UNIVERSIDADE BEIRA INTERIOR
Engenharia

Multi-Factor and Continuous Verification of Identity during Remote Assessments and Individual Personalized Interactions

Versão final após defesa

Pedro Miguel Ferreira de Oliveira

Dissertação para obtenção de grau de mestre.
Engenharia Informática
(2º ciclo de estudos)

Orientador: Professor Pedro R. M. Inácio
Co-orientador: Professor Hugo Pedro Proença

Covilhã, Janeiro 2022

Acknowledgements

Em primeiro lugar quero agradecer à minha família, namorada e amigos por todo o incondicional apoio que me deram durante o meu percurso académico. Sem eles tudo isto seria ainda mais desafiante.

De seguida, um agradecimento ao Professor Doutor Pedro Inácio e ao Professor Doutor Hugo Proença por toda a orientação, paciência e disponibilidade aquando da realização desta dissertação.

Por fim, quero agradecer aos meus colegas de curso, pois desempenharam um papel fundamental nesta etapa da minha vida.

The work reflected in this dissertation was carried out in the Multimedia Signal Processing Laboratory of the Instituto de Telecomunicações delegation at Covilhã and in the University of Beira Interior, Covilhã, Portugal.

Resumo

A Autenticação é um dos aspetos mais importantes da sociedade atual. Quase todas as ações concretizadas pelas pessoas que a compõem têm um aspeto tecnológico. Caso os sistemas não sejam implementados segura e corretamente, nomeadamente através do controlo de acesso via autenticação, vários aspetos de (ciber)segurança e privacidade podem ser violados por pessoas mal intencionadas. Este acesso pode dar-se de muitas formas, sendo uma delas fazendo-se passar por um outro utilizador.

Esta dissertação visa a criação de um protótipo de sistema que consiga verificar a identidade do utilizador autenticado no sistema. Esta verificação será concretizada através da análise de biométricas comportamentais, neste caso, dinâmicas de rato e teclado, para realizar a coleta de interações com o sistema e aprendizagem automática para prever, através da informação coletada, se esta corresponde à pessoa autenticada. Caso o algoritmo detete uma anomalia, o computador irá bloquear, de modo a que apenas pessoas autorizadas consigam fazer uma reautenticação.

Palavras-chave

Aprendizagem Automática, Biométrica Comportamental, Cibersegurança, Verificação Contínua

Resumo alargado

Visto que grande parte desta dissertação foi escrita utilizando a língua inglesa, por forma a explicá-la também em língua portuguesa, este capítulo irá resumir mais amplamente todo o trabalho realizado.

Introdução

Esta dissertação tem como objetivo a construção de um protótipo para um sistema de verificação contínua de utilizadores utilizando multi-fatores biométricos. A abordagem escolhida foi fatores biométricos comportamentais, onde serão usados rato e teclado de forma a recolher informação, visto que é necessária uma interação continuada com o sistema.

Motivação e Enquadramento

Existe uma vasta quantidade de ramificações na área de engenharia informática e todas desempenham um papel vital no dia a dia. Os computadores e outras tecnologias desempenham funções muito importantes na sociedade, desde o entretenimento, a guardar e tratar dados sensíveis de grandes empresas. Visto que as tecnologias estão intrinsecamente ligadas à nossa sociedade, é necessário que estas funcionem devidamente e com o máximo de segurança e privacidade possível. Com o aumento dos dispositivos pertencentes à Internet das Coisas (abreviado IoT, do inglês *Internet of Things*), vão existir cada vez mais dispositivos que se conectam entre si, criando uma grande abertura para novas suscetibilidades, reforçando assim a necessidade de sistemas cada vez mais aptos a lidar com tais falhas.

O melhor método para lidar com estes possíveis problemas tem o nome de cibersegurança. A definição apresentada pelo NIST (*National Institute of Standards and Technology*) é "processo de proteger informação através de prevenção, deteção e resposta a ataques" [Nat18]. A cada ano que passa, os governos e empresas gastam vastas quantidade de dinheiro e recursos com o objetivo de melhorar a segurança das suas infraestruturas tecnológicas contra todo o tipo de ataques informáticos. Portugal, por exemplo, tem uma série de leis que obrigam as empresas e entidades a manter uma infraestrutura informática regulamentada [Assb] sob pena de multa e leis que punem quem for acusado de cometer crimes informáticos [Assa] com multas e/ou pena de prisão. Foram criados também o Centro de Ciberdefesa e o Centro Nacional de Cibersegurança. O primeiro tem como principal objetivo vigiar e proteger o sistema informático da Defesa Nacional, o segundo tem como função consciencializar e educar a população neste assunto.

Grande parte das interações que os utilizadores têm com aparelhos tecnológicos envolvem algum tipo de autenticação, como desbloquear o telemóvel ou utilizar o cartão de crédito.

Um dos métodos de autenticação que se tornou mais prevalente é a autenticação biométrica, a qual está presente em quase todos os *smartphones* e *tablets* de última geração, através do uso de reconhecimento facial ou de impressão digital. Esta categoria de autenticação, que faz parte do grupo *something you are* é geralmente mais seguro que métodos de palavra-passe e até mesmo que autenticação de dois fatores. Isto deve-se ao facto de o utilizador ser estritamente necessário para aceder ao dispositivo. No entanto, esta informação nunca pode ser mudada, portanto, se for comprometida, não poderá ser usada novamente.

Aquando da escrita desta dissertação, o mundo enfrenta uma pandemia global que obrigou a que trabalho e acessos remotos se tornassem uma realidade constante. Isto levantou questões sérias sobre cibersegurança. Pois, sem a pessoa presente, nunca se tem a total certeza de que quem está por de trás do computador é quem de facto deveria estar. A melhor forma de garantir isto é utilizando verificação contínua.

Problemas e Objetivos

Como foi dito anteriormente, o objetivo principal desta dissertação é a criação de um sistema de verificação contínua utilizando informação biométrica comportamental. Ao criar um programa capaz disto, surgem algumas questões sobre privacidade, portanto o sistema nunca poderá registar o que o utilizador está a escrever nem onde está a clicar, a solução é registar tempos de ação. Uma série de objetivos foram traçados para a criação deste projeto:

1. Analisar e delinear o estado da arte relativo à verificação contínua de forma a compreender o que já foi feito e como funciona;
2. Compreender quais a melhores características a retirar para ter os melhores *datasets* possíveis;
3. Encontrar e implementar o algoritmo de aprendizagem automática que melhor se adeque ao problema;
4. Ajustar os hiperparâmetros do algoritmo de forma a obter os melhores resultados;
5. Testar e validar o sistema.

Método para Resolução dos Problemas

Primeiramente encontrar artigos que se encaixem no tema pretendido e realizar uma leitura compreensiva dos mesmos. Seguidamente investigar algoritmos que já tenham sido utilizados anteriormente em sistemas semelhantes para conhecer possíveis candidatos. Depois, criar um subsistema para coleta de informação dos utilizadores para montar os *datasets* e implementar o subsistema de aprendizagem que criará os modelos utilizados

posteriormente na avaliação. Por fim, ajustar os hiperparâmetros e implementar o subsistema de verificação.

Contribuições Principais

Principais contribuições retiradas deste trabalho:

- Um estudo do estado da arte relativo à verificação contínua de utilizadores utilizando informação biométrica;
- O protótipo de um sistema de verificação contínua com uso de informação biométrica comportamental;
- Uma dissertação escrita em inglês para obtenção do grau de mestre em engenharia informática.

Background e Trabalhos Relacionados

O capítulo 2 tem como objetivo enquadrar o leitor no tema, explicando diversas noções quer de segurança informática, quer de aprendizagem automática, de forma a que este compreenda o trabalho. Inicialmente é apresentado um sumário dos métodos de autenticação que existem seguido de uma revisão de vários algoritmos de aprendizagem automática que pudessem ter interesse no contexto deste trabalho.

Posteriormente é apresentado o estudo de artigos científicos sobre a temática em questão, onde são analisados sistemas que utilizam informação biométrica para realizar avaliações, estudos comparativos de algoritmos e estudos sobre a influência das emoções na utilização do computador.

Requisitos e *Design* do Sistema

Neste capítulo são apresentados os requisitos funcionais e não funcionais a que este projeto está sujeito, assim como o *design* e funcionalidade do mesmo. Os requisitos funcionais descrevem o que o sistema deve fazer, de uma forma direta. Os requisitos não funcionais explicam especificações do sistema de forma mais técnica.

Estes requisitos irão influenciar os métodos de implementação do sistema. É apresentada a figura 3.1 onde está ilustrado como o sistema funciona e a figura 3.2 onde se podem ver as características retiradas do uso do teclado.

Implementação do Sistema

Neste capítulo é feita uma descrição de todas as tecnologias usadas, assim como as bibliotecas *python* utilizadas para a construção do protótipo. De seguida está uma descrição

detalhada da implementação de cada parte do sistema, começando no *enrollment*, onde os utilizadores se registam no sistema e onde está também uma descrição dos hiperparâmetros utilizados no algoritmo.

De seguida encontra-se a verificação onde é mostrada a figura 4.1 que ilustra a lógica deste subsistema. Finalmente, uma reflexão dos obstáculos encontrados e como foram superados.

Resultados e Discussão

Neste antepenúltimo capítulo serão discutidos os materiais usados no sistema assim como todos os resultados obtidos nos testes realizados durante a implementação e após o sistema se encontrar operacional. A apresentação dos materiais permite que, caso haja uma tentativa de utilização do sistema, este seja compreendido mais profundamente e que o computador utilizado possa ser comparado ao que foi usado para a criação do sistema.

De seguida está presente uma exposição de todos os testes realizados ao sistema antes, durante e após a implementação. Nestes testes é possível encontrar os passos que levaram a criar os *datasets* daquela forma específica, como os hiperparâmetros foram ajustados, as melhorias que o sistema obteve com estes testes e, por fim, uma série de testes com o objetivo de comprovar a eficácia do sistema de verificação contínua quando está perante o utilizador real e um falso. Finalmente, uma discussão sobre os resultados dos testes feitos anteriormente.

Conclusão

No último capítulo encontram-se as conclusões globais do trabalho realizado e uma reflexão sobre o trabalho que poderia ainda ser melhorado ou realizado para que o sistema fosse mais versátil, ou para que tivesse mais funcionalidades.

Abstract

Authentication is one of the most important aspects of the present society. Almost everything people do has a technological aspect. If systems are not implemented correctly and safely, namely by access control via authentication, several aspects of (cyber)security and privacy can be exploited by ill-intentioned people. Unauthorised access can be made in multiple ways, pretending to be another user is one of them.

This dissertation aims at the creation of a system prototype that can verify if the logged-in user is the actual person using the system. This is achieved utilising behavioural biometrics, such has keyboard and mouse dynamics, to collect interactions with the system and machine learning to predict if the collected information matches the person one says to be. In case the algorithm detects an anomaly, the computer locks itself, so that only authorised people with true access to the computer can re-log in.

Keywords

Behavioural Biometrics, Continuous Verification, Cybersecurity, Machine Learning

Contents

List of Figures

List of Tables

Listings

Acronyms

2FA	Two Factor Authentication
ACM	Association for Computing Machinery
ATM	Automated Teller Machine
AUC	Area Under Curve
CCS	Computing Classification System
COTS	Commercial off-the-shelf
COVID-19	Coronavirus Disease
CPI	Counts Per Inch
CSV	Comma-Separated Values
DDR3	Double Data Rate 3
DLL	Dynamic-Link Library
IDE	Integrated Development Environment
IoT	Internet of Things
IT	Information Technology
KNN	K-Nearest Neighbours
NIST	National Institute of Standards and Technology
PCA	Principal Component Analysis
PIN	Personal Identification Number
RAM	Random Access Memory
ROC	Receiver Operating Characteristic
SSD	Solid State Drive
SVC	C-Support Vector Classification
USB	Universal Serial Bus

Chapter 1

Introduction

The project behind this dissertation aims at the development of a prototype for a continuous verification software using multi-factor authentication. Behavioural biometrics via keyboard and mouse was the chosen approach to achieve the continuous verification due to the necessary contact the user has to maintain with both peripherals in order to use the computer. These have a cost benefit, as well as the fact that every ordinary computer has both. The present chapter is organised as follows:

- 1.1 - **Motivation and Scope** - contains the presentation of the motivation and discusses the scope of this dissertation;
- 1.2 - **Problem Statement and Objectives** - includes an explanation of the general problem and the objectives that must be reached;
- 1.3 - **Proposed Approach** - discusses the steps taken to reach the proposed goal;
- 1.4 - **Main Contributions** - lists the main contributions that are expected to be taken from this work;
- 1.5 - **Document Organisation** - presents how this document is organised.

1.1 Motivation and Scope

There are a vast number of areas under computer science and engineering, and all of them have a vital role in our daily life. Computers and other technologies play a main role in current society, from entertainment to handling and storing corporate sensitive data. Since they are so intrinsically connected with the society, they have to work properly and with maximum security and privacy possible. Given the rise of the Internet of Things (IoT), a large amount of devices are constantly connected, opening more doors for attacks, which means more precise and fault-tolerant security algorithms/systems are needed.

To address this problem, cybersecurity is the go to. By the National Institute of Standards and Technology (NIST) definition, cybersecurity is "the process of protecting information by preventing, detecting, and responding to attacks" [Nat18]. With every passing year, governments and companies are spending large amounts of money and resources to see their Information Technology (IT) infrastructure protected against all types of attacks. Portugal has a series of laws regarding not only the establishment of legal framework for cyberspace security [Assb], which has to be applied to all companies and entities that utilise information technologies under penalty of fine, but also on punishment for cybercrimes [Assa], which can be financial penalties or imprisonment. Other efforts to raise

awareness and educate the population for the importance of cybersecurity are the creation of *Centro de Ciberdefesa* and *Centro Nacional de Cibersegurança*. The *Centro de Ciberdefesa* has the duty of surveillance and protection of the National Defence information systems, while the *Centro Nacional de Cibersegurança* promotes cyberspace usage in a free, reliable and secure way. This is achieved with many policies and campaigns, one of them was the creation of the *Quadro Nacional de Referência para a Cibersegurança* [Cen19], where an ensemble of the best cybersecurity practices was compiled, which is expected to promote voluntary risk reduction by companies and individuals simply by following the provided guidelines.

Most daily activities involving technologies have some sort of authentication step, like unlocking a smartphone or using a credit card. Although the most common type of user authentication is password-based, many new smartphones and laptops are capable of biometric authentication, using fingerprints or facial recognition. Biometric authentication is usually safer than password-based methods, and even when compared to two-step authentication, because the user is not required to remember or carry anything and is a big barrier against theft or loss since the user is needed to access the contents of said device. Albeit if this information is stolen, the user can never change this characteristics in contrary to passwords.

Since these security methods are present in many recurrent activities, usability is a key factor as well. If a user is to unlock a protected phone several times during the day, it is expected that the process goes as smooth, easy and with the least intrusiveness as possible. Complicating this process may drive the users to remove or try to bypass the process, creating security flaws.

Due to the Coronavirus Disease (COVID-19) pandemic, the world saw an even bigger shift to remote assessments and connections. Online classes and remote work are more and more popular as governments and companies enforce these measures in order to avoid COVID-19 outbreaks. Privacy and security in corporations is a topic with an increasing importance. If people are not present in the workplace, verifying that the person authenticated in a given computer is who they say they are is a challenging task. In this scenario, a continuous verification system would help give certainties to system administrators from that company. A system like this must also grant that the privacy of that user is never broken. A wide variety of methods are possible to use, but not all of them are suitable for all situations, and this dissertation will explore this topic.

Following the 2012 version of the Association for Computing Machinery (ACM) Computing Classification System (CCS), the categories in which this dissertation might be related to are as follows:

- **Security and privacy;**
- *Authentication*;

- *Biometrics*;
- Security services.

1.2 Problem Statement and Objectives

Based on the aforementioned motivation and scope, this dissertation objective is to study and evaluate continuous authentication methods so that a system prototype can be designed using what was found. Using keyboard and mouse dynamics can raise a lot of questions, specially when the topic is privacy. The system being unable to detect which keys are being pressed is of major importance, as well as not understanding where specifically the mouse is and where it is clicking. In any way an authentication system must be able to detect what kind of activities the user is doing, this being even more relevant in a continuous authentication system.

Hence, the ensuing objectives can be defined:

1. Analyse and outline the state of the art regarding continuous verification / authentication systems to get a better understanding on the technologies that can be used to implement the system in the most secure and usable way;
2. Understand the most suitable features to extract in order to develop an effective data collection subsystem;
3. Find and implement an appropriate machine learning algorithm for the presented objectives;
4. Tune machine learning algorithm hyperparameters in order to produce the best possible outcomes;
5. Test the proposed system to make sure everything works as expected.

1.3 Proposed Approach

To meet the aforementioned objectives, the execution plan shown below was put into practice:

1. First and foremost, a reading of research papers describing the state of the art is to be performed in order to understand existing solutions regarding the problem at hands, as well as to know what challenges might be found;
2. Investigate previously used algorithms and comparisons between diverse algorithms to get a better idea of which to use;
3. Create a subsystem capable of thoroughly collect the features defined as ideal for this project;

4. Create a prototype of the system where continuous authentication, using the approach of this work, is needed and where the machine learning algorithm will build a model to be used on the verification step;
5. Tune the hyperparameters to achieve the best performance possible;
6. Develop the continuous verification subsystem. Created models for both keyboard and mouse will be used to continuously authenticate the respective users.

1.4 Main Contributions

After this project is complete, the main contributions expected to derive from it are the following:

- An in-depth study of the state of the art regarding continuous verification of users using biometrics;
- A prototype of a continuous verification system using biometrics.

1.5 Document Organisation

This dissertation is divided in six chapters. The remaining chapters can be described as follows:

1. Chapter 2 - **Background and Related Works** - provides a review on authentication factors, machine learning and an overview on related works;
2. Chapter 3 - **Workflow and Requirements** - dedicated to the system requirements and to how the development flowed;
3. Chapter 4 - **System Implementation** - describes the technologies used in this dissertation and the implementation details and challenges;
4. Chapter 5 - **Results and Discussion** - presents the specifications of the computer used in this work, the tests and results and the discussion on those;
5. Chapter 6 - **Conclusion** - presents the conclusion for this dissertation, as well as the future work.

Chapter 2

Background and Related Works

2.1 Introduction

This chapter describes the current state of continuous authentication methods, security and machine learning algorithms. Having a good understanding of these topics, led to a more fluid and comprehensive development of the proposed system, as well as a swifter solving of problems and errors encountered during this development. In order to reach the goal, this chapter is split as follows:

- 2.2 - **Background** - explains definitions and notions vital to the dissertation context;
- 2.3 - **Related Works** - analysis of past and present works on user authentication and continuous user authentication;
- 2.4 - **Conclusion** - summary of the information taken from this chapter.

2.2 Background

There is a wide variety of technologies, definitions and methods on the context of user authentication. Therefore, to better understand the underlying implications of this dissertation, all of these must be given some level of explanation.

2.2.1 Authentication Factors

First and foremost, knowing what authentication factors exist is an important starting point. There are three authentication factors:

- Something you know: Being the most widely known and used, this factor is something the user knows, as the name suggests [Ros20]. A password or a Personal Identification Number (PIN) are the most well known examples;
- Something you have: An item the user possesses, such as tokens or a smart card [Ros20];
- Something you are: Biometric methods. Fingerprints, retina scan, iris scan, etc. can be used. These are called physiological biometrics, because they are physiological traits the user has [Ros20]. Behavioural biometrics identify the user using intrinsic patterns, like typing rhythm or mouse usage.

Although the *something you know factor* is the most used, it is also the easiest to surpass. PINs usually are very short, with only four digits, and are tipically made of only numbers. Passwords tend to be more versatile, accepting numbers, letters and symbols while also being longer. An inherent problem to password usage is the memory of the user. Remembering what to write in so many different applications and websites may lead to using the same password in every single one of them or using weak combinations.
To always carry an asset to successfully authenticate oneself is sometimes impractical. Something you have factor is haunted by this problem, because sometimes authentication may be needed in a place where that asset is not present. Furthermore, losing it is an even bigger problem, given it can fall in bad hands.
The *something you are factor* is the most secure of the three, but if not used correctly, its practical uses drop to zero. Biometrics suffer from two distinct types of error: *false rejection* and *false acceptance*. The *false rejection* error lowers the usability of a system, while false acceptance errors lower its security. Reduced usability comes from the need for repeated authentication due to the system taking action after falsely rejecting a legitimate user. Although this is not a security issue, it makes the system feel cheap and clunky. *False acceptance* of an individual gives system access to an illicit user, compromising its effective security. Weak algorithms or faulty biometric scanners can be the cause.

There are two different ways to use these factors. Single-factor authentication is much less difficult to use than multi-factor, but with less security, as it is shown further:

- Single-factor authentication makes use of only one mean of authentication. For example, to access many social media websites, the only barrier is a password (something you know). In one way, the user access is facilitated, but in exchange, the security layer is thinner than it should;

- Multi-factor authentication uses a combination of two or more factors. For the most part only two are used, being then called Two Factor Authentication (2FA). The most widely used example is the credit/debit card usage. For a user to withdraw money from an Automated Teller Machine (ATM) or make a payment, first the card must be presented (something you have) and after that, the corresponding PIN combination (something you know) [oST].

2.2.2 Machine Learning Algorithms

This work will resort to machine learning algorithms as a means to achieve the main objetive of performing continuous verification. As the system collects data, these algorithms are repeatedly applied to ensure the collected data belongs to the right person. Choosing the most suitable algorithm to fulfil the proposed objectives is utterly important and rather difficult. There are three main machine learning algorithm categories [SR18], each one of them with a distinct assortment of specific algorithms, for specific purposes:

Supervised Learning: Given an amount of data consisting of input-output pairs, the algorithm will learn a function that maps inputs to respective outputs, based on it. Then the algorithm should be able to predict the class of a given input, if the training ends up successful [Bro21] [Edu]. The most common algorithms of supervised learning are:

- Classification: Determination of to what class a new observation belongs based on pattern recognition; classification algorithms are often called “classifiers”. The algorithm used to develop this project, XGBoost, is included in this classification.
- Regression: Estimates the relationship between a dependent variable and one or more independent ones, therefore can be used for prediction and forecasting numeric labels.

Unsupervised Learning: As the name says, this class of techniques do not need human supervision. Unlabelled datasets are fed to these algorithms that will find and extract features that enable grouping of dataset elements [Bro21] [Edu]. Examples of techniques that can be included in unsupervised learning are Cluster analysis and Principal Component Analysis (PCA):

- Cluster Analysis: Also called clustering, is the task of separating the dataset into clusters (groups) where, in each cluster, each instance is more similar to its peer than to other instances on other clusters;
- Principal Component Analysis (PCA): In a graphical manner, is the task is to find the projected data direction, in the original space, with minimal projection error. It is typically used as a pre-processing technique to reduce dimensionality while preserving data relations, to which machine learning algorithms can then be applied.

Reinforcement Learning: Comparing reinforcement learning with a real life example is perhaps the best way to explain it. When a child is learning about his surrounding world, often the parents/caretakers reward the child when they do something of value, and punish them otherwise. Afterwards the child will remember, and learn, what should or should not do. With these algorithms the premise is the same, given an environment the best actions are to be taken in order to maximise a cumulative reward, unveiling the optimal path [Bro21] [Edu].

2.3 Related Work

A state of the art review regarding continuous user authentication is presented is this section. A summary of each article is presented, followed by a reflection on what stands out and can be used for this dissertation. In table 2.1 a summary of the studied works can be seen.
The list discussed below is a subset of the works found by searching terms such as ”Continuous” AND ”Verification” OR ”Biometrics” OR ”Keystrokes” OR ”Mouse” in search engines such as IEEE Xplore.

Continuous Verification Using Multimodal Biometrics:
Sim *et al.* [SZJK07] - Construction of a system to continually verify the presence of a logged-in user through the use of passive biometrics, fingerprint scan and facial recognition. The system was put together using a holistic approach and a Bayesian framework. This allowed the system to predict if the user was still there or not, even in the absence of observations. The observations were reduced to a score based system, were the scoring after the observation dictated if an action should be taken or not. A hidden Markov Model is behind the integrator used in order to evaluate both the results from fingerprint and face recognition, along with a decaying factor for when there are no observations.

User Identity Verification via Mouse Dynamics:
Feher *et al.* [FEM$^+$12] - Describes a behavioural biometrics method using the mouse. Despite working with keyboard as well, the focus of the work shifted primarily to mouse due to the main objective of the work is to be used on the web browser, place where mouse usage outweighs the keyboard. System flow starts on feature acquisition, capturing user interactions with the mouse, following feature extraction, where a signature is constructed with the acquired features. Then a classifier builds a model based upon the extracted features and saves this to the signature database, so it can be used for verification. Results were overall positive, but mouse type fluctuations make this approach less viable.

Keystroke/Mouse Usage Based Emotion Detection and User Identification:
Shikder *et al.* [SRAA17] - Emotions are an intrinsic trait of living beings, therefore it is really difficult to emulate and detect them artificially. In order to detected emotions, a lot of resources are needed, so this paper aims at presenting a simpler and cheaper way of doing so. Using an assortment of carefully chosen videos, emotional states are induced on the user, and keyboard and mouse interactions are collected after, via a custom-made survey. Nine videos were presented, to try inducing nine different emotions, varying from happiness to fear. This study tested 35 participants. Tests showed that for many of the used classifiers a set of emotions were dominant over the rest, having around 60% accuracy, which was considered relatively low.

Emotions and User Interactions with Keyboard and Mouse:
A. Pentel [Pen17] - Different emotional states are a crucial topic to be aware of when trying to create more adaptive systems. When people communicate with each other, not only they listen to what the others have to say, but they also pay attention to their body language, facial expressions, etc. However when dealing with a computer, the machine is unable to do so without special equipment. In this work, a method of reading the user emotional state is presented via the usage of mouse and keyboard dynamics. One of the objectives set with this work was to give a simple way to use software like this, not needing dedicated and complicated software, so JavaScript was the chosen language. Emotions to be emulated were: amusement, happiness, sadness and fear. To induce these emotions,

a series of videos and tasks were presented to the users, where they were reduced to a two-dimensional scale: positive-negative and active-passive. Data was then collected and models constructed, utilising various algorithms. Mouse usage got the best results out of the two.

Continuous User Authentication Using Temporal Information:
Koichiro and Jain [NJ10] - Creation of a continuous verification system that does not require active user participation. For this, the system must be simple, but not too simple, otherwise it would be less effective, so it has multimodal biometrics. A new approach was developed were the colour of the clothes the user is wearing is added to facial recognition. Clothing colour information is added with every different log in to the system, fusing it to the facial recognition every time (this information is permanent). In order to get this system to work in a wide variety of scenarios, three main objectives were defined: As long as the user stays visible, no active re-authentication is needed (Usability); If the user is to walk away from the visible area, then re-authentication is a must (Security); Items used for capturing image must be Commercial off-the-shelf (COTS) items. After system testing, results showed a high tolerance for posture changes, no need for special backgrounds working even with random background changes, making it very viable and accurate.

Analysis of Algorithms for User Authentication using Keystroke Dynamics:
Singh *et al.* [SIKP20] - System based on keyboard dynamics on password input where several different algorithms was tested. Uses three features based on time parameters: dwelling time (interval between pressing a key and releasing it) and flight time (interval between consecutive keypress or release of two immediate keys). A previously created dataset was used for this study. For each of the tested algorithms, hyperparameter tuning was performed. From K-Nearest Neighbours (KNN), C-Support Vector Classification (SVC), Random Forest and XGBoost, the latter got the best precision result, with 93.6%, followed by Random Forest with 87.16%. Confusion matrix and Receiver Operating Characteristic (ROC) metrics for XGBoost were analysed after, where conclusions showed an ROC value of one in almost every user and outstanding confusion matrix values.

Keystroke Dynamics for Continuous Authentication:
Ananya and Singh [AS18] - Creation of a continuous authentication system that does not require user authentication. In order to achieve the aforementioned objective, on initial login, the template is created as a 26 by 26 matrix (*e.g.* aa,..., zy, zz) for mean flight times for each pair of keys the user inputs. Operation of this system is divided in four parts: 1 - user logs in; 2 - immediately after login, the system enters template creation mode where it will stay until a certain number of cells is filled on the matrix; 3 - system asks for user re-login in order to verify user authenticity; 4 - verification and template updating mode, where the system will continue from then on. In verification mode, the user has a score. While inputs are coming, each input is subtracted with the correspond-

ing matrix value and if the absolute value of that calculation is lower or equal to a certain threshold, their score is decremented by a certain factor. If the aforementioned subtraction gives a higher value than the threshold, user scoring is punished with an increment. when user score reaches a certain value, the system mode goes back to 3 and the user must re-log in. Results were promising, specially for a system that requires no pre-registration.

A Comparison of Keystroke Dynamics Techniques for User Authentication:
Anusas-amornkul and Wangsuk [AaW15] - Method that adds keyboard dynamics to password input and tests an array of algorithms/techniques and compares all results. Features extracted are key hold (dwell time), interkey (interval time), up-to-up and down-to-down times. The techniques used are the following: a simple statistic method based on confidence interval, k-means clustering and trajectory dissimilarity (previous work from the author). For data collection, each user typed their username in three sets, and each username was typed ten times, for a total of thirty records per user. The first set is used to create a master profile for each user, while the remaining two were used to measure performance. As the main metric used to evaluate the algorithms, accuracy is the chosen method. This system was developed in C#. After hyperparameter optimisation, results showed that trajectory dissimilarity got the best result with 96.00% accuracy, followed by statistics using confidence interval with 94.42% and then k-means clustering at 87.75%.

Continuous Authentication and Non-repudiation for the Security of Critical Systems:
E. Schiavone *et al.* [SCB16] - The present paper focuses on continuous authentication and non-repudiation. Authors believe that proving a user interacted with the system is useful sometimes, specially if some kind of problem or controversy happens and people try to avoid responsibilities. In cases like this, non-repudiation enables prosecution or accountability by the companies or responsible staff. To test the system, a wide variety of participants will engage in four tasks on the same workstation, running the authentication program in the background. False acceptance rate and false rejection rates were calculated and then efficiency by tracking time since initial user authentication until the session is terminated, as well as time until the system rejects an impostor. An array of different configurations will be tested and evaluated as well. Conclusions taken from this work are ambiguous, since the authors do not give a concrete answer.

2.4 Conclusion

This chapter provides an overview of works related with continuous authentication, behavioural and physiological biometrics. This overview was performed with the objective of gathering a variety of information regarding the topic. After this analysis, a clearer image of how these systems work and how they are used was present, as well as other underlying concepts, like non-repudiation, temporal information and emotions. Emotions are specially important, since studies showed high data variance on same user with differ-

ent emotional states. With the analysis of [SIKP20] and [AaW15], the chosen algorithm to build the system was XGBoost, despite not having information on its performance for mouse dynamics, the results seemed promising and the overall performance is excellent. Table 2.1 presents a all works studied in this chapter. On the following chapter, the requirements, system design and functionality will be explained.

Title	Year	Topic
Continuous Verification Using Multimodal Biometrics	2007	Passive biometrics usage to verify the presence of a logged-in user.
User Identity Verification via Mouse Dynamics	2012	Usage of mouse dynamics to verify a user interacting with a web browser.
Keystroke/Mouse Usage Based Emotion Detection and User Identification	2017	Emotion detection using behavioural biometrics based on keyboard and mouse usage.
Emotions and User Interactions with Keyboard and Mouse	2017	Detection of positive vs negative emotions on users through the use of keyboard and mouse dynamics.
Continuous User Authentication Using Temporal Information	2010	Continuous user verification using facial recognition and clothe recognition.
Analysis of Algorithms for User Authentication using Keystroke Dynamics	2020	Comparison between several algorithms on keystroke dynamics for user authentication.
Keystroke Dynamics for Continuous Authentication	2018	A continuous authentication system that doesn't require initial user authentication (log in).
A Comparison of Keystroke Dynamics Techniques for User Authentication	2015	Analysis of algorithms when applied to keystroke dynamics.
Continuous Authentication and Non-repudiation for the Security of Critical Systems	2016	Creation of a system for continuous user authentication and non-repudiation of performed interactions with the system.

Table 2.1: Summary table of all works studied on the state of the art.

Chapter 3

Requirements and System Design

3.1 Introduction

This chapter explains the method used to develop the system. Here the reader can understand how this system came to be, how to use it and what to expect from it. This chapter is organised as follows:

- Section 3.2 - **System Requirements** - a summary of functional and non-functional requirements;
- Section 3.3 - **System Design and Functionality** - explanation of the system organisation, design and functionalities;
- Section 3.4 - **Conclusion** - conclusions taken from what was accomplished during this chapter.

3.2 System Requirements

Next are the requirements, both functional and non-functional, as well as the development method used during the conception of this system.

3.2.1 Functional Requirements

The project needs to be functional in the long run, so it needs a way to save user information in order to avoid the need to enrol users every time they use the system. There is no restriction for application environments where this system is going to operate, so it will work on the whole computer, despite the application. If the system fails to verify that the actual user is the logged-in user, the computer will lock, and a new system login must be made. The system must strive to offer usability, so a low number (< 10%) of false rejections must be ensured as well as security, with a low number (< 10%) of false acceptances.

User inputs are saved on a Comma-Separated Values (CSV) output file, which will contain the username of said user and timings for the key presses. CSV files are typically used in this area of knowledge, representing simple means to store this kind of values and use as input to machine learning algorithms. Preserving user privacy is of utmost importance, so any information about what keys the user presses is ignored. This was possible by transforming each key into a code, rather than the actual key name. These codes are saved in volatile memory, on a set data structure, and are only used to do the calculations needed

to produce the data the system wants to collect. Given this, the model presented by [AS18] could not be used in this system.

3.2.2 Non-Functional Requirements

Besides the need of immediate results while on the verification step, no limitations were set for the system creation. Python was the programming language chosen to develop this project since it is very versatile and uses relatively low resources. Python has access to very good machine learning libraries and is easy to use.

Project development took into consideration potential future changes, and its implementation favoured parameterization to ease alterations, like parameters, file paths, functions, etc. Since all development and data collection was conducted using only one computer, the same mouse and keyboard were always used as well. After inputting the username, the user can utilise the computer normally, since the program will collect information and process it in the background. Keyboard and mouse are regular hardware that can be found with all computers. In order to use the system, only a standard Universal Serial Bus (USB) keyboard and mouse are necessary.

3.3 System Design and Functionality

This project consists of two steps: enrolment and continuous verification. For the verification to take place, the user must go through the enrolment process in the first time they interact with the system, since their username will not be present. For the time being, the entirety of the system works locally.

As figure 3.1 shows, when a user utilises the system for the first time, a username is collected and a verification of its presence in the system is conducted. If the user is not present, then a process of typing in the keyboard and using the mouse starts. After the data is collected, it is saved in a CSV file. This output file consists only of times, in seconds, and the correspondent username, for the keyboard, and mouse velocities with corresponding username, for mouse usage. The program does not collect nor save any more information besides that. Figure 3.2 illustrates what features are collected from the keyboard. In case the user is present in the system, a verification step takes place, where the user can type freely on the keyboard as well as using the mouse while the system collects the previously mentioned information, sends it to the XGBoost algorithm to evaluate and produce a result saying if the user matches the information, or not.

For the enrolment procedure, the system will create a file, or add to the already existing file, 3000 keyboard inputs and 2000 mouse inputs from the user. These will be transformed into groups of 9 keys and 20 movements which then are added to the main files. With growing datasets, all these values can be changed.

For the verification procedure, the system collects initial information the same way it does for the enrolment. After collecting information for 30 seconds, it is formatted to match the model and sent to be evaluated by the XGBoost algorithm. The accuracy result produced dictates whether the console locks or not. Established accuracy values are 75% for keyboard and 50% for mouse, these values will be explained in chapter 5.2

Data was collected with the same conditions in mind: same computer, keyboard and mouse with unchanged settings.

3.4 Conclusion

Given that the continuous authentication subsystem is to presumably work with granted access to mouse and keyboard events, its design should be built with focus on security, privacy and usability, instead of other constraints. Security and privacy are the most important, since the main objective of this system is to verify if a given user is the actual person utilising the computer. This assumes the computer where the system was implemented has sensitive information that should only be accessed or altered by certain people. Usability guarantees that the person utilising the system has a smooth experience, without having to continually insert passwords or have their thought process and work interrupted. After completing the above chapter, many questions were answered and a better development experience was achieved.

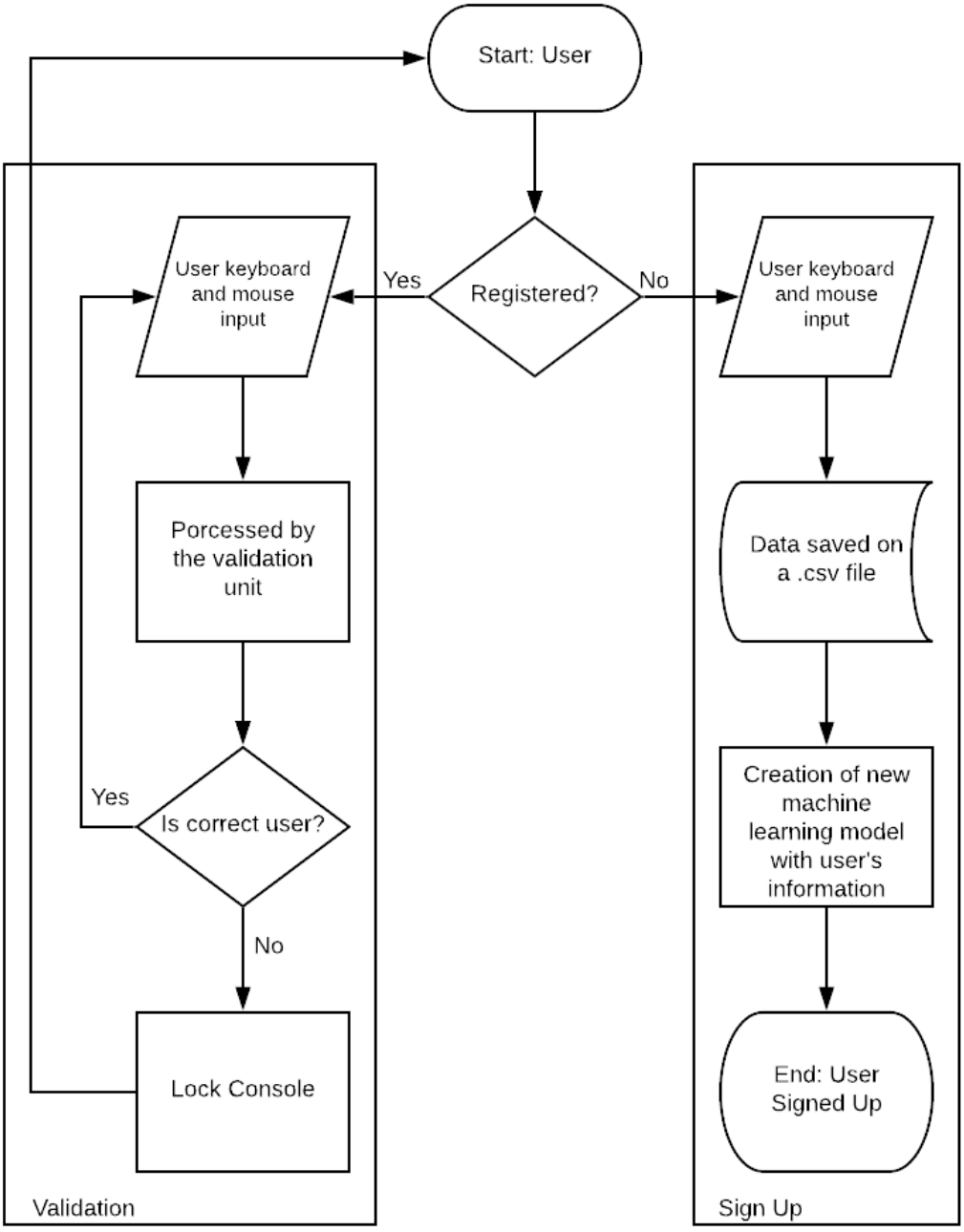

Figure 3.1: Flowchart illustrating the enrollment and verification procedures.

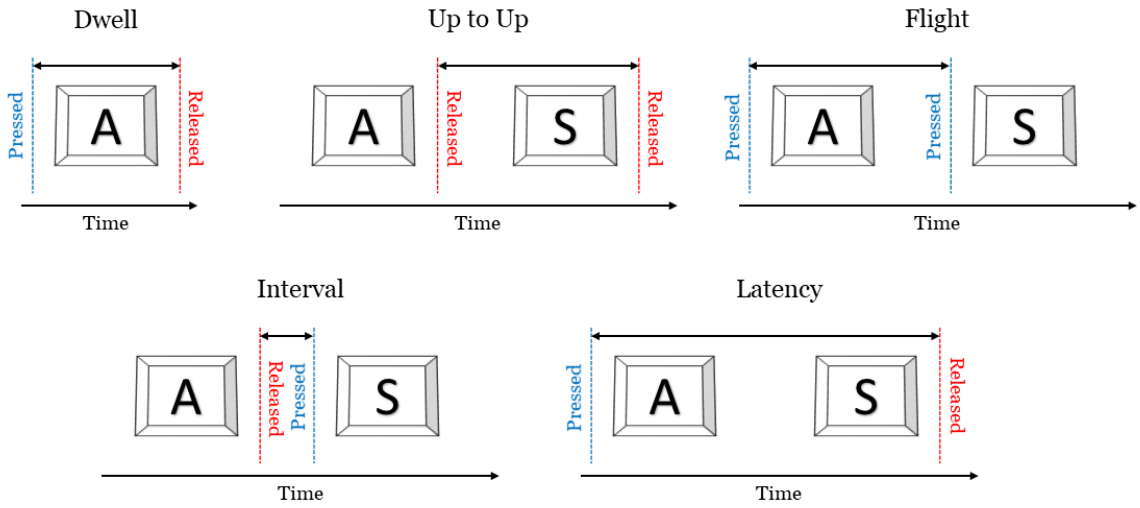

Figure 3.2: Keyboard features visualization based on [SIKP20] and [AaW15].

Chapter 4

System Implementation

4.1 Introduction

The present chapter will dwell on the overall system implementation. There is a review of technologies and libraries used, details about the implementation as well as encountered challenges. Some code samples will be shown too. This chapter is divided as follows:

- Section 4.2 - **Technologies and Libraries** - all technologies and libraries used to create the presented system;
- Section 4.3 - **Implementation Details** - explanation of the system implementation, how it was thought and done;
- Section 4.4 - **Implementation Challenges** - discussion on the challenges encountered while implementing the system;
- Section 4.5 - **Conclusion** - conclusions about all the information exhibited on the present chapter.

4.2 Technologies and Libraries

As previously mentioned, the chosen language for this project was Python. This decision lies on the fact that Python is simple to understand, has a wide variety of libraries focused on machine learning and tools to assist on it, and also due to familiarity with it gained through classes where Python was the main tool for machine learning. PyCharm Integrated Development Environment (IDE) was used for a better experience using Python, since it provides error report and debugging tools. The libraries utilised in the system are:

- `csv` - library used to handle CSV files. it was mostly used to read and write to files, for example, to save user enrolment data, format the files to have the appropriate number of key presses and mouse movements, read that data to build a model, etc.;
- `ctypes` - library that enables the use of C compatible data types and calling several Dynamic-Link Library (DLL). Here it is used to lock the computer utilising `ctypes.windll.user32.LockWorkStation()` and to show a system box telling users that the necessary enrolment data has been collected;
- `datetime` - library that provides date and time manipulation classes. It enables the creation of key timings and the calculations for mouse velocity, making it easier to

collect information. This class was important in the context of this work because its functioning determines how consistent the timings of the events are;

- `itertools` - this library offers a variety of iterators for a wide array of uses. Was used to aid in the creation of the enrolment files, so the collected data has the same amounts of information for each class;
- `math` - module that implements a diverse range of mathematical functions present in C. Used to calculate mouse velocities;
- `NumPy` - library for scientific computing that provides utilities for multidimensional array manipulation. Usage in the system is to create the `bestPredictions` array to save the best predictions for each class after learning step and to calculate true positive, false positive, true negative and false negative values from the matrix.
- `os` - module that provides a way to use system dependent functionalities, like getting the operating system name. For this system, it is used to verify the existence of the initial file path in the computer;
- `pandas` - package primarily used for data science and data analysis. Throughout all machine learning implementation, pandas is used to create and edit the necessary dataframes for XGBoost to use on learning and prediction jobs;
- `scikit-learn` - library that supplies an extended set of machine learning tools, from algorithms to metrics. The main usage in this system was to create metrics using, for example, `metrics.classification_report`. Another use for this library comes in the form of creating the train and test sets using `metrics.train_test_split`;
- `time` - module that, similarly to `datetime`, provides means to use time related functions. In this case, `time.sleep()` was used to wait for the verification step to collect data before sending it to be evaluated;
- `XGBoost` - library that offers optimised distributed gradient boosting. This library was built to be highly flexible, efficient and portable. XGBoost was the chosen algorithm for this work after reading [SIKP20] and comparing it also to [AaW15] and the results showed it was a good choice, specially for keyboard dynamics. This algorithm creates a model, which is a mathematical structure that takes an observation and produces a result. The model is a decision tree, whose training is done by boosting. Boosting operates while taking in consideration the errors produced by previous trees [xgb].

4.3 Implementation Details

Before starting to implement the system, the best way of proceeding was devised: First and foremost, a general system flow, like the one present in figure 3.1; Then a data collection and enrolment subsystem are needed to build the dataset; After the dataset creation,

use it in the machine learning algorithm to create a model; When satisfactory results are achieved, implement the verification subsystem.

4.3.1 Enrolment

Enrolment is the first step to start using the system. If a provided user does not exist in the system, enrolment ensures the user is created and that data is collected to create a new collective model for the system to recognise the users. When a user is added to the dataset, a new class is created. All the initial verification and creation of the user is made using the `signup()` function. A snippet of it can be seen in listing 4.1.

```
if os.path.exists(keyboardPath):
    with open(keyboardPath, 'a+') as checkFile:
        checkFile.seek(0)
        csvReader = csv.reader(checkFile)
        while True:
            tempUser = ''.join(tempUser.split()).lower()
            for row in csvReader:
                if tempUser in row[0]:
                    print("User already exists, "
                          "please type 1 if you pretend to add info or 2 "
                          "if this is not your username:")
```

Listing 4.1: User presence verification on the dataset.

To collect keyboard inputs, the `pynput` library was used. Natively it does not maintain a list with the continuously pressed keys, so in order to register continuously pressed keys as only one press, the `set()` datatype is used. Sets are unordered, unchangeable and do not allow duplicates. When a key is pressed, its code is added to the set and continues there until that same key is released. Then a dictionary is created with that key code and the time it was pressed, and another for when the key is released. By using this mechanism, all features can be extracted. In listing 4.2, after the key is released, up-to-up time and dwell time are calculated. For all the other keyboard features, the procedure is equivalent.

```
if unifiedKeyCode in continuousPressSet:
    keysReleaseTime[unifiedKeyCode] = releaseTime
    # Calculate up-to-up time. Only occurs after the second key release.
    # Time between previous key release and current key release.
    if releaseFirstTime:
        up2UpTime = releaseTime - releaseFirstTime
        up2UpTimeArray.append(up2UpTime.total_seconds())
        releasePreviousTime = releaseFirstTime

    # Calculate dwell time. This is the time a single key was pressed.
    if unifiedKeyCode in keysPressTime:
```

```
12          totalPressTime = keysReleaseTime[unifiedKeyCode] - keysPressTime[
                unifiedKeyCode]
13          dwellTimeArray.append(totalPressTime.total_seconds())
```

Listing 4.2: Up-to-up and dwell time calculation.

For mouse data collection, initial position is saved and every time it leaves a 100 pixel radius, x and y velocity is calculated, as well as overall pixel to pixel velocity. Initial position is refreshed to the final position and the procedure repeats itself.

When 3000 key features or 2000 mouse features are collected, a window appears on screen to let users know they have the respective component necessary information. When all information is collected, the user presses the `ESC` key and all the gathered information is written to a corresponding CSV file. If the necessary information is not collected upon the `Esc`. key being pressed, a prompt asking for confirmation that the user intends to leave appears. If the user wants to leave, all gathered data is deleted. Otherwise, data collection continues. After a successful data gathering, the created file is formatted to have sets of nine keys per row. Finally, this file is appended to the `compilation.csv` file, where all users are present. Mouse procedure is exactly the same, but moves are grouped in 20 instead of nine. Reasoning and testing for this conclusion is presented in chapter 5.

Before the final step, hyperparameter tuning on the XGBoost algorithm, using cross validation, took place. Hyperparameter tuning is a process that helps with algorithm performance by finding the most suitable hyperparameters to produce the best results possible. This procedure was done using cross validation (also called k-fold due to the k value dictating how the data is divided), which takes data samples from the whole dataset and tests them with each set of values given in the gridsearch dictionary. In the end, the hyperparameter set that produced the best outcome is outputted. The keyboard dynamics algorithm was the first to be tuned. The chosen approach to fine tune the hyperparameters was to create sets of two hyperparameters at a time and try to fine-tune them separately. Initial hyperparameters were set based on [SIKP20] and can be seen in table 4.1. These hyperparameters are:

- `n_estimators` - number of gradient boosted trees. This value is not used on the final hyperparameters, since it was not being used by the algorithm;
- `max_depth` - maximum depth of a tree. Higher values contribute to increase its complexity, so lower values avoid overfitting. Higher values also make for an aggressive memory consumption;
- `eta` - learning rate for the algorithm. Lower values avoid overfitting since the step sizes are smaller;
- `colsample_bytree` - ratio of columns used when build each of the trees. Smaller values avoid overfitting;

- `subsample` - ratio of training instances. In table 4.1 this value is 0.7, which means that the algorithm will utilise 70% of the training data before growing a new tree. Lower ratios avoid overfitting;
- `min_child_weight` - minimum hessian sum of instance weights. If the hessian sum is lower than `min_child_weight`, further construction in that node is abandoned. Higher values avoid overfitting;
- `reg_alpha` - L1 regularisation term for weights. This regularisation tries to estimate data median value. A higher value avoids overfitting;
- `reg_lambda` - L2 regularisation term for weights. This regularisation tries to estimate data mean value. Higher values avoid overfitting;
- `gamma` - minimum loss reduction for a node to split into two more nodes. Larger values avoid overfitting.

Adding to the above hyperparameters, there are three more that are static for both keyboard and mouse:

- `'objective': 'multi:softprob'` - this hyperparameter sets the algorithm objective to return a vector containing, for data point of each class, the predicted probability;
- `'num_class': 6` - because of the use of `multi:softprob` as the objective, the number of classes to predict has to be specified. For this system, that number is six, corresponding to the number of people that helped testing the prototype: `fernando`, `joana`, `laura`, `mariana`, `pedro` e `tiago`.
- `'eval_metric': 'mlogloss'` - logistic loss for multiclass problems. It is calculated with the following formula:

$$L_{log}(y, p) = -(y \log(p) + (1 - y) \log(1 - p)$$

  where $y \in \{0, 1\}$ and the probability estimate $p = Pr(y = 1)$. The lower the `mlogloss` value, the better, since it is based on the likelihood function, which dictates how likely the model is to correctly fit into the given data.

The created sets of hyperparameters to tune were chosen based on the relation between their functions and were the following:

1. `max_depth and min_child_weight` - the presented hyperparameters are connected to each other, since `max_depth` dictates the maximum depth a tree can reach and `min_child_weight` dictates if a node should be abandoned or not;
2. `eta and min_split_loss` - eta has a very important role, since it defines the learning rate of the algorithm. This parameter is put together with `min_child_weight` because the latter is also connected with the previous two when constructing new trees;

3. `colsample_bytree` and `subsample` - both hyperparameters manage how the dataset information is divided during algorithm usage;

4. `reg_alpha and reg_lambda` - are both regularisation hyperparameters to tune the weights of the model.

Values tested on tuning are discussed in chapter 5.2.

hyperparameter	values
n_estimators	450
max_depth	8
eta	0.2
colsample_bytree	0.8
subsample	0.7
min_child_weight	3
reg_alpha	0
reg_lambda	1.5
gamma	0.2

Table 4.1: Initial XGBoost hyperparameters as presented in [SIKP20].

Data splitting was seeded in order to produce more viable results. A random number was selected between zero and 100, the number 42 was picked. Other two important static hyperparameters are the number of training iterations, which was set to 1001, and the early stopping rounds, which was set to 15. This means the algorithm will make a maximum of 1000 iterations, but if `mlogloss` does not go lower for 15 consecutive iterations, it stops earlier. Hyperparameter tuning was computationally costly and time consuming, but as later can be seen, on chapter 5, there was a significant performance boost, both in accuracy and `mlogloss`. Final keyboard hyperparameters are shown in table 4.2.

hyperparameter	values
eta	0.1
max_depth	4
colsample_bytree	0.8
subsample	0.7
min_child_weight	1
reg_alpha	0
reg_lambda	0.25
gamma	0.1

Table 4.2: Tuned hyperparameters for keyboard dataset.

Below, final mouse hyperparameters can be seen, in table 4.3.

hyperparameter	values
eta	0.06
max_depth	4
colsample_bytree	0.525
subsample	0.7
min_child_weight	3
reg_alpha	0.75
reg_lambda	0.25
gamma	0.25

Table 4.3: Tuned hyperparameters for mouse dataset.

After all hyperparameters are tuned, the algorithms can create the keyboard and mouse models to be used to verify the users. After its creation, the model is saved to be used for future verification, in the path `keyboardModelPath` with the name `keyboardModel.model`. This ensures a more usable system, since it does not need to create a new model with every log in, using the already created model. In order to separate the hyperparameters from the algorithm, a dictionary was created called `tunedParamsKeyboard` where the algorithm will read that information. For mouse model creation, the steps are the same: the algorithm creates a model file called `mouseModel.model` in the path `mouseModelPath` and the hyperparameter dictionary is called `tunedMouseParams`.

4.3.2 Verification

After enrolment takes place, the user is fully registered into the system and ready to start the verification step which from the user side is quite simple, since they only need to use the computer as normal, but from the back-end side many operations are taking place. While users type and use the mouse, information produced is stored in memory the same way it is stored during enrolment. After 30 seconds of data gathering, two outcomes can happen: first, if not enough data was gathered, the computer locks, since a minimun quantity of data is expected to be collected; the second outcome is the collected data is sent to be formatted and verified. Here another two outcomes can happen: either keyboard accuracy is 75% and above or mouse accuracy is 50% and above, which lets the user continue their connection; or both keyboard and mouse do not reach the accuracy threshold and the system locks the computer. Previous explanation is illustrated in figure 4.1.
At every 30 seconds, the gathered keyboard information is sent to a function called `sendKeyboardInfo()` which is depicted in listing 4.3.

```
1 # Function to send keyboard formatted continuous information to be verified.
2 def sendKeyboardInfo():
3     global dwellTimeArray, flightTimeArray, up2UpTimeArray, finalKeyboardList
4
5     # Firstly format the raw info sent from verification.py. This alters
        finalKeyboardList. If not empty, the formatted information is sent to
        get verified.
```

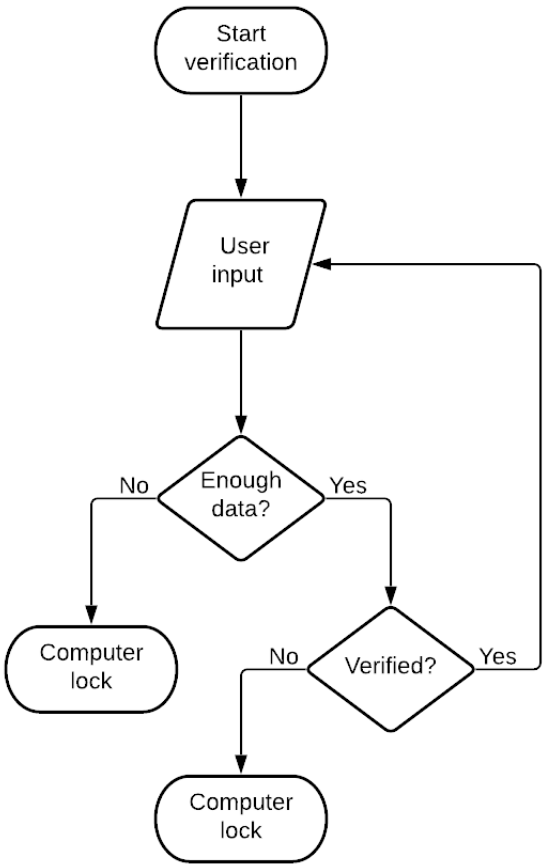

Figure 4.1: Explanation of verification decision making.

```
6     formatKeyboardInfo()
7     if finalKeyboardList:
8         algorithms.predictKeyboardXGBoost(finalKeyboardList)
9         finalKeyboardList = []
10        return algorithms.keyboardAccuracy
11    else:
12        return 0
```

Listing 4.3: Function that sends keyboard data to be formatted and verified.

Function `formatKeyboardInfo()` takes the raw input data, evaluates if it is enough to be verified and transforms it from a $(6 * n)$ array into a $(46 * n)$ array. If data is not enough, returns an empty `finalKeyboardList`.

When the formatted data is fed to `predictKeyboardXGBoost()`, the algorithm will take the data, make a prediction, and return an accuracy value, which will decide what happens to the system. The prediction process is shown in listing A.1 and decision making is presented in listing 4.4.

```
# Function to evaluate both keyboard and mouse.
def predict():
    global finalMouseList, finalKeyboardList
    keyboardAccuracy = sendKeyboardInfo()
    mouseAccuracy = sendMouseInfo()

    # Lock computer if both metrics are below the threshold. This values can be
        changed by an admin.
    if keyboardAccuracy >= 0.75 or mouseAccuracy >= 0.50:
        print("-------------VERIFICATION PASSED-------------")
    else:
        print("----------------BLOCK CONSOLE----------------")
        ctypes.windll.user32.LockWorkStation()
```

Listing 4.4: Function for decision making after prediction results.

As of enrolment procedure, mouse logic is exactly the same as keyboard with some different parameters. Verification happens on the background, so this system can be utilised on a wide variety of environments, like writing emails, messaging or browsing online.

4.4 Implementation Challenges

Implementation challenges encountered while developing the system will be discussed in this section. The first one was the lack of suitable keyboard datasets for the presented problem. Most of them were based on repeated password input, like [SIKP20], or simply not provided. Given this, creation of a suitable dataset was a need. For this to take place, a more close contact with people was sometimes necessary, but was made difficult by the COVID-19 pandemic, so the final datasets are composed of data of only six users, as shown in chapter 5.1. Adjacent to the creation of the datasets is the managing of all consequent files, how to use them correctly and in the correct order.

To use the `pynput` library properly, a way to save continuously pressed keys had to be created. This issue was solved using the `set()` data type, as explained beforehand. A very important aspect adjacent to the previous challenge is how fast the algorithm can process all this information on the verification step, since this step was to be performed regularly (e.g. at each 30 seconds in the implemented prototype). While using XGBoost, with its renowned speed, this issue was immediately solved, since during testing no problems of this nature were detected.

The biggest challenge was to utilise XGBoost properly. It can perform a variety of tasks and every task has a large quantity of parameters and hyperparameters as well as different ways to use. In order to facilitate its usage, a lot of effort was put into learning how and when to utilise it.

4.5 Conclusion

While implementation took place, challenges arouse and were dealt with. This made it necessary to adapt the work plan to suit and solve the problems found along the way. Although the system prototype was finished, many more features could be added to make the system better. This will be discussed in chapter 6.2. Final implementation is satisfactory, with all planned features implemented and working well together.

Chapter 5

Results and Discussion

This next chapter is used to present the necessary material to implement and utilise the system, all tests conducted with corresponding results, a discussion of the results and a final conclusion. Chapter organisation is as follows:

- Section 5.1 - **Materials** - presentation of all means used to accomplish the system, including hardware, datasets, software, etc.;
- Section 5.2 - **Test and Results** - presentation of all tests made to tune and evaluate the system during and after completion;
- Section 5.3 - **Discussion** - in-depth discussion of the results produced;
- Section 5.4 - **Conclusion** - verdict on all accomplishments made while building present chapter.

5.1 Materials

Since the main objective for this dissertation is the construction of a system prototype and testing of machine learning algorithms, using the same computer and external hardware throughout all stages of development is essential to prevent data irregularity issues. By doing this, data homogeneity is guaranteed to every user. The computer used is a laptop. Its specifications are Intel® Core™ i7-5700HQ @ 2.70GHz, 16 GB Double Data Rate 3 (DDR3) Random Access Memory (RAM) @ 1600 MHz, Solid State Drive (SSD) 120 GB with Windows 10 Pro 64 bit. No external keyboard was used. Mouse had 1250 Counts Per Inch (CPI), and Windows sensitivity set to 10 with no mouse acceleration. The computer was connected to the charger at all times, since it was found during the initial experiments that it might influence data granularity, both in enrolment and verification steps, which may then lead to incorrect results. Though this should not be a general problem, one must not forget that the environment where the authentication system may run in the future might not guarantee this particular aspect at all times. This will need to be further studied in the future.

Software/tools used to develop the system were: PyCharm IDE to create all of the python code, Microsoft Excel to manage all the CSV files in an initial state, Git [git] to save all work progress, Human Benchmark [ben] to assist in mouse data collection and TypeLit [lit] to aid with keyboard data collection. Human Benchmark [ben] is a website that provides a series of tests the users can complete by using almost only the mouse. TypeLit [lit] is a website to train typing, by letting users rewrite books in different languages, such as

"Alice in Wonderland" or "Os Maias".

When the enrolment procedure ends, the biometric data is saved into two CSV files, as can be seen in table 5.1 and table 5.2. Afterwards, two more CSV files are created with the keyboard having rows with nine key presses and the mouse having 20 movements. This is achieved by repeating the nine or 20 consecutive rows from the same file. First row from the new CSV file would be rows one to nine from the original, the second row would be rows two to 10, and so on. Same process is applied to the mouse file. This method creates a notion of pattern, since it is impossible to verify a user with one key press or one mouse movement.

label	dwellTime	U2UTime	flightTime	intervalTime	latencyTime
pedro	0.21499	0.001988	0.105836	0.21499	0.087128
pedro	0.111142	0.156264	0.196282	0.216978	0.235004
pedro	0.071124	0.220698	0.149864	0.267406	0.221689
pedro	0.141958	0.144151	0.142949	0.291822	0.233086
pedro	0.14316	0.158583	0.232095	0.286109	0.31394
pedro	0.069648	0.191259	0.225005	0.301743	0.33143

Table 5.1: Snippet of the initial keyboard dataset, where each row corresponds to a key and its relation with the next.

label	xVelocity	yVelocity	mouseVelocity
pedro	2758.244	1485.209	3132.691605
pedro	4888.346	2777.469	5622.300157
pedro	6599.56	2133.191	6935.754965
pedro	406.509	56.91126	410.4734793
pedro	155.6227	244.9617	290.2148811
pedro	3.124993	78.12482	78.18729177

Table 5.2: Snippet of the initial mouse dataset, where each row saves mouse velocity for 100 pixels.

After this process is complete, new formatted files are created and saved to the corresponding compilation file, which are used to train the XGBoost models. In the end, input files are:

- two initial files with direct input from the user;
- two formatted files, one with nine key presses, one with 20 mouse movements;
- two files with data regarding all users in the system;
- two XGBoost models, one for mouse, one for keyboard.

5.2 Tests and Results

Present section shows all tests and results obtained while developing this system.

5.2.1 Datasets

To progress in the system development, after compiling the initial datasets, a series of compilation files were also created. For keyboard data, files for one to 10 key presses were created and tested. Tests conducted this way had no seed on the splitting process. Each file was run 10 times and statistics were calculated to build the charts discussed herein. Figure 5.1 shows a chart for each set accuracy, figure 5.2 the corresponding `standard deviation`, figure 5.3 a chart for `mlogloss` values with `standard deviation` values being presented in figure 5.4.

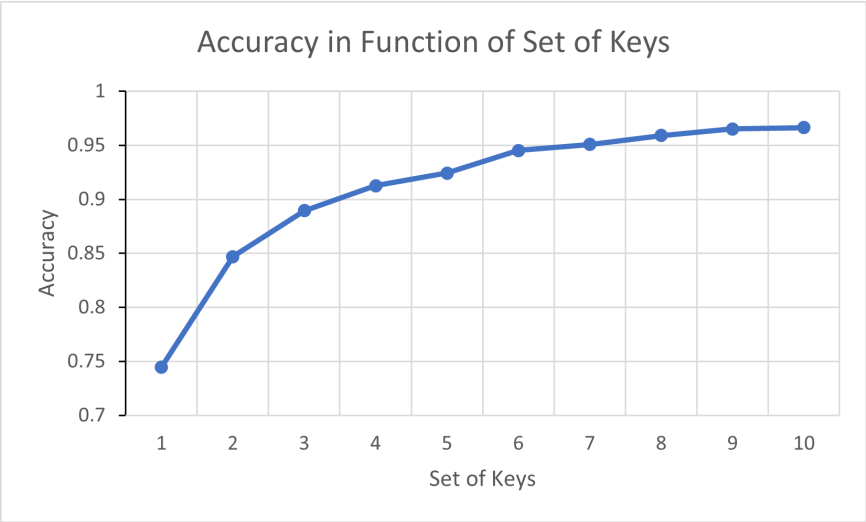

Figure 5.1: Accuracy in function of set of keys.

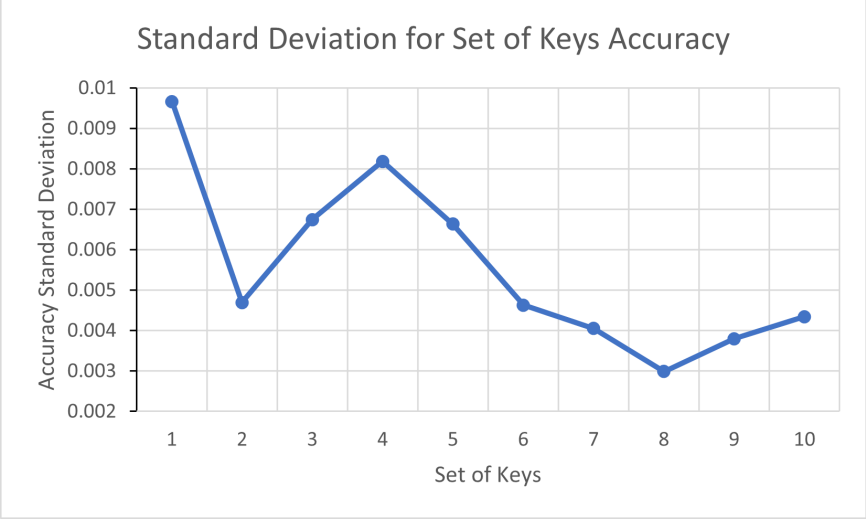

Figure 5.2: Standard deviation for set of keys accuracy.

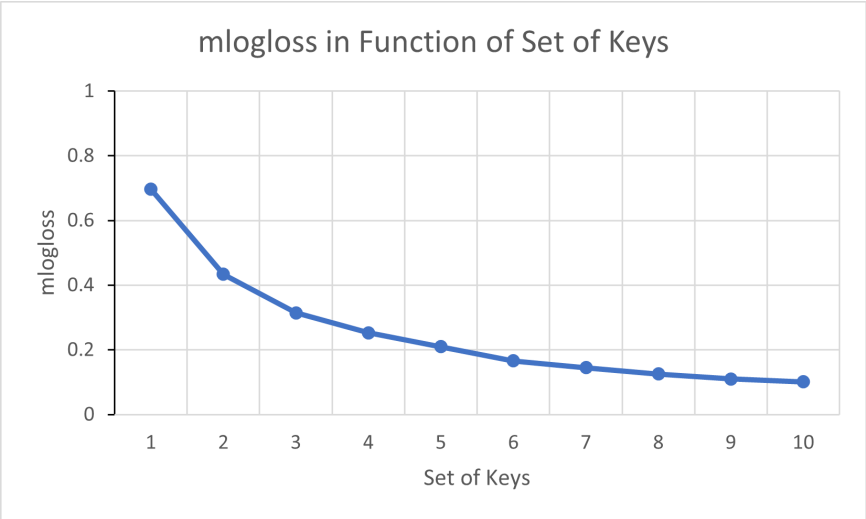

Figure 5.3: Mlogloss in function of set of keys.

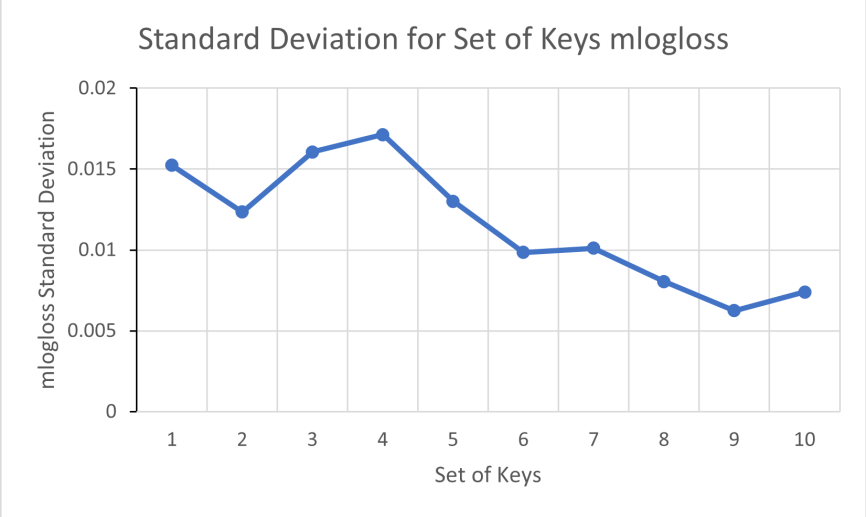

Figure 5.4: Standard deviation for set of keys mlogloss.

Upon analysing the information above, it was concluded that a set of nine keys exhibited a good compromise in terms of accuracy, since accuracy was starting to climb slower and `standard deviation` value was almost insignificant, but specially because `mlogloss` value was really good and almost the same as the 10 key set. This is due to the algorithm being able to evaluate a wider case. With each additional key in the set, the algorithm had an increasingly better notion of continuity. In conclusion, using a single key press to do the verification is almost impossible, while a set of 10 key presses is much more capable. There was still room to try bigger sets since the trend shows that using more keys leads to better accuracy, but unfortunately, due to the increasing amount of time each iteration was taking, it was not feasible.

For mouse usage, the same guidelines were followed. In figure 5.5 mouse accuracy can be found and in figure 5.6 its `standard deviation`. For `mlogloss` and matching `standard`

`deviation` there are figures 5.7 and 5.8, respectively.

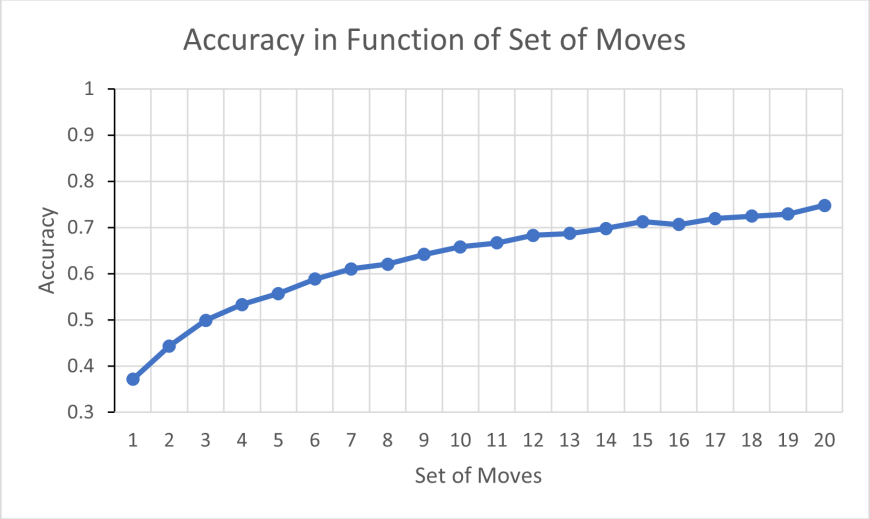

Figure 5.5: Accuracy in function of set of moves.

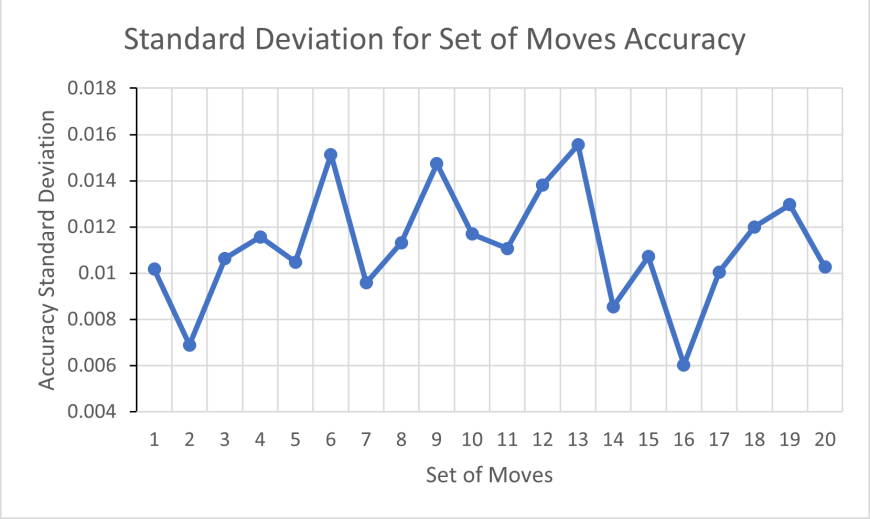

Figure 5.6: Standard deviation for set of moves accuracy.

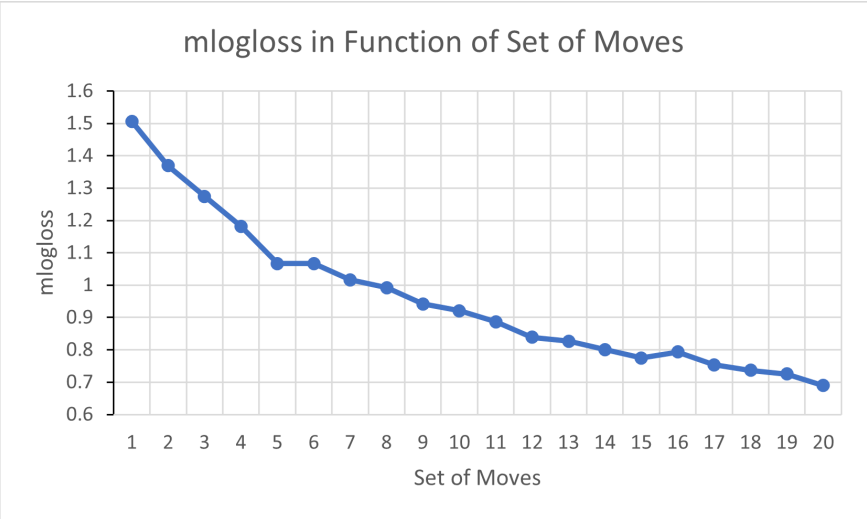

Figure 5.7: Mlogloss in function of set of moves.

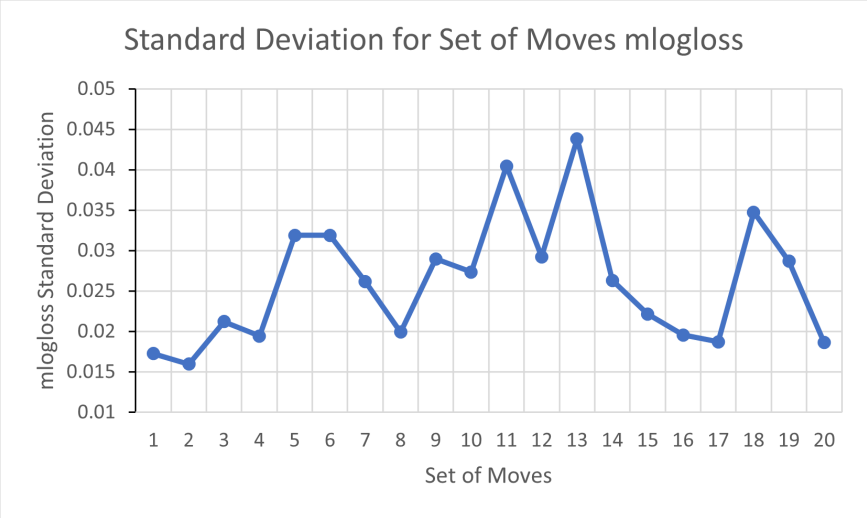

Figure 5.8: Standard deviation for set of moves mlogloss.

Conclusions on mouse datasets were harder to take, since the dataset was smaller, not only in number of features but, also in data quantity. This is due to the data collection for the mouse being harder to do than it was initially predicted because users did not use the mouse as extensively as the keyboard. It shows that the bigger the move set, higher was the accuracy and lower the `mlogloss`, both with residual `standard deviations`. Without hyperparameter tuning, figure 5.9 and figure 5.10 show ROC and Area Under Curve (AUC) for keyboard and mouse, respectively, with mouse having worse results than keyboard. ROC represents the correlation between sensitivity (true positive rate) and specificity (1 - false positive rate), the closer to one, the better. This classifier is binary, so in order to successfully apply it to the problem at hands, a one-vs-all approach was taken, where each class is compared with the rest. AUC represents how well the algorithm can distinguish between classes.

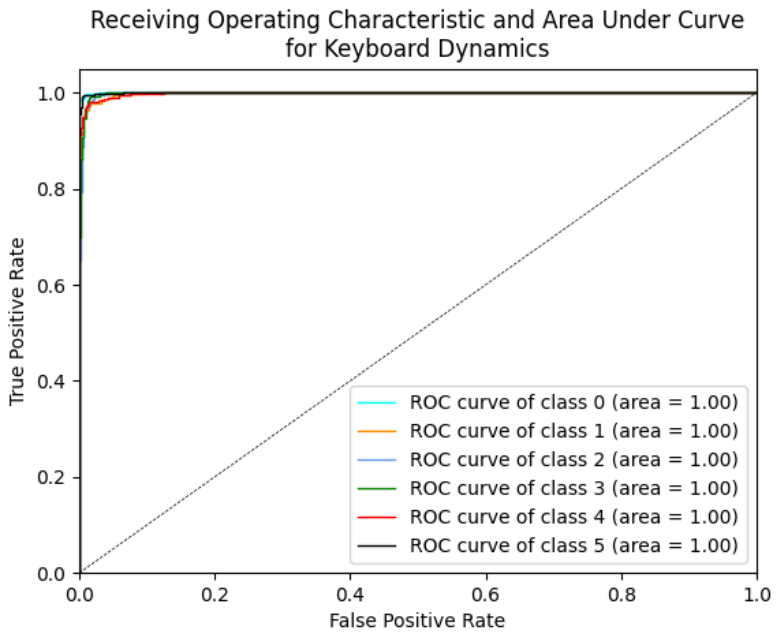

Figure 5.9: ROC and AUC for keyboard before hyperparameter tuning. Each class represents a user.

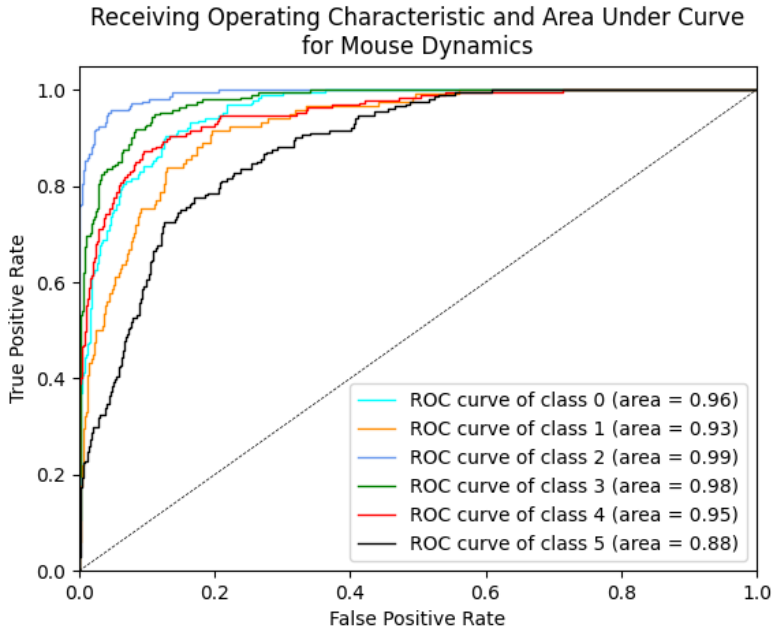

Figure 5.10: ROC and AUC for mouse before hyperparameter tuning. Each class represents a user.

5.2.2 Hyperparameter Tuning

This development step was already mentioned in chapter 4.3, where initial hyperparameters were shown in table 4.1, final values for keyboard on table 4.2 and for mouse on table 4.3. Even though values were mentioned, how those conclusions were reached was not, so this section will show the process behind them. Utilising XGBoost cross validation, it was possible to test several values on specific hyperparameters and get the best `mlogloss` result for each. Here the splitting process is seeded. The same value sets were

applied to both datasets, as seen in table 5.3.

Hyperparameters	Values Used
max_depth	[1, 2, 3, 4, 5, 6, 7, 8,9, 10]
min_child_weight	[1, 2, 3, 4, 5, 6, 7, 8,9, 10]
min_split_loss	[0.1, 0.25, 0.5, 0.75, 1]
eta	[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
colsample_bytree	[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
subsample	[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
reg_alpha	[0, 0.25, 0.5, 0.75, 1]
reg_lambda	[0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2]

Table 5.3: Values used on cross validation for both keyboard and mouse datasets.

Given that the mouse dataset is less representative than the keyboard one, another test on the hyperparameter `eta` was made. The new test set was [0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1]. Final hyperparameter values are available, as mentioned beforehand, in table 4.2 and table 4.3.

Before the tuning process, `mlogloss` values were 0.100 for keyboard and 0.693 for mouse. After the process, keyboard `mlogloss` was 0.085 and for the mouse was 0.685. Improvements can be seen when analysing ROC and AUC for keyboard and mouse, on figure 5.11 and figure 5.12, respectively.

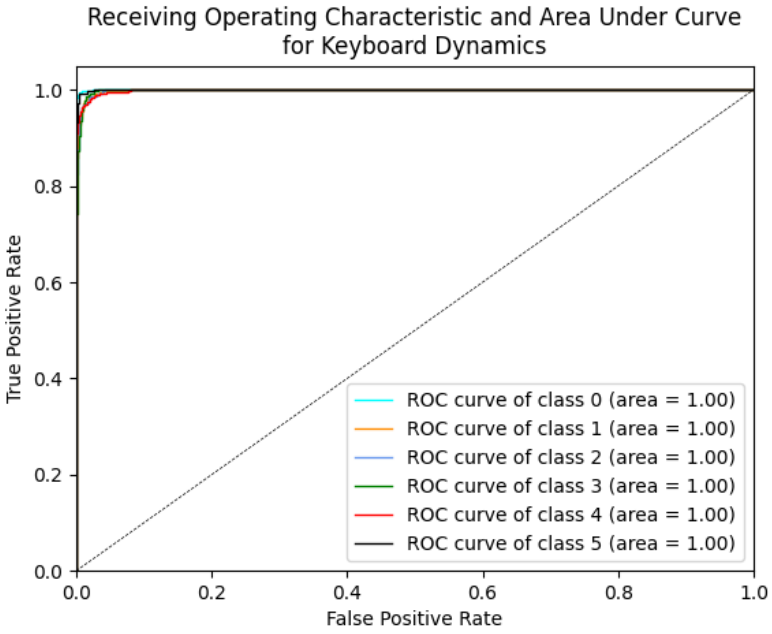

Figure 5.11: ROC and AUC for keyboard after hyperparameter tuning.

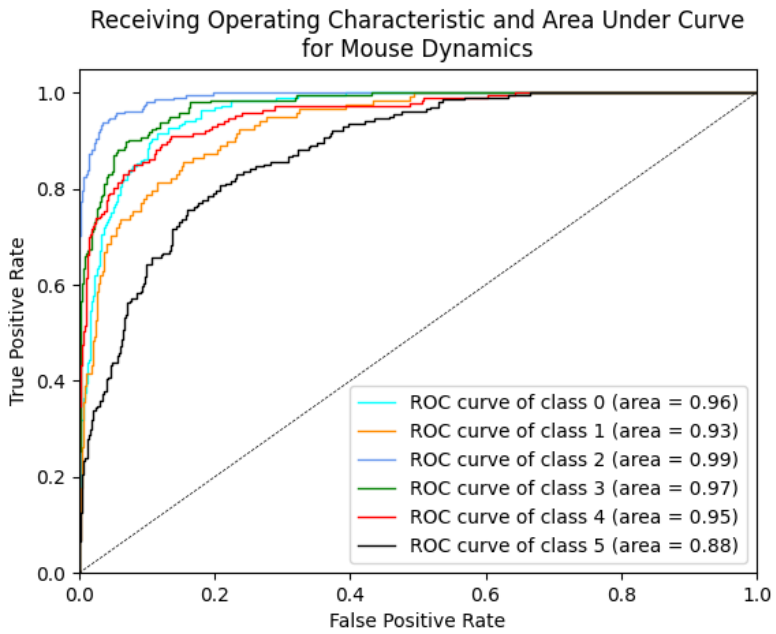

Figure 5.12: ROC and AUC for mouse after hyperparameter tuning.

5.2.3 Confusion Matrix Analysis

With hyperparameter tuning complete, analysing confusion matrices was the next step. This test acts as a validation test in order to prove the system got more viable and efficient. On table 5.4, confusion matrix for keyboard dataset is shown, where class four presents the highest confusion, and class zero the lowest. Table 5.5 shows confusion matrix for mouse before parameter tuning.

	0	1	2	3	4	5
0	326	0	3	0	0	0
1	0	313	15	0	1	0
2	1	6	320	2	0	0
3	0	0	2	323	5	0
4	0	0	0	20	310	0
5	1	9	2	0	1	316

Table 5.4: Confusion matrix for keyboard dataset before hyperparameter tuning.

	0	1	2	3	4	5
0	154	7	0	15	2	25
1	13	64	3	2	25	25
2	1	0	184	6	6	7
3	17	0	6	155	4	7
4	9	3	2	6	138	13
5	31	16	11	11	15	117

Table 5.5: Confusion matrix for mouse dataset before hyperparameter tuning.

After tuning process, confusion matrix for keyboard is present in table 5.6 and for mouse in table 5.7.

	0	1	2	3	4	5
0	326	0	3	0	0	0
1	0	313	15	0	1	0
2	0	3	323	3	0	0
3	0	0	1	322	7	0
4	0	0	0	21	307	2
5	1	7	0	0	0	321

Table 5.6: Confusion matrix for keyboard dataset after hyperparameter tuning.

	0	1	2	3	4	5
0	152	5	0	17	1	28
1	9	71	1	1	19	31
2	3	0	183	4	5	9
3	19	1	7	152	3	7
4	7	3	2	7	134	18
5	29	12	13	9	19	119

Table 5.7: Confusion matrix for mouse dataset after hyperparameter tuning.

For a better comparison of these matrices, a group of tables comparing true positive rate and false positive rate for both keyboard and mouse was created. On table 5.8 mean value for true positive rate is higher after hyperparameter tuning, while false positive rate is lower. This means that all efforts to make the system better got positive results with the keyboard dataset.

	True Positive Rate		False Positive Rate	
Class	Before	After	Before	After
0	0.9909	0.9909	0.0012	0.0006
1	0.9514	0.9514	0.0091	0.0061
2	0.9726	0.9818	0.0134	0.0115
3	0.9788	0.9758	0.0134	0.0146
4	0.9394	0.9303	0.0043	0.0049
5	0.9605	0.9757	0	0.0012
Mean	0.9656	0.9677	0.0069	0.0065

Table 5.8: Comparison of true positive rates and false positive rates on keyboard dataset before and after hyperparameter tuning.

Table 5.9 presents true positive rates and false positive rates for the mouse dataset. There, the mean value for the true positives increased slightly, which is good. On the other hand, the false positive mean value was better before the hyperparameters were tuned.

	True Positive Rate		False Positive Rate	
Class	Before	After	Before	After
0	0.7586	0.7488	0.0792	0.0747
1	0.4848	0.5379	0.0269	0.0217
2	0.9020	0.8971	0.0246	0.0257
3	0.8201	0.8042	0.0439	0.0417
4	0.8070	0.7836	0.0560	0.0506
5	0.5821	0.5920	0.0857	0.1034
Mean	0.7258	0.7273	0.0527	0.0530

Table 5.9: Comparison of true positive rates and false positive rates on mouse dataset before and after hyperparameter tuning.

5.2.4 Verification

The testing verification step consisted of the following:

- For 10 minutes straight, a user interacts with the system as they would normally do. There are two verifications per minute (at each 30 seconds);
- If during these 10 minutes more than one system block happens, the current session stops and a new test doing the TypeLit and/or Human Benchmark will take place, in order to verify if the activities made during enrolment procedure influence the models for the worse;
- Another test will be conducted where a user will be logged in, and a different user will try to pretend to be the initial one while using the system normally;
- In case the intruder is detected at least two consecutive times, the test is considered a success, otherwise, a test utilising TypeLit and/or Human Benchmark will be made.

All tests presented next were conducted by user `pedro`. First modality to be tested is keyboard and then mouse.

After logging in as himself, user `pedro` started to use the computer as normal, writing an email, parts of the dissertation or engaging in online conversation. Throughout the 10 minutes the computer was used, and every verification check was passed with success. In total there were 156 false negatives and a significant 2848 true positives, which represents 94.81% accuracy. For the impersonation test, the user of the system was logged in as `mariana` and after 2 verifications, the system had already rejected the user behind the keyboard twice, with 258 true negatives and eight false positives, which means 96.99% accuracy on the rejections.

When testing the mouse, after two verifications the system blocked twice with 70 true positives and 267 false negatives, translating to 20.77% accuracy. As seen before in chapter 5.2, results produced during mouse training were not the best, and logically they influence verification results a lot. So the next test will be a repetition of the exercises utilised to gather data on Human Benchmark. After 10 minutes of system usage on Human Benchmark, all verifications were accepted with 2291 true positives and 440 false negatives, having 83.89% accuracy. Now on the impersonation test, with two verifications the system already passed the test, the console got blocked twice. The system detected three false positives and 366 true negatives, giving 99.19% rejection accuracy.

5.3 Discussion

After analysing all results produced during the making of the system, it was concluded that changes made were very positive, despite some of them only enhancing very little. The biggest disparity registered was during key set and move set size testing, with keyboard accuracy going up more than 20% and mouse more than 30%. Values for `mlogloss` had significant drops too, with keyboard going from roughly 0.7 to 0.1 and mouse values from 1.5 to 0.7, since `mlogloss` is the most important metric for probability problems, these results are the most significant ones.

When talking about AUC-ROC, much better results can be seen on keyboard ROC values on figure 5.11, where classes 1 and 4 have the more significant changes, despite AUC being the same. For mouse, early curve steepness is better after tuning, as figure 5.12 shows when compared with 5.10, even though AUC values are mostly the same, but class 3 had a loss of 0.01 in that metric.

Verification testing showed that keyboard model is much more versatile and useful than the mouse model since it can continuously verify the user while performing completely normal tasks with almost 95% accuracy, while the latter can only do so when evaluating information gathered from usage in Human Benchmark. When looking at the confusion

matrix on table 5.4, it is possible to see that class four is confused only with class three, like class one is confused with class two, a repeating pattern even after hyperparameter tuning, shown in table 5.6. This event may happen due to the writing patterns of the evaluated user converging into another one in specific occasions, or even due to mood or posture changes. Despite its flaws, the system can detect both an actual user and an impersonating one with ease.

With this discussion, it is possible to see that mouse dataset has much different results than keyboard dataset. This may be due to a big chunk of early development being aimed at having a functional keyboard verifier and only then focusing on mouse features. Another aspect is that the mouse dataset is lacking in terms of feature diversity (3 versus 5) and with significantly fewer amounts of data (10996 versus 19753). One last aspect that may be of relevance is that, as studied in [SIKP20], XGBoost was tested on keyboard datasets, with great success, but no information on mouse datasets was found.

5.4 Conclusion

This chapter compiled all tests made during system development, which helped with evaluating the work made and what changes should take place to optimize the system. In the end, keyboard results were very satisfactory, both in user verification and impersonation detection, while doing completely normal activities. Meanwhile, mouse results had much different results. For normal computer usage, mouse verification did not work well, since it could never identify correctly who was using the mouse. The second test produced much better results, identifying the user correctly while performing similar actions made during enrolment, as well as detecting impersonation for the same activities. In short, despite mouse verification not working as intended, all objectives set previously were accomplished.

Chapter 6

Conclusion and Future Work

In the final chapter, overall conclusions are laid out along with what was not (possible to be) accomplished. It also discusses what the future holds in terms of work to be done in order to improve the system and provide an even more positive experience.

6.1 Main Conclusions

In a world where everything is more and more connected due to IoT, new technologies must be developed to cover for inevitable flaws that may occur. Nowadays remote work and remote assessments are a reality more common than ever, so in order to make sure all accesses and interactions with these systems are secure and performed by the legitimate and authorised people, many strategies were implemented, with some of them failing or not being ideal.

The project behind this dissertation aimed at the creation of a system prototype to help in such circumstances. To do so, behavioural biometrics were applied to the XGBoost machine learning algorithm to predict if the person behind the keyboard and mouse was the right user. To do so, keyboard and mouse dynamics were utilised to feed the algorithm.

The final system prototype has its flaws, but in the end, the initial purpose of this dissertation was achieved. User verification by the system is rather accurate when using the keyboard, and with some improvements, mouse verification may become viable as well. Since users are humans, emotions insert another possible layer to the correct identification of who is behind the computer. Although this was not tested during development, it can be inferred from [Pen17] and [SRAA17].

6.2 Future Work

By simply analysing mouse results, it is possible to deduce that at least in that topic, there is much more that can be done. Mouse dataset should be more robust by having more features, such as click timings for left, right and middle buttons, scroll distance and speed, etc. More data should be collected as well, with specialised methods to not induce the system in a bias, as it was shown in chapter 5.2.

For this system to be applied to remote assessments and interactions, a method to send information, in a secure way, to an admin or responsible can be implemented, were absolutely no information of what said user is doing is sent, but a way to inform them that

something might be wrong. Before this tool goes live, more intense tests and tuning should be made to ensure everything works smoothly. Since the system is practically invisible on the computer, not much is needed to make it ideal in that sense.

Bibliography

[AaW15] Tanapat Anusas-amornkul and Kasem Wangsuk. A comparison of keystroke dynamics techniques for user authentication. In *Proceedings of 2015 International Computer Science and Engineering Conference (ICSEC)*, pages 1–5, 2015. xv, 10, 11, 17, 20

[AS18] Ananya and Saurabh Singh. Keystroke dynamics for continuous authentication. In *Proceedings of 2018 8th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, pages 205–208, 2018. 9, 14

[Assa] Assembleia da República. Act no. 109/2009. Republic Diary No. 179/2009, Series I of 2009-09-15. Available from: `https://dre.pt/pesquisa/-/search/489693/details/maximized` (last accessed on 22/07/2021). vii, 1

[Assb] Assembleia da República. Act no. 46/2018. Republic Diary No. 155/2018, Series I of 2018-08-13. Available from: `https://data.dre.pt/web/guest/pesquisa/-/search/116029384/details/maximized` (last accessed on 22/07/2021). vii, 1

[ben] Human benchmark. Available from: `https://humanbenchmark.com/` (last accessed on 25/03/2021). 29

[Bro21] Sara Brown. Machine learning, explained, Apr 2021. Available from: `https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained` (last accessed on 05/09/2021). 7

[Cen19] Centro Nacional de Cibersegurança. Quadro nacional de referência para a cibersegurança, 2019. Available from: `https://www.cncs.gov.pt/docs/cncs-qnrcs-2019.pdf` (last accessed on 22/07/2021). 2

[Edu] IBM Cloud Education. What is machine learning? Available from: `https://www.ibm.com/cloud/learn/machine-learning` (last accessed on 05/09/2021). 7

[FEM+12] Clint Feher, Yuval Elovici, Robert Moskovitch, Lior Rokach, and Alon Schclar. User identity verification via mouse dynamics. *Information Sciences*, 201:19–36, October 2012. Available from: `https://doi.org/10.1016/j.ins.2012.02.066`. 8

[git] Git. Available from: `https://git-scm.com` (last accessed on 10/10/2021). 29

[lit] Typelit.io. Available from: `https://www.typelit.io` (last accessed on 25/03/2021). 29

[Nat18] National Institute of Standards and Technology. Framework for improving critical infrastructure cybersecurity, version 1.1. 2018. Available from: `https:`

//doi.org/10.6028/NIST.CSWP.04162018 (last accessed on 05/09/2021). vii, 1

[NJ10] Koichiro Niinuma and Anil K. Jain. Continuous user authentication using temporal information. In B. V. K. Vijaya Kumar, Salil Prabhakar, and Arun A. Ross, editors, *Biometric Technology for Human Identification VII*, volume 7667, pages 201 – 211. International Society for Optics and Photonics, SPIE, 2010. Available from: https://doi.org/10.1117/12.847886. 9

[oST] National Institute of Standards and Technology. Back to basics: Multi-factor authentication (mfa). Available from: https://www.nist.gov/itl/applied-cybersecurity/tig/back-basics-multi-factor-authentication (last accessed on 05/09/2021). 6

[Pen17] A. Pentel. Emotions and user interactions with keyboard and mouse. In *Proceeding of 2017 8th International Conference on Information, Intelligence, Systems Applications (IISA)*, pages 1–6, 2017. 8, 43

[Ros20] Ronald Ross. Security and privacy controls for information systems and organizations, 2020-09-23 2020. 5

[SCB16] Enrico Schiavone, Andrea Ceccarelli, and Andrea Bondavalli. Continuous authentication and non-repudiation for the security of critical systems. In *Proceedings of 2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*, pages 207–208, 2016. 10

[SIKP20] Shivshakti Singh, Aditi Inamdar, Aishwarya Kore, and Aprupa Pawar. Analysis of algorithms for user authentication using keystroke dynamics. In *Proceedings of 2020 International Conference on Communication and Signal Processing (ICCSP)*, pages 0337–0341, 2020. xv, xvii, 9, 11, 17, 20, 22, 24, 27, 41

[SR18] Gangadhar Shobha and Shanta Rangaswamy. Chapter 8 - machine learning. In Venkat N. Gudivada and C.R. Rao, editors, *Computational Analysis and Understanding of Natural Languages: Principles, Methods and Applications*, volume 38 of *Handbook of Statistics*, pages 197–228. Elsevier, 2018. Available from: https://www.sciencedirect.com/science/article/pii/S0169716118300191. 6

[SRAA17] R. Shikder, S. Rahaman, F. Afroze, and A. B. M. A. Al Islam. Keystroke/-mouse usage based emotion detection and user identification. In *Proceedings of 2017 International Conference on Networking, Systems and Security (NSysS)*, pages 96–104, 2017. 8, 43

[SZJK07] T. Sim, S. Zhang, R. Janakiraman, and S. Kumar. Continuous verification using multimodal biometrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(4):687–700, 2007. 8

[xgb] Introduction to boosted trees. Available from: https://xgboost.readthedocs.io/en/stable/tutorials/model.html (last accessed on 7/1/2022). 20

Appendix A

Annexes

This annex presents the code snippet mentioned in chapter 4 and shows how the keyboard predictions are processed using `XGBoost` algorithm.

```
def predictKeyboardXGBoost(featureList):
    global endProgram, keyboardAccuracy
    # Label total.
    header = ['label']
    setOfKeys = 9
    # Number of features times user input.
    for x in range(5 * setOfKeys):
        y = x
        header.append(f'f{y}')

    # Creation of the DataFrame with the information delivered from
        verification.py
    df = pd.DataFrame(featureList, columns=header)

    df2 = pd.read_csv(keyboardSetPath)
    df.label = pd.Categorical(df.label)
    df['code'] = userIndex
    y = df["code"]
    X = df.drop(['label', 'code'], axis=1)
    training = xgb.DMatrix(X, label=y)

    # Boost algorithm with the parameters and load model.
    bst = xgb.Booster(tunedParamsKeyboard)
    bst.load_model(keyboardModelPath)

    predictions = bst.predict(training)
    bestPredictions = np.asarray([np.argmax(line) for line in predictions])

    keyboardAccuracy = accuracy_score(y, bestPredictions)
```

Listing A.1: Prediciton process made by XGBoost algorithm.