VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

DYNAMIC TEMPLATE ADJUSTMENT IN CONTINUOUS KEYSTROKE DYNAMICS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. MARTIN KULICH
AUTHOR

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

DYNAMICKÁ ÚPRAVA VZORU PŘI PRŮBĚŽNÉ AUTENTIZACI POMOCÍ DYNAMIKY ÚHOZŮ

DYNAMIC TEMPLATE ADJUSTMENT IN CONTINUOUS KEYSTROKE DYNAMICS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. MARTIN KULICH
AUTHOR

VEDOUCÍ PRÁCE doc. Ing. MARTIN DRAHANSKÝ, Ph.D.
SUPERVISOR

BRNO 2015

Abstrakt

Dynamika úhozů kláves je jednou z behaviorálních biometrických charakteristik, kterou je možné použít pro průběžnou autentizaci uživatelů. Vzhledem k tomu, že styl psaní na klávesnici se v čase mění, je potřeba rovněž upravovat biometrickou šablonu. Tímto problémem se dosud, alespoň pokud je autorovi známo, žádná studie nezabývala. Tato diplomová práce se pokouší tuto mezeru zaplnit. S pomocí dat o časování úhozů od 22 dobrovolníků bylo otestováno několik technik klasifikace, zda je možné je upravit na online klasifikátory, zdokonalující se bez učitele. Výrazné zlepšení v rozpoznání útočníka bylo zaznamenáno u jednotřídového statistického klasifikátoru založeného na normované Euklidovské vzdálenosti, v průměru o 23,7 % proti původní verzi bez adaptace, zlepšení však bylo pozorováno u všech testovacích sad. Změna míry rozpoznání správného uživatele se oproti tomu různila, avšak stále zůstávala na přijatelných hodnotách.

Abstract

Keystroke dynamics is one of behavioural biometric characteristics which can be employed for continuous user authentication. As typing style on a keyboard changes in time, the template adapting is necessary. No study covered this topic yet, as far as the author knows. This master thesis tries to fill this gap. Several classification techniques were exercised with help of keystroke data from 22 volunteers in order to test if they can be improved to unsupervised online classifiers. A significant improvement in impostor recognition was noted at one-class statistical classifier based on normed Euclidean distance. The impostor could make 23.7 % actions less than in offline version on average but the improvement was obseved with all test sets. In contrary, the genuine user recognition varied from user to user but it still kept at acceptable values.

Klíčová slova

průběžná autentizace, online learning, dynamika úhozů, behaviorální biometrika, strojové učení, klasifikace

Keywords

continuous authentication, online learning, keystroke dynamics, behavioural biometrics, machine learning, classification

Citace

Martin Kulich: Dynamic Template Adjustment in Continuous Keystroke Dynamics, diplomová práce, Brno, FIT VUT v Brně, 2015

Dynamic Template Adjustment in Continuous Keystroke Dynamics

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. Ing. Martina Drahanského, Ph.D. a s konzultacemi s panem prof. Patrickem Boursem z Høgskolen i Gjøvik (Gjøvik University College) v Norsku.

. . . .
Martin Kulich
May 26, 2015

Contents

Chapter 1

Introduction

When discussing *biometric characteristics*, many people understand the physiological biometrics only. That includes fingerprints, iris scans, face recognition, etc. However, there are also *behavioural biometric characteristics*, such as a signature, speech, a gait or computer usage manners [16, 13].

For many years biometrics has been used to verify the user's identity in order to allow access to a system or not. This approach is usually referred to as *static authentication*. Once the user is logged in the static authentication system, he is not asked to re-authenticate himself any more. If we consider a computer system, no re-authentication may result in a security breach since the running session might remain unlocked (e.g. the user forgets to lock it when leaving, he loses his mobile device, etc.) and an impostor can use the device with the genuine user's identity.

The *continuous authentication*[1] is a much younger discipline. It studies how to recognise if the user working with a system is still the same person. The continuous authentication system should run in the background, preferably without being notified by the working user. However, if it evaluates the working person changed, it has to lock the screen and force the person to re-authenticate with a kind of static authentication method (e.g. with a password). That implies the biometric system must be designed so that it can operate without user's intervention. If the system is to be deployed easily, it should also require no special hardware. [2, 44]

As the authentication system is running in the background, one should also look after its speed. In order to allow comfortable work, the system should not slow down the computer, thus it needs to be computationally effective.

The stated constraints eliminate most of the physiological characteristics[2] and require usage of common hardware input devices only. In the current research, these devices are mainly represented by a keyboard, a mouse and a web camera. Equipped with that, keystroke dynamics, mouse dynamics and face recognition can be performed.

This work focuses on the keystroke dynamics (KD). A lot of research on its usage for static biometrics has been performed, usually as an additional authentication factor to the traditional method of login with a username and a password. However, only few studies concerned employing KD in continuous biometric systems.

A biometric system captures a biometric sample from an individual and compares it with the reference template created earlier, during the enrolment phase. As keystrokes

[1] Also referred to as *continuous verification* [44] or *dynamic analysis*. [13]

[2] Some trials with physiological characteristics for continuous authentication in computer systems were also performed, e.g. a fingerprint scanner placed on a mouse. [9]

represent a characteristic with low permanence (since the user can get better in typing or can suffer an injury), the template should change in time as well. Periodic re-enrolment in order to update the template is uncomfortable for the user. Dynamic reference adaptation would represent a better way. Unfortunately, there is no research concerning this problem, as far as the author knows. Therefore the research question is set:

> *Is it possible to adjust the template during the authentication phase in a continuous keystroke dynamics system? How does it differ for various classifying methods?*

To answer this question, 22 participants collected their keystroke data to be analysed later. Most participants collected tens of thousands keyboard events. The data were used for simulating data from one user as genuine and the rest as impostor.

Work organising

The rest of work is organised as follows.

Chapter 2 introduces the reader in the authentication using biometric methods. It explains terms biometric system, biometric method and sets conditions for choosing biometric features.

Continuous authentication builds on biometric methods. Chapter 3 describes how to authenticate a user in continuous setting, i.e. in situation he or she is not aware of being authenticated. It discusses how to build dynamic template and how to evaluate such samples.

In Chapter 4, the findings from Chapters 2 and 3 are applied to the keystroke dynamics, one of the behavioural biometric characteristics. Several approaches proposed by various researches are presented, with respect mainly to the choice of features and classification methods.

Chapter 5 is about gathering data from users. It reviews several capturing tools and publicly available databases and selects one, BeLT, for data collecting. BeLT's capabilities and drawbacks are described more thoroughly. Process of seeking participants and explaining their task is also delineated.

Next chapter describes processing the collected data. It includes filtering, transforming to Python data structures, extracting the significant features and packing them so the feature extraction can be skipped in later times.

The possible classifiers themselves are elaborated in Chapter 7. The chapter starts with an introduction to machine learning terminology and `scikit-learn` library. Several classifiers are outlined together with proposals how to make them learn continuously.

The last chapter describes observed results with different classifiers and settings.

Chapter 2

Biometric-based authentication

Nowadays, the privacy becomes a more and more demanded feature in computer systems. In most of the systems, users can work within isolated sessions and they have to authenticate themselves prior to enter the session, e.g. with a combination of a username and a password.

The authentication system can generally use one or more of the following *authentication factors* (or *authenticators*):

1. what the user knows – *knowledge-based*,
2. what the user possesses – *token-based*,
3. who the user is – *ID-based*.

Let us now look at the factors more in deep.

1) Knowledge-based The easiest and still widely used authentication methods are based on some information that the user remembers. The information is most often a password, a passphrase or a PIN code. This group contains also an "obscure" information related to the person that is secret to most people, such as the user's favourite colour or his mother's maiden name [34].

2) Token-based Those methods are based on something the user physically possesses – which is called a *token*. This category includes smart-cards, one-time key generators or metal keys. The major drawback of token-based authenticators is that they can be lost or stolen. If the token is used as the only authenticator, an impostor is able to authenticate himself only with the token. To overcome that, the token-based methods often cooperate with another factor, such as additional PIN code or a password.

3) ID-based The last authentication factor group contains authenticators, which are unique for the user. Usually, *biometrics* is considered as an ID-based authenticator, however, documents unique for the person such as a driving license or a passport fall into this category too. The major advantage of the ID-based methods lies in the security, since it *"cannot rely on secrecy, but instead on the difficulty of replicating it."* [34] However, when they are compromised and replicated, their replacement is difficult.

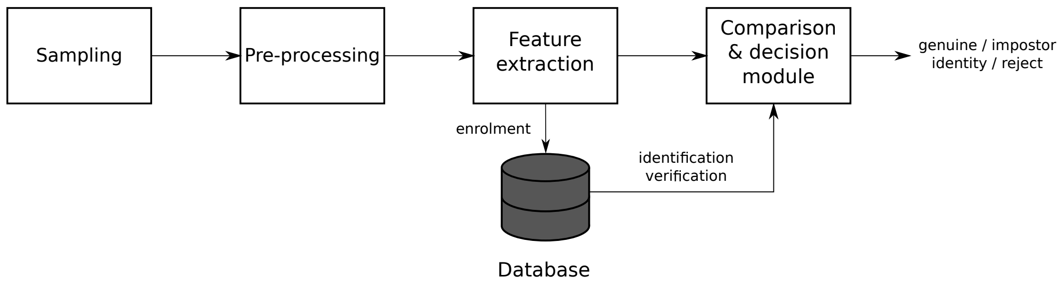


Figure 2.1: Block diagram of a biometric system.

Biometric characteristic *Biometric characteristics* are measured from unique physical, chemical or behavioural human attributes, which are usually called *traits, indicators, identifiers* or *modalities.* The biometric-based authentication has many advantages over the previous two categories. It is something natural and therefore the user needs not look after it. A biometric characteristic is much more difficult to fake and it cannot be lost, stolen or shared. As Jain and Ross [17] mention, the biometric-based authentication offers two more advantages over knowledge-based and token-based systems:

- *Negative recognition* can prevent a single person from using more identities. This *identification*[3] capability is useful for welfare benefits or any other systems where nobody should be able to get the benefit twice (even under different names).
- *Non-repudiation* can log the user's activities and prove his responsibility for the performed actions later.

2.1 Biometric system

A *biometric system* consists basically of several building blocks. The number of blocks varies in the literature, but they can be generalised in five blocks. The system design is shown in Figure 2.1. As it is, in fact, a pattern recognition system (although it does not perform *exact* comparison and rather produces a comparison score, as will be shown later), it is predetermined to contain a reference pattern database and a comparing module. To acquire biometric data, the system also consists of a sensor module and a feature extraction module. The latter one also takes care of the sample quality before extracting the features.

2.1.1 Operational modes

Prior to describing the individual building blocks of a biometric system, its operational modes are presented. First, users must enrol in the system before they can use it. After that, the system can operate *identification* or *verification* mode. [5] Those modes differ from each other not only in the number of comparisons against the reference database, but also in the suitability for continuous authentication.

[3] The term *identification* is explained in Section 2.1.1.

Enrolment A user has to be enrolled in the authentication system before he can use it. The condition holds for a biometric-based system as well. During the enrolment phase[4], the system extracts features from the acquired samples for each user and creates their reference records (in the later text also referred as the *user's template*). Those are then stored in the database, alongside some personal information about the user.

In order to create a representative template, more input samples may be required. [8] The data capturing may be also supervised by a human, who verifies the user's identity and guaranties genuineness of the template. The template is validated against different dataset of the same user before it is released for authentication. The purpose of this additional step is to ensure small *intra*-class distance or to set personal threshold η.

Identification In the identification mode, the system performs one-to-many comparison between the acquired sample and all the records in the reference database in order to find the most similar records to the sample. This mode is used only in static authentication systems, mainly for physiological biometric features. As such, identification can play a big role in forensics. Searching the whole database brings also the capability of *negative recognition* (as discussed in the introduction of this chapter), which helps to prevent *double dipping*[5].

Verification In the verification mode, the system only checks whether the user is the one he claims to be. That means, the sample is compared only with one user-specific database entry and the procedure therefore operates much faster. According to whether the verification is performed statically (while proving the claimed identity) or dynamically (by monitoring whether the user is still the same person), we distinguish *static* and *continuous verification.*

Since it is impossible to perform a continuous identification, the *continuous biometrics* is usually understood as a synonym of continuous verification [44]. For that reason, *verification is the operational mode this thesis focuses on.* All the samples are compared to a single template, which determines that each user's template can use different weights for particular traits.

2.1.2 System operation

In the next few paragraphs, the five basic building blocks of a biometric system are described. The diagram of their operation is shown in Figure 2.1. It is also explained how their operation depends on the current operation mode.

Sensor module The sensor module works as a mediator between the biometric system and the user. The module is usually realised as a special piece of hardware. However, sometimes common hardware can be used, for example for behavioural biometrics. Speed of the sensor module is essential for the overall system speed.

Pre-processing Before extracting features, the sample is verified in order to be suitable for further processing. Quality of the data may be improved, e.g. noise can be removed. However, sometimes the sample is so poor that the user must provide the data again.

[4] Sometimes also referred to as *learning* or *training* phase, especially in connection with machine learning-based systems.

[5] Encyclopedia of Biometrics [23] characterises the *double dipping* as *"the unethical act of seeking compensation, benefits, or privileges from one or more sources, given only a single legitimate entitlement."*

Feature extraction In this module, the biometric data is processed and discriminatory information is extracted. The extracted set of features, referred to as the *user's template*, should evince small intra-class and large inter-class distance. Another reason for simplifying is to reduce dimensionality. The proper choice of features to extract influences the performance of matching module in a large manner. Therefore the features should be selected with respect to the matching algorithm.

Comparison & decision modules The comparing module compares incoming set of features (extracted from the acquired sample) with the template stored in the database. Generally, the comparison is made not only in verification (one-to-one comparison) and identification (one-to-many comparison) modes, but also during the enrolment phase, in order to ensure that the user is not enrolled in the system yet. In fact, the comparing module works in the identification mode during the enrolment. As a biometric feature is not completely stable in time, the matching algorithm does not perform the exact comparison. Instead, the comparing module generates a *comparison score* and lets the decision module to evaluate it. In the verification mode, the modules validate the claimed identity. In the identification mode, a list of candidate identities is returned or the sample is rejected when no match is found.

Database The system database acts as a storage for users' templates along with some biographic information such as name, address, username or PIN. The templates are extracted and saved during the enrolment phase. During the recognition phase, both in identification and verification mode, the templates are passed to the comparing module to be compared with the current sample.

2.2 Biometric methods

Various biometric *modes*[6] are commonly used for user authentication. In this section, the criteria for proper choosing the method are discussed. As this work focuses on the continuous authentication on computers, some methods which are usable for that purpose will be briefly introduced as well.

2.2.1 Behavioural methods

Although majority of researchers in the area of biometrics are interested in physiological modes, such as fingerprints, iris scans, voice recognition, etc., the behavioural modes deserve at least the same attention. The behavioural biometric characteristics evince higher variance, but they can be favourably used in a smaller circle.

As behaviour is rather a long-term process, behavioural biometrics (BB) is predetermined to be used in continuous verification. In computer use, keystroke dynamics, mouse dynamics or software interaction (such as e-mail behaviour, GUI interaction or programming style) can be counted in behavioural characteristics. Besides that, many motor skills (gait, lip movement, signature recognition) can be classified as behavioural characteristics as well.

Yampolskiy and Govindaraju [49] published an extensive survey on behavioural biometrics, concerning many different behavioural characteristics (even very rare) and compared

[6] According to [15], the *mode* is defined as a *"combination of a biometric characteristic type, a sensor type and a processing method."*

them. They pointed out that almost every aspect of human behaviour can be used as a basis for personal profiling and many of them also for biometric verification. The behaviour profiling is already being employed in web-usage analysis, tracking down shopping manners or customising user interface. Several experiments were also performed to show that BB are suitable for continuous verification.

2.2.2 Feature suitability

Not all traits are equally convenient for biometric recognition use. Jain et al. [16] introduced seven aspects which should be considered when choosing the method. Yampolskiy and Govindaraju [49] extended their explanation for behavioural biometrics.

1. **Universality:** Every user of the system should possess the trait. Although the universality of behavioural characteristics is low in the population, it is high enough for the applicable domains.

2. **Uniqueness:** The trait should be unique in the set of users of the system and should evince small intra-class and large inter-class distance. The behavioural features are expected to show larger intra-class distance. They are still unique enough for verification, but it is difficult to identify an individual from his behaviour.

3. **Permanence:** The modality should be sufficiently stable in time. As a user can learn new ways of accomplishing tasks, the permanence of his behavioural characteristics is low. Therefore the template should be periodically updated to overcome this drawback.

4. **Measurability** (also *collectability*): It should be easy to acquire and process the trait. Computer input devices handle this problem easily and without obtruding the user, who sometimes does not even notice capturing the data.

5. **Performance:** This property encapsulates the recognition accuracy which hugely varies according to the operational mode. For verification, the performance is usually high enough even for BB, however, it depends on the observed characteristic in the identification mode.

6. **Acceptability:** The capturing method should be unobtrusive for the user. To provide an example, the footprint-based biometrics is proven to be usable as a biometric mode [33], but we can hardly expect Europeans to take off their shoes to be scanned.[7] Behavioural characteristics, as usually collected without the user cooperation, evince high acceptability, but might be disapproved for ethical or privacy reasons [49].

7. **Circumvention:** The effort for imitating the trait should be very high to prevent obfuscating the system. BB systems are very difficult to circumvent, since it is difficult to get to know someone's behaviour and imitate it.

[7] The authors of the research meant footprint-based biometrics for usage in Japanese environment, where taking off the shoes is a common habit when entering the dwelling.

2.2.3 Using computer input devices

In biometric systems common hardware can be utilised as a sensor module in applications in the personal computer. Although special hardware can be used for the login procedure, such as a fingerprint scanner, only the methods which can perform continuous verification will be discussed in this work. Nowadays, most of the research on this topic revolves around computer-related behaviour – typing keyboard or pointing with mouse. However, continuous verification based on physiological biometrics is also possible, as will be shown later. As far as the author knows, no commercial product for continuous biometric authentication exists.

Let's now look at the three most significant methods.

Keystroke dynamics The keystroke dynamics biometrics is based on the way a user types. A computer keyboard is used as a sensor module. The signal from the keyboard is processed by the operating system (OS) that extracts low-level keyboard events – the *key-down* and the *key-up* events. The feature extraction module transforms a sequence of the low-level events to the sequence of features, which usually include timings of single keystrokes and digraphs[8]. However, some researchers [8, 25, 43] wanted to utilise the advantage of wider context and include also n-graphs ($n >= 3$) or whole words in the template. Such an approach, however, has one large drawback for practical use – a huge amount of data is required for a user to enrol. The whole Chapter 4 is dedicated to the topic of the keystroke dynamics.

Mouse dynamics Using mouse (and pointing devices in general) movements for biometric recognition is a subject of study for much shorter time than the keystroke dynamics. It is obviously caused by expanding usage of pointing devices for controlling the computer. During the continuous authentication, the low-level mouse events data is being collected from the OS. Unlike keystroke dynamics, the low-level mouse events are too detailed to be processed directly. Pre-processing of such events is therefore necessary and aggregation is also used quite often [19]. Feher et al. [11] introduced an extensive study on the topic of the mouse-based user verification. They included many features in the template – not only movements, but also clicks and composed features such as *point-and-click* or *drag-and-drop*.

Face recognition Sim et al. [44] proposed a method of multimodal continuous verification using a web-camera and a fingerprint scanner placed on a mouse. Performing stand-alone face recognition is also possible, but it has to overcome several difficulties including liveness detection or different poses and angles of capturing. However, it is usually considered as one of the less intrusive methods [26].

Summary

Biometric methods are considered ID-based authentication factors. A biometric characteristic is a unique physical, chemical or behavioural human attribute and is the most difficult to imitate from all the authentication factors.

A biometric system is a complex authentication mechanism that collects and processes individual samples, compare them to a user template stored in database and based on the

[8] A *digraph* is an ordered pair of two consequent characters.

comparison predicts either the user identity (identification mode) or whether they are who they claim to be.

Behavioural methods observe specific parts of human behaviour and are suitable for authentication as well. However, they evince much smaller precision than physiological features and thus the authenticators can use combination of more behavioural traits.

Chapter 3

Continuous biometric authentication

Although most researchers focus on static biometric recognition, in some situations it can be advantageous to monitor continuously the user's identity. Considering the user is monitor continuously, the biometric system has to evaluate a large number of samples in a short time. Therefore its template has to be simpler than for static biometrics and it should be possible to update it regularly.

Moreover the continuous biometric system should forgive short-term deviation from the template. Concept of trust handles this problem by maintaining value of *trust level* which expresses rate how much the system believes the user is still the same person. If the trust level drops below pre-set value, the user is locked out.

3.1 Dynamic template

The *dynamic template* is usually understood as a database of biometric features which is being updated regularly during the continuous authentication. In contrast to the static template, the dynamic one must satisfy several requirements:

1. The dynamic template must be **simple.** Comparing a sample with the template is very frequent, since every single sample is examined individually. Several samples can appear every second and they must be processed in a short time with as little resources as possible.

2. It should allow **adding new training samples** during the run time with as little overhead as possible. It implies creating a simple procedure that does not need to process the whole enrolment dataset again. This is especially important for classifiers with long enrolment phase, such as neural networks.

The constraints above require much simpler features than for a static template. For example, for the static keystroke dynamics of a single password, the durations of characters and latencies between them[9] are usually stored. The durations and latencies are directly bound to their positions in the password. For a password of n letters, such a template contains at least $2n-1$ features (n durations and $n-1$ latencies). Additionally, some other features can be included, e.g. overall typing speed.

[9] For the explanation of terms *duration* and *latency* see Section 4.1.1.

In contrast to that, a dynamic template can consist of a table of durations for every key (or a selected subset of the most frequent keys) and a table of latencies between digraphs. Of course, the template structure is adapted to the needs of the particular classifier but generally is composed of simpler features than a static template.

3.1.1 Template adjusting methods

As mentioned before, a typical way to create a biometric template is to capture the user's traits during an enrolment phase. It is possible to divide the template creation into three phases: *capturing, usage* and *adjusting.* The typical operation – I will call it **late adjustment** method – keeps the mentioned order.

However, it is also possible to create the template during the recognition phase, which is more meaningful for behavioural biometrics. Let me call this approach **early adjustment** method. As the name suggests, early adjustment method starts the classification with a template made of very few samples (or even with a completely empty template). During the system operation, the template is being adjusted. From the beginning, the system will show higher error values (refer to Section 3.3). But as the template grows, the system gets more and more adapted to the user's behaviour and is not so impacted by negative effects during the enrolment such as stress or confusion. The early adjustment method also requires less attention from the user.

3.2 Biometric evaluation

In static biometrics, a distance metric and a threshold is usually used to evaluate genuineness of a particular sample. It implies that the genuine user's mistake can cause rejection when logging in. This quite simple approach does not suit continuous authentication, since the user types in a common way and he can make mistakes. Therefore, Bours [6] introduced the *concept of trust* to overcome that problem.

3.2.1 Concept of trust

As mentioned in Section 3.3, the FAR and FRR do not suit measuring quality of a continuous biometric system. They are limited to a certain number of samples, so they can be used at best for evaluating periodic authentication. In a dynamic system, the biometric evaluation should be done with every sample (i.e. in context of this work with every keystroke). Since nobody is perfect and not every sample from a genuine user is mated, the user must not be locked out immediately after one non-mated sample. On the other hand, the system should allow only a limited number of "bad" actions in order to reveal an impostor quickly.

Therefore Bours [6] implemented the *trust level (TL)* – a scale of genuineness of a user. The TL is expressed as the probability that the currently typing user is genuine. When the system has started and the genuine user has just logged in, the TL is set to the value 100 to express 100 % genuineness of the user. Then, while he types, each sample is compared and classified[10] whether it belongs to the genuine user or not.

Once the sample is classified, the trust level value is adjusted. Bours [6] introduced the *penalty & rewards function* for that purpose (see below). If the TL drops below the configured threshold $T_{lockout}$, the currently typing user is treated as an impostor and is

[10] Many approaches for making this decision exist. Their application for keystroke dynamics is discussed in Section 4.2.

locked out. He has then to re-authenticate statically to restore the session (and to reset the TL back to 100 %).

3.2.2 Penalty & rewards function

When a sample is recognised as genuine, the user should be rewarded, i.e. his TL should increase. On the other hand, when it is non-mated, he should be penalised. It depends on the recognition algorithm, however, using statistical methods is a quite popular solution [6, 8]. The distance between the sample and the template is calculated and compared with the distance threshold η as mentioned in Section 2.1.1.

There are more options how to implement the penalty & rewards function. They vary in how much they increase or decrease the trust level with a single sample. Basically, fixed or variable changes can be used. Practically, the function usually contains both of those options.

Using a fixed change, the trust level is adjusted for a certain fixed Δ^+ when a sample is mated and another fixed Δ^- for a non-mated sample. By way of contrast, Δ^+ and Δ^- based on the distance can be used as the variable change. It is also possible to combine those approaches and, e.g., to use a fixed Δ^+ and a variable Δ^-. Making that decision is up to the developer.

Of course, the distance can be calculated only for those samples that have patterns included in the template. If a sample is not found in the template, it may be ignored or the trust level may be decreased by a small constant.

3.3 Error metrics

The performance of a **static** biometric authentication system is usually measured by two error rates.

false acceptance rate (FAR) expresses how many times an impostor would gain access to the system, i.e. how many times the system would classify impostors as genuine users. FAR is defined as [2]:

$$FAR = \frac{\text{\# of false matches}}{\text{Total \# of impostor attempts}} \tag{3.1}$$

false rejection rate (FRR) tells how many times the system would not recognise the genuine user. FRR is defined as [2]:

$$FRR = \frac{\text{\# of false rejections}}{\text{Total \# of genuine user attempts}} \tag{3.2}$$

Example 3.1 *Let's consider a static biometric system for logging in a program. In the table below, T_i denotes the template of user i, S_i^j denotes j^{th} sample of user i and the number in the table represents the distance between the particular sample and the particular template. For clarity, the genuine samples are highlighted.*

	T_1	T_2	T_3
S_1^1	**94**	102	166
S_1^2	**99**	124	122
S_1^3	**131**	105	148
S_2^1	240	**99**	112
S_2^2	133	**61**	147
S_2^3	201	**105**	126
S_3^1	188	121	**124**
S_3^2	135	102	**87**
S_3^3	144	194	**104**

Let's now consider a global threshold η for classifying the sample as genuine or not. If the distance d is lower or equal to η, the sample is mated.

Let's set $\eta = 125$ and calculate corresponding FAR and FRR. We will denote that as FAR_{125} and FRR_{125}. Let's start with simpler FRR_{125}. We need to count how many genuine samples (i.e. the highlighted ones) are above η. According to Equation 3.2 we get:

$$FRR_{125} = \frac{1}{9} \approx 11.1\,\% \tag{3.3}$$

For calculating FAR_{125} we have to count the number of non-highlighted samples with the distance lower or equal η. We get:

$$FAR_{125} = \frac{7}{18} \approx 38.9\,\% \tag{3.4}$$

The FAR in the example above is unacceptably high, it means that approximately two of five impostor's trials to log into the system would be successful. We can change the threshold η to achieve better FAR, but one should note that the FAR and the FRR change simultaneously. When one indicator increases, the second decreases and vice versa.

A frequently used option how to display the quality of a biometric system is to plot the dependency of the FAR and the FRR on the threshold η. Such a plot for Example 3.1 is shown in Figure 3.1. At the point where the FAR and the FRR are equal, there lies a significant point – equal error rate (EER), a frequently used indicator for measuring quality. [2, 6]

Nevertheless, those metrics do not suit the continuous biometrics, since we need to express how fast the system locks out an impostor or a genuine user. Average number of impostor actions (ANIA) and average number of genuine actions (ANGA) metrics fit this [29]. We can express them for each particular user or as an average over the whole dataset.

Average number of impostor actions (ANIA) metric expresses an average number of actions an impostor can perform before the system recognises him and locks him out. Naturally, a general effort is to decrease this number as much as possible, i.e. to reduce the damage the impostor can make.

Average number of genuine actions (ANGA) metric is the opposite of ANIA. It shows how many actions can a genuine user perform on an average before being locked out. As the system should be as unobtrusive as possible, ANGA should limit to infinity, i.e. a genuine user should be never locked out.

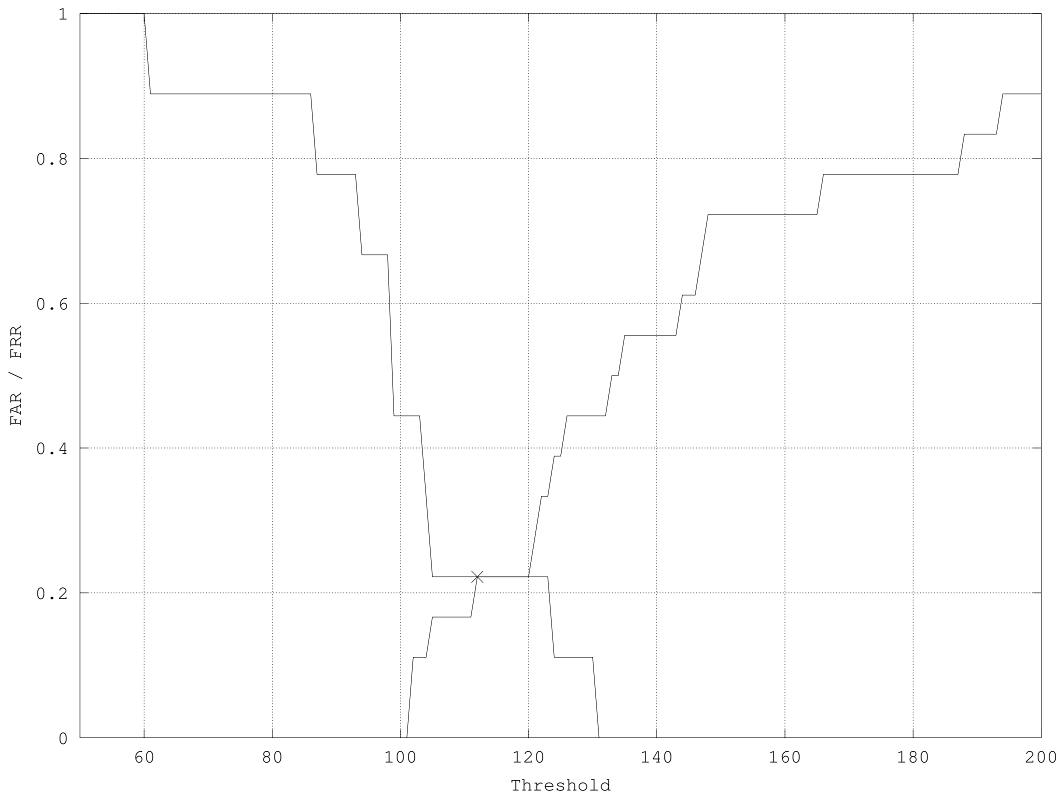


Figure 3.1: Plot of dependency FAR (increasing) and FRR (decreasing) on the set threshold with marked EER point.

Example 3.2 *Let's now consider periodic authentication with the keystroke dynamics, performed every $n = 30$ keystrokes. Periodic authentication allows us to transform the FAR and FRR metrics to ANIA and ANGA. Let the FAR be 3 % and the FRR 0.2 %.*

First, let's transform FAR to ANIA. When an impostor comes to a computer, he can always type 30 keystrokes. After that, he is not recognised and can type further 30 keystrokes with a $FAR = 3\,\%$ probability. ANIA is therefore defined as the infinite series:

$$ANIA = \sum_{i=0}^{\infty} n \cdot FAR^i = \sum_{i=0}^{\infty} 30 \cdot 0.03^i \approx 30.928 \tag{3.5}$$

In the same way, we can transform FRR to ANGA:

$$ANGA = \sum_{i=0}^{\infty} n \cdot (1 - FRR)^i = \sum_{i=0}^{\infty} 30 \cdot 0.998^i \approx 15000 \tag{3.6}$$

Example 3.3 *If we use the continuous verification and Bours's trust model as announced in Section 3.2.1 instead, we have to test the system with the genuine user's data (different from the enrolment dataset) and with some impostor data. Let's use vectors* **G** *and* **I** *for collecting the numbers of actions since last lockout for a genuine user and an impostor (in that order). Both vectors are initially empty. Let's start the simulation of the authentication system and every time the user is locked out, let's append the number of actions since last lockout to the vectors* **G** *(for the genuine user lockouts) and* **I** *(for the impostor lockout). After each lockout the lock is removed and the trust level restored to 100 % value.*

The error metrics would be then calculated in the following way:

$$ANGA = \frac{\sum \mathbf{G}}{|\mathbf{G}|} \tag{3.7}$$

$$ANIA = \frac{\sum \mathbf{I}}{|\mathbf{I}|} \tag{3.8}$$

Summary

The biometric-based authentication is one of the methods from ID-based authentication category. Although physiological biometrics is quite popular, the behavioural biometrics is more suitable for continuous authentication. Static biometric recognition usually evaluates a whole large feature set at once. In opposite, continuous authentication operates upon much smaller sets, since every single sample is evaluated individually.

Due to its lower classification precision for a single sample, the prediction are aggregated using so-called concept of trust. That allows a genuine user not to correspond his template perfectly while it can still recognise an impostor in a short time.

For quality evaluation, several error metrics are used. The differ for static and continuous authentication due to impossibility to compare samples directly. While FAR, FRR and EER are the most common error metrics for static biometrics, ANGA and ANIA describe average success rate in impostor detection.

Chapter 4

Keystroke dynamics

Every human being types the keyboard in a different way. In the history, even the skilled telegraphers were able to recognise who was transmitting on the wire. The way how one types depends on his typing skill, the context (i.e. the surrounding n letters, so called n-graph), the application and the language he types in, handedness, frame of mind, familiarity with used vocabulary and many other circumstances [8, 43, 6, 4, 40]. It can also be temporarily influenced by hand injuries, typing with one hand or by typing on different types of keyboards [40]. The human's keystroke dynamics (KD) is not only unique, but it is also hardly artificially imitable and is therefore robust against automated attacks [45].

If we consider a static biometric recognition by the KD, we may face a problem with high FRR, especially when the conditions during enrolment and recognition are different (e.g. typing a PIN code on different ATM keyboards). High rejection rate makes the system less user-friendly, so the system administrators are still moderate with deploying behavioural biometrics solutions in order to give their customers no reason for discontent. [25]

The static biometric authentication can operate in a *challenge-response* mode, in which the user is attempted to copy the displayed text. It can also be used for strengthening authenticating with a username and a password. This approach is often referred to as *credential hardening* [4].

In 1995, Shepherd [42] showed KD is also capable of user recognition on the free text. It was a very simple algorithm based on keystroke cadence[11] without distinguishing keys.

In the last few years, KD on a free text has become more popular method of continuous biometrics, because it requires neither any special hardware nor user interaction. The typing manner can be captured on a computer keyboard or any input device with physical or emulated keys, which includes mobile phones, PDAs or tablets. Every device which can capture timing information can be used. [23, 6]

Several studies focus on the keystroke dynamics usage for continuous authentication. They vary in data acquisition, in classifying methods and in features contained in the template.

According to [13], the first trial keystrokes dynamics analysis based on a free text was performed in 1997. Before that, studies used to concern the static and predefined text only, although the text was sometimes quite long. Authenticating the user by copying a text gave quite good results for static recognition (e.g. observing whether the password was written

[11] Although term *keystroke cadence* means number of keystrokes per second, Shepherd used the term for average value of duration and latency (see Section 4.1.1).

in a genuine way). However, it gave very poor results for continuous recognition, since it did not reflect a way of typing a common text.

4.1 Features

A base of KD is a *keystroke*. It is delimited by two events recognised by an operating system: a *key-down* and a *key-up* event [4]. We can derive more features from a single keystroke or from a sequence but all of them are based on those two events. The features are more deeply discussed in Section 4.1.

Creating a template during the enrolment phase means to select significant characteristics of the sample set. The majority of research studies use the *duration* of a single keystroke[12] and the *latency* of an *n-graph* (a digraph, a trigraph, etc.). The features are related to a particular key or n-graph. It is good to point here that not all the keys must be stored in a template. For example, Bours [6] selected the most frequent characters and digraphs in English in order to obtain a representative pattern. However, this approach has one large drawback which should be pointed. Restricting the template to a language-characteristic-based subset limits world-wide spreading the algorithm.

According to [2], few older studies tried to include also a key pressure in the template. However, a special keyboard was necessary, which goes against the collactability requirement mentioned in Section 2.2.2. Other advanced features for long-term user authentication include typing speed, frequency of correcting errors (i.e. frequency of using `Backspace` and `Delete`), use of `Shift` key to capitalise letters, using navigation keys (arrows, `PageDown`, `PageUp`, ...), etc. [49, 40].

The user environment is sometimes also taken into consideration [2, 6]. The most observed variables are the keyboard layout and the running application the user is typing in. A separate template can exist for each combination of the environment variables.

4.1.1 Duration and latency

Having a digraph composed from keys K_1 and K_2, two events for each key press can be observed, as shown in Figures 4.1 and 4.2: the *key-down* events when the key was pressed down (in times $t_{down}^{(1)}$, $t_{down}^{(2)}$) and the *key-up* events when it was released ($t_{up}^{(1)}$, $t_{up}^{(2)}$). We can then calculate the **duration** *dur* of key K_i as [6]:

$$dur(K_i) = t_{up}^{(i)} - t_{down}^{(i)} \tag{4.1}$$

The duration can be only a positive number, since the key is always released *after* it is pressed down.

It should be stressed here that the *key-down* event is fired for all the time the key is held pressed. The speed of generating the event is customisable and is usually referred to as *repeat rate*[13]. Anyway, the intermediate key-down events between the initial key-down and finishing key-up should be ignored when extracting the duration. To prevent such a long pressed key from influencing the template, a maximum-time threshold for duration can be set.

The **latency** *lat* **of a digraph**[14] (K_1, K_2) is a bit more confusing property, since studies differ from each other in defining the latency. A frequently used approach is defining latency

[12] Also reffered to as *held time* [43] or *dwell* [40].
[13] In Microsoft Windows, the repeat rate is a parameter of the `WM_KEYDOWN` event messsage. [27]
[14] Also referred to as *inter-key* or *flight time*.

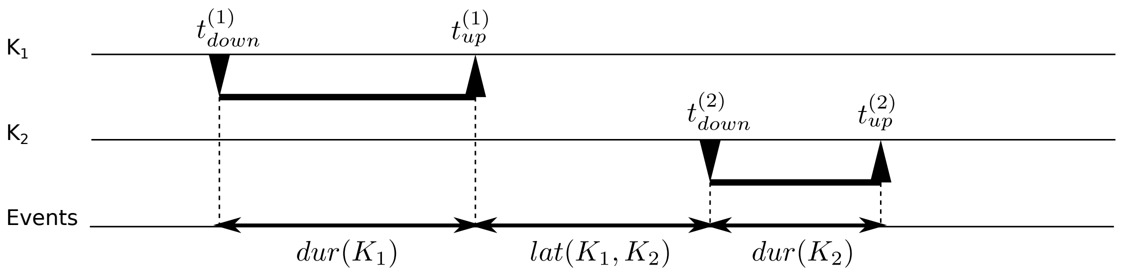


Figure 4.1: Events of a digraph with non-overlapping keys.

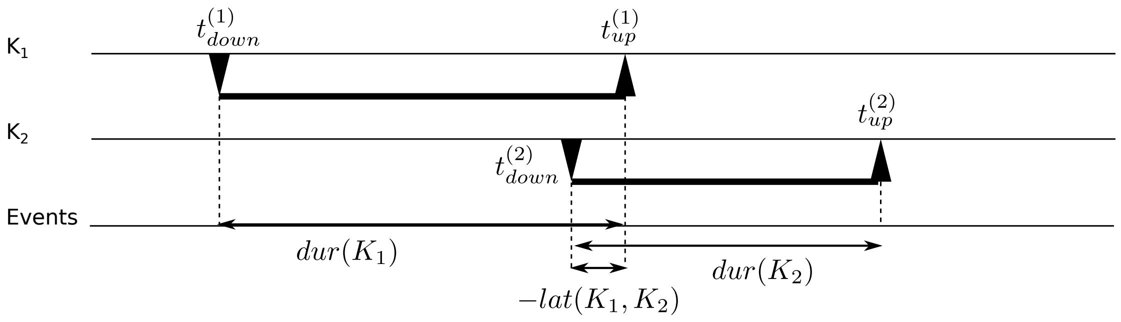


Figure 4.2: Events of a digraph with overlapping keys.

of a digraph as the time delay between releasing the first key and pressing down the next key [6, 8, 25, 43]:

$$lat(K_1, K_2) = t_{down}^{(2)} - t_{up}^{(1)}$$

As defined this way, latency of a digraph can be negative when the second key is pressed before the first is released. This approach is usually called *release-to-press* or *inter-key* time. Stefan and Yao [45] declare in their research that many users tend to write with negative inter-key times.

Other works define the latency by *press-to-press* or *release-to-release* time [2] which always results in a positive value. They are also easily extended to a **latency of a general** n**-graph**. For example, Sim and Janakiraman [43] describe it as *„the time interval between the down keyevents of the first and last keystrokes that make up the n-graph“*.

Since only the n-graphs that two samples have in common are used during the authentication phase, n could be limited to a certain maximum value. Sim and Janakiraman [43] observed that n should be limited up to four to keep the n-graph discriminative.

4.1.2 Advanced features

As mentioned earlier, the template should store only the significant features extracted from the user's KD. The decision what information to store depends mainly on the classification method, this problem is discussed in Section 4.2. However, some general features must be considered with all the classifiers.

Keyboard layout A user is usually more familiar with a certain layout than with another one. This does not apply only to the computer keyboard where different logical layouts can be used, but also for other devices. For example, if a user gets a new mobile phone with a different size or a different key layout, his typing behaviour would probably take a while to adjust and keep stable. The same stands for different types of laptops etc.

Modifier keys Special keys `Shift`, `Ctrl` and `Alt` which modify functions of other keys can be handled either as any other key (and then e.g. `Shift + T` is considered as a digraph) or they can be stored in the template as a flag. Separate templates for any combination of these modifier keys can be then generated. The last approach is to ignore them completely in order to keep the design simple, but then a piece of information about different typing manners with the modifier key pressed are lost.

Dead keys Dead keys do not generate a character but modify a key pressed right after it. Examples of those are `´` (acute) or `ˇ` (caron). Pressing `´` and `a` generates `á`. The keystroke dynamics should always be based on keys, not on characters.

Automatic key repeat When a letter key is held for a long time, the operating system starts to repeat writing down the letter automatically. Although the manner of using this feature may be included in the template, the circumstances when this event occurs happen so rarely that it does not pay off to use it. To eliminate this event, we can set a threshold of maximum time between key-down and key-up event. The repeat rate was more thoroughly discussed in Section 4.1.1.

Frequency of errors The less experienced typist, the more errors occur in the text. Error frequency can be measured as number of `Backspace` or `Delete` keys depressions. [42]

How the user feels Stress and tiredness also influence typing behaviour largely. However, including this kind of information in the authentication system is almost impossible in order to keep it unobtrusive. The template should be compiled from a large enough number of the reference records to be able to handle different users' temper.

Environment Results of the experiments also vary according to the environment where the users attend the experiment. Two environment classes are usually distinguished: *controlled* environment, typically run in a lab on the lab computer with programs specified by the researcher. *Uncontrolled* environment represents the second class. The user works on his own, familiar computer and performs common work as usual. The capturing program runs on the background and captures and stores the keyboard events.

4.2 Existing solutions

Using statistics (i.e. statistical algorithms or statistical classifiers), artificial neural networks (NNs) and machine learning (supervised or unsupervised) are the most popular approaches for the sample classification. [2]

Killourhy and Maxion [20] tested 14 classification techniques proposed by various authors in order to compare their results on a unified dataset. Although their work operates with static recognition, the classification techniques correspond to continuous recognition.

They covered distance-based probabilistic and statistical approaches with miscellaneous distance metrics, as well as supervised and clustering machine learning (ML) methods. Seven of tested classifiers were observed as sufficient, but the authors chose none of them as the best, since the results were compared with several methods. Moreover, those results were obtained using static recognition and their validity for continuous KD should be verified.

4.2.1 Statistical methods

The statistical methods operate with aggregating functions of the feature vectors. The mean and the standard deviation of every key and every digraph are quite frequent, but some studies also operate with minimum and maximum values and other measures. In statistics, a distance metric is calculated to evaluate how much the sample and the template differ from each other. A distance threshold based on the standard deviation is set to classify the sample as genuine or not. However, several systems also use a system-wide threshold value.

Distance metrics

Among other methods, Killourhy and Maxion [20] compare also the performance of Euclidean, Manhattan and Mahalanobis metrics. Generalisation of the first two is called *Minkowski* distance.

Minkowski distance [32] $d(\mathbf{X}, \mathbf{Y})$, where $\mathbf{X}, \mathbf{Y}$ are feature vectors of the same length n, is a parametric metrics with a parameter p. It is defined as:

$$d(\mathbf{X}, \mathbf{Y}) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}} \tag{4.2}$$

If p is substituted with 1, resp. 2, we get well-known Manhattan ($p = 1$, see Equation 4.3) and Euclidean ($p = 2$, see Equation 4.4) distance metrics, respectively.

$$d_M(\mathbf{X}, \mathbf{Y}) = \sum_{i=1}^{n} |x_i - y_i| \tag{4.3}$$

$$d_E(\mathbf{X}, \mathbf{Y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{4.4}$$

Mahalanobis distance [23] can be used in situations when the feature vectors are too complex for Minkowski distance. The distance is calculated between mean feature vector $\mathbf{X}$ and the sample feature vector $\mathbf{Y}$. $\mathbf{S}$ denotes covariance matrix.

$$d_{MH}(\mathbf{X}, \mathbf{Y}) = (\mathbf{X} - \mathbf{Y})^T \mathbf{S}^{-1} (\mathbf{X} - \mathbf{Y}) \tag{4.5}$$

If the covariance matrix is diagonal, the resulting distance metric is called *normalised Euclidean distance:*

$$d_{EN}(\mathbf{X}, \mathbf{Y}) = \sqrt{\sum_{i=1}^{N} \frac{(x_i - y_i)^2}{\sigma_i^2}} \tag{4.6}$$

where σ_i denotes standard deviation of i^{th} feature in the training set. Its huge advantage over the Euclidean distance is its range. The minimum range of Euclidean distance is zero (meaning identity) but the maximum range is unknown. With normalisation, the distance will remain in multiples of σ_i which wouldn't usually exceed 3 for normal distribution.

4.2.2 Machine learning approaches

The user verification is a 2-class classification problem (a genuine user vs. an impostor). However, we have to handle the problem with non-available impostor data. Although the reference data of other users in the system can be employed as the impostor data to train the classifier, they are not always available (e.g. when running the continuous authentication system on a single-user PC). This problem can arise rather in static authentication, since the other users would have to type the same password as the genuine user. On the other hand, continuous KD system can run on a single-user computer.

Marsters [25] employed RapidMiner framework for testing three different classifiers on a continuous KD dataset – *Bayesian Belief Network (BayesNet), K-Star* and *RandomForest* classifiers. He tested them against the key durations set. They hugely vary in training time, but both BayesNet and RandomForest fit under one minute with the error rate of $2.39\,\% \pm 0.88\,\%$ and $2.25\,\% \pm 0.98\,\%$, respectively[15].

Yu and Cho [50] selected a support vector machine (SVM) for its short training time, which is 1,000 times shorter than the time required by a NN. They also introduced a novelty approach for selecting features to be included into the template and to pass to classifiers. The method employs genetic optimisation algorithm. They achieved the average error rate of 0.81 %.

Revett et al. [36] employed a probabilistic neural network (PNN) for authentication. The PNN operates in the supervised mode and both genuine and impostor samples are required to train it. They reported a much faster learning phase compared to a back-propagation based NN. The PNN algorithm achieved approximately 4 % error rate.

As seen from this short overview and the overview in [2], the ML algorithms can achieve similar results like statistical approaches. However, only a few studies worked with the dynamic verification and more research is needed to compare statistical and ML approaches.

4.3 Performance

A performance of the keystroke dynamics in continuous authentication is somewhat variable, depending on many factors. In general, better results were achieved with digraphs, especially if a word [43] or an application [6] context was taken into consideration. However, the penalty & rewards function plays a big role as well. The best results in the studied literature were achieved by Bours [6] with detecting an impostor in 98 keystrokes on average (considering an application context).

Summary

In this chapter, several studies exercising the keystroke dynamics were presented. The features mainly concerning duration and latency of a digraph were discussed. Several classification methods were introduced, including both statistical and machine learning meth-

[15] Marsters, however, does not specify what error metrics he uses. We can only suppose it as EER.

ods. Although machine learning approaches perform quite well for static authentication, no study employed them for the continuous keystroke dynamics yet. The statistical approach is very popular with continuous verification researchers and several distance metrics were shown in the text.

Chapter 5

Collecting user data

It is not necessary to implement the whole authenticating system to show characteristics of different kinds of templates. Instead, I simulated the authenticating system on captured keyboard events captured on volunteers' computers during their common work.

The keyboard events are usually implemented as hardware interrupts and as such are processed by the operating system. The operating system transforms the interrupts to the form of *event messages*. Therefore it is possible for a program to capture and process these messages along with the OS. Several tools for capturing keyboard events already exist in the research world. A short overview of available applications is provided in section 5.2.

Since a large amount of data is essential for proper analysis, many participants should be involved and observed for a long time period. A capturing program should follow similar guidelines as a continuous authentication system, especially be unobtrusive and completely automated. Additionally, it should consider users' privacy and therefore it should not log any sensitive information such as passwords or bank account numbers.

The continuous biometric authentication is still a young discipline and only a few tools are available for that purpose. Many of the tools are moreover intended for using in different research areas such as human-computer interaction (HCI) (e.g. [21]) and they do not meet all the demands placed.

5.1 Existing databases

In the literature, many researchers collect data for their work. However, those datasets vary in quality and are often adapted to needs of the particular research. Moreover, most of the publicly available databases are designed for static authentication research (such as [12]) due to the risk of present sensitive information in the common work recording.

Montalvão and Freire [30] built a publicly available database of free-text samples from 15 participants. The sample is unfortunately pretty short, it consists of only 10 rows of text, about 110 keystrokes each. In addition, the text was collected during only two sessions, which is a very small number for the purpose of my experiment.

Banerjee and Woodard's survey [2] provide an overview of existing KD databases. Most of them concern static biometrics and, moreover, almost none of them is available now. Only one of the databases is marked as dynamic in the survey – the Montalvão's and Freire's mentioned in the previous paragraph.

To summarise it, I have not found any publicly available database for continuous KD. I understand the worries about privacy of participants of the experiments, however, making such a database public could allow others to compare their results to each other.

5.2 Capturing tools

Many of the available tools are originally not meant to be employed in an authentication research, but rather in HCI area. However, some tools dedicated for biometric research have been developed.

Unfortunately, I have not found any truly cross-platform tool. Most are targeted for Microsoft Windows systems, however few of them cover also Mac OS X or GNU/Linux[16].

RUI (Recording User Input) is a tool introduced by Kukreja et al. [21]. It is intended for research in area of HCI and is able to capture keyboard and mouse events. The binaries are available for Microsoft Windows and Mac OS X operating systems. Unfortunately, as the tool is developed primarily for HCI research, it records only key-press events, not both key-down and key-up events. For that reason, it is useless for behavioural biometrics research, which needs to measure duration and latencies of the keys.

AppMonitor could be a great tool for logging keyboard events if it was extended a little. As its authors mention, only two applications (Microsoft Word and Adobe Reader) are supported and only special key combinations are captured in order to protect user privacy. [1]

Inputlog logs both key-down and key-up events and a researcher can obtain it on request. However, a participant has to start and stop recording manually and has to remember to stop the tool when typing sensitive data like passwords or bank account numbers. [22]

BAKER by Marsters [25] favours user privacy and therefore captures wider context of the typed key, a *trigraph* for capturing durations and a *quadgraph* for storing statistical data about latencies. Therefore both duration and latency are stored in a 3-dimensional matrix, which the author preferred to ordered logs to keep the users' privacy. The statistical data are represented by the count of occurrences, the mean and the variance.

For all the mentioned reasons, BAKER would look as an ideal program for collecting data. However, the websites proposed in the work are not available any more, and the tool neither.

TUBA is not really a tool for the continuous authentication, but rather for periodic authentication triggered by certain combinations of network and typing events. TUBA is mentioned since it employs X window system (X11) and therefore represents the only tool available for Linux of those I have found. Its architecture is composed from a remote authentication server and a client. If a network security breach is detected, the server challenges the user to re-authenticate himself.

TUBA is also interesting for reducing the timing vector dimensions using principle component analysis (PCA), a method from data mining area. [45]

[16] The Stefan and Yao's work [45] covers particularly Linux systems with *X Windows System* (X11 in short). Actually, X11 can run also on other operating system, so it is partially platform-independent.

BeLT (Behaviour Logging Tool) by Stenvi, Øverbø and Johansen [46] is a tool specially developed for capturing user interaction in Microsoft Windows. It captures keyboard, mouse and software interaction events and relevant information about hardware (such as screen resolution etc.).

It observes user interface interaction employing UI Automation framework. [28] Therefore it can recognise types of user input fields and does not record passwords. This is a huge advance over the other tools since it improves users' feeling of security.

The program starts on system startup, runs minimised in the system tray and does not require any additional action from the user. Thus it fulfils the requirement of unobtrusiveness.

I was permitted to use BeLT for collecting the data for this master's thesis.

5.3 Data capturing

This section describes the structure of participant set and how they took part in collecting the data.

5.3.1 Target participants

I had to restrict possible participants to Microsoft Windows users as BeLT is only available for Windows. The participants should actively use the computer in a period of at least two weeks to obtain enough data.

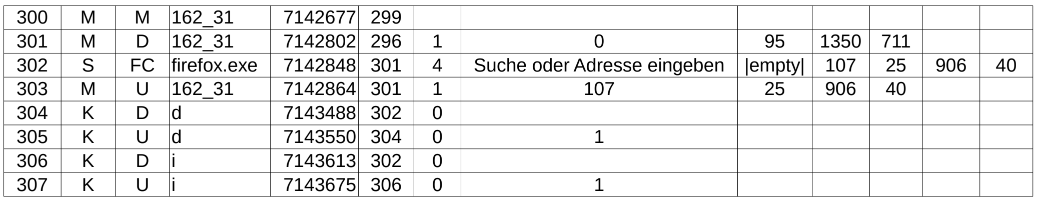
Originally, I intended to ask only people who currently learn to type the keyboard. However, I found only six high school students willing to participate and all of them rejected later for various reasons. Therefore the participant set consists users with different level of typing skill and wide range of how often they use the computer.

In total, almost 50 people promised to participate but only 22 of them eventually delivered the data.

5.3.2 Participants' task

The participants were informed about the purpose of the experiment and how to collect the data to keep high quality of the samples. They were acquainted with BeLT installation, interface, proper settings and operation. Every participant was also informed about the risk of collecting sensitive data and instructed how to prevent it. In order to keep the data free of other people's samples, I also asked the participants to pause BeLT when they lend the computer to someone else, even for a short time.

The task was to do their common computer work with BeLT running in the background for at least two weeks. After that period, they were asked to send me the data for further analysis.



300	M	M	162_31	7142677	299							
301	M	D	162_31	7142802	296	1	0	95	1350	711		
302	S	FC	firefox.exe	7142848	301	4	Suche oder Adresse eingeben	|empty|	107	25	906	40
303	M	U	162_31	7142864	301	1	107	25	906	40		
304	K	D	d	7143488	302	0						
305	K	U	d	7143550	304	0	1					
306	K	D	i	7143613	302	0						
307	K	U	i	7143675	306	0	1					

Figure 5.1: Excerpt of BeLT CSV file (opened in LibreOffice Calc).

5.4 BeLT data format

BeLT exports captured data into CSV[17] files. Each row in the file represents one captured event message. BeLT captures keyboard, mouse and some software events and also logs some hardware information (mainly the screen resolution).

The row format varies[18] depending on the event type. A sample excerpt from BeLT CSV file is shown in Figure 5.1. The first three columns have identical meaning for all event types. The first contains event ID, a number unique within every file. In the second field, there is a basic event type (`K` for keyboard, `M` for mouse and `S` for software). An action is determined by the third value: e.g. for keyboard events, `D` denotes key down event and `U` denotes key up event.

Considering only the keyboard events, next columns provide information about the key, the event timestamp (in miliseconds from starting the computer), an ID of related event (for key up event refers to its respective key down event), flags indicating active system keys (e.g. Shift, Ctrl, Alt etc.) and repeat count (see Section 4.1.1). [46]

BeLT can also store the data in raw format[19]. It is not a binary format as one might suppose, but an ordinary text file. Its rows are more verbose than rows of the CSV format.

However I discovered a bug in recording the time. On some special occasions (e.g. switching the keyboard layout with `Shift + Alt`), a `LCtrl` key up event is generated (with no related key down event) and the time jumps about 50 days forward. Therefore the CSV format is a preferred way for collecting data.

5.5 BeLT drawbacks

It is advisable to be able to compare users from different countries. In real use, it would be necessary to build a solid database for training the classifier. Therefore storing the numerical key code (which is transformed to a key based on set keyboard layout in the OS) would be a better choice than storing the key. Although the keyboard is the same (usually generic 104/105 layout), the keys are not. I encountered this issue when I created templates for several users who had set a Cyrillic layout.

[17] Comma-separated values (CSV) is a simple file format for tabular data. The format was specified in RFC 4180 [41].

[18] The BeLT-exported data does not comply with the CSV specification due to variable column count and data types.

[19] BeLT's raw format uses file extension `.raw`. This can be a bit confusing since `.raw` files usually store photographs in the camera.

The problem also occurred with Czech characters that have to be typed using a *dead key*[20]. In that case BeLT interpreted the combination `Shift + ˇ, n` as `ˇn` instead of `ň`. Particularly this issue discouraged about 10 potential data collectors from collaboration.

When processing the data, storing data in single-byte encoding proved not to be very comfortable to work with. The files are stored in system-wide set encoding instead of UTF-8 which is treated as present-day standard. Since the data collectors were of various nationalities, I had to convert each user's data to UTF-8 separately. Without that, the data would not be comparable.

Summary

In this chapter, the existing tools for capturing keystroke dynamics were presented, as well as some existing databases. Unluckily, none of the tools and the databases has sufficient capabilities for the purpose of continuous dynamics. BAKER software would represent one exception, if it had been still available. Fortunately, I was permitted to utilise BeLT for acquiring the data from the users which made the data collection reachable.

BeLT is a program users install to their computers and keep it running in the background. It collects data about how the user behaves, notably it tracks keyboard and mouse actions. BeLT stores data in CSV files where each line corresponds to one action.

[20] Regarding dead keys, refer to Section 4.1.2.

Chapter 6

Data processing

Before the data can be pushed to classifiers, it has to be prepared to a suitable form. That includes converting from BeLT format to list of features. This way from BeLT to Python is not that straightforward as it might seem. It consists of four steps we will take a look at in the following sections:

Pre-processing normalises file names, encoding and formats (see Section 5.4 for more information on BeLT formats).

Converting CSV to `EventList` transforms each BeLT line to an instance of `Event` class and joins them in a list.

Extracting features step takes `EventList` as an input and creates features consisting of more events.

Packing for later use is a necessary step preventing from extracting the data again every time the data is required.

The first step was performed manually due to need of manual interception for selecting proper encoding. The others were run as a batch using script `extractfeatures.py`.

In the real continuous authenticator, the feature extraction would run online as soon as the data was captured. However, batch processing is much more convenient for the simulation.

6.1 Pre-processing

In Sections 5.4 and 5.5 we discussed how BeLT stores the data. The first precondition for convenient work with `bash`[21] is to remove spaces from file names and convert them to common encoding. That was achieved with `rename` and `iconv` commands.

In order to determine proper initial encoding, python package `chardet` was employed. It provides also Linux command that can be called directly without starting Python shell and evaluates probabilities of various encodings. The proper one should be selected manually with knowledge of the originator operating system.

Moreover, two of the participants did not follow the instructions properly and collected data to the raw BeLT format. I wrote converting script `raw2csv.py` that performs the

[21] Bash is a UNIX shell providing easy scripting language. It allows direct calls of programs and is therefore more convenient for semi-batch processing of files.

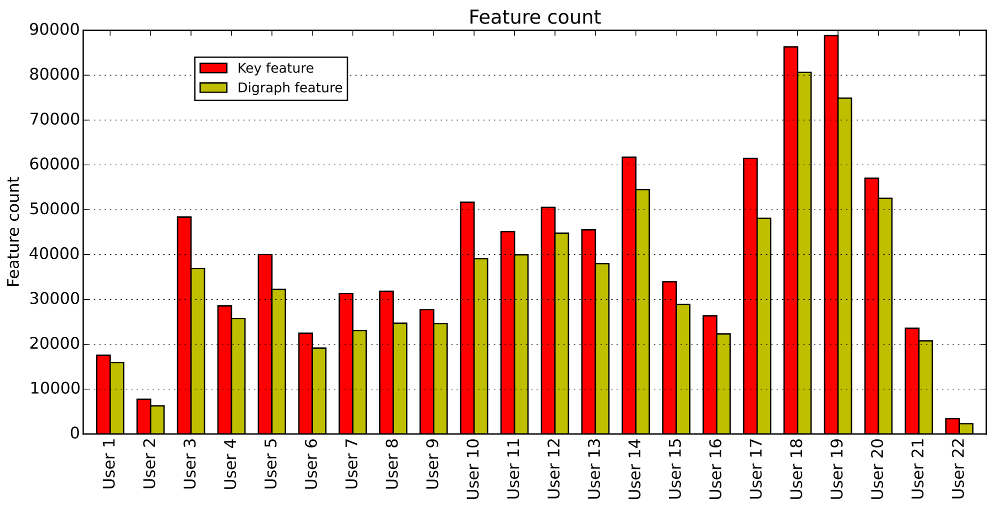


Figure 6.1: Feature count of participants by feature type.

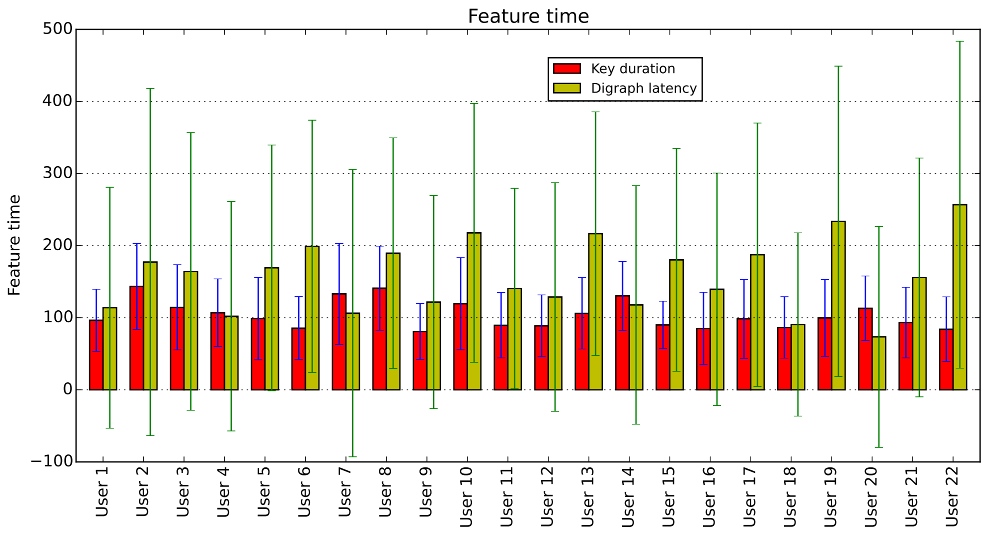


Figure 6.2: Feature time mean and standard deviation of participants.

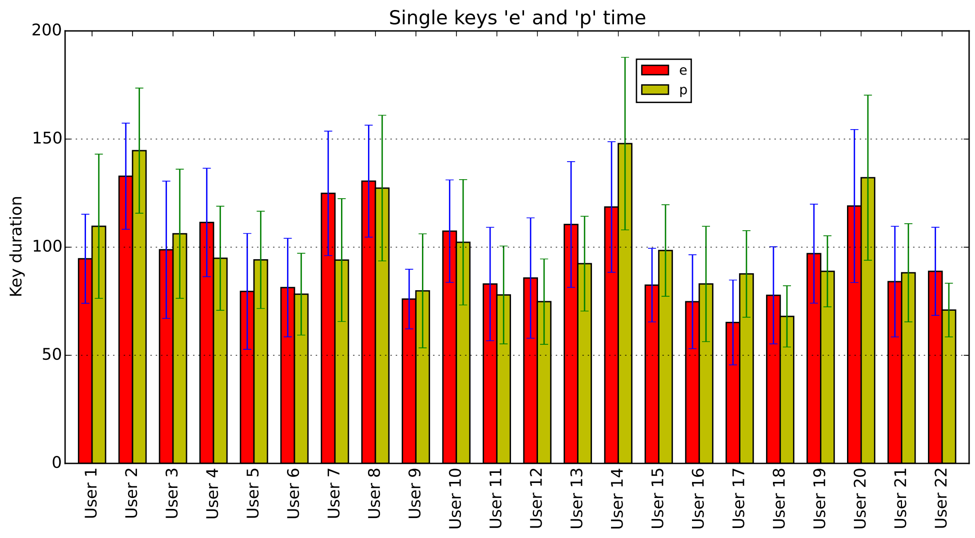


Figure 6.3: Single keys `e` and `p` durations and their respective standard deviations.

conversion. The time bug discussed in Section 5.5 is not bypassed by the script, since only relative times between two consequent events are calculated. Therefore time jump twice a day would not make such a big difference to be worth fixing it.

6.2 Creating event list

The cleansed data are processed by Python `csvreader` and converted to a dictionary[22]. Only keyboard data are currently stored but the script is designed to be easily extended.

6.3 Feature extraction

When the list of events is generated, the features can be extracted. Although it is against common naming conventions in biometrics, here *features* denote type of higher-level events. Two type of features are extracted from keyboard events – a *key feature* and a *digraph feature*. Properties of features (i.e. biometric traits) are called *feature properties*. I collected key, time (duration or latency) and modifier keys of the feature, as defined in Section 4.1. Moreover, I extracted further properties which are calculated also from the samples in the past. Those are:

Context is an estimated position in the text. Its value can gain one of the following values: *first key of the word, in-word key, space* (i.e. space, backspace or delete), *shortcut* (i.e. a key pressed together with `Ctrl`, `Alt` or `Win` key) or *last digraph in the word.* Note there is no such context for last single key feature, since it is not possible to recognise it in the stream incoming to classifier[23].

[22] Python dictionary is essentially an associative array as known from other programming languages.
[23] In the simulated environment the experiment was taken at, it would naturally be possible. But in order to keep the simulated reality in place, the last key feature flag was omitted.

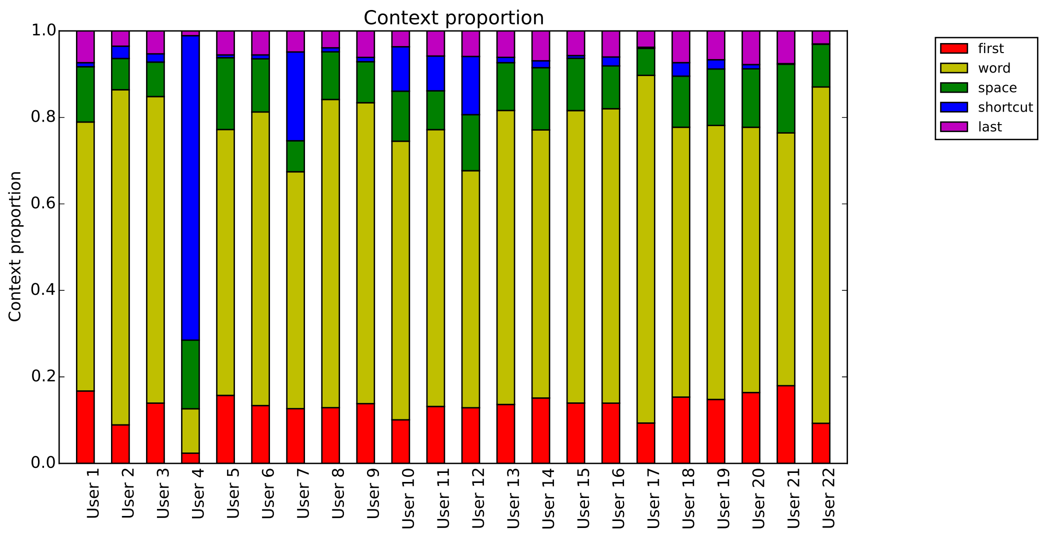


Figure 6.4: Context of each participant.

Typing speed measurement is implemented as two counters: number of observed features in the last 15 and 60 seconds. This property was added as a trial to eliminate impact of typing speed variance.

By experimental testing with Gaussian Naïve Bayes classifier(see Chapter 7), these features improved the results slightly.

6.4 Packing

Because feature extraction takes about 5 minutes for the dataset, the final feature list is packed using the built-in Python package `cPickle`. The package is designated for serialising Python objects (the method is called *pickling*).

Unfortunately, there is a drawback of `cPickle` in storing the object together with its methods. Therefore when you *unpickle* the feature list, you can work with the methods only in the version when it was pickled. For that reason, it is a good idea to inherit the feature list from some built-in type and after unpickling use only the built-in methods.

6.5 Data statistics

This section provides several statistics on the extracted data.

Figure 6.1 shows number of features that were captured by each participant. As it can be observed in the plot, the number of total features varies from $166,942$ (User 18) to $5,725$ (User 22). The average feature count is $72,468$ features.

The collected data contained slightly more key features than digraphs. This small difference points out that most of the keystrokes were captured when typing a word or pressing a keyboard shortcut. Contexts of the features are illustrated in Figure 6.4 and they also support that hypothesis.

Average keystroke durations and digraph latencies are depicted in Figure 6.2. Keystroke durations are slightly shorter than digraph latencies for most of the participants. However, that does not hold for User 11 and User 17; according to Figure 6.1 are both fairly advanced typists. The digraph latency error bars in Figure 6.2 confirm assumption from Section 4.1.1 that latencies can be also negative, especially for advanced typists.

Although key durations might seem not to have required variance, it turns out, when filtered by a particular key, this variance is in place. Figure 6.3 gives such example for keys `e` and `p`. The differences between average durations of `e`'s and `p`'s are relatively small but should be enough for correct classification. Obvious contrast can be observed by Users 7, 13, 17 and 22. One can suppose similar differences are present for other participants with different combination of keys.

Summary

This section provided an overview on what has to be done before one can use the BeLT data. The users' data were processed by Linux and Python tools to eventually receive a Python list of features per user.

Chapter 7

Training the classifier

As the user types, the keyboard events are generated by operating system and processed by a classifier. The incoming event stream can be generalised as a data vector $\mathbf{D} = (d_1, d_2, d_3, \dots)$. The classifier tries to uncover regular patterns in $\mathbf{D}$ and select the best-suiting hypothesis h from the hypothesis space[24] $\mathbf{H} = \{h_1, h_2, \dots, h_N\}$. Based on the classifier, the hypothesis space can be finite or infinite.

As it was explained in Section 2.1.1, the biometric system can operate in *identification* or *verification* mode. In verification mode, the classifier evaluates $\mathbf{D}$ and decides between hypotheses $h_{genuine}$ (denoting the last observed data point belongs to the genuine user) and $h_{impostor}$.

In identification mode, the classifier operates upon the database of N users and estimates likelihood for each hypothesis in $\mathbf{H} = \{h_1, h_2, \dots, h_N, h_{none}\}$.

Each hypothesis $h_i \in \mathbf{H}$ has its *prior probability* $p(h_i)$ that expresses probability of h_i without observing any data. By normalisation, $\sum_i p(h_i) = 1$. Probability of hypothesis h_i after observing data vector $\mathbf{D}$, $p(h_i|\mathbf{D})$, is called *a posteriori probability.*

The data vector $\mathbf{D}$ is usually, especially under supervised setting, split in train data $\mathbf{D}_{train}$, validation data $\mathbf{D}_{val}$ and test data $\mathbf{D}_{test}$. Train data are used for learning the classifier *(fitting)* and correct labels *(target)* are required for fitting. Since many classifiers are set by parameters invariant on train data, validation data is usually used to estimate best setting of these parameters. As you can see, there is need of a large amount of data for setting the classifier properly. Fortunately, *cross-validation* (see Section 7.3) may help to solve the problem with data amount and allows omitting validation data.

The data, originally stored in a feature list (see Chapter 6 for how the raw data is transformed to a feature list), has to be transformed to suit individual classifier's needs. Section 7.1 describes how the data is processed.

The best hypothesis, $\arg\max_{h_i} p(h_i|\mathbf{D})$, is then used for predicting class of each sample. However, as it was described in Section 3.2.1, the decision about a user's genuineness cannot be made from a single sample. A wrapper class `TLClassifier` observes values predicted by nested classifier and adjusts the trust level according to the predictions.

As the behaviour changes in time, it is favourable to maintain the user's template up-to-date. In machine learning, this approach is known as *continuous* or *online* learning. The online learning keeps learning also in the recognition phase. The concept of online learning is discussed in Section 7.4.

[24] In context of this work, hypothesis space represents space of all possible templates.

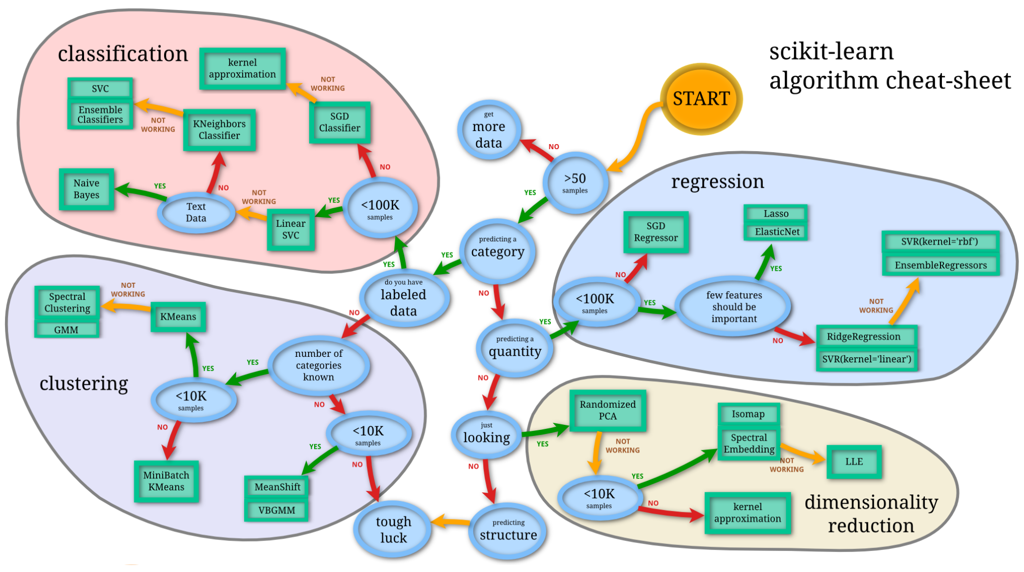


Figure 7.1: Overview of machine learning algorithms `scikit-learn` library provides. [47]

7.1 Introduction to `scikit-learn` library

Machine learning algorithms are nothing new. It would be unwise to code them again when specialised well-tested libraries already exist. One of such libraries is `scikit-learn` [35] which I decided for because it covers pretty wide area of machine learning and data mining including also pre-processing, selecting useful features and model evaluation. This section provides a short overview of `scikit-learn` design and capabilities. Thanks to the library's interfaces it is possible to insert custom code that cooperates with the library functions.

The basic building block of the library is an *estimator*. Those blocks can be *pipelined* in such manner that the each block in the pipeline receives the output of the preceding block to its input. The first block receives the raw data.

Developers can create a custom estimator if non of those included in the library do not satisfy their needs. However, the library itself provides a large set of estimators which are of one of the following types:

Transformer is an estimator used to pre-process the data according to needs of the following blocks in the pipeline.

An example of such transformer is class `DictVectorizer` that takes a list of Python dictionaries as input and transforms it into 2-dimensional array of real numbers.

Most of the transformers need some data to create rules how to transform the data. This operation is called *fitting* and all estimators have to implement method `fit()` in order to be integrated in a pipeline. `DictVectorizer` scans keys in the dictionary and creates a column in the output array for each input numerical feature. This cannot be applied to string features, so those are transformed to M boolean indicators where M denotes number of unique values for the key in the input list. This behaviour allows to specify categorical (even numerical) data simply by transforming them to string before handing them in the `DictVectorizer`.

Once fitting is finished, the following data is only transformed (by method `transform()` each transformer has to implement) without modifying the transformation rules.

Classifier is usually the last step in a pipeline in such setting that the data should be divided in classes based on previously seen examples. Classifier therefore works in supervised mode. An unsupervised analogue is a *cluster* (see below).

As each estimator, a classifier has to provide method `fit()` that is used for training the classifier. Moreover, several classifiers can operate also in an online setting. The classifier can be learnt online if it provides method `partial_fit()`.

Once the classifier is trained, it can predict novel data using methods `predict()` or `predict_proba()`. The former method simply returns the most probable class, the latter estimates likelihood for all classes that the novel sample belongs to it.

Regressor is similar to classifier in the manner it works as the last step of a pipeline. However it differs in prediction target. While classifiers tries to find a model that determines a class for the data, regressors approximates a mathematical function that generates the data.

Cluster is an estimator that does not need to fit data since it is an instance of unsupervised classifier. Its aim is to divide the incoming data into clusters based on their distance. Several distance metric were discussed in Section 4.2.1. Note that distance metric are not solely clustering domain, they can be used in supervised statistical learning as well.

The estimators can be pipelined together, making one compact estimator to work with. The pipeline automatically decides what actions should be taken by each of the estimators. It usually starts with some pre-processing transformers, continues with optional feature selection that reduces complexity of the data and finishes with a classifier or a regressor as the last step.

Overall, `scikit-learn` library is a great helper to work with and makes creating custom estimators much easier.

7.2 Supervised classifiers

Classifiers that need to observe several training examples before predicting classes for unseen samples, are called *supervised*. The supervisor is commonly referred to as a *teacher*.

This section provides overview of three classifiers that were compared in this work. The first two of them, Naïve Bayes a k-Nearest Neighbours are *multiclass* classifiers, i.e. they can generally predict class of all the samples. I used them in *two-class* setting though (genuine user vs. impostor) to be able to compare them with the third classifier, a *one-class* statistical classifier.

7.2.1 Naïve Bayes

One of the simplest supervised is classifiers is the *naïve Bayes* classifier. It assumes *i.i.d.* (independent and identically distributed) data. In practice, this assumption is often violated but even though the naïve Bayes classifier performs surprisingly well. Its performance can be even improved by boosting[25]. [3, 39]

The classifier employs Bayes probabilistic model with independent variables. For a data point $\mathbf{x} = (x_1, x_2, \dots, x_D)$ where x_i denotes one discrete variable, the probability of being classified as class $c \in C$ is:

$$\begin{aligned} p(c|\mathbf{x}) &= \frac{p(c) \cdot p(\mathbf{x}|c)}{p(\mathbf{x})} \\ &= \frac{p(c) \cdot p(\mathbf{x}|c)}{\sum_{c_i \in C} p(c_i) \cdot p(\mathbf{x}|c_i)} \end{aligned} \tag{7.1}$$

Prior probability $p(c)$ can be set explicitly or be proportional to c occurrence in the training set. From the i.i.d. assumption, the $p(\mathbf{x}|c)$ is just a simple product:

$$p(\mathbf{x}|c) = \prod_{i=1}^{D} p(x_i|c) \tag{7.2}$$

Discrete variables For some applications, the likelihood of variable x_i assuming class c can be deducted from a model. But usually the underlying model is unknown and in that case, $p(x_i)$ can approximately expressed as:

$$p(x_i|c) \approx \frac{count(x_i \wedge c)}{count(c)} \tag{7.3}$$

where $count(a)$ denotes number of occurrences of a in the training set [10].

Continuous variables When dealing with continuous data, those are usually assumed to be distributed according to Gaussian distribution [3, 39]:

$$p(x_i|c) = \frac{1}{\sqrt{2\pi\sigma_{i,c}^2}} e^{-\frac{(x_i - \mu_{i,c})^2}{2\sigma_{i,c}^2}} \tag{7.4}$$

where $\mu_{i,c}$ denotes mean and $\sigma_{i,c}^2$ denotes standard deviation of continuous variable x_i in class c.

Classification For two-class classification $C = \{c_0, c_1\}$, the final decision is made based on comparison:

$$p(c_0|\mathbf{x}) > p(c_1|\mathbf{x}) \tag{7.5}$$

$$\frac{p(c_0) \cdot p(\mathbf{x}|c_0)}{p(\mathbf{x})} > \frac{p(c_1) \cdot p(\mathbf{x}|c_1)}{p(\mathbf{x})} \tag{7.6}$$

[25] Boosting is a process of iterative training where hypotheses from all iterations are used for creating the final hypothesis.

Each iteration consists of weighted training and validating with the same training set. In the first iteration, all weights are set to the same value. The classifier is trained and weights adjusted – weights of those samples that were classified correctly are decreased and weights of incorrectly classified samples increased. This process repeats K times and the final hypothesis is a weighted majority combination of hypotheses from all K rounds. [38]

As the denominator has only normalising function, it can be omitted which simplifies the comparison:

$$P(c_0) \cdot p(\mathbf{x}|c_0) > p(c_1) \cdot p(\mathbf{x}|c_1) \tag{7.7}$$

$$p(c_0) \cdot \prod_{i=1}^{D} p(x_i|c_0) > p(c_1) \cdot \prod_{i=1}^{D} p(x_i|c_1) \tag{7.8}$$

If the expression is true, then the classifier evaluates $\mathbf{x}$ as belonging to class c_0, otherwise as belonging to class c_1.

For multi-class classification, $p(c_i|\mathbf{x})$ is calculated for each class c_i and the data point is classified as member of class with the highest probability.

7.2.2 K-nearest neighbours

Another classification method uses distance metric (as mentioned in Section 4.2.1) for determination of nearest samples in the training set[26]. [3]

When a novel sample $\mathbf{x}$ is to be classified, the distance to each sample in the training set is calculated and k nearest samples (those are called *neighbours*) are selected. The class of $\mathbf{x}$ is given by the most numerous class within the k nearest neighbours. In the case of ambiguity, $(k+1)^{\text{st}}$ nearest neighbour is selected and so on.

The template of k-nearest neighbours classifier is very simple, it is a list of training samples. It determines the speed: although the training is very fast (it consists only of loading the training set), classification of a novel sample takes n comparisons where n is the size of the template.

Having the template so simple however simplifies operation in online setting, since extending the template means only appending the novel training sample.

However, one should be aware of growing the template since the larger the template, the longer time is required to classify a sample. This problem can be overcome by replacing a sample in the template with the novel example, instead of appending. If the furthest sample of the template is replaced, it might lead to reducing the perimeter of the cluster and thus to rise of false rejections. A better way is designing the template as a circular buffer, so that always the oldest sample is replaced. This enables adapting the template.

The method of classification also implies a requirement for the template to contain both genuine and impostor samples. However, it is possible to modify the algorithm to operate as *one-class classifier*. Instead of predicting the class of a novel sample, the algorithm calculates distance from the nearest neighbour and checks if it falls into predefined (manually or dynamically based on the training set) threshold. If so, then the novel sample is classified as genuine, otherwise as impostor.

7.2.3 One-class statistical classifier

A simple classifier proposed by Bours [6] can handle only continuous properties. In the original paper, only timing information is relevant and the template is stored separately for the keys. The idea is though extendible for any number of continuous properties.

The key point here is that the template is created solely from the genuine user data. Such approach is called *one-class classification*.

[26] *K-nearest neighbours* are also often used in unsupervised setting, as a clustering algorithm. In that case, the distance is measured from every sample in the set to each other and clusters are estimated according to the distance between samples.

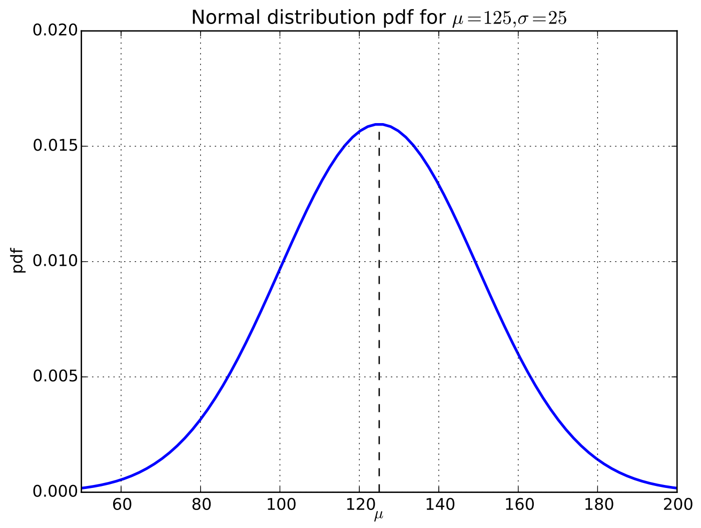


Figure 7.2: Probability density function of normal distribution with $\mu = 125$ and $\sigma = 25$.

The underlying concept is an assumption that data from user u and key ki are generated under normal probabilistic distribution as depicted in Figure 7.2. Maximum of probability density function $pdf(x)$ lies in $x = \mu$.

Fitting For a key k and i^{th} feature of the feature vector created of N samples, the template $T_k = (\mu_k, \sigma_k^2)$. For simplicity, the calculation here is shown only for one feature. In the implemented classifier, this template holds for every feature in the feature vector. The mean μ_k and standard deviation σ_k^2 for key k are calculated as follows:

$$\mu_k = \frac{\sum_{i=1}^{N} x_{k,i}}{N} \qquad \sigma_k^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{k,i} - \mu_k)^2 \tag{7.9}$$

However, this form does not suit the online learning very well, since it has to store all previously seen examples. It is favourable to disassemble the equation to the following form:

$$\mu_k = \frac{1}{N}\sum_{i=1}^{N} x_{k,i} \tag{7.10}$$

$$\sigma_k^2 = \frac{1}{N}(\sum_{i=1}^{N} x_{k,i}^2 - 2\mu_k \sum_{i=1}^{N} x_{k,i}) + \mu_k^2 \tag{7.11}$$

and simplified by using two additional variables:

$$Sum_k(N) = \sum_{i=1}^{N} x_{k,i} \qquad Sum_k^2(N) = \sum_{i=1}^{N} x_{k,i}^2 \tag{7.12}$$

Substituing in (7.10) gives:

$$\mu_k = \frac{Sum_k(N)}{N} \tag{7.13}$$

$$\sigma_k^2 = \frac{Sum_k^2(N) - 2\mu_k Sum_k(N)}{N} + \mu_k^2 \tag{7.14}$$

Now, if we are about to fit $(N+1)^{\text{st}}$ example:

$$Sum_k(N+1) = Sum_k(N) + x_{k,i} \tag{7.15}$$

$$Sum_k^2(N+1) = Sum_k^2(N) + x_{k,i}^2 \tag{7.16}$$

Incrementing in (7.15), (7.16) and recalculating mean (7.13) and standard deviation (7.14) are the only operations to be taken to fit a novel example. This approach saves memory space for storing template.

Classification A prediction is made based on calculating distance from the mean. Even though Bours employs scaled Manhattan distance, i.e. Manhattan distance from (4.3) divided by the respective variance:

$$d_M S(\mathbf{X}, \mathbf{Y}) = \sum_{i=1}^{n} \frac{|x_i - y_i|}{\sigma_i} \tag{7.17}$$

This metric scaling can be applied to any distance metric. One should just be aware of using the right denominator, e.g. Euclidean distance (4.4) should be divided by standard deviation σ_i^2 instead of variance. The advantage of using a scaled metric is that that enables classifying by a fixed threshold.

If the distance of the sample is within some margin around the mean, the sample is treated genuine. The margin threshold beyond which the sample is treated impostor can be set even as a hard value or be calculated dynamically according to the train set.

Based on the prediction, the trust level is adjusted. If no template is found for the particular key or the template consists of very few examples, Bours proposed to decrease the trust level slightly.

7.3 Cross-validation

When training a supervised classifier, one may face two problems that may happen with improperly set learning parameters. The first one is *underfitting* and occurs when the classifier cannot even predict the train set correctly. There trying better configuration can help.

The more common problem is *overfitting*, i.e. fitting to train examples perfectly. Perfect fit means it includes also outliers or devious samples. Such classifier performs very badly with unseen examples. A usual way is to take another part of labelled data, the *validation set*. The classifier is trained on training set and predicting the validation set can detect overfitting.

However, one is often dealing with small training set, why should it be shrunk by taking a part of it as a validation set? Fortunately, *cross-validation* can help to fix this problem. The basic approach is dividing the training set in k parts of the same size. Training then works in rounds. In the first round, the training set is formed with $k-1$ parts and the validation set with the remaining part. In any next round, another part is chosen as the validation set and the training is repeated. Each part is used for validation exactly once.

The final score is calculated as an average score for all the rounds.

This can help with choosing the best parameters for the classifier. `sklearn` library makes automated testing of many different settings possible by providing *grid search* functions. The search runs over a grid of manually set parameters by trying each possible combination. The best score classifier is then selected and can be directly used for prediction. [35]

An example of such grid is:

```
param_grid = [{
    "tl_threshold": [60, 70, 80, 90],
    "tl_learn": [80, 85, 90, 95],
    "online": [True],
    "tl_history_size": [20, 50, 100],
    "distance_threshold": [1.0, 1.5, 2.0]
}, {
    "tl_threshold": [60, 70, 80, 90],
    "online": [False],
    "distance_threshold": [1.0, 1.5, 2.0]
}]
```

The example shows two separate grids. The former one exercises the classifier in an online mode and therefore provides also parameters for online learning (`tl_learn` and `tl_history_size`). The latter one needs not be trained for all values of the online-learning-specific parameters as they have no impact to the performance.

I utilised grid search with cross-validation for every exercised classifier.

7.4 Online learning

A machine learning classifier expects the data to keep independent and identically distributed over time. The behavioural biometrics, however, cannot guarantee that expectation since the behaviour usually changes over time.

The online learning classification aims to overcome this problem. It is capable to process data that change rapidly in a short time or a large dataset that changes gradually. By contrast to the *offline* learning (the kind of ML we have seen so far) which stops learning once it switches from training to recognition phase, the online learning continues learning also in recognition phase. Some online classifiers even start in recognition phase and learn from scratch.

In the same way as with the offline classifiers, one can distinguish training with and without supervision. Both groups are described in the following sections.

However, it is possible to modify supervised offline classifiers to be trained in supervised setting but adjust their template during recognition.

7.4.1 Supervised pure-online classifier

Block diagram of a supervised classifier is depicted in Figure 7.3. For every novel sample x_i, the classifier predicts $\hat{y}_i$ and returns the prediction. After the prediction is made, the

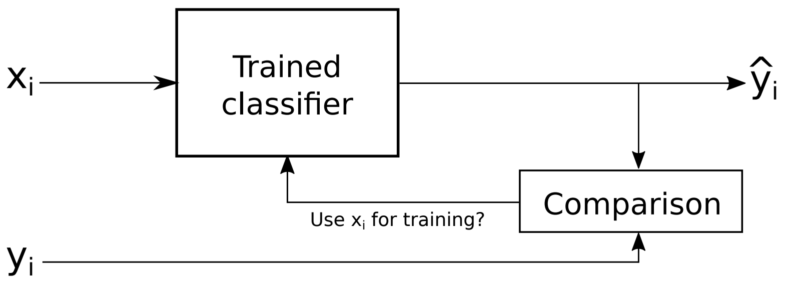


Figure 7.3: Supervised online-learned classifier.

classifier is provided the correct answer y_i and based on comparison between y_i and $\hat{y}_i$ it decides whether to update the hypothesis (template).

An example of such algorithm is **randomised weighted majority algorithm** for boolean classifiers, described by Russell and Norvig [38]. It operates on a set of boolean classifiers. Each classifier has an assigned weight which describes its trustworthiness and which is updated with every sample.

Formally, it is described in Algorithm 7.1. Real number β on the line 9 ranges from 0 to 1 and determines how much the classifier is penalised if it provides an incorrect prediction.

```
 1: procedure RWMA($\mathbf{C} = \{C_1, \ldots, C_K\}$)
 2:     Initialize $\mathbf{W} = \{w_1, \ldots, w_K\}$ all to 1
 3:     for every incoming sample $\mathbf{x}$ do
 4:         $\{\hat{y}_1, \ldots, \hat{y}_K\} = \textsc{Predict}(\mathbf{C}, \mathbf{x})$
 5:         Randomly choose a classifier $k*$, in proportion to its weight: $P(k) = w_k / \sum_{k'} w_{k'}$
 6:         Provide $\hat{y} = \hat{y_{k*}}$
 7:         Receive correct answer $y$
 8:         for each classifier $i$ such that $\hat{y}_k \neq y$ do
 9:             $w_k \leftarrow \beta w_k$
10:         end for
11:     end for
12: end procedure
```

Algorithm 7.1: Randomised weighted majority algorithm. [38]

However, the set of possible classifiers is very large, even for a small number of features. For example, it needs 1024 classifiers for a boolean 10-feature space.

Winnow [24] is another classifier operating on a boolean feature space. On the contrary to the previous algorithm, Winnow manages processing a very large feature space in a short time.

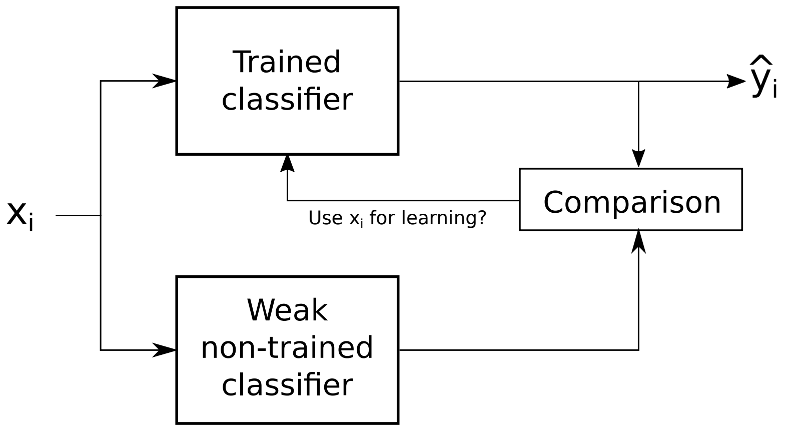


Figure 7.4: Unsupervised online-learned classifier.

7.4.2 Unsupervised pure-online classifier

Littlestone [24] introduced a supervised online-classifier Winnow[27] that is able to efficiently eliminate irrelevant features from large boolean feature space.

The algorithm assumes there exists a hyperplane that linearly separates the sample space $X = \{0,1\}^n$. One can find the hyperplane by finding a correct prediction function in a monotone disjunctive form[28]:

$$f(x_1, \dots, x_n) = x_{i_1} \vee \dots \vee x_{i_k} \tag{7.18}$$

The hyperplane is then given by $x_{i_1} + \dots + x_{i_k} = \frac{1}{2}$.

Such a function is called a *target function* and the set of all possible target functions is called *target class*.

Winnow maintains vector of weights $\mathbf{w} = (w_1, \dots, w_n)$ (as well as RWMA in Section 7.4.1). The prediction is based on comparing $\sum_{i=1}^{n} w_i x_i$ with a predefined threshold Θ (the recommended value is $\Theta = n/2$). Every x_i for which the classifier made a mistake ($x_i \neq y$) is either promoted ($w_i := \alpha w_i$, where α is a predefined constant), eliminated (Winnow 1, $w_i := 0$) or demoted (Winnow 2, $w_i := \frac{w_i}{\alpha}$). Thus, in a relatively small number of steps, the relevant features are selected.

The original Winnow is in fact a supervised online learner. In order to use Winnow for automated motion detection, Nair and Clark [31] equipped the algorithm with an *automatic labeller*. That is a low accurate classifier that satisfies two requirements:

1. *Automatic failure recognition:* The labeller must be able to tell when the output it provides is not reliable and thus the sample should not be used for updating the weight vector.

2. *Unbiased labelling:* The labeller must be aware of any biases and in case of uncertainty, it should rather label the sample as unreliable.

[27] Although Littlestone calls Winnow a reinforcement-learned classifier, it is in fact supervised classifier since the reinforcement comes with every sample.

[28] *Monotone disjunctive form* denotes such function that does not contain negated literal in any term.

The labeller acts for Winnow as the teacher[29]. Based on the labeller's feedback, Winnow decides whether and how to update the weight vector.

Nair and Clark also introduce how to transform Winnow target class from binary feature space to integer space. The base idea is to transform an integer value in interval $[0, N]$ to vector of boolean features of the form $x < t$ and $x \geq t$ for $t \in [1, N]$. Although this idea allows to use Winnow, it changes the sample space size dramatically.

An implementation that kept value of every such boolean feature would be very space-consuming. Therefore Nair and Clark propose to maintain the weight vector virtually by storing one weight for entire interval $[t_1, t_2)$.

However, the target function for KD is not in monotone disjunctive form. It can be expressed as sum of more disjunctive terms where each term corresponds to a particular key or digraph. But the disjunctive form functions can be used separately for each key, so the final form would be similar to:

$$\begin{aligned} \mathbf{x} &= \mathit{toBoolean}(\mathit{timing}) \\ f(k, \mathbf{x}) &= x_{k_1} \vee \cdots \vee x_{k_n} \end{aligned} \tag{7.19}$$

where k denotes the particular key or digraph and $\mathit{toBoolean}()$ denotes function transforming integer value to boolean vector, as described above. The integer value can really acquire only values from interval $[0, MAX_{key}]$ for single keys or $[-MAX_{digraph}, MAX_{digraph}]$, respectively.

Unfortunately, there rose a problem when I tried to adapt it for keyboard dynamics. Nair and Clark had a weak classifier for image recognising a person in an image but there is no such unsupervised classifier that would make such decision. One could raise an objection TL can be used. But TL is initialised to a fixed value and relies on prediction of another classifier.

I decided not to exercise this classifier any more due to this deadlock. However, combining Winnow and the one-class statistical classifier from Section 7.2.3 as a weak classifier might bring interesting results.

7.4.3 Adjusting supervised offline classifiers

Beyond the pure-online classifiers proposed above, one can also adjust supervised offline classifiers to operate in online mode. The basis is a trained supervised classifier that with each predicted sample decides whether to include it in the template.

That excludes classifiers which need to process whole training set again to update their template. Neural networks represent an example of such classifiers because they are trained iteratively over the whole training set.

Next important step is how the classifier decides which sample should update the template and which not. I tested two possible approaches.

Prediction probability is essentially a probability that the sample belongs to a particular class. From tested classifiers, such probability is provided only by naïve Bayes classifier.

Although `scikit-learn` implementation of k-NN classifier does not provide the probability prediction, it can be roughly expressed as number of samples in k-neighbourhood

[29] Note that Winnow is in fact a supervised method but the labeller is an unsupervised classifier. The joined classifier is therefore treated as unsupervised.

that belong to a particular class. However, the probability would not evince enough entropy for commonly used values of k ($k = 3$, $k = 5$). For that reason, this approach was not implemented with k-NN classifier.

The statistical classifier works with continuous Gaussian probability distribution and as such the probability of the exact value is always zero. A range around the sample value would have been selected. But what size should be the range? And how to calculate joint probability for more features? This approach was not therefore used with statistical classifier as well.

The approach with prediction probability has another large shortcoming. As it selects only samples close to the previous, it adds almost no new information. It follows that new training samples must be selected from wider context to bring new information.

Trust level history Another approach is to maintain a window of H last values of trust history. If all the values in the window are higher than some threshold t_{learn} then the sample can be used for online learning.

This approach is generally independent on the classifier. The learning threshold t_{learn} should be set slightly below the maximum TL value. Thus even if the initial TL is at its maximum value then it drops fast under t_{learn} for incoming stream of impostor samples. Value of t_{learn} therefore depends not only on the window size but also on the specific penalty & rewards function.

One should also consider what value to fill the history window with. If impostor stream comes to the input then high level might cause including impostor actions in the template. On the other hand if the window is filled with value less than t_{learn}, the stream of genuine has to wait until all the values in the history window are overwritten.

Perhaps the best option is to use t_{learn} as the initial filling of history window and initialize TL to t_{learn}. Then genuine samples are directly learned and impostor samples are isolated from online learning.

I decided to start with maximum TL since that little learning examples do not influence template significantly. This is because size of training set; I used 10,000 samples for each user.

Summary

Classification is a way how to automatically predict class of previously unseen data on the base of training examples. The common assumption is the data samples are i.i.d. (independent and identically distributed) and that is therefore possible to estimate function that generates them, a hypothesis.

Since machine learning is a well-documented science area, there exist libraries for many programming languages. The library I used is called `scikit-learn` and covers both supervised and unsupervised offline learning. I selected naïve Bayes and k-nearest neighbours classifiers for their ability to adjust the hypothesis. Moreover, I added one-class statistical classifier.

In the following sections, several pure-online classifiers are described but non of them is suitable for continuous authentication using keystroke dynamics. Therefore I propose how to adjust the offline classifiers to work in online setting. The biggest difficulty is picking samples that should be used for online learning. One of the approaches is based on a single sample, another one on the trust level value of recently evaluated samples.

Chapter 8

Experimental results

During the experiments, I trained the classifiers with 10,000 samples for each user. The training set was composed of the first 5,000 genuine user sample in their feature lists and 5,000 randomly chosen samples from other users, all shuffled in random.

I excluded User 22 from experiments due to very little data, as Figure 6.1 shows.

With each classifier, every user was tested against the rest of his dataset for testing genuine samples and against each other user for exercising the classifier with impostor actions. The performance was measured by average number of genuine actions (ANGA) and average number of impostor actions (ANIA) separately for each combination of user datasets (in total then 21 tests for ANGA and $21 \cdot 20 = 420$ tests for ANIA) that were calculated in the following way.

If the user was not locked out at all, the ANIA / ANGA value equals to the number of test examples. Otherwise the metric is computed as total number of action from the beginning to the last lockout divided by number of lockouts:

$$\text{ANA} = \frac{\#\text{ actions until last lockout}}{\#\text{ lockouts}}$$

The higher value of ANGA and the lower value of ANIA, the better the classifier performs. As those values are calculated for each user, the presented results are averaged.

8.0.4 Classifiers

In total, I exercised the following four classifiers.

Winnow I rejected the first one, Winnow, after I encountered it is not possible to use TL as a weak classifier. The problem was already described in Section 7.4.2.

Naïve Bayes classifier For naïve Bayes classifier (NB), I employed the `scikit-learn` class `GaussianNB` which is a classifier suitable for processing continuous variables. The NB evinced on average excellent 45,448 of genuine actions (ANGA) but very poor 36,108 impostor actions. That means almost no one was ever locked out. Adjusting trust level threshold did not help here because the TL was keeping around its maximum. Therefore I left naïve Bayes classifier behind as well.

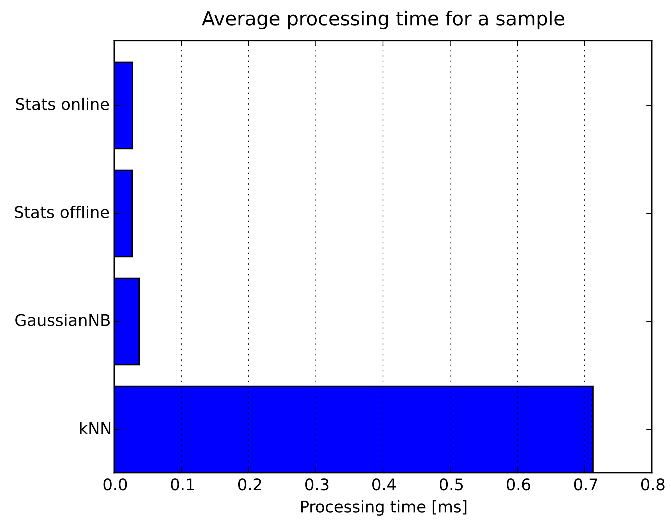


Figure 8.1: Average time for predicting one sample by different classifiers. Training set consisted of 10,000 samples.

***k*-nearest neighbours classifier** Next classifier I exercised, was k-nearest neighbours classifier. I tested it in default settings, namely with automatic algorithm selection (according to documentation, usually the `BallTree` algorithm is chosen) and $k = 5$. It performed much better than NB: a genuine user could make 1,483 actions on average and an impostor could make on average 145 actions. However it suffered from another large drawback, the recognition time. Figure 8.1 shows recognition time for all the classifiers. Whilst running all tests for NB or the statistical classifier took about 20 minutes, tests with k-nearest neighbours classifier (k-NN) took almost 5 hours each.

Moreover, the k-NN can basically use three different structures for storing the key, according to used algorithm. For brute-force algorithm, i.e. testing each novel sample with each template sample, the template consists simply from a list of sample and is therefore easily extensible. But the recognition time with the training set of 10,000 samples is almost three times longer than presented in Figure 8.1.

The other algorithms use a tree structure as a template. Nevertheless, the tree structure is a packed C library and `scikit-learn` library does not provide source codes for those. That means, the tree would have to be generated each time a novel sample comes into the template, which would increase the recognition time even more.

Statistical classifier I implemented the statistical classifier by myself, roughly following Bours' work [6]. Beyond that, I implemented the trust level history window for online learning.

Due to quite large number of possible configurations, I employed `GridSearchCV` (see Section 7.3) to find best values for the parameters. Grid search discovered several configurations with same score for each user. I decided for the most common setting: TL history window size = 20, TL threshold for learning = 80 and lockout threshold = 60.

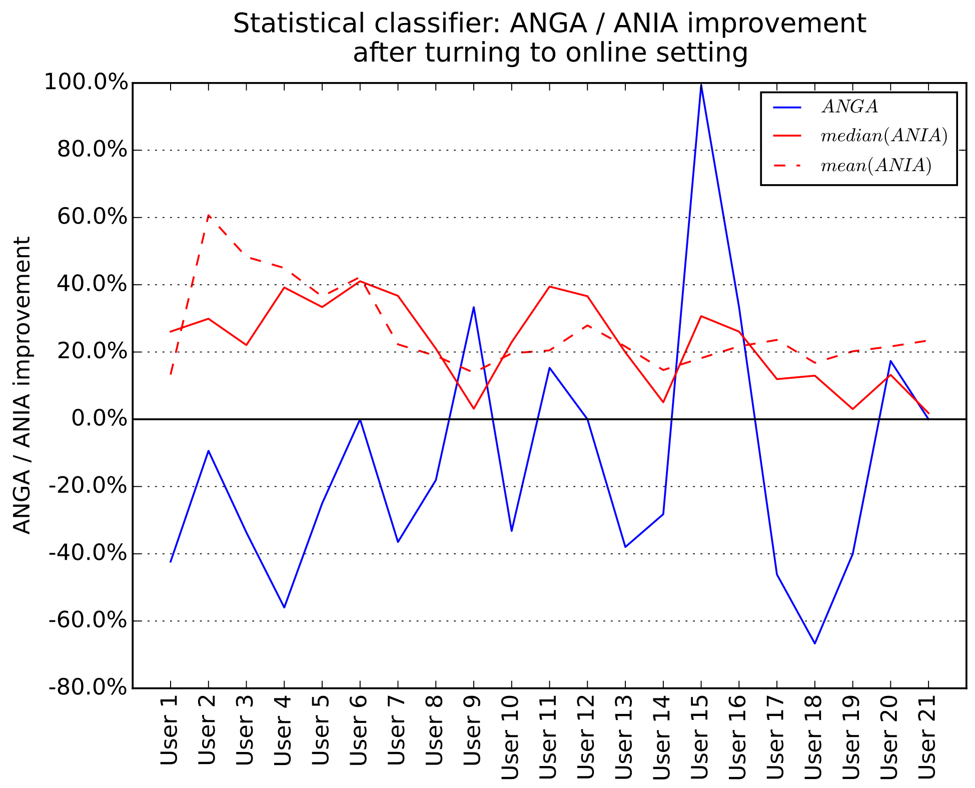


Figure 8.2: Improvement of ANGA and ANIA for the statistical classifier with TL history window size 20, $t_{learn} = 80$ and $t_{lockout} = 60$.

In the offline setting, the average ANGA value was 18,560 actions and average ANIA value 16,455. However this value is hugely influenced by several large numbers and the median ANIA value is 4,772.

After switching to online setting, the average ANGA value slightly decreased to 18,539 actions, the average value of ANIA decreased significantly to 12,538 and the median value to 3,374. That is an average ANIA improvement of 23.7 %.

The proportional change of ANGA and ANIA values from all the users is shown in Figure 8.2. One can notice that even though the ANGA became worse for most of the users[30] (however making no significant difference on average), the ANIA value, representing resistance against impostors, has improved for every tested user.

The large changes in ANGA values are caused by small number of genuine user lockouts both in offline and online setting.

[30] Note the almost 100 % improvement for User 15. He was locked out once in the online setting, approximately in the half of the test set. Online setting did not locked him at all so the ANGA value increased rapidly for him.

Chapter 9

Conclusion

This master thesis aimed to exercise whether it is possible to automatically adjust the template of a continuous biometric system after it is trained. As far as I know, this is the first work concerning this problem.

Almost 50 people agreed with collecting their keystroke data but only 22 of them eventually completed the capturing. I used an external program BeLT which runs under Microsoft Windows for collecting the data. I had to overcome several problems associated with BeLT because it does not consider different keyboard layouts and system-wide encoding. Thus all data files were normalised to multi-byte encoding UTF-8. It also allows to store data in two different file formats and several users selected by accident other file format than the others. I had to write the script for converting between those two formats.

Next challenge was pre-processing the data. As BeLT captures low-level system events, those were aggregated to form single key and digraph features. List of such features were stored for each user, in total 163 MB of data.

The consequent testing exercised four classifiers in order to answer the research question. The best performing offline classifier was k-nearest neighbours classifier but it operated very slow and was hard to extend for online learning.

However, an online improvement was achieved using a simple one-class statistical classifier. The template was adapted based on 20 last seen examples and this enhancement improved an average impostor recognition rate by 23.7 %.

The first research question is thus answered: yes, it is possible to adjust the template during recognition phase. Unfortunately, I cannot answer how it differs for various classification method as I have found no more methods worth experimenting with.

This topic could be extended in future work by exercising methods of adjusting and speeding up k-nearest neighbours classifier since it evinced very good results in online setting. Unfortunately it's implementation in `scikit-learn` library forced to choose between inefficient brute force and closed-source trees.

Bibliography

[1] J. Alexander, A. Cockburn, and R. Lobb. "AppMonitor: A Tool for Recording User Actions in Unmodified Windows Applications". English. In: *Behavior Research Methods* 40.2 (2008), pp. 413–421. ISSN: 1554-351X. DOI: 10.3758/BRM.40.2.413.

[2] S. P. Banerjee and D. L. Woodard. "Biometric Authentication and Identification Using Keystroke Dynamics: A survey". In: *Journal of Pattern Recognition Research* 7 (2012), pp. 116–139. ISSN: 1558-884X.

[3] D. Barber. *Bayesian reasoning and machine learning.* Cambridge University Press, 2012. 697 pp. ISBN: 978-0-521-51814-7.

[4] N. Bartlow. *Keystroke Recognition.* In: *Encyclopedia of Biometrics.* Ed. by S. Z. Li and A. K. Jain. Vol. 2. Springer, 2009, pp. 877–882. ISBN: 978-0-387-73002-8.

[5] S. Bleha, C. Slivinsky, and B. Hussien. "Computer-Access Security Systems Using Keystroke Dynamics". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 12.12 (1990), pp. 1217–1222. ISSN: 0162-8828. DOI: 10.1109/34.62613.

[6] P. Bours. "Continuous Keystroke Dynamics: A Different Perspective Towards Biometric Evaluation". In: *Information Security Technical Report* 17.1–2 (2012). Human Factors and Bio-metrics, pp. 36–43. ISSN: 1363-4127. DOI: 10.1016/j.istr.2012.02.001.

[7] Y. Chen and J.-C. Fondeur. *Biometric Algorithms.* In: *Encyclopedia of Biometrics.* Ed. by S. Z. Li and A. K. Jain. Vol. 1. Springer, 2009, pp. 64–68. ISBN: 978-0-387-73002-8.

[8] P. S. Dowland and S. M. Furnell. "A Long-Term Trial of Keystroke Profiling Using Digraph, Trigraph and Keyword Latencies". In: *Security and Protection in Information Processing Systems.* Ed. by Y. Deswarte et al. Vol. 147. IFIP International Federation for Information Processing. Springer US, 2004, pp. 275–289. ISBN: 978-1-4020-8142-2. DOI: 10.1007/1-4020-8143-X_18.

[9] R. R. Dunton. *Fingerprint Detecting Mouse.* US Patent 6,337,919. 2002. URL: http://www.google.com/patents/US6337919.

[10] C. Elkan. *Boosting and naive Bayesian learning.* Tech. rep. Technical Report CS97-557, University of California, San Diego, 1997.

[11] C. Feher et al. "User Identity Verification via Mouse Dynamics". In: *Information Sciences* 201 (2012), pp. 19–36. ISSN: 0020-0255. DOI: 10.1016/j.ins.2012.02.066.

[12] R. Giot, M. El-Abed, and C. Rosenberger. “GREYC Keystroke: A Benchmark for Keystroke Dynamics Biometric Systems”. In: *Biometrics: Theory, Applications, and Systems, 2009. BTAS’09. IEEE 3rd International Conference on*. IEEE. 2009, pp. 1–6. DOI: 10.1109/BTAS.2009.5339051.

[13] D. Gunetti and C. Picardi. “Keystroke Analysis of Free Text”. In: *ACM Transactions on Information and System Security (TISSEC)* 8.3 (2005), pp. 312–347. ISSN: 1094-9224. DOI: 10.1145/1085126.1085129.

[14] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95.

[15] *Information technology – Vocabulary – Part 37: Biometrics*. Norm. Dec. 15, 2012.

[16] A. Jain, R. Bolle, and S. Pankanti. “Introduction to Biometrics”. In: *Biometrics: Personal Identification in Networked Society*. Kluwer Academic Publishers, 1998, pp. 1–41. ISBN: 0-7923-8345-1.

[17] A. K. Jain and A. Ross. “Introduction to Biometrics”. In: *Handbook of Biometrics*. Ed. by Anil K. Jain, Patrick Flynn, and Arun A. Ross. Springer, 2008, pp. 1–22. ISBN: 978-0-387-71040-2.

[18] E. Jones, T. Oliphant, P. Peterson, et al. *SciPy: Open source scientific tools for Python*. 2001–. URL: http://www.scipy.org/.

[19] Z. Jorgensen and T. Yu. “On Mouse Dynamics As a Behavioral Biometric for Authentication”. In: *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*. ASIACCS ’11. ACM. ACM, 2011, pp. 476–482. ISBN: 978-1-4503-0564-8. DOI: 10.1145/1966913.1966983.

[20] K. S. Killourhy and R. A. Maxion. “Comparing Anomaly-Detection Algorithms for Keystroke Dynamics”. In: *Dependable Systems Networks, 2009. DSN ’09. IEEE/IFIP International Conference on*. 2009, pp. 125–134. DOI: 10.1109/DSN.2009.5270346.

[21] U. Kukreja, W. E. Stevenson, and F. E. Ritter. “RUI: Recording user input from interfaces under Windows and Mac OS X”. In: *Behavior Research Methods* 38.4 (2006), pp. 656–659. ISSN: 1554-351X. DOI: 10.3758/BF03193898.

[22] M. Leijten and L. Van Waes. *Inputlog. Description of Inputlog version 4.0Beta [concept version]*. University of Antwerp. 2009, p. 21. URL: http://www.inputlog.net/docs/DescriptionInputlogMay2009.pdf.

[23] S. Z. Li and A. K. Jain, eds. *Encyclopedia of Biometrics*. Springer, 2009. ISBN: 978-0-387-73002-8.

[24] N. Littlestone. “Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm”. In: *Machine learning* 2.4 (1988), pp. 285–318.

[25] J.-D. Marsters. “Keystroke Dynamics as a Biometric”. PhD thesis. University of Southampton, 2009.

[26] A. M. Martinez. *Biometric Algorithms*. In: *Encyclopedia of Biometrics*. Ed. by S. Z. Li and A. K. Jain. Vol. 1. Springer, 2009, pp. 355–358. ISBN: 978-0-387-73002-8.

[27] Microsoft. *About Messages and Message Queues*. Jan. 13, 2014. URL: http://msdn.microsoft.com/en-us/library/windows/desktop/ms644927(v=vs.85).aspx (visited on 2014-01-13).

[28] Microsoft. *Accessibility*. URL: https://msdn.microsoft.com/en-us/library/ms753388(v=vs.110).aspx (visited on 2015-05-18).

[29] S. Mondal and P. Bours. “Continuous Authentication Using Mouse Dynamics”. In: *International Conference of the Biometrics Special Interest Group (BIOSIG), 2013*. 2013, pp. 1–12.

[30] J. R. Montalvão Filho and E. O. Freire. “On the Equalization of Keystroke Timing Histograms”. In: *Pattern recognition letters* 27.13 (2006), pp. 1440–1446. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2006.01.010.

[31] V. Nair and J. J. Clark. “An unsupervised, online learning framework for moving object detection”. In: *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*. Vol. 2. IEEE. 2004, pp. II–317.

[32] A. Nait-Ali and R. Fournier. *Signal and Image Processing for Biometrics*. ISTE. Wiley, 2012. ISBN: 9781118588116. URL: http://books.google.co.uk/books?id=DCJHNZ5SYc8C.

[33] K. Nakajima et al. “Footprint-Based Personal Recognition”. In: *IEEE Transactions on Biomedical Engineering* 47.11 (2000), pp. 1534–1537. ISSN: 0018-9294. DOI: 10.1109/10.880106.

[34] L. O’Gorman. “Comparing Passwords, Tokens, and Biometrics for User Authentication”. In: *Proceedings of the IEEE* 91.12 (2003), pp. 2021–2040. ISSN: 0018-9219. DOI: 10.1109/JPROC.2003.819611.

[35] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[36] K. Revett et al. “A Machine Learning Approach to Keystroke Dynamics Based User Authentication”. In: *International Journal of Electronic Security and Digital Forensics* 1.1 (2007), pp. 55–70. ISSN: 1751-911X. DOI: 10.1504/IJESDF.2007.013592.

[37] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 3rd edition. Pearson Education, Inc., 2010. ISBN: 978-0-13-207148-2.

[38] S. J. Russell and P. Norvig. “Learning from examples”. In: *Artificial Intelligence: A Modern Approach*. 3rd edition. Pearson Education, Inc., 2010, pp. 693–767. ISBN: 978-0-13-207148-2.

[39] S. J. Russell and P. Norvig. “Learning probabilistic models”. In: *Artificial Intelligence: A Modern Approach*. 3rd edition. Pearson Education, Inc., 2010, pp. 802–829. ISBN: 978-0-13-207148-2.

[40] M. Rybnik, M. Tabedzki, and K. Saeed. “A Keystroke Dynamics Based System for User Identification”. In: *Computer Information Systems and Industrial Management Applications, 2008. CISIM ’08. 7th*. 2008, pp. 225–230. DOI: 10.1109/CISIM.2008.8.

[41] Y. Shafranovich. *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. 2005. URL: https://tools.ietf.org/pdf/rfc4180.pdf (visited on 2015-03-26).

[42] S.J. Shepherd. “Continuous authentication by analysis of keyboard typing characteristics”. In: *Security and Detection, 1995., European Convention on.* 1995, pp. 111–114. DOI: 10.1049/cp:19950480.

[43] T. Sim and R. Janakiraman. “Are Digraphs Good for Free-Text Keystroke Dynamics?” In: *IEEE Conference on Computer Vision and Pattern Recognition, 2007. CVPR'07.* IEEE. 2007, pp. 1–6.

[44] T. Sim et al. “Continuous Verification Using Multimodal Biometrics”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 29.4 (2007), pp. 687–700.

[45] D. Stefan and D. Yao. “Keystroke-Dynamics Authentication Against Synthetic Forgeries”. In: *6th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2010.* IEEE. 2010, pp. 1–8.

[46] R. Stenvi, M. Øverbø, and L. Johansen. “Behaviour Logging Tool – BeLT”. bachelor thesis. Høgskolen i Gjøvik, 2013, p. 86. URL: http://brage.bibsys.no/xmlui/bitstream/id/99084/LTJohansen_RStenvi_MOverbo.pdf.

[47] *scikit-learn tutorial. Choosing the right estimator.* URL: http://scikit-learn.org/0.16/tutorial/machine_learning_map/ (visited on 2015-05-15).

[48] S. van der Walt, S. C. Colbert, and G. Varoquaux. “The NumPy Array: A Structure for Efficient Numerical Computation”. In: *Computing in Science Engineering* 13.2 (2011), pp. 22–30. ISSN: 1521-9615. DOI: 10.1109/MCSE.2011.37.

[49] R. V. Yampolskiy and V. Govindaraju. “Behavioural Biometrics: A Survey and Classification”. In: *International Journal of Biometrics* 1.1 (2008), pp. 81–113.

[50] E. Yu and S. Cho. “Keystroke Dynamics Identity Verification – Its Problems and Practical Solutions”. In: *Computers & Security* 23.5 (2004), pp. 428 –440. ISSN: 0167-4048. DOI: 10.1016/j.cose.2004.02.004.

Acronyms

***k*-NN** *k*-nearest neighbours classifier. 48, 50

ANGA average number of genuine actions. 15, 17, 47, 49

ANIA average number of impostor actions. 15, 17, 47, 49

BB behavioural biometrics. 8, 9, 17

EER equal error rate. 15, 17, 23

FAR false acceptance rate. 13–17

FRR false rejection rate. 13–18

GUI graphical user interface. 8

HCI human-computer interaction. 25, 26

KD keystroke dynamics. 3, 12, 13, 18–20, 22–26, 29, 45

ML machine learning. 7, 22–24, 42

NB naïve Bayes classifier. 47, 48

NN neural network. 21, 23

OS operating system. 10, 25, 28

PCA principle component analysis. 26

PIN personal identification number. 5, 8, 18

PNN probabilistic neural network. 23

RFC request for comments. 28

SVM support vector machine. 23, 56

TL trust level. 13, 14, 45–49

X11 X window system. 26

Appendix A

Used software

This master thesis would not be created without several software packages. They are listed below.

BeLT v2.0.21 BeLT (Behaviour Logging Tool) [46] is a program developed at Høgskolen i Gjøvik (Gjøvik University College) in Norway to help with collecting users' data for behavioural biometrics research. It's behaviour is more thoroughly described in Chapter 5 including the file format and several bugs.

Python v2.7.9 Data processing and simulation was written in Python 2. I chose the language for it's suitable for fast scripting and prototyping as well as for larger software packages. Moreover, there exist many scientific libraries for Python that are described in following paragraphs.

`numpy` library v1.9.2 [48] is an efficient library for mathematic computation in Python. Especially important for this master thesis was support for operations over arrays and matrices and generating random numbers according to specified probability distribution.

`scikit-learn` library v0.16.2 [35] is a Python library implementing classical machine learning algorithms such as Naïve-Bayes, SVM or k-Nearest Neighbours. It provides a unified interface for classifiers so all the classifiers implemented this interface so it could be used in the simulator.

`scipy` library v0.15.1 [18] is a base library for `scitkit-learn` and is required for its run.

`matplotlib` library v1.4.3 [14] is a Matlab-style library for 2D plots. The plots are highly customizable and are generated as vector graphics, therefore suitable for including in printed work.

Appendix B

CD contents

You can find several subdirectories in the root directory of the attached CD.

```
src/
    +- Makefile
    +- common/
        +- <source files common for extraction and processing>
    +- helpers/
        +- graphs.py      (generating graphs)
        +- raw2csv.py     (transform BeLT RAW file to CSV)
        +- stats.py       (process statistics of results)
    +- tests/
        +- <tests for covering major classes>
    +- extractfeatures.py (creates features lists from CSV)
    +- run\_grid.py       (compares different classifiers)
    +- run\_simulation.py (run simulation and write results)
    +- runtests.py        (runs all tests)
log/
    +- <unprocessed results in CSV>
tex/
    +- Makefile
    +- <LaTeX source files>
```

Please note the user data are not stored on the CD nor the extracted feature. This is due to their private character. The data were burn to a separate CD which is stored by my supervisor doc. Drahanský.

Directory `src/` contains Python scripts for extracting features, classification and simulation. Purpose of individual files is annotated in the listing above or can be deducted from in-file documentation. Files are documented in compliance with Python best practices[31].

[31] The documentation conventions are described in PEP 257 (https://www.python.org/dev/peps/pep-0257/)