biometrics. As opposed to regular authentication, it is performed continuously. The details depend on the method used for the authentication process, but in general there are several key characteristics: it is invisible to the user and does not require additional interaction past the enrollment phase[1], is computationally expensive, and usually has low accuracy[2] compared to

[1] unless a confident rejection is returned by the verifier

[2] due to multiple authentications with small samples instead of one attempt with a large batch of data

non-continuous authentication. It is becoming widely used as an additional security measure on top of existing Two-Factor Authentication solutions.

3.2 Frequently used terms

3.2.1 Keystroke dynamics

Dwell Time

Dwell Time refers to the amount of time between pressing and releasing a single key. It marks how long a given key was held down. Also known as Duration Time or Hold Time.

Flight Time

Flight Time is a "duration" of a key sequence, also called latency. It is calculated by taking the timing of the first keystroke and the last keystroke in a sequence and finding the time difference. Press-to-Press, Press-to-Release, Release-to-Press, and Release-to-Release are all used in the literature. While the exact method of calculating the Flight Time does not influence the research, it is naturally important to select only one of them and consequently use it in one's research. Without loss of generality, in this paper the Flight Time for each sequence shall be calculated as Press-to-Release, i.e., the time difference between pressing the first key and releasing the n-th key in a sequence. Additionally, referring to FT of a 1-key sequence will be equivalent to referring to its DT.

N-graph

As described in [29], N-graph (often referred to as *N-gram*) is the name given to the timing measurement between three or more consecutive keystroke events. N-graphs are defined as contiguous sequences of length n from a given sample. In this paper, n-gram and n-graph are used interchangeably and refer to any key or word sequence for $n \geq 1$. In literature, 2-letter sequences are also called bigrams (also digraphs or digrams), and 3-letter sequences trigraphs or trigrams.

3.2.2 Evaluation

When evaluating a model, test samples are fed to the model, and the responses are gathered and assessed. For authentication purposes, the model can only respond with one of two options: acceptance (meaning that the sample is considered to belong to the user in question) or rejection (meaning that the sample is considered to belong to someone else).

The model may either be right or wrong. Therefore, four scores may be calculated for each model after running the tests: TP (True Positive), TN (True Negative), FP (False Positive), FN (False Negative). The combination of these metrics is called the confusion matrix. Positive means that, in this context, the model accepted the sample, meaning the authentication was successful. The opposite is Negative, in which the model rejects the sample, essentially saying that the authentication failed. True result means that the model was correct. False means that the model was mistaken.

False Rejection Rate

False Rejection Rate, or FRR, is the rate of authentication instances in which the model incorrectly rejected the test sample. FRR depends on the sensitivity of a given model, or its acceptance threshold. If it is low, false rejections (or any rejections for that matter) are less likely. In other words, a model that always accepts the sample will have a FRR of 0.

$$\mathrm{FRR} = \frac{FN}{TP + FN}$$

False Acceptance Rate

False Acceptance Rate, or FAR, is the rate of authentication instances where the model incorrectly accepts the test sample. Similarly to FRR, FAR also depends on the acceptance threshold. If said

threshold is high, acceptance will be rarer, and the same goes for false acceptance. A model with infinitely high acceptance threshold will have a FAR of 0.

$$\text{FAR} = \frac{FP}{TN + FP}$$

Equal Error Rate

Commonly denoted as EER, also known as Crossover Error Rate (CER). If a model has a high threshold, its FAR will be low and FRR high. If the threshold is low, the result will be opposite. EER is used to describe the overall accuracy of a biometric model - the value at which FAR is equal to FRR. Sometimes, one of the error rates is more important in some context. Therefore, in literature, XFAR@YFRR or XFRR@YFAR can be found instead of EER. These indicators correspond to "X FAR at Y FRR" and "X FRR at Y FAR" respectively. For example, in the case of biometric authentication it is usually more important to reduce FAR than FRR - while rejecting genuine samples is an inconvenience to the user of a system, accepting impostor samples constitutes a security breach.

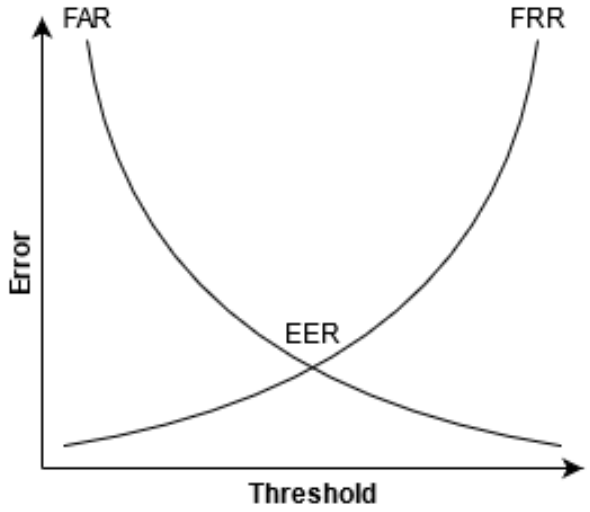


Figure 3.1: The graphical representation of EER. The intersection between FAR and FRR is used to evaluate the overall accuracy of the system with optimal threshold.

Precision

Precision answers the question of how many of the positive identifications were correct. A model that produces no false positives has a precision of 1.0.

$$\text{precision} = \frac{TP}{TP + FP}$$

Recall

Recall tells us what proportion of positive samples was identified correctly. A model that produces no false negatives has a recall of 1.0.

$$\text{recall} = \frac{TP}{TP + FN}$$

F1 Score

Precision and recall partially address the problems that may arise when one simply uses the rate of correct guesses for evaluating a model. Due to the natural trade-off between them, it may prove difficult to compare two separate models. F1 Score is calculated as a harmonic mean of precision and recall.

Using this metric allows to assign a single number to a model's results, enabling comparisons between models with vastly different precision, recall, and accuracy scores.

$$\mathrm{F}_1 = 2 \times \frac{\mathrm{precision} \times \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

The F1 Score metric used throughout this thesis refers to weighted F1 score, whereby the F1 score in multiclass classification is calculated using F1 scores for each class weighted by their support.

k-Fold Cross Validation

Feeding the model a dataset is usually preceded by splitting said dataset into training and test sets. The most common split is 80/20, meaning that 80% of the dataset is used for training the model, and then 20% of the dataset is used for checking whether the model's answers reflect reality, which is the basis of model evaluation. Due to random chance an atypical split can be produced - a split which causes the model to appear to perform exceptionally well or exceptionally bad compared to its actual abilities. The more unbalanced the dataset, the greater the risk of witnessing such an event. In order to reduce such an event's influence on model evaluation k-fold cross validation is performed.

In k-fold cross validation, the parameter k can be set to any integer, but most commonly is set to 5 or 10. In the process, the original dataset is split into k subsets of approximately equal size. Each subset is used as test data for a model trained on the remainder of the dataset. The evaluations of these models are aggregated to show the model's overall metrics, with the bias arising from rare splits greatly reduced.

Previous Work

Keystroke Dynamics has been the topic of research for many years, especially in the context of adding an additional layer of security to password-based login mechanisms.

One of the cornerstones in this field is [17]. The authors had suggested that people's typing patterns may be as uniquely identifying as their handwriting. They proposed a rudimentary statistical classifier, where the distance between the reference and test sample would be compared against a predefined threshold. 33 subjects were selected for the research. They each had to type four strings eight times in the enrollment phase. Then, 975 trials were conducted to test the validator. The authors reported 0.25% FAR@16.36%FRR, using the threshold of 60% of test features within 0.5 of standard deviations from the reference profile.

In a similar vein, the authors of [22] repeated the research and the methods, but in a slightly different context. They had 42 subjects type a group of 15 words 10 times in the enrollment phase. They also collected a brief free-text snippet from each of the subjects. The goal was to evaluate the possibility of verifying a subject typing a brief free-text snippet based on the data from the enrollment phase. Despite having collected far more data than the researchers in [17], and having accounted for context (proven important in [27]), their best result was 0.47% FAR@94.87%FRR. In their own words, that is "not acceptable in a real-world scenario". However, it needs to be remembered that this was applied to authentication based on a very short free-text snippet, with a scarce amount of data from the enrollment phase.

In [30], the authors proposed a scoring algorithm, where matching the keystrokes to the profile is based on the coefficient of variation. While their method is not covered in this thesis, the coefficient of variation is used for finding the features with the most variability. Their method achieved an EER of around 5%.

In [32] the authors proposed that an unknown subject's age and gender could be inferred with high accuracy. The authors focused on guessing a person's gender, employing several models based on different principles: support vector machine (SVM), random forest (RF), naïve Bayes classifier (NB), multi-layer perceptron (MLP), and radial basis function network (RBFN). The best result achieved was the F1 score of 0.956.

Due to more flexible usage scenarios in real-life applications and greater potential for Continuous Authentication, newer research focuses mostly on free-text or mobile devices. Free-text was however not the main focus of this thesis on account of far greater datasets being required for free-text research. Similarly, mobile devices are typically not used in fixed-text research because they are far less convenient for subjects to use when typing a fixed test. Instead, data from mobile devices tends to be free-text because it allows for passive data gathering.

In [27] the authors evaluated the use of various n-grams in free-text authentication. While this thesis does not focus on free-text applications, the fixed-text analysis tends to be less demanding and some conclusions can be compared between the two approaches. In the paper, the authors proposed using probability density functions and Bhattacharyya distance for evaluating whether an n-gram is discriminative. The authors also proposed context-based n-gram analysis, in which the same n-grams are treated separately if they are parts of different words. After performing their analysis supported by roughly 9.5 million keystrokes from 22 subjects, they have concluded that digraphs are not enough for free-text authentication without context analysis. This does not automatically mean that the same applies to fixed-text, but it draws attention to the fact that taking the context into consideration might substantially influence the result.

Similarly, in [15] the researchers conducted an analysis of free-text data divided into 1000–keystroke samples. In their original model the EER was around 6%, with FAR equal to around 1% and FRR over 11%. What is important is that since they used 1000-keystroke samples, it was likely that the adversary would only provide few keystrokes before the next round of verification was initiated. This led to checking

whether mixing impostor samples with the genuine keystrokes would lead to higher rejection rates. It turned out that when about 10% of the keystrokes in the sample were coming from an impostor, the average rejection rate was about 51%. This clearly shows that using impostor samples greatly reduces some models' confidence.

k-Nearest Neighbor was used successfully in [24] and [16]. Both papers dealt with small amounts of free-text data, but their methodology can be carried over to fixed-text.

In [24], the authors captured samples from 63 subjects and used, among others, a regular k-Nearest Neighbor classifier with Euclidean distance. The reported "correct identification rate" was at 83.22%. Adding custom scoring and weights raised that rate to 87.18%. It is unclear how many trials were conducted.

In [16], the authors limited the digraphs to the 50 most frequent ones, and they also used 10 mouse-related features. They took free-text input from 10 subjects, and their results contain several important pieces of information: The keyboard-only and mouse-only methods seemed to have higher accuracy (about 90% and 85% respectively) than the combination thereof (about 70% accuracy); The number of subjects in the dataset has been shown to be negatively correlated with accuracy – the strength of the correlation depends on several factors.

Support Vector Machines were proposed as a faster alternative to Auto-Associative Multilayer Perceptrons in [34]. The authors obtained 75 genuine and 75 impostor samples from 21 subjects and 15 impostors. Then, the SVM with a Gaussian kernel was used for distinguishing between the subjects and impostors. The authors reported an average 17.55% FRR@0FAR for all features, and 6.28% FRR@0FAR when "best feature subset" is selected. According to the authors, training AaMLP with a comparable accuracy is 4 orders of magnitude more time-consuming.

There are several notable works that are out of the scope of this thesis due to severe differences in the goals and methodology. In [31] the authors focus on detecting fraudulent messages with Keystroke Dynamics. A number of works is devoted to improving mobile security using Keystroke Dynamics by itself [21, 26], or combined with data from accelerometers [19], gyroscopes, and other data sources typically used for gait recognition [20]. All of these papers offer an insight into the plenitude of mobile-specific applications of keystroke dynamics, which are rising in popularity in recent years. In [14], the authors focus on applying machine learning to reduce the burden on a user during enrollment.

For more information about keystroke dynamics, please see [29]. The authors evaluated the research done prior to 2013 by aggregating and summarizing the conclusions from 163 different sources, most of which were research papers. They provided reference for further work in the field, as well as outlined the terminology associated with Keystroke Dynamics, common approaches and differences in methodology, and compiled results from the experiments in a single table. The survey is also a milestone, because the vast majority of traditional desktop-based Keystroke Dynamics research was conducted earlier. In subsequent years the focus shifted to mobile devices as well as novel settings and requirements.

For more recent information, refer to [25], where subsequent research is discussed and the practical applications of keystroke dynamics are evaluated, especially in the context of mobile devices.

Data Acquisition

The process of biometric authentication revolves around applying an algorithm to check whether an observed pattern matches a given subject profile. In the case of keystroke dynamics, the pattern is simply the keystroke data presented in a specific way. The profile is created from data samples obtained in the enrollment phase. In order to start working with any model, sufficient data is required. This chapter covers the data acquisition process.

5.1 Existing datasets

5.1.1 .tieRoanl

.tieRoanl[18] is the most well-known keystroke dataset. This dataset published by researchers from the Carnegie Mellon University is frequently cited in keystroke dynamics research. The dataset consists of the data generated by 51 subjects who typed a password `.tieRoanl` 400 times.

This dataset was not applicable in this thesis for two main reasons:

- **The dataset contains too few features** - The public version of the dataset is already processed and contains 21 distinct features. Additional data processing could expand that number to a theoretical maximum of 45 features, however these new features would not add new information. Making accurate predictions based on the keystroke data requires at least an order of magnitude more features.
- **The dataset is already cleansed** - The dataset does not contain any information about errors made during typing. This renders the dataset useless for the purpose of evaluating the methods of data cleansing.

5.1.2 John Vincent Monaco

Several keystroke-related datasets can be obtained from John Vincent Monaco of Pace University[4]. These datasets are significantly more relevant to this thesis, however most of the datasets contained too little data. One of them, the Villani keystroke dataset[28, 23], was relatively large. It contained about $100,000$ keystrokes from over a hundred subjects, along with the basic demographic information. However, the vast majority of these keystrokes were free-text, which naturally creates obstacles in creating and comparing typing patterns of various subjects.

5.1.3 Other datasets

Many other datasets, for example [13, 33], had similar problems - being too small in terms of the number of subjects or features, or simply having been already sanitized. In some cases, the datasets cited in research or surveys [25] were nowhere to be found and could not be evaluated.

5.2 Custom design

As none of the publicly-accessible keystroke datasets seemed appropriate for the task, the only option was to gather the data for this research. The first step would be to decide between free-text and fixed-text. In this case, fixed-text would be the better choice, as it made it easier to find errors in the keystroke data. With free-text, there is no baseline to compare the data to, and so manual analysis is required to detect typing errors. The amount of data required for this research made it infeasible to conduct manual

review. In addition, with free-text there are fewer n-grams common to all subjects, i.e., they do not all type the same things. The result is that it is more difficult to compare various features for several (or all) subjects, because few are shared.

The length of text and the number of subjects was another decision that needed to be made. It was determined that the text should be long - around 800 words – equivalent to 20 minutes of typing at 40 words per minute. The obvious drawback is that the long text would discourage some of the potential subjects from participating. On the other hand, the keystroke data from subjects would be quite detailed and would enable better analysis.

The desired subject count was measured in dozens. Since each keystroke pattern is unique, and more data means better models, the general assumption was "the more the better". As it was hard to determine the number of willing subjects a priori, a decision was made to use all available channels for communicating about the research, see the response, and – if needed – try to find more subjects.

5.3 Crafting the text

The first step was to craft the text that would be typed by the subjects. There were several limitations that defined the entire process. Most importantly, the text had to be coherent English. This was due to the fact that typing seemingly random sequences does not reflect the usual typing patterns of English-speaking subjects. Additionally, seemingly random text could discourage many prospective subjects from typing the entire form. Another goal was for the text to be around 800 words long, split into about 10 separate pieces.

If the text was to be comprised of sentences, it meant that there would also be apostrophes, commas, dots, and capital letters. It made sense to consider whether digits would be useful as well. Including digits in the text meant that a relatively strong distinction could be made between the subjects that use the numeric keypad and the ones who do not, enabling more accurate verification. The downside was, however, that the text would need to include multiple numbers - repeating all digits several times for the training and test datasets. This did not seem reasonable, and thus digits were not included in the text. Punctuation and uppercase characters improved subject convenience by making the text seem more natural, and at the same time did not come at a significant cost, so they became part of the character space.

Creating the text was a creative task, and was not optimizable due to its nature. It was impossible to define an optimal outcome while being limited by the coherence of the resulting text. Therefore, the number of sufficient n-gram repetitions was used as a baseline for evaluation.

At first snippets from public domain books were taken and modified to achieve more n-gram repetitions. After that, some core words and character sequences were selected, such as words `people, street` or suffixes `-ine, -our`, and other fragments were written with them in mind. The public domain fragments were sourced from Project Gutenberg [9].

As the text was being written, various fragments were compared and evaluated using an arbitrary formula. The score was based on the amount of n-grams. Initially, only the number of n-grams of various length (from 1 to 5 characters, and 1-word sequences) was taken into consideration. This, obviously, meant that the score reflected the length of the snippet - with 100 characters, there was always 100 1-character sequences, 99 2-character sequences, and so on. Even discarding spaces and punctuation, this was strictly a linear score that relied on length only. The first adjustment was to compensate for the text length, since the score was supposed to reflect the "quality" of the text, and not its length. This still did not solve the problem, as now the text would have a similar score regardless of its length, but nothing resembling quality was measured.

Assuming that for an n-gram to be properly recorded enough times to be used for training, test, and to compensate for typing errors, 10 repetitions of an n-gram seemed to be a reasonable threshold to make one relevant. Since, theoretically speaking, only relevant n-grams were supposed to be used as data source for authentication, it stood to reason to score only per n-gram with enough repetitions. Similarly, a longer n-gram is more valuable than a short one - after all, the 5-letter word `hours` is far rarer than each of the letters that comprised it, or than common English articles `a, an, the`. Generally, the longer the sequence, the fewer times it can be found in a text. It was decided to factor that into the equation, making the difference between n-grams of various length exponential rather than linear. In the end, the

score for each n-gram, $\text{score}_{\text{ngram}}$ was calculated the following way:

$$\begin{cases} \text{score}_{\text{ngram}} = 2^{l_{\text{ngram}}}, & \text{if } c_{\text{ngram}} \geq t \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

where l_{ngram} is the length of an n-gram (the number of characters for character-based n-grams and assuming 4.79 characters per word [2]), c_{ngram} is the count - a number of repetitions - of an n-gram, and t is a pre-defined threshold at which an n-gram is considered relevant. Here, the examples will assume that $t = 10$. The score of the entire text was the sum of scores for all n-grams considered[1], adjusted for the text length.

In the end, despite adjusting for the text length, the score still favored longer texts. This time it was not due to the length itself, but due to the fact that a long text was more likely to have more n-grams above the threshold. Therefore, the parallels in this chapter will be drawn between fragments of comparable length. The table below contains various metrics related to text evaluation.

		relevant n-grams by length							
Text	Characters	1	2	3	4	1 (word)	Score	Adj. score	Adj. score (no words)
Dickens	4645	20	84	24	5	12	1004.0	1.2125	0.8116
Fitzgerald	4511	19	81	15	2	12	862.0	1.0410	0.6401
Rand	4221	22	73	19	3	12	880.0	1.0628	0.6618
Crafted text	4408	22	77	23	7	12	972.3	1.1743	0.7923

Table 5.1: Texts are denoted by the author's last name - Charles Dickens, F. Scott Fitzgerald, and Ayn Rand respectively. The original fragments can be found in "A Tale of Two Cities" by Charles Dickens, "The Great Gatsby" by F. Scott Fitzgerald, and "Anthem" by Ayn Rand. The crafted text can be read in Rewritten text. The 828-word samples were taken from the same locations in the books as the fragments used in the crafted text. The "Characters" column contains the number of characters in each sample. The following 4 columns contain the amount of alphabetic 1-, 2-, 3-, and 4-grams with enough repetitions within the text sample. The following column contains the number of 1-word sequences with the required number of repetitions. The "Score" column contains the score calculated for each text, which is the sum of individual n-gram scores. The score is calculated for character sequences of length (1, 5), and for 1-word sequences. The adjusted score is the score divided by the number of words (828 in each case). The last column contains the adjusted score, if the scores for 1-word sequences were not used in the score calculation.

It is clear that the crafted text and the sample from Charles Dickens outclass other samples. The word score was not helpful, as for texts this short the words usually included `I, and, the, they, in` and similar, artificially inflating the score. The reason why the crafted text was chosen instead of a matching sample from, for example, "A tale of two cities" was twofold - firstly, the vocabulary in Dickens' book was quite advanced and usually did not contain commonly used words. Examples include `idleness, jostling, regrinding, besmirched, handkerchiefs, embankments, gridiron, ploughed`. It was deemed that asking the subjects with varying levels of English proficiency to type unfamiliar words could significantly influence their typing patterns in the process, adding a lot of noise to the data. The crafted text replaced obscure words with common equivalents, sacrificing the poetic language for utility. The second reason were longer n-grams. Creating sentences around predefined prompts enabled the possibility to include more n-grams of length 4 or 5. The end result was not significant due to the limited length of the texts, but the large amount of sufficient 4-gram repetitions is not a coincidence. At this stage it was still unclear whether that would be sufficient to use 4-grams for prediction.

Another important metric is the relative frequency of n-grams. Using data from Peter Norvig's analysis [2], the typical frequencies of various n-grams can be used as reference. It needs to be remembered that the frequency of particular n-grams is not as important as the general distribution of frequencies.

[1] Character sequences from of length 1 to 5, word sequences of length 1

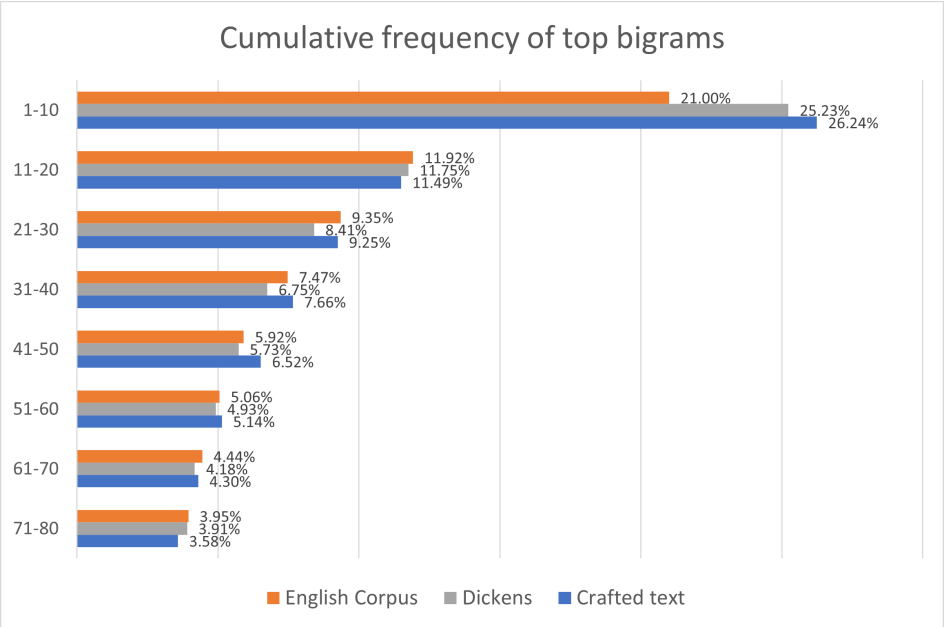


Figure 5.1: Cumulative bigram frequencies. The first row is for the top 10, the next one for the following 10, etc. The English corpus data is taken from Peter Norvig's analysis [2] of over 743 billion English word occurrences

It can be seen on the graph that both the crafted text and the Dickens sample are biased towards the most common bigrams. This can be explained with the fact that both samples are relatively short, and, due to this fact, common English words like `the` will slightly skew the results. The subsequent rows, however, convey an interesting image - in the case of the crafted text, the bigrams in positions from 11 to 60 are more uniformly distributed in the crafted text than in the English corpus, meaning that comparatively rarer English bigrams are slightly over-represented in the text, whereas more common bigrams are used less frequently. In the case of the Dickens sample, the inverse can be observed in this range - the distribution is even more skewed towards the more common bigrams. This can be explained by the fact that the distribution becomes closer to uniform in the range exceeding the scope of the graph. Due to the fact that the samples had 78 and 87 relevant bigrams respectively, it means that for these 80 top bigrams, the crafted text contains the most repetitions with respect to word count. This is also confirmed by taking the cumulative frequency of top n-grams, which for the crafted text is equal to 74.18%, as compared to 69.12% in the English corpus and 70.89% in the Dickens sample.

5.4 Gathering the data

5.4.1 The process

The subjects were invited to visit a dedicated web page of the form which collected keystroke data. The first page of the form was devoted to a disclaimer, which outlined the purpose of the research and described how the data would be stored.

After affirming their consent, the subjects would be presented with the relevant part of the form. The form had four pages with eleven paragraphs of text in total. On each page the subjects would type up to three paragraphs and move to the next page. After switching to the next page, the data from the previous page would be sent to the server and stored. This was done for two purposes - firstly, every page contained typing data equivalent to over one thousand keystrokes. Splitting the load would allow the server to better process the incoming data and would prevent potential timeouts when using the database connection across several concurrent threads. Additionally, subjects who decided that they do not wish to complete the form, for example due to the length thereof, would not have typed the text in vain - the data would not disappear had they decided to cease typing before completing the entire form. It turned out that roughly 20% of all subjects had not finished the form, and their data would have been lost otherwise.

After submitting the keystroke data, the subjects had the option to submit demographic data - their

age group, gender, and dominant hand, as well as a comment if they so liked. This data was optional to submit and was not required for the form to be completed. Roughly 75% of all subjects elected to do so.

Aside from the demographic information, the identity of the subjects remained anonymous.

5.4.2 Technical details

The data was collected remotely, via a website. The server which hosted the website used the Apache server framework [1] and PM2 [8] for process management. The web application was written using Node.js [7] framework with the Express [3] library. The client side was written in HTML5 with Vue.js [10] and JQuery [5] for data collection and sending requests. The database ran on MySQL Server [6].

The keystroke data was collected by monitoring the Javascript events `keyDown` and `keyUp` in the dedicated fields where the subjects typed. After a key was released, the information about the keystroke length would be stored in a collection of events in the form

```
{ 't0': '1616529040468', 't1': '1616529040554', 'k': 'KeyA' },
```

where the fields correspond to time of key press, time of key release, and the key code respectively. In this particular case the information stored pertains to the key `a` being pressed on the keyboard for 86 milliseconds. The data for each paragraph was stored separately.

The data was sent to the Node.js application in POST requests. It was then slightly reorganized and inserted into a database. The table with the keystroke events had five columns: The unique subject ID, the ID of the text, and three columns for the data from the previous example.

Several security measures were implemented to prevent automated attacks from polluting the database. Before adding the data to the database, several metrics were checked:

- Levenshtein distance between the typed text and the source text (checking whether the source text was actually rewritten)
- Ratio of characters to keystrokes (there cannot be fewer keystrokes in the resulting event chain than characters in the source text, unless the subject is using macros instead of typing)
- Time taken to complete the page (even the fastest typists cannot type hundreds of words in several seconds)

The server-side security measures were triggered on two occasions, none of which being the result of an automated attack. Despite allowing for significant margins of error, the security measures incorrectly recognized human subjects as bots in these two instances. The mistakes resulted in losing about three thousand keystroke events in total.

5.5 Outcome

5.5.1 Characteristics of the data

In total, there were submissions for 115 unique subject IDs, 81 of which submitted the demographic information, with 77 having submitted the entire form (eleven paragraphs).

After removing duplicate entries (created for testing purposes) and accounting for some subjects being counted more than once, these numbers are 103, 77, and 77 respectively. The raw data from these subjects contains information about 413749 keystrokes.

5.5.2 Approach evaluation

The primary advantage of crafting a custom text was having a very long text with many n-grams for subjects to type. Custom text also meant being able to add more n-grams or ensure that some more obscure n-grams are repeated enough times. Additionally, allowing for a remote form completion made it relatively easy to reach many people.

Many subjects complained about the text being long - the completion time exceeded 30 minutes in some cases. It can be reasonably argued that many prospective subjects were discouraged from taking part due to the sheer volume of text that was to be typed. The non-laboratory environment meant that some issues could not be controlled - the data provided by the subjects not only contained impurities and errors that were expected to occur, but also some more unexpected keystroke series, e.g., backtracking

with the cursor. Another disadvantage of this particular approach was that some factors could not be controlled or added post-factum. Examples are discussed in the Summary chapter.

Data Processing

All the data processing for this thesis was done with Python 3.8.8, henceforth referred to as Python. Specific non-standard libraries used are be outlined in this chapter. Prototyping was mostly done with the help of Jupyter Notebook.

6.1 Data import

At the beginning, all of the data is stored in several tables in a database, exactly where it was uploaded by the web server described in Data Acquisition.

6.1.1 Database tables

The data is stored in 5 tables. One of them contains UserAgent strings for manual verification and correction purposes, and another one stores the actual texts typed by the subjects for similar reasons. Neither of these two are covered in this chapter, as they are not relevant to the process.

Keystrokes (text)

The keystrokes from the subjects are stored in a dedicated table. It has four columns. The first column is named `SID`, which stands for Session ID and uniquely identifies a given session. It is a 32-character hexadecimal string. With delimiters, the true length of the string is 36 characters. The second column is named `text` and it contains a short integer in the range $[0, 10]$ designating the paragraph (the fragment of text) corresponding to each keystroke. The next column is `key`, which stores the `event.code` of `keyDown` and `keyUp` events. The code is unique for each character on a standard QWERTY keyboard. The last two columns are called `start` and `end` and they store the given keystroke's milisecond UNIX timestamp[11] for the `keyDown` and `keyUp` events respectively.

Keystrokes (comment)

This table is very similar to the previous table, with the exception that there is no `text` column, as each subject submits one comment at most. These keystrokes are stored in a separate table due to organizational reasons only.

Demographic information

The last table stores the demographic information optionally submitted by the subjects. The first column is `SID` to make it possible to connect demographic data and keystrokes belonging to the same person. The second column, `gender`, contains a single character encoding the declared gender. The next column, `age` contains an approximate age range: there were 7 age ranges for subjects to choose from. The `handedness` column contains a single character encoding the declared dominant hand. The last column, named `comment` contains written comments submitted by the subjects.

6.1.2 Database connection

With the help of Python library `mysql`, a database connection is made and the data is imported in its entirety. Firstly, the list of subjects is established with one `SELECT *`, and then subsequently the keystrokes for each subject are imported using a `WHERE` clause matching SID strings.

6.2 Feature extraction

With the raw data loaded from the database, the next step is to turn the stream of keyboard events into meaningful data. The general logic of the tokenization process is outlined in algorithm 1.

Algorithm 6.1: Event data tokenization for a User object. The algorithm takes each text (paragraph typed by the subject) and splits its events into event chains (of consecutive events) of specified length. Then, the event chains are traversed and the n-gram Flight Time information is recorded in a nested dictionary

```
Input: user: User, maxTokenizationDepth: int
initialization;
tokens ← {};
foreach text in user.getTexts() do
    events ← text.events;
    Get all sequences of consecutive events of the specified length from the array;
    eventChains ← getSequentialSubsets(events, length=maxTokenizationDepth);
    foreach chain in eventChains do
        firstKeypress ← chain[0].start;
        previousKeys ← [];
        foreach event in chain do
            key ← event.key;
            previousKeys.append(key);
            Store the timing information under the specified "path" in the dictionary;
            recordTime(tokens, prevKeys + ["times"], event.end - firstKeypress);
        end foreach
    end foreach
end foreach
user.setTokens(tokens);
```

This algorithm builds a tree-like dictionary for each subject. Its maximum depth is k, the parameter supplied as the maxTokenizationDepth in the algorithm.

Such a structure has a lot of redundancy, but its advantage is that the timing of each keystroke series shorter or equal to k can be easily found by traversing the dictionary. For example, the Flight Time of every `the` typed by a person can be accessed under `tokens["t"]["h"]["e"]["times"]`. This also enables checking the times of every subset of that string, as well as checking "similar" strings, i.e., what keystrokes occur after a given sequence, as the subsequent keystrokes are stored in the dictionary entry for `["t"]["h"]["e"]`.

For example, if the parameter `maxTokenizationDepth` is set to k, the depth of the resulting dictionary is also k, and therefore the dictionary contains the timings of every consecutive sequence of k and fewer keypresses, with their respective timings. To further showcase how the information is stored, let's consider the structure of the tree when the depth parameter is set to 3 and the sequence of keystrokes is `[t, h, a, t, c, h]`, assuming each keystroke has a duration of 50ms:

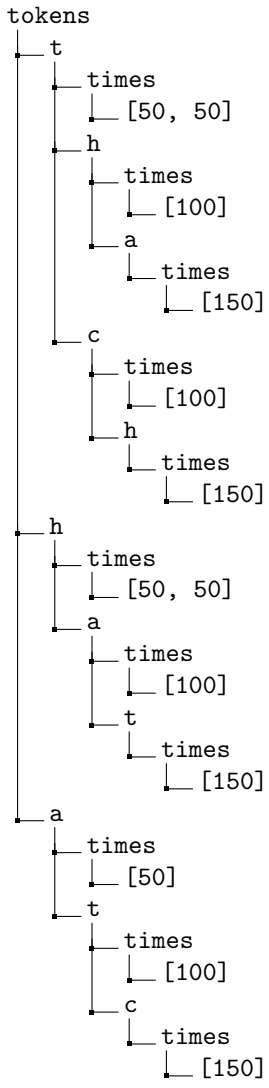


Figure 6.1: Example of a token dictionary. For example, all keystroke Flight Times of the sequence `t, c`, it can be found under `["t"]["c"]`. In addition, this node also contains the keystrokes that occur after `t, c`, provided that the sequence length does not exceed k. Similarly, every latency for `t` can be found under `["t"]`. Note that the order of the children in a given parent node is not relevant, and that some keys may be omitted in the resulting dictionary as per one of the techniques mentioned in section 6.3.

6.3 Data cleansing

6.3.1 Filtering during tokenization

During data analysis it was noticed that some subjects pressed keys that were not supposed to be pressed during their typing session - for example `Digit2, Ctrl, OS` etc. The most likely reason behind these anomalies in the data can be explained by the human factor - typographical errors of this kind are expected to occur when people type. The data for these keys was scarce and unnecessary for the upcoming steps. One way to remove the data corresponding to these keystrokes is to apply a filter during the tokenization phase. In simple words, the tokenization procedure can be altered to omit the current `eventChain` if the `key` does not fit a predefined pattern. An example pattern could be the regular expression `^[a-z]$`, rejecting key that is non-alphabetic or corresponds to something longer than one character.

6.3.2 Removing anomalies

Anomalous data in this context means keystroke latencies that are far outside of the norm for a given subject. The general method is that for each n-gram, if the latency exceeds a certain margin, the keystroke is not taken into consideration. While for singular keystrokes it does not always make sense to consider too short latencies as well, for longer n-grams with, say, mean latency of $380ms$ with standard deviation of around $10ms$ it is a reasonable approach. With higher numbers and longer durations in the dataset, high leverage points can alter the subject's profile too much towards either direction. There are two distinct methods that can be used to remove the outliers.

During tokenization

The first method requires a set-up procedure, where all keystrokes of a given subject are recorded in a data structure, so that during tokenization the average time of each n-gram can be checked. If a given n-gram appears enough times for the analysis to be reasonable (for example at least 10 times) its Flight Time is compared to all of the Flight Times for this n-gram for this subject. If the latency exceeds a certain threshold, for example by deviating by at least 2 standard deviations[1] from the mean, then it is rejected. In addition all the n-grams containing that particular keystroke would also be dropped, preventing poisoned data from being used. The complexity of this solution is $O(n)$ on top of the complexity of tokenization, assuming constant access time to the data structure in question. n is the number of individual keystrokes.

After tokenization

Another method to achieve a similar result can be applied after the tokenization is complete. During the subsequent transformation, the list of keystrokes for each n-gram can be analyzed, removing the outliers (recursively or not, depending on the setting). This leaves other n-gram latencies that include this datapoint[2] alone, provided that they themselves do not exceed their respective thresholds. This is also a quicker method, as the check is performed only once for each n-gram. In practice, the overhead of the recursive approach is heavily dependent on the threshold used in the process - with a low threshold, the recursive calls keep eating away at the freshly cleansed set, reducing it to a fraction of its original size. With a high threshold, repeated recursive calls are rare.

6.4 Data transformation

A tree with tokenized keystroke data, cleansed or not, is required for the next step, in which it is be turned into something more useful for the models. There are three steps: the flattening of the tree, converting the flattened tree to a tabular format, and filling the gaps in the table. The DataFrame class from the `pandas` library was chosen for tabular representation of the data due to its versatility and vast documentation. In this section and the subsequent ones, some thresholds and arbitrary numbers are referred to. They are further discussed in the chapter Results, because in each case an optimal value was found for the data submitted by the subjects. In this chapter only example values are provided.

6.4.1 Flattening (and combining subject data)

The tree-like structure obtained from Algorithm 1 is useful for checking the Flight Times of any given combination of keystrokes, but is not useful for more general analysis. For that, the tree has to be flattened, i.e., the nested data has to be moved to the root level. The tree has two interesting properties: firstly, every series of keys is at the same time the n-gram that corresponds to the node's timing information; secondly, every n-gram typed by the subject has one and only one "path" in the tree. This made the flattening easier. The pipe character "|" was chosen to denote spaces between keystrokes. Therefore, after flattening, the entry that was so far accessible by using `tree["a"]["n"]["d"]` and contained the timing information for the subject's every trigram "and", is now moved to `flatTree["a|n|d"]`. Another thing done during this process is combining the trees of all the subjects. To use the same example, subject

[1] This is an example - the optimal numbers are discussed in the chapter Results

[2] Meaning that if an abnormally long "an" gets removed, the "and" with this "an" is only removed if it itself exceeds the margins for "and". This is different from the previous method, whereby if an "an" is found to be abnormally long, all longer n-grams that include it are also removed

`ABC`'s data for the aforementioned trigram is now stored under `flatTree["a|n|d"]["ABC"]`, meaning that the flattened tree contains all the keystrokes of all the subjects in a format that resembles a table.

6.4.2 Conversion to a DataFrame

Datapoints

One of the ways to create a table given the data from the flattened tree is to map every pair of keys (subject, N-gram) to a value and then put it in a DataFrame. The DataFrame contains all the keystroke latencies of each subject, with the "User" column for the corresponding subject's ID, and with the number placed in the column describing the n-gram in question. Such a dataset has c columns, where $c-1$ is the number of elements in the flattened tree's root[3]. It also has k rows (with one number in each row), where k is the total number of keystrokes in the dataset. It is a large, mostly empty, table. A transformation can be applied to "merge" the entries where different columns were filled. This still results in a table that has a lot of empty space, because if there are, for example 200 keystrokes for the key `e` but only 20 sets for the bigram `en`, then the column `en` is expected to be empty in at least 90% of the rows. This can be remediated by filling in the gaps, which is described in the next subsection.

Aggregations

Some models require aggregated data - in particular the mean and standard deviation of each n-gram for each subject. The process of aggregation goes as follows: the flattened tree is traversed just like in the process of flattening the original tree, and at each "stop" the mean and standard deviations are calculated for each n-gram. If the number of repetitions for a given n-gram is too low, the n-gram is discarded. This results in a table where each row corresponds to one subject, and each column corresponds to one n-gram's mean or standard deviation. Then, the columns with too many empty fields are removed, which leaves a mostly full table.

6.4.3 Filling the gaps

Datapoints

The models using the table with datapoints cannot tolerate empty fields, and so all gaps have to be filled in this case.

For the table with datapoints, there are two steps that need to be taken: firstly, filter out the rows which still have a lot of empty fields (i.e., the ratio of empty fields exceeds some threshold value), and then fill the remaining values. Removing the rows which are too empty is based on an arbitrary value, and there are three methods of filling in the remaining fields. The first one is to set the value of the field to the mean of the entire column (mean of all Flight Times for the same n-gram). The second one is setting the value to the mean of the entire column where the subject is the same (mean of all Flight Times for the same n-gram for the same subject). The last one is setting the empty values to something arbitrary, for example 0. Their respective effectiveness is discussed in Results.

Aggregations

For the table with aggregations there are also several options of filling in the missing values, though the models using the aggregations do not explicitly require all fields to be filled.

One option is to simply ignore them - if there are relatively few empty fields in the table, this should not affect the results greatly. Another option is to set the value to a row average - an average latency for the same subject, or an average latency for n-grams of the same length for the same subject. This somewhat reflects the timing of a particular subject, though very inaccurately. The last option is to enter the column mean - the average mean or standard deviation across all subjects, which partially neutralizes a particular column's influence on the results returned by the models.

[3] One additional column is required for the subject ID

6.5 Feature selection

There are several optional methods which make it possible to further limit the amount of features in the tables. One of them is to select k "top" columns, thereby limiting the number of columns to a predefined integer based on some statistical features of the dataset. In literature, the "top" features usually refers to the ones with the most repetitions in the typed text. In this thesis, the algorithm for finding top features does something different - the coefficient of variation is calculated for each feature, and the ones with the highest coefficients are selected. While this is an imperfect measure, it reflects the variation within the feature itself and allows the model to use the features for which there are the greatest differences between subjects. The calculations required for this operation scale linearly with the number of keystroke events.

Another option is to limit the range of features by applying an n-gram length filter, i.e., removing all columns but the ones corresponding to bigrams, or trigrams, or any combination of possible values of n in the n-grams. This way, comparisons can be made between, say, the accuracy of the models using only bigrams, the accuracy of using only trigrams, and the accuracy when both kinds are used.

6.6 Progress storage

Most of the import and processing took a long time – the exact figure varied depending on the scope (full dataset or a subset used for development and testing purposes) and some of the parameters, but usually it was measured in minutes. This meant that it was not feasible to perform the whole process with each test during development. The state of certain objects was saved by using the Python library called `pickle`, which enables storing Python objects in binary files and later retrieving them. Saving progress at each step of data processing enabled smoother development and reduced the initialization time to around $50ms$ in the best case and several seconds in the worst case, either way significantly reducing the overhead required for efficient development, running quick tests, and making adjustments.

Models

7.1 Aggregation-based

These models take the reference samples for each subject and aggregate them, calculating the mean and standard deviation for all n-gram latencies. Then, test samples are treated the same way and the models make their decision based on essentially two condensed datapoints for each subject.

7.1.1 Majority vote

In this model, for each reference sample s, standard deviation σ_{f_s} and mean μ_{f_s} are calculated for each feature f. In the case of test samples only the means $\mu_{f_{test}}$ need to be calculated. Then, for each feature f in the test sample, the closest (in terms of the number of standard deviations away) reference sample for the same feature f is found. The model then returns the reference sample s with the most matches when trying to identify a subject, and in the case of authentication returns True or False based on whether the number of matches exceeds a predefined threshold t. The threshold can be optimized for efficiency.

The advantage of this method is that theoretically, if the subject s maintains the same typing style when submitting the test sample, the patterns for reference and test samples should be quite close, yielding the most matches between the actual subject.

Unfortunately, in practice there is a lot of variability in a specific person's typing patterns. This is something that greatly weakens all aggregation-based methods. Additionally, if the subject space is large enough, there is bound to be many samples which are coincidentally close to the test sample in the case of certain features. These samples will reduce the true subject's "score", worsening the model's performance on large datasets. In addition, if two subjects have very similar typing patterns, they will "steal" matches from each other, potentially enabling another subject with worse overall match to be selected by the model.

It is a simplified form of the model described next. Its goal is to check how much the simplification will affect the results.

This is a naïve model relying on strong assumptions. Its simplicity means that implementation and understanding it are not a challenge, and it does not require complicated calculations.

7.1.2 Sieve

This model takes the test sample and the reference sample belonging to the subject and compares the μ_{f_s} and $\mu_{f_{test}}$. If the samples are sufficiently close, i.e., at least t features are within d standard deviations from the reference values, then the test sample is accepted. Otherwise, the model rejects the sample.

The strength of this model comes from the fact that while a subject's typing patterns are subject to some changes, the variability should not be significant. Therefore, the model is expected to detect unique patterns in the keystroke data. In addition, parameters t and d can be tweaked to improve the accuracy of the model.

The weakness of this model stems from the fact that it still relies on the same assumptions as the Majority vote. Another caveat is that the optimization of parameters may not be possible – a group of subjects with similar typing patterns in the dataset will force the thresholds to be low, potentially greatly increasing the model's FRR.

The model is directly based on the model used in [17, 22]. The parameters will be optimized separately. Due to its simplicity and lack of reliance on highly specialized tools, it does not have a specific name. It is referred to as "statistical model". To avoid ambiguity, it is named sieve in this thesis. The name reflects the process of separating the "close" and "far" features.

7.1.3 k-Nearest Neighbor

Also known as KNN, k-Nearest Neighbor is a statistical technique widely used in classification problems. It maps the training samples with d features to a d-dimensional hyperplane. Then, for each test sample the nearest k neighbors are found and the model's verdict is based on the labels assigned to these neighbors. The most common linear method of calculating distance between points is Euclidean distance. In the case of aggregations, for identification or authentication only 1NN makes sense, as there is one training and one test sample for each subject. In the case of demographic classification, k can be larger than 1 and can be tweaked for optimal results.

The strength of this model lies in the fact that all features are taken into consideration equally. Technically, the problem does not need additional data inputs or parameter tuning – the only things that can be influenced are the methods for calculating the distance and the value of k. In either case, there is little room for optimization, and the theoretical foundations of KNN make it far stronger than simple statistical analysis.

For large feature sets, the curse of dimensionality applies, i.e., the datapoints in high-dimensional space seem sparse and dissimilar, negatively affecting their analysis. In addition, since the method relies on points in a d-dimensional space, missing values in the vectors have to be somehow filled in – in the case of aggregations this is no easy task. The solution for both of these problems is to vastly reduce the feature space, discarding a significant amount of data in the process.

kNN was successfully used for keystroke dynamics in [24] and [16].

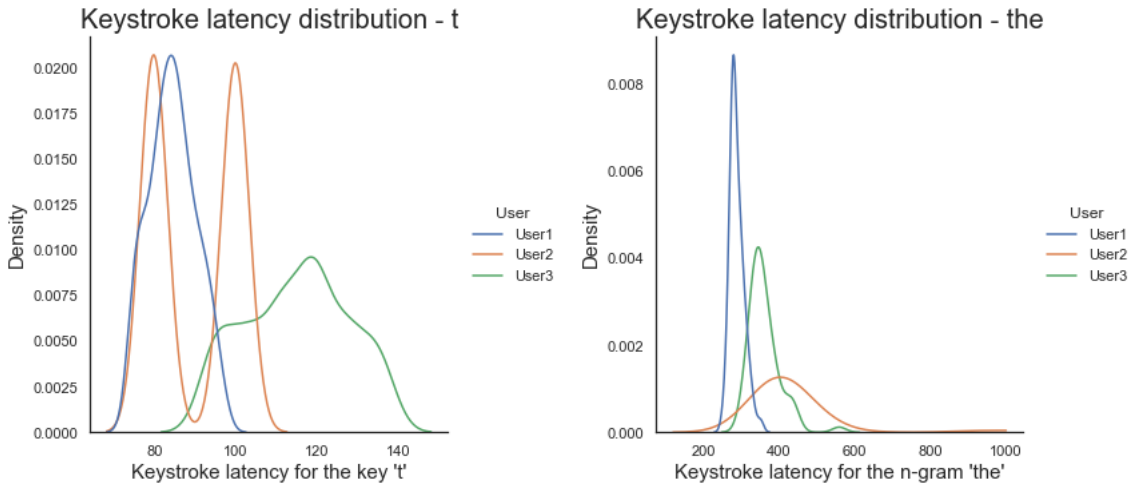


Figure 7.1: The problem with aggregations is connected with two phenomenons. Firstly, they are based on an assumption that the keystroke latencies follow a specific distribution – for example, normal distribution. The example shows keystroke latency distributions for two n-grams: `t` (single key) and `the`. In the first case it is clear that only one of three presented subjects types the key `t` in a manner resembling normal distribution. In the case of User 2, their mean for that key would be around 90ms, a value which is rare for them. Additionally, there are multiple intersections, making it impossible to accurately identify the person who types. This problem could, in theory, be alleviated with more data and more features, but only partially. In the case of the longer n-gram, all of the distributions resemble normal distribution, but again there is a clear intersection between these distributions. With about 100 subjects in the dataset, analyzing even 100 such graphs for various features might not yield conclusive results during an identification or authentication attempt.

7.2 Datapoint-based

These models use multiple datapoints per subject. The keystroke latencies are presented in a tabular format, and then processed so that each subject has multiple datapoints associated with them, each number in the datapoint boiling down to a single n-gram latency.

7.2.1 k-Nearest Neighbor

KNN is already described in the context of aggregations in the previous section. As for the datapoint-based format, there are several key differences. Firstly, the parameter k is always configurable, due to the fact that there is more than 1 sample per subject. Secondly, the requirement of having no empty values in the d-dimensional vectors is even more problematic, and there are also far more values that need to be filled in.

7.2.2 Support Vector Machine

It is a supervised learning model for binary classification problems. In short, a support vector machine takes the d-dimensional observations and tries to find such a $d-1$-dimensional hyperplane that produces the cleanest split of the two classes. SVM has many configurable parameters – kernel, kernel-related parameters, optional weights for each class etc. It is a highly configurable, relatively high-accuracy and low-cost alternative to neural networks.

Given a multiclass problem, the model splits it into multiple binary classification problems. Depending on the approach, the split can result in $n-1$ problems (One-versus-Rest) or $n \times \frac{n-1}{2}$ separate classifiers (One-versus-One).

Support Vector Machine with a Gaussian kernel was used for keystroke dynamics authentication in [34]. Its authors also proved that SVM can be as effective as Multilayer Perceptrons, requiring only a fraction of their training time.

Results

In this chapter, the results of various tests are presented and discussed. The tests were all conducted on a Dell Inspiron 5577 with an Intel® Core™ i7-7700HQ processor running at 2.80 GHz and 16GB of DDR4 RAM.

Due to random fluctuations in resource usage, the timing information cannot be presented with high accuracy. Therefore, the processes that last under a second were executed multiple times, the time per single run being approximated. Moreover, to be able to put the numbers in context, the timing information will also be presented in time per 1000 keystrokes when discussing data processing, or time per sample (assumed to be 1000 keystrokes long) when comparing models. It is the number used in [15], however the cited research focuses on free-text. In both cases, however, it needs to be remembered that the exact sample size is heavily dependent on the usage scenario. In the case of continuous authentication, a balance has to be found between the authentication frequency, authentication accuracy, and the time taken for the process to be complete once a test sample is generated.

8.1 Optimizing parameters

Parameter optimizations are done in steps. The tables included in this section do not reflect the final results – they merely show the accuracy trends given various factors. In order to avoid brute-force tests, the optimizations are evaluated sequentially. First data cleansing methods are compared to see which one has the most potential. Then, various methods for replacing missing values are compared to gauge which one is the most promising. After that, all the models are optimized for working with the data processed using the methods proven to be the best in the former steps. For optimized models, it is then checked whether feature filtering helps improve the results. The best possible scores are outlined in Results – summary. While such a greedy approach may not yield the globally optimal results, it is capable of providing results that are sufficiently good and showing the influence of various techniques on the models' accuracy. Compared to a brute-force search, such an approach reduces the number of tests in this thesis by several orders of magnitude.

8.1.1 Data cleansing

The first step that can be optimized is the data cleansing process, in which abnormal samples are removed from the dataset. The abnormality is measured in standard deviations from the mean. If the number of standard deviations exceeds a threshold t, a keystroke (or a series thereof, depending on the method) is removed from the dataset. The table below contains accuracy results for various methods.

		During tokenization			After tokenization		
	No cleansing	t = 1	t = 2	t = 3	t = 1	t = 2	t = 3
Sieve	0.398	0.646	0.475	0.378	0.031	0.169	0.311
SVM (zero-padded)	0.635	0.939	0.795	0.699	0.928	0.892	0.742

Table 8.1: F1 Score for various data cleansing techniques. The NaN datapoints for the SVM model were replaced with zeros for this test. Each of the cleansing techniques is tested for several possible thresholds, represented as the number of standard deviations from the mean. The datapoints outside of this margin are discarded as explained in Data cleansing.

It can be seen that lower values of t seem to yield better results. Interestingly, the SVM model overall performs slightly better when only the anomalous keystrokes are removed, and the n-grams with them are kept, though for low values of t this method does not yield the best result. Possibly this is due to the

fact that for longer n-grams the abnormalities are not significant enough, and when they are, they are removed all the same. In the case of aggregations, it can be clearly seen that this method is significantly outperformed by removing the anomalies during tokenization, along with their descendant n-grams. Due to its overall superiority, the data processed with the cleansing technique of removing n-grams during tokenization with $t = 1$ will be used for other tests.

The base time for processing the entire dataset was in the order of 160 seconds (for a total of 106 subjects). With cleansing during tokenization, for $t = 1$ the time was in the order of 60 seconds, and for higher values of t, the time did not noticeably differ from the baseline. With cleansing after tokenization, the same relation was observed, with the time for $t = 1$ being in the order of 50 seconds. It can be argued that for $t = 1$ a significant number of datapoints was removed during tokenization, and therefore the latter stages were much faster.

8.1.2 Replacing missing values

Aggregations

The tables containing aggregations need to be filled in, because in some cases there are too few repetitions per subject of a given n-gram. This may occur, e.g., due to typographical errors.

	None	Row-based	Column-based	Row- and n-gram-based	Row- and n-gram-based AND Column-based
Majority Vote	0.0639	0.1169	0.1217	0.0857	0.0857
Sieve	0.3452	0.3937	0.1807	0.1570	0.1464
KNN	N/A	0.1187	0.1321	N/A	0.2642

Table 8.2: The F1 scores for Majority Vote, Sieve and Aggregation KNN. Each column corresponds to one way of filling in the missing values. The first method is to not fill them at all, in which case KNN cannot be used. The second method is filling all the empty values with the row means (mean keystroke latency for the same subject). Another method is filling in the empty values with column mean (mean for the same n-gram). The fourth method is using a given subject's mean but of n-grams of the same length, e.g., a missing value for `and` will use a mean for tri-grams for the same subject, whereas a missing value for `ut` will use a mean for bi-grams for the same subject. The last method combines the fourth and third methods, since the fourth method does not suffice to fill everything by itself in several rare cases.

According to the data from the table, every model has its optimal conditions, which are not easily deducible. The best approach would be to match the method with other parameters to find the optimal set of conditions for each model, as these results do not suggest a clearly superior solution.

Datapoints

The tables containing datapoints always need to be filled in, because some features are less numerous than others, which causes some timings to be missing in the processed dataset.

It can be clearly seen from the table 8.3 that using the subject average for an n-gram is the best possible choice. It also seems that using zeros is the better approach than the column mean. Perhaps, it is because using zeros reduces an n-gram's significance in the model, whereas using n-gram mean across all subjects brings all datapoints closer together, increasing the rate of false positives.

These results already look promising. It needs to be stressed that in each test case there were 2132 samples used for testing, which was 20% of the original sample count, so the promising results are reliable.

The time required for the models to perform the authentication is negligible in this case. Preparing the entire dataset for the models (replacing missing values and removing excessive entries from a processed table) is in the order of tenths of a millisecond, and so is the highest per-authentication time required by any of the models: for SVM that was between 1 and 2s for all 2132 authentications. For KNN, that number was in the range of 200-500ms in total.

0.95 was set as the threshold for empty values in the original table, meaning that each row (and then column), where over 95% values were empty was removed. The remaining empty fields were filled using the aforementioned methods. In the process the dataset shrank from the original 32625 entries with 214

	Zeros	Column-based	Column-based (same person) AND Zeros	Column-based (same person) AND Column mean
KNN	0.5122	0.4311	0.6898	0.3616
SVM	0.6163	0.6102	0.7310	0.4705

Table 8.3: The F1 Scores for KNN and SVM models. Each column corresponds to one way of filling in the missing values (after prior removal of excessive entries). The first method is filling all empty values with zeros, the second method is filling all empty values with the mean for their respective columns (for the same n-gram across all subjects). The third and fourth methods first fill the empty fields with the corresponding column mean for the same subject (within the training or test dataset), and then fill the remaining empty fields with zeros or the mean for the entire column respectively. Using just the user mean does not fill all of the empty fields in the case of some n-grams.

features to 10660 entries with 72 features.

8.1.3 Model parameters

In many cases, the models themselves are adjustable in terms of some parameters. For each model, the optimal operating parameters are found at this stage.

Majority vote

In the case of the majority vote, only one parameter can be optimized: the threshold for authentication. The parameter would then decide how many "votes" would be sufficient to successfully authenticate a sample.

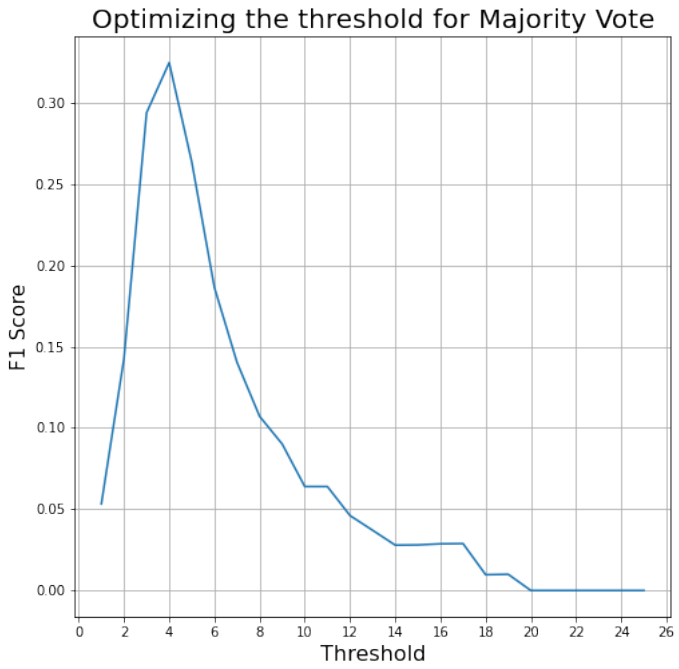


Figure 8.1: The F1 score of the Majority Vote classifier for various thresholds. Number of trials for each value: 11236.

From the graph it is clear that the optimal threshold for this dataset is 4, which results in F1 Score of 0.325. While it is not nearly comparable to other methods, this is a very rudimentary model, and as

such cannot be expected to perform on par with the ones used in literature. Still, it is not a necessarily bad result given the simplicity of the model.

Sieve

With the sieve model, there are two parameters which can be optimized – the percentage of columns that need to be within a threshold to pass and the threshold itself. The values proposed in [17] were 60% and 0.5 standard deviations respectively, and they were used as a baseline for testing.

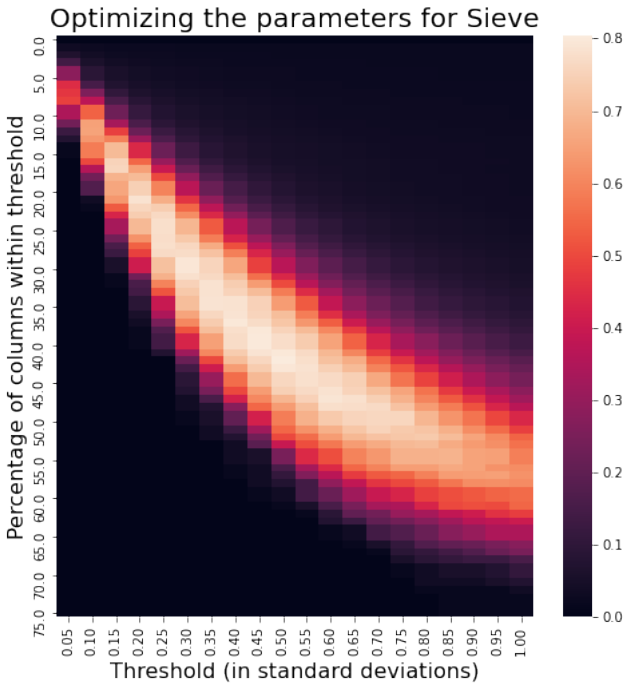


Figure 8.2: The heatmap of F1 scores for various parameter combinations (tested on 5929 samples). The threshold for "accepting" a column is on the X axis, ranging from 0.05 to 1.0 standard deviations in increments of 0.05 (preliminary testing has shown that values greater than 1 standard deviation do not produce promising results). The percentage of columns that need to be within the threshold ranged from 1% to 75%, in increments of 1. It is clear that for near-optimal values for one parameter, the "stripe" is wider, meaning that the model's tolerance for suboptimal values of the other parameter is greater. As the parameter values get farther away from the optimum, a "window" of acceptable values for the second parameter gets narrower

In the tests, 1500 combinations of parameters were tested. The combination yielding the best F1 score was 42% and 0.5 standard deviations, with a result of 0.804. From the graph it is clear that the two parameters are somewhat correlated, i.e., that raising one requires raising the other one to preserve a similar score.

Interestingly, for nearly any reasonable value of one of the parameters, the other parameter can be set such that the F1 score exceeds 0.6.

KNN

In the case of KNN, there are two parameters that can be optimized. Firstly, the number of neighbors. Secondly, the distance function. 200 tests were conducted, with 2132 trials each, to find the optimal parameters. The values for neighbors in the range [1, 100] were tested, as well as two distance functions - Euclidean Distance and Manhattan Distance. In this subsection only datapoint-based KNN is considered.

The overall result achieved with 10-fold cross-validation was 0.7902 for 3 neighbors and Manhattan distance, which were found to be the optimal settings.

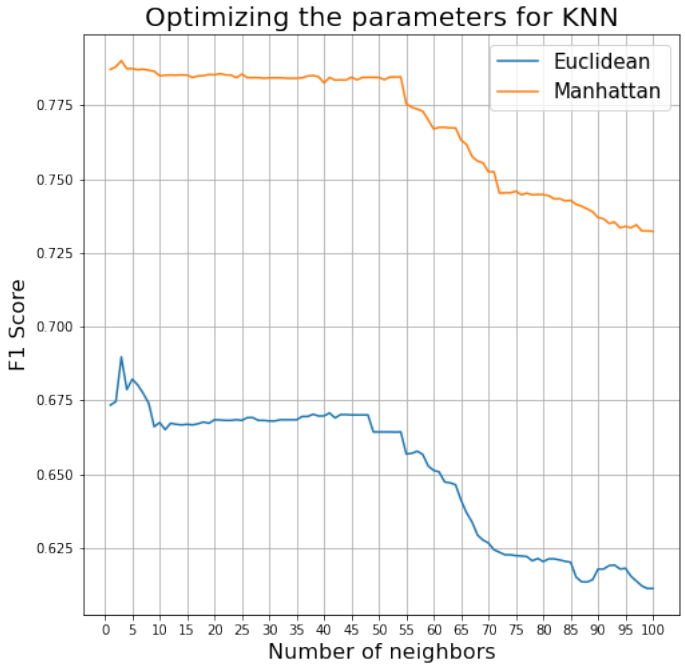


Figure 8.3: The F1 score of the KNN classifier depending on the number of neighbors (in one of the test runs). Separate plots for Euclidean Distance and Manhattan Distance.

It can also be generalized that Manhattan distance seems better suited to the task, and the number of neighbors does not significantly affect the result, provided it is in the 10s to 50s range.

Given that there were 2132 test samples ($\frac{1}{4}$ of the training dataset) per 106 subjects, it means that for each subject there were about 20 test samples and 80 training samples on average[1]. The results suggest that the subject's test and reference samples were close enough, especially using the Manhattan scale, and so the test samples were properly classified. As the number of neighbors increased, the reference samples slowly started "running out" for some subjects, causing the model to get progressively worse at around 50 neighbors.

The takeaway is that for other applications (e.g., demographic classification) the neighbor limit might be higher due to a larger number of samples for each class.

The demographic classification (for datapoints) was a very similar problem, albeit with far fewer classes. The easiest approach would be to map the demographic features to users and repeat the classification, but using the demographic classes this time. The obvious problem with this approach is that if a given subject's data is in both the training and test set, the model will heavily rely on subject identification, rather than demographic features for its guesses. To counteract the problem, the training and test sets have to be split in a way that ensures that each subject's data is in one of them only, to ensure that the model infers based on the underlying similarities between different subjects.

The best results for demographic inference (confirmed with 10-fold cross validation, and removing groups with fewer than 10 subjects) were as follows:

- Handedness: F1 score of 0.4797 for 3 neighbors and Manhattan distance (1853 samples in 2 classes)
- Age: F1 score of 0.3955 for 2 neighbors and Euclidean distance (1731 samples in 3 classes)
- Gender: F1 score of 0.7157 for 6 neighbors and Manhattan distance (1832 samples in 2 classes)

It is clear that KNN performs poorly in the case of handedness inference. For age, it needs to be noted that the dataset was heavily biased in favor of one of several classes, with over $\frac{1}{2}$ of all samples

[1]The total number of samples submitted depended on the user. The values ranged from 22 to 139 of total samples per subject, with a mean of 100, a standard deviation of 35.45, and a median of 117. The samples were divided into test and training samples with a 20/80 split.

belonging to just one class. In the case of gender, the support split was 615/1217. While the result for gender classification is clearly better than a coin toss, accounting for the unbalanced data makes the result slightly less impressive. Nonetheless, these results suggest that at least a person's gender can be inferred with relative ease using keystroke analysis.

SVM

In the case of SVM, there are three hyperparameters that will be optimized in this thesis. The first one is kernel: the kernel reflects a decision boundary used by the SVM. Some kernels take parameters, and in such a case they too need to be tested. Additionally, if one is aiming to keep FAR or FRR at a specific level, weights can be altered to ensure that. In addition, a Support Machine Classifier can have two different shapes of the decision function. They are called OvR and OvO - one versus rest and one versus one respectively. OvR splits an m-class problem into $m-1$ binary problems, creating "One versus rest" classifiers in the process, whereas OvO splits the same problem into $\frac{m\times(m-1)}{2}$ binary class versus class problems.

Weights will be mentioned again when comparing the results with the literature, but other parameters can be optimized at this stage.

None of the tested parameters seemed to have an effect on the linear kernel - the overall F1 score was 0.7390.

When testing the polynomial kernel, the degree of the polynomial (in the range 2 to 8) only very slightly affected the F1 score, favoring the lower values. Other factors seemed to have no impact on the result. The overall F1 score was similar to the score of the linear kernel - 0.7121

The results of the RBF kernel were slightly worse, with the F1 score equal to 0.6900 for the optimal parameters of: `{"C": 100, "decision_function_shape": "ovr", "gamma": 1e-08}`.

Henceforth, the linear model will be used when referring to SVM-based authentication in this thesis due to its superior results.

The time taken by the linear model is slightly below 1ms per authentication, and that includes the time required for fitting the data to the model. The time taken by the polynomial model is comparable.

When it comes to demographic classification, the RBF kernel was certainly better than Poly. Linear kernel took an enormous amount of time for these problems and was therefore not considered. The demographic classification results are as follows:

- Handedness: F1 score of 0.7451 (1853 samples in 2 classes)
- Age: F1 score of 0.2523 (1731 samples in 3 classes)
- Gender: F1 score of 0.7143 (1832 samples in 2 classes)

It is clear that, except for age, SVM is good at inferring demographic features. It is especially good at inferring handedness, surpassing KNN by about 0.27 in terms of F1 score for that category. The testing set was almost an even split (930/923) in terms of one's handedness, meaning that these results are not skewed by one large class.

8.1.4 Feature filtering

It needs to be checked whether the ensemble of various n-grams is a hindrance to the models or not. This can be done in two ways: Limiting the n-gram space to, say, only bigrams to see if this will increase a model's accuracy, or selecting only "top" features from a subset. The "top" features are often the ones with the most occurrences in the literature, but they may just as well be the ones with the highest coefficient of variation across subjects.

Given that there is a performance gap between authentication and demographic classification, it may so occur that feature filtering will be purpose-specific, e.g., improving a model's results only for authentication or demographic classification, or maybe even only for one kind of classification. Therefore, optimized authentication and demographic classification variants will be tested to see if feature filtering can improve the results.

Limiting n-gram space

In the test, the feature space was filtered based on the n-gram length. In the cleansed dataset, there are 80 features for aggregations, and 72 features for datapoint-based models. In aggregations there are

22 features corresponding to 1-grams, 52 2-grams, and 6 3-grams. Longer n-grams had been filtered out by this point. For datapoints, these numbers are 22, 46, and 3 respectively.

	Length of N-grams used					
	1	2	3	1,2	2,3	1,2,3 (baseline)
Sieve	0.3693	0.0370	0.1006	**0.4412**	0	0.4296
KNN (Auth)	**0.8649**	0.7571	0.1053	0.7736	0.7658	0.7902
KNN (Handedness)	**0.5997**	0.4425	0.4622	**0.4833**	0.3840	0.4797
KNN (Gender)	0.5972	0.6772	0.5748	**0.7330**	0.6397	0.7157
SVM (Linear Auth)	**0.8673**	0.7221	0.1113	**0.7451**	0.7074	0.7390
SVM (RBF Auth)	0.1639	0.7050	0.0370	0.7277	0.6955	0.7353
SVM (Handedness)	0.5394	0.4829	0.4283	0.7057	0.5744	0.7451
SVM (Gender)	0.5588	0.5417	0.5303	0.6943	0.5926	0.7143

Table 8.4: Approximate F1 scores for various models. Clear improvements over the baseline are in bold. Auth means authentication, anything else in the brackets means demographic classification for a given feature. In the case of SVM, both the linear and RBF kernel were used to showcase the differences that were not seen in the parameter optimization phase. For demographic classification, the RBF kernel is used. The results were obtained without applying k-fold cross validation, because their purpose was to show the changes in the model efficiency based on applied filters. Therefore, they are only approximations of the actual model accuracies.

It seems that focusing more on 1-grams (single keystrokes) or excluding 3-grams in the case of KNN yields better results. The same can be said about SVM with a linear kernel. For all other models it seems that more features yield better results. The RBF kernel in SVM seems to fare exceptionally bad with non-bigrams.

Top features

In the previous example, it was shown that feature filtering proved mostly helpful for the KNN demographic classifier. However, another filter might be able to improve other models' results

In this test, the feature space (originally 80 features for aggregations and 72 features for datapoints) is further reduced. There are several ways to do this. For example, just the most common features can be left in the dataset as is frequently done in the literature. Such an approach favors shorter n-grams, as they are more plentiful. When limited to a constant length, such an approach will produce better results, however it has already been shown in this thesis that limiting the feature space to n-grams of specific length generally worsens a model's performance.

The results from figure 8.5 support the hypothesis that utilizing more features (at least in the scope of this thesis) generally tends to improve a model. There is, however, one exception.

It seems that the KNN handedness classifier leans heavily on the top 10 features for its assessment, and that they correctly reflect the underlying differences. Therefore, further testing is required to establish which features are discriminative of one's handedness.

	Number of features chosen					
	10	20	30	40	50	Baseline
Sieve	0.0993	0.0175	0.0187	0.0187	0.0187	0.4296
KNN (Auth)	0.2944	0.4887	0.6387	0.7334	0.7672	0.7902
KNN (Handedness)	**0.7341**	0.3051	0.3836	0.3267	0.3708	0.4797
KNN (Gender)	0.4251	0.6106	0.5014	0.6921	0.6759	0.7157
SVM (RBF Auth)	0.2804	0.4910	0.5883	0.7027	0.6969	0.7353
SVM (Linear Auth)	0.2849	0.5361	0.6119	**0.7408**	0.7084	0.7390
SVM (Handedness)	0.3507	0.3124	0.7218	0,5052	0.6446	0.7451
SVM (Gender)	0.6077	0.6359	0.4786	0.5476	0.6676	0.7143

Table 8.5: F1 scores for the models used in the previous examples, with the same baseline. The sieve model seems to behave somewhat randomly, but all other models tend to improve given more features. One clear exception is the KNN demografic classifier for handedness.

8.1.5 Comparison with the literature

Sieve

The sieve model closely resembles the technique from [17] and [22]. In [17], the top result was 0.25%FAR@16.36%FRR. In [22], it was 0.47%FAR@94.87%FRR.

Using the suggested parameters of 60% and 0.5 standard deviations, the best achieved result is 0FAR@87%FRR. Using optimization, the best two results are: 0.804 F1 Score with 1.37%FAR@32.1%FRR or
0FAR@40.6%FRR. The model therefore has worse performance compared to [17] and better performance than [22]. The difference between this thesis and [22] can be explained by the fact that in [22] the dataset was smaller and it was free-text.

In the case of [17] there were several methodological differences. Firstly, the data was not aggregated, but instead multiple samples per subject were used. This alone can explain the discrepancies in the results. Secondly, the impostor samples were not taken from the users that were to be authenticated. That is, there were two sets of subjects - genuine users and impostors. In this thesis all subjects were treated as genuine users.

KNN

In [24], the authors claimed the "correct identification rate" of 87.18% in the best case. It is unclear what they meant by "correct identification rate", but with KNN every test is an identification test, so most likely they were referring to overall accuracy or their F1 score. Similarly, in [16] the reported accuracy was at the level of 90%. As shown in the previous section, a similar accuracy has been achieved in this thesis.

It is difficult to assess how much data the authors of [24] were using, as they had only mentioned gathering free-text data from 63 subjects over the course of 11 months. It is likely to be a sizeable dataset, but in the case of free-text, a lot more empty values need to be filled in, which is crucial for the performance of a KNN classifier. It is unclear how the authors dealt with that problem. Similarly, it is unclear how many samples were used for testing and where they were sourced from.

In the case of [16], the authors disclose that 52 keyboard features were used and 20 subjects participated in the research. While their aim was somewhat different from the aim of this thesis, they have shown an overall accuracy of 90% using KNN with just the keyboard features, which is comparable to keystroke dynamics applications in this thesis.

Overall, it has been shown in this thesis that for a large dataset of fixed-text keystrokes (without manual error correction during data acquisition), the KNN classifier can be optimized to an accuracy of at least 86.5%, as the best recorded result was an F1 score of 0.8640 with an EER of about 12.4%.

SVM

In [34], the authors used 150 samples obtained from 36 subjects for testing. They used SVM with a Gaussian (also known as RBF) kernel and reported a result of 17.55% FRR@0FAR for all features, and 6.28% FRR@0FAR when "best feature subset" is selected.

The optimized SVM model discussed in this thesis had an F1 score of 0.8660 with an EER of about 13.2%

While the SVM models from this thesis have not been able to reach the numbers cited in the literature, they are quite promising. Further analysis is needed to improve their performance in this case.

8.2 Distinguishing characteristics

The results obtained in the previous section show that an individual's typing style can easily be recognized by a well-trained model. In addition, some demographic features (such as age, handedness, or gender) can be inferred with some accuracy, given an appropriate model and parameters. In this section some underlying differences between individuals' typing styles will be explored.

8.2.1 Individual users – classification

Timing benchmarks

The timing information for each subject was calculated by taking the number of the words in all the typed paragraphs and dividing it by the sum of times for each paragraph. This data was not cleaned, but the breaks between paragraphs did not influence the calculated WPM (words per minute) rate.

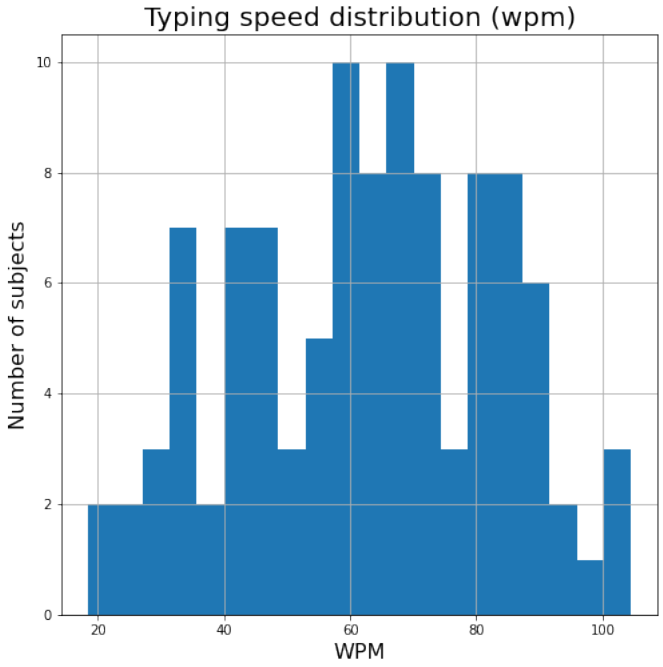


Figure 8.4: The histogram of WPM typing speed of all subjects. The distribution clearly resembles normal distribution, as is expected. The values range from 18.5 to 104.5 WPM with a mean of 62.9 and a standard deviation of 20.36

Using Pearson correlation, it does not seem that the typing speed is highly correlated with age or gender (correlation coefficient of 0.08 and 0.03 respectively), but the dataset suggests that the correlation is relatively high with handedness - a coefficient of nearly 0.40 suggests that the left-handed people tend

to type faster on average. While this does not prove any underlying relations per se, it suggests that there might be a causal link between these two features.

One of the possible explanations of this phenomenon relies on the assumption that one types faster with their dominant hand. If a person uses both hands to type on a QWERTY keyboard (used by the vast majority of subjects), then they would use their left hand to type the 3 most common characters in the English corpus (`E, T, A` - 29.81% of the entire corpus [2]). Using the 10 finger method, the left side of the QWERTY keyboard contains characters that make up 58.68% of the English corpus. It stands to reason that left-handed people will have faster keystrokes nearly 59% of the time.

The timing information suggests that typing speed could be used as metadata for supporting verifiers – while it is prone to manipulation and the method of calculating it is subject to interpretation, it seems that it does not vary much for a given subject.

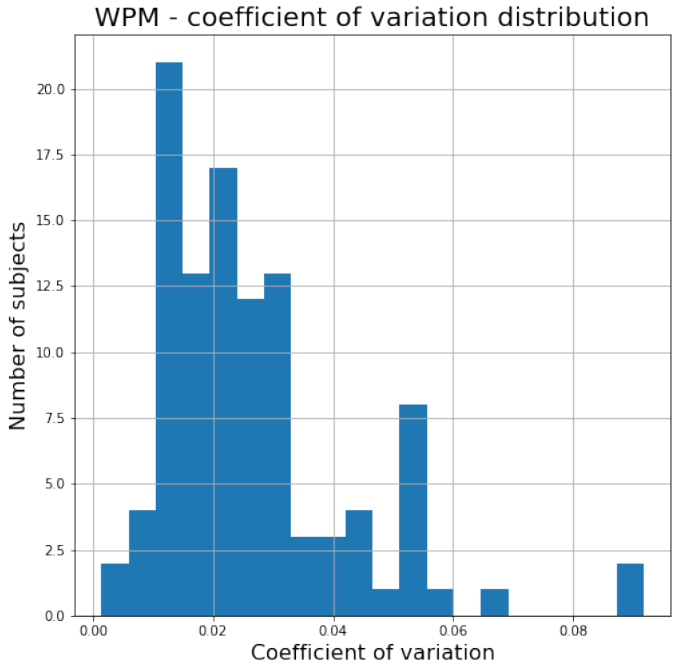


Figure 8.5: The histogram of coefficients of variation for an individual's WPM. The histogram does not include one clear outlier, which had a value of 1.19

Excluding one clear outlier (value of 1.2), the mean value of a coefficient of variation was 0.057, meaning that a typical subject had a coefficient of under 6%, which supports the theoretical viability of using WPM typing speed to support KDA models.

8.2.2 Demographics - dominant hand

As shown in Top features, there is a clear relation between the ability of a KNN classifier to infer handedness and the number of "top" features fed to the model.

Additional testing concluded that the optimal number for the feature count is indeed 10, as shown in figure 8.6. These features are `"e|a", "n|t", "w|e", "v|e", "e|e", "c|o", "e|t", "m|e", "o|m", "u|r"`.

While it is clear that 10 is the optimal number for this dataset and these features, it is still unknown whether the coefficient of variation is correlated with the good results or it is a coincidence and the tenth feature is just very discriminative.

One way to check this is to run the classifier for every single feature and check the individual results. They are shown in figure 8.7.

The next logical step is to take the "best" features for this classifier and see how the F1-score changes with the number of features. This can be seen in figure 8.8.

While the number of used features was again not directly correlated with the F1 score, one of the best

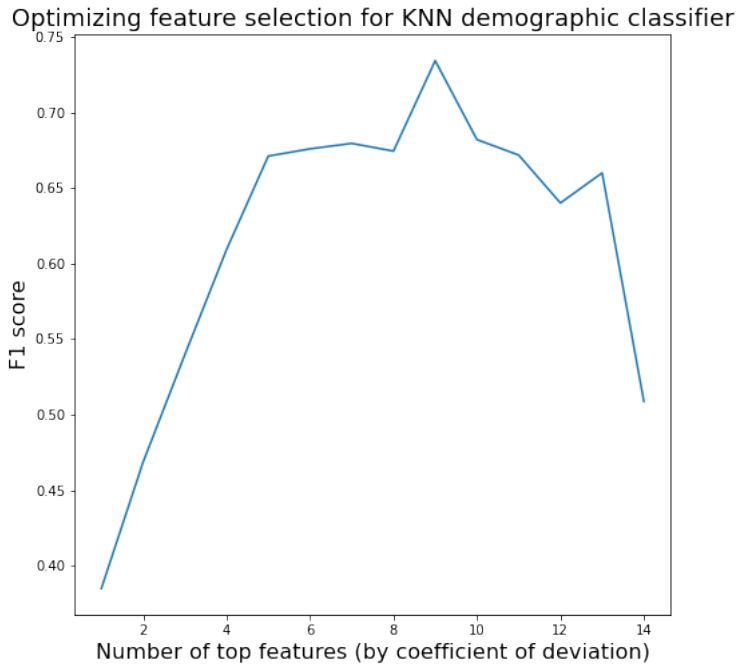


Figure 8.6: The number of top features used versus the F1 score of the model trained and tested using only these features. Top means the features with the highest coefficient of deviation for their respective latencies.

results was achieved when 5 top features (`["k", "l|e", "p", "n|g", "v|e"]` were fed to the model, yielding a F1 score of 0.8581. The best result overall was for 22 top features, with a slightly higher F1 score of 0.8725. Both results have been confirmed with 10-fold cross validation.

It means that, among other features, these five n-grams are so different between left- and right-handers that they themselves act as a good telltale sign of one's dominant hand.

As the last step of this analysis, let's compare the distribution for two features - the most discriminative `k`, and the least discriminative 'a|s'. They are shown in figure 8.9.

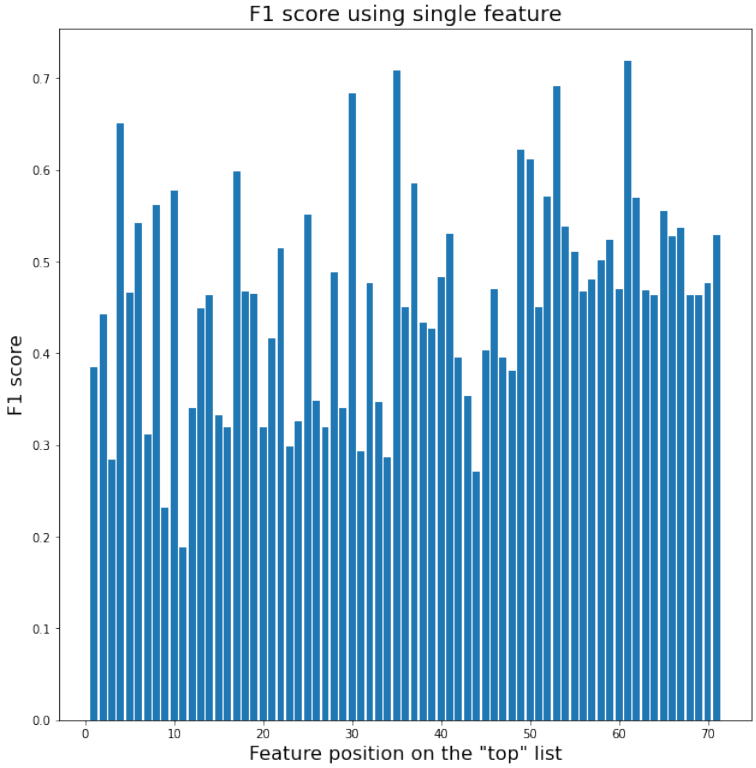


Figure 8.7: The F1 score achieved when training and testing a KNN model using only one feature. The horizontal axis reflects the position of the feature on the top feature list, where top means the highest coefficient of variation.

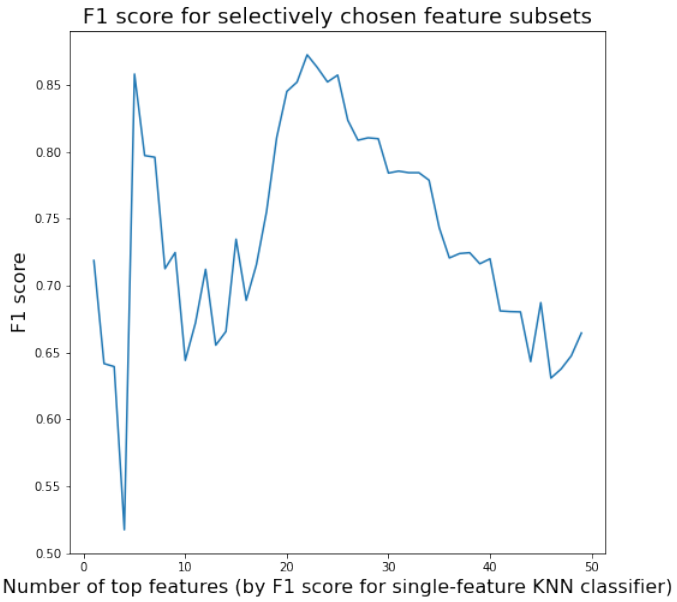


Figure 8.8: The F1 score for a model trained and tested using the first "top" x features, using the values from the previous figure. Top means the feature with the highest value in the previous figure, top 2 means two features with the highest values and so on.

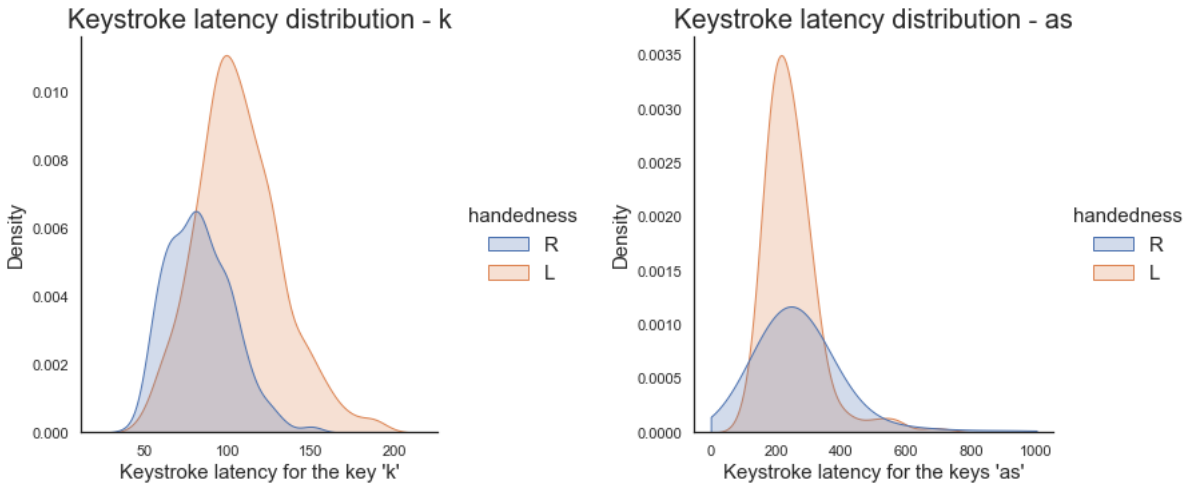


Figure 8.9: The keystroke density for two n-grams. The first one is the key `k`, which was shown to be most discriminative in terms of dominant hand. The second one is `as`, which was deemed one of the least discriminative features. It can clearly be seen that in the first case, the mean keystroke latency is significantly different for the two classes - while there is an overlap between the samples from left- and right- handed subjects, the difference is sufficient to tell the two distributions apart. In the second case, there is very significant overlap, and the means for both distributions are very similar. The greater density for left-handed people is not a mistake, as they constituted the majority of the subjects in this study.

8.2.3 Other factors

There were several other factors that were not explicitly used for classification, but could be used to improve the authentication process. A good example would be typing errors (measured by e.g., Levenshtein distance), which varied greatly across users - from 0 or very close to 0, to over 20% of the length of the typed text. Similarly, the number of used backspaces (for removing errors) also could be fed to a meta-model.

For more advanced analysis, the type of typographical errors could be taken into account - some errors stem from incorrect order of keystrokes, others from double clicking, holding, or missing a key. Yet others from hitting a neighboring key along with the target key. Such an analysis, however, requires vast resources and dedicated techniques for finding such patterns in the keystroke streams.

8.3 Time considerations

Total duration of an authentication attempt can be derived from timing data for each step. Due to random fluctuations, the results may vary, however it can be assumed that the approximations can be trusted if they are a result of the processing of hundreds of thousands of keystrokes in the test scenarios.

The initial data import for the entire dataset (430 thousand keystrokes)[2] lasted around 355s, making it about 826ms per 1000 keystrokes. It needs to be remembered, however, that the initial import and processing time greatly exceed the actual time needed to upload a single sample, because the time required for the import does not scale linearly with the size of the dataset. For comparison, the test dataset, which was smaller by a factor of 3 (about 117 thousand keystrokes), took 43 ms per 1000 keystrokes. It can be inferred that in real-life applications, importing data in batches of 1000 keystrokes at a time will be even faster.

The average tokenization speed was in the order of 30ms per 1000 keystrokes. This value was constant regardless of the size of the dataset.

Data processing time scaled almost linearly with respect to the dataset size. It also depended on the target model - whether the data was to be presented in an aggregated format or not. Similarly, some

[2]That number includes the subjects, the data of which was not later used. Further processing only dealt with 413 thousand keystrokes.

methods of cleansing shortened the time taken. For the full dataset, the processing with the optimal cleansing took 6 and 116ms per 1000 keystrokes for the aggregations and datapoints respectively. For the smaller dataset, these values were similar, at 6 and 89ms respectively.

Additional data processing, such as replacing missing values took about 0.013ms per sample for the most time-consuming data processing task done on aggregations. For the datapoints, that number was about 0.047ms.

Authentication time varied depending on the model. The best metric here would be time per single authentication. Therefore, the timing for each model is:

- Majority vote: 0.61ms per authentication[3],
- Sieve: 0.68ms per authentication[4],
- KNN (aggregations): 0.28ms per authentication[5],
- KNN (datapoints): 0.23 to 0.75ms per authentication[6],
- SVM: 0.47 to 0.70ms per authentication[7].

Summarizing, there are several time-taking steps that need to be performed, namely the import (measured at 43ms per 1000 keystrokes[8]), tokenization at about 30ms per 1000 keystrokes, the data processing (6ms for aggregations and about 100ms for datapoints-based models), the additional data processing (sub-milisecond time in both cases) and the authentication time (again, sub-millisecond time in all cases).

This leads to a conclusion that, after submitting a 1000-keystroke sample, it is expected for an authentication scenario to last at most between about 80ms with an accuracy of about 80% (F1 score about 0.80 for the sieve) and about 200ms with a higher accuracy (F1 score of 0.866 with an EER of 12.4% for KNN). This makes both approaches viable for continuous authentication scenarios.

8.4 Results – summary

Briefly summarizing, here are the top results achieved for each tested model

	Authentication	Demographic classification (age)	Demographic classification (gender)	Demographic classification (handedness)
Majority Vote	0.3247	N/A	N/A	N/A
Sieve	0.8043	N/A	N/A	N/A
KNN	0.8649	**0.3955**	**0.7330**	**0.8581***
SVM	**0.8683**	0.2523	0.7143	0.7451

Table 8.6: The top F1 scores for each model in each category, where applicable. The best result in a given category is in bold. The results obtained outside of testing in section Optimizing parameters are marked with an asterisk. When it comes to KNN, the best result for the demographic classifier for handedness was actually 0.7341 according to the tests performed in Optimizing parameters, however in Distinguishing characteristics the model in question is further refined, achieving better results.

The SVM model with the linear kernel has been proven the best model for authentication given this particular dataset, along with the proposed data cleansing and processing methods, as well as the hyper-parameter space that was tested in the optimization process. The KNN model works better when used for demographic classification tasks, but its authentication model is nearly as good as SVM.

[3] Averaged for 11236 samples over 6.8 seconds
[4] 11236 samples over 7.6 seconds
[5] 106 samples over 30ms
[6] Averaged for 2132 samples, typically 500ms for Euclidean distance and 1600ms for Manhattan distance
[7] 2132 samples over an average of 1000ms for linear and poly kernel. About 1500ms for RBF kernel
[8] The time taken in both recorded cases was heavily skewed in favor of longer times, because the entire data set was being processed at the same time. In a real-life application, processing a single sample of 1000 keystrokes will be done in negligible time.

Adjusting the number of subjects

There is also an important comment to be made here. In additional testing it was found that making the dataset smaller (by taking a random subset of the subjects) actually improved the results. It is possible that a large amount of datapoints increases the rate of false positives, because similar subjects cannot be discerned by the model. Taking a random subset of 30% subjects raised the KNN F1 score to 0.9411, and linear SVM F1 score to 0.9134 when combined with feature set optimization. For 25% these results were 0.9477, and 0.9283 respectively. The results have been confirmed with cross-validation. These subsets contained 27 and 32 subjects respectively, which is comparable to many works in the field, for example [17] with 33 subjects or [34] with 36 subjects. In many other works the number of subjects is somewhere between 30 and 60, sometimes even lower. It is important to consider this factor when comparing studies.

Comparison with the literature

As for the literature, here is a brief summary of how this thesis compares to other research in the field:

		Reported accuracy						
Source	Model	FAR (%)	FRR (%)	Trials	Subjects	Test type	Text	Time (ms)
[17]	Sieve	0.25	16.36	975	33	Online	Fixed	N/A
[22]	Sieve	0.47	94.87	N/A	40	Offline	Fixed	N/A
This thesis	Sieve	1.37	31.10	5929	106	Offline	Fixed	0.86
[24]	KNN	7.9% EER		N/A	63	Offline	Free	N/A
This thesis	KNN	12.4% EER		2132	106	Offline	Fixed	0.75
[34]	SVM	0	6.28	150	36	Offline	Fixed	N/A
This thesis	SVM	13.2% EER		2132	106	Offline	Fixed	0.70

Table 8.7: The results for various models from this thesis compared with the literature in a table. In some cases the number of trials or the time per authentication were unknown and have been left blank. All of the studies presented in this table used only physical keyboards, and the research associated with virtual keyboards has not been included. It can be seen that the sieve and KNN from this thesis have comparable results to the literature, whereas SVM is falling behind. Two factors can be responsible for that though. Firstly, in [34] the reported results are for the "best feature subset" and it is not clearly explained how this subset was selected. Regular results from this paper are more comparable to this thesis, with an 17.55%FRR@0FAR. It has to be remembered that the dataset size can also negatively affect the results, as discussed earlier. In this thesis, the number of subjects is significantly higher than in the works it is compared to.

The current State-of-the-Art is difficult to measure due to vast differences in dataset sizes, used techniques, the way the data is collected, and so on. A sizeable portion of contemporary research revolves around neural networks, as mentioned in [25, 35]. Even looking at only the statistical methods it is difficult to find the best results, as they are often not comparable. It has been shown in this thesis how many factors during data processing have influence on the results. The same can be said about data gathering. Generally, it can be assumed that for statistical and KNN models, the best reliably documented results are in the viccinity of 5% EER, for example reported in [36].

Summary

The main goal of this thesis was creating keystroke dynamics authentication models, using a large fixed-text dataset created specifically for this purpose. The most important parts were data processing to eliminate problems arising from typographical errors, as well as optimizing the feature set and the models to achieve results comparable with the literature. Of special interest was the program execution time, as the keystroke dynamics are frequently mentioned in the context of continuous authentication, which requires the data processing and authentication to be performed seamlessly – both in terms of resource usage and time.

The results reported in the literature have been matched in the case of KNN and SVM models. In the case of the statistical model based on aggregations, the results were poorer than cited in the literature, however this may be due to underlying methodological differences.

It has been shown that, in a reasonable scenario, a high accuracy authentication can be performed on a sample of 1000 keystrokes in under 1s.

Conclusions

Using the far less time-consuming models based on aggregating keystrokes (using mean and standard deviation) seems to nearly match the optimized KNN/SVM models. Further research is needed to see if this is generally true for most applications and whether the aggregation-based models can be further refined.

The KNN and SVM classifiers are certainly very good at performing authentication provided that the dataset and the model parameters have been optimized for that purpose. With their low training and prediction times, they are a good replacement for neural networks, which may be more accurate, but are too slow for any time-sensitive application.

It has also been shown that some models can infer a person's dominant hand or gender with an accuracy far exceeding random chance. Further research is required to improve the methodology and results of such classifiers. The same has not been shown conclusively for a person's age, but it does not disprove the possibility of doing so.

Additionally, a seemingly counter-intuitive trend has been presented – the size of the dataset can negatively impact the accuracy of some models.

Limitations

There is a number of factors that were not taken into consideration in this research but have non-negligible impact on the analysis. Firstly, the volunteers spoke different languages. There were at least 3 languages which were native to the subjects - English, Polish, and German. Since the subjects were not asked about their native language or their proficiency with English, it is unclear whether there were more of them. This is a significant factor that was not accounted for - while there is no specific study proving the influence of language proficiency on typing patterns, this is a commonly accepted notion in the research field[1].

Another important factor is the keyboard layout. Some subjects reported using the DVORAK, AZERTY or QWERTZ layouts as opposed to QWERTY. While using a different layout, or a different keyboard type (mechanical/membrane keyboard) does not impact the patterns on an individual level - they are still unique to said individual - it may affect pattern analysis in a given population. To give an example - A person trained in 10 finger typing on a DVORAK keyboard will use the same finger for keys `P, U, K`, whereas a person using QWERTY keyboard will use a separate finger for each of them. This is bound to result in significant differences in Flight Times for N-grams comprised of these characters (or

[1] The basic reasoning revolves around the fact that low proficiency (or text coherence) prevents remembering longer fragments, causing frequent involuntary breaks for consulting the original text.

other combinations of characters that have the same underlying differences between different keyboard layouts). During pattern analysis, these differences may lead to increased error rates.

Despite using fixed-text, some parts of the analysis were imperfect due to insufficient data. Namely, analyzing patterns with KNN and SVM models was imperfect because of the fact that the datasets needed to have no empty values for the models to use them. This meant filling the empty values, as described in Optimizing parameters. Such an action comes at a cost, because some prominent features in the dataset become obscured. Therefore, in further research it is important to consider this factor to enable more thorough analysis.

Further research

In further research, it is important to account for subjects' expertise with keyboards and their language proficiency. For simplicity, it is also worth it to ask about the subjects' keyboard layouts. It will make it easier to analyze related trends or correct incorrectly recorded datapoints.

It might also be beneficial to feed other keystrokes (such as shift, comma, spacebar, etc.) to the models and evaluate their performance for a larger set of features.

It is also important to perform deeper analysis on the models described in this thesis. While a general approach can yield good results, the intricate analysis presented in some of the literature (albeit on relatively small feature sets) makes it possible to greatly increase a model's performance. A hybrid approach is required, whereby a meta-analysis is performed on large datasets to see and utilize discriminative patterns in the data.

Bibliography

[1] Apache http server project by the Apache Software Foundation. https://httpd.apache.org/.

[2] English letter frequency counts: Mayzner revisited by peter norvig. http://norvig.com/mayzner.html.

[3] Express homepage. https://expressjs.com/.

[4] John vincent monaco website. https://vmonaco.com/datasets. Accessed: 2021-05-08.

[5] JQuery homepage. https://jquery.com/.

[6] MySQL developer portal. https://dev.mysql.com/.

[7] Node.js homepage. https://nodejs.org/.

[8] PM2 project homepage. https://pm2.keymetrics.io/.

[9] Project gutenberg website. https://www.gutenberg.org.

[10] Vue.js homepage. https://vuejs.org/.

[11] Wikipedia entry for unix time. https://en.wikipedia.org/wiki/Unix_time.

[12] Directive (EU) 2015/2366 of the European Parliament and of the Council of 25 November 2015 on payment services in the internal market, amending Directives 2002/65/EC, 2009/110/EC and 2013/36/EU and Regulation (EU) No 1093/2010, and repealing Directive 2007/64/EC (Text with EEA relevance). https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32015L2366, 2015.

[13] R. Giot, M. El-Abed, and C. Rosenberger. Greyc keystroke: A benchmark for keystroke dynamics biometric systems. pages 1 – 6, 10 2009.

[14] R. Giot and A. Rocha. Siamese networks for static keystroke dynamics authentication. In *2019 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6, 2019.

[15] J. Huang, D. Hou, and S. Schuckers. A practical evaluation of free-text keystroke dynamics. pages 1–8, 02 2017.

[16] H. Jagadeesan and M. S. Hsiao. A novel approach to design of user re-authentication systems. In *2009 IEEE 3rd International Conference on Biometrics: Theory, Applications, and Systems*, pages 1–6, 2009.

[17] R. Joyce and G. Gupta. Identity authentication based on keystroke latencies. *Commun. ACM*, 33(2):168–176, Feb. 1990.

[18] K. Killourhy and R. Maxion. Comparing anomaly-detection algorithms for keystroke dynamics. pages 125 – 134, 08 2009.

[19] J. Kim and P. Kang. Freely typed keystroke dynamics-based user authentication for mobile devices based on heterogeneous features. *Pattern Recognition*, 108:107556, 2020.

[20] I. Lamiche, G. Bin, Y. Jing, Z. Yu, and A. Hadid. A continuous smartphone authentication method based on gait patterns and keystroke dynamics. *Journal of Ambient Intelligence and Humanized Computing*, 10, 11 2019.

[21] H. Lee, J. Y. Hwang, S. Lee, D. I. Kim, S.-H. Lee, J. Lee, and J. S. Shin. A parameterized model to select discriminating features on keystroke dynamics authentication on smartphones. *Pervasive and Mobile Computing*, 54:45–57, 2019.

[22] S. Modi and S. J. Elliott. Keystroke dynamics verification using a spontaneously generated password. In *Proceedings 40th Annual 2006 International Carnahan Conference on Security Technology*, pages 116–121, 2006.

[23] J. V. Monaco, N. Bakelman, S.-H. Cha, and C. C. Tappert. Developing a keystroke biometric system for continual authentication of computer users. In *2012 European Intelligence and Security Informatics Conference*, pages 210–216, 2012.

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

[25] S. F. N. Sadikan, A. A. Ramli, and M. F. M. Fudzee. A survey paper on keystroke dynamics authentication for current applications. *AIP Conference Proceedings*, 2173(1):020010, 2019.

[26] A. Salem and M. S. Obaidat. A novel security scheme for behavioral authentication systems based on keystroke dynamics. *Security and Privacy*, 2(2):e64, 2019.

[27] T. Sim and R. Janakiraman. Are digraphs good for free-text keystroke dynamics? pages 1 – 6, 07 2007.

[28] C. Tappert, M. Villani, and S.-H. Cha. Keystroke biometric identification and authentication on long-text input. 01 2009.

[29] P. S. Teh, A. Teoh, and S. Yue. A survey of keystroke dynamics biometrics. *The Scientific World Journal*, 2013:408280, 11 2013.

[30] S. Tenreiro de Magalhaes, K. Revett, and H. Santos. Password secured sites - stepping forward with keystroke dynamics. pages 6 pp.–, 09 2005.

[31] C.-J. Tsai and P.-H. Huang. Keyword-based approach for recognizing fraudulent messages by keystroke dynamics. *Pattern Recognition*, 98:107067, 2020.

[32] I. Tsimperidis, A. Arampatzis, and A. Karakos. Keystroke dynamics features for gender recognition. *Digital Investigation*, 24:4–10, 2018.

[33] T. Wesolowski, P. Porwik, and R. Doroz. Keystroke dynamics database. 2020.

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

[35] Y. Zhong and Y. Deng. *A Survey on Keystroke Dynamics Biometrics: Approaches, Advances, and Evaluations*, pages 1–22. 01 2015.

[36] Y. Zhong, Y. Deng, and A. K. Jain. Keystroke dynamics for user authentication. In *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 117–123, 2012.

Summary in Polish

Cel

Celem pracy było utworzenie modeli uwierzytelniających osoby w oparciu o sposób pisania na klawiaturze (ang. Keystroke Dynamics) ze szczególnym naciskiem na oczyszczanie danych zebranych poza warunkami laboratoryjnymi oraz zastosowaniem w systemach uwierzytelniania ciągłego (ang. Continuous Authentication).

Każdy system uwierzytelniania wymaga maksymalnej możliwej dokładności, chociaż w zależności od potrzeb ta dokładność może być różnie mierzona - w niektórych przypadkach istotna może się okazać minimalizacja błędów typu pierwszego (wyników fałszywie dodatnich), a w innych ważniejsza może być minimalizaca błędów typu drugiego (wyników fałszywie ujemnych). Gdy kontekst nie pozwala określić, która optymalizacja jest pożądana, z reguły szuka się parametrów, dla których poziom obu błędów jest możliwie niski.

Dodatkowym celem projektu było sprawdzenie możliwości inferencji cech demograficznych takich jak dominująca ręka, grupa wiekowa oraz płeć na podstawie sposobu pisania na klawiaturze. Przez sprawdzenie rozumiane jest potwierdzenie wyników z literatury oraz sprawdzenie, czy modele uwierzytelniające oraz klasyfikatory demograficzne mają takie same wymagania i optymalne parametry, czy też klasyfikacja demograficzna wymaga odmiennego podejścia. W literaturze zostały opisane przypadki inferencji płci oraz grupy wiekowej, natomiast kwestia dominującej ręki jest jedynie przyjmowana jako fakt, bez odpowiedniego poparcia badaniami.

Zbiór danych

Zbiór danych użyty w tej pracy został pozyskany przy pomocy anonimowych uczestników. Do zebrania danych utworzona została dedykowana strona internetowa z formularzem, w którym uczestnicy badania przepisywali akapity tekstu. Tekst był napisany w języku angielskim i miał łączną długość 828 słów.

Po uporządkowaniu zebranych danych, zbiór danych zawierał informacje dotyczące 413749 wciśnięć klawiszy od 103 osób. Spośród nich, 77 osób podało również swoje dane demograficzne takie jak dominująca ręka, grupa wiekowa oraz płeć.

Wykorzystane modele

Wykorzystane modele były podzielone według rodzaju danych, które przyjmowały. Jeden rodzaj modeli opierał się na agregacjach, tj. dane każdego uczesnika były podzielone na zbiór testowy i treningowy. Każdy z tych zbiorów został zagregowany, czyli dla każdego n-gramu (ciągu wciśnięć n klawiszy) liczona była średnia oraz odchylenie standardowe.

Modele opierające się na agregacji to: statystyczne głosowanie, statystyczne sito, oraz 1 najbliższy sąsiad (przypadek szczególny k najbliższych sąsiadów)

Inna grupa modeli opierała się na wielu próbkach na pojedynczego uczestnika - wciśnięcia były zapamiętane oraz zamienione na wektory cech. Brakujące dane zostały uzupełnione z wykorzystaniem odpowiedniej techniki. Takie wektory cech były wykorzystywane w modelach k najbliższych sąsiadów oraz maszyny wektorów nośnych (ang. Support Vector Machine). Te modele miały również wariant klasyfikatora demograficznego, który dodatkowo wymagał, by próbki uczestników były rozdzielone między zbiorem treningowym i testowym tak, aby dane każdego uczestnika znajdowały się w wyłącznie jednym z nich, żeby mieć pewność że model uczy się inferencji, a nie uwierzytelniania.

Testy

Przeprowadzono testy dotyczące wielu czynników związanych ze skutecznością modeli. Poza czasem trwania poszczególnych procesów badano również wpływ przetwarzania danych na wydajność modeli.

Sprawdzono kilka sposobów usuwania anomalii ze zbioru danych. Dodatkowo, sprawdzono również techniki uzupełniania brakujących danych w wektorach cech.

W przypadku uwierzytelniania oraz klasyfikacji demograficznej znaleziono optymalne parametry dla wszystkich modeli. Sprawdzone zostało również w jaki sposób ograniczenie zbioru cech lub ograniczenie ilości danych wpływa na wydajność modeli.

Podsumowanie

Maksymalna uzyskana dokładność modelu uwierzytelniającego na pełnym zbiorze była równa 86.6%. Wynik ten został potwierdzony 10-krotną walidacją krzyżową (ang. 10-fold cross validation). Przy całkowitym czasie nieprzekraczającym sekundy na pojedynczą próbkę składającą się z 1000 wciśnięć, jest to wynik pozwalający na rozważanie zastosowania sprawdzanych modeli w systemach uwierzytelniania ciągłego. Głębsza analiza może umożliwić poprawienie dokładności, natomiast należy również pamiętać o tym, że w praktycznych zastosowaniach często bardziej istotne mogą być poziomy błędu typu pierwszego bądź drugiego, z myślą o których powinno się optymalizować modele.

Warto również nadmienić, że sprawdzono także wydajność modeli na pomniejszonym zbiorze danych. Przy losowym wyborze podzbioru uczestników, najlepsze odnotowane wyniki przekraczają 94% po wzięciu ok. $\frac{1}{3}$ lub mniej uczestników, co zostało potwierdzone walidacją krzyżową.

W przypadku cech demograficznych, udało się osiągnąć dokładność ok. 71.5% przy inferencji płci oraz 74.5% przy inferencji dominującej ręki. W przypadku dominującej ręki ograniczenie zbioru cech w oparciu o charakterystyczne dla obu grup wzorce pozwoliło na podniesienie maksymalnej dokładności do 87.25%. Jest to dokładność wyraźnie niższa niż w przypadku uwierzytelniania, jednakże umożliwia ona korzystanie z modeli demograficznych w celu wsparcia działania modeli uwierzytelniających.

Rewritten text

The full text was split into 11 separate paragraphs. The creation of the text is outlined in the chapter Data Acquisition. In total there are 828 words, 3581 characters without spaces, and 4398 characters including spaces.

Each paragraph is a separate text block from the form that was used to gather data.

A large cask of wine had been dropped and broken in the street. The accident had happened when it was being removed from a cart. The cask had tumbled down the street, hit a stone just outside the door of the wine shop, and shattered into many pieces. All the people around had stopped whatever they were doing to run to the spot and drink the wine. Some people kneeled, for the lack of wine glasses used their hands, and sipped, before the wine had all run out between their fingers. Others dipped in the puddles makeshift mugs, and even cloth, which was promptly squeezed dry. A group of people used the stones laying around to stem the wine as it ran, some of them running around to cut off the new streams of wine that were being created. There was no drainage to carry off the wine, and yet it all disappeared shortly after the cask had shattered. After the wine enthusiasts left the street, no trace of the commotion remained.

The appearance of our visitor was a surprise to me since I had expected a typical country practitioner. He was a very tall, thin man, with a long nose like a beak, placed between his grey eyes, set closely together and sparkling brightly from behind a pair of glasses.

They walked along the vines which ran on the building's wall. The sun shined in the sky above, and the newly hatched birds were spreading their wings for the first time in the branches of the nearby trees.

It is a grey house on a narrow street. There is a sundial in its courtyard, by which the inhabitants can tell the hours of the day and when to ring the bell. When the bell rings, we all arise from our beds. The sky is green and cold in our windows to the east. The shadow on the sundial marks off a half-hour while we dress and eat our breakfast in the dining hall, where there are five long tables with twenty clay plates and twenty clay mugs on each table. Then we go to work. In five hours, when the sun is high, we return home and we eat our midday meal, for which half an hour is allowed. Then we go to work again. In five hours, the shadows are blue on the streets, and the sky is blue. We come back home to have our dinner, which lasts one hour. Then the bell rings and some free time is allowed. After five more hours the bell rings and our day concludes.

The voices and footsteps of the many servants who had come with the carts resounded as they shouted in the yard and in the house. The count had been out since morning hours. The countess had a headache caused by all the noise and was lying down in the new sitting room with a vinegar compress on her head.

One after another, the people started coming from nearby streets. The crowd grew and grew, beyond what anyone could count, until it seemed that the whole nation was there. They all stood there, waiting for what seemed like hours. Finally, the time has come. They entwined their arms and started chanting in elation.

Around the evening I judged that we were only several miles from the destination. I wanted to push on, but the manager told me that, as the sun was already pretty low, it would be best to wait until the next morning. He also pointed out that it makes sense to approach in daylight, not during late hours. While I was annoyed at the delay, I agreed that it would be the best course of action. We had no other options, so we ate the rations in silence, watching the trees around us.

I left the rainy streets and entered the building. It was an old winery, full of people avoiding the weather outside, drinking strange potions. There were still several hours until the meeting, so I decided that it was best to wait for the rain to subside.

There were almost no people in the village. What barely passed for a street in the city was the main artery here. Nevertheless, I enjoyed the change. Every day, I would go to the main square to just walk around and chat with people for a few hours.

The waves were crashing around the ship, the wind pounding against the hull, and thunder ringing in my ears. There were people running around the deck, trying to counteract the weather's effect on the ship's course. The wind calmed down soon though and then stopped altogether. After the sky had cleared, I could see treetops on the horizon.

I went over to his place a little after five, and wandered around among swirls and eddies of people I didn't know - though here and there was a face I had noticed on the commuting train.

Data Samples

In this appendix, data samples are shown for the purpose of explaining the structure of the gathered data and showcasing the problems encountered during analysis.

The first example: regular raw data without undesirable patterns:

SID	text	key	start	end
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	0	ShiftRight	1616529040385	1616529040529
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	0	KeyA	1616529040468	1616529040554
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	0	Space	1616529040571	1616529040710
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	0	KeyL	1616529040696	1616529040802
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	0	KeyA	1616529040818	1616529040946
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	0	KeyR	1616529040927	1616529041037
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	0	KeyG	1616529041141	1616529041221
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	0	KeyE	1616529041321	1616529041430
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	0	Space	1616529041401	1616529041496
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	0	KeyC	1616529041556	1616529041661
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	0	KeyA	1616529041651	1616529041816
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	0	KeyS	1616529041783	1616529041898
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	0	KeyK	1616529041886	1616529041966

Table C.1: Raw data for keystrokes when typing `A large cask`

Another example: keystrokes with an abnormal delay

SID	text	key	start	end	2-key FT
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	1	KeyC	1616529235380	1616529235490	N/A
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	1	KeyL	1616529235442	1616529235514	134
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	1	KeyO	1616529235588	1616529235665	223
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	1	KeyS	1616529235639	1616529235741	153
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	1	KeyE	1616529236658	1616529236759	**1120**
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	1	KeyL	1616529236753	1616529236843	185
eae749ae-b7e7-4cf9-978e-f9e0cf855e6c	1	KeyY	1616529236969	1616529237031	278

Table C.2: Typing the text `Closely`. For easier analysis a helper column is added. It shows the Flight Time between a given keystroke and its predecessor. Press-to-Release is used to determine the Flight Time.

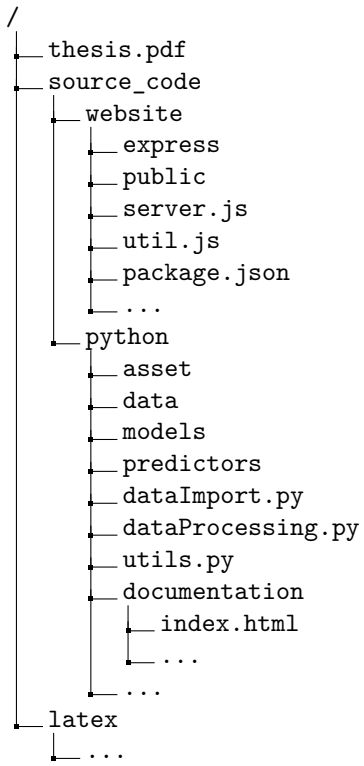
For this particular subject, a typical 2-key FT is in the range $[100, 300]$ milliseconds, meaning that the abnormally high FT between the key `S` and `E` is not part of a typical typing pattern - rather, it was caused by some sort of break in the flow during typing. This kind of errors is relatively easy to detect, especially when they are of this magnitude, but may be difficult to mitigate. Some ideas for dealing with these problems are proposed in Data cleansing.

It also needs to be stressed that this is a frequent occurrence with keys like `Space`, `Shift`, or `Backspace` due to subjects taking breaks between words or being distracted by noticing an error in the text they had typed. This makes the aforementioned keys less useful in terms of analyzing keystroke latencies. As it can be seen from the example, however, the delay can occur between any two keys, and it needs to be accounted for during analysis.

There are typographical errors that are present (and expected) in the dataset, such as double key-presses ("textt" instead of "text"), missing a key ("tet"), or hitting the keys in the wrong order ("tetx"). These problems, however, are out of the scope of this thesis and therefore are not further discussed.

Contents of the DVD

The DVD submitted with the thesis contains, inter alia, the following files:



This document is `thesis.pdf`. The `source_code` directory contains the entire source code used for creating this thesis - it is divided into two main directories, `website`, and `python`.

The former contains the files of the web application used for the thesis. Of particular interest is `server.js`, the main file of the web application. `package.json` contains the information about the modules required to run it. The `express` directory contains `.html` files used in the web application.

The latter contains the `.py` and `.ipynb` files used for building and evaluating the models. All the `test*.ipynb` files in the `python` directory contain various conducted tests. The `dataImport.py` handled the data import from the database. The `dataProcessing.py` handled the data processing prior to building models. `util.py` contains utilities used throughout the project. The `data` directory is empty, as per the agreement with subjects participating in the study. The `models` directory contains definitions of abstractions used in the project. The `predictors` directory contains definitions of predictive models used in the project.

The `documentation` directory contains the documentation for the Python source code. The `index.html` is the index page of the generated documentation.

The `latex` directory contains the LaTeX project files used to create this document.