Browser User Privacy - Identifying Users via Browser Interactions

Submitted by

Zhaoyi Fan

for the degree of Doctor of Philosophy

of the

Royal Holloway, University of London

2023

Declaration

I, Zhaoyi Fan, hereby declare that this thesis and the work presented in it is entirely my own. Where I have consulted the work of others, this is always clearly stated.

Signed............................(Zhaoyi Fan)
Date:

To my family

Contents

List of Figures

List of Tables

Abstract

Browser Application Programming Interfaces (APIs) allow web developers to create complex functionality for a website and enrich web user experience. Browser APIs also allow web sites to access a wide variety of user host data. Because of this, these APIs have enabled a wide range of user privacy and security issues and attacks. Of particular relevance to the work described in this thesis, the *KeyboardEvent* APIs, the *MouseEvent* APIs, the *Orientation* APIs and *Motion* APIs can be potentially be used to extract user behaviour patterns. This observation has motivated the work described in this thesis.

In recent years, a range of studies have shown that keystroke and mouse dynamics can be used for user authentication and/or identification. As a result, it may well be possibly for a curious web site to identify the individual human who is using a device to access that site. In this thesis, we conduced experiments for collecting user data with such APIs and applied machine learning techniques to analyse the collected data. We also tested our machine learning models on public datasets and achieved promising results.

To assess the real-world significance of the results, we examined various browsers across a range of platforms. We found that all common browsers can potentially reveal a user's identity without user permission or knowledge. Finally, we give suggestions for simple modifications to browsers to give users greater control over their privacy.

Acknowledgements

My deepest gratitude goes first and foremost to my supervisor, and mentor, Professor Chris Mitchell for his patience, guidance and support. Without his insightful ideas and precious feedback, this thesis would never become possible.

I would like to thank my colleagues and friends, Wanpeng Li, Nasser Al-Fannah, Fatma Al Maqbali and Angela Heeler for the wonderful reading group and delicious food.

Thank you to all my friends who helped through my experiments, Boxuan Han, Tao Zhang, Jiaqing Zhao, Yueming Jin, Ran Yi, Xiao Sa, Shi Wang, Qianjiang Wu, Yingjie Zhao, Jing Sun and Xiao Wang, Rondo, Lina Song, Feng Gao, Yu, Erdou, Pang, Ting and Jiao Wu. Thank you for your trust and support.

A very special thank goes to my friends who trained together in the gym. I will always remember the friendly and motivating atmosphere.

I am extremely grateful to my father, Hua Fan and mother Yumei Zhao for their support and love in every possible way. They have always been there at any time. I extend my thanks to all my relatives in my big family. Thank you all for the encouragement and care.

Last, but not least, I would like to express my gratitude to my beautiful wife, Shenshen Liu, for her love, support and understanding. I thank for everything she gave me.

List of Abbreviations

ACC	Accuracy
AUC	Area Under the Curve
AI	Artificial Intelligence
ANN	Application Programming Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
DOS	Denial of Service
EER	Equal Error Rate
FAR	False Acceptance Rate
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPR	False Positive Rate
FRR	False Rejection Rate
GBM	Gradient Boosting Machine
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
JS	JavaScript
JSON	JavaScript Object Notation
K-NN	K-Nearest Neighbours

ML	Machine Learning
OS	Operating System
PPV	Positive Predictive Value
RF	Random Forest
ROC	Receiving Operating Characteristic Curve
SVM	Support Vector Machine
TN	True Negative
TNR	True Negative Rate
TP	True Positive
TPR	True Positive Rate
URL	Uniform Resource Locater
XGBoost	eXtream Gradient Boosting

Chapter 1

Introduction

1.1 Introduction

Websites are capable of learning a wide range of information about the platform on which a browser is executing. One major source of such information is the set of standardised Application Programming Interfaces (APIs) provided within the browser, which can be accessed by JavaScript downloaded to a browser by a website; this information can then either be used by the JavaScript or sent back to the originating site.

Such APIs have been used to enable a wide range of user privacy threats and security attacks. Al Fannah and Li [7] pointed out that a large number of websites use *Browser fingerprinting* techniques to collect user data with such APIs. The APIs could be used to reveal various types of data relating to a web user, e.g., Internet Protocol (IP) address [6], user browser information [88], etc. Additionally, various attacks based on web APIs have been described [17, 39, 54, 64, 78]. There seems little doubt that the various services offered by browser APIs enable user devices to be tracked across the Internet.

Apart from the security and privacy issues affecting end user devices based on web APIs, as we examine in this thesis, browser APIs also pose a direct threat to the privacy of human users. The *KeyboardEvent* and *MouseEvent* APIs can be used to continuously provide data when a user is interacting with the browser via a keyboard and mouse. Mobile sensor APIs can be used to continuously provide data when a user is interacting with the browser on a mobile device. Such data could potentially be gathered and analysed via machine learning techiques to generate either a user biometric template (e.g. for identifying the same user in future) or for comparing with an existing set of templates. Instead of using a browser fingerprint to track a specific device, the web user identity could potentially be revealed directly using such an approach.

This chapter provides an overview of the thesis, and is organised as follows. Section 1.2 introduces the motivation for the research described in this thesis. Section 1.3 describes the main objectives of this research. Section 1.4 outlines the major contributions. The structure of the thesis is described in Section 1.5.

1.2 Motivation

In recent years various user privacy issues arising from the use of browser APIs have been discussed. Although a number of steps to reduce the privacy impact of browser API use have been taken by relevant authorities [21], there are still some APIs (e.g. the *KeyboardEvent* APIs, the *MouseEvent* APIs, and the *Motion* APIs) that can be used to reveal user information without any user permission or knowledge.

At the same time, over the last 30 years, much research has focussed on the use of keystroke and mouse dynamics as methods for biometric authentication and identification — see, for example, [4, 10, 13, 34, 46, 59, 60]. However, as far as the author is aware, none of the previous studies consider their use for a user identification process in a browser based environment. This observation, combined with the possibility that user identification could potentially be performed without the knowledge or consent of the user, has motivated the research described in this thesis.

1.3 Objectives

This thesis attempts to answer the following main question.

- To what extent can a user be identified by a web server using behavioural data recovered from the web browser in an uncontrolled environment?

In attempting to answer this question we examine four subsidiary questions, based on the main types of behavioural data available to a web server from a browser.

- How effectively can browser-provided keystroke dynamics data be used for user identification in an uncontrolled environment?
- How effectively can browser-provided mouse dynamics data be used for user identification in an uncontrolled environment?
- How effectively can a combination of browser-provided keystroke and mouse dynamics data be used for user identification in an uncontrolled environment?
- How effectively can browser-provided mobile touch data be used for user identification in an uncontrolled environment?

1.4 Contributions

To enable a range of experiments to be performed, we first surveyed the APIs in modern browsers and established an open source environmental platform which can continuously gather a visiting user's keystroke and mouse dynamics data.

- We developed a publicly available (`https://github.com/fanzhaoyi/DataCollector`) Chrome extension that can continuously and quietly collect browser user keystroke and mouse movement information, and a server which can receive and store the data from the extension. This has been made freely available for use by any individuals who would like to conduct similar experiments or research.

In our second group of contributions, we demonstrated that browser APIs on both PC and mobile platforms can be used to reveal a user's identity. We used machine learning techniques for analysing keystroke and mouse dynamics data which were collected in the experiments conducted with our developed experimental platform, and used similar machine learning methods for analysing a public mobile sensor dataset. We achieved promising experimental results.

- We firstly performed experimental analysis on keystroke dynamics data obtained from a set of 20 participants. The experimental environment for the participants was completely free, i.e. the users could freely type any text as in their daily routine. The experimental results showed that, using 20 continuous keystrokes on a PC platform, a user could be identified with 89.2% f1-score.

- We secondly performed experimental analysis on mouse dynamics data obtained from the same set of 20 participants. The experimental results showed that, using 10 move-and-click mouse actions in a browser-based environment on a PC platform, a user could be identified with 94.2% f1-score. We also did a similar analysis using a much larger public mouse dataset (the Bogazici dataset [52]) which involved 19 users and also achieved a promising 95.0% f1-score.

- We thirdly performed experimental analysis on the combination of keystroke and mouse dynamics data obtained from the same set of 20 participants. The experimental results indicated that the identification performance improved when keystroke and mouse dynamics were combined, compared to using either one independently. By using 5 move-and-click mouse actions and 5 consecutive keystrokes in a browser environment on a PC platform, a user could be identified with 94% f1-score.

- We fourthly performed experimental analysis on mobile keystroke dynamics data from a public dataset [51]. The experimental results showed that, using 10 continuous keystrokes on a mobile device, a user could be identified with 96.1% f1-score. We then developed a mobile web page which can use mobile browser APIs to collect the same type of user data as was in the public dataset we worked on [51].

Finally, we examined a range of browsers running on various platforms to check which browsers and platforms our results applied to. We then gave some simple suggestions for browser functionality which would enable users to exert greater control over their privacy.

1.5 Structure of the thesis

The remainder of this thesis is organised as follows.

- Chapter 2 provides background material on Browser APIs and briefly summarises existing security and privacy issues arising from such APIs.
- Chapter 3 introduces the machine learning techniques relevant to this thesis.
- Chapter 4 reviews the behavioural biometrics based on keystroke dynamics, mouse dynamics and mobile sensor data.
- Chapter 5 introduces the experimental platform.
- Chapter 6 describes the analysis and experimental results on keystroke dynamics.
- Chapter 7 describes the analysis and experimental results on mouse dynamics.
- Chapter 8 describes the analysis and experimental results on combining keystroke and mouse dynamics.
- Chapter 9 describes the analysis on touch dynamics on mobile device.
- Chapter 10 outlines the attack models and possible mitigations for the identified privacy issues.
- Chapter 11 concludes the thesis and highlights possible areas for future work.

Chapter 2

Browser APIs

2.1 Introduction

The JavaScript language is widely used by websites to enable dynamics behaviour. In the context of a web browser, JavaScript code is downloaded along with the web page. In general, APIs (application programming interfaces) are constructs made available in programming languagues to allow developers to create complex functionality more easily. Browser-supported APIs, which can be accessed by website-originated JavaScript, provide website developers with more efficient ways to accomplish their goals and provide browser users with a better experience.

Client-side JavaScript has many available APIs [61]. Browser APIs allow developers to create web pages that incorporate information from the user's environment [90]. In HTML5, a range of browser APIs (e.g. audio, application cache, canvas, fullscreen, geolocation, local storage, notifications, pointerlock, video and web database [20]) can be used to provide user information, and can potentially be used by a malicious website to reveal user personal data.

This chapter provides background material on Browser APIs and briefly summarises the security and privacy issues arising from such APIs and is structured as follows. Section 2.2 introduces browser APIs and the information that can be obtained by browser APIs. Section 2.3 provides a detailed discussion of all the attacks based on browser APIs. Section 2.4 introduces prior art on device fingerprinting techniques. Section 2.5 discusses the other possible attacks based on browser APIs and Section 2.6 concludes this chapter.

2.2 Browser APIs

2.2.1 Types of Browser API

When a user visits a web page, JavaScript code from different sources is downloaded into the user's browser. Running in the user's browser, this JavaScript code is able to access to different information via browser APIs. For example [61], the Document Object Model (DOM) API is able to manipulate HTML and CSS on a web page. The Fetch APIs can be used to handle requests and responses in a network activity. the WebGL and Canvas APIs allow uses to programmatically update the pixel data contained in an HTML element to create 2D and 3D scenes. The Audio and Video APIs provide interaction between users and multimedia. The Device APIs are commonly used for manipulating and retrieving data from modern device hardware, e.g. camera, microphone and other device sensors. Client-side storage APIs are able to store data on the client-side. This section provides a detailed introduction to such browser APIs.

2.2.1.1 APIs for manipulating documents in the browser

The commonly discussed API is probably the DOM API which can create, remove and change HTML and CSS. The *Document* interface represents any web page loaded in the browser and serves as an entry point into the web page's content, which is the DOM tree. The *Event* interface, which is one of the various interfaces that provided by the DOM. The *Web Workers* API can run a script operation in a background thread which is separate from the main execution thread of a web application. The *Performance* interface is part of the High Resolution Time API. It provides a DOMHighResTimeStamp which represents the number of milliseconds elapsed since a reference instant. The interface that can be accessed by *event* is as follows.

2.2.1.1.1 KeyboardEvent It describes a user interaction with the keyboard. Each event describes a key with the event type of *keydown*, *keypress* and *keyup*.

2.2.1.1.2 MouseEvent It represents the events that occur due to the user interacting with a pointing device, such as a mouse. It can record the events such as moving a mouse, clicking button, etc.

2.2.1.1.3 FocusEvent It provides handlers tasked with managing focus events, which correlate with the instances when elements gain or lose focus.

2.2.1.1.4 WheelEvent It passes the *wheel* event, which is triggered when the mouse wheel is rotated or a comparable component like a touchpad is used.

2.2.1.1.5 MutationObserver It offers the capability to observe any modifications to the DOM tree.

2.2.1.1.6 CompositionEvent It allows a user to enter characters that may not be present on a physical keyboard.

Table 2.1: Examples of Event APIs

API calls	properties
KeyboardEvent	It describes a user interaction with the keyboard. Each event describes a key with the event type of *keydown*, *keypress* and *keyup*.
MouseEvent	It represents the events that occur due to the user interacting with a pointing device, such as a mouse. It can record the events such as moving a mouse, clicking button, etc.

2.2.1.2 APIs that fetch data from the server

The *XMLHttpRequest* and *Fetch* API allow modern websites and applications to transfer data between client and server. The *Fetch* APIs can handle and modify *requests*, *responses* and other network requests.

2.2.1.3 APIs for drawing and manipulating graphics

These APIs are widely used in drawing and manipulating graphics. The most popular ones are the *Canvas* and the *WebGL*. The *Canvas* can be used to render a variety of text and graphics. The *WebGL* exposes different attributes of web browsers and hardware, such as GL version, max texture size and supported WebGL extensions, etc.

2.2.1.4 Audio and video APIs

The Web Audio API provides a powerful and versatile system for controlling audio on the web, allowing developers to choose audio sources, add effects to audio, create audio visualizations, apply spatial effects and much more. The Web Real-Time Communications (WebRTC) is a technology which enables Web applications and sites to capture and optionally steam audio and/or video media, as well as to exchange arbitrary data between browsers without requiring an intermediary. It allows to share data and perform teleconferencing peer-to-peer, without requiring that the user install plug-ins or any other third-party software. However, Al-Fannah [6] suggests that WebRTC has brought a new threat to user privacy. Various client IP addresses could be visible due to the use of the WebRTC API even if a VPN service is in use.

2.2.1.5 Device APIs

Device APIs refer to the APIs that manipulate and retrieve data from modern device hardware. The *Geolocation* API gives the Web content access to the device's location. The *ambientLightSensor API* returns an interface for reading the current light level. The *deviceMotionEvent* provides web developers with information about the speed of changes for the device's position. The *microphone* and *camera* APIs allow the web application access to the microphone and camera of a device. Table 2.2 shows the common API calls of the *DeviceMotionEvent*.

Table 2.2: The *deviceMotionEvent* API

API calls	**properties**
acceleration	An object giving the acceleration of the device on the three axis X, Y and Z. Acceleration is expressed in m/s^2.
accelerationIncludingGravity	An object giving the acceleration of the device on the three axis X, Y and Z with the effect of gravity. Acceleration is expressed in m/s^2.
rotationRate	An object giving the rate of change of the device's orientation on the three orientation axis alpha, beta and gamma. Rotation rate is expressed in degrees per seconds.
interval	A number representing the interval of time, in milliseconds, at which data is obtained from the device.

2.2.1.6 Mobile device APIs

When a user is interacting with a web page by touching the touch screen, various data can be accessed by mobile APIs [62], e.g., the *Accelerometer* and *Orientation* APIs.

2.2.1.6.1 Earth coordinate frames It is important to understand coordinate system where *DeviceMotionEvent* and *DeviceOrientationEvent* are used. The Earth coordinate frame is fixed on the center of the Earth, as described as **X**, **Y**, and **Z**. The axes are aligned based on the direction of gravity and the magnetic north orientation.

- **X** axis: follows along the ground plane, perpendicular to the Y axis and positive toward the east.
- **Y** axis: follows along the ground plane, and is positive toward north.
- **Z** axis: is perpendicular to the ground plane, and is positive upward.

2.2.1.6.2 Device coordinate frame The device coordinate frame is fixed on the center of the device, as described as **x**, **y**, and **z**.

- **x** axis: follows along the plane of device screen, perpendicular to the **y** axis and positive toward the right.
- **y** axis: follows along the plane of device screen, and is positive toward top.
- **z** axis: is perpendicular to the plane of device screen, and is positive outward from the screen.

2.2.1.6.3 Rotation direction Rotation is measured in degrees, describing the difference between the device's coordinate frame and the Earth coordinate frame. It has the following values.

- *Alpha*: when rotating the device around the **z** axis. The *alpha* angle is 0 when top of the device is pointing to the north and decreases as rotating toward the left.
- *Beta*: when rotating the device around the **x** axis, which is toward of away from the user. The *beta* angle is 0 when the **y** axis of device coordinate frame is perpendicular to the **Z** axis of the Earth coordinate frame, and decreases as tipping away from the user.
- *Gamma*: when tilting the device toward left or right. The *gamma* angle is 0 the **x** axis of device coordinate frame is perpendicular to the **Z** axis of the Earth coordinate frame, and decreases as tilting toward the left.

2.2.1.6.4 DeviceMotionEvent The *DeviceMotionEvent* API provides the speed of changes for device's position and orientation. It has the following properties:

- *acceleration*: returns the amount of acceleration recorded by the device (without the effect of gravity force).
- *accelerationIncludingGravity*: returns the amount of acceleration recorded by the device (with the effect of gravity force). It has three values according to the three axes: x for **X** axis, y for **Y** axis and z for **Z** axis.
- *rotationRate*: returns the rate at which the device is rotating around each of its three axes in degrees per second. It has three values according to the three axes: *aplha* for **z** axis, *beta* for **x** axis and *gamma* for **y** axis.

2.2.1.6.5 DeviceOrientationEvent The *DeviceOrientationEvent* API provides the physical orientation of the device. It has the following properties.

- *alpha*: returns the rotation of the device around the **Z** axis, ranged between 0 to 360 degrees.
- *beta*: returns the rotation of the device around the **X** axis, ranged between -180 to 180 degrees.

- *gamma*: returns the rotation of the device around the **Y** axis, ranged between -90 to 90 degrees.

2.2.1.6.6 Gyroscope The *Gyroscope* API provides the value angular velocity of the device along all three axes. It has the following properties.

- *x*: returns the angular velocity of the device along the device's **x** axis.
- *y*: returns the angular velocity of the device along the device's **y** axis.
- *z*: returns the angular velocity of the device along the device's **z** axis.

2.2.1.6.7 TouchEvent Apart from the *Accelerometer* and *Motion* sensor APIs, mobile browser also has the Browser APIs similar to those on a PC platform, e.g. the *TouchEvent* API which describes one or more points of contact with the screen. There are three common types of the *TouchEvent* API, as described as follows.

- *touchstart*: triggered when one or more touch points are placed on the touch screen.
- *touchmove*: triggered when one or more touch points are moved on the touch screen.
- *touchend*: triggered when one or more touch points are removed from the touch surface.

Additionally, the *touchEvent* API has the following properties.

- *screenX*: returns the X coordinate of the touch point relative to the left edge of the screen.
- *screenY*: returns the Y coordinate of the touch point relative to the top edge of the screen.
- *clientX*: returns the X coordinate of the touch point relative to the left edge of the browser.
- *clientY*: returns the Y coordinate of the touch point relative to the left edge of the browser.
- *pageX*: returns the X coordinate of the touch point relative to the left edge of the document, including horizontal scroll.
- *pageY*: returns the Y coordinate of the touch point relative to the top edge of the document, including horizontal scroll.

2.2.1.6.8 KeyboardEvent The *KeyboardEvent* API which describes a user interaction with the keyboard. One of its common properties is *KeyboardEvent.key* which returns the key value of the key represented by the event. There are two common types of *KeyboardEvent*, described as follows.

- *keydown*: triggered when a key is pressed.
- *keyup*: triggered when a key is released.

2.2.1.7 Client-side storage APIs

These APIs grant web browsers the ability to store data on the client side. Table 2.3 explains the common API calls of storage APIs.

Table 2.3: Common storage APIs

API calls	**properties**
sessionStorage	This gives a separate storage area for each given origin that's available for the duration of the page session.
localStorage	This has the similar function as *sessionStorage* but persists even when the browser is closed and reopened.

2.3 Examples of browser API attacks

A malicious website can make use of these APIs. A typical example is phishing websites. An attacker uses a fake website which is pretended to be a genuine one. The phishing website collects the user's credentials, (e.g. account information, bank card information), and sends back to the attacker's server.

Additionally, a number of side-channel attacks based on the high resolution timestamp provided by the *performance* API are described in [39, 54, 65, 87]. The High-resolution Time API (e.g. *performance.now*) is a sub-millisecond time measurement which provides precise timestamp. Oren et al. [65] presented a micro-architectural side-channel cache attack running entirely in the browser. The attackers used the controlled web page with malicious content to collect timing information. When the victim visited the untrusted webpage, their behaviour can be successfully tracked. Gruss et al. [39] presented a page-deduplication attack that can collect a user's private information. Lipp et al. [54] used this APIs to infer a user's PIN input, and URLs input in the browser.

Mobile sensor APIs provide various functionalities for users and Web developers. Such APIs can also be used to enable a wide range of user privacy threats and security attacks. The attackers may use sensor APIs to collect data and use machine learning techniques to analyse such data and achieve user personal information. Different side-channel attacks were demonstrated to compromise a user's privacy, with using various sensor APIs, e.g. the *camera* and *microphone* APIs [78], the *Light sensor* API[79], the *Stereoscopic* microphone and *Gyroscope* APIs [64], the *Orientation* and *Motion* APIs [58].

2.3.1 Attacks based on High-resolution Time API

2.3.1.1 Cache attack

Oren et al. [65] demonstrated a cache attack which is one of the general class of micro-architectural side-channel attacks. They suggested that the *performance* API could be used to distinguish the cache changes in the cache attack. In the attack model, the victim views a web page that contains the malicious content, e.g. an advertisement controlled by the attacker. Instead of requiring the victim to interact with the malicious web page, the web page runs its JavaScript code to launch the cache attack, which allows the attacker to track accesses to the victim's last-level cache over time. They showed that this attack can be used to track a user's behaviour. They achieved an accuracy of more than 80% for detecting the user's browsing activity.

2.3.1.2 Page deduplication attack

The page-deduplication attack is one of side-channel attacks, which exploit timing differences in write accesses on deduplicated pages. Gruss et al. [39] demonstrated a page deduplication attack based on JavaScript. This attack allows an adversary to gather a victim's private information, e.g. whether a program or website is currently opened. In their attacks, they used *malloc* to create a large array to fill the page. To measure the write-access time, they used *performance.now()* which provides a high resolution timestamp. They found that images and CSS style sheets in websites are page-aligned in memory. Thus, they leveraged this property to extract the page content. They applied their attacks on a wide range of platforms (e.g. mobile phone, personal computer and multi-tenant cloud systems). When a victim visited a web page containing the malicious contents, the browser downloaded the JavaScript code and then executed it. After a period of time, the results were sent back to the attacker. They showed that such attack could be used to extract sensitive information, e.g. browsing behaviour of a user.

2.3.1.3 Timing attacks

When a user visits a web page, the web server responses according to the requests sent from the browser. The size of responding message normally differs from the requesting content. Timing measurements are related to such difference and could be used to expose the users' privacy. Van et al. [87] demonstrated the timing attacks which could expose personal information of user social networks (e.g. Facebook, Twitter and LinkedIn), and searching history of Google and Amazon. They used *Performance* API to measure the timestamp and leveraged the timing information as the side channel to expose personal data.

Lipp et al. [54] used keystroke timestamp as a side channel and demonstrated the attacks on a range of platforms, e.g. personal computer, laptops, and smartphones. They designed their attack into two phases, online phase for gathering timing traces and offline phase where the collected data were processed. Although the *performance.now* resolution was changed from 1 microsecond to 5 microseconds [21], it was still possible to use this API to launch timing attack. In the online phase, the authors used consecutive sliced single-threaded event loop with an interval of 4ms as an endless loop to record the timestamp. Additionally, *Web Worker* API was used to allow JavaScript code to be executed in the background. When a user pressed the keyboard, there was an obvious peak in the loop. Over time, thousands of traces can be gathered. In the offline phase, the authors used k-Nearest Neighbours (k-NN) algorithm as the classifier to calculate the correlation of traces. As a result, URLs of 10 most visited websites that a victim entered into the address bar in the browser could be inferred, with a lowest accuracy of 67% and highest accuracy of 96% respectively. In their experiment of touchscreen interactions, the user's PIN input could be inferred with an average accuracy of 50%. It should be noted that this attack still worked even if the controlled web page run in the background.

2.3.2 Attacks based on device APIs

Modern mobile phones use various hardware and sensors to provide move functionality and better user experience, (e.g. camera, microphone, light sensor, motion and orientation sensor). An application on the mobile phone can access to various user data via the APIs provided by such sensors. However, if malicious applications have the permission to use these APIs, the system would be vulnerable. A range of side channel attacks based on mobile sensor APIs have been employed — see, for example, [17, 58, 64, 78].

2.3.2.1 Attacks using sound/light sensors

Simon and Anderson [78] developed a system called PIN Skimmer to demonstrate a side-channel attack by using the front camera and microphone. They designed a game in the PIN Skimmer to collect training data. The user interface of the game consisted of 12 icons of emoji which shares the similar layout as the PIN pad. In the data collecting phase, when a user touched the screen, the PIN Skimmer toke a picture with the front camera. Then the picture was uploaded to a remote server for analysis. At the server side, relevant features were extracted from the collected data to build profile for the user. When a victim used other application and entered his/her PIN, the PIN Skimmer turned on the front camera to film. Then the video was uploaded to the server for data processing with the trained profile. They pointed out that the camera use might attract the victim's attention, such as the LED of the front camera and shutter sound of taking pictures. However, in the root mode, both the LED and shutter sound can be disabled via Android APIs. They showed that 50% of the 4-digit PINs could be correctly guessed after 5 guessing attempts. They indicated that the result might be improved with other sensor involved (e.g. accelerometer and gyroscope).

Narain et al. [64] showed that the *stereoscopic* microphone API could leak sensitive information. They used stereo-microphones and gyroscope to record the tap sounds and vibrations. They designed their attack model into 5 phases. First, the adversary developed the malicious application. Second, they lured the victim to install the controlled application by some means (e.g. social engineering). Third, when the permission was granted, the application started to collect the victim's typing behaviour through the custom keyboard and sensor data from the stereo microphone and gyroscope. Fourth, the collected data was regarded as training data and sent to a remote server to create specific profile for the victim. Fifth, the applications run in the background and monitored the opened application in the system and the current location information. When a specific application was opened, e.g. bank application, the malicious application started to collect sensor data and upload to the server for inferring the victim's keystrokes. They achieved a high accuracy over 90% of inferring the corresponding keys.

2.3.2.2 Attacks using motion sensors

When a user types on the soft keyboard on the mobile phone, the vibration occurs. Cai and Chen [17] designed an application called ToughLogger based on Android system. Once the application was installed into the system and was granted the motion sensor access, it started to spy on the keystroke via mobile sensors. *DeviceOrientationEvent* was used to read the vibration and collect this orientation data while typing. TouchLogger extracted features from the motion signals and used supervised learning methods to infer keystrokes. They trained the TouchLogger with a data set that consists of motion data with corresponding keys. TouchLogger achieved a 71.5% accuracy. Their work indicated that motion signal on the mobile system could be leveraged as a side channel and personal information could be leaked via such side channel.

Mehrnezhad et al. [58] designed a system called TouchSignatures which compromised the web security in a browser based on motion sensor data. In contrast to applications installed in a system, TouchSignatures does not require any permissions as it runs in the browser. They developed a *listener* for recording sensor data and a web page as an interface for collecting user data. When a user visited the web page, the JavaScript code was downloaded automatically and was executed to collect data. In a meanwhile, *socket.IO* was used to set up a socket connection between the client and the server. The collected raw data was sent through this socket connection. A number of features were extracted from the raw data. They used various classification methods in different data processing phases. They designed the experiments into two phases, i.e., identifying user touch actions and identifying PINs. In the first phase, they achieved an overall 87% accuracy of identifying a user's touch actions, (e.g. click, hold, scroll and zoom). In the second phase, they achieved an 77% accuracy of inferring a PIN. They employed their attacks on different platforms with various scenarios.

They showed that, when the malicious web page is active, or an *iframe* window with the controlled JavaScript code is opened, the attack could be successful in almost every browser on both the Android and IOS system. Even the web page run in the background and the screen is locked, there were still some vulnerable browsers supporting such attacks.

2.3.3 WebRTC IP address leakage

Al-Fannah [6] pointed out that IP addresses could be leaked via the *WebRTC* APIs even a VPN service was in use. He showed that five types of client IP address could be revealed via the *WebRTC* API, i.e., the public IPv6 address, the public temporary IPv6 address, the unique local address assigned by LAN, the private IP address assigned by the VPN server and the private IPv4 address assigned by LAN. He examined five common browsers (e.g. the Chrome, Firefox, Edge, Safari and Opera browsers) on Windows and MacOS, with five chosen VPN programs. The experimental results including 40 testing cases showed that expect for the Safari browser on MacOS, the rest of browsers could cause IP address leakage with or without a VPN service to some extends.

2.4 Device fingerprint

Device fingerprinting, also called browser fingerpinting, is commonly used as a tool for tracking users. Eckersley [28] designed a web site that could identify its visitor by collecting information provided by the vistor's browser, e.g. IP address, User Agent, HTTP headers, cookies usage, screen resolution, timezone, browser plugins, plugin versions and Multipurpose Internet Mail Extensions (MIME) types, and system fonts. The results showed that a large number of the observed browsers had unique fingerprints. Even though the browser fingerprint changed rapidly, the 'upgraded' version of the browser fingerprint techniques still could be inferred with 99.1% accuracy of guesses.

A wide range of information can be used for fingerprinting. Alaca and Oorschot [8] summarised the common device fingerprinting vectors. They divided the vectors into four categories:

Browser-provided information: software and hardware details, WebGL, system time, battery, evercookies, webRTC, password autofill.

Inference based on device behaviour: HTML5 canvas, system performance, hardware sensors, scroll wheel, CSS feature detection, JavaScript standards conformance, URL scheme handlers, video RAM detection, font detection, audio processing.

Browser extensions and plugins: browser plugin (e.g. Java, Flash, Silverlight, etc.), browser extension.

Network- and protocol-level techniques: IP address, geolocation, active TCP/IP stack fingerprinting, passive TCP/IP stack fingerprinting, protocol, DNS resolver, clock skew, counting host behind NAT, ad blocker.

Fifield and Egelman [32] suggested that the onscreen dimensions of font glyphs could be used as a feature for web browser fingerprinting. These information can be collected by a web server and potientially reveal the browser behaviour. Eckersley [28] pointed out that the trade off between fingerprintability and users' privacy needs to be considered.

2.5 Other possible attacks

Permission policies have been widely used in common browsers. When a web application requests to access to specific content via the mobile sensors, e.g. geolocation, camera, microphone, JavaScript, notifications, popups, etc., a user can choose to allow or deny the permission. However, some contents do not require any permissions. *Event* APIs provide web application accesses to DOM content. The *KeyboardEvent* and *MouseEvent* APIs are widely used by web applications. For example, the *KeyboardEvent* API is able to indicate what is happening on a key, e.g. pressing down a key, holding a pressed key or releasing a key. This could be leveraged by a phishing website to record a victim's keyboard inputs. Additionally, keystroke dynamics and mouse dynamics has been studied for many years. Ahmed and Traore [3] demonstrated that keystroke dynamics technique could be used in a biometric recognition system. By analysing key press value and timing information, they achieved an equal error rate of 2.45% with 53 users involved in their experiments. Mondal and Bours [60] showed that a combination of keystroke and mouse could be used as a soft biometric in continuous authentication.

In a web browser, the *keyboardEvent* can easily record both a key press value and timing information, mouse actions can be recorded by the *MouseEvent* API, and mobile sensor APIs can be used to provide user data. When a user visits a website with malicious content, the JavaScript code is downloaded automatically and executed. The website can record the interacting data from a visiting user and generate a unique profile for the user. When the profile becomes more complex and sophisticated, the user's identity could potentially be recognized by any parties with such profile. An advertisement company may leverage this to generate targeted advertising for the victims. Contrasting with traditional credentials, e.g. password and PINs, the user behaviour patterns may be difficult to change in a very short period of time. Thus, a potential threat to the online users' privacy could be posed once a website gathers sufficient user behavioural data.

2.6 Conclusion

In this chapter, we introduced the prior art on user privacy threats and security attacks arising from the misuse of the Browser APIs. Section 2.2 introduces the Browser APIs that are widely used by web developers and the sensitive information revealed by such APIs. Section 2.3 describes a range of side-channel attacks with various channels. Section 2.4 introduces the prior art on browser fingerprinting techniques. Section 2.5 discusses possible other security and privacy issues arising from the Browser APIs. Although the Browser APIs bring web users more exciting experience, the awareness of security and privacy should be considered.

Chapter 3

Machine learning

3.1 Introduction

We live in a digital world with various data, e.g. Internet of Things (IoT) data, cybersecurity data, mobile data, business data, social media data and health data. [74]. Such data can be used for different intelligent applications, e.g. voice recognition [83], spam filter [22], traffic prediction [29] and disease diagnostic [30]. Artificial Intelligence (AI), particularly, machine learning (ML) has gained more popularity in recent years for analysis with such data.

Machine learning allows a system to learn and make decisions based on the given data. There are roughly six steps for analysing data with machine learning techniques [35], i.e., data collection, data preparation, model selection, training with data, model evaluation and make predictions, as is shown in Figure 3.1. In this chapter, we introduce the relevant machine learning techniques. The structure of this chapter is organised as follows. Section 3.2 describes data types in machine learning. Section 3.3 introduces data preparation process, e.g. data cleansing and feature engineering. Section 3.4 introduces the type of machine learning techniques. Section 3.5 introduces common machine learning algorithms. Section 3.6 explains model evaluating process and section 3.7 concludes this chapter.

3.2 Data collection

The first step for analysing data with machine learning is collecting data. As Sarker [74] introduced, the data can be divided into 4 categories: structured data, unstructured data, semi-structured data and metadata.

3.2.1 Structured data

Structured data has a well-defined data structure and it is highly organised. A program can easily access and use such data (e.g. names, dates, addresses,

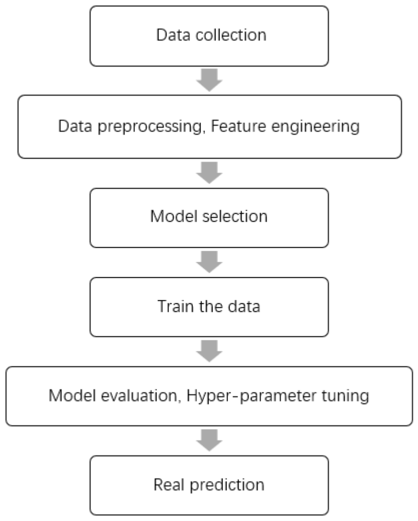

Data collection
Data preprocessing, Feature engineering
Model selection
Train the data
Model evaluation, Hyper-parameter tuning
Real prediction

Figure 3.1: Machine learning flowchart

credit card numbers, stock information, geolocation, genders and colours).

3.2.2 Unstructured data

Unstructured data does not follow a standard format or data structure, i.e., it is more challenging to capture, process or analyse (e.g. sensor data, emails, audio files, video files and images).

3.2.3 Semi-structured data

Semi-structured is between the structured and unstructured data. It is not well-defined but follows certain rules (e.g. HTML document, XML document and JSON documents).

3.3 Data preparation

3.3.1 Data cleaning

The collected data could be unusable without preprocessing as there could be some missing or abnormal values in a well-structured data. Normally, data cleansing is applied for processing corrupt or inaccurate data within a dataset [92].

3.3.2 Feature extraction

Unstructured or semi-structured data, e.g. contents in a webpage or images, may not directly provide useful features for data analysis. Feature extraction can create useful and meaningful new features from such dataset. This provides better understanding of the data, better model performance and reduce computational cost [74].

3.3.3 Feature selection

Unlike feature extraction, which creates new features, feature selection will not create any new features. The main method of feature selection is to select better features from the existing ones by using various algorithms, e.g. Variance threshold [67], Pearson correlation [67] and Principal component analysis (PCA) [67].

3.4 Types of machine learning techniques

This section introduces the common machine learning types. Based on the existence of human supervision, machine learning can be roughly divided into four groups, i.e., supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning [35].

3.4.1 Supervised learning

In supervised learning, the model observes input and output pairs and learns a function mapping from input to output. The input can be called the features and output can be called the labels [72]. A typical task of such supervised learning is regression, e.g. predicting price of products.

Another typical task of supervised learning is classification. Classification can be roughly divided into three categories, i.e., binary classification, multiclass classification and multi-label classification.

Binary classification refers to a classification task with only two class labels, e.g. *True* or *False* in a spam filter system.

Multiclass classification often refers to the classification tasks with multiple classes more than two. An example of multiclass classification is classifying different types of networks attacks, e.g. the Denial of Service Attack (DoS), the User to Root Attack, the Root to Local Attack and the Probing attack [74].

Multi-label classification refers to a classification task with multiple outputs for an input [35], i.e., a movie can be classified as an action movie and an English movie at the same time.

3.4.2 Unsupervised learning

In unsupervised learning, the model learns from the data without labels. It can be used to extract generative features, identifying trends and structures results, grouping in results, etc. The common tasks of unsupervised learning are: clustering, density estimation, feature learning, dimensionality reduction, anomaly detection, etc [35].

3.4.3 Semi-supervised learning

Semi-supervised learning uses both supervised and unsupervised learning methods. A good application example is photo-hosing service [35]. The system automatically divides the photos into groups according to person. Then the system needs the users to add labels for each recognised person. Some other applications are machine translation, fraud detection, labelling data and text classification [74].

3.4.4 Reinforcement learning

In reinforcement learning system, the system can observe the environment, select and perform optimal behaviour and get rewards or penalties and then get the best strategy with most rewards and least penalties (e.g. robotics, autonomous driving tasks, and DeepMind AlphaGo) [35, 74].

3.5 Machine learning methods

As the primary machine learning tasks in Chapters 6, 7, 8 and 9 are classification, this section mainly introduces the common classification algorithms.

3.5.1 K-Nearest Neighbours

K-Nearest Neighbours (k-NN) is probably one of the most fundamental and simple classification methods [68]. It is based on a majority vote of the k nearest neighbours to a testing point, using distance (e.g., Euclidean distance) for measurement. K-NN can be used for both classification and regression task.

3.5.2 Support Vector Machine

Support Vector Machine (SVM) is widely used for classification, regression, and outlier detection [35]. In high-dimensional space, SVM constructs a hyper-plane to separate the data. It can be utilised with different kernel functions, e.g. linear, polynomial, radial basis function (RBF) and sigmoid kernels.

3.5.3 Decision Tree

Decision trees are popular for classification and regression tasks. A decision tree model is a flowchart-like structure model with various decision nodes [73].

3.5.4 Random Forest

A Random Forest classifier is an ensemble classification technique that operates by constructing multiple decision tree classifiers and using majority voting or averaging to determine the outcome. A Random Forest model with its multiple decision trees is often more effective than a single decision tree model [74].

3.5.5 Extreme Gradient Boosting

Gradient Tree Boosting, also known as Gradient Boosting Machine (GBM), is an ensemble method that builds upon multiple weak prediction models, e.g. decision tree. A Gradient tree boosting model iteratively learns from weak classifiers with adjusting weights and finally achieves a strong classifier [25]. EXtreme Gradient Boosting (XGBoost) is a more powerful Gradient Tree Boosting model with faster training speed and better performance [19].

3.6 Model evaluation

This section introduces model evaluation and the relevant evaluating metrics.

3.6.1 Cross-validation

It is important that a machine learning model performs evenly well on all training data. K-fold cross-validation method can be used for such purpose. Normally, 5-fold or 10-fold is selected based on the dataset size. The training set is divided into k groups and will be computed for k iterations until all the data are being trained and tested. In each iteration, one group is regarded as temporary testing set and the rest $k-1$ groups are regarded as temporary training set. Then an average result is computed as the model performance. Cross-validation method is usually used with hyper-parameter tuning methods to improve the performance of a model [35].

3.6.2 Hyper-parameter tuning

Hyper-parameters are higher-level parameters which need to be set manually before training [2]. Most of the *sci-kit learn* machine learning models have their default hyper-parameter settings. It is recommended to search the hyper-parameters for the best *cross-validation* score to improve model performance [67].

3.6.3 Evaluating metrics

To evaluate a machine learning model, specific evlauating metrics are used for certain tasks. This subsection [23] introduces the common metrics used in classification tasks. Table 3.1 instroduces the binary classification outcomes, as shown as following:

- True Positive (TP): the model correctly predicts the positive case, e.g. a spam filter correctly filtered a spam email.
- True Negative (TN): model correctly predicts the negative case, e.g. a spam filter correctly identified a genuine email.
- False Positive (FP): the prediction on a negative case is positive, e.g. a spam filter incorrectly labelled a genuine email as a spam email.
- False Negative(FN): the prediction on a positive case is negative, e.g. a spam filter incorrectly labelled a spam email as a genuine email.

Table 3.1: Binary classification confusion matrix

	Predicted: True	Predicted: False
Actual: True	True Positive (TP)	False Negative(FN)
Actual: False	False Positive (FP)	True Negative (TN)

Based on the above mentioned metrics, more metrics can be calculated as follows:

- True positive rate (TPR): also known as recall, can be calculated as :

$$TPR, recall = \frac{TP}{TP + FN} \tag{3.1}$$

- True Negative rate (TNR): can be calculated as :

$$TNR = \frac{TN}{TN + FP} \tag{3.2}$$

- False Positive Rate (FPR): can be calculated as :

$$FPR = \frac{TP}{TN + FP} \tag{3.3}$$

- False Negative Rate (FNR): can be calculated as :

$$FNR = \frac{FN}{FN + TP} \tag{3.4}$$

- Positive predictive value (PPV): also known precision, can be calculated as :

$$PPV, precision = \frac{TP}{TP + FP} \tag{3.5}$$

- Accuracy (ACC): can be calculated as :

$$ACC = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.6}$$

- F1-score: a harmonic mean of the precision and recall [67], can be calculated as :

$$F1score = 2 \cdot \frac{precision \cdot recall}{precision + recall} = \frac{2TP}{2TP + FN + FP} \tag{3.7}$$

- Receiving operating characteristic curve (ROC): can be used for displaying between TPR and FPR at different threshold.

- Area Under the ROC (AUC): can be used for calculating the area under the ROC curve.

- False acceptance rate (FAR): the liklihood that a biometric system incorrectly labelled an imposter as a genuine user [4], similarly as False positive rate [81].

- False rejection rate (FRR): the liklihood that a biometric system incorrectly labelled an genuine user as an imposter [4], similarly as False negative rate [81].

- Equal error rate (EER): the probability when FAR equals to FRR.

- Confusion matrix: a summary of prediction results, as shown in Table 3.1

3.7 Conclusion

In this chapter, we mainly introduced the machine learning techniques relevant to the analysis in this thesis. Section 3.1 explains the common steps of a machine learning model. Section 3.2 introduces the data types in machine learning. Section 3.3 explains the common process of preparing the collected data. Sections 3.4 and 3.5 introduce the type of machine learning techniques and common machine learning algorithms. Section 3.6 introduces the model evaluating methods and common measuring metrics.

Chapter 4

Behavioural biometrics

4.1 Introduction

The biometric system [86] is an identification and authentication system that uses unique measurable physical characteristic [86], e.g. fingerprint, palmprint, face, iris, retina, etc., or behavioural traits [86], e.g. gait, keystroke dynamics [40], mouse dynamics [4], etc.

Biometric identification involves recognizing an individual based on distinct physiological and/or behavioural traits [44]. A biometric system operates as a pattern recognition system that verifies an individual's identity by authenticating a unique physiological or behavioural attribute.

Normally, this system consists of enrolment and identification modules. During the enrollment phase, a biometric sensor scans the individual's biometric trait, converting it into a digital format. Then a feature extractor extracts the features and stores them as 'template'.

During the recognition phase, the biometric system captures the individual's characteristic and analyse it. This analysed data is then processed by the feature extractor to compare with the stored template to determine the individual's identity.

Most traditional biometric technologies that are based on physical characteristics can be utilised in various scenarios and often achieve good performance. However, they possess certain drawbacks [14][86]. Typically, they require additional sensors to scan the subject and collect the necessary information, requiring subjects actively participate in the recognition process. In contrast, some behavioural biometrics do not require such expensive and infrequently-used sensors. For example, keystroke and mouse dynamics biometrics only require the user to interact with a keyboard and mouse, while biometrics based on mobile sensors simply need access to these sensors on the mobile device. These methods are less costly and do not demand the user's active attention.

This study aims to identify browser users by analysing their keystroke and mouse dynamics data, as well as mobile sensor data, by applying existing behavioural biometric technologies. In this chapter, we review prior work on behavioural biometrics focusing on keystroke dynamics, mouse dynamics, and mobile sensors in terms of experimental designs, data processing, and machine learning algorithms.

The remainder of this chapter is structured as follows: Section 4.2 introduces keystroke dynamics. Section 4.3 discusses mouse dynamics. Section 4.4 reviews the combined approach using keystroke and mouse dynamics. Section 4.5 explores behavioural biometrics based on mobile sensor data. Finally, Section 4.6 discusses the classification tasks in this study and concludes the chapter.

4.2 Keystroke dynamics

The field of keystroke biometrics has been studied since the 1990s [13]. It is concerned with the behavioural analysis of users, seeking to identify or authenticate individuals through the distinctive patterns in their typing rhythms. Such patterns include the duration of key presses, the delay between consecutive keystrokes, the average speed of typing, the frequency and nature of typographical errors, individual preferences in keystroke sequences, and the force exerted on keys [13, 47]. This section reviews prior work on behavioural biometrics performance based on keystroke dynamic.

4.2.1 Experimental designs

The keystroke studies can be roughly divided into two categories: static analysis and dynamics analysis. Static keystroke analysis means all the participants are asked to type in the same predetermined text during the experiments. Killourhy and Maxion [49] set **.tie5Roanl** as the predetermined text for their experiments. They recruited 51 subjects and asked them to complete 8 data-collection sessions (of 50 passwords each), for a total of 400 password-typing samples.

Araujo et al. [13] selected 10-digit strings as their experimental texts. They recruited 30 users and set the experiments on three machines with two different keyboards. They categorized the experiments into three types: legitimate user authentication, impostor user authentication, and observer impostor user authentication. The experimental results indicated that it is difficult to learn other user typing behaviour in a short period of time. The familiarity of a target string could significantly influence the authentication performance. They suggested that adaptation mechanism should be used to improve the authentication performance.

Hosseinzadeh and Krishnan [43] set up their experiments with 41 participants over a period of four weeks. They developed an application for keystroke pattern authentication, i.e. *KbApp*. The users were asked to install the application on their own PC and provide keystroke samples regularly.

Kang and Cho [46] developed keystroke data collection programs for different devices, e.g. traditional PC keyboard, a soft keyboard and touch keyboard. Each

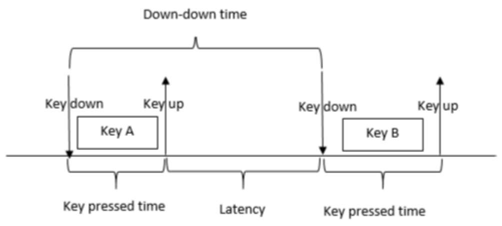


Figure 4.1: Keystroke timing features

user was asked to provide at least 3,000 characters via the program on their devices.

Gunetti and Picardi [40] set up their experiments for six months. They recruited 41 volunteers (as legal users) to provide 15 typing samples each. Under the same condition, another 165 users (as impostors) were asked to provide one typing sample. Users were free to choose what they would like to provide as the typing samples. The gathered samples varied in length from 700 to 900 characters.

Rodrigues et al. [71] recruited 20 users and set up their experiments on a *Pentium IV* microcomputer platform, using only numeric keyboard part. Each user was free to choose an eight-digit string as the experimental text. In total, 1400 samples were collected.

Giot et al. [37] used the **GREYC** keystroke database [36] which involved 133 users with a total number of 7555 samples within 2 months.

Kasprowski et al. [48] conducted their research using the public Buffalo dataset [82]. This dataset consists of both fixed length of input and free text from 148 participants. Participants were asked to use a keyboard to type their answers to several questions. The average interacting period for each user is about 100 minutes. In their classification tasks, data from 20 users were selected.

Lu et al. [56] used the Buffalo dataset [82] and the Clarkson dataset [89] in their study. A total of 39 participants contributed to the data collection, with a mixture of passwords, fixed text, and free text across two sessions, each lasting two hours per user. On average, each participant generated approximately 21,533 keystrokes.

Altwaijry [9] invited 85 users for their data collection. Each user was asked to type in a fixed length of password and a phrase for 400 times in both English and Arabic in a controlled environment.

4.2.2 Data extraction

Features can be extracted from raw typing data. While typing, keyboard inputs, e.g. the pressed key value and relevant timing information, are recorded. Each key press and release generates two timestamps. Using these timestamps, various timing features can be derived, as illustrated in Figure 4.1:

- DD time (Down-Down Time): The latency between consecutive key presses.
- UD time (Up-Down Time): The latency between a key release and the subsequent key press.
- DU time (Down-Up Time): The duration of a single key press.
- UU time (Up-Up Time): The latency between consecutive key releases.

A review of timing features employed in prior research is summarized in Table 4.1. Pisani and Lorena [69] suggested that DU and UD times are the most commonly used features. Furthermore, Chang et al. [18] incorporated keystroke pressure in their study of mobile device usage. However, pressure data cannot be collected through traditional keyboards as they require additional sensors.

Table 4.1: Extracted timing features in keystroke dynamics studies

Authors	**Extracted features**
Giot et al. [38]	DU, DD, UD
Giot et al. [37]	DU, DD, UD
Hosseinzadeh and Krishnan [43]	DU, DD, UD, UU
Killourhy and Maxion [50]	DU, DD, UD
Chang et al. [18]	DU, DD, UD
Araujo et al. [13]	DU, DD, UD
Kang and Cho [46]	DU, UD
Liu et al. [55]	DU, DD, UD
Gunetti and Picardi [40]	DD
Bartlow and Cukic [15]	DU, UD
Rodrigues et al. [71]	DU, DD, UD
Kasprowski et al. [48]	DU, DD, UD, UU
Lu et al. [56]	DU, DD, UD, UU
Altwaijry [9]	DD, DU, UD

Giot et al. [38] applied DD, UD, DU, and UU times in their research, along with a measure of total typing time. Hosseinzadeh and Krishnan [43] reported DD and DU times as the most popular features. Killourhy and Maxion [50] examined various combinations of timing features, noting that authentication performance varied accordingly. Chang et al. [18] concentrated on password keystroke authentication for touchscreens, employing timing and pressure features in their study. In addition to timing features, Araujo et al. [13] included key code data in their analysis, utilizing mean and standard deviation to process timing information. They observed that 98% of the collected timing data fell between 10 ms and 900 ms, with 1-ms precision. Kang and Cho [46] selected DU and UD times for their study, highlighting the space key as the most frequently utilised. The vowels 'a', 'e', 'i', 'o' and the consonants 't' and 'n' were among the most commonly typed characters on a PC keyboard. The syllables 'he', 'th', 'an', 'in', and 'er' were the most prevalent in their dataset. Gunetti and

Picardi [40] employed n-graphs with DD time as their primary timing feature, while Bartlow and Cukic [15] integrated shift-key patterns into their feature set, yielding 41 feature vectors for each input sequence. Araújo et al. [13] employed the Manhattan distance metric to process their data, applying DU, DD, and UD timing features in their experiments. They demonstrated that a combination of these three features resulted in better experimental outcomes compared to using a single feature alone.

4.2.3 Machine learning approaches

Classification generally aims to find the category that best matches the data being classified. A variety of classification algorithms have been used in previous studies, as outlined in Table 4.2.

Table 4.2: Summary of classification methods in keystroke dynamics studies

Authors	**Classification methods**
Giot et al. [38]	SVM, Neural Network
Giot et al. [37]	SVM, Statistical method
Hosseinzadeh and Krishnan [43]	Gaussian Mixture Model
Killourhy and Maxion [50]	Nearest Neighbour, Neural Network
Chang et al. [18]	Statistical method
Araujo et al. [13]	Statistical method
Kang and Cho [46]	Statistical method, k-NN
Liu et al. [55]	Statistical method
Gunetti and Picardi [40]	Statistical method
Bartlow and Cukic [15]	Random Forest (RF)
Rodrigues et al. [71]	Hidden Markov Model(HMM), Statistical method
Kasprowski et al. [48]	Artificial Neural Network (ANN)
Lu et al. [56]	Convolutional Neural Network (CNN) and Recursive Neural Network (RNN)
Altwaijry [9]	AdaBoost, Decision Tree (DT), RF, SVM

4.2.3.1 Support Vector Machine

Support Vector Machine (SVM) is a widely-used supervised learning algorithm that constructs a decision boundary with the largest possible margin between different classes, based on support vectors [59]. Giot et al. [37] employed a two-class SVM during the enrollment phase. Each training set, $\{x_i, y_i\}$, consists of an enrolled vector x_i and a label $y_i \in (-1, 1)$ denoting genuine or impostor users. A threshold was established to determine the legitimacy of a user. If the verification is successful, an update mechanism is used to refresh the training sets.

4.2.3.2 Nearest neighbour

K-NN algorithm finds the k most similar instances based on a certain similarity metric, e.g. the Euclidean distance, the Manhattan distance or the Mahalanobis distance [46]. Killourhy and Maxion [50] used Nearest Neighbour (Manhalanobis) algorithm. In the training phase, training vectors were stored in the detector and covariance matrix was calculated. In the test phase, the detector calculates the Manhalanobis distance between the testing set and training set according to the calculated covariance matrix.

4.2.3.3 Manhattan distance

Araujo et al. [13] used Manhattan (scaled) distance in their research. The mean and deviation of each timing feature is calculated in the training phase. In the test phase, the score of testing sample is calculated as (4.1)

$$D = \frac{1}{n}\sum_{i=1}^{n}\frac{x_i - \mu_i}{\sigma_i} \tag{4.1}$$

where μ and σ represents the mean and the standard deviation of the training sample respectively , i is the ith elements in the samples and n is the number of elements of the training sample.

4.2.3.4 R measure and A measure

Gunetti and Picardi [40] used n-graphs and applied R measure and A measure in their study. The R measure referred to the relative typing speed of a user. The rationale behind the R measures was that the a user's typing speed may change due to some reasons. But the changes were expected to affect all the typing in a similar way. Normally a user's typing speed in a warm room condition is faster than that in a cold condition. However, if typing speed of each digraph in a sample S_1 is exactly twice as the typing speed of the same digraph in a sample S_2, the R measure fails to distinguish these two samples. Thus, the A measure was introduced, referring to the absolute typing speed of each keystroke. Kang and Cho [46] suggested that the combination of the R and A measures achieved the best performance in their algorithms.

4.3 Mouse dynamics

Mouse dynamics has been explored for several years as a potential 'soft' biometric trait for authentication systems [31, 59, 60, 76]. Similar to keystroke dynamics, mouse dynamics analysis does not require specialized hardware, leveraging the standard mouse, a primary PC interface, for data acquisition. This section reviews previous work on the performance of behavioural biometrics based on mouse dynamics.

4.3.1 Experimental designs

Ahmed and Traore [4] engaged 22 participants and developed client software to record mouse actions and transmit the data to a server in real time. The users were asked to install the software on their own machines. Running in the background, the software started monitoring when the user logged in and stopped when the logged out. Data was collected over a total of 998 sessions, averaging 45 sessions per user. Over 9 weeks, they collected 284 hours of raw mouse data, with an average input of 13 hours per user.

Shen et al. [76] developed a Windows application guide users through the same mouse-operated tasks on the same desktop computer. They invited 37 participants who provided data twice daily for up to 60 days, yielding 5550 samples.

Mondal and Bours [59] used a public mouse dynamics dataset [63] in their study. 48 users were asked to use their computer and mouse in a normal way, without any restrictions on the tasks. There is a huge variation in the number of samples per user (i.e., minimum 3736, maximum 333789, average 60701).

Kılıç et al. [52] introduced the Bogazici dataset, a large-scale public mouse dynamics dataset with 24 users and 2550 hours of data. An application in Python programming language was developed to continuously listen to mouse movements and clicks and then record them into a file with timestamps, interactive applications types and mouse action details. Due to the inadequacy of training data and the weak training performance associated with the 5 users, their data were considered potential external threats.

Siddiqui et al. [77] recruited 10 users in their experiment. During this study, the participants were instructed to play in a 20-minute gaming session of *Minecraft*, while their mouse activities were captured through a Python script.

Rahman and Basak [70] refined a public dataset [33]. The improved version of this dataset has ten users. Each user contributes between 5 to 7 sessions. Every session in this dataset is close to 120 minutes.

Antal et al. [11] accumulated a large dataset from 120 subjects. They developed a browser based gaming script. Each user was asked to play the designed game and follow the instruction, such as clicking on a triangle or double-clicking on a square, etc.

4.3.2 Data extraction

The raw data collected from a mouse device can be roughly divided into three categories: the coordinates of the mouse pointer, the click actions and the corresponding timestamps. Various features can be extracted from such raw data. Table 4.3 outlines the mouse actions described in [31, 59, 60, 76]. Features can be extracted as follows.

Table 4.3: Mouse action types and descriptions

Actions	Description
Mouse-Move (MM)	General mouse movement
Left button down(LD)	Press the left mouse button
Left button up(LU)	Release the left mouse button
Right button down(RD)	Press the right mouse button
Right button up(RU)	Release the right mouse button
Mouse wheel(W)	Scroll the mouse wheel
Drag-and-Drop (DD)	Begins with mouse button down, movement, and then mouse button up
Point-and-Click (PC)	Mouse movement followed by a click or a double-click
Silence	No movement detected

- **Travelled Distance Histogram (TDH)**: The distribution of the travelled distance for every action type.
- **Action Type Histogram (ATH)**: The relative frequency of the MM, DD and PC actions within a session.
- **Movement Direction Histogram (MDH)**: The ratio of actions performed in each one of the eight directions. This feature is represented by 8 values, as is shown in figure 4.2.
- **Average Movement Speed per Movement Direction (MDA)**: The average speed over all the actions performed in each one of the eight directions. This feature is represented by 8 values.
- **Average Movement Speed per Types of Actions (ATA)**: The average speed of performing the MM, DD and PC actions.
- **Movement Speed compared to the Travelled Distance (MSD)**: Approximation of the average traveling speed for a given travelling distance.
- **Movement elapsed-Time Histogram**: The time distribution for performing an action.

4.3.3 Machine learning approaches

Table 4.4 summarises the classification methods employed in previous mouse dynamics research. Mondal and Bours [59] used the SVM and ANN classifiers in their studies. Feher et al. [31] applied the Random Forest algorithm in their work. Shen et al. [76] applied a one-class SVM and Neural Network classifiers. Additionally, k-NN with Mahalanobis distance was used. During the training

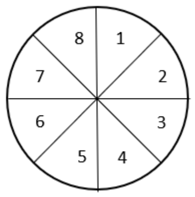


Figure 4.2: Diagram of the 8 Cardinal Directions for Mouse Movement

Table 4.4: Classification methods in mouse dynamic

Classification methods	**References**
SVM	Shen et al. [76], Mondal and Bours [59], Antal et al. [11]
Random Forest	Feher et al. [31]., Siddiqui et al. [77], Rahman and Basak [70]
K-NN (Mahalanobis)	Shen et al. [76], Rahman and Basak [70]
Neural Network	Ahmed and Traore [4], Shen et al. [76]

phase, the covariance matrix of the training features was calculated. In the testing phase, the nearest neighbour classifier computed the Mahalanobis distance from the new feature sample to each sample in the training data.

4.4 Behavioural biometrics based on combining mouse and dynamics

The previous two sections have conducted a review of keystroke dynamics and mouse dynamics. Each has demonstrated promising potential for classification on its own. In practical settings, it is typical for PC users to simultaneously engage with both keyboard and mouse during a given period. This section reviews the combined use of keystroke and mouse dynamics in behavioural biometrics.

4.4.1 Experimental designs

Bailey et al. [14] conducted experiments with 31 participants, collecting data from keyboard, mouse, and GUI interactions. They developed a software application running on Windows 7 to capture the users' operations. Participants were required to answer three questions and write a report ranging from 400 to 500 words for each question. The gathered data was segmented into samples, each covering a 10-minute interval.

Mondal and Bours [60] recruited a total of 53 participants. Participants were required to install a specially designed software on their personal computers. They were given complete freedom in terms of their activities and behaviours on the computer, with no restrictions or specific instructions imposed. Throughout the duration of the experiment, which spanned 5-7 days, an average of 700,000 events were recorded for each participant. Among these events, keystroke and mouse events constituted 12.4% and 83.3% of the total, respectively. To differentiate between 'genuine users' and 'imposter users', binary classifiers were applied. Around 35% of the total data was used for training classifiers, while approximately 10% was used for tuning hyper-parameters. The remaining 55% of the data was reserved for the testing phase.

Earl et al. [27] recruited 240 participants in their study, with a 225 right handed and 15 left handed users. The distribution of age, and gender between female and male participants was approximately even. The users were asked to do follow various clicking tasks according to the instructions on the screen. Additionally, they were asked to type in a long fixed text. Instead of focusing on authentication/identification tasks for users' identities, they aimed to identify users' handedness, ages and gender.

Wang et al. [91] invited 41 participants in data collection. The participants were aked to carry out four tasks: typing, browsing Taobao website, browsing Weibo website, and gaming on the same laptop. Each task took one hour. For the gaming task, participants played a customized game on Windows, following the on-screen instructions and move and click mouse. The two website browsing tasks required the users to use mouse for most of the time and a small amount of keystroke activities. During the typing task, users were required to type the same article in Microsoft Office Word.

4.4.2 Data extraction

Bailey et al. [14] computed the average Up-Up (UU) and Down-Up (DU) times as keytroke dynamics features. In terms of mouse dynamics, they calculated features such as speed, direction, travel distance, and click timing information derived from mouse movements and clicks.

Mondal and Bours [60] used digraphs and extracted DD, DU, UD, and UU for the keystroke features. As for the mouse actions, the study focused on Mouse Move and Drag-Drop actions, extracting various parameters such as angles, speed, accelerations, and distance, based on the cursor's coordinates and timestamps.

Wang et al. [91] segmented based on time windows ranging from 10s to 480s. Then they proposed a feature extraction method 'fusing the Scene-Irrelated features and User-Related features for User Authentication' (SIURUA). They suggest that users' keystroke and mouse behaviours may vary across different scenes. So they abandoned such high scene-correlated features as they considered such features may influence the authentication accuracy. Therefore, scene-irrealted features were selected and used. In contrast, some features vary significantly between users but show minimal variation for an individual user.

These features are called ‘user-related features’ as they are more distinguishable to users than other features. DD, DU, UD, UU features were extracted for the keystroke dynamic. Data related to mouse dynamics, including movement speed, direction, travel distance, and timing were extracted.

4.4.3 Machine learning approaches

Bailey et al. [14] employed two classifiers, BayesNet and LibSVM in their analysis. A combination of classification methods was used by Mondal and Bours [60] , including Artificial Neural Network (ANN), Counter-Propagation Artificial Neural Network (CPANN), and Support Vector Machine (SVM). Instead of focusing solely on authentication metrics such as FRR or FAR, the study introduced the Average Number of Imposter Actions (ANIA) and the Average Number of Genuine Actions (ANGA) as alternative evaluation metrics. These metrics provide insights into the average number of actions an imposter is able to perform before being locked out by the authentication system. Earl et al. [27] tested on various algorithms: Random Forest, SVM, Decision Tree, Gaussian Naive Bayes and KNN. Wang et al. [91] used Multiple Kernel Learning method to combine kernels from multiple source to improve the performance of the target kernel and used SVM as the classification method.

4.5 Behavioural biometrics based on mobile platform

The footprint of mobile devices expands continuously in this digital world. Xiaomei et al. [93] report daily activities involving approximately 400,000 Apple and 1.3 million Android devices. Mobile applications are capable of using various mobile sensors (e.g., GPS, gyroscope, compass and accelerometer) and accessing to user data [58]. Similarly, various browser APIs can access user data during interactions with webpages. The potential security and privacy risks posed by mobile APIs have been highlighted, with Bojinov et al. [16] raising concerns that mobile sensors could be exploited to compromise user anonymity. In addition, Mehrnezhad et al. [58] demonstrated how orientation and motion sensors might be leveraged to decipher a user’s PIN entry. Furthermore, Zhang et al. [93] indicated that interactions with mobile devices could produce various collectable data, which could be instrumental as behavioural biometric identifiers. This section reviews prior work on behavioural biometrics based on mobile platform.

4.5.1 Experimental designs

Zheng et al. [94] invited over 80 participants in their experiment. The users were asked to type 4-digit and 8-digit PINs respectively for 25 times on a provided Samsung Galaxy Nexus, with holding the phong with left hands and tapping with their right hand index fingers.

Gascon et al. [34] recruited 315 users in their experiment. A softkeyboard prototype for Android OS was developed an installed on the prepared testing smartphone. All users were asked to type a predefined text with approximately 160 characters.

Antal et al. [12] invited 42 people in their study. The experimental duration was two weeks. An android application was developed and installed on a Nexus 7 tablet and a Mobil LG Optimus L7 device for data collection. The users were asked to enter a pre-defined password (**.tie5Roanl**) for 30 times.

Teh et al. [84] recruited 150 users in their study. A collection tool using Java and Android API was developed and installed on Sumsung Galaxy tablet. The experimental location was chosen by the involved users at their best convenience, including offices, homes, inside cars, classrooms, and public areas. The users were asked to enter a 4-digit PIN and a 16-digit PIN for 10 times respectively.

Kim and Kang [51] invited 50 participants in the experiment. An Android-based application was developed for data collection and installed on a provided Samsung Galaxy S8. Each user was asked to type 20 pre-defined text using thumbs of both hands in a sitting position.

Acien et al. [1] utilised a publicly available keystroke dataset [66] contributed to by 260,000 users. During each session, participants were required to memorize a randomly chosen sentence from a pool of 1,525 examples and then type it as quickly and accurately as possible on a browser webpage using their personal mobile device. Each participant was instructed to complete at least 15 such sessions. Due to the uncontrolled experimental conditions, only 23% of the participants (equating to 60,000 users) provided usable data. Acien et. focused their analysis on these 60,000 participants and their initial 15 sessions, extracting solely timing-based keystroke dynamics and keycode information, as no motion sensor data were collected in this study. The length of each keystroke sequence varied between 70 and 100 keystrokes.

4.5.2 Data extraction

Various types of data can be captured through mobile APIs when a user interacts with a mobile device's webpage. These data types can be leveraged with machine learning techniques for in-depth analysis, as follows.

- **Gyroscope**: Data provided by the *Gyroscope* API [26, 34, 51, 53, 93, 94].
- **Pressure**: Data provided by the *Pressure* API [12, 24, 26, 45, 75, 84, 94].
- **Orientation**: Data provided by the *DeviceOrientationEvent* API [34, 51].
- **Coordinate**: Coordinates of touch points [12, 26, 45].
- **Area**: Sizes of the touchpoint contacts [24, 26, 51, 53, 84, 94].
- **Accelerometer**: Data provided by the *Accelerometer* API [24, 34, 41, 51, 53, 94].

- **Time**: Timing based information, e.g., timestamps of keytouch down (KD) and up (KU) [1, 12, 24, 26, 34, 41, 45, 46, 51, 53, 75, 80, 84, 94].

4.5.3 Machine learning approaches

Table 4.5: Machine learning methods for mobile touch dynamics

Classification methods	**References**
SVM	Antal et al. [12], Gascon et al. [34], Ho [41], Jain et al. [45]
Random Forest	Antal et al. [12], Ho [41]
Nearest Neighbours	Antal et al. [12], Kang and Cho [46]
Distance based	Antal et al. [12], Zhen et al. [94], Lee et al. [53], Ho [41]
TyperNet	Acien et al. [1]
Transformer	Stragapede et al. [80]

Table 4.4 summarises the classification methods applied to mobile touch dynamics in previous research. Antal et al. [12] explored various machine learning algorithms, i.e., *distance-based* method, *k-NN*, *LibSVM* and *Random Forest*. Gascon [34] used *SVM* as the classification algorithms. Zheng et al. applied a *distance-based* method. Jain et al. [45] used *SVM* algorithm in their study. Lee et al. [53] used both *distance-based* and *SVM* methods. Acien et al. [1] developed a novel classification framework named TypeNet, which was used for both authentication and identification tasks. Stragapede et al. [80] employed the *Transformer* architecture as the machine learning approach.

4.6 Discussions and conclusions

4.6.1 Discussions

Biometric systems can have two distinct functions: authentication (verification) and identification (recognition) [12]. The verification function operates as a binary decision-making process. Here, the system evaluates whether to accept or reject the identity asserted by an individual. This function acts as a gatekeeper, ensuring that only those whose biometric data matches the pre-registered information gain access or authentication.

On the other hand, the identification system should analyse an input pattern and determine its correspondence to one of the 'N' predefined classes (multi-class) in its database. It is not a matter of 'yes' or 'no', but a question of 'which one'. Identification is important when the user's claimed identity is unknown. The system has to look through its stored biometric data to see to find a match.

This is important in situations where we need to figure out who someone is just from their biological or physical characteristics, without them telling us who they are first.

In this study, our primary goal is to reveal browser users' real identity via their interactions with the browser, rather than authenticating them whether they are the ones they claimed to be. Thus, identification via multi-class classification would be our main task in all scenarios.

4.6.2 Conclusions

In this chapter, we have presented an overview of behavioural biometrics systems. More specifically, we reviewed into the details of behavioural biometrics, examining keystroke dynamics in Section 4.2, mouse dynamics in Section 4.3, the combination of keystroke and mouse dynamics in Section 4.4, and the utilization of mobile sensors in Section 4.5. Across these four sections, we have reviewed previous research concerning experimental frameworks, data processing techniques, and machine learning methodologies. This section provides a conclusion to the chapter.

Chapter 5

Experimental platform

5.1 Introduction

Websites are capable of learning a wide range of information about the platform on which a browser is executing. One major source of such information is the set of standardised Application Programming Interfaces (APIs) provided within the browser, which can be accessed by JavaScript downloaded by a website; this information can then either be used by the JavaScript or sent back to the originating site. As has been widely discussed, much of this information can threaten user privacy. The main purpose of this paper is to document a publicly available platform designed to enable further investigation of one class of such threats, namely those based on analysing user behavioural data. The platform has two main components: a Chrome extension that gathers user keystroke and mouse data via browser APIs, and server software that collects and stores this data for subsequent experimentation.

The JavaScript language is widely used by websites to enable dynamics behaviour. In the context of a web browser, JavaScript code is downloaded along with the web page. This JavaScript is executed automatically by the browser, and can access a variety of information via a set of browser-provided Application Program Interfaces (APIs).

In general, APIs are constructs made available in programming languagues to allow developers to create complex functionality more easily. Browser-supported APIs, which can be accessed by website-originated JavaScript, provide website developers with more efficient ways to accomplish their goals and provide browser users with a better experience.

In a modern browser, Client-side JavaScript can access a range of APIs [61]. In general, browser APIs allow developers to create web pages that incorporate information from the user's environment [90]. In HTML5, a wide variety of information can be accessed via browser APIs, such as audio, application cache, canvas, fullscreen, geolocation, local storage, notifications, video and web database [20]. However, browser APIs can also be used by websites to learn potentially privacy-sensitive information about the browser, the platform on which the browser is executing, and even the user of the browser. For example, platforms can be tracked across multiple web visits using browser fingerprinting techniques [28], it may be possible to learn secret user data such as PINs [78] and the user location [6], and it may even be possible to learn about the human user.

The possibility of behavioural monitoring using browser APIs motivates the work described in this paper, namely the development of a platform to enable the collection of behavioural data. The platform has been designed to support experiments which aim to determine the degree to which individual users can be identified via browser APIs. When a web user is browsing a website, the standard Document Object Model (DOM) enables executing JavaScript to access information about keyboard and mouse events. These keyboard and mouse events can potentially be used to identify individual users using techniques developed for biometric identification and authentication [14]. The platform we describe in the remainder of this paper has been designed to support experiments aimed at understanding how effective such identification can be. However, the platform is not restricted to this application, and is being made available as a potential tool for further research on browser security and privacy.

The platform involves a Chrome extension that has been implemented to collect keystroke and mouse data from browser users. In addition, server functionality has been developed to receive and store this data.

The remainder of this chapter is structured as follows. Section 5.2 explains the motivation of the work in this chapter. Section 5.3 introduces the browser APIs used by the extension. Section 5.4 then addresses the implementation of the extension. Section 5.5 explains the ethical issues of the experiments. Section 5.6 introduces the user guidance of the experimental platform. Section 5.7 describes possible ways in which the current platform might be extended and Section 5.8 contains concluding remarks.

5.2 Motivation

When a user browses a web page, the providing website can download JavaScript that gathers keystroke and mouse data using browser APIs. The main objective of the planned research is to understand the degree to which such gathered data can be used to identify the user. To perform this research a means of collecting reasonably large data sets from a significant number of different users is needed.

The first proposed approach for collecting the data was to develop a special website (and accompanying JavaScript) to collect data from users. Test subjects would then be asked to interact with this website. However, this approach was abandoned for two main reasons. Firstly, developing a website which is sufficiently feature-rich to generate meaningful user interactions would be a very difficult and time-consuming task. Secondly, persuading a sufficient number of users to engage with this website for the time necessary to generate the required volume of data is also likely to be very difficult.

This analysis led to the development of the current experimental platform, involving a browser extension and a server to collect data from the extension. The browser extension gathers keystroke and mouse use data using the browser APIs in exactly the same way as website-provided JavaScript would. From time to time the extension uploads the gathered data to the server for storage and later processing. Experimental participants are simply asked to install the extension and sign in whenever they start using their browser, and their keystroke and mouse use data is gathered without any further user involvement. This should enable large datasets to be gathered from a significant number of different individuals.

One shortcoming of this approach is that it gathers more data than a single website would be able to gather, since all web activity is monitored and not just activity with one website. That is, it has the ability to gather more data than a single website would be able to. However, this seems reasonable as a first step in gaining an understand of what is possible – also, many websites use JavaScript provided by third parties (such as Google – see, for example, [5]), and such third party providers would thereby be able to gather keystroke and mouse data from user interactions with a multiplicity of sites. At a later stage the extension could be modified to only monitor keystroke and mouse activity when the user is visiting a particular site.

5.3 Background

5.3.1 Keyboard and mouse events

We first introduce the browser API calls used by the experimental platform to collect data on user behaviour. Specifically we detail APIs which provide information regarding user interactions with keyboards and pointing devices (e.g. a mouse).

The most commonly discussed browser API is probably the DOM API which can create, remove and change HTML and CSS. The Document interface provides a representation of a web page loaded in the browser, and serves as an entry point into the web page content in the form of the DOM tree. The *Event* API, one of the interfaces provided by DOM, provides the *KeyboardEvent* and *MouseEvent* APIs.

The *KeyboardEvent* API provides information regarding user interactions with a keyboard. Each event is triggered by an action on a key, and can have an event type of *keydown*, *keypress* or *keyup*. Table 5.1 summarises the API calls for the *KeyboardEvent* API.

The *MouseEvent* API delivers details on interactions with a pointing device, e.g. a mouse, capturing various activities — moving a mouse, clicking a button, etc. Table 5.2 outlines the API calls related to the *MouseEvent*.

Table 5.1: API calls and their properties for keyboard events

API call	**Property description**
KeyboardEvent.code	Returns a string that represents the physical key code of the key associated with the event.
KeyboardEvent.key	Returns the value of the key associated with the event.
keydown	Triggered when the key is depressed.
keyup	Triggered when the key is released.

5.3.2 Chrome extensions

Chrome extensions are programs designed to to customize the browsing experience. They enable users to modify Chrome functionality and behaviour according to individual needs or preferences. Built on web technologies, e.g. HTML, JavaScript, and CSS, these extensions can leverage browser-supported APIs to collect user data and transmit it to remote servers.

Extensions consist of various components, which can include background scripts, content scripts, a manifest file and option pages. We next briefly review these component types.

The manifest file provides information about the extension. Typically the file *manifest.json* contains general information about the extension (name, version, etc.), its permissions (download, URLs, etc.) and settings for option pages.

Background scripts can work with or without a background HTML file. A background page is loaded when it is needed and unloaded when it becomes idle. It can communicate with content scripts and popup pages by sending messages and listening for an event.

By using the DOM, content scripts can read details of web pages visited by the browser, make changes to them and pass information to other script pages. Working in an isolated environment, a content script is able to change the JavaScript environment of a loaded page without conflicting with the page or other scripts. It can be regarded as a part of the loaded page; however, a content script is not permitted to access any variables or functions created by the web page. The *Cross-Origin XMLHttpRequest* policy prevents a content script from directly sending a *XMLHttpRequest* to a remote server. Instead, message passing can be used to pass messages from a content script to the background script.

5.4 The experimental platform

This experimental platform is to provide an approach that collects the user's keystroke and mouse dynamics data in a fully unconstrained way. An extension

Table 5.2: API calls and their properties for mouse events

API call	**Property description**
click	Triggered when the mouse button is pressed and released on a single element.
mousemove	Triggered when the pointer moves over an element.
mousedown	Triggered when the mouse button is pressed on an element.
mouseup	Triggered when the mouse button is released on an element.
wheel	Triggered when the mouse wheel is crolling over an element.
pageX	Returns the X coordinate of the mouse pointer relative to the whole document.
pageY	Returns the Y coordinate of the mouse pointer relative to the whole document.
screenX	Returns the X coordinate of the mouse pointer in global (screen) coordinates.
screenY	Returns the Y coordinate of the mouse pointer in global (screen) coordinates.
clientX	Returns the X coordinate of the mouse pointer in the applications' client area. It changes when the page is scrolled.
clientY	Returns the Y coordinate of the mouse pointer in the applications' client area. It changes when the page is scrolled.

is able to run in the browser without any influence to the browser users. A Chrome extension for collecting user's keystroke data and mouse movement data was developed, and a remote server for receiving and storing data was established. We next describe their development and operation. The structure of the experimental platform is shown in Figure 5.1.

5.4.1 Development environment

Software development was performed on Windows 10 system, with details as follows.

- **Programming tool**: Sublime Text (Version 3.1.1, Build 3176).
- **Programming languages:** HTML and JavaScript (client-side) and ASP (server-side).
- **Google Chrome version:** 68.0.3440.106 (Official Build).

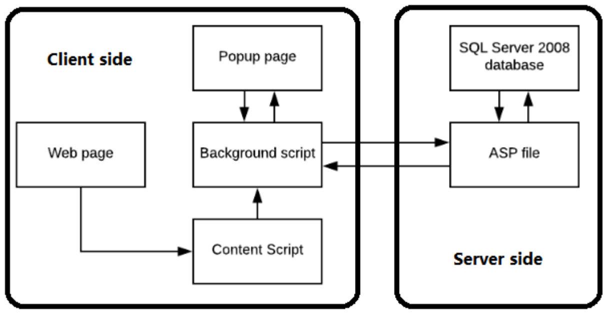


Figure 5.1: The structure of the experimental platform

To set up the server, an Alibaba Cloud Web hosting service was used ,with supporting the ASP language and a 1GB SQL database.

5.4.2 The Chrome extension

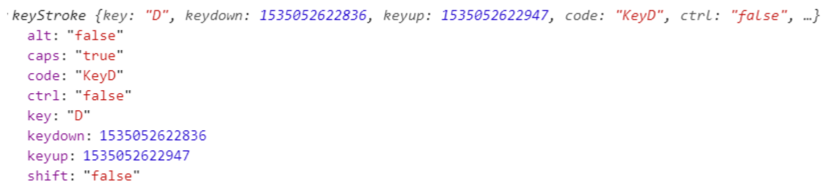
The extension consists of a manifest.json file, a popup HTML page, a popup.js script file, a background.js script file and a content.js script file. Upon clicking the extension icon, the popup page appears, prompting users to register and log in. After the user logs in, the content script starts to monitor web pages visited by the user. Both *KeyboardEvent* and *MouseEvent* APIs are used to track the user's keyboard and mouse actions. Since content scripts cannot directly communicate with a remote server, the *sendMessage* method is called to pass the parameters from the content script to the background script. When the background script receives the data, the *XMLHttpRequest* method is used to communicate with the server.

Figure 5.1 illustrates how the content script captures keystroke and mouse data using browser APIs before forwarding this data to the background script. The popup page interacts with the background script, updating the user's login status. Permissions specified in *manifest.json* allow the extension to access URLs and use cookies.

Content scripts can read the DOM of a web page. In this extension, the content script has six event listeners: *mousemove, mousedown, mouseup, wheel, keydown* and *keyup*. Each event activation generates a timestamp, captures the corresponding data, and sends it to the background script.

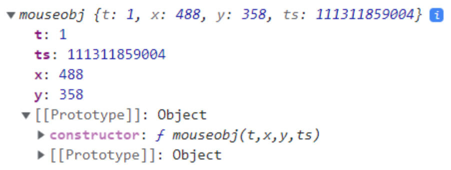
Data is categorized into two action types: keystroke and mouse. For a single keystroke, the data is regarded as an object with eight elements: keystroke timestamps, key value, and the usage of functional keys (Ctrl key, Alt key, Shift key and CapsLock key), as shown in Figure 5.2. When a key is depressed, an object of type keystroke with eight elements is created. When the key is released, the timestamps are updated and the data is sent to the background script.

Each single mouse action data consists of one of eight mouse action types



```
keyStroke {key: "D", keydown: 1535052622836, keyup: 1535052622947, code: "KeyD", ctrl: "false", …}
  alt: "false"
  caps: "true"
  code: "KeyD"
  ctrl: "false"
  key: "D"
  keydown: 1535052622836
  keyup: 1535052622947
  shift: "false"
```

Figure 5.2: Keystroke raw data at client side



```
▼mouseobj {t: 1, x: 488, y: 358, ts: 111311859004}
   t: 1
   ts: 111311859004
   x: 488
   y: 358
  ▼[[Prototype]]: Object
   ▶constructor: ƒ mouseobj(t,x,y,ts)
   ▶[[Prototype]]: Object
```

Figure 5.3: Mouse raw data at client side

(mouse move, left button down, left button up, right button down, right button up, wheel rolls, wheel down and wheel up), mouse coordinates (X and Y) and the timestamp, as shown in Figure 5.3.

The background script receives the request message from content script and popup script with the following code:

```
chrome.runtime.onMessage.addListener(
  function(request, sender, sendResponse){
    if (request.type == action type ){
      SendToServer({
        data: request.data,
        type: request.type,})}})
```

The primary role of the background script is communicating with other scripts and the server. When the background script receives a message from another script, an *XMLHttpRequest* is established to send the data to remote server:

```
xmlhttp.open("GET", URL+data, TRUE)
xmlhttp.send()
```

Users can complete registration and login via the popup page, and subsequent login status is passed to the server through the background script.

Since the extension records all keystroke and mouse movements, it may inadvertently collect sensitive personal data, e.g. passwords for certain websites, bank card number, date of birth. Thus if a user has any concerns that the data they are typing is possibly sensitive, the user is recommended to log out by clicking the 'logout' button; the user can log in again when the sensitive interactions have been completed.

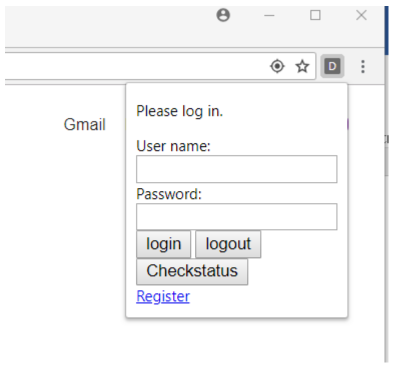


Figure 5.4: Popup page user interface

5.4.3 The server

At the server side, *ASP* files are used to interact with the client and database. The data is stored in a SQL Server 2008 database. To prevent potential SQL injection attacks, a regular expression check is used when data is received. To connect with the SQL server database, the *ADODB.CONNECTION* method is used in a ASP file:

```
Set Conn=server.CreateObject("ADODB.CONNECTION")
StrConn=" Provider=SQLOLEDB;Data Source= domain;User ID=
    userID;Password= pwd;Initial Catalog= databaseName"
Conn.open StrConn
```

When registration is completed, relevant data is then sent to the server. To distinguish data sent from different subjects, the server creates a cookie for each user:

```
response.cookies("uname") = username
```

When a user logs out, the server deletes the cookie. The server creates two databases for each user, one to store keystroke data and the other to store mouse use data.

5.5 Ethical issues

The use of this experimental platform to gather keystroke and mouse use data involves certain ethical issues. In particular, since all key depressions are monitored and recorded, all the data typed into the browser by each user, including passwords and other sensitive data, is recorded by the server.

As a result, before a participant installs the browser extension, he/she is given a written statement of what data is gathered and how it will be treated, which the participant is required to sign to give informed consent. All data gathered in this experiment will be held on a server to which only I have access, and the data will only be processed automatically for the purposes of performing the necessary processing. Unfortunately, this means that the dataset can never be made public, at least unless it has been very carefully processed first (e.g. to leave only timing information).

A formal description of the proposed experiments and the ethical issues involved, has been approved by the ISG, see Figures A.1, A.2, A.3 and A.4 in Appendix A. Consent to perform the experiments has been obtained before large-scale data gathering was performed, see Figure B.1 in Appendix B.

5.6 User guidance

This section introduces the user guidance of using the extension. This extension is not available on Chrome Extension Store because it may gather user sensitive and privacy information. Thus, instead of installing from the Stroe, the user needs to manually install step by step. The installation procedure is introduced as follows.

- Step 1: Click the 'three dots' icon (Customise and control Google Chrome) at the right above of Chrome browser, go to More tools and then go to Extensions.
- Step 2: Turn on the 'Developer mode'.
- Step 3: Click 'Load unpacked' and find the location of the extension file, then click OK.

Once installation is completed, the extension will show on the browser. To make the data collection start, the user need to register and log in. Once the user logs in, the extension will automatically record every keystroke and mouse movement data and send them to the server when the user is interacting with the browser.

The extension is publicly available on `https://github.com/fanzhaoyi/DataCollector`. It is completely free to anyone who would like to conduct and/or participate in such experiments. Due to the ethical issues, the dataset cannot be publicly available. It will only be accessed by the experimental conductor. For any concerns of this platform, the author can be reached at `zhaoyi.fan.2016@live.rhul.ac.uk`.

5.7 Possible future work

As illustrated in Chapter 4, the keystroke and mouse dynamics can be used in behavioural biometrics systems for user authentication. The prior work has demonstrated promising performance outcomes. The *KeyboardEvent* and *Mouse* can effectively collect data that could be potentially used in such biometric analysis. Additionally, keystroke and mouse dynamics do not need extra devices or sensors — only keyboard and mouse which are probably the most commonly used input devices on a personal computer. Beyond the *KeyboardEvent* and *Mouse* APIs, alternative APIs may lack the same level of efficacy and robust functionality for collecting user data from keyboard and mouse interactions. Consequently, *KeyboardEvent* and *Mouse* was selected as the primary data gathering APIs.

The experimental platform is currently capable of gathering keyboard event and mouse event data. However, there may be other behaviour-related (and hence privacy-sensitive) data that could be gathered by the platform. For example, motion sensor data for a mobile device is accessible via motion sensor APIs. Extensions to the platform are envisaged which would gather such data.

5.8 Conclusions

This chapter described the functioning and use of an experimental platform designed to gather user-related data from a browser. The platform has been developed to support experiments aimed at understanding the degree to which individual users can be identified using browser-gathered behavioural data. It is hoped that the platform will be of value to other researchers performing similar experiments.

Section 5.2 provides the motivation of this work and Section 5.3 introduces the browser APIs used by the experimental platform. Section 5.4 provides details of the implementation. Section 5.5 explains the possible ethical issues. Section 5.6 provides the user guidance of the experimental platform. Section 5.7 describes possible further development of the platform.

Chapter 6

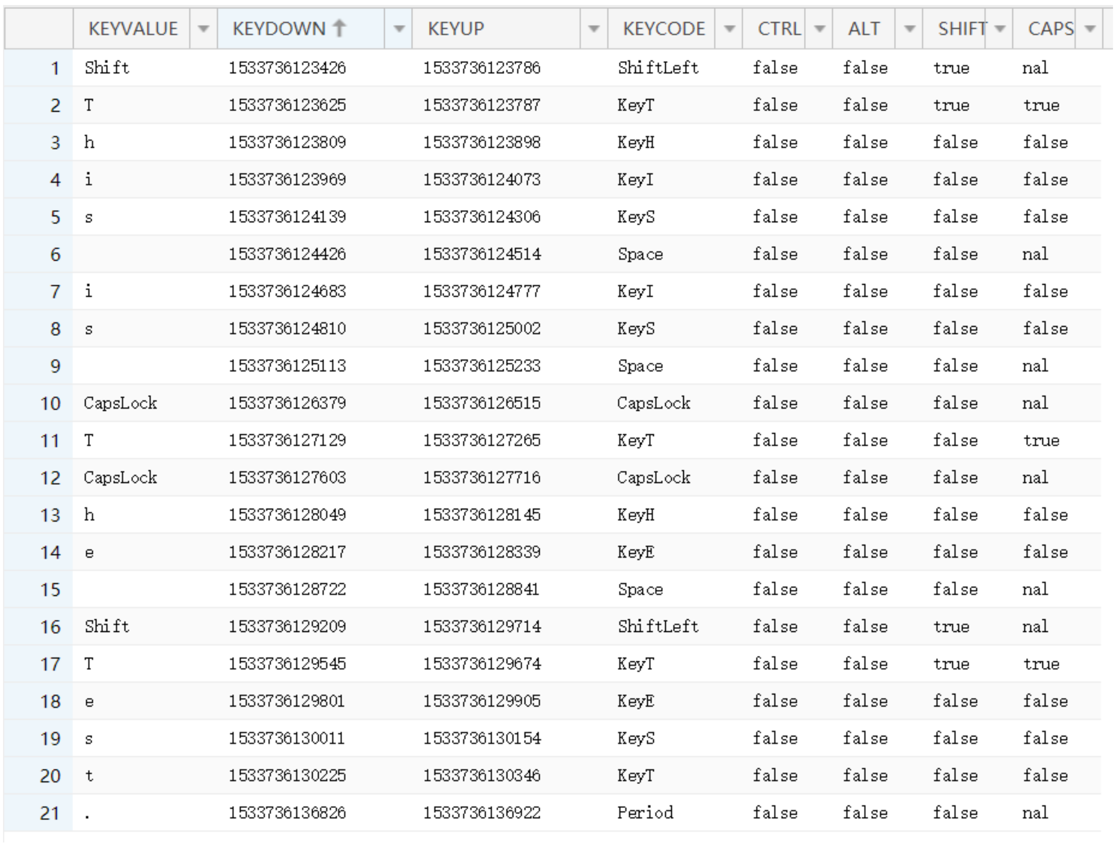
Performing the keystroke dynamics experiments

6.1 Introduction

Keystroke biometrics has been studied for decades since the 1990s [13]. It involves measuring behaviour that attempts to identify users by analysing their typing rhythm patterns, e.g. the duration of a key hold, the latency between two keystrokes, average typing speed, typing error, typing preference and keystroke pressure [13, 47]. Most prior work focused on the user authentication process based on keystroke dynamics. As far as the author is aware, none of the previous works apply their studies on a user identification process through keystroke dynamics data obtained via browser APIs in a browser-based environment. The rich features of the Browser APIs can also provide sufficient data for such identification analysis. This observation motivates the work in this chapter.

In this chapter, we set up experiments to collect user keystroke dynamics data. The goal of the experiments is to use part of the collected keystroke dynamics data as the training data to develop a biometric template for each user, and then to use the remaining data as the testing data to see how well a user can be identified. We analysed the keystroke dynamics data with machine learning techniques. Python was selected as the primary language for data analysis. The source code used in this chapter is listed in Appendix D.3.1. The experimental results indicate that the web users could potentially be identified via their typing habits in a web environment.

The structure of this chapter is organised as follows. Section 6.2 introduces the data collection. Section 6.3 introduces the data preparing and model building processes. Section 6.4 describes the experimental results based on the collected dataset and public dataset with machine learning techniques. Section 6.5 provides the discussions based on the experimental results and concludes this chapter.



	KEYVALUE	KEYDOWN	KEYUP	KEYCODE	CTRL	ALT	SHIFT	CAPS
1	Shift	1533736123426	1533736123786	ShiftLeft	false	false	true	nal
2	T	1533736123625	1533736123787	KeyT	false	false	true	true
3	h	1533736123809	1533736123898	KeyH	false	false	false	false
4	i	1533736123969	1533736124073	KeyI	false	false	false	false
5	s	1533736124139	1533736124306	KeyS	false	false	false	false
6		1533736124426	1533736124514	Space	false	false	false	nal
7	i	1533736124683	1533736124777	KeyI	false	false	false	false
8	s	1533736124810	1533736125002	KeyS	false	false	false	false
9		1533736125113	1533736125233	Space	false	false	false	nal
10	CapsLock	1533736126379	1533736126515	CapsLock	false	false	false	nal
11	T	1533736127129	1533736127265	KeyT	false	false	false	true
12	CapsLock	1533736127603	1533736127716	CapsLock	false	false	false	nal
13	h	1533736128049	1533736128145	KeyH	false	false	false	false
14	e	1533736128217	1533736128339	KeyE	false	false	false	false
15		1533736128722	1533736128841	Space	false	false	false	nal
16	Shift	1533736129209	1533736129714	ShiftLeft	false	false	true	nal
17	T	1533736129545	1533736129674	KeyT	false	false	true	true
18	e	1533736129801	1533736129905	KeyE	false	false	false	false
19	s	1533736130011	1533736130154	KeyS	false	false	false	false
20	t	1533736130225	1533736130346	KeyT	false	false	false	false
21	.	1533736136826	1533736136922	Period	false	false	false	nal

Figure 6.1: Keystroke raw data

6.2 Data gathering

In our experiments, we invited 20 users. Each user used the experimental tool for a period of at least one month. All users in the experiments are bilingual — Chinese (native level) and English (proficient level). The experimental environment was free of controls. They could type any of these two languages at any time and were totally free to stop the extension by clicking logout whenever they wanted.

One user's data was separated from the other 20 users as the data size was too small. This user's data was used for initial data analysis. The data of the rest 20 users were used for further training and testing. Figure 6.1 illustrates the keystroke raw data stored in the database. The data is ordered ascendantly by *keydown* timestamp. Each line of the data represents a single keystroke with 8 elements as described in Table 6.1. In our experiments, five elements were selected for further analysis, i.e., *keycode* (K), *keydown* (KD), *keyup* (KU), *Caps* and *Shift*. For each keystroke, the *keyup* value should always be greater than

Table 6.1: Description of the eight elements in keystroke raw data

Element	**Explanation**
KEYVALUE	Key value generated by the keystroke
KEYDOWN	Timestamp when the key is pressed down, denoted as D
KEYUP	Timestamp when the key is released, denoted as U
KEYCODE	Physical key position on the keyboard, denoted as K
CTRL	Usage of the *Ctrl* key during the keystroke; 'false' indicates not in use, 'true' indicates in use, denoted as $Ctrl$.
ALT	Usage of the *Alt* key during the keystroke; 'false' indicates not in use, 'true' indicates in use, denoted as Alt.
SHIFT	Usage of the *Shift* key during the keystroke; 'false' indicates not in use, 'true' indicates in use, denoted as $Shift$.
CAPS	State of the *CapsLock* key during the keystroke; 'false' indicates not active, 'true' indicates active, 'nal' indicates the key is not one of the 26 English letters, denoted as $Caps$.

its *keydown* value and the *keydown* value of its next keystroke would always be greater than its *keydown* value due to the natural data gathering process. It should be noted that, the *keyup* values of two consecutive keystrokes may not have an absolute relationship. The number of keystrokes for each user in raw data ranges from 7816 to 33212, as shown in Figure 6.2.

6.3 Data processing

Before using the collected data, it is essential to process it to extract the relevant features. As examples, we next describe possible approaches to such processing, used in experiments that were performed with the aid of the platform we introduced in Chapter 5.

Collected data can be downloaded into the local machine and processed with Python packages (*sklearn* [67]). To avoid potential data leakage and to simulate the actual identification process, the raw data was simply divided into two parts: first 80% of the data gathered for each user were used for generating a biometric classifier, and the remaining 20% of the gathered data for each user were used for further user identification.

Figure 6.3 shows the number of keycodes used for each user. The differences may result from user occupations, daily routines, keyboard layouts or operating

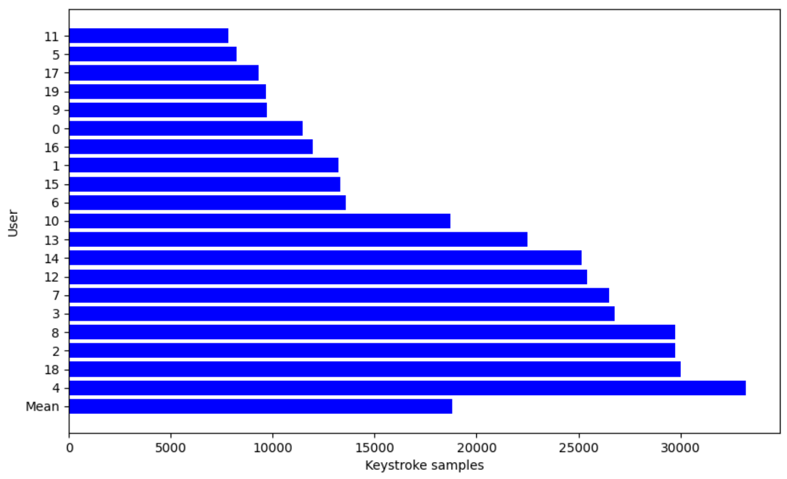


Figure 6.2: Number of keystrokes from each user's raw data

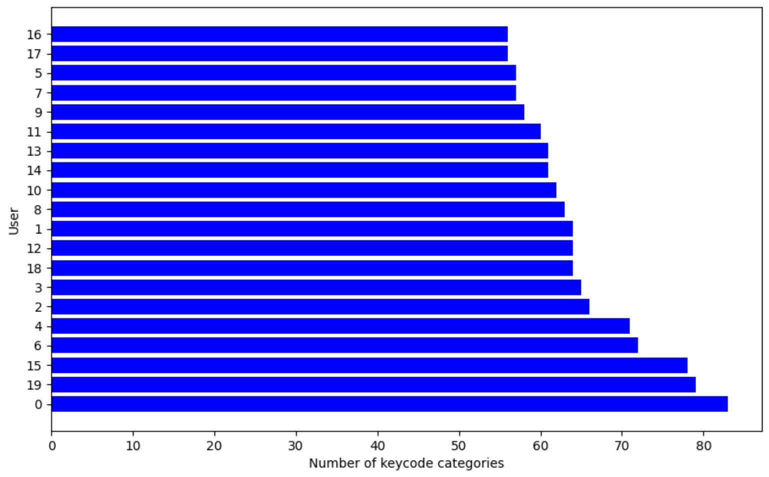


Figure 6.3: Distribution of keycode categories for each user in the raw dataset

systems etc. For example, a standard PC keyboard with US-international or UK layout does not have *Volume* control key but has a keypad area at the right side, while keyboard on a 13-inch MacBook Pro has the *Volume* control key but does not have the keypad area. These differences could potentially reveal a user's device information but not user behaviour. Thus, only the keys used by all users were selected in this experiment. Table 6.2 shows the occurrence of top 10 frequently used keys for each user. The *Space* and *Key I* keys are the two most frequently used keys and the five vowels (A,E,I,O,U) are in the top 10.

Table 6.2: Occurrence of the Top 10 Most Frequently Used Keys for Each User

User	**Space**	**I**	**A**	**N**	**Backs**	**E**	**U**	**H**	**O**	**G**	**Enter**
0	884	504	687	658	638	407	388	363	424	352	209
1	821	635	425	579	1287	638	206	185	254	166	601
10	1879	1869	1634	1464	869	1074	980	970	917	808	451
11	485	390	368	416	601	270	239	194	194	186	144
12	2670	2355	2092	1907	1328	1535	1328	1248	1278	1114	538
13	2446	2167	1913	1758	1135	1482	1270	1257	1132	969	496
14	2556	2456	2090	2005	1285	1622	1259	1300	1159	1153	534
15	1120	725	877	634	1022	566	463	428	475	258	438
16	1255	679	675	711	822	518	388	410	404	378	584
17	1094	533	471	521	739	440	283	320	290	241	431
18	3095	2665	2312	2182	1500	1877	1404	1551	1405	1127	645
19	573	422	438	415	721	445	296	238	388	226	431
2	3307	2928	2559	2287	1427	1917	1710	1493	1422	1158	676
3	2943	2644	2239	2033	1471	1806	1439	1345	1231	1040	508
4	3297	3110	2690	2490	1772	2068	1883	1663	1588	1301	805
5	616	521	527	531	515	342	231	301	350	270	375
6	947	524	639	470	753	543	293	270	336	197	291
7	2918	2563	2179	2144	1388	1614	1482	1458	1274	1137	553
8	3123	2843	2416	2390	1510	1823	1570	1451	1403	1351	667
9	532	494	510	547	852	471	307	355	340	289	484
Total	36561	31027	27741	26142	21635	21458	17419	16800	16264	13721	9861

6.3.1 Data segmentation

The main method of distinguishing users via their keystroke behaviours is to find out their typing rhythms. A single keystroke is not regarded as a standalone sample, as it may be relative to its preceding or subsequent keystroke, thus the data needs to be split into several multi-keystroke segments. Araujo et al. [13] suggested that more than 98% of the latencies between two consecutive keys are less than one second. Therefore, we set one second as the threshold for segmenting data in our experiments. That is, in each segment, the DD time difference between two consecutive keystrokes is less than 1,000 (milliseconds). The *Train_test_split* method was used to split the segments into training and testing sets for model evaluation. It should be noted that one segment may not be regarded as a sample, as it may contain a number of keystrokes. The total number of segments for all users is 30,491 and the average length for all segments is 9.5, as shown in Table 6.3.

Table 6.3: Segmentation Statistics for Each User

User	Number of segments	Minimal length	Maximum length	Average length
0	1439	1	95	6.3
1	1977	1	50	4.8
10	669	1	152	22.0
11	723	1	55	7.3
12	2206	1	103	9.1
13	1138	1	126	15.7
14	1333	1	157	15.0
15	1882	1	82	5.4
16	2663	1	46	3.5
17	1552	1	65	4.3
18	1551	1	173	15.3
19	963	1	80	7.5
2	1755	1	110	13.4
3	1893	1	79	11.2
4	1458	1	159	17.8
5	1384	1	57	4.4
6	1651	1	70	4.4
7	1198	1	141	17.5
8	1681	1	203	13.9
9	1375	1	68	5.5
9	1375	1	68	5.5
Total	30491	20	203	9.5

6.3.2 Feature extraction

This section introduces the feature extraction process. The features can be divided into two categories: typing rhythms based on timing information, and typing habits based of functional key (*Ctrl, Alt, Shift, CapsLock*) use. Digraph was used as the experimental samples in majority of prior work. A common digraph could exist in many different scenarios. For example, the digraph *en* could be in word *pen* or word *end*. This situation also may apply to 3-graph — *pen* could be in *pen*, *pencil*, *happen* or *open*. The typing rhythm of *pen* would probably be different while a user is typing in such four words. Tables (C.1 and C.2), Tables (C.3 and C.4), Tables (C.5 and C.6) and Tables (C.7 and C.8) (see appendix) describe the most common used digraphs, 3-graphs, 4-graphs and 5-graphs for the users, with a total number of 211, 87, 20 and 5 respectively. It should be noted that there are no 6-graphs being used by all uses at the same time. In this research, 4-graph was used as the experimental samples for feature extraction for the following reasons: it has various types and includes a range of frequently used digraphs and 3-graphs.

Each 4-graph sample consists of 4 keystrokes. Each keystroke consists of 4 elements and can be described as $S = \{T, F\}$ where T and F represents 28 timing features and 8 features of function key usage respectively. Timing features T can be divided into 5 groups, as described as follows.

- **Duration**: time elapsed from one key pressed to released.
- **DD time**: time elapsed from one key pressed to its followed keys pressed.
- **DU time**: time elapsed from one key pressed to its followed keys released.

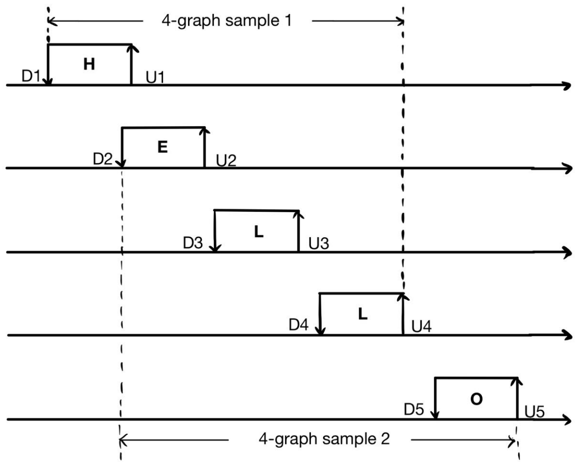


Figure 6.4: Keystroke Data Extraction for the Word 'HELLO'

- **UU time**: time elapsed from one key released to its followed keys released.
- **UD time**: time elapsed from one key released to its followed keys pressed.

F features can be described as: $\{S, C\}$ where S represents the *Shift* key usage and C represents the *CapsLock* key usage. For a given 4-graph, 4 *Duration* features, 6 DD features, 6 DU features, 6 UU features, 6 UD features, 4 S features and 4 C features can be extracted.

One segment may consist of more than 4 keystrokes. Figure 6.4 illustrates the feature extracting process for word *HELLO*. Two 4-graphs can be extracted, i.e., *HELL* and *ELLO*. Although both *HELL* and *ELLO* has a overlap of 3-graph (*ELL*), the position of *ELL* in each 4-graph sample are different.

6.3.3 Evaluating metrics

The main area for this research is multi-class identification rather than binary-class authentication. Thus, f1-score, precision score, recall score and confusion matrix were used.

6.3.4 Classification methods

The machine learning algorithms presented in 6.4 have demonstrated good performance in various scenarios [57] [42]. Determining the most effective algorithm for our study from existing literature is challenging due to variations in sample characteristics and experimental methodologies. Therefore, we determined our classifier choice through testing results, consistent with the approach utilised in Chapters 7 and 9. We tested various machine learning algorithms, e.g., SVMs,

Table 6.4: Comparative Performance Metrics of Various Classifiers on Keystroke Dynamics Data

Classifier	**F1 score**	**Precision**	**Recall**	**ACC**	**Time**
XGBoost	0.440	0.487	0.441	0.525	109s
Random Forest	0.355	0.412	0.408	0.357	1188s
LinearSVC	0.111	0.145	0.156	0.252	5755s
SVC(linear kernel)	0.181	0.251	0.200	0.307	12891s
SVC(rbf kernel)	0.257	0.351	0.270	0.374	9323s

Random Forest and eXtreme Gradient Tree Boosting (XGBoost). For SVMs, three different approaches were tested — *LinearSVC* (One vs Rest), *SVC* (One vs One) with linear kernel, *SVC* (One vs One) with rbf kernel were selected. Each classifier was set by using default hyper-parameters and *10-fold* cross validation was used. As is shown in Table 6.4, the XGBoost classifier achieved both the highest scores and lowest running time. Although tuning the hyper-parameters could change the performance of a classifier, the significant difference in performance led to the selection of the XGBoost classifier as the classification method in this approach. The data analysis was processed on a PC with an AMD 7800X3D CPU and an RTX 4090 GPU.

6.3.5 Feature selection

Figure 6.5 shows the scoring for each feature. The use of *Shift* key (f32, f33, f34, f35) gains more weights than other features. The results indicate that the user habit of typing uppercase letters, i.e., some user may choose use *CapsLock* while others may choose to hold *Shift* key, may be an interesting feature to distinguish users.

6.3.6 Model evaluation

Table 6.5: Best Hyper-parameters of XGBoost Classifier on Keystroke Dynamics

Hyper-parameter	**Value**
objective	*multi:softprob*
eval_metric	*mlogloss*
tree_method	*gpu_hist*
learning_rate	*0.3*
n_estimators	*600*
max_depth	*5*
min_child_weight	*5*

The *Grid search* method was used for tuning hyper-parameters with *10-fold* cross validation. Table 6.5 shows the tuned hyper-parameters. Additionally, *sample_weight* was used to solve sample imbalance. Figure 6.6 shows the confusion matrix of classification result with tuned hyper-parameters on one 4-graph sample.

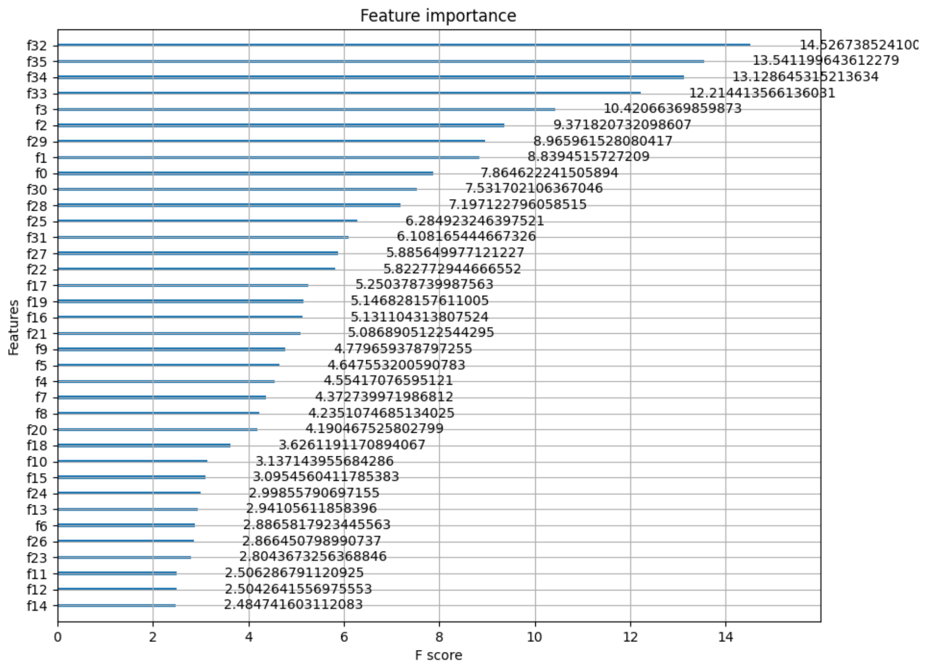


Figure 6.5: Feature importance scores obtained from the XGBoost model for keystroke dynamics analysis

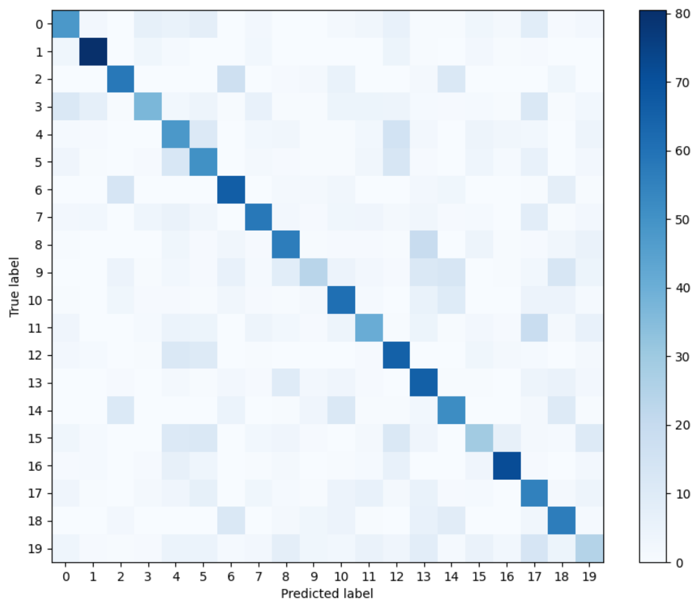


Figure 6.6: Confusion matrix representing the classification results for a single 4-graph keystroke sample using the tuned XGBoost model

Table 6.6: Classification Scores on Keystroke Training Set from 1 to 30 Samples

Samples	**F1 score**	**Precision**	**Recall**	**ACC**
1	0.514	0.510	0.523	0.559
2	0.575	0.575	0.581	0.622
3	0.622	0.630	0.624	0.671
4	0.663	0.678	0.661	0.712
5	0.700	0.722	0.695	0.749
6	0.730	0.758	0.722	0.779
7	0.756	0.790	0.746	0.805
8	0.779	0.816	0.766	0.826
9	0.797	0.836	0.784	0.844
10	0.814	0.854	0.799	0.860
11	0.831	0.872	0.815	0.874
12	0.845	0.887	0.829	0.886
13	0.857	0.898	0.840	0.896
14	0.868	0.909	0.850	0.906
15	0.878	0.918	0.860	0.914
16	0.885	0.926	0.867	0.921
17	0.892	0.933	0.874	0.927
18	0.899	0.939	0.880	0.932
19	0.905	0.944	0.886	0.937
20	0.910	0.948	0.891	0.942
21	0.915	0.951	0.896	0.945
22	0.919	0.954	0.900	0.948
23	0.923	0.958	0.905	0.951
24	0.926	0.960	0.908	0.953
25	0.929	0.963	0.911	0.956
26	0.932	0.965	0.914	0.958
27	0.935	0.967	0.918	0.960
28	0.937	0.969	0.920	0.962
29	0.939	0.97	0.921	0.963
30	0.941	0.971	0.924	0.964

6.3.7 Accumulative methods

When a user is continuously typing with a keyboard, more actions are generated and more data can be gathered. For each 4-graph sample S, the predicting result can be computed in a probability way with *predict_proba* and described as $\{p_1, p_2, ..., p_n\}$ where p_i represents the predicting probability of this action belonging to $user_i$. Thus, the probability prediction of m 4-graph samples $\{s_1, s_2, ..., s_m\}$ from n users can be described as:

$$P_{mn} = \begin{Bmatrix} p_{11} & \cdots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{m1} & \cdots & p_{mn} \end{Bmatrix} = \{\frac{1}{m}\sum_{j=1}^{m} p_{j1}, ..., \frac{1}{m}\sum_{j=1}^{m} p_{jn}\} \tag{6.1}$$

Hence, the user corresponding to the maximum probability value in P_{mn} is identified as the prediction result. As shown in Table 6.6 , the classification performance improves consistently with the increment of keystroke samples.

Apart from using 4-graph, other n-graphs were also tested, as shown in Figure 6.7. We concluded that the classification result increases consistently for each n-graph situation; for a given series of continuous keystrokes, the 4-graph achieves the best result in most cases.

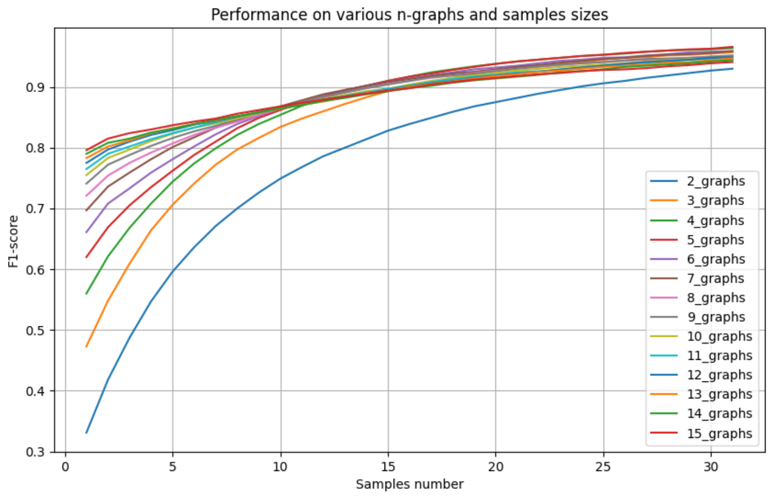


Figure 6.7: Comparison of F1-scores for 1 to 15-graphs across 1 to 30 sample increments, highlighting the optimal performance of 4-graph samples

6.4 Experimental results

In Section 6.3, we described the implementation of the *XGBoost* classifier on the training set. In this section, we discuss the performance of our well-tuned models on our testing dataset and on another public dataset.

6.4.1 Results on real testing set

As previously mentioned, the raw data were separated into two sets. The first 80% of the raw data was used as the training set for model setup and the remaining 20% of the raw data was used for testing the established model. Table 6.7 displays the number of keystrokes for each user in the testing set.

Table 6.7: Keystroke Counts per User in the Testing Set

User	**Raw**	**User**	**Raw**	**User**	**Raw**
0	2298	**7**	5304	**14**	5036
1	2646	**8**	5947	**15**	2669
2	5947	**9**	1941	**16**	2393
3	5357	**10**	3749	**17**	1863
4	6644	**11**	1565	**18**	6000
5	1647	**12**	5089	**19**	1938
6	2723	**13**	4500	**Total**	75256

All data in the testing set were extracted into several 4-graph samples. The testing samples were not shuffled but retained the same order as when they were collected. This approach simulates the actual scenario where a user's keystroke

Table 6.8: Classification Performance Metrics for Keystroke Samples by Sample Size on the Testing Set

Samples	F1 score	Precision	Recall	ACC
1	0.496	0.496	0.509	0.545
2	0.559	0.559	0.570	0.610
3	0.606	0.610	0.613	0.659
4	0.648	0.657	0.651	0.701
5	0.684	0.699	0.683	0.737
6	0.715	0.735	0.711	0.768
7	0.743	0.766	0.737	0.795
8	0.766	0.791	0.757	0.816
9	0.786	0.815	0.776	0.835
10	0.802	0.834	0.789	0.849
11	0.817	0.850	0.803	0.863
12	0.829	0.864	0.814	0.873
13	0.840	0.876	0.825	0.884
14	0.850	0.886	0.834	0.893
15	0.859	0.894	0.842	0.900
16	0.867	0.903	0.850	0.907
17	0.874	0.910	0.856	0.913
18	0.881	0.916	0.863	0.918
19	0.887	0.922	0.869	0.923
20	0.892	0.927	0.874	0.927
21	0.897	0.930	0.879	0.930
22	0.902	0.935	0.884	0.934
23	0.906	0.938	0.889	0.937
24	0.909	0.941	0.892	0.940
25	0.913	0.944	0.896	0.942
26	0.916	0.947	0.899	0.944
27	0.919	0.949	0.903	0.947
28	0.922	0.952	0.905	0.948
29	0.924	0.954	0.907	0.950
30	0.926	0.955	0.909	0.952

information is continuously gathered. Table 6.8 shows the classification results, i.e., continuously increasing from 1 4-graph sample to 30-graph samples. Figure 6.8 illustrates the confusion matrix for all users on one single 4-graph sample.

6.4.2 Results on fixed text dataset

We also conducted tests on another public dataset. Killourhy and Maxion [49] provided a keystroke dataset based on a fixed text of '**.tie5Roanl**'. It is a strong password with 10 characters, containing both uppercase and lowercase letters, numbers and punctuation. 51 participants were invited in this data collection. They were asked to type this password correctly 400 times in at least 2 weeks. This approach was intended to minimize the proficiency bias that might occur if a user types the same password within a very short period.

The dataset comprises 31 timing features for each sample. We used *train_test_split* method to partition the data into a training set (80%) and a testing set (20%). A variety of machine learning algorithms were evaluated, all of which achieved commendable classification performance, as detailed in Table 6.9.

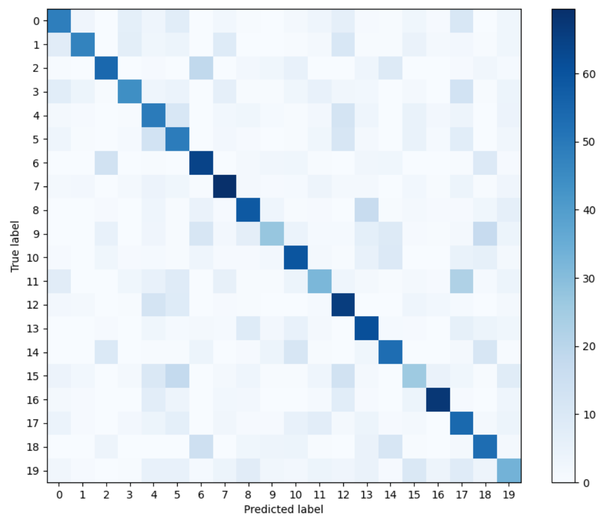


Figure 6.8: Confusion Matrix for a Single 4-Graph Keystroke Sample Classification

Table 6.9: Classification Performance on the DSL Password Dataset

Classifiers	**Hyper-parameters**	**F1-score**	**Precision**	**Recall**	**ACC**
XGBClassifier	default	0.921	0.922	0.921	0.921
RandomForest	default	0.924	0.926	0.924	0.924
SVC	*kernel*:linear	0.836	0.837	0.837	0.837
SVC	*kernel*:rbf	0.870	0.875	0.869	0.869
k-NN	*k*:5, *metric*:manhattan	0.895	0.901	0.895	0.895
LightGBM	default	0.935	0.936	0.935	0.935

6.4.3 Comparisons

Despite the differences in experimental approaches from those used in prior work, it remains significant to conduct a comparison, as depicted in Table 7.11. The prior studies discussed in this chapter were primarily focused on authentication tasks, with the exception of the research by Kasprowski et al. [48], which addressed both authentication and identification tasks. While none of the earlier studies employed browser APIs for data collection, an exception is found in the research by Earl et al. [27]. However, their experimental environment was highly controlled and they combined keystroke and mouse dynamics data, which will be discussed in Chapter 8. To the best of our knowledge, the study presented in this chapter is the first to describe the identification of users through keystroke dynamics data obtained via browser APIs in a completely uncontrolled browser-based environment. A summary of prior work is as follows.

A comparison with prior work is described in Table 6.10.

Kasprowski et al. [48] conducted their research based on public dataset — the Buffalo dataset [82]. This dataset consists of both fixed length of input and free text from 148 participants. They used ANN as the classification methods. They achieved an accuracy of 82% for identification task with 20 users selected.

Lu et al. [56] used the Buffalo dataset [82] and the Clarkson dataset [89] in their study. A total of 39 participants contributed to the data collection, with a mixture of passwords, fixed text, and free text across two sessions, each lasting two hours per user. On average, each participant generated approximately 21,533 keystrokes. They achieved 1.89% FRR and 2.83% FAR on the Buffalo dataset, 6.61% FRR and 5.31% FAR on the Clarkson dataset based on a sequence length at 50 for each keystrokes sample.

Altwaijry [9] invited 85 users for the data collection. Each user was asked to type in a fixed length of password and a phrase for 400 times in both English and Arabic in a controlled environment. AdaBoost achieved the best performance with accuracies of 98.9% and 99.1% on their fixed phrase dataset for Arabic and English respectively.

It should be noted that, while prior studies have demonstrated promising results in authentication tasks, their datasets and methods may not be suitable for this study. In a browser-based real-world scenario, users are unlikely to repeatedly input their passwords for a single website. Furthermore, modern browsers offer features such as password management tools, allowing users to bypass manual password input in secure environments.

Additionally, the requirement to type a phrase of a long fixed length is not applicable to the context of this study. In real-world scenarios, repeatedly typing a fixed-length phrase would challenge a user's patience and could result in an unpleasant experience. This negative condition could potentially influence the user's typing behaviour. Moreover, users rarely have the need to enter a long, fixed phrase multiple times at a single website.

In contrast, our dataset is unconstrained and is based on a browser environment, including a wide variety of real-life scenarios. That is, we believe that the dataset we gathered represents a more likely real-world scenario than has been examined previously.

6.5 Conclusions and possible future work

6.5.1 Conclusions

In this chapter, we studied the identification of browser users based on their keystroke typing information. We conducted experiments to collect user keyboard and mouse interaction data using the experimental platform described in Chapter 5. We recruited 21 bilingual users (Chinese native and English proficient) in our experiments. Each participant used the experimental tool for a period of 4 to 6 weeks without any restrictions, as the experimental environ-

Table 6.10: Comparative Overview of Keystroke Dynamics Studies Across Different Environments and Methods with Corresponding Performance Metrics

Author	**Year**	**Users**	**Environment**	**Browser APIs**	**Method**	**Task**	**Performance**
Giot et al. [37]	2011	100	controlled	No	SVM	auth	15.28% ERR
Kang et al. [46]	2015	35	controlled	No	*R* and *A* measures	auth	5.64% EER
Lu et al. [56]	2020	39	controlled	No	RF, KNN	auth	1.74% FPR,29.23% FNR
Kasprowski et al. [48]	2022	20	controlled	No	ANN	auth	82% accuracy
Altwaijry et al. [9]	2023	85	controlled	No	AdaBoost, DT, RF, SVM	auth	99.1% accuracy
This study	2023	20	uncontrolled	Yes	XGBoost	iden	0.802 F1-score

RF: Random Forest, SVM: Support Vector Machine, ANN: Artificial Neural Network, DT: Decision Tree, auth: authentication, iden: identification

ment was unrestricted. 20 users provided sufficient keystroke and mouse data for analysis in this chapter and in Chapter 7 respectively.

The keystroke data were divided into two categories: fixed text password and free text. In the fixed text experiment, *DSL-password* dataset (**.tie5Roanl**) [49] was used. The dataset contains 51 users and each user provides 400 samples. 31 timing features were extracted in each sample. We achieved an experimental result of 93.5% (f1-score).

In the free text experiments, we analysed data collected from 20 participants. Every keystroke from a user was captured and sent to the server, containing 8 elements: keycode, timestamp of key pressed, timestamp of key released, the *CapsLock*, *Ctrl*, *Alt* and *Shift* key uses. Features based on 4-graph were extracted from the raw data. The *XGBoost* classifier was selected as the classification algorithm, and the f1-score was chosen as the primary metric. We achieved a classification result based on one single 4-graph sample containing four consecutive keystrokes of 49.6% f1-score. With 20 consecutive keystrokes, we achieved an experimental result of 89.2% f1-score. The results indicate that a browser user can be identified via their typing habits without their knowledge in a browser-based environment and the identification performance could improve with more keystroke inputs within one identifying session. Another interesting finding is that the method of typing an uppercase letter, whether using the *CapsLock* key or holding the *Shift* key, can be a distinctive feature in such analysis.

6.5.2 Possible future work

There are many aspects of work that could be pursued in the future, such as recruiting more participants from various scenarios, extracting better features from raw data with sophisticated data pre-processing techniques, using better machine learning tools for classification. Additionally, the combination of function key uses might also be an interesting feature for distinguishing users.

Chapter 7

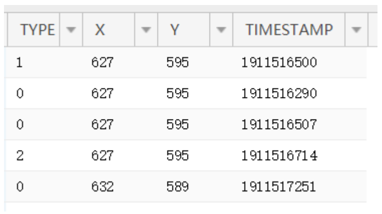
Performing the mouse dynamics experiments

7.1 Introduction

Mouse dynamics have been studied for years. A number of prior studies indicate that mouse dynamics can be used as soft biometric in an authentication system [31, 59, 60, 76]. Similar to keystroke dynamics, mouse dynamics analysis does not require a special device for data collection. A mouse device may be the primary interactive device with the computer. In a web environment, the *MouseEvent* API provided by DOM can be accessed via JavaScirpt when a web page is loaded. Such APIs can potentially be used to reveal user data via machine learning techniques.

In this chapter, we described the data analysis with machine learning techniques based on the mouse dynamics data from the same set of 20 users described in Chapter 6. We used part of the collected mouse dynamics data as the training data to develop a biometric profile for each user, and then to use the remaining data as the testing data see how well a user can be identified. The Python scripts used in this chapter are listed in Appendix D.3.2. We also applied our machine learning models on a public mouse dynamics dataset (the Bogazici dataset [52]). We achieved promising experimental results. It strongly indicates that a user can be identified via his/her mouse using habit in a web environment.

This chapter is structured as follows. Section 7.2 explains the mouse dynamics data collection with the experimental tools described in Chapter 5. Section 7.3 provides data processing with machine learning techniques in details. Section 7.4 illustrates the experimental results on our experimental dataset and public dataset and section 7.5 concludes this work.



TYPE	X	Y	TIMESTAMP
1	627	595	1911516500
0	627	595	1911516290
0	627	595	1911516507
2	627	595	1911516714
0	632	589	1911517251

Figure 7.1: Mouse dynamics raw data

7.2 Data gathering

We used the mouse dynamics data obtained from the same set of 20 participants described in Chapter 6. The extension recorded every mouse action of users. The experimental environment was free of controls. During the experiments, the users did not need to do any specific actions. Hence, there was no limits or tasks for the use of mouse. They were totally free to stop the extension by clicking logout whenever they wanted.

One participant's data was separated from the other 20 users as the data size is too small. This user's data was used for initial data analysis. The rest 20 users' data were used for further training and testing.

Figure 7.1 shows the mouse raw data. A mouse event can be triggered by a number of different action types, such as mouse movement, left button click, right button click and wheel scrolling. Each mouse event was regarded as a single object containing the following elements: type of event action, X coordinate of the mouse, Y coordinate of the mouse, and a timestamp of corresponding action, as described in Table 7.1.

Table 7.1: Explanation of Mouse Raw Data Elements

Elements	**explanation**
TYPE	type of event action
X	X coordinate of the mouse on the screen
Y	Y coordinate of the mouse on the screen
TIMESTAMP	timestamp of the corresponding action

The *TYPE* elements represents the mouse action types, as described as follows.

- **0**: mouse moving.
- **1** and **2**: left button pressed down and released respectively.
- **3** and **4**: right button pressed down and released respectively.
- **5**: mouse wheel pressed.
- **6** and **7**: mouse wheel scrolling up and down respectively.

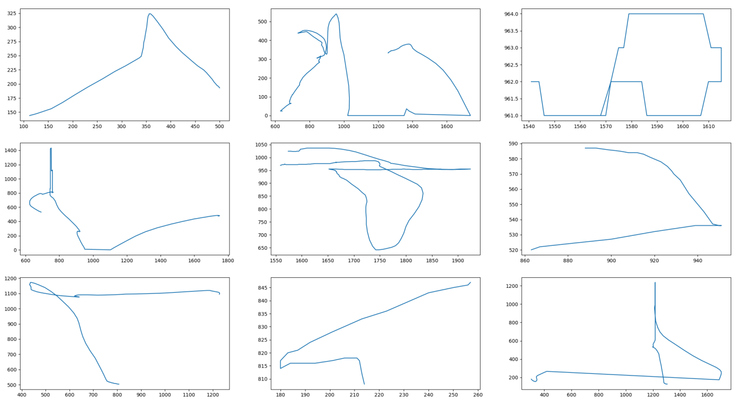


Figure 7.2: Nine random examples of mouse moving paths

- **8**: unexpected actions.

7.3 Data processing

The collected data can be downloaded into the local machine and processed with Python packages (sklearn). To prevent potential data leakage and to mimic the actual identification process, the raw data were divided into two parts: first 80% of the data gathered for each user were used for generating a biometric classifier, and the remaining 20% were used for further user identification.

7.3.1 Data segmentation

In most first-person shooter (FPS) video games, the operating behaviour (e.g. the efficiency and accuracy of aiming and firing) of players can differ due to individual proficiency. Similarly, the behaviour of web browser users, for example, moving the mouse cursor onto a specific element and clicking, can also vary. While a game player may enhance performance through rigorous training and concentration, web users are less likely to intentionally and rapidly change their mouse operation habits. Based on this observation, Point-and-Click movements were selected as the primary mouse action for analysis of such mouse behaviour for the following reasons.

- The number of left-click actions is much higher than that of right-click actions.
- Point-and-click actions contain both mouse movement and left click.

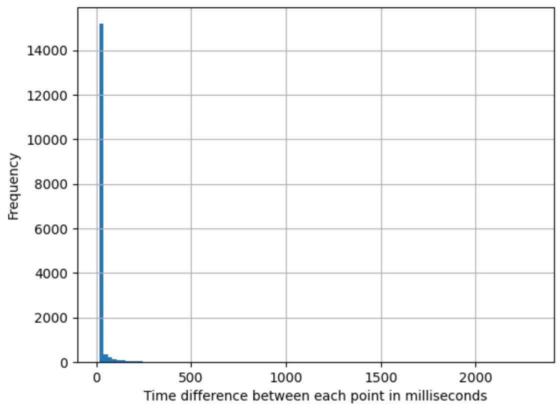


Figure 7.3: Initial time differences between points in mouse movement

- Point-and-click actions occur more frequently than other types of actions [10].

Table 4.3 lists nine types of mouse actions. To isolate point-and-click action, raw data were segmented, retaining only the points with types ‘0’, ‘1’, and ‘2’. Each segment contained all three types of points, starting with a type ‘0’ (i.e., moving action) and ending with a type ‘2’ (i.e., left button release), denoted as $S_i = [P(0), P(0), ..., P(1), ..., P(2)]$, where the subscript indicates the action type, and the number of points between $P(1)$ and the subsequent $P(2)$ can be zero or more. Segments had a timing difference between each point. To reduce noise from inactivity, the total time span from start to end was capped at 3 seconds. The nine random examples in Figure 7.2 illustrate actual mouse movement paths with x and y denoting cursor coordinates. However, such data do not directly indicate the efficiency and accuracy of a user’s point-and-click actions due to excessive noise. For example, a user aiming to move the cursor to a specific point and click will not perform random circular movements as depicted in the fifth example in Figure 7.2. Therefore, further noise reduction was necessary. Figure 7.3 displays the initial time intervals between points. As shown, most intervals are under 500 ms. The time differences correspond to various moving action types, such as smooth and continuous movement, brief pauses, or sudden changes in direction. It is important to note that the minimal interval between two consecutive points depends on the software and hardware configuration. To discern these differences in each segment, a threshold was initially set from one to five times the interquartile range (IQR), calculated as $(2.5Q_3 - 1.5Q_1)$ where Q_3 and Q_1 represent the third and first quartiles, respectively. The chosen threshold affected the number of samples extracted, as shown in Table 7.2. In practice, if the interval between two consecutive points exceeded the threshold, a breakpoint was introduced, discarding all preceding points. For our experiments, a threshold of three times the IQR was selected.

Table 7.2: Number of segments extracted at varying IQR thresholds

user	orginal segements	1 IQR	2 IQR	3 IQ3	4 IQR	5 IQR
0	4158	1471	2431	2742	2937	3071
1	4590	1523	2495	2737	2899	3003
2	5020	1774	2951	3397	3615	3751
3	4444	1305	2331	2769	3061	3298
4	9351	2750	4982	5595	6059	6319
5	3004	688	1272	1460	1665	1811
6	11710	3087	4428	4617	4724	4818
7	4966	2053	3140	3486	3730	3878
8	13968	4199	5978	6343	6550	6679
9	8911	2748	3845	4066	4185	4263
10	3495	1170	1940	2267	2434	2552
11	2726	925	1493	1651	1754	1832
12	12798	3997	5523	5846	6008	6123
13	5714	1739	2970	3318	3527	3692
14	4589	1605	2832	3093	3218	3283
15	4062	1423	2282	2570	2717	2832
16	4418	1563	2387	2720	2934	3065
17	7104	2058	3994	4520	4854	5058
18	15996	4443	6250	6586	6771	6918
19	8002	2771	3804	3989	4131	4224

7.3.2 Feature extraction

This section introduces the feature extraction process. Figure 7.4 displays nine random examples of mouse movement paths after noise reduction. For each segment, all points can be divided into two sets: the moving set as the first subset, followed by the clicking set as the second subset. All features are extracted based on four original attributes of each point—action type, x-coordinate, y-coordinate, and timestamp. A segment can be regarded as one sample and described as:

$$\mathbf{S} = \{\mathbf{M}, \{L_d, L_u\}\}$$

M refers to moving set and the points in this set can be described as the following 4 elements:

$\mathbf{a} = \{a_i\}_{i=1}^{n}$, the action type of point i.

$\mathbf{x} = \{x_i\}_{i=1}^{n}$, the x coordinate of point i.

$\mathbf{y} = \{y_i\}_{i=1}^{n}$, the y coordinate of point i.

$\mathbf{t} = \{t_i\}_{i=1}^{n}$, the timestamp of point i.

In the second subset, L_d refers to left click button pressed down point and L_u represents left button released up point. These two points can be described as:

$$L_d = \{a_d, x_d, y_d, t_d\}$$

$$L_u = \{a_u, x_u, y_u, t_u\}$$

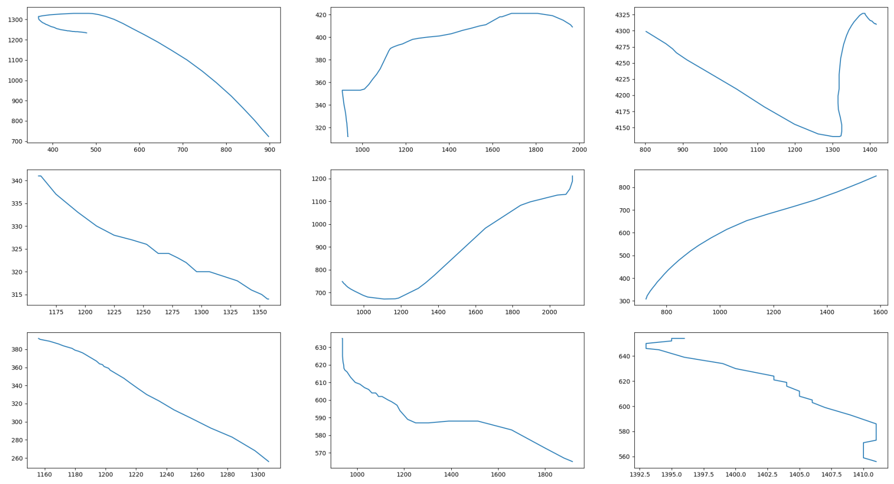


Figure 7.4: Nine random examples of refined mouse movement paths after noise reduction

For a given segment with n points in moving set and a clicking set, extracted features can be described as following:

Time: time elapsed from the first point to the last point in moving set. Formally:

$$t = t_n - t_1 \tag{7.1}$$

Points: the number of total points of moving set. Formally:

$$p = n \tag{7.2}$$

Time (mean): mean time difference between each pair of consecutive points in moving set. Formally:

$$mean_t = \frac{1}{n-1}t \tag{7.3}$$

Gap: time difference between left button pressed and the point it followed. Formally:

$$t_g = t_d - t_n \tag{7.4}$$

Distance (straight): Euclidean distance between the first and last point in moving set. Formally:

$$d_s = \sqrt{(x_n - x_1)^2 + (y_n - y_1)^2} \tag{7.5}$$

Distance (curl): sum of the Euclidean distances between each pair of consecutive points in moving set. Formally:

$$d_c = \sum_{i=2}^{n} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \tag{7.6}$$

Direction: the angel from the positive y axis to the vector from the first point to the last point. The angel of segment j is described in a trigonometric way as $(\sin_j, \cos_j)$. Formally:

$$\sin = \frac{x_n - x_1}{\sqrt{(x_n - x_1)^2 + (y_n - y_1)^2}} \tag{7.7}$$

$$\cos = \frac{y_n - y_1}{\sqrt{(x_n - x_1)^2 + (y_n - y_1)^2}} \tag{7.8}$$

Speed (straight): moving speed for Euclidean distance between the first point to the last point in moving set. Formally:

$$v_s = \frac{d_s}{t} \tag{7.9}$$

Speed (each): moving speed at each point in moving set. Formally:

$$v_i = \frac{\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}}{t_i - t_{i-1}} \tag{7.10}$$

where $i \in [2, n]$. For the first point where $i = 1$, we give that $v_1 = 0$
Speed (curl): moving speed for sum of the distances between each pair of consecutive points in moving set. Formally:

$$v_c = \frac{d_c}{t} \tag{7.11}$$

Angel: for each point in moving set, angel feature refers to the angel between the vector from its previous point to itself and the vector from itself to its next point. Formally:

$$an_i = \arccos \frac{(x_i - x_{i-1})(x_{i+1} - x_i) + (y_i - y_{i-1})(y_{i+1} - y_i)}{\sqrt{[(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2][(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2]}} \tag{7.12}$$

where $i \in [2, n-1]$. Additionally, 4 features are extracted here — the maximum angel,the minimum angel, the mean angel and standard deviation of angel. Formally:

$$\max_{an} = \max\{an_i\}_{i=2}^{n-1} \tag{7.13}$$

$$\min_{an} = \min\{an_i\}_{i=2}^{n-1} \tag{7.14}$$

$$mean_{an} = \frac{1}{n-2} \sum_{i=2}^{n-1} an_i \tag{7.15}$$

$$s_{an} = \sqrt{\frac{1}{n-3} \sum_{i=2}^{n} (an_i - mean_{an})^2} \tag{7.16}$$

Width: the vertical distance from a point to the main vector (from the first point to the last point in moving set). Formally:

$$w_i = \frac{(x_i - x_1)(y_1 - y_n) - (x_1 - x_n)(y_i - y_1)}{\sqrt{(x_1 - x_n)^2 + (y_1 - y_n)^2}} \tag{7.17}$$

where $i \in [2, n-1]$. Additionally, 3 features are extracted here — difference between the maximum width and the minimum width, the mean width and standard deviation of width. Formally:

$$d_w = \max\{w_i\}_{i=2}^{n-1} - \min\{w_i\}_{i=2}^{n-1} \tag{7.18}$$

$$mean_w = \frac{1}{n-2} \sum_{i=2}^{n-1} w_i \tag{7.19}$$

$$s_w = \sqrt{\frac{1}{n-3} \sum_{i=2}^{n} (w_i - mean_w)^2} \tag{7.20}$$

Acceleration: the ratio of speed and time change from the previous point of one point to itself in moving set. Formally:

$$acc_i = \frac{v_i - v_{i-1}}{t_i - t_{i-1}} \tag{7.21}$$

where $i \in [2, n]$. Additionally, 4 features are extracted here — maximum, minimum, mean and standard deviation. Formally:

$$\max_{acc} = \max\{acc_i\}_{i=2}^{n} \tag{7.22}$$

$$\min_{acc} = \min\{acc_i\}_{i=2}^{n} \tag{7.23}$$

$$mean_{acc} = \frac{1}{n-1} \sum_{i=2}^{n} acc_i \tag{7.24}$$

$$s_{acc} = \sqrt{\frac{1}{n-2} \sum_{i=2}^{n} (acc_i - mean_{acc})^2} \tag{7.25}$$

Absolute speed change: the pure speed change from the previous point of one point to itself in moving set. Formally:

$$\Delta v_i = v_i - v_{i-1} \tag{7.26}$$

where $i \in [2, n]$. Additionally, 4 features are extracted here — maximum, minimum, mean and standard deviation. Formally:

$$\max_{\Delta v_i} = \max\{\Delta v_i\}_{i=2}^{n} \tag{7.27}$$

$$\min_{\Delta v_i} = \min\{\Delta v_i\}_{i=2}^{n} \tag{7.28}$$

$$mean_{\Delta v_i} = \frac{1}{n-1}\sum_{i=2}^{n}\Delta v_i \tag{7.29}$$

$$s_{\Delta v_i} = \sqrt{\frac{1}{n-2}\sum_{i=2}^{n}(\Delta v_i - mean_{\Delta v})^2} \tag{7.30}$$

Correlation with speed (4): correlation between speed change and angle change for each point. Formally:

$$c_{(v)i} = \begin{cases} (an_i + \frac{|an_i|}{an_i})\Delta v_i + (\Delta v_i + \frac{|\Delta v_i|}{\Delta v_i})an_i & \Delta v_i \neq 0, an_i \neq 0 \\ an_i & \Delta v_i = 0 \\ \Delta v_i & an_i = 0 \end{cases} \tag{7.31}$$

where $i \in [2, n]$. Additionally, 4 features are extracted here — maximum, minimum, mean and standard deviation. Formally:

$$\max_{c_{(v)i}} = \max\{c_{(v)i}\}_{i=2}^{n} \tag{7.32}$$

$$\min_{c_{(v)i}} = \min\{c_{(v)i}\}_{i=2}^{n} \tag{7.33}$$

$$mean_{c_{(v)i}} = \frac{1}{n-1}\sum_{i=2}^{n}c_{(v)i} \tag{7.34}$$

$$s_{\Delta v_i} = \sqrt{\frac{1}{n-2}\sum_{i=2}^{n}(c_{(v)i} - mean_{c_{(v)i}})^2} \tag{7.35}$$

Correlation with acceleration: correlation between acceleration and angel change for each point. Formally:

$$c_{(acc)i} = \begin{cases} (an_i + \frac{|an_i|}{an_i})acc_i + (acc_i + \frac{|acc_i|}{acc_i})an_i & acc_i \neq 0, an_i \neq 0 \\ an_i & acc_i = 0 \\ acc_i & an_i = 0 \end{cases} \tag{7.36}$$

where $i \in [2, n]$. Additionally, 4 features are extracted here — maximum, minimum, mean and standard deviation. Formally:

$$\max_{c_{(acc)i}} = \max\{c_{(acc)i}\}_{i=2}^{n} \tag{7.37}$$

$$\min_{c_{(acc)i}} = \min\{c_{(acc)i}\}_{i=2}^{n} \tag{7.38}$$

$$mean_{c_{(acc)i}} = \frac{1}{n-1}\sum_{i=2}^{n} c_{(acc)i} \tag{7.39}$$

$$s_{c_{(acc)i}} = \sqrt{\frac{1}{n-2}\sum_{i=2}^{n}(c_{(acc)i} - mean_{c_{(acc)i}})^2} \tag{7.40}$$

Backward relatively: this feature refers to the times that current moving direction is backward relatively. If at any point, the angel between its current moving vector and its previous moving vector is greater than 90 degree, then this point will be regarded as a relative backward point. The angel can be described formally:

$$an_{(r)i} = \arccos\frac{(x_i - x_{i-1})(x_{i+1} - x_i) + (y_i - y_{i-1})(y_{i+1} - y_i)}{\sqrt{[(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2][(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2]}} \tag{7.41}$$

Backward absolutely: this feature refers to the times that current moving direction is backward absolutely. If at any point, the direction of the vector projection of its current moving vector on its main vector is opposite to the direction of the vector projection of its previous vector, then this point will be regarded as a absolute backward point. The angel of current moving vector and main vector can be described formally:

$$an_{(a)i} = \arccos\frac{(x_n - x_1)(x_{i+1} - x_i) + (y_n - y_1)(y_{i+1} - y_i)}{\sqrt{[(x_n - x_1)^2 + (y_n - y_1)^2][(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2]}} \tag{7.42}$$

Backward both: this feature refers to the times that current moving direction is backward both relatively and absolutely. If any point is both relative backward point and absolute backward point, this point will be regarded as a both-backward point.

A total of 50 features were extracted and labeled as a single sample. Figure 7.5 illustrates the distribution of the number of samples extracted per participant. Overall, 42,970 samples were derived from 80,540 raw segments.

7.3.3 Evaluating metrics

F1-score, precision score, recall score and confusion matrix were used in our experiments for the multi-class identification tasks.

7.3.4 Classification methods

The SVM, Random Forest, and eXtreme Gradient Boosting (XGBoost) algorithms were evaluated. Three SVM configurations were tested: *LinearSVC* (One vs Rest), *SVC* (One vs One) with a linear kernel, and *SVC* (One vs One) with an RBF kernel. Default hyper-parameters were applied, and 10-fold cross-validation was utilised. As Table 7.3 demonstrates, the XGBoost classifier achieved the highest scores and the shortest running time. Although hyper-parameter tuning and feature engineering could further refine performance and

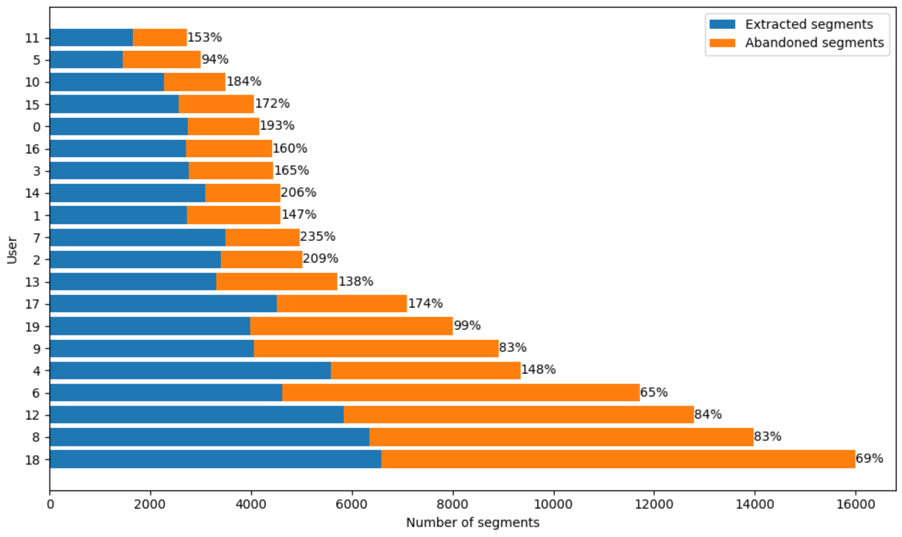


Figure 7.5: The proportion of segments per participant

Table 7.3: Comparative Performance Metrics of Different Classifiers

Classifier	F1 score	Precision	Recall	ACC	Time
XGBoost	**0.601**	**0.610**	**0.598**	**0.614**	**146s**
Random Forest	0.502	0.529	0.500	0.529	279s
LinearSVC	0.281	0.306	0.307	0.341	1668s
SVC(linear kernel)	0.398	0.412	0.403	0.425	1345s
SVC(rbf kernel)	0.403	0.422	0.405	0.436	1102s

affect rankings, the XGBoost classifier was selected for this research due to its significant lead.

7.3.5 Feature selection

Figure 7.6 presents the importance scores assigned to each feature. Despite their high performance, the features *timepp* and *scurlpp* were excluded because they are directly influenced by a user's hardware and software configuration, which may vary with a change in devices.

7.3.6 Model evaluation

It is recommended to search for the optimal hyper-parameters through cross-validation before training the classifier with data. Therefore, the *Grid Search* method was utilised to tune the hyper-parameters using *10-fold* cross-validation, as shown in Table 7.4. Additionally, *sample_weight* was employed to address sample imbalance issues. The classification results based on a single mouse point-and-click action are presented in Table 7.5, while Figure 7.7 depicts the

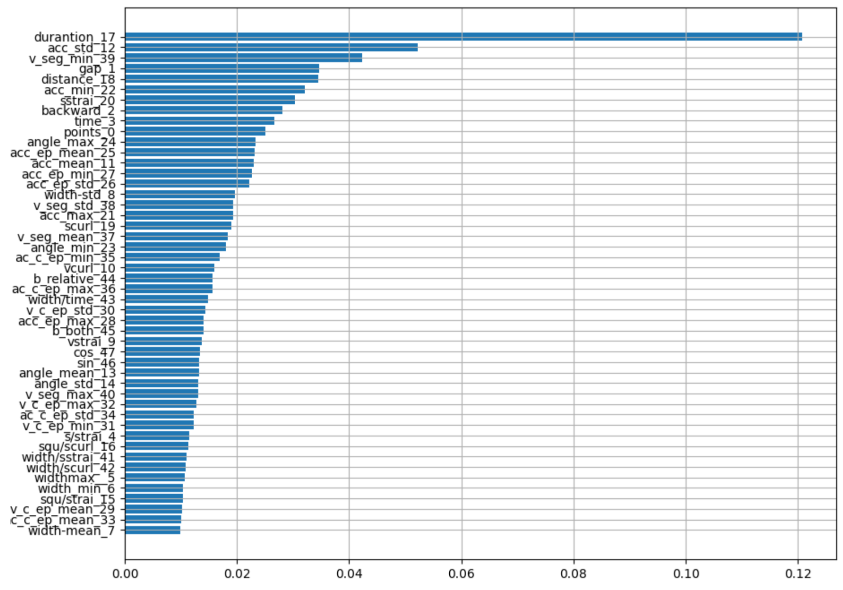


Figure 7.6: Importance scores for individual features

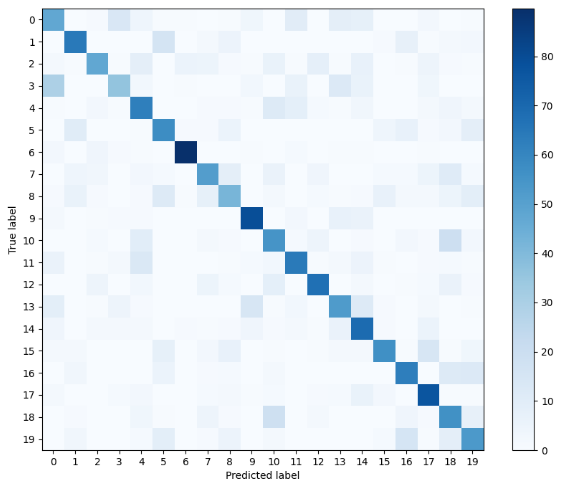


Figure 7.7: Confusion matrix for the classification results on the training set

Table 7.4: Tuned hyper-parameters of the XGBoost model

Hyper-parameter	**Value**
objective	*multi:softprob*
eval_metric	*mlogloss*
tree_method	*gpu_hist*
learning_rate	*0.3*
n_estimators	*600*
max_depth	*5*
min_child_weight	*3*

Table 7.5: Performance metrics of the tuned XGBoost classifier

Classifier	**F1 score**	**Precision**	**Recall**	**ACC**
XGBoost	0.601	0.610	0.598	0.614

confusion matrix for these results.

7.3.7 Accumulative methods

Table 7.5 presents the classification result from a single action, which indicates that the ability to identify a web user's identity based on a single mouse point-and-click action stands at 78.1%. As a user continues to engage with a device, an increased number of actions can be collected. For each action sample S, the predictive result can be calculated probabilistically using *predict_proba*, represented as $\{p_1, p_2, ..., p_n\}$ where p_i represents the probability that this action is associated with $user_i$. Consequently, the combined probability prediction for m-action samples $\{s_1, s_2, ..., s_m\}$ from n users can be described as:

$$P_{mn} = \begin{Bmatrix} p_{11} & \cdots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{m1} & \cdots & p_{mn} \end{Bmatrix} = \{\frac{1}{m}\sum_{j=1}^{m} p_{j1}, ..., \frac{1}{m}\sum_{j=1}^{m} p_{jn}\} \tag{7.43}$$

Therefore, the user number of the maximum probability value in P_{mn} would be predicting result. As is shown in Table 7.6, the classification performance is improved consistently from 1-action to 30-action samples.

7.4 Experimental results

7.4.1 Results on testing set

As previously mentioned, the testing set was partitioned from the raw data. Therefore, it can be considered a realistic representation of an identification scenario. The data processing procedure for the testing set is identical to that used for the training set. Table 7.7 displays the number of extracted samples for each user in the testing set.

Table 7.6: Classification Performance Metrics by Sample Size on the Training Set

Samples	F1 score	Precision	Recall	ACC
1	0.597	0.600	0.596	0.608
2	0.725	0.732	0.724	0.739
3	0.814	0.824	0.809	0.825
4	0.869	0.881	0.862	0.879
5	0.901	0.912	0.895	0.911
6	0.923	0.933	0.918	0.932
7	0.938	0.947	0.933	0.948
8	0.953	0.961	0.949	0.961
9	0.958	0.966	0.955	0.967
10	0.964	0.972	0.961	0.972
11	0.970	0.977	0.967	0.977
12	0.975	0.982	0.972	0.982
13	0.979	0.985	0.977	0.985
14	0.981	0.987	0.979	0.988
15	0.982	0.988	0.979	0.988
16	0.982	0.989	0.980	0.989
17	0.983	0.990	0.981	0.990
18	0.985	0.991	0.982	0.991
19	0.985	0.991	0.983	0.992
20	0.986	0.992	0.984	0.992
21	0.987	0.992	0.985	0.993
22	0.986	0.992	0.984	0.993
23	0.988	0.993	0.986	0.994
24	0.988	0.993	0.986	0.994
25	0.988	0.993	0.986	0.994
26	0.987	0.993	0.985	0.993
27	0.988	0.993	0.986	0.994
28	0.988	0.993	0.986	0.994
29	0.988	0.993	0.986	0.994
30	0.988	0.993	0.986	0.994

Figure 7.8 presents the confusion matrix for the classification results based on a single-action sample from the testing set. Table 7.8 details the classification outcomes for samples ranging from one-action to thirty-action increments.

7.4.2 Results on Bogazici dataset

The Bogazici dataset can be categorized into five distinct groups based on usage, as illustrated in Figure 7.9. For this research, only the *browsing* data were extracted to replicate a similar scenario. The Bogazici dataset encompasses 1,058,772 samples from 19 users, as detailed in Table 7.10. In this dataset, *user 18* contributed only 177 samples. Therefore, the data of user 18 was excluded due to significant imbalance. Classification outcomes ranging from a single-action sample to 30-action sample are presented in Table 7.9.

7.4.3 Comparisons

Although the experimental process is different from those employed in previous work, it is still meaningful to make a comparison, as in Table 7.11.

The prior studies discussed in this chapter were primarily focused on authentication tasks. Among the related literature, two studies are particularly relevant to the scope of this chapter. Mondal and Bours [59] utilised a public

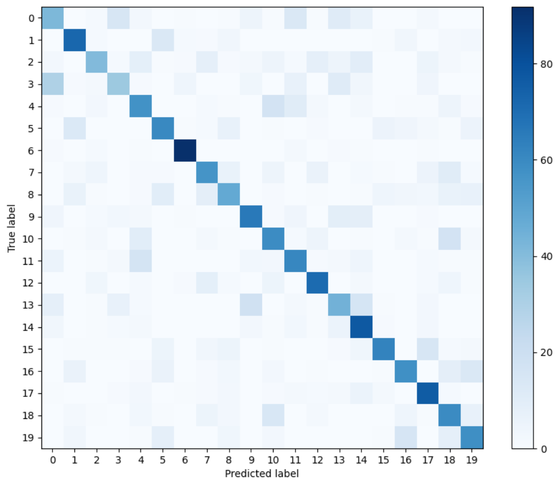


Figure 7.8: Confusion matrix of the classification performance on the testing set

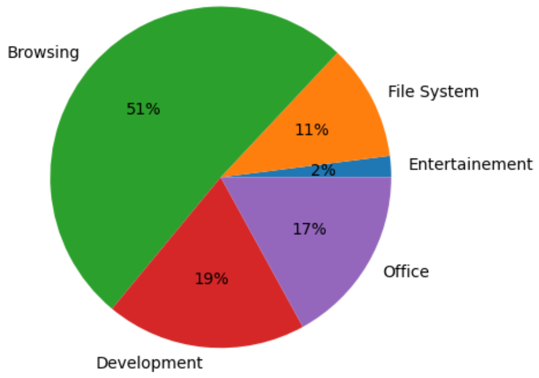


Figure 7.9: Visualization of the Bogazici dataset usage

Table 7.7: Comparison of the Number of Extracted Samples to the Raw Samples for Each User in the Testing Set

User	Extracted	Raw	User	Extracted	Raw
0	888	1427	**10**	569	898
1	477	779	**11**	410	678
2	811	1198	**12**	1773	3833
3	904	1712	**13**	537	865
4	1765	2921	**14**	1067	1508
5	367	770	**15**	657	958
6	1138	2729	**16**	634	1010
7	669	1004	**17**	691	1096
8	1193	2795	**18**	1816	4933
9	868	2208	**19**	1002	1944
			Total	18294	35266

dataset that included the behaviour of 48 users in an entirely uncontrolled setting. Nonetheless, it is important to note that this dataset also comprises activities not based on browser usage and it was not collected via any browser APIs. Antal et al. [11] developed a JavaScript-based web application and employed browser APIs similar to those discussed in chapter for data acquisition. And However, this is a highly controlled environment. To the best of our knowledge, the study presented in this chapter is the first to describe the identification of users through mouse dynamics data obtained via browser APIs in a completely uncontrolled browser-based environment.

Shen et al. [76] developed a specialized Windows-based application for desktop computers designed to capture user interaction through mouse-operated tasks. They engaged 37 participants who were instructed to complete identical mouse-operating tasks in two separate data collection sessions per day. The dataset comprised 5,550 samples across the 37 subjects. They tested with SVM, Neural Network and Nearest Neighbour and achieved the best performance with 8.74% FAR and 7.69% FRR for authentication tasks within 11.8 seconds.

Mondal and Bours [59] used a publicly available mouse dynamics dataset [63] in their studey. 48 users were asked to use their computer and mouse in a normal way, without any restrictions on the tasks. They used SVM and ANN as the machine learning methods and achieved 2% FNMR and 0.17% FMR for continuous authentication tasks.

Siddiqui et al. [77] recruited 10 users in their experiment. Participants were tasked with a 20-minute session of the *Minecraft* game, during which their mouse interactions were recorded via a Python script. Multiple binary RF classifiers were employed for their authentication process and achieved 92% of accuracy.

Rahman and Basak [70] applied the public dataset [33] with ten users data. Each user provided 5 to 7 sessions. Every session in this dataset is close to 120 minutes. They compared Random Forest and KNN and achieved better results on Random Forest with 1.74% FPR and 29.23% FNR for a 10-second authentication task. It should be noted that the FNR is much higher than the FPR. It practical applications, this comination of a low FPR and a high FNR will prevent unauthorised user access but also reject a genuine user or need longer input session before gaining access.

Table 7.8: Classification Performance Metrics by Sample Size on the Testing Set

Samples	F1 score	Precision	Recall	ACC
1	0.594	0.595	0.598	0.612
2	0.708	0.711	0.714	0.728
3	0.787	0.793	0.789	0.804
4	0.839	0.847	0.838	0.853
5	0.874	0.882	0.872	0.885
6	0.895	0.902	0.894	0.905
7	0.912	0.918	0.910	0.921
8	0.927	0.933	0.925	0.936
9	0.937	0.943	0.935	0.945
10	0.942	0.949	0.940	0.950
11	0.949	0.955	0.947	0.956
12	0.954	0.960	0.951	0.961
13	0.957	0.963	0.955	0.965
14	0.961	0.967	0.959	0.969
15	0.964	0.969	0.962	0.971
16	0.966	0.972	0.964	0.973
17	0.968	0.973	0.965	0.974
18	0.969	0.975	0.967	0.976
19	0.971	0.977	0.968	0.977
20	0.972	0.978	0.969	0.978
21	0.973	0.979	0.970	0.979
22	0.974	0.981	0.972	0.981
23	0.975	0.982	0.972	0.982
24	0.975	0.983	0.973	0.982
25	0.977	0.983	0.974	0.983
26	0.977	0.984	0.975	0.984
27	0.978	0.985	0.975	0.984
28	0.978	0.986	0.975	0.985
29	0.978	0.986	0.975	0.985
30	0.978	0.987	0.975	0.985

Antal et al. [11] invited 120 participants in their study. Each user was instructed to play the designed game on in a designed browser-based page. One-class SVM was selected as their machine learning algorithm. They achieved 0.94 AUC within 15-second authentication process.

7.5 Conclusions and possible future work

7.5.1 Conclusions

In this chapter, we conducted an experimental analysis on mouse dynamics data. We employed *XGBoost* classifier for classification tasks using the mouse dynamics data from the 20 users described in Chapter 6. The raw mouse data was segmented. Each segment represents a sequence of mouse movements and followed by a left clicking action (point-and-click). We extracted 48 features from each segment based on timing and mouse coordinate data. The *XGBoost* classifier was then utilised for classification. In Section 7.4, we report promising experimental outcomes. Using a single mouse action, we attained an f1-score of 59.4%. This increased to 94.2% with 10 consecutive mouse actions. Furthermore, using the public Bogazici dataset [52], we observed f1-scores of 44.7% for a single mouse action and 94.5% for 10 consecutive actions. The results suggest

Table 7.9: Classification Performance Metrics by Sample Size on the Bogazici Dataset

Samples	**F1 score**	**Precision**	**Recall**	**ACC**
1	0.445	0.434	0.516	0.507
2	0.585	0.566	0.662	0.664
3	0.689	0.667	0.755	0.763
4	0.769	0.748	0.819	0.832
5	0.829	0.811	0.863	0.877
6	0.870	0.856	0.894	0.907
7	0.900	0.890	0.917	0.928
8	0.922	0.915	0.933	0.944
9	0.937	0.932	0.946	0.955
10	0.950	0.946	0.956	0.964
11	0.959	0.956	0.963	0.970
12	0.966	0.964	0.969	0.975
13	0.972	0.970	0.974	0.979
14	0.975	0.974	0.977	0.982
15	0.979	0.978	0.981	0.985
16	0.982	0.981	0.984	0.987
17	0.984	0.983	0.986	0.988
18	0.986	0.986	0.988	0.990
19	0.988	0.987	0.989	0.991
20	0.989	0.989	0.991	0.992
21	0.991	0.990	0.991	0.993
22	0.992	0.991	0.992	0.994
23	0.992	0.992	0.993	0.994
24	0.993	0.993	0.994	0.995
25	0.994	0.993	0.994	0.995
26	0.995	0.994	0.995	0.996
27	0.995	0.995	0.996	0.996
28	0.996	0.995	0.996	0.997
29	0.996	0.996	0.996	0.997
30	0.996	0.996	0.997	0.997

that a website can identify the browser users via their mouse activities in a browser-based environment.

The prior art on mouse dynamics mainly focused on user authentication [10, 59, 76, 85] while the work in this chapter aims at user identification. Two different dataset were applied in this research: dataset collected from experimental extension (Extension dataset) and Baziciga dataset. Section 7.3 showed that 52% of the raw data could be processed and extracted as samples for further analysis. One extracted sample is based on a series of mouse moving actions and followed by a left clicking action. Each sample consists of 48 features and a label. XGBoost classifier model was built and applied as classification method. The experimental results showed that 10 valid mouse clicking actions may provide a promising identification result of 95% f1-score. This proves that a web user's identity can be identified simply via mouse using activities without any knowledge.

7.5.2 Possible future work

It would be valuable to collect user data across diverse known usage scenarios. Users may have different mouse devices, so it would be intriguing to standardize the experimental data collection by having all users employ the same device.

Table 7.10: Distribution of extracted samples per user in the Bogazici dataset

User	Extracted	User	Extracted	User	Extracted
User1	17709	**User7**	94404	**User13**	48186
User2	8961	**User8**	36285	**User14**	58560
User3	91992	**User9**	64029	**User15**	7422
User4	37365	**User10**	19104	**User16**	5802
User5	103674	**User11**	121731	**User17**	83673
User6	61446	**User12**	50388	**User19**	147864
				User18	177
Total	1058772				

Table 7.11: Comparative Overview of Mouse Dynamics Studies across Different Environments and Methods with Corresponding Performance Metrics

Author	Year	Users	Environment	Browser APIs	Method	Task	Performance
Feher et al. [31]	2012	25	controlled	No	SVM, ANN	auth	10% ERR
Shen et al. [76]	2013	37	controlled	No	SVM	auth	8.74% FAR, 7.69% FRR
Modal and Bours [59]	2015	49	uncontrolled	No	SVM, ANN	auth	2% FNMR, 0.17% FMR
Siddiqui et al. [77]	2021	10	controlled	No	RF	auth	92.73% accuracy
Rahman and Basak [70]	2021	10	controlled	No	RF, KNN	auth	1.74% FPR, 29.23% FNR
Antal et al. [11]	2021	120	controlled	Yes	SVM	auth	0.94 AUC
This study	2023	20	uncontrolled	Yes	XGBoost	iden	0.94 F1-score

RF: Random Forest, SVM: Support Vector Machine, GNB: Gaussian Naive Bayes, ANN: Artificial Neural Network, CPANN: Counter-Propagation Artificial Neural Network, DT: Decision Tree, MKL: Multiple Kernel Learning, auth: authentication, iden: identification

The performance of well-trained machine learning models on such standardized data could provide insightful results.

Chapter 8

Combining keystroke and mouse dynamics

8.1 Introduction

The previous two chapters have thoroughly investigated the dynamics of keystrokes and mouse movements, demonstrating promising performance for classification tasks when used independently. In real scenario, it is very common for browser users to interact with a browser using both the keyboard and mouse simultaneously over a specific duration. Combining keystroke and mouse dynamics could potentially enhance such identification performance, which motivates the method of this chapter.

This chapter takes a closer look at using both keyboard and mouse together, demonstrating a picture of how they work in real situations. The Python scripts used in this chapter are listed in Appendix D.3.3.

The structure of this chapter is organised as follows. Section 8.2 introduces the data gathering, data preparing and model building processes. Section 8.3 describes the experimental results on the collected dataset with machine learning techniques. Section 8.4 provides the discussions based on the experimental results and concludes this chapter.

8.2 Data processing

For our analyses, we utilised the comprehensive dataset previously mentioned in Chapters 6 and 7. This dataset was subjected to the same preprocessing and feature extraction techniques as detailed earlier.

8.2.1 Data processing

We deployed two separate *XGBoost* classifiers to assess the mouse and keystroke dynamics independently. The configuration of these classifiers remained aligned with the parameters and setups from Chapters 5 and 6 for the reasons outlined below:

- The origin of the data is identical.
- It has been noted that sometimes, we might encounter situations where only one type of data, either keystroke or mouse, is accessible. And such a condition may persist for an unpredictable period. In such context, a classification method relying solely on a single data source becomes applicable, leading to the same circumstances as described in Chapters 6 or 7.
- Searching for hyper-parameters and potential alternative classifiers with two source of data together would require substantial time investment.

Given the considerations mentioned above, the data processing procedure remain the same and we choose to approach our data analysis from the perspective of successful interactions, such as key presses and mouse activities, rather than focusing on timing intervals.

8.2.2 Classifier configurations

We choose to use the same configurations as described in Chapters 5 and 6, and as shown in Table 8.1

Table 8.1: Tuned hyper-parameters of each classifier

Hyper-parameter	**Keystroke**	**Mouse**
objective	*multi:softprob*	*multi:softprob*
eval_metric	*mlogloss*	*mlogloss*
tree_method	*gpu_hist*	*gpu_hist*
learning_rate	*0.3*	*0.3*
n_estimators	*600*	*600*
max_depth	*5*	*5*
min_child_weight	*5*	*3*

8.2.3 Accumulative methods

In Chapters 6 and 7, we introduced an accumulative approach to handling action samples. For a give action sample S, we can calculate the prediction probabilities as $\{p_1, p_2, ..., p_n\}$, using the *predict_proba* function, where p_i indicates the probability that the action sample belongs to $user_i$. As users interact with their devices over time, we accumulate a significant volume of data from both keyboard and mouse inputs. For each keystroke sample K, the predicted probabilities are computed using *predict_proba* and represented as $\{k_1, k_2, ..., k_n\}$, where k_i denotes the probability that this keystroke sample belongs to $user_i$. Similarly, for each mouse sample M, the predicted probabilities are computed and denoted as $\{m_1, m_2, ..., m_n\}$, with m_i indicating the probability for $user_i$.

Therefore, for a series of actions consisting of k-keystroke samples and m-mouse samples for n users, the combined probability prediction can be expressed as:

$$P_{kmn} = \{\frac{1}{k}\sum_{j=1}^{k} k_{j1} + \frac{1}{m}\sum_{j=1}^{m} m_{j1}, ..., \frac{1}{k}\sum_{j=1}^{k} k_{jn} + \frac{1}{m}\sum_{j=1}^{m} p_{jn}\} \tag{8.1}$$

Here, k and m represent the number of keystroke and mouse samples, respectively, and n is the number of users, which is at most 20, as that is the total number of participants in this experiment.

8.3 Experimental results

This section demonstrates the experimental results from the combined use of *XGBoost* classifiers for keystroke and mouse dynamics.

8.3.1 Results on training set

Figure 8.1 displays the confusion matrix for training set, which includes data from one keystroke and one mouse dynamics sample. Figure 8.3 illustrates the F1-scores obtained for various combinations of keystroke and mouse dynamics samples within the training set. An F1-score of 0.950 was achieved with a combination of five valid mouse actions and one 4-keystroke sample.

8.3.2 Results on testing set

Figure 8.2 shows the confusion matrix for the testing set, which includes data from one keystroke and one mouse dynamics sample. Figure 8.4 illustrates the F1-scores for various keystroke and mouse dynamics samples combinations in testing set. An F1-score of 0.914 was achieved with the combination of five valid mouse actions and a single 4-keystroke sample.

8.3.3 Comparisons

Although our experimental methods differ from those employed in previous studies, it is still meaningful to make comparisons to the most recent work, as in Table 8.2.

The prior studies discussed in this chapter were primarily focused on authentication tasks, with the exception of the research by Wang et al. [91], which addressed both authentication and identification tasks. Among them, three

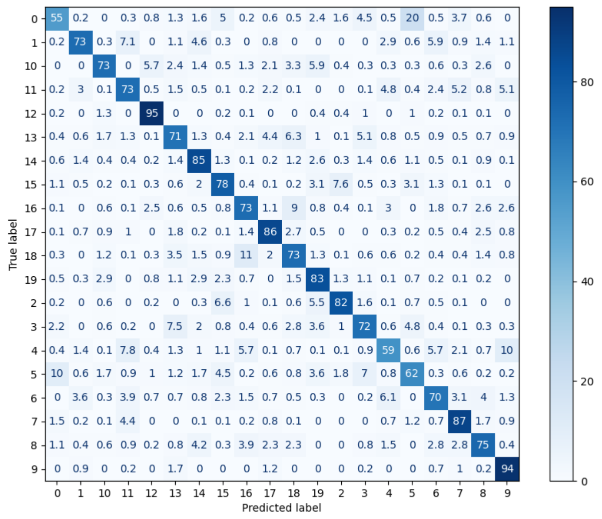


Figure 8.1: Confusion Matrix for the Training Set Using Combined Keystroke and Mouse Dynamics

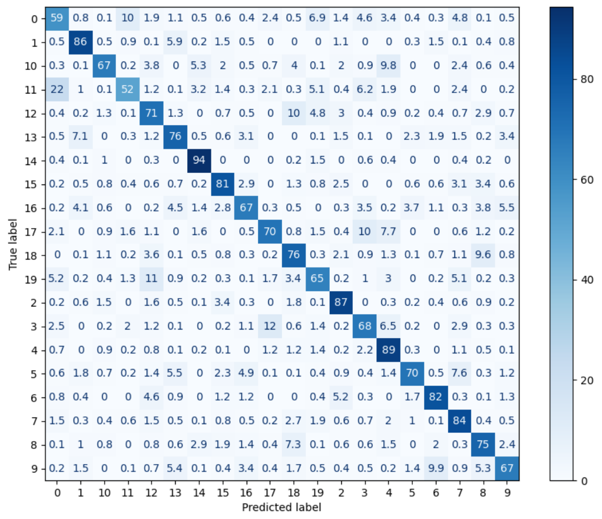


Figure 8.2: Confusion Matrix for the Testing Set Using Combined Keystroke and Mouse Dynamics

previous studies are more close related to the scope of this chapter than other studies. Although Wang et al. [91] instructed the participants in their experiment to accomplish several designed tasks in a browser-based environment, they did not utilise any browser APIs to collect the experimental data. In Modal and Bour's experiments [60], participants were free to use their own computers, however, data was gathered through specially designed software. Earl et al. [27] employed the similar browser APIs as described in this study in their experiments to collect experimental data, however, their experimental environment was fully controlled — users were asked to perform several highly designed tasks to generate experimental data. To the best of our knowledge, the study described in this chapter is the first to present the identification of users by combining user keystroke and mouse dynamics data obtained via Browser APIs in a completely uncontrolled browser-based environment. A summary of prior work is as follows.

Bailey et al. [14] invited 31 participants in their study. Each participant was asked to answer three inquiries, composing a report of 400 to 500 words for each on the designed Window software. The gathered data was segmented into samples, each covering a 10-minute interval. For keystroke dynamics, they computed the average Up-Up (UU) and Down-Up (DU) times as features. In terms of mouse dynamics, they calculated features such as speed, direction, travel distance, and click timing information derived from mouse movements and clicks. The keystroke and mouse data were then combined, and two classifiers, BayesNet and LibSVM, were employed for the analysis. In the authentication tasks, they achieved a FAR of 3.76% and a FRR of 2.51% when using the BayesNet classifier. The LibSVM classifier resulted in a FAR of 11.67% and a FRR of 18.80%.

Mondal and Bours [60] invited 53 individuals for their study. These participants installed a custom software on their computers and used them freely for 5-7 days, generating an average of 700,000 events per user. Keystroke and mouse interactions accounted for 12.4% and 83.3% of events, respectively. Digraphs was used in keystroke while mouse movements and Drag-Drop was applied for mouse dynamic. Parameters such as angles, speed, acceleration, and distance were extracted for the analysis. The study used binary classifiers to distinguish between legitimate users and imposters, allocating 35% of data for training, 10% for tuning, and the remaining 55% for testing. ANN, CPANN, and SVM were employed for classification. The study also proposed the metrics ANIA and ANGA to measure the effectiveness of the authentication system. In terms of authentication performance, imposters were detected after an average of 252 actions.

Earl et al.[27] recruited 240 participants in their study, with 15 left handed and 225 right handed users. The distribution of age, and gender between female and male participants was approximately even. Participants engaged in a series of designated clicking tasks as per on-screen instructions. Additionally, they were required to input a fixed text passage. Instead of focusing on authentication or identification tasks for users' identities, they focused on identifying users' handedness, ages and gender. Various classifiers was employed, including

Random Forest, SVM, Decision Tree, Gaussian Naive Bayes, and KNN. Among them, Random Forest achieved the best performance with accuracy of 68.3% in identifying gender, 64% of the handedness, and 26.9% of the age. While the classification results might not seem highly accurate, they present valuable insights in this field.

Wang et al. [91] invited 41 participants in data collection. The participants were instructed to perform four tasks on the same laptop: typing, browsing Taobao website, browsing Weibo website, and gaming. Each task was assigned for one hour. The gathered data was divided into segments, varying from 10s to 480s. They employed Multiple Kernel Learning method to improve the performance of the target kernel. SVM was selected as the classification method. They achieved best classification performance with a 300s time window sample with a FAR of 16.9%, a FRR of 15% and F1-score of 83.9%.

Most of the prior work focused on segmenting the data based on a timing interval, such as 10 minutes in [14] and 300 seconds in [91]. Wang et al. [91] provided a variety of data across different contexts which make the data closer to real life scenarios. However their participants actually took part in four intense tasks and they used 300s as the time window for sample segmentation. Given the intensity of such tasks, users could generate plenty of data within the 300 seconds. In our study, only 5 valid mouse clicks and 4 consecutive keystrokes gave promising classification results.

Table 8.2: Summary of Combined use of Keystroke and Mouse Dynamics Studies across Different Environments and Methods with Corresponding Performance Metrics

Author	**Year**	**Users**	**Environment**	**Browser APIs**	**Method**	**Task**	**Performance**
Bailey et al. [14]	2014	31	controlled	No	BayesNet, SVM	auth	3.76% FAR and 2.51% FRR, 11.67% FAR and 18.80% FRR
Mondal and Bours [59]	2015	53	uncontrolled	No	ANN, CPANN, SVM	auth	imposters detected after 252 actions
Earl et al. [27]	2021	240	controlled	Yes	RF, SVM, DT, GNB, KNN	auth	68.3% accuracy of gender, 64% of handedness, 26.9% of age
Wang et al. [91]	2022	41	controlled	No	MKL, SVM	auth, iden	16.9% FAR and 15% FRR, 83.9% F1-score
This study	2023	20	uncontrolled	Yes	XGBoost	iden	0.95 F1-score

RF: Random Forest, SVM: Support Vector Machine, GNB: Gaussian Naive Bayes, ANN: Artificial Neural Network, CPANN: Counter-Propagation Artificial Neural Network, DT: Decision Tree, MKL: Multiple Kernel Learning, auth: authentication, iden: identification

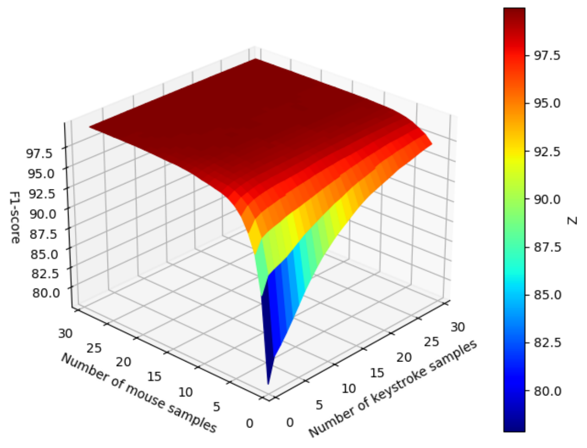


Figure 8.3: F1-Scores across Various Combinations of Keystroke and Mouse Dynamics Samples in the Training Set

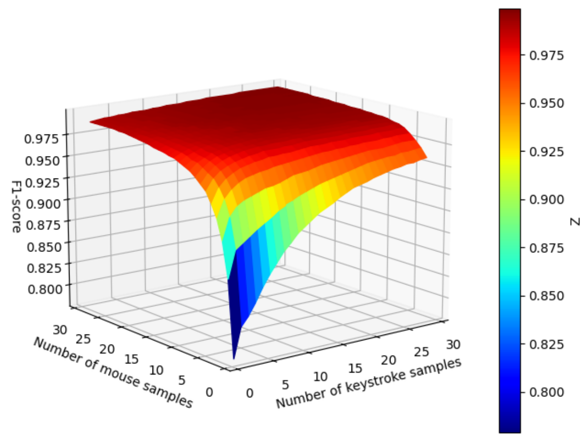


Figure 8.4: F1-Scores across Various Combinations of Keystroke and Mouse Dynamics Samples in the Testing Set

8.4 Conclusions and possible future work

8.4.1 Conclusions

In this chapter, we performed experimental analysis that combined keystroke and mouse dynamics data. Then we applied the *XGBoost* classifier for multi-class classification, covering both keystroke dynamics (outlined in Chapter 6) and mouse dynamics (detailed in Chapter 7). By fusing these two classifiers, we obtained promising results with a 73% F1-score for individual mouse clicks and keystrokes. The F1-score increased to 95% as mouse clicks raised to 5 and consecutive keystrokes is 5. The results indicate that the browser users could be identified through their mouse and keystroke dynamics data in a browser based environment. Moreover, integrating keystroke and mouse dynamics can improve classification performance.

8.4.2 Possible future work

It would be very valuable to collect user data from different using scenarios as user behaviour can vary across different contexts. With appropriate post-processing of user data, a new data gathering experiment with recording the browser's web url would provide a valuable extension to this work.

Chapter 9

Study on mobile touch behaviour

9.1 Introduction

We live in a digital world where mobile devices have become increasingly popular. Xiaomei et al. [93] have indicated that daily activity encompasses 400,000 Apple and 1.3 million Android devices. Various mobile sensors (e.g., GPS, gyroscope, compass and accelerometer) can be used by mobile applications to gain access to user data [58]. Meanwhile, Browser APIs can obtain user data during interactions with a webpage. It has been pointed out that mobile APIs could potentially pose user security and privacy issues. Bojinov et al. [16] suggested that mobile sensors could be used to de-anonymize mobile device. Mehrnezhad et al. [58] demonstrated an attack on PIN input using *Orientation* and *Motion* sensors. Zhang et al. [93] suggested interactions with mobile devices produce collectable data that may serve as behavioural biometric features.

While browsing a webpage on a mobile device, the hosting website may employ browser APIs to collect user data. The primary focus of this chapter is to explore the extent to which such collected data can contribute to user identification. Python scripts were used for data analysis in this chapter, and are listed in Appendix D.3.4.

The structure of this chapter is organised as follows. Section 9.2 introduces data analysis with Kim and Kang's dataset [51] dataset. Section 9.3 introduces the implementation of the web page for data collection. Section 9.4 concludes the work in this chapter and highlights potential possible future work.

9.2 Experiments on Kim and Kang dateset

In this section, we examined the classifications result using the *XGBoost* classifier on the dataset (**K** dataset) provided by Kim and Kang [51]. The purpose of this experiment is to determine whether mobile users can be identified through mobile sensor data. The reasons for selecting the **K** dataset are as follows.

- The **K** dataset is of a sufficient size to apply machine learning techniques effectively.
- The dataset includes timing information, touchpoint coordinates, and data from the *Motion* API.
- The data was collected using 20 different pre-defined text scripts. Compared to the use of PINs with a fixed length, the **K** dataset is the closest to an uncontrolled free text environment.
- It was the only publicly available dataset that provides keystroke dynamics data for mobile devices in the context of free and lengthy text entry.

9.2.1 Feature engineering

Each sample in the **K** dataset was derived from digraph data. For each sample, 32 features were extracted into four categories: timing-based features (6 in total), key values (2 in total), coordinate-based features (12 in total), and motion-sensor-based features (12 in total). 75% of the data were selected as the training set and the remaining 25% was used as the testing set.

9.2.1.1 Timing-based data

As previously introduced in Chapter 6, timing-based features can calculated from the *keydown* (D) and *keyup* (U) timestamps. Within a digraph sample (D_1, U_1, D_2, U_2), six combinations can be extracted: D_1U_1, D_1U_2, D_2U_1, D_2U_2, D_1D_2 and U_1U_2.

9.2.1.2 Key values

This feature refers to the key value of input, such as key ‘A’ and key ‘B’.

9.2.1.3 Coordinate-based data

This feature refers to the coordinates of a touch point. When a user taps on the screen, there are a number of points (X_i, Y_i) that could be touched. The minimum (X_{min}, Y_{min}), average (X_{mean}, Y_{mean}) and maximum (X_{max}, Y_{max}) of each sample were used.

9.2.1.4 Motion-sensor-based features

This feature refers to the *acceleration* values (x,y,z) when a key is pressed down and released respectively.

9.2.2 Model tuning

XGBoost was selected as the classifier for this multiclass classification task. For the best performance of the classifier, the *gridsearch* method was used as the hyper-parameter tuning methods with *5-fold* cross-validation. The hyper-parameters that yielded the best performance are shown in Table 9.1. The classification results with *5-fold* cross-validation are shown in Table 9.2.

Table 9.1: Optimized Hyper-parameters for the XGBoost Classifier Applied to the **K** Dataset

Hyper-parameter	**Value**
objective	*multi:softprob*
eval_metric	*mlogloss*
tree_method	*gpu_hist*
learning_rate	*0.2*
n_estimators	*600*
max_depth	*6*
min_child_weight	*3*

Table 9.2: XGBoost Classification Performance on the **K** dataset Using 5-fold Cross-validation

Classifier	**F1 score**	**Precision**	**Recall**	**ACC**
XGBoost	0.5652	0.5646	0.5696	0.5724

9.2.3 Experimental result

Table 9.3 presents the classification results of the *XGBoost* classifier, utilizing the hyper-parameters specified in Table 9.1 on the test set. An f1-score of 96.1% was attained using 10 samples.

9.2.4 Comparisons

Despite the differences in experimental methodologies between our study and prior research, a comparison remains valuable, as described in Table 9.4. It is important to note that the approach employed in this chapter is for identification tasks, in contrast to previous studies that concentrated on the authentication process.

Gascon et al. [34] invited more than 300 participants in their study. 303 users were asked to enter a fixed text (160 characters) on an Android smartphone only once, acting as imposters, while the rest 12 participants entered the same text for 10 times which was considered as authorized users. They extracted features from *gyroscope*, *orientation*, *accelerometer* and timing based keystroke information. SVM was selected as the classification method. They

Table 9.3: Classification Results on the **K** Test Dataset for Samples Ranging from 1 to 30

Samples	**F1 score**	**Precision**	**Recall**	**ACC**
1	0.598	0.598	0.600	0.603
2	0.695	0.697	0.698	0.701
3	0.771	0.774	0.773	0.776
4	0.828	0.831	0.829	0.832
5	0.869	0.872	0.870	0.872
6	0.899	0.902	0.900	0.901
7	0.922	0.924	0.922	0.923
8	0.939	0.941	0.939	0.940
9	0.952	0.953	0.951	0.952
10	0.961	0.962	0.961	0.962
11	0.968	0.969	0.968	0.969
12	0.974	0.975	0.974	0.974
13	0.979	0.979	0.979	0.979
14	0.982	0.983	0.982	0.982
15	0.985	0.986	0.985	0.985
16	0.988	0.988	0.988	0.988
17	0.989	0.990	0.989	0.989
18	0.991	0.991	0.991	0.991
19	0.992	0.993	0.992	0.992
20	0.993	0.993	0.993	0.993
21	0.994	0.994	0.994	0.994
22	0.995	0.995	0.995	0.995
23	0.996	0.996	0.996	0.996
24	0.996	0.996	0.996	0.996
25	0.997	0.997	0.997	0.997
26	0.997	0.997	0.997	0.997
27	0.997	0.997	0.997	0.997
28	0.998	0.998	0.998	0.998
29	0.998	0.998	0.998	0.998
30	0.998	0.998	0.998	0.998

have heterogeneous classification performance, for each user, the AUC ranges from 0.5 to 0.9, and a high FPR at 35%.

Antal et al. [12] invited 42 users in their experiment. The users were asked to type in the same password for 30 times during 2 sessions. They extracted coordinates, pressure and timing based keystroke information from the samples. Random Forest, Bayesian nets and SVM were tested as the classifiers. They achieved an accuracy of 93.04% with Random Forest, 91.94% with Bayesian nets, and 88.33% with SVM.

Lee et al. [53] invited 22 participants in the study. An Android application was developed for the data collection. Each user was asked to input 6-digit PIN '766420' for 100 times on the same mobile device. They extracted features from *gyroscope* data, the area of touching points, *accelerometer* data and timing based keystroke information. They used distance-based classificaiton algorithms and achieved 7.89% EER for the authentication tasks.

Kim and Kang [51] invited 50 participants in their study. An Android application was developed for data collection. Each participant was asked to type 20 predefined text using both hands in in a sitting position with the same device. Each text consists of approximately 200 keystrokes. 32 features based on *accelerometer*, touch coordinate and timing based keystroke information were extracted. R measure was selected as the classification methods. They achieved an EER below 0.05% with one sample of 200 keystrokes for both Korean and English input.

Acien et al. [1] developed a novel classification framework named TypeNet, which they applied to both authentication and identification tasks. In the authentication task, they achieved an EER of 9.2%. They achieved an accuracy of 94.2% in the identification task, with a test set comprising 1,000 users for each evaluation.

Stragapede et al. [80] used the same mobile keystroke dataset described in [1]. They extracted features from digraphs within samples containing between 50 to 70 keystrokes. They employed *Transformer* architecture as the machine learning approach and achieved an EER of 3.15% for the authentication tasks with 10 enrolment sessions of 50 samples each.

Table 9.4: Comparative Overview of Mobile Touch Dynamics Studies across Different Environments and Methods with Corresponding Performance Metrics

Author	**Year**	**Subjects**	**Environment**	**Method**	**task**	**Performance**
Antal et al. [12]	2015	42	controlled	RF, Bayesian, SVM	auth	93.04%, 91.94%,88.33% of accuracy
Gascon et al. [34]	2014	300	controlled	SVM	auth	35% FPR
Lee et al. [53]	2018	22	controlled	Distance-based	auth	7.89% EER
Kim and Kang [51]	2020	41	controlled	R measure	auth	0.05% EER
Acien et al. [1]	2021	60,000	controlled	TypeNet	auth, iden	0.05% EER,94.2% accuracy
Stragapede et al. [80]	2023	60,000	controlled	Transformer	auth	3.15% EER
This study	2023	41	uncontrolled	XGBoost	iden	0.95 of F1-score

RF: Random Forest, SVM: Support Vector Machine, GNB: Gaussian Naive Bayes, auth: authentication, iden: identification

9.3 Data collection via mobile webpage

In Section 9.2, we demonstrated promising classification results using data from the Android generic sensor APIs. Similarly, browser APIs on mobile devices can provide access to such data. In this section, we introduce a webpage that collects data via sensor APIs while a user interacts with the browser on a mobile device. The source code for the webpage is provided in Appendix D.2.

9.3.1 Development environment

All software development was carried out on a Windows 10 system. The development tools and programming languages used were as follows.

- **Programming tool**: Sublime Text (Build 4126).
- **Programming languages:** HTML and JavaScript (client-side).

9.3.2 Web page development

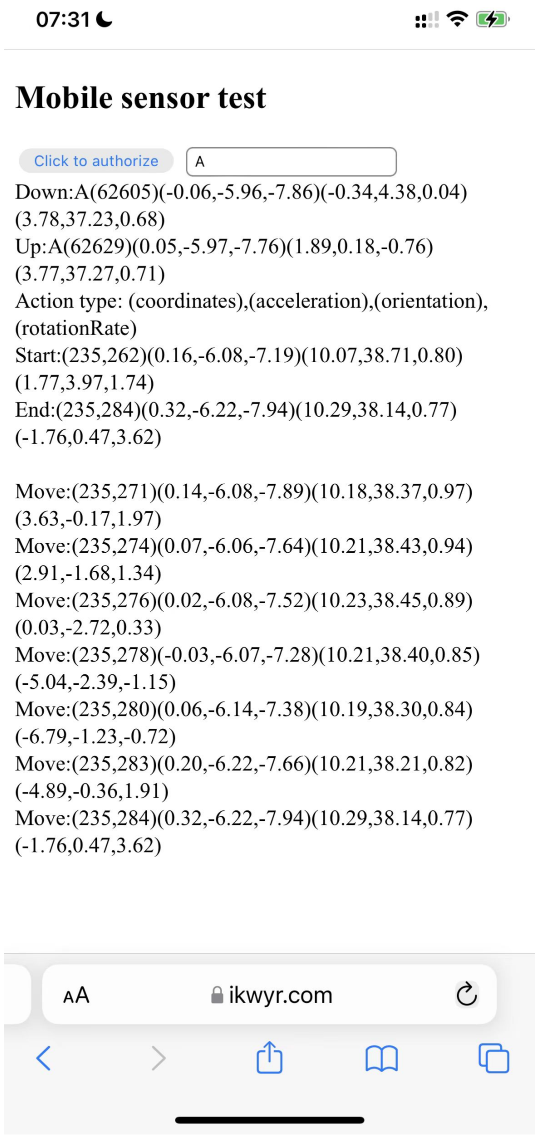
The web page was developed to gather user data via mobile browser APIs. *DeviceOrientationEvent* and *DeviceMotionEvent* were used to access sensor data as follows.

```
if (window.DeviceOrientationEvent){
   window.addEventListener('deviceorientation',
        deviceOrientationHandler, false);
   function deviceOrientationHandler(e){
      var gamma = e.gamma;
      var beta = e.beta;
      var alpha = e.alpha;
      }
   }
}
if (window.DeviceMotionEvent){
   window.addEventListener('devicemotion',function(event){
      var gacc = event.accelerationIncludingGravity;
      var gx = gacc.x;
      var gy = gacc.y;
      var gz = gacc.z;
      var rotation = event.rotationRate;
      if (rotation){
         var rx = rotation.alpha;
         var ry = rotation.beta;
         var rz = rotation.gamma;
      }
   }, false)
}
```

9.3.3 Data collection

The web page was tested in Safari browser on a iPhone 12. Figure 9.1 displays the data collection interface. The first four lines show the data collected when the key *A* is pressed and released. For each keystroke, four types of data are collected: timing information, and data from the *Accelerometer*, *Orientation* and *rotationRate* APIs.

Furthermore, when a user interacts with the mobile browser, such as scrolling through the webpage, additional data are also captured. This includes the coordinates of the touch point and data from the *Accelerometer* API, *Orientation* and *rotationRate* APIs.



Mobile sensor test

```
Down:A(62605)(-0.06,-5.96,-7.86)(-0.34,4.38,0.04)
(3.78,37.23,0.68)
Up:A(62629)(0.05,-5.97,-7.76)(1.89,0.18,-0.76)
(3.77,37.27,0.71)
Action type: (coordinates),(acceleration),(orientation),
(rotationRate)
Start:(235,262)(0.16,-6.08,-7.19)(10.07,38.71,0.80)
(1.77,3.97,1.74)
End:(235,284)(0.32,-6.22,-7.94)(10.29,38.14,0.77)
(-1.76,0.47,3.62)

Move:(235,271)(0.14,-6.08,-7.89)(10.18,38.37,0.97)
(3.63,-0.17,1.97)
Move:(235,274)(0.07,-6.06,-7.64)(10.21,38.43,0.94)
(2.91,-1.68,1.34)
Move:(235,276)(0.02,-6.08,-7.52)(10.23,38.45,0.89)
(0.03,-2.72,0.33)
Move:(235,278)(-0.03,-6.07,-7.28)(10.21,38.40,0.85)
(-5.04,-2.39,-1.15)
Move:(235,280)(0.06,-6.14,-7.38)(10.19,38.30,0.84)
(-6.79,-1.23,-0.72)
Move:(235,283)(0.20,-6.22,-7.66)(10.21,38.21,0.82)
(-4.89,-0.36,1.91)
Move:(235,284)(0.32,-6.22,-7.94)(10.29,38.14,0.77)
(-1.76,0.47,3.62)
```

Figure 9.1: Screenshot of the web page for data collection

9.4 Conclusions and possible future work

9.4.1 Conclusions

In this chapter, we explored user identification on mobile devices through mobile sensor APIs. In Section 9.2, we used the public mobile keystroke dataset [51] and employed *XGBoost* algorithm for classification. The original raw data was gathered using various mobile APIs through an experimental application on an Android device and was processed into digraph samples. 30 features were extracted based on timing, motion and orientation information. We achieved F1-scores of 59.8% with a single sample and 96.1% with ten samples, respectively. The experimental results indicate that the mobile user can be identified with such data. In Section 9.3, we then developed a web page capable of collecting user data in a browser-based environment using mobile browser APIs, e.g. the *Orientation* and *Motion* APIs. The dataset collected by our custom-designed webpage shares similarities with the data from the study by Kim and Kang [51]. ,indicating that machine learning techniques could potentially unmask the identity of a mobile browser user through these browser APIs.

9.4.2 Possible future work

We aim to create a mobile webpage enriched with features that can capture meaningful user interactions effectively. Additionally, we plan to conduct an experiment to gather data on browser user interactions using such a webpage on mobile devices.

Chapter 10

Security and privacy recommendations

10.1 Introduction

This chapter discusses the security and privacy issues arising from the findings of Chapters 6, 7, 8 and 9.

The structure of this chapter is as follows. Section 10.2 provides context for the remainder of the chapter by examining the availability of the API functionality employed in the work described in Chapters 5, 6 and 7 for a range of browsers and host platforms. Section 10.3 discusses possible attack scenarios. Section 10.4 gives recommendations for possible mitigations to the security and privacy issues described in Chapters 6, 7, 8 and 9. Of course, enabling user identification in a continuous fashion without user involvement may also offer benefits, and Section 10.5 considers possible applications.

10.2 Browser API functionality

Before looking in detail at the findings of Chapters 6, 7, 8 and 9, we consider the degree to which the choice of browser and platform influences the effectiveness of user identification. In particular, we examined the availability of the browser API information that was used in the reported experiments on a range of common user browsers and computing platforms. Clearly, if a browser does not provide the information that was used, then the identification process cannot operate, and user privacy is protected.

When examining the various browsers, we examined both ‘normal mode’ and ‘private mode’, where ‘private mode’ is an option provided by all the browsers we examined which is designed to prevent users being tracked. We examined browsers on both PC platforms (Windows and MacOS), and mobile platforms (Android and IOS).

Table 10.1: Browser version, OS versions and device information on PC platform

Browser	Device	OS
PC		
Chrome 97.0.4692.99 (64-bit)	Desktop PC	Windows 10 Home 19044.1466
Firefox 96.0.3 (64-bit)	Desktop PC	Windows 10 Home 19044.1466
Microsoft Edge 97.0.1072.76 (64-bit)	Desktop PC	Windows 10 Home 19044.1466
Chrome 97.0.4692.71 (x86_64)	MacBook Pro	macOS 11.6.2 (20G314)
Firefox 96.0.3 (64-bit)	MacBook Pro	macOS 11.6.2 (20G314)
Safari 15.2 (16612.3.3.1.8, 16612)	MacBook Pro	macOS 11.6.2 (20G314)

Table 10.2: PC platform test environments

Browser	Device	OS
Mobile		
Chrome 97.0.4692.84	iPhone 12 Pro	IOS 15.0.0 (19A404)
Firefox 60.0 (7403)	iPhone 12 Pro	IOS 15.0.0 (19A404)
Safari 15	iPhone 12 Pro	IOS 15.0.0 (19A404)
Chrome 97.0.4692.98	Galaxy A02s	Android 11
Firefox 96.2.0	Galaxy A02s	Android 11
Samsung Internet 16.0.6.23	Galaxy A02s	Android 11

10.2.1 PC platforms

We examined the browsers on **Windows** and **MacOS**. Table 10.1 summarises the testing environment that was employed. For Windows, we examined the Firefox, Chrome and Microsoft Edge browsers, in both normal and private modes. For MacOS, we examined the Firefox, Chrome and Safari browsers, again in both normal and private modes.

The results were consistent across all browsers and platforms; that is, the *KeyboardEvent* and *MouseEvent* APIs were accessible in all browsers, on all platforms, and at all times, i.e. regardless of whether the browser was operating in normal or private mode.

10.2.2 Mobile platforms

We also examined key browsers on mobile platforms, i.e. on the **Android** and **IOS**. Table 10.2 shows the testing environments for the mobile platforms. For IOS, we examined the Firefox, Chrome and Safari browsers; for MacOS, we examined the Firefox, Chrome and Samsung Internet browsers. Table 10.3 summarises the results. The results indicate that all the relevant APIs can be accessed, although in some cases explicit user consent is required. The *KeyboardEvent* and *TouchEvent* APIs can be accessed and without user consent on all the browsers we examined. On the IOS platform, all browsers require user consent to access the *Motion* and *Orientation* sensors, whereas on the Android

Table 10.3: Mobile platform test environments

	Android			**IOS**		
	Firefox	Chrome	Samsung	Firefox	Chrome	Safari
KeyboardEvent	y/n	y/n	y/n	y/n	y/n	y/n
TouchEvent	y/n	y/n	y/n	y/n	y/n	y/n
Orientation	y/n	y/n	y/n	y/y	y/y	y/y
Acceleration	y/n	y/n	y/n	y/y	y/y	y/y
RotationRate	y/n	y/n	y/n	y/y	y/y	y/y

y/n: accessible/no permission required, y/y: accessible/permission required

platform, no permissions were required by any of the browsers.

10.3 Possible attack scenarios

Keyboard and mouse are probably the two most widely used input devices for PC users. It is almost impossible to interact with a browser without using either keyboard or mouse simultaneously. These interactions can generate data, potentially revealing a user's true identity. Chapters 6, 7, 8 and 9 illustrate such methods, demonstrating promising experimental results. This section will explore potential attack scenarios derived from such findings.

10.3.1 Scenario One

User A frequently visits website B, providing B with an extensive dataset from A's mouse movements and keystrokes over time. By employing appropriate techniques which are introduced in Chapters 6, 7, 8 and 9, B has successfully transformed this data into a distinctive user template, effectively capturing User A's unique behaviour signature.

In an effort to protect privacy, User A opts to visit B in a private browsing mode and without logging in to the site, which seems to be an anonymous way. However, due to the unique user template that B has generated, B still can find out this 'anonymous' user's identity, thereby compromising the user's privacy.

10.3.2 Scenario Two

In this scenario, User A is a regular visitor to website B. B continuously collects vast amounts of data from A's mouse and keystroke interactions. By employing appropriate techniques which are introduced in Chapters 6, 7, 8 and 9, B has successfully transformed this data into a distinctive user template, essentially capturing the unique behaviour signature of User A.

Website B shares user A's template with Website C for some reasons. Armed with User A's template, C can easily identify User A during visits, even in private browsing mode, which means the user's privacy has been breached. Once the user's identity is confirmed, C can also share such data with other potential third parties.

10.3.3 Consequences

The aforementioned scenarios pose potential detrimental consequences, especially when the user's identity falls into the wrong hands. This risks the user's privacy in several ways:

- Targeted advertising: A website can present highly personalised ads.
- Third-party sharing: Once the user's template is established, the website could potentially sell the user's information to third parties without the user's consent.
- Phishing and Social engineering: With access to personalized information, attackers can deploy more convincing and dangerous phishing and social engineering attacks, making users more vulnerable.

10.3.4 Discussions

It is important to note that a single website is typically unable to track visitors across multiple sites, which means that the experiments we conducted may not yield reasonable data to support the above mentioned attack scenarios. In our experiments, the user data collected was derived from various websites. The user template in our experiments reflects general keystroke and mouse usage patterns rather than being specific to a single site, whereas in scenarios One and Two, the adversary is a specific website.

User behaviour may be more consistent across similar types of sites, such as news sites like BBC and CNN. In scenario Two, if websites B and C belong to similar categories and offer similar content, user behaviour may remain relatively consistent.

On the other hand, user behaviour can vary significantly across different webpages; for example, interacting with Gmail versus browsing BBC News. It might be possible to divide the users' experimental data by analysing the cursor position and the browser size. However, categorizing the webpage content visited by users is a challenging task:

- It is technically difficult to initially distinguish user activity by capturing URL entries within the address bar. The *Event listener* used in this study responds to *keydown* and *keyup*. Theses APIs are only active when the user interacts with the main content of the web page. The address bar, located outside the main content area, does not trigger these APIs. Additionally, analysing the keystrokes within the address bar may raise significant privacy and ethical concerns. For these reasons, it is very unlikely to ascertain tab-switching behaviour by analysing URL addresses.

- The event handlers *Mousemove*, *mouseup* and *mousedown* are restricted to interactions within the main body of a webpage. Since browser tabs reside above the main content area, they fall outside the scope of these listeners. While a cursor's Y-coordinate may return '0' when moved beyond the upper boundary of the web content area, this does not provide reliable information about whether a user has clicked to change tabs. Consequently, it is unlikely that mouse activity alone will accurately indicate tab-switching events.

- A user may switch applications by holding the *Alt* key and then pressing the *Tab* key. In such instances, the webpage or extension (in this study) can only log the activity of the *Alt* key; it cannot detect the subsequent *Tab* keystroke. Once the application switch is complete, the web page loses focus, leading to the detection of a successful switch to a different application or browser tab. However, in the new switched tab, web page only gets focus when the user interact with it in the main content area, which means it is very difficult to ascertain the type of the tab-switching activity.

Considering the aforementioned limitations, it is very difficult and challenging to distinguish the user's webpage content based on tab-switching behaviour. Assigning specific tasks to users may probably be a easier way to access to such data. However, that approach would require a different experimental design within a controlled environment, which vary from the methodology of this study.

10.4 Recommendations

Building on the findings reported in Section 10.2, we propose the following recommendations for browser users and manufacturers.

On a PC platform, none of the browsers we examined require user consent to access the *KeyboardEvent* and *MouseEvent* APIs. Additionally, disabling the *KeyboardEvent* and *MouseEvent* APIs may negatively affect important functionality of a web page. Based on these two points, we give the following suggestions.

- We suggest that browser providers can let the users decide whether they are willing to grant such permissions.

- We suggest the browser users try to behave in a new way such as changing keystroke typing habits or changing mouse moving speed, etc.

For a mobile plaform, we strongly recommend that the *Orientation* and *Motion* permissions are not granted to untrusted websites, except in special circumstances.

10.5 Possible positive usage

Contrasting with the security and privacy issues posed by browser APIs, these APIs may also be used for genuine purpose.

- In an email system, if the typing or mouse using behaviour of the email sender is significantly different from usual, then a security check in background could be conducted.
- For a website, if a user is using web scraping techniques and generating significantly unusual clicking behaviour, then he/she could be marked and security check could be conducted.

10.6 Conclusions

In this chapter, we discussed the security and privacy issues posed by various browser APIs. We first examined various browsers on a range of platforms. We then demonstrated potential attack scenarios and at last we proposed possible suggestions for counter-measures.

Chapter 11

Conclusions and possible future work

11.1 Summary and conclusions

This work described in this thesis was motivated by the growing privacy threats arising from the functionality in Browser APIs and the availability of modern machine learning techniques.

In Chapter 2, we outlined the relevant aspects of Browser APIs, and reviewed the security and privacy issues posed by such APIs. Chapter 3 introduced the machine learning techniques relevant to the analysis in this thesis. In Chapter 4, we introduced the behavioural biometrics based on keystroke dynamics, mouse dynamics, and mobile sensors data.

In Chapter 5, we described the functionality and implementation of the experimental platform that was developed to enable investigation of the possible privacy threat posed by user-activity-related sensors. We developed a Chrome extension for collecting user keystroke and mouse dynamics data and set up a server for receiving data from the extension and storing such data.

We conducted experiments for collecting users' keyboard and mouse dynamics data with the experimental platform described in Chapter 5. We invited 21 participants who are all bilingual (Chinese native and English proficient) in the experiments. Each user used the experimental tool for a period from four to six months. The experimental environment was totally free of controls. 20 of them provided sufficient data which were used for data analysis in Chapters 6 and 7.

In Chapter 6, we provided a description of the application of machine learning techniques (*XGBoost*) for classification tasks based on keystroke dynamics data collected from 20 users. The keystroke dynamics data was used to generate 4-graph samples incorporating various features including timing information and function key use. The f1-score was selected as the measuring metrics for this multiclass classification task. The result based on four keystrokes (one single 4-graph sample) was 49.6%, and the result based on 20 consecutive keystroke inputs was 89.2%. The results indicate that the web users can be identified via their typing habits without their knowledge in a browser-based environment, and the identification accuracy improves with more keystroke inputs within a single identifying session.

In Chapter 7, we provided a description of the application of machine learning techniques (*XGBoost*) for classification tasks based on mouse dynamics data collected from 20 users. The raw mouse data was processed and split into segments. Each segment represents a single mouse action (point-and-click) as one sample. For each sample, various features were extracted based on timing and mouse coordinate information. The f1-score was selected as the measuring metrics for this multiclass classification task. The result based on one single mouse action was 59.4%. By combining five mouse actions, we achieved a result of 87.4% which increased further to 94.2% when the action times were increased to 10. We also worked with a Bogazici dataset containing data for a set of 19 users and with a larger data size and achieved promising experimental results. The results for a single mouse action and 10 mouse actions were 44.5% and 95.0% respectively. These results strongly suggest that a web user could be identified via their mouse use without his/her knowledge in a browser-based environment.

In Chapter 8, we provided a description of the application of machine learning techniques (*XGBoost*) for classification tasks based on combining keystroke and mouse dynamics data collected from 20 users used in Chapters 6 and 7. The F1-score was selected as the measuring metrics for this multiclass classification task. The result based on one single mouse action and one 4-graph keystroke sample was 72.7%. This performance is better than solely using either keystroke dynamics or mouse dynamics. By combining five mouse actions and 5 keystrokes (two 4-grpah samples), we achieve a result of 93.5%. These results suggest that a web user could be identified via their keystroke and mouse use without his/her knowledge in a browser-based environment, and combining keystroke and mouse dynamics can improve the identification performance.

In Chapter 9, we investigated the application of machine learning techniques (*XGBoost*) for classification tasks on the public mobile keystroke dataset from the work of Kim and Kang [51]. The raw data was processed into digraph samples. From each sample, timing information and 3-dimensional coordinates based on *Orientation* and *Motion* sensors were extracted. The f1-score was selected as the measuring metrics of this multiclass classification task. The results based on 1 sample and 10 samples were 59.8% and 96.1% respectively. These results suggest that mobile device sensor data could be used for user identification. Contrasting with previous work, which has mostly been based on special-purpose Android apps, we developed a web page which can collect the same type of user data via mobile browser APIs as were gathered by Kim and Kang [51]. Based on the machine learning results and the web page we developed, we concluded that it may well be possible to discover the identity of a user when they are using mobile browser.

Building on the findings reported in Chapters 6, 7, 8 and 9, in Chapter 10 we examined a range of browsers running on various platforms. We concluded that all the browsers we examined potentially allow compromise of a user identity via the browser APIs in normal mode, and even in privacy-protected mode. Additionally, we described the possible attack models. We considered the effectiveness of existing privacy-relevant counter-measures against such privacy issues. It is important to observe that disabling the *KeyboardEvent* and *MouseEvent* APIs may adversely affect important functionality of a web page, and hence this simple countermeasure needs to applied judiciously. We therefore suggest allowing users to decide whether they are willing to grant a visited website permissions to access to the *KeyboardEvent* and *MouseEvent* APIs on a PC platform. For a mobile platform, we strongly recommend that *Orientation* and *Motion* permissions are not granted to untrusted websites, except in special circumstances.

11.2 Possible future work

In Chapter 5, we introduced a novel experimental platform for collecting the data necessary to conduct the experiments. Because sensitive information may be recorded during the experiments, only relatively few participants were willing to take part in the experiments, although it was formally stated to participants that all the collected data would be highly confidential and be accessed only by the author and only for the specified purpose. This limited the number of users involved in our experiments. Clearly, if we could work with larger datasets, we could obtain more definitive results.

It would also be very valuable to collect user data from different working scenarios. In the experiments described in this thesis, all users used their own devices, which means that they are likely to have a high level of proficiency in use of their device while participating the experiments. Hence, it would be interesting to require all the users to use the same device to provide testing data and then see how the machine learning model performs with such new data.

In Chapters 6 and 7, the use of sophisticated machine learning algorithms during the feature engineering stage might improve the quality of extracted features and improve the classification performance. In the context of the work described in Chapter 9, it would be better to conduct an experiment for collecting data via the implemented webpage, although designing a mobile web page with sufficiently rich features for generating meaningful user interactions would be a very difficult and challenging task. Additionally, recruiting participants in a face-to-face environment could be problematic in a pandemic-affected world. We hope that such an experiment could be conducted in the future. Given that browsers continue to evolve rapidly, it is likely that new API functionality will continue to emerge; in this context, it is vitally important to monitor such new functionality to see if novel, and possible unexpected, user privacy and security threats emerge.

Bibliography

[1] Alejandro Acien, Aythami Morales, John V Monaco, Ruben Vera-Rodriguez, and Julian Fierrez. Typenet: Deep learning keystroke biometrics. *IEEE Transactions on Biometrics, Behavior, and Identity Science*, 4(1):57–70, 2021.

[2] T Agrawal. Hyperparameter optimization in machine learning: Make your machine learning and deep learning models more efficient. *Apress: New York, NY, USA*, 2020.

[3] Ahmed A. Ahmed and Issa Traoré. Biometric recognition based on free-text keystroke dynamics. *IEEE Trans. Cybernetics*, 44(4):458–472, 2014.

[4] Ahmed Awad E. Ahmed and Issa Traoré. A new biometric technology based on mouse dynamics. *IEEE Trans. Dependable Sec. Comput.*, 4(3):165–179, 2007.

[5] N. M. Al-Fannah, W. Li, and C. J. Mitchell. Beyond cookie monster amnesia: Real world persistent online tracking. *To appear in: Proceedings of ISC 2018: The 21st Information Security Conference Guildford, Surrey, UK, September 9-12, 2018.*

[6] Nasser Mohammed Al-Fannah. One leak will sink a ship: Webrtc IP address leaks. In *International Carnahan Conference on Security Technology, ICCST 2017, Madrid, Spain, October 23-26, 2017*, pages 1–5. IEEE, 2017.

[7] Nasser Mohammed Al-Fannah and Wanpeng Li. Not all browsers are created equal: comparing web browser fingerprintability. In *International Workshop on Security*, pages 105–120. Springer, 2017.

[8] Furkan Alaca and Paul C. van Oorschot. Device fingerprinting for augmenting web authentication: classification and analysis of methods. In Stephen Schwab, William K. Robertson, and Davide Balzarotti, editors, *Proceedings of the 32nd Annual Conference on Computer Security Applications, ACSAC 2016, Los Angeles, CA, USA, December 5-9, 2016*, pages 289–301. ACM, 2016.

[9] Najwa Altwaijry. Authentication by keystroke dynamics: The influence of typing language. *Applied Sciences*, 13(20):11478, 2023.

[10] Margit Antal and Elod Egyed-Zsigmond. Intrusion detection using mouse dynamics. *IET Biometrics*, 02 2019.

[11] Margit Antal, Norbert Fejér, and Krisztian Buza. Sapimouse: Mouse dynamics-based user authentication using deep feature learning. In *2021 IEEE 15th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, pages 61–66. IEEE, 2021.

[12] Margit Antal, László Zsolt Szabó, and Izabella László. Keystroke dynamics on android platform. *Procedia Technology*, 19:820–826, 2015.

[13] Lívia C. F. Araújo, Luiz H. R. Sucupira, Miguel Gustavo Lizárraga, Lee Luan Ling, and João Baptista T. Yabu-uti. User authentication through typing biometrics features. *IEEE Trans. Signal Processing*, 53(2-2):851–855, 2005.

[14] Kyle O. Bailey, James S. Okolica, and Gilbert L. Peterson. User identification and authentication using multi-modal behavioral biometrics. *Computers & Security*, 43:77–89, 2014.

[15] Nick Bartlow and Bojan Cukic. Evaluating the reliability of credential hardening through keystroke dynamics. In *17th International Symposium on Software Reliability Engineering (ISSRE 2006), 7-10 November 2006, Raleigh, North Carolina, USA*, pages 117–126, 2006.

[16] Hristo Bojinov, Yan Michalevsky, Gabi Nakibly, and Dan Boneh. Mobile device identification via sensor fingerprinting. *arXiv preprint arXiv:1408.1416*, 2014.

[17] Liang Cai and Hao Chen. Touchlogger: Inferring keystrokes on touch screen from smartphone motion. In Patrick D. McDaniel, editor, *6th USENIX Workshop on Hot Topics in Security, HotSec'11, San Francisco, CA, USA, August 9, 2011*. USENIX Association, 2011.

[18] Ting-Yi Chang, Cheng-Jung Tsai, and Jyun-Hao Lin. A graphical-based password keystroke dynamic authentication system for touch screen handheld mobile devices. *Journal of Systems and Software*, 85(5):1157–1165, 2012.

[19] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[20] Chrome. Chrome JavaScript Web APIs. `https://developer.chrome.com/apps/api_other`. Accessed: 28-February-2018.

[21] Chrome. Chrome Update log. `https://bugs.chromium.org/p/chromium/issues/detail?id=158234#c110`. Accessed: 28-February-2018.

[22] Michael Crawford, Taghi M Khoshgoftaar, Joseph D Prusa, Aaron N Richter, and Hamzah Al Najada. Survey of review spam detection using machine learning techniques. *Journal of Big Data*, 2(1):1–24, 2015.

[23] Pratap Dangeti. *Statistics for machine learning.* Packt Publishing Ltd, 2017.

[24] Ignacio de Mendizabal-Vazquez, Daniel de Santos-Sierra, Javier Guerra-Casanova, and Carmen Sánchez-Ávila. Supervised classification methods applied to keystroke dynamics through mobile devices. In *2014 International Carnahan conference on security technology (ICCST)*, pages 1–6. IEEE, 2014.

[25] Thomas G Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning*, 40(2):139–157, 2000.

[26] Benjamin Draffin, Jiang Zhu, and Joy Zhang. Keysens: Passive user authentication through micro-behavior modeling of soft keyboard interaction. In *International Conference on Mobile Computing, Applications, and Services*, pages 184–201. Springer, 2013.

[27] Sally Earl, James Campbell, and Oliver Buckley. Identifying soft biometric features from a combination of keystroke and mouse dynamics. In *International Conference on Applied Human Factors and Ergonomics*, pages 184–190. Springer, 2021.

[28] Peter Eckersley. How unique is your web browser? In Mikhail J. Atallah and Nicholas J. Hopper, editors, *Privacy Enhancing Technologies, 10th International Symposium, PETS 2010, Berlin, Germany, July 21-23, 2010. Proceedings*, volume 6205 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2010.

[29] Aniekan Essien, Ilias Petrounias, Pedro Sampaio, and Sandra Sampaio. A deep-learning model for urban traffic flow prediction with traffic events mined from twitter. *World Wide Web*, 24(4):1345–1368, 2021.

[30] Meherwar Fatima, Maruf Pasha, et al. Survey of machine learning algorithms for disease diagnostic. *Journal of Intelligent Learning Systems and Applications*, 9(01):1, 2017.

[31] Clint Feher, Yuval Elovici, Robert Moskovitch, Lior Rokach, and Alon Schclar. User identity verification via mouse dynamics. *Inf. Sci.*, 201:19–36, 2012.

[32] David Fifield and Serge Egelman. Fingerprinting web users through font metrics. In Rainer Böhme and Tatsuaki Okamoto, editors, *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, volume 8975 of *Lecture Notes in Computer Science*, pages 107–124. Springer, 2015.

[33] Á. Fülöp, L. Kovács, T. Kurics, and E. Windhager-Pokol. Balabit mouse dynamics challenge data set. https://github.com/balabit/Mouse-Dynamics-Challenge, 2016.

[34] Hugo Gascon, Sebastian Uellenbeck, Christopher Wolf, and Konrad Rieck. Continuous authentication on mobile devices by analysis of typing motion behavior. *Sicherheit 2014–Sicherheit, Schutz und Zuverlässigkeit*, 2014.

[35] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[36] Romain Giot, Mohamad El-Abed, and Rosenberger Christophe. Greyc keystroke: a benchmark for keystroke dynamics biometric systems. In *IEEE International Conference on Biometrics: Theory, Applications and Systems (BTAS 2009)*, Washington, District of Columbia, USA, 2009. IEEE Computer Society.

[37] Romain Giot, Mohamad El-Abed, Baptiste Hemery, and Christophe Rosenberger. Unconstrained keystroke dynamics authentication with shared secret. *Computers & Security*, 30(6-7):427–445, 2011.

[38] Romain Giot, Mohamad El-Abed, and Christophe Rosenberger. Keystroke dynamics authentication for collaborative systems. In *2009 International Symposium on Collaborative Technologies and Systems, CTS 2009, Baltimore, Maryland, USA, May 18-22, 2009*, pages 172–179, 2009.

[39] Daniel Gruss, David Bidner, and Stefan Mangard. Practical memory deduplication attacks in sandboxed javascript. In Günther Pernul, Peter Y. A. Ryan, and Edgar R. Weippl, editors, *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part I*, volume 9326 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2015.

[40] Daniele Gunetti and Claudia Picardi. Keystroke analysis of free text. *ACM Trans. Inf. Syst. Secur.*, 8(3):312–347, 2005.

[41] Grant Ho et al. Tapdynamics: strengthening user authentication on mobile phones with keystroke dynamics. *Technicalreport, StanfordUniversity*, 73:74, 2014.

[42] Wandong Hong, Xiaoying Zhou, Shengchun Jin, Yajing Lu, Jingyi Pan, Qingyi Lin, Shaopeng Yang, Tingting Xu, Zarrin Basharat, Maddalena Zippi, et al. A comparison of xgboost, random forest, and nomograph for the prediction of disease severity in patients with covid-19 pneumonia: implications of cytokine and immune cell profile. *Frontiers in cellular and infection microbiology*, 12:819267, 2022.

[43] Danoush Hosseinzadeh and Sridhar Sri Krishnan. Gaussian mixture modeling of keystroke patterns for biometric applications. *IEEE Trans. Systems, Man, and Cybernetics, Part C*, 38(6):816–826, 2008.

[44] Anil Jain, Lin Hong, and Sharath Pankanti. Biometric identification. *Communications of the ACM*, 43(2):90–98, 2000.

[45] Lohit Jain, John V Monaco, Michael J Coakley, and Charles C Tappert. Passcode keystroke biometric performance on smartphone touchscreens is superior to that on hardware keyboards. *International Journal of Research in Computer Applications & Information Technology*, 2(4):29–33, 2014.

[46] Pilsung Kang and Sungzoon Cho. Keystroke dynamics-based user authentication using long and free text strings from various input devices. *Inf. Sci.*, 308:72–93, 2015.

[47] M. Karnan, M. Akila, and N. Krishnaraj. Biometric personal authentication using keystroke dynamics: A review. *Appl. Soft Comput.*, 11(2):1565–1573, 2011.

[48] Pawel Kasprowski, Zaneta Borowska, and Katarzyna Harezlak. Biometric identification based on keystroke dynamics. *Sensors*, 22(9):3158, 2022.

[49] Kevin S. Killourhy and Roy A. Maxion. Comparing anomaly-detection algorithms for keystroke dynamics. In *Proceedings of the 2009 IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2009, Estoril, Lisbon, Portugal, June 29 - July 2, 2009*, pages 125–134, 2009.

[50] Kevin S. Killourhy and Roy A. Maxion. Why did my detector do *That*?! - predicting keystroke-dynamics error rates. In Somesh Jha, Robin Sommer, and Christian Kreibich, editors, *Recent Advances in Intrusion Detection, 13th International Symposium, RAID 2010, Ottawa, Ontario, Canada, September 15-17, 2010. Proceedings*, volume 6307 of *Lecture Notes in Computer Science*, pages 256–276. Springer, 2010.

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

[52] Arjen Aykan Kılıç, Metehan Yıldırım, and Emin Anarım. Bogazici mouse dynamics dataset. *Data in Brief*, 36:107094, 2021.

[53] Hyungu Lee, Jung Yeon Hwang, Dong In Kim, Shincheol Lee, Sung-Hoon Lee, and Ji Sun Shin. Understanding keystroke dynamics for smartphone users authentication and keystroke dynamics on smartphones built-in motion sensors. *Security and Communication Networks*, 2018, 2018.

[54] Moritz Lipp, Daniel Gruss, Michael Schwarz, David Bidner, Clémentine Maurice, and Stefan Mangard. Practical keystroke timing attacks in sandboxed javascript. In Simon N. Foley, Dieter Gollmann, and Einar

Snekkenes, editors, *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II*, volume 10493 of *Lecture Notes in Computer Science*, pages 191–209. Springer, 2017.

[55] Chao-Liang Liu, Cheng-Jung Tsai, Ting-Yi Chang, Wang-Jui Tsai, and Po-Kai Zhong. Implementing multiple biometric features for a recall-based graphical keystroke dynamics authentication system on a smart phone. *J. Network and Computer Applications*, 53:128–139, 2015.

[56] Xiaofeng Lu, Shengfei Zhang, Pan Hui, and Pietro Lio. Continuous authentication by free-text keystroke based on cnn and rnn. *Computers & Security*, 96:101861, 2020.

[57] Dorothy Martin and Soo See Chai. A study on performance comparisons between knn, random forest and xgboost in prediction of landslide susceptibility in kota kinabalu, malaysia. In *2022 IEEE 13th Control and System Graduate Research Colloquium (ICSGRC)*, pages 159–164. IEEE, 2022.

[58] Maryam Mehrnezhad, Ehsan Toreini, Siamak Fayyaz Shahandashti, and Feng Hao. Touchsignatures: Identification of user touch actions and pins based on mobile sensor data via javascript. *J. Inf. Sec. Appl.*, 26:23–38, 2016.

[59] Soumik Mondal and Patrick Bours. A computational approach to the continuous authentication biometric system. *Inf. Sci.*, 304:28–53, 2015.

[60] Soumik Mondal and Patrick Bours. A study on continuous authentication using a combination of keystroke and mouse biometrics. *Neurocomputing*, 230:1–22, 2017.

[61] Mozilla. Client-side Web APIs. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction. Accessed: 28-February-2018.

[62] Mozilla. Resources for developers. https://developer.mozilla.org/en-US/docs/Web/API. Accessed: 30-01-2022.

[63] Youssef Nakkabi, Issa Traoré, and Ahmed Awad E. Ahmed. Improving mouse dynamics biometric performance using variance reduction via extractors with separate features. *IEEE Trans. Systems, Man, and Cybernetics, Part A*, 40(6):1345–1353, 2010.

[64] Sashank Narain, Amirali Sanatinia, and Guevara Noubir. Single-stroke language-agnostic keylogging using stereo-microphones and domain specific machine learning. In Gergely Ács, Andrew P. Martin, Ivan Martinovic, Claude Castelluccia, and Patrick Traynor, editors, *7th ACM Conference on Security & Privacy in Wireless and Mobile Networks, WiSec'14, Oxford, United Kingdom, July 23-25, 2014*, pages 201–212. ACM, 2014.

[65] Yossef Oren, Vasileios P. Kemerlis, Simha Sethumadhavan, and Angelos D. Keromytis. The spy in the sandbox: Practical cache attacks in javascript and their implications. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 1406–1418. ACM, 2015.

[66] Kseniia Palin, Anna Maria Feit, Sunjun Kim, Per Ola Kristensson, and Antti Oulasvirta. How do people type on mobile devices? observations from a study with 37,000 volunteers. In *Proceedings of the 21st International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 1–12, 2019.

[67] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[68] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[69] Paulo Henrique Pisani and Ana Carolina Lorena. A systematic review on keystroke dynamics. *J. Braz. Comp. Soc.*, 19(4):573–587, 2013.

[70] Md Muhaimenur Rahman and Sarnali Basak. Identifying user authentication and most frequently used region based on mouse movement data: A machine learning approach. In *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 1245–1250. IEEE, 2021.

[71] Ricardo N. Rodrigues, Glauco F. G. Yared, Carlos R. do N. Costa, João Baptista T. Yabu-uti, Fábio Violaro, and Lee Luan Ling. Biometric access control through numerical keyboards based on keystroke dynamics. In *Advances in Biometrics, International Conference, ICB 2006, Hong Kong, China, January 5-7, 2006, Proceedings*, pages 640–646, 2006.

[72] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach, eBook, Global Edition.* Pearson Education, 2021.

[73] S.R. Safavian and D. Landgrebe. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3):660–674, 1991.

[74] Iqbal H Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, 2(3):1–21, 2021.

[75] Sougata Sen and Kartik Muralidharan. Putting 'pressure'on mobile authentication. In *2014 seventh International Conference on mobile computing and ubiquitous networking (ICMU)*, pages 56–61. IEEE, 2014.

[76] Chao Shen, Zhongmin Cai, Xiaohong Guan, Youtian Du, and Roy A. Maxion. User authentication through mouse dynamics. *IEEE Trans. Information Forensics and Security*, 8(1):16–30, 2013.

[77] Nyle Siddiqui, Rushit Dave, and Naeem Seliya. Continuous user authentication using mouse dynamics, machine learning, and minecraft. In *2021 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, pages 1–6. IEEE, 2021.

[78] Laurent Simon and Ross J. Anderson. PIN skimmer: inferring pins through the camera and microphone. In William Enck, Adrienne Porter Felt, and N. Asokan, editors, *SPSM'13, Proceedings of the 2013 ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, Co-located with CCS 2013, November 8, 2013, Berlin, Germany*, pages 67–78. ACM, 2013.

[79] Raphael Spreitzer. PIN skimming: Exploiting the ambient-light sensor in mobile devices. In Cliff Wang, Dijiang Huang, Kl Singh, and Zhenkai Liang, editors, *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices, SPSM@CCS 2014, Scottsdale, AZ, USA, November 03 - 07, 2014*, pages 51–62. ACM, 2014.

[80] Giuseppe Stragapede, Paula Delgado-Santos, Ruben Tolosana, Ruben Vera-Rodriguez, Richard Guest, and Aythami Morales. Mobile keystroke biometrics using transformers. In *2023 IEEE 17th International Conference on Automatic Face and Gesture Recognition (FG)*, pages 1–6. IEEE, 2023.

[81] Shridatt Sugrim, Can Liu, Meghan McLean, and Janne Lindqvist. Robust performance metrics for authentication systems. In *Network and Distributed Systems Security (NDSS) Symposium 2019*, 2019.

[82] Yan Sun, Hayreddin Ceker, and Shambhu Upadhyaya. Shared keystroke dataset for continuous authentication. In *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6. IEEE, 2016.

[83] Nishtha H Tandel, Harshadkumar B Prajapati, and Vipul K Dabhi. Voice recognition and voice comparison using machine learning techniques: A survey. In *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 459–465. IEEE, 2020.

[84] Pin Shen Teh, Ning Zhang, Andrew Beng Jin Teoh, and Ke Chen. Recognizing your touch: Towards strengthening mobile device authentication via touch dynamics integration. In *Proceedings of the 13th International Conference on Advances in Mobile Computing and Multimedia*, pages 108–116, 2015.

[85] Issa Traore, Isaac Woungang, Mohammad S Obaidat, Youssef Nakkabi, and Iris Lai. Combining mouse and keystroke dynamics biometrics for risk-based authentication in web environments. In *2012 fourth international conference on digital home*, pages 138–145. IEEE, 2012.

[86] J.A. Unar, Woo Chaw Seng, and Almas Abbasi. A review of biometric technology along with trends and prospects. *Pattern Recognition*, 47(8):2673–2688, 2014.

[87] Tom van Goethem, Wouter Joosen, and Nick Nikiforakis. The clock is still ticking: Timing attacks in the modern web. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 1382–1393. ACM, 2015.

[88] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. Fp-stalker: Tracking browser fingerprint evolutions. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 728–741. IEEE, 2018.

[89] Esra Vural, Jiaju Huang, Daqing Hou, and Stephanie Schuckers. Shared research dataset to support development of keystroke authentication. In *IEEE International joint conference on biometrics*, pages 1–8. IEEE, 2014.

[90] W3C. W3C JavaScript Web APIs. https://www.w3.org/standards/webdesign/script.html. Accessed: 28-February-2018.

[91] Xiujuan Wang, Yutong Shi, Kangfeng Zheng, Yuyang Zhang, Weijie Hong, and Siwei Cao. User authentication method based on keystroke dynamics and mouse dynamics with scene-irrelated features in hybrid scenes. *Sensors*, 22(17):6627, 2022.

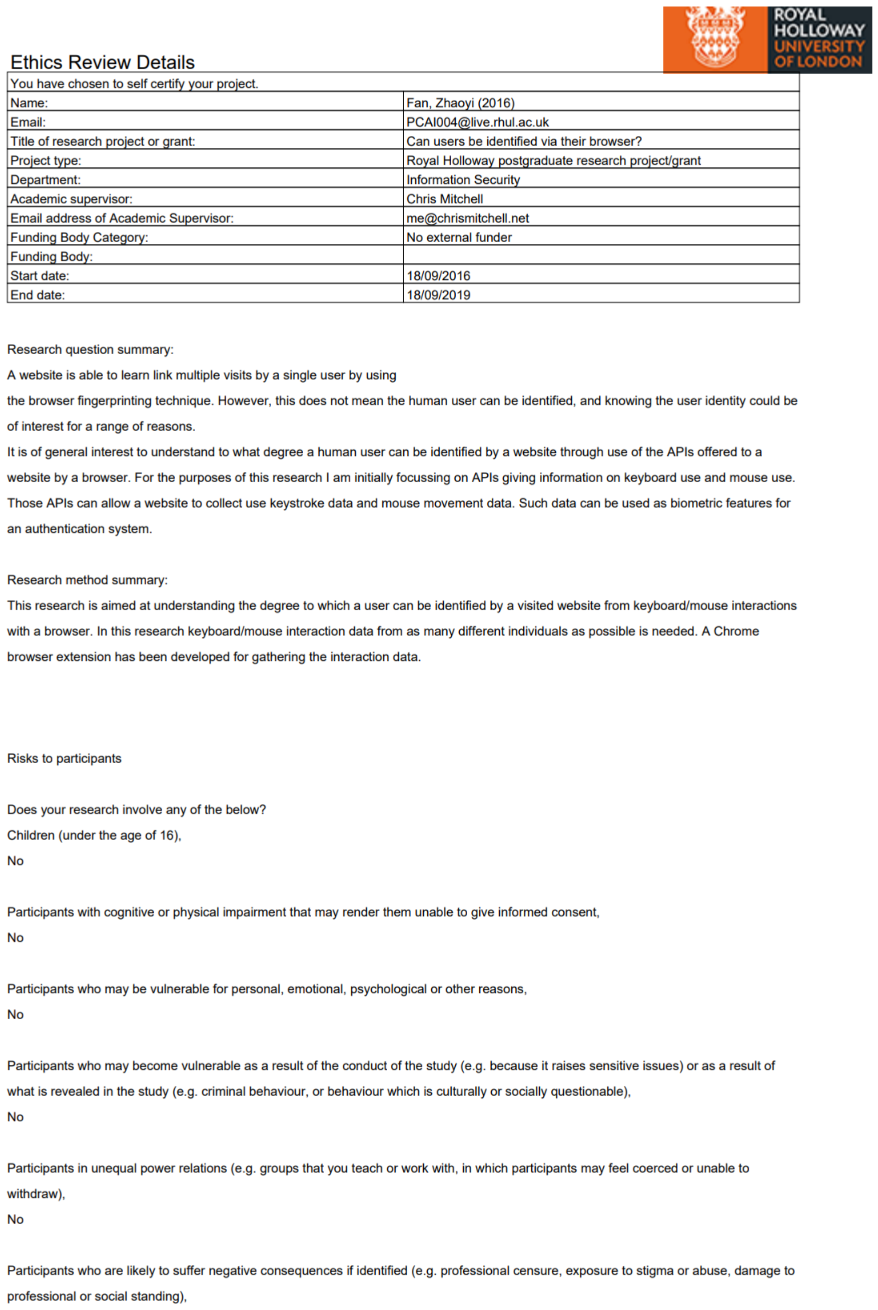
[92] Shaomin Wu. A review on coarse warranty data and analysis. *Reliability Engineering & System Safety*, 114:1–11, 2013.

[93] Xiaomei Zhang, Pengming Zhang, and Haomin Hu. Multimodal continuous user authentication on mobile devices via interaction patterns. *Wireless Communications and Mobile Computing*, 2021:1–15, 2021.

[94] Nan Zheng, Kun Bai, Hai Huang, and Haining Wang. You are how you touch: User verification on smartphones via tapping behaviors. In *2014 IEEE 22nd International Conference on Network Protocols*, pages 221–232. IEEE, 2014.

Appendix A

Ethics review approval



ROYAL HOLLOWAY UNIVERSITY OF LONDON

Ethics Review Details

You have chosen to self certify your project.	
Name:	Fan, Zhaoyi (2016)
Email:	PCAI004@live.rhul.ac.uk
Title of research project or grant:	Can users be identified via their browser?
Project type:	Royal Holloway postgraduate research project/grant
Department:	Information Security
Academic supervisor:	Chris Mitchell
Email address of Academic Supervisor:	me@chrismitchell.net
Funding Body Category:	No external funder
Funding Body:	
Start date:	18/09/2016
End date:	18/09/2019

Research question summary:

A website is able to learn link multiple visits by a single user by using
the browser fingerprinting technique. However, this does not mean the human user can be identified, and knowing the user identity could be of interest for a range of reasons.

It is of general interest to understand to what degree a human user can be identified by a website through use of the APIs offered to a website by a browser. For the purposes of this research I am initially focussing on APIs giving information on keyboard use and mouse use. Those APIs can allow a website to collect use keystroke data and mouse movement data. Such data can be used as biometric features for an authentication system.

Research method summary:

This research is aimed at understanding the degree to which a user can be identified by a visited website from keyboard/mouse interactions with a browser. In this research keyboard/mouse interaction data from as many different individuals as possible is needed. A Chrome browser extension has been developed for gathering the interaction data.

Risks to participants

Does your research involve any of the below?

Children (under the age of 16),

No

Participants with cognitive or physical impairment that may render them unable to give informed consent,

No

Participants who may be vulnerable for personal, emotional, psychological or other reasons,

No

Participants who may become vulnerable as a result of the conduct of the study (e.g. because it raises sensitive issues) or as a result of what is revealed in the study (e.g. criminal behaviour, or behaviour which is culturally or socially questionable),

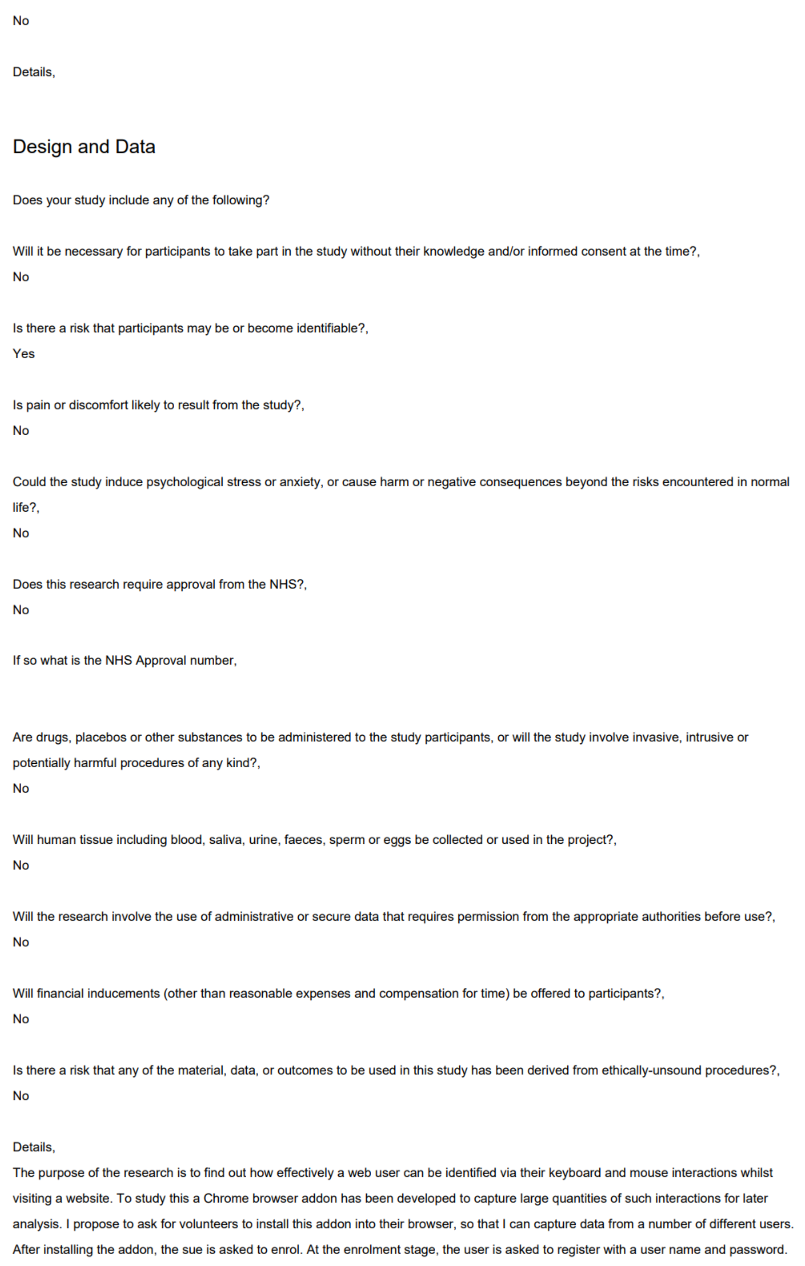
No

Participants in unequal power relations (e.g. groups that you teach or work with, in which participants may feel coerced or unable to withdraw),

No

Participants who are likely to suffer negative consequences if identified (e.g. professional censure, exposure to stigma or abuse, damage to professional or social standing),

Figure A.1: Page 1 of ethics review approval



No

Details,

Design and Data

Does your study include any of the following?

Will it be necessary for participants to take part in the study without their knowledge and/or informed consent at the time?,
No

Is there a risk that participants may be or become identifiable?,
Yes

Is pain or discomfort likely to result from the study?,
No

Could the study induce psychological stress or anxiety, or cause harm or negative consequences beyond the risks encountered in normal life?,
No

Does this research require approval from the NHS?,
No

If so what is the NHS Approval number,

Are drugs, placebos or other substances to be administered to the study participants, or will the study involve invasive, intrusive or potentially harmful procedures of any kind?,
No

Will human tissue including blood, saliva, urine, faeces, sperm or eggs be collected or used in the project?,
No

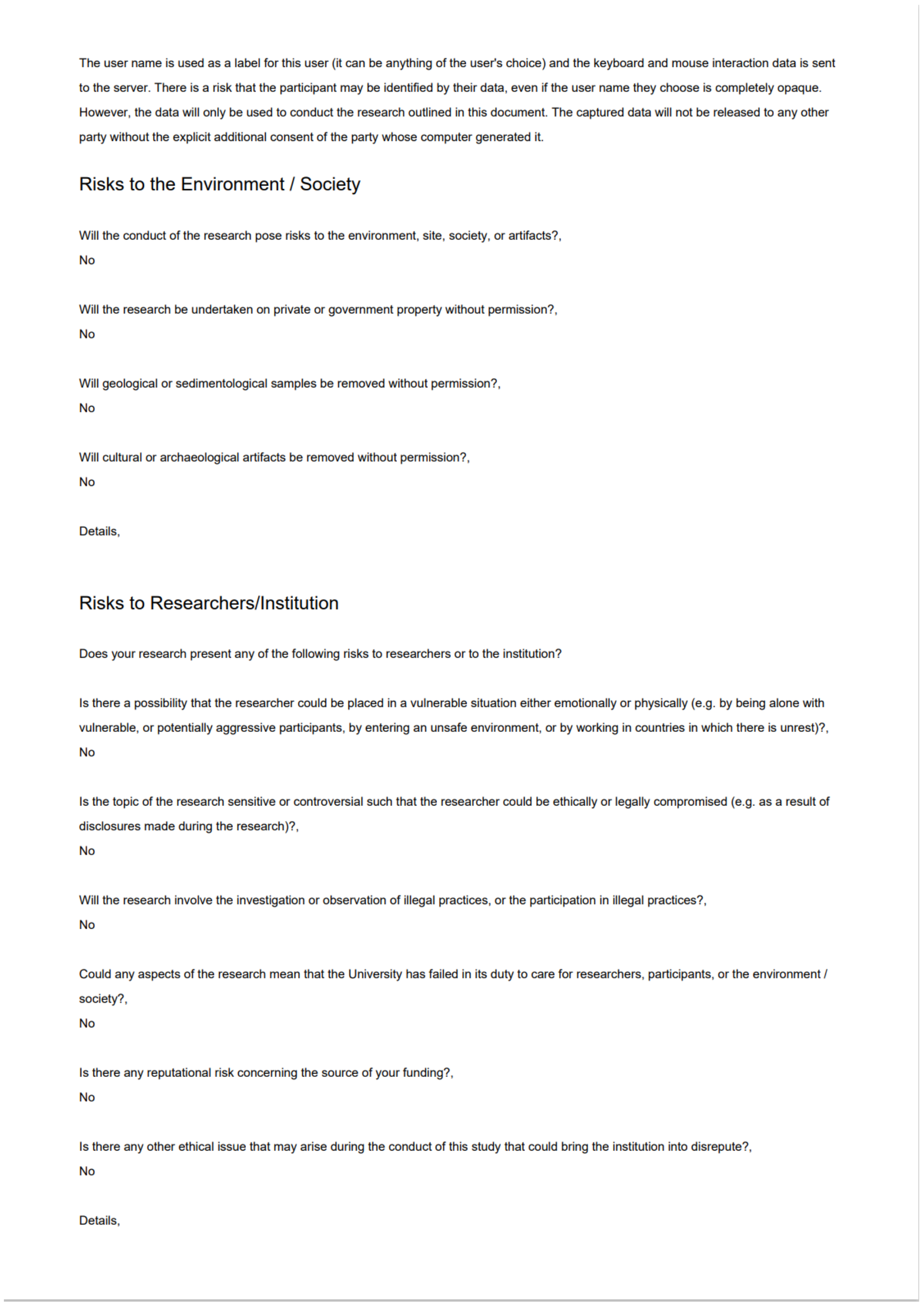
Will the research involve the use of administrative or secure data that requires permission from the appropriate authorities before use?,
No

Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?,
No

Is there a risk that any of the material, data, or outcomes to be used in this study has been derived from ethically-unsound procedures?,
No

Details,
The purpose of the research is to find out how effectively a web user can be identified via their keyboard and mouse interactions whilst visiting a website. To study this a Chrome browser addon has been developed to capture large quantities of such interactions for later analysis. I propose to ask for volunteers to install this addon into their browser, so that I can capture data from a number of different users. After installing the addon, the sue is asked to enrol. At the enrolment stage, the user is asked to register with a user name and password.

Figure A.2: Page 2 of ethics review approval



The user name is used as a label for this user (it can be anything of the user's choice) and the keyboard and mouse interaction data is sent to the server. There is a risk that the participant may be identified by their data, even if the user name they choose is completely opaque. However, the data will only be used to conduct the research outlined in this document. The captured data will not be released to any other party without the explicit additional consent of the party whose computer generated it.

Risks to the Environment / Society

Will the conduct of the research pose risks to the environment, site, society, or artifacts?,
No

Will the research be undertaken on private or government property without permission?,
No

Will geological or sedimentological samples be removed without permission?,
No

Will cultural or archaeological artifacts be removed without permission?,
No

Details,

Risks to Researchers/Institution

Does your research present any of the following risks to researchers or to the institution?

Is there a possibility that the researcher could be placed in a vulnerable situation either emotionally or physically (e.g. by being alone with vulnerable, or potentially aggressive participants, by entering an unsafe environment, or by working in countries in which there is unrest)?,
No

Is the topic of the research sensitive or controversial such that the researcher could be ethically or legally compromised (e.g. as a result of disclosures made during the research)?,
No

Will the research involve the investigation or observation of illegal practices, or the participation in illegal practices?,
No

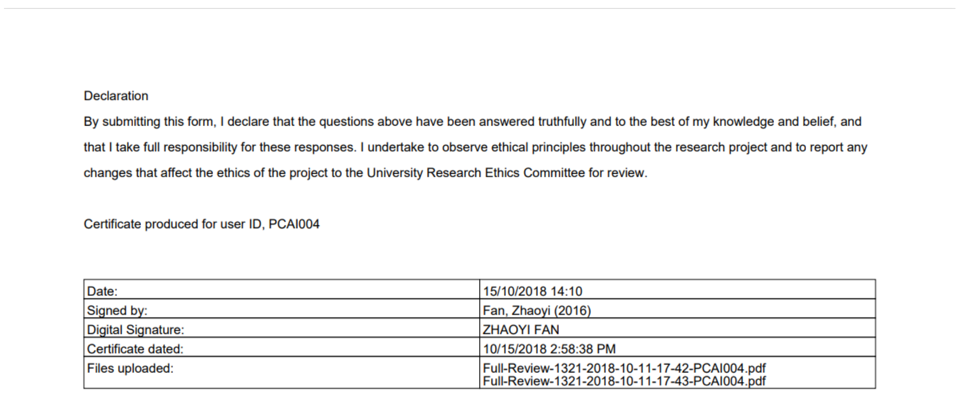
Could any aspects of the research mean that the University has failed in its duty to care for researchers, participants, or the environment / society?,
No

Is there any reputational risk concerning the source of your funding?,
No

Is there any other ethical issue that may arise during the conduct of this study that could bring the institution into disrepute?,
No

Details,

Figure A.3: Page 3 of ethics review approval



Declaration

By submitting this form, I declare that the questions above have been answered truthfully and to the best of my knowledge and belief, and that I take full responsibility for these responses. I undertake to observe ethical principles throughout the research project and to report any changes that affect the ethics of the project to the University Research Ethics Committee for review.

Certificate produced for user ID, PCAI004

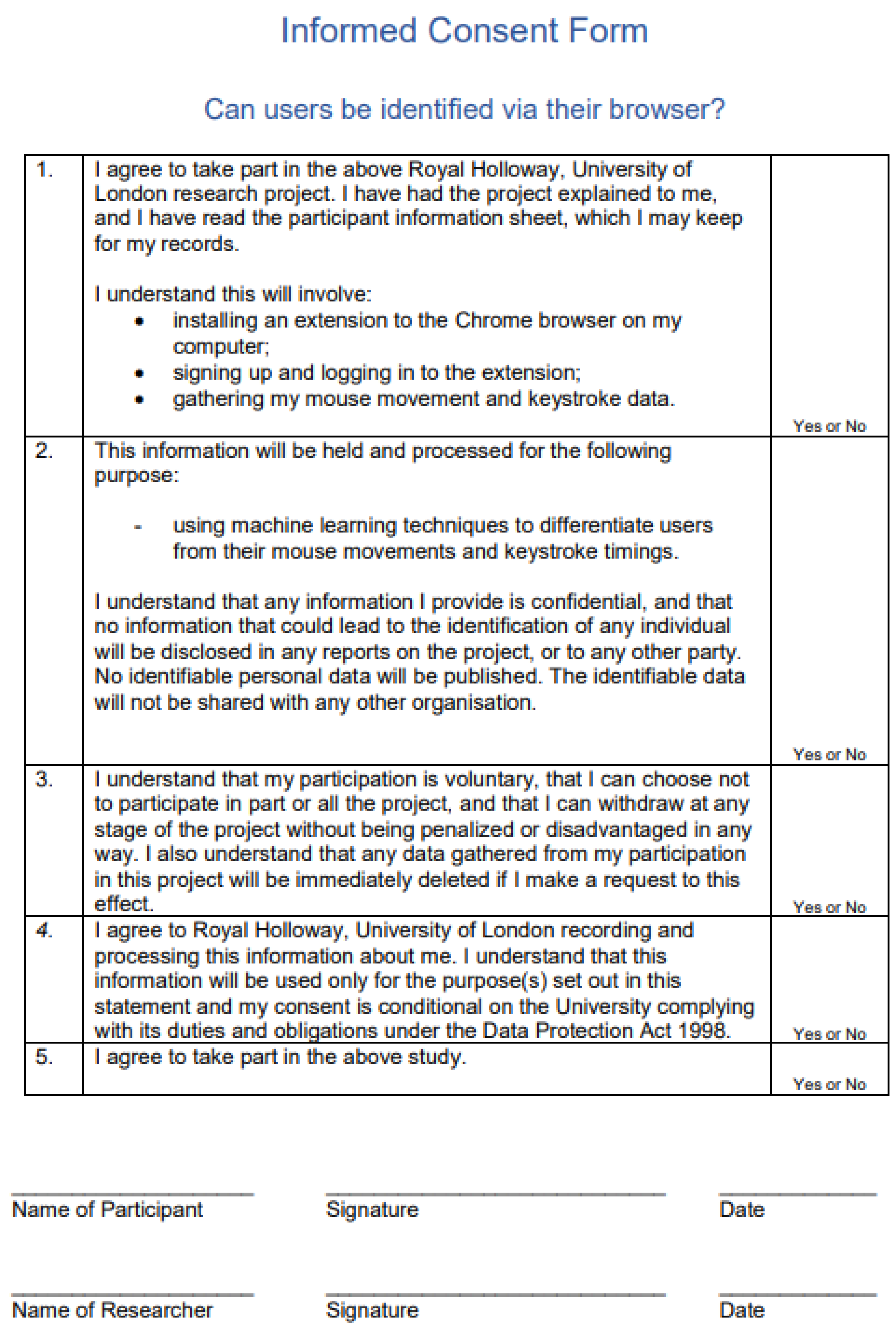
Date:	15/10/2018 14:10
Signed by:	Fan, Zhaoyi (2016)
Digital Signature:	ZHAOYI FAN
Certificate dated:	10/15/2018 2:58:38 PM
Files uploaded:	Full-Review-1321-2018-10-11-17-42-PCAI004.pdf Full-Review-1321-2018-10-11-17-43-PCAI004.pdf

Figure A.4: Page 4 of ethics review approval

Appendix B

Consent form

Figure B.1 illustrates the consent form used in the experiments described in Chapters 6, 7 and 8. All users have signed the form before participating the experiments.



Informed Consent Form

Can users be identified via their browser?

1.	I agree to take part in the above Royal Holloway, University of London research project. I have had the project explained to me, and I have read the participant information sheet, which I may keep for my records. I understand this will involve: • installing an extension to the Chrome browser on my computer; • signing up and logging in to the extension; • gathering my mouse movement and keystroke data.	Yes or No
2.	This information will be held and processed for the following purpose: - using machine learning techniques to differentiate users from their mouse movements and keystroke timings. I understand that any information I provide is confidential, and that no information that could lead to the identification of any individual will be disclosed in any reports on the project, or to any other party. No identifiable personal data will be published. The identifiable data will not be shared with any other organisation.	Yes or No
3.	I understand that my participation is voluntary, that I can choose not to participate in part or all the project, and that I can withdraw at any stage of the project without being penalized or disadvantaged in any way. I also understand that any data gathered from my participation in this project will be immediately deleted if I make a request to this effect.	Yes or No
4.	I agree to Royal Holloway, University of London recording and processing this information about me. I understand that this information will be used only for the purpose(s) set out in this statement and my consent is conditional on the University complying with its duties and obligations under the Data Protection Act 1998.	Yes or No
5.	I agree to take part in the above study.	Yes or No

Name of Participant	Signature	Date
Name of Researcher	Signature	Date

Figure B.1: Consent form used for experiments described in Chapters 6, 7 and 8

Appendix C

N-graph tables

Appendix C shows the common used n-graphs described in Chapter 6.

C.1 Digraph table

Table C.1 illustrates the common used digraphs described in Chapter 6.

Table C.1: Frequently used digraphs

Keys	**Total**	**U0**	**U1**	**U2**	**U3**	**U4**	**U5**	**U6**	**U7**	**U8**	**U9**
Back Back	10781	334	628	407	361	625	542	637	463	438	412
A N	10482	271	80	617	164	822	753	849	253	201	154
N G	9269	225	71	572	125	750	631	766	167	265	167
E Space	5969	118	74	305	20	455	524	503	122	45	38
I Space	5591	135	19	379	20	461	429	450	110	61	47
S H	5457	78	21	344	30	482	416	407	106	126	114
I A	5390	52	25	333	30	436	399	478	125	103	55
E N	5146	87	92	303	59	390	345	413	81	136	90
I N	4303	82	247	218	81	282	255	309	111	143	113
N Space	4276	99	59	246	66	338	290	340	121	60	64
J I	4160	50	33	257	44	339	279	358	69	82	61
A O	4145	79	20	268	19	388	328	352	75	80	46
H I	3534	49	22	219	29	284	275	320	59	42	49
H E	3443	63	15	165	31	275	240	259	84	118	104
A I	3355	62	18	257	27	253	283	271	41	40	36
Z H	3328	34	7	188	41	252	258	301	61	58	42
H U	3230	76	8	184	48	236	258	215	89	59	43
Space Enter	3204	73	31	194	46	201	189	214	148	89	91
H A	3056	101	42	187	41	241	222	229	83	86	50

X I	3017	55	15	199	11	234	219	260	42	70	38
O U	2885	68	26	196	16	238	216	227	61	43	34
O Space	2881	77	13	160	14	234	233	224	54	34	20
G Space	2872	99	26	149	24	239	207	209	77	46	43
A Space	2833	129	8	197	10	206	218	215	51	14	13
D E	2828	37	54	155	8	244	245	211	33	31	26
U Space	2723	55	11	186	18	206	215	205	76	32	30
O N	2586	100	36	138	49	207	171	189	88	92	80
Y I	2291	25	6	165	55	150	162	204	43	49	31
U A	2150	99	8	120	46	172	152	154	52	58	42
U O	2110	29	7	131	14	179	175	152	28	17	11
E I	2017	38	54	122	26	161	147	140	22	27	29
L E	1856	31	58	97	10	128	160	169	27	14	26
D A	1794	28	18	114	16	136	138	145	36	43	30
W O	1759	11	6	123	1	154	164	136	11	5	1
Space D	1757	52	17	75	7	163	129	121	55	7	11
M E	1703	20	24	86	8	110	124	136	31	22	7
C H	1682	30	47	82	48	90	92	137	68	48	32
L I	1622	35	42	56	29	115	103	120	63	85	53
H O	1542	15	12	92	11	109	102	135	51	41	26
G E	1516	24	53	87	8	125	130	144	21	28	9
Space Space	1512	16	6	107	18	137	133	140	30	14	5
Space Y	1456	41	2	92	5	117	118	129	27	13	8
Space Back	1408	99	33	82	21	87	101	90	46	8	19
U I	1394	20	3	78	12	150	119	107	30	25	25
Space Z	1388	14	1	93	2	114	122	100	24	5	6
B U	1364	24	11	74	9	111	128	98	16	13	4
Q I	1363	15	1	77	17	101	109	104	7	39	23
T A	1326	20	18	102	9	135	106	101	19	14	16
Y A	1286	44	14	70	6	122	77	111	19	12	3
M A	1270	37	39	58	13	54	70	71	39	62	39
Space S	1265	33	15	73	18	102	97	88	45	7	16
N I	1264	20	16	72	6	78	98	90	17	30	25
I E	1255	13	30	89	10	80	82	114	23	22	21
I U	1241	10	5	61	7	87	87	106	8	40	22
I S	1218	26	45	77	18	99	83	75	38	12	28
B A	1167	27	11	72	22	89	89	111	20	37	23
T I	1113	8	40	65	18	92	86	74	34	23	12
R E	1110	13	32	53	27	84	47	92	32	29	25
D I	1062	8	21	64	13	90	74	86	39	21	20
G U	1055	19	4	60	23	88	68	102	17	19	17
I J	1046	9	1	75	21	81	60	88	16	21	24
Space H	1008	12	8	61	4	72	84	81	18	6	5

E R	1002	11	82	36	23	68	35	57	51	30	34
N A	986	41	25	64	17	51	85	62	14	17	7
Space B	973	22	13	57	10	69	68	81	26	10	11
Space J	962	11	2	65	5	73	70	97	18	5	4
Space W	956	13	9	63	3	74	83	72	16	4	9
N D	955	20	28	42	15	68	64	82	29	15	28
G A	946	17	5	57	12	80	92	83	17	19	12
I D	929	16	6	55	7	66	87	82	19	22	21
I Y	918	16	5	48	6	61	75	73	17	19	10
U E	917	26	25	41	6	62	82	71	7	8	5
U N	894	32	41	45	15	45	56	67	35	36	33
F A	876	10	16	52	6	75	57	70	25	21	19
Y U	833	30	7	42	8	64	77	72	18	34	20
U S	831	16	15	51	12	60	51	52	31	17	13
D U	829	17	4	48	10	63	64	75	30	15	18
N S	821	15	8	45	10	53	49	59	21	36	31
W E	814	16	25	39	10	75	56	60	27	15	6
L A	811	17	23	33	14	65	63	67	13	28	26
N T	805	2	181	17	14	57	45	44	25	22	18
I L	804	11	18	42	5	71	57	58	21	13	12
Q U	771	11	4	44	9	56	62	49	22	14	4
Space G	751	30	7	47	5	71	61	67	7	4	1
B I	741	7	8	48	5	50	48	80	35	7	8
N E	737	15	30	40	14	46	47	60	27	15	22
D O	732	52	4	22	7	66	45	47	16	26	16
K A	722	19	5	53	2	49	68	49	9	10	5
Space M	709	23	19	35	3	49	54	44	11	9	6
Space N	709	11	12	47	1	37	60	52	11	6	9
B E	701	22	21	24	21	57	38	31	15	22	36
W A	686	17	7	45	10	44	54	56	14	19	12
Space L	682	27	2	28	6	60	59	51	19	11	12
I Back	657	4	16	36	15	41	45	63	20	6	7
E S	642	6	23	36	15	42	26	47	26	22	22
Space C	639	19	8	27	8	37	38	53	15	8	12
G S	628	16	5	49	9	57	42	49	11	10	7
I H	617	17	6	45	4	49	48	54	5	17	5
R A	617	4	21	39	35	43	30	46	30	10	15
E D	615	10	4	27	6	38	47	37	20	11	21
O Y	613	3	4	35	3	59	50	47	20	16	11
N J	608	2	4	37	5	40	41	60	12	9	8
Z I	598	7	2	46	9	44	44	51	18	11	6
I G	597	15	7	37	11	54	46	50	13	5	9
I Z	593	3	1	38	4	50	42	67	5	10	11

K E	591	16	20	30	6	31	46	49	27	6	11
A L	584	20	59	21	14	22	24	39	19	15	24
O L	581	17	10	32	3	40	40	57	12	18	11
Y E	561	9	3	25	9	46	50	54	6	18	1
U L	556	8	21	24	14	49	51	35	14	3	3
G O	555	4	5	37	7	59	52	53	4	9	13
I T	548	12	17	36	20	47	38	29	20	18	14
S U	548	4	15	25	14	52	34	36	28	12	6
S I	545	14	13	32	16	56	34	36	13	6	10
T O	532	12	6	49	20	34	36	34	27	17	21
O D	531	4	4	43	3	47	41	34	13	11	10
I B	524	3	4	31	6	43	56	40	10	3	6
M I	523	12	23	28	9	32	23	41	16	14	23
N Z	522	1	5	33	5	26	39	49	16	4	5
I C	518	9	27	21	13	39	36	37	25	16	15
E Y	516	11	3	31	4	26	32	47	12	14	8
O M	511	17	17	25	12	24	11	40	31	20	13
N L	510	3	5	35	9	27	45	41	15	10	12
O S	508	7	11	32	2	48	39	28	23	11	1
C A	505	7	2	17	13	33	31	26	11	13	16
Back S	500	12	11	15	1	30	39	37	29	11	16
T E	496	15	59	22	14	16	15	21	41	23	17
N Y	487	30	6	20	4	47	34	47	8	11	9
G D	481	2	2	39	1	58	29	51	5	14	9
U D	468	4	23	24	4	36	32	37	10	13	9
N Back	461	3	13	35	18	20	31	30	14	2	8
E M	449	7	16	23	3	19	15	26	7	32	16
Back Z	443	7	4	16	8	39	32	38	11	3	8
U Y	437	13	2	32	6	26	36	30	6	5	4
F E	435	13	3	21	4	39	21	33	10	13	3
A S	432	7	19	18	14	23	22	29	39	19	28
G Y	432	6	1	34	10	32	25	41	9	13	13
Back D	431	12	4	30	4	28	36	28	23	6	8
A Back	419	5	7	30	11	41	31	26	12	5	8
T H	416	16	4	5	7	15	10	26	54	11	20
G Z	415	3	2	23	5	34	28	35	3	8	6
P I	406	3	4	23	7	28	26	19	26	12	10
Space F	405	16	4	19	8	26	22	35	14	4	4
H Space	402	3	14	26	4	18	37	30	15	5	4
N H	398	5	2	35	5	34	30	28	12	11	3
U Back	397	1	5	17	13	24	24	36	14	5	6
A G	395	2	43	15	6	20	29	38	9	3	3
A R	394	10	20	12	8	24	7	32	28	20	26

G L	391	3	5	8	5	19	20	45	8	20	10
C O	380	16	26	7	17	14	4	20	24	19	17
R I	380	10	131	3	18	12	4	3	42	18	13
Back H	371	23	4	20	10	23	34	22	15	5	6
Back Y	371	10	6	23	6	30	24	24	10	5	9
O Back	367	6	10	17	7	32	33	23	14	3	7
R Space	364	12	33	10	8	15	23	28	19	8	12
G G	354	1	2	27	3	23	39	25	6	6	3
S T	350	14	84	2	18	9	2	8	31	16	19
T U	348	3	28	14	7	27	18	11	14	8	14
U G	337	2	1	21	6	31	31	37	7	3	3
P A	334	19	4	11	8	15	11	26	22	8	12
S Space	334	5	22	4	14	16	2	19	37	5	17
E Back	333	4	16	16	11	19	25	28	13	1	11
G B	333	7	1	17	2	17	18	28	9	10	3
U R	331	8	12	11	13	11	5	13	21	16	8
D Space	326	11	18	9	9	17	8	18	24	5	11
L U	325	6	4	10	12	26	12	21	10	16	10
O Z	320	7	1	31	1	20	29	24	2	5	3
A T	317	2	20	12	18	17	7	12	39	9	14
E L	315	1	28	11	6	22	22	22	15	11	16
T Space	312	6	37	4	8	22	12	19	34	4	12
Back A	301	8	9	20	7	23	16	20	20	4	3
O R	299	8	29	5	18	8	9	10	27	9	9
U B	297	10	4	12	5	22	21	18	11	5	4
R O	295	22	14	2	15	8	12	19	32	16	13
U T	295	8	2	17	2	26	27	16	15	7	7
F U	294	8	5	13	12	17	17	9	13	21	12
N B	292	34	3	14	2	26	13	19	4	3	1
O T	289	4	8	13	1	12	14	17	10	11	12
B O	285	43	13	2	2	13	7	8	20	9	6
Back T	285	5	10	19	5	22	13	14	22	4	8
A B	284	2	11	7	4	22	19	28	9	11	3
A D	283	8	3	15	4	9	12	17	19	16	7
Back N	281	4	6	15	4	14	17	27	10	3	2
D Back	276	2	5	6	1	4	1	3	113	1	3
L O	275	24	26	4	4	12	6	6	19	15	20
Back Enter	273	3	28	14	7	4	8	19	23	7	11
Back G	273	13	3	18	1	17	19	21	6	6	3
G M	272	1	3	14	3	18	14	16	3	8	10
I O	271	13	11	16	15	20	10	7	13	2	8
Back X	263	3	2	23	2	16	12	20	7	1	6
S A	256	3	2	24	3	14	13	17	10	21	8

Back W	255	4	2	13	1	18	24	16	9	1	2
H Back	254	5	5	20	5	20	17	22	8	4	4
A Y	252	6	1	26	4	22	16	18	19	4	5
Back B	250	13	7	12	4	15	21	19	9	5	5
E T	247	5	25	10	18	16	10	18	10	3	15
O O	247	3	8	13	1	2	7	8	12	26	3
Back L	246	7	15	10	1	17	10	17	12	7	9
I K	245	2	5	15	1	18	19	9	3	3	3
Back M	243	8	4	9	2	18	15	18	9	4	8
Back E	242	3	16	13	9	11	15	10	17	9	16
P E	242	5	22	24	10	15	9	9	17	4	6
N C	240	5	1	14	12	13	5	22	14	13	4
Back I	236	7	9	7	4	17	11	17	17	8	4
R U	231	1	12	12	6	22	11	7	7	11	17
G Back	228	8	1	12	6	16	19	23	2	4	4
U C	226	3	2	14	8	10	7	20	10	4	6
E A	219	15	3	6	5	14	7	8	19	9	10
M O	219	3	6	5	2	17	8	13	25	19	9
U M	219	2	5	13	1	13	8	26	3	3	2
I R	218	2	7	10	27	9	13	21	2	12	5
A X	212	21	6	12	2	9	7	12	6	7	4
Back Space	211	3	26	12	7	13	14	13	20	4	4
Back C	208	6	4	10	10	13	9	15	14	3	4
O H	207	3	3	9	2	21	25	18	6	4	4
V E	206	9	12	9	4	7	5	14	17	5	21
L L	203	7	34	4	2	8	8	7	15	8	4
Digit2 Enter	202	5	3	16	5	22	8	10	8	4	6
S O	201	7	8	11	3	15	15	9	14	14	11
Y Space	196	3	1	2	15	9	6	7	27	7	15
O F	194	5	4	9	11	20	10	2	7	1	4
M U	193	3	13	6	7	16	9	12	11	7	3
N N	191	6	13	10	4	12	17	9	13	5	4
S Back	189	2	5	11	2	9	5	11	18	1	5
E X	182	2	8	9	4	7	12	13	17	4	3
G R	182	6	2	9	1	18	14	22	2	4	5
I P	181	1	1	12	7	7	11	9	15	9	8
E C	179	1	5	7	7	8	9	8	14	4	12
U K	178	1	3	9	2	7	14	15	8	5	4
C I	170	1	3	17	10	11	6	12	7	2	7
E W	168	5	4	7	6	16	10	10	4	4	1
O P	159	8	7	4	2	5	3	6	10	3	2
C E	155	1	1	3	7	7	4	9	17	6	8
Back F	153	3	3	8	2	12	9	14	5	2	4

Back R	149	2	14	9	6	8	8	2	10	3	9
L Space	144	2	6	3	8	13	3	9	23	4	3
A H	133	1	19	5	5	9	11	6	6	2	5
Space E	127	1	5	7	2	10	10	9	9	1	1
P O	109	4	7	5	5	5	5	7	7	4	7
A K	98	3	2	3	2	5	3	3	7	3	8
R Back	84	3	4	3	4	5	2	8	8	1	3
Space U	73	2	7	4	1	1	1	5	11	3	2
C Back	65	1	2	4	3	6	2	6	3	3	1
Total	249702	5374	4786	14064	3398	18578	17426	19054	6909	5332	4522

Table C.2 illustrates the common used digraphs described in Chapter 6.

Table C.2: Frequently used digraphs

Keys	**Total**	**U10**	**U11**	**U12**	**U13**	**U14**	**U15**	**U16**	**U17**	**U18**	**U19**
Back Back	10781	750	380	658	733	881	291	306	675	738	522
A N	10482	919	145	1025	881	1018	158	126	938	972	136
N G	9269	773	157	763	715	899	195	118	801	945	164
E Space	5969	596	65	622	612	640	27	88	510	589	16
I Space	5591	498	76	617	476	610	50	64	531	528	30
S H	5457	510	50	471	499	550	87	49	479	468	170
I A	5390	481	55	596	509	507	71	68	463	548	56
E N	5146	414	102	467	410	510	87	86	422	496	156
I N	4303	382	78	295	300	387	116	83	325	388	108
N Space	4276	390	52	470	305	410	25	68	378	446	49
J I	4160	369	36	426	353	414	87	46	345	459	53
A O	4145	345	44	403	323	432	74	45	340	425	59
H I	3534	344	53	377	314	362	24	41	343	306	22
H E	3443	305	29	282	287	321	88	67	257	301	152
A I	3355	302	50	335	294	376	53	29	292	293	43
Z H	3328	295	64	323	272	374	63	61	267	327	40
H U	3230	282	41	293	273	372	47	57	284	308	57
Space Enter	3204	252	77	257	173	377	94	77	268	266	87
H A	3056	268	34	245	226	316	76	40	258	244	67
X I	3017	269	28	278	298	353	39	36	243	290	40
O U	2885	253	91	291	241	300	45	26	225	250	38
O Space	2881	248	40	290	240	328	56	32	261	295	28
G Space	2872	251	38	259	243	255	52	42	268	304	41
A Space	2833	263	20	308	246	352	16	26	265	260	16
D E	2828	260	35	288	286	312	19	32	252	281	19
U Space	2723	221	85	293	227	271	39	21	241	261	30

O N	2586	216	27	205	151	212	56	78	192	244	55
Y I	2291	153	27	239	206	253	32	24	216	224	27
U A	2150	161	40	192	137	221	20	32	194	214	36
U O	2110	189	24	203	203	277	22	29	209	194	17
E I	2017	167	23	196	195	203	45	34	171	175	42
L E	1856	167	13	189	175	195	20	44	150	173	10
D A	1794	135	17	177	169	188	47	23	129	168	37
W O	1759	172	16	199	187	229	12	3	175	140	14
Space D	1757	192	31	213	166	145	11	18	162	173	9
M E	1703	177	62	164	155	189	23	42	152	150	21
C H	1682	144	29	142	110	171	55	32	165	131	29
L I	1622	115	37	134	107	130	29	38	118	139	74
H O	1542	120	37	151	116	168	32	36	126	142	20
G E	1516	128	48	135	126	151	8	17	103	159	12
Space Space	1512	181	4	118	124	139	8	21	160	147	4
Space Y	1456	104	11	178	145	140	9	7	144	159	7
Space Back	1408	142	57	112	96	147	10	18	126	108	6
U I	1394	111	11	151	104	157	22	13	128	115	13
Space Z	1388	142	19	196	110	164	5	7	120	141	3
B U	1364	95	5	160	138	173	9	8	153	130	5
Q I	1363	127	14	126	160	135	7	19	120	148	14
T A	1326	97	21	114	140	135	11	8	136	112	12
Y A	1286	109	7	150	125	158	17	12	106	109	15
M A	1270	107	41	118	89	121	34	37	76	88	77
Space S	1265	108	23	106	130	132	10	28	108	116	10
N I	1264	85	11	133	119	169	52	20	113	99	11
I E	1255	119	18	113	120	140	21	11	79	137	13
I U	1241	82	15	126	121	140	24	10	111	128	51
I S	1218	125	12	95	102	129	19	25	109	83	18
B A	1167	113	9	93	82	125	22	13	84	104	21
T I	1113	131	12	95	94	70	11	31	85	105	27
R E	1110	131	14	100	89	109	11	40	90	82	10
D I	1062	92	17	102	95	86	19	23	89	78	25
G U	1055	87	17	101	73	107	21	12	109	89	22
I J	1046	81	9	92	96	108	35	9	68	124	28
Space H	1008	100	8	99	78	122	8	13	115	101	13
E R	1002	79	69	76	57	73	27	44	60	57	33
N A	986	81	8	91	91	106	25	16	62	113	10
Space B	973	74	8	114	81	95	6	10	89	122	7
Space J	962	102	5	110	80	103	6	5	99	101	1
Space W	956	85	15	102	106	99	5	10	96	86	6
N D	955	88	16	89	82	82	11	19	79	70	28
G A	946	63	10	103	89	92	13	2	76	89	15

I D	929	78	5	97	79	98	8	12	74	88	9
I Y	918	81	14	95	97	119	12	8	75	82	5
U E	917	72	6	120	86	121	1	17	82	75	4
U N	894	62	14	66	62	67	7	20	69	80	42
F A	876	84	11	73	49	80	24	14	90	82	18
Y U	833	54	11	92	50	83	7	12	71	64	17
U S	831	91	7	75	69	84	9	11	66	77	24
D U	829	64	11	78	64	85	1	23	74	74	11
N S	821	76	10	59	56	69	33	11	59	48	73
W E	814	65	13	83	78	67	9	10	58	88	14
L A	811	64	19	69	49	72	24	15	65	57	28
N T	805	79	8	59	46	47	22	20	44	41	14
I L	804	65	25	60	68	71	15	15	68	75	34
Q U	771	58	6	76	82	81	13	6	88	82	4
Space G	751	55	5	68	65	83	9	6	74	79	7
B I	741	66	2	77	55	75	4	18	49	92	7
N E	737	50	16	53	61	78	8	17	73	44	21
D O	732	71	61	59	47	63	5	11	43	57	14
K A	722	65	6	69	72	81	12	2	65	78	3
Space M	709	55	12	86	67	78	14	12	63	63	6
Space N	709	57	9	73	83	77	11	13	68	68	4
B E	701	55	38	53	41	49	33	18	28	65	34
W A	686	50	10	66	63	52	11	11	68	65	12
Space L	682	50	12	66	63	66	5	11	51	68	15
I Back	657	66	6	56	51	68	11	9	65	61	11
E S	642	79	12	54	36	55	12	38	41	41	9
Space C	639	80	17	61	29	60	9	23	58	65	12
G S	628	55	6	48	41	60	13	4	55	79	12
I H	617	49	1	81	42	72	3	7	47	61	4
R A	617	65	14	45	52	50	12	18	46	31	11
E D	615	58	12	50	69	74	8	27	49	39	8
O Y	613	46	6	70	42	65	14	12	55	38	17
N J	608	48	4	64	50	65	21	4	54	78	2
Z I	598	54	9	60	44	68	5	10	50	54	6
I G	597	59	6	72	40	56	3	8	50	44	12
I Z	593	46	12	56	58	66	8	12	46	53	5
K E	591	31	6	62	66	51	20	8	60	38	7
A L	584	51	11	50	37	49	30	20	26	37	16
O L	581	56	10	65	42	45	5	8	52	49	9
Y E	561	59	3	59	40	56	3	4	55	54	7
U L	556	41	18	47	52	62	5	16	41	43	9
G O	555	45	4	51	27	51	8	5	42	64	15
I T	548	38	13	38	36	46	14	27	43	36	6

S U	548	45	8	53	45	62	6	14	54	29	6
S I	545	39	9	42	35	65	6	20	40	52	7
T O	532	45	8	50	23	26	11	19	39	44	11
O D	531	45	14	41	48	61	13	4	36	41	18
I B	524	42	7	64	42	50	5	4	51	55	2
M I	523	34	11	41	44	61	15	12	25	47	12
N Z	522	49	6	60	51	64	9	8	47	44	1
I C	518	37	8	33	24	72	16	18	28	36	8
E Y	516	35	9	70	48	72	3	1	43	43	4
O M	511	41	37	30	22	45	17	17	28	30	34
N L	510	50	4	53	20	59	4	19	48	42	9
O S	508	40	11	41	41	60	10	11	43	42	7
C A	505	55	16	46	38	55	14	19	26	54	13
Back S	500	52	18	39	33	49	14	9	39	35	11
T E	496	27	24	24	30	41	9	35	23	22	18
N Y	487	45	3	44	36	45	4	3	40	40	11
G D	481	28	6	47	27	61	9	6	34	45	8
U D	468	35	4	42	45	58	13	3	37	34	5
N Back	461	38	4	42	28	46	4	6	50	55	14
E M	449	45	6	36	32	41	24	10	26	28	37
Back Z	443	44	18	38	32	36	5	3	43	47	11
U Y	437	31	10	45	41	58	2	12	29	46	3
F E	435	31	6	68	33	49	10	22	22	27	7
A S	432	35	8	28	31	22	3	18	22	31	16
G Y	432	30	1	39	23	60	7	4	46	31	7
Back D	431	30	9	37	37	39	5	15	39	37	4
A Back	419	36	3	41	40	47	7	3	24	39	3
T H	416	46	23	26	25	20	4	45	35	18	6
G Z	415	41	6	26	36	57	14	2	36	47	3
P I	406	44	8	31	22	31	7	5	38	55	7
Space F	405	41	8	31	37	41	6	18	29	36	6
H Space	402	55	3	35	32	30	3	6	48	32	2
N H	398	34	4	41	39	36	15	3	34	23	4
U Back	397	43	5	40	27	57	1	5	34	32	8
A G	395	45	3	28	35	34	2	7	28	31	14
A R	394	30	49	16	10	15	16	24	17	8	22
G L	391	36	3	27	29	45	4	4	39	48	13
C O	380	28	36	19	7	21	17	27	14	23	24
R I	380	22	4	6	12	6	10	22	15	12	17
Back H	371	30	5	32	22	43	3	8	27	32	7
Back Y	371	28	9	34	28	43	4	8	30	34	6
O Back	367	47	3	31	24	32	2	6	31	33	6
R Space	364	33	10	31	26	27	1	13	34	19	2

G G	354	26	6	32	25	31	6	3	47	36	7
S T	350	34	5	9	9	7	6	35	8	12	22
T U	348	36	12	34	32	20	8	8	21	27	6
U G	337	21	4	43	21	33	4	1	28	33	7
P A	334	25	10	34	29	23	4	19	26	11	17
S Space	334	37	6	20	17	31	6	28	21	23	4
E Back	333	37	4	24	24	39	1	7	28	23	2
G B	333	33	19	21	28	26	32	5	21	32	4
U R	331	37	11	31	17	25	4	26	24	22	16
D Space	326	41	19	16	25	16	7	24	18	22	8
L U	325	25	6	32	19	35	6	6	10	38	21
O Z	320	24	3	26	24	41	7	4	30	35	3
A T	317	24	19	8	21	16	5	36	11	17	10
E L	315	21	19	21	14	25	4	11	20	15	11
T Space	312	28	8	20	20	9	3	34	17	13	2
Back A	301	30	4	27	27	30	2	3	19	27	2
O R	299	37	28	11	13	9	9	20	21	15	4
U B	297	24	15	28	22	42	1	9	21	21	2
R O	295	16	8	11	11	20	9	26	9	14	18
U T	295	17	9	30	29	20	4	10	29	17	3
F U	294	23	4	25	27	31	7	9	9	25	7
N B	292	23	11	18	28	40	2	4	19	24	4
O T	289	24	31	14	23	17	6	13	19	32	8
B O	285	14	36	13	9	13	29	4	8	26	10
Back T	285	34	9	12	23	25	1	10	23	20	6
A B	284	26	8	27	28	34	2	8	12	20	3
A D	283	18	10	17	29	24	3	30	12	18	12
Back N	281	14	3	26	25	37	10	5	22	30	7
D Back	276	10	3	9	6	7	2	82	10	7	1
L O	275	13	8	14	9	11	17	12	21	9	25
Back Enter	273	12	18	18	6	25	9	23	10	16	12
Back G	273	27	4	24	16	41	6	6	21	17	4
G M	272	32	4	27	20	31	4	5	17	26	16
I O	271	38	8	18	26	12	1	16	14	11	12
Back X	263	22	4	25	22	31	1	5	28	29	4
S A	256	26	3	23	16	21	12	3	20	12	5
Back W	255	14	4	33	24	29	2	2	21	34	2
H Back	254	26	1	23	16	26	4	1	29	15	3
A Y	252	15	3	17	17	21	7	5	21	18	7
Back B	250	17	9	24	21	23	6	3	13	19	5
E T	247	11	8	14	12	11	7	24	14	7	9
O O	247	23	13	11	17	15	24	6	10	14	31
Back L	246	16	10	20	15	23	4	8	17	15	13

I K	245	24	2	14	22	32	14	2	30	22	5
Back M	243	27	10	12	19	31	3	7	16	18	5
Back E	242	18	12	10	18	14	7	8	22	9	5
P E	242	11	5	16	17	15	4	12	19	16	6
N C	240	22	4	22	14	13	8	9	25	16	4
Back I	236	19	6	22	12	22	2	8	16	21	7
R U	231	14	5	30	12	15	6	1	19	18	5
G Back	228	18	5	17	17	23	4	1	22	24	2
U C	226	17	2	23	18	22	5	2	21	28	4
E A	219	19	40	6	8	6	8	14	10	5	7
M O	219	22	3	5	5	14	21	6	14	12	10
U M	219	22	1	21	18	32	7	5	14	18	5
I R	218	27	3	8	16	15	5	4	22	9	1
A X	212	17	5	16	21	31	1	2	11	19	3
Back Space	211	14	2	9	10	11	4	23	10	5	7
Back C	208	19	4	8	18	22	2	10	21	14	2
O H	207	15	2	20	24	16	5	1	20	6	3
V E	206	19	12	4	4	14	5	19	9	9	8
L L	203	18	9	6	10	11	7	20	8	7	10
Digit2 Enter	202	13	2	18	9	16	12	6	9	19	11
S O	201	23	4	12	3	6	3	6	13	18	6
Y Space	196	14	5	9	14	10	5	26	10	8	3
O F	194	24	5	12	11	18	1	18	21	9	2
M U	193	14	4	19	9	15	1	5	7	21	11
N N	191	11	5	9	17	23	2	8	7	14	2
S Back	189	14	42	7	9	8	4	5	15	15	1
E X	182	10	6	9	8	27	3	11	8	17	4
G R	182	26	4	12	10	10	1	14	11	8	3
I P	181	22	7	8	9	9	5	2	15	17	7
E C	179	15	4	13	7	8	4	15	12	12	14
U K	178	17	5	10	13	18	17	3	14	7	6
C I	170	9	4	8	10	19	2	6	11	23	2
E W	168	24	5	11	15	9	1	8	9	18	1
O P	159	22	9	3	13	8	12	6	9	19	8
C E	155	20	5	6	4	7	3	23	8	11	5
Back F	153	9	5	15	8	14	4	5	12	15	4
Back R	149	7	8	15	13	11	1	4	9	7	3
L Space	144	16	5	8	6	11	1	11	6	4	2
A H	133	6	2	9	10	11	6	2	7	7	4
Space E	127	5	6	11	13	7	1	7	13	8	1
P O	109	6	2	3	10	5	8	7	4	6	2
A K	98	8	5	10	9	7	2	8	5	3	2
R Back	84	2	5	2	8	7	1	7	7	3	1

Space U	73	1	1	3	3	4	8	7	4	1	4
C Back	65	4	1	2	4	5	1	5	5	5	2
Total	249702	21532	4505	22786	20111	24820	4245	4744	20553	22239	4724

C.2 3-graph table

Table C.3 illustrates the common used 3-graphs described in Chapter 6.

Table C.3: Frequently used tri-graphs

Keys	**Total**	**U0**	**U1**	**U2**	**U3**	**U4**	**U5**	**U6**	**U7**	**U8**	**U9**
Back Back Back	5599	165	381	185	228	293	256	321	230	269	257
A N G	3304	62	17	203	38	285	226	256	56	76	54
I A N	3294	27	10	218	22	273	219	313	76	55	36
N G Space	2822	97	25	149	23	237	205	207	70	45	43
A N Space	2439	46	6	139	44	202	188	211	58	27	30
S H I	2402	29	12	158	12	208	207	211	30	21	21
I N G	2398	46	39	137	37	178	149	191	48	74	69
O N G	1923	88	4	120	23	173	154	160	34	61	30
H E N	1802	38	6	87	23	148	127	135	26	86	66
X I A	1609	33	13	101	3	131	116	145	23	26	12
E N G	1526	25	2	107	24	104	97	150	24	39	13
U A N	1508	72	4	65	40	122	110	108	42	43	35
H I Space	1390	24	3	79	4	126	109	130	22	15	11
A O Space	1365	38	1	83	9	113	115	118	30	26	12
J I A	1285	6	3	83	7	92	100	125	31	32	15
Z H E	1189	7	1	68	2	103	111	109	15	13	4
A I Space	1127	29	2	77	1	76	95	89	14	5	6
I A O	1113	18	11	57	3	105	93	77	32	21	6
O U Space	1035	28	1	74	3	74	82	79	14	8	13
H A N	985	66	1	67	25	81	57	71	29	26	11
I J I	965	7	1	71	19	79	57	83	13	20	21
S H E	950	30	4	39	3	77	53	48	9	65	58
S H U	942	11	2	75	4	78	84	55	31	6	6
Space Back Back	939	65	11	54	15	58	71	59	36	1	14
E N Space	938	31	14	69	6	72	56	69	10	21	9
H A O	917	27	3	58	3	74	77	62	20	25	13
J I N	902	13	27	48	20	76	47	68	4	22	25
Space S H	899	19	3	59	2	81	76	65	23	2	1
H U A	854	41	3	49	23	59	68	61	24	23	16
H O U	831	9	1	57	4	66	63	83	14	10	3
J I U	766	2	3	45	1	62	64	68	6	2	8

Z H I	760	4	1	47	8	54	55	81	13	13	19
M E N	700	8	4	38	5	38	36	66	3	9	2
H U I	658	7	1	39	7	85	51	40	5	13	6
U S H	623	11	2	43	1	49	44	44	7	11	6
N G S	619	15	4	49	9	57	42	48	11	10	7
N S H	608	11	2	35	1	41	44	37	7	27	25
H E Space	598	13	3	31	3	40	42	43	28	7	8
G E Space	558	10	19	30	2	41	59	59	12	14	1
R E N	552	3	8	39	6	50	36	51	3	2	1
H O N	551	3	1	34	5	33	32	46	27	23	12
Z H O	543	3	1	38	4	40	38	51	7	13	5
I Space Enter	536	10	7	32	9	46	20	37	25	25	15
G U O	526	15	2	27	4	49	41	43	5	10	6
M E I	520	7	4	30	1	42	51	37	1	3	2
D A N	504	2	7	24	3	53	38	44	3	6	5
E Space Enter	496	11	6	24	4	39	48	43	15	3	2
F A N	478	1	6	23	2	46	35	35	15	16	4
L I A	477	5	2	28	5	49	49	31	13	6	2
N G D	476	2	2	39	1	55	29	50	5	14	9
D A O	469	5	2	31	3	32	33	47	3	11	6
B E I	447	14	5	19	13	44	30	26	2	11	23
A N D	430	3	13	20	3	34	26	47	15	2	9
I Back Back	429	3	11	21	10	26	28	36	13	6	6
I E Space	429	3	6	28	3	20	27	31	6	5	3
T I A	418	1	4	29	2	30	28	37	6	13	4
C H U	417	9	1	16	11	19	27	37	17	5	8
I N Space	415	9	17	14	4	30	24	25	21	4	5
N G Z	410	3	2	23	5	33	28	35	3	8	6
A N S	401	9	4	22	6	40	32	34	12	6	3
C H A	401	2	2	21	6	29	26	33	16	14	6
Space H A	392	6	3	26	2	27	47	34	1	2	2
G Space Enter	385	12	1	19	4	19	17	29	24	20	23
N Space Enter	385	8	2	26	15	18	18	25	24	13	18
Z H U	383	5	3	12	16	21	29	32	10	9	5
N J I	371	2	4	21	2	23	19	38	10	7	8
A N J	369	2	2	22	4	24	32	31	6	4	4
I Z H	368	3	1	23	4	30	24	46	5	5	1
N G L	368	3	3	8	5	18	20	45	6	17	10
Space D A	358	4	8	19	1	26	33	18	9	2	1
W E N	352	1	2	12	1	35	34	32	9	4	1
E N D	350	3	8	17	1	27	34	24	5	3	5
I U Space	346	4	1	19	2	26	28	24	2	7	1
B A N	343	2	5	20	4	24	36	31	7	1	8

A N Y	329	30	4	12	2	29	20	32	5	5	9
D O N	320	42	1	7	4	32	19	24	10	22	15
Back S H	315	3	2	10	1	21	27	28	8	6	6
C H E	315	2	4	21	20	16	22	24	12	10	9
U Space Enter	299	5	2	24	5	10	14	18	20	9	15
O Space Enter	284	2	1	19	4	16	22	15	14	7	4
I D A	279	8	2	17	2	17	22	21	4	10	7
E N S	274	3	1	13	1	10	10	12	5	25	23
N Back Back	267	2	7	18	12	13	19	17	8	1	7
Space H U	267	3	1	14	2	16	20	17	6	3	1
T H E	265	10	1	4	2	10	9	20	25	6	9
U Back Back	265	1	2	7	9	18	15	27	8	4	1
I C H	264	3	23	14	7	19	20	13	6	7	9
I N T	263	1	127	6	4	14	10	10	6	1	5
A O Y	262	2	2	10	2	22	18	19	10	13	9
A I L	257	3	8	12	4	28	20	14	2	7	7
A Back Back	253	3	5	18	8	28	17	19	7	4	4
Back Back S	252	7	2	8	1	17	23	18	12	5	8
I Space Back	249	15	1	14	3	12	24	15	4	2	1
N Z H	246	1	1	11	5	11	20	18	13	2	3
M I N	245	9	15	10	3	18	14	22	3	8	11
M A I	242	10	8	12	3	8	21	13	1	6	7
Back Back Z	237	3	2	10	6	19	19	19	7	2	5
O Back Back	230	3	8	10	3	17	24	19	7	2	3
E Back Back	229	4	12	10	5	16	16	22	8	1	5
T O N	228	1	1	20	6	18	25	12	2	6	15
Back Back D	227	7	2	15	3	12	18	13	9	2	4
Back X I	222	2	1	22	1	14	10	19	4	1	3
Back Back Y	220	9	3	18	4	14	13	14	4	2	4
B A O	206	4	1	13	4	17	13	26	1	15	2
Back Back H	206	15	1	9	7	10	22	10	9	2	3
I O N	198	6	9	12	13	17	6	4	10	1	6
Space D I	174	1	3	6	1	18	7	17	12	1	2
Back Back T	162	1	5	11	2	9	7	11	14	2	3
Space W A	158	3	1	11	2	14	12	13	1	1	4
Back Back B	135	7	5	10	3	10	8	14	5	2	2
G Back Back	135	5	1	6	6	9	9	11	2	2	2
Space B E	135	6	4	4	3	12	12	11	5	3	4
Back Back L	126	6	5	5	1	10	4	6	5	4	5
G Space Back	107	6	1	6	4	6	8	7	3	2	1
Back Back A	105	5	7	5	2	7	5	7	9	2	1
H I N	94	2	1	6	14	4	11	4	2	2	1
Back B A	84	4	1	3	3	8	6	11	2	2	1

O U R	80	4	2	3	2	1	2	3	1	4	3
Total	79947	1725	1102	4515	1077	6115	5665	6299	1842	1793	1470

Table C.4 illustrates the common used 3-graphs described in Chapter 6.

Table C.4: Frequently used tri-graphs

Keys	**Total**	**U0**	**U1**	**U2**	**U3**	**U4**	**U5**	**U6**	**U7**	**U8**	**U9**
Back Back Back	5599	357	208	309	357	438	181	161	315	346	342
A N G	3304	291	54	301	269	354	64	28	307	323	40
I A N	3294	301	30	373	326	304	31	33	284	333	30
N G Space	2822	243	38	254	238	252	52	41	264	298	41
A N Space	2439	215	24	271	196	238	13	27	227	259	18
S H I	2402	235	28	252	240	246	13	19	242	207	11
I N G	2398	214	31	162	185	205	86	48	191	232	76
O N G	1923	152	10	166	121	181	26	19	167	203	31
H E N	1802	125	18	131	137	153	61	32	139	137	127
X I A	1609	150	16	159	186	173	15	12	146	131	18
E N G	1526	112	59	122	137	153	17	19	131	177	14
U A N	1508	109	28	135	89	144	17	26	143	150	26
H I Space	1390	143	24	160	125	129	8	9	150	117	2
A O Space	1365	122	7	133	97	149	28	10	109	150	15
J I A	1285	102	14	156	95	132	22	21	87	153	9
Z H E	1189	119	8	122	123	144	9	18	94	116	3
A I Space	1127	111	13	141	104	142	11	5	96	104	6
I A O	1113	97	15	126	85	101	18	17	106	112	13
O U Space	1035	97	63	115	82	93	23	7	79	90	10
H A N	985	96	20	67	69	88	17	8	88	74	24
I J I	965	76	7	86	86	93	34	9	61	114	28
S H E	950	55	12	43	71	62	47	13	65	72	124
S H U	942	94	4	89	94	123	9	5	77	87	8
Space Back Back	939	107	37	75	63	99	6	8	86	72	2
E N Space	938	79	15	113	65	94	10	12	86	91	16
H A O	917	74	1	80	68	109	29	15	72	90	17
J I N	902	94	8	59	51	82	44	7	74	101	32
Space S H	899	79	12	78	103	106	7	5	86	89	3
H U A	854	64	12	73	62	78	8	21	68	81	20
H O U	831	59	7	90	67	101	15	8	75	88	11
J I U	766	57	1	97	83	88	3	1	88	83	4
Z H I	760	67	17	95	53	67	5	12	69	77	3
M E N	700	85	44	66	51	84	8	3	73	72	5
H U I	658	57	4	72	51	90	5	8	65	45	7

U S H	623	70	2	64	59	60	8	4	56	65	17
N G S	619	55	5	47	40	59	12	4	55	78	12
N S H	608	56	5	40	45	51	22	6	55	33	65
H E Space	598	74	3	55	58	54	6	16	47	63	4
G E Space	558	56	2	49	47	55	2	8	30	60	2
R E N	552	48	6	58	48	74	5	7	50	54	3
H O N	551	51	3	49	44	57	13	22	41	48	7
Z H O	543	42	4	54	40	67	24	5	50	51	6
I Space Enter	536	53	6	43	31	54	13	21	41	36	12
G U O	526	35	7	57	40	67	12	7	55	34	10
M E I	520	54	6	63	59	62	4	2	46	40	6
D A N	504	42	3	57	54	56	4	4	37	57	5
E Space Enter	496	41	9	45	28	71	8	4	47	42	6
F A N	478	39	6	41	26	50	16	7	57	44	9
L I A	477	36	5	50	43	34	3	5	48	56	7
N G D	476	28	6	46	27	61	9	6	34	45	8
D A O	469	40	2	53	45	46	15	1	42	42	10
B E I	447	33	4	30	26	37	26	6	23	49	26
A N D	430	38	6	47	42	34	5	12	33	35	6
I Back Back	429	48	3	34	36	39	11	6	43	44	5
I E Space	429	44	12	54	43	49	2	3	31	57	2
T I A	418	49	4	36	30	36	6	2	43	52	6
C H U	417	31	7	43	31	47	7	7	58	34	2
I N Space	415	44	9	47	22	38	1	6	38	52	5
N G Z	410	39	6	25	35	57	14	2	36	47	3
A N S	401	37	7	39	36	36	8	5	39	22	4
C H A	401	36	9	34	31	37	8	3	49	27	12
Space H A	392	39	3	43	27	45	5	5	37	32	6
G Space Enter	385	24	7	22	17	32	22	4	32	35	22
N Space Enter	385	25	6	26	15	50	7	10	30	38	11
Z H U	383	34	20	28	20	46	3	13	15	49	13
N J I	371	27	3	37	25	35	21	2	36	49	2
A N J	369	28	3	35	42	35	14	1	36	42	2
I Z H	368	29	7	30	31	48	3	8	34	33	3
N G L	368	35	3	27	27	44	4	3	38	44	8
Space D A	358	40	6	48	38	24	9	2	33	34	3
W E N	352	33	3	49	29	29	3	1	20	52	2
E N D	350	41	1	34	35	35	2	4	41	29	1
I U Space	346	20	3	42	30	37	2	2	51	43	2
B A N	343	39	2	27	27	50	3	2	27	24	4
A N Y	329	28	2	31	23	28	2	3	28	30	6
D O N	320	21	5	15	11	25	4	6	15	32	10
Back S H	315	36	7	23	25	32	10	3	34	26	7

C H E	315	21	2	26	21	26	17	7	21	26	8
U Space Enter	299	10	40	19	14	30	16	3	20	11	14
O Space Enter	284	22	2	27	12	40	11	3	24	31	8
I D A	279	16	1	33	24	31	1	6	21	31	5
E N S	274	17	1	9	11	23	19	3	15	14	59
N Back Back	267	21	2	25	14	28	3	6	19	34	11
Space H U	267	27	1	25	25	33	1	5	36	28	3
T H E	265	42	4	23	20	17	3	18	23	16	3
U Back Back	265	28	4	27	21	39	1	4	23	21	5
I C H	264	18	2	23	16	35	14	1	21	11	2
I N T	263	15	2	16	7	16	1	4	10	7	1
A O Y	262	16	1	32	17	19	11	7	19	18	15
A I L	257	19	24	22	22	16	3	4	13	16	13
A Back Back	253	19	1	23	22	29	3	1	14	26	2
Back Back S	252	29	10	19	18	21	4	3	22	19	6
I Space Back	249	25	12	26	14	29	4	4	24	19	1
N Z H	246	20	3	23	21	29	8	8	28	20	1
M I N	245	13	3	23	20	31	1	3	10	21	7
M A I	242	17	25	17	14	23	9	3	18	13	14
Back Back Z	237	25	9	23	14	17	4	1	19	26	7
O Back Back	230	35	3	15	11	21	2	1	19	24	3
E Back Back	229	26	3	16	17	19	1	7	24	16	1
T O N	228	11	1	25	12	11	1	8	21	25	7
Back Back D	227	18	6	15	23	22	5	2	24	23	4
Back X I	222	18	3	21	22	23	1	5	23	25	4
Back Back Y	220	20	3	20	16	24	3	5	19	20	5
B A O	206	16	2	13	11	15	7	1	9	32	4
Back Back H	206	20	3	16	14	21	1	3	16	18	6
I O N	198	31	7	14	18	11	1	14	6	8	4
Space D I	174	25	1	24	12	12	2	2	14	13	1
Back Back T	162	21	6	8	13	11	1	5	17	14	1
Space W A	158	9	3	18	18	12	1	4	20	8	3
Back Back B	135	11	8	9	11	9	3	1	5	9	3
G Back Back	135	13	4	10	10	13	4	1	14	12	1
Space B E	135	13	1	9	11	10	3	2	6	13	3
Back Back L	126	8	6	8	8	10	2	2	9	13	9
G Space Back	107	10	1	10	13	7	1	1	8	11	1
Back Back A	105	6	3	13	7	11	1	1	3	9	1
H I N	94	9	5	2	4	8	1	9	4	3	2
Back B A	84	5	1	5	9	11	1	1	4	4	2
O U R	80	7	3	11	2	5	1	7	8	5	6
Total	79947	6908	1339	7291	6326	7896	1497	1113	6774	7446	1754

C.3 4-graph table

Table C.5 illustrates the common used 4-graphs described in Chapter 6.

Table C.5: Frequently used 4-graphs

Keys	**Total**	**U0**	**U1**	**U2**	**U3**	**U4**	**U5**	**U6**	**U7**	**U8**	**U9**	
Back*4	2967	77	253	89	155	137	107	160	120	167	156	168
A N G Space	1007	32	4	65	9	78	70	70	25	16	14	80
X I A N	953	19	2	69	3	85	62	92	12	17	6	100
I A N G	935	13	2	62	4	87	68	67	16	16	7	85
S H E N	868	30	4	34	2	69	53	48	8	60	56	48
I N G Space	752	20	13	35	7	65	40	53	19	10	20	86
H A N G	687	12	1	50	14	57	41	60	17	17	5	83
H E N G	627	6	2	37	13	49	53	72	17	21	8	42
O N G Space	530	28	1	22	5	52	57	31	13	10	6	42
J I A N	529	2	1	33	6	48	26	55	20	9	11	39
J I N G	514	9	1	30	11	46	25	37	1	17	20	50
H O N G	505	3	1	34	5	33	31	46	14	19	9	48
E N G Space	502	14	1	26	2	41	36	49	12	8	3	34
H U A N	451	16	1	15	19	37	37	36	19	12	11	29
N G Space Enter	382	12	1	19	4	19	16	29	23	19	23	24
Z H O N	359	1	1	25	4	25	25	31	7	9	4	38
U A N G	349	8	1	16	10	21	22	28	7	12	12	18
I J I N	343	3	1	15	14	31	18	24	3	11	17	29
L I A N	290	4	2	18	4	19	28	18	12	3	2	28
D O N G	289	42	1	6	4	31	19	24	7	19	6	21
A N Space Enter	233	6	1	10	12	10	13	17	14	9	14	12
C H E N	220	2	1	18	17	11	14	22	8	7	5	16
I N G D	209	1	1	13	1	25	15	20	1	11	8	11
A N J I	197	2	2	12	2	12	17	16	4	4	4	16
C H A N	196	2	1	14	5	19	13	18	6	1	2	13
I Back Back Back	181	2	6	12	9	10	10	19	4	3	5	19
Z H U A	164	5	1	4	8	7	12	22	7	7	3	7
A N G Z	146	1	2	9	2	12	7	7	1	2	1	10
Back Back Back S	121	4	1	4	1	8	15	8	4	3	6	12
N G Space Back	102	5	1	6	4	6	8	7	3	2	1	8
I A N S	101	2	1	8	3	11	5	11	1	2	2	5
M A I L	98	1	8	2	2	6	2	3	1	4	5	3
A Back Back Back	93	1	2	3	5	12	5	8	4	1	1	6
A I Space Enter	80	2	2	2	1	4	3	7	6	3	1	10
Total	15980	387	324	817	367	1183	973	1215	436	531	454	1240

Table C.6 illustrates the common used 4-graphs described in Chapter 6.

Table C.6: Frequently used 4-graphs

Keys	**Total**	**U11**	**U12**	**U13**	**U14**	**U15**	**U16**	**U17**	**U18**	**U19**	
Back*4	2967	168	127	146	165	222	111	85	147	162	213
A N G Space	1007	80	16	111	88	100	13	5	104	97	10
X I A N	953	100	8	94	117	102	4	6	83	69	3
I A N G	935	85	8	107	92	103	11	5	85	94	3
S H E N	868	48	10	39	63	54	42	9	54	65	120
I N G Space	752	86	9	48	66	54	25	21	63	83	15
H A N G	687	83	16	54	50	64	16	2	65	53	10
H E N G	627	42	11	48	52	64	6	17	40	60	9
O N G Space	530	42	5	47	42	49	7	5	46	54	8
J I A N	529	39	7	66	34	51	9	7	36	66	3
J I N G	514	50	6	24	26	42	37	7	43	53	29
H O N G	505	48	3	46	42	57	13	6	40	48	7
E N G Space	502	34	8	45	41	47	6	10	50	62	7
H U A N	451	29	5	31	23	41	7	17	37	46	12
N G Space Enter	382	24	7	22	17	32	22	4	32	35	22
Z H O N	359	38	3	41	33	41	8	2	28	29	4
U A N G	349	18	4	30	29	35	6	10	23	45	12
I J I N	343	29	3	21	18	24	27	4	21	33	26
L I A N	290	28	2	24	26	18	3	3	31	39	6
D O N G	289	21	2	14	11	24	3	2	15	32	6
A N Space Enter	233	12	2	16	11	29	5	7	15	24	6
C H E N	220	16	1	16	13	22	6	5	13	20	3
I N G D	209	11	2	13	12	27	8	2	10	22	6
A N J I	197	16	2	11	17	12	14	1	21	26	2
C H A N	196	13	6	19	21	17	2	1	26	4	6
I Back Back Back	181	19	1	8	17	11	6	4	15	17	3
Z H U A	164	7	3	12	8	16	1	6	8	18	9
A N G Z	146	10	2	9	8	22	14	2	11	21	3
Back Back Back S	121	12	6	7	9	7	3	2	8	9	4
N G Space Back	102	8	1	10	12	6	1	1	8	11	1
I A N S	101	5	2	7	8	7	3	1	13	8	1
M A I L	98	3	23	5	7	3	3	3	1	3	13
A Back Back Back	93	6	1	7	6	9	3	1	6	10	2
A I Space Enter	80	10	1	7	8	8	2	2	5	3	3
Total	15980	1240	313	1205	1192	1420	447	265	1203	1421	587

C.4 5-graph table

Table C.7 illustrates the common used 5-graphs described in Chapter 6.

Table C.7: Frequently used 5-graphs

Keys	**Total**	**U0**	**U1**	**U2**	**U3**	**U4**	**U5**	**U6**	**U7**	**U8**	**U9**
Back*5	1556	40	169	46	106	53	43	76	64	101	84
X I A N G	563	12	1	44	1	47	45	48	10	9	2
Z H O N G	354	1	1	25	4	25	24	31	7	9	4
I J I N G	264	3	1	12	9	27	13	17	1	10	16
H E N G Space	248	3	1	10	1	24	19	23	10	3	2
H U A N G	219	6	1	7	10	12	15	14	5	11	9
C H E N G	166	2	1	12	10	8	10	22	7	6	3
Z H U A N	145	5	1	2	8	7	10	15	7	7	3
Total	3515	72	176	158	149	203	179	246	111	156	123

Table C.8 illustrates the common used 5-graphs described in Chapter 6.

Table C.8: Frequently used 5-graphs

Keys	**Total**	**U10**	**U11**	**U12**	**U13**	**U14**	**U15**	**U16**	**U17**	**U18**	**U19**
Back*5	1556	78	77	65	68	112	66	44	65	73	126
X I A N G	563	52	5	58	62	68	4	5	46	43	1
Z H O N G	354	36	3	40	32	41	8	2	28	29	4
I J I N G	264	19	2	14	10	13	25	4	16	26	26
H E N G Space	248	16	4	18	22	28	2	10	18	27	7
H U A N G	219	11	2	16	16	21	5	9	16	24	9
C H E N G	166	9	1	16	9	20	4	3	8	12	3
Z H U A N	145	6	2	10	8	14	1	5	8	17	9
Total	3515	227	96	237	227	317	115	82	205	251	185

Appendix D

Source code

Appendix D lists important source code used in Chapters 5, 6, 7, 8 9 and 10. Full version of the source code can be accessed at https://github.com/fanzhaoyi/ThesisCode.

D.1 Experimental platform code

The full version of experimental platform can be accessed at https://github.com/fanzhaoyi/DataCollector

D.2 Mobile web page code

This section lists the sources of code used on the mobile web page described in Chapter 9. This web page is for examining the

D.2.1 Mobile test web page

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="UTF-8"/>
    <meta name="viewport"
          content="width=device-width, initial-scale=1.0,
              maximum-scale=1.0, minimum-scale=1.0, user-scalable=no">
    <meta name="apple-mobile-web-app-capable" content="yes">
    <meta name="apple-touch-fullscreen" content="yes">
   <title>Mobile Touch Test</title>
</head>
<body>
   <h2 id='out'>Mobile sensor test</h2>
   <button id="btn">Click to authorize</button>
   <input type="text" id='inputtext'>
   <div type='text' id="kd"></div>
   <div type='text' id="ku"></div>
   <div id="msg">Action type: (coordinates), (acceleration),
       (orientation), (rotationRate)</div>
   <div type='text' id="motion"></div>
   <div type='text' id="orientation"></div>
   <div type='text' id="otest"></div>
   <div type='text' id="rotation"></div>
   <div type='text' id="start">start</div>
   <div type='text' id="end">end</div>
   <div type='text' id="move">move</div>
</body>

<script type="text/javascript">
   document.getElementById("btn").addEventListener("click", function() {
        if (typeof DeviceMotionEvent.requestPermission === 'function') {
            DeviceMotionEvent.requestPermission()
            .then(permissionState => {
                if (permissionState === 'granted') {
                    window.addEventListener('devicemotion',
                        deviceMotionHandler, false);
                    function deviceMotionHandler(){}
                }
            })
            .catch((err) => {
                alert("Not authorized");
            });
        } else {
            alert("not found")
        }
    })
   window.addEventListener('load', load, false);
   var Xg=Yg=Zg=0;
    var Xo=Yo=Zo=0;
    var Xr=Yr=Zr=0;
    function load (){
       document.addEventListener('touchstart', touch, { passive: false
           });
       document.addEventListener('touchmove', touch, { passive: false });
       document.addEventListener('touchend', touch, { passive: false });
       function touch (event){
           var event = event || window.event;
           var g = "("+ Xg+ ", "+ Yg+ ", "+ Zg+ ")";
           var r = "("+ Xr+ ", "+ Yr+ ", "+ Zr+ ")";
           var o = "("+ Xo+ ", "+ Yo+ ", "+ Zo+ ")";
           switch(event.type){
               case "touchstart":
                   outstart = event.changedTouches[0].clientX.toFixed(0)
                       + ", " +
                       event.changedTouches[0].clientY.toFixed(0);
                   document.getElementById('start').innerHTML="Start:("+
                       outstart+ ")"+ g+ o+ r;
                   document.getElementById('move').innerHTML="";
                   break;
               case "touchend":
                   outend = event.changedTouches[0].clientX.toFixed(0) +
                       ", " + event.changedTouches[0].clientY.toFixed(0);
                   document.getElementById('end').innerHTML="End:("+
                       outend+ ")"+ g+ o+ r;
                   break;
               case "touchmove":
                   event.preventDefault();
                   outmove = event.changedTouches[0].clientX.toFixed(0) +
                       ", " + event.changedTouches[0].clientY.toFixed(0);
                   document.getElementById('move').innerHTML=document.getElementById('move').innerHTML+
                       "</br>"+ "Move:("+ outmove+ ")"+ g+ o+ r;
                   break;
           }
       }
   }

   if (window.DeviceOrientationEvent){
       window.addEventListener('deviceorientation',
           deviceOrientationHandler, false);
       function deviceOrientationHandler(e){
           var gamma = e.gamma; var beta = e.beta; var alpha = e.alpha;
           if (e.gamma){
               var out= e.gamma.toFixed(2);
               Xo=alpha.toFixed(2);
               Yo=beta.toFixed(2);
               Zo=gamma.toFixed(2);
           }
       }
   }
   if (window.DeviceMotionEvent){
      window.addEventListener('devicemotion', function(event){
            var gacc = event.accelerationIncludingGravity;
            var gx = gacc.x; var gy = gacc.y; var gz = gacc.z;
            Xg=gx.toFixed(2);
            Yg=gy.toFixed(2);
            Zg=gz.toFixed(2);
            var rotation = event.rotationRate;
            if (rotation){
                var rx = rotation.alpha; var ry = rotation.beta; var rz =
                   rotation.gamma;
                Xr=rx.toFixed(2);
                Yr=ry.toFixed(2);
                Zr=rz.toFixed(2);
            }
        }, false)
    }
    document.onkeydown=keyDown;
    document.onkeyup = keyUp;
    function keyDown(e){
       var g = "("+ Xg+ ", "+ Yg+ ", "+ Zg+ ")";
        var r = "("+ Xr+ ", "+ Yr+ ", "+ Zr+ ")";
        var o = "("+ Xo+ ", "+ Yo+ ", "+ Zo+ ")";
        e=e||event;
        if (e.key){
            var thekey='Down:'+ e.key;
        }else{
            var thekey ='Down:'+ e.code;
        }
        var t = Date.now();
        document.getElementById('kd').innerHTML=thekey+ "("+
            t.toString().substr(-5, 5)+ ")"+ g+ r+ o;
    }

    function keyUp(e){
       var g = "("+ Xg+ ", "+ Yg+ ", "+ Zg+ ")";
        var r = "("+ Xr+ ", "+ Yr+ ", "+ Zr+ ")";
        var o = "("+ Xo+ ", "+ Yo+ ", "+ Zo+ ")";
        e=e||event;
        var t = Date.now();
        if (e.key){
            var thekey='Up:'+ e.key;
        }else{
            var thekey ='Up:'+ e.code;
        }
        document.getElementById('ku').innerHTML=thekey+ "("+
            t.toString().substr(-5, 5)+ ")"+ g+ r+ o;
    }
</script>
</html>
```

D.3 Python scripts for data analysis

This section lists the Python scripts for data analysis in Chapters 6, 7, 8 and 9.

D.3.1 Python scripts for analysing keystroke dynamics

List D.3.1 lists the Python scripts for keystroke dynamics analysis used in Chapter 6.

```
import numpy as np
import sys
import csv
import pandas as pd
import math
from numpy import *
from time import time
import time
import datetime
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn import svm
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.multiclass import OneVsOneClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
import random
import sklearn
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import KFold,StratifiedKFold
from sklearn.metrics import make_scorer
from sklearn import linear_model
from sklearn.metrics import r2_score
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import f1_score,precision_score,recall_score
from sklearn.metrics import accuracy_score
from sklearn import tree
from xgboost import XGBClassifier,plot_importance,DMatrix,cv
from xgboost import XGBRegressor
from sklearn.ensemble import VotingClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
from sklearn.metrics import classification_report
from collections import Counter
import os
import string
from sklearn.utils import class_weight
from itertools import combinations
from functools import reduce
import gc
import scikitplot as skplt
import lightgbm as lgb
import seaborn as sns

pd.set_option('display.max_rows',500)
pd.set_option('display.max_columns',500)
pd.set_option('display.width',1000)
pd.set_option('max_colwidth',1000)
np.set_printoptions(suppress=True)


figure_path1 = 'C:/Zhaoyi_Fan/Figures/'
figure_path2 = 'C:/Zhaoyi_Fan/Latex/Figures/'

def myweight(y=arange(19)):
   myweights = {}
   for j in np.unique(y):
      myweights[j]=len(y)/(y.tolist().count(j)*len(np.unique(y)))
   try:
      for i in mychanges:
         myweights[i[0]]=myweights[i[0]]*i[1]
   except:
      mychanges=[]
   return myweights,mychanges

def tobi(strr):
   if strr=='true':
      return 1
   else:
      return 0

def iskeyX(d):
   if d!='':
      if d[-1].isupper() and len(d)==4:
         return True

def switch(d):
   if d!='':
      if d[-1].isupper() and len(d)==4:
         return d[-1]
```

```
        elif d=='Backspace':
            return 'Back'
        else:
            return d
    else:
        return d

def is_bp(k):
    bad = ['Backspace','Arrow','keycode','Volume']
    for b in bad:
        if b in k[3]:
            return True
    return False

def extractfeatures_keystroke():
    path = 'C:/Zhaoyi_Fan/Dataset/Combo/'
    groups = ['training','testing','All']
    for group in groups:
        if group =='All':
            continue
        newpath = path+group+'/samples/'
        users = os.listdir(newpath)
        for user in users:
            dest = path+group+'/features/'+user+'/'
            if not os.path.exists(dest):
                os.makedirs(dest)
                print (dest,'created')

            files = os.listdir(newpath+user+'/')
            for the_file in files:
                the_temp_name = the_file.split('.csv')[0]
                if the_temp_name.split('_')[1]=='mouse':
                    continue
                #frequently used keys: k_keys
                f_keys = ['Space', 'Backspace', 'KeyI', 'KeyA', 'KeyN',
                     'KeyE', 'KeyO', 'KeyU', 'KeyH', 'Enter', 'KeyG', 'KeyS',
                     'KeyC', 'KeyD', 'KeyT', 'KeyL', 'KeyR', 'KeyY', 'KeyM',
                     'ShiftLeft', 'Digit1', 'KeyB', 'Digit0', 'KeyZ',
                     'Digit2', 'KeyJ', 'KeyW', 'KeyX', 'KeyP', 'KeyF',
                     'KeyV', 'Digit3', 'KeyK', 'Period', 'Digit9', 'KeyQ',
                     'Digit4', 'Digit6', 'Digit7', 'Equal', 'Comma', 'Minus']
                x=[]
                y=[]
                X=[]
                Y=[]

                file = newpath+user+'/'+the_file
                name = the_file.split('.csv')[0]
                with open(file,'r',encoding='UTF-8') as f:
                    data = np.array(pd.DataFrame(csv.reader(f))).tolist()[1:]
```

```
out = []
temp=[]
the_data= []
for i in data[1:]:

   if len(temp) == 0:
      temp.append(i)
   else:
      d=i[3]
      if int(i[2])-int(i[1])<10000 and int(i[2])-int(i[1])>0
          and d!='' and d!='ArrowRight' and d!='ArrowLeft'
          and d!='ArrowDown' and d!='ArrowUp' and
          d!='keycode' and 'Volume' not in d:

         if int(i[1])-int(temp[-1][1])<1000:
            temp.append(i)
         else:
            out.append(temp)
            X.append(temp)
            Y.append(name)
            temp=[]
            temp.append(i)
n=4
for d,name in zip(X,Y):
   if len(d)>n-1:
      for i in range(len(d)-n+1):
         boolean = True
         for boo in range(n-1):
            if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
               boolean=False
         if boolean:
            durations = []
            DD = []
            UU = []
            DU = []
            UD = []

            caps=[]
            shifts=[]

            for j in range(n):
               durations.append(int(d[i+j][2])-int(d[i+j][1]))
               caps.append(tobi(d[i+j][-1]))
               shifts.append(tobi(d[i+j][-2]))

            combo = list(combinations(np.arange(n),2))
            for c in combo:
               DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
               UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
               DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
```

```
                    UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

                  c = durations+DD+UU+DU+UD+caps+shifts

                  my_feature=c[:]
                  the_data.append(my_feature+[name])
          pd.DataFrame(the_data).to_csv(dest+the_file.split('.csv')[0]+'_features.csv',header=0,index

          print (user,group,'data extraction
              completed',len(the_data),'samples involved')

def keystroke_raw_size():
   path = 'C:/Zhaoyi_Fan/Dataset/Combo/training/samples/'
   users = os.listdir(path)
   size = []
   for user in users:
      with
          open(path+user+'/'+user+'_keystroke_train.csv','r',encoding='UTF-8')
          as f:
        data = np.array(pd.DataFrame(csv.reader(f)))
        size.append(len(data))
   sorted_data = sorted(zip(users,size), key=lambda x: (x[1]))
   users,size = zip(*sorted_data)
   users=list(users)
   size = list(size)
   print (size[0],size[-1])
   users.append('Mean')
   size.append(np.mean(size))
   plt.figure(figsize=(10, 6))
   plt.barh(users, size, color='blue')
   plt.xlabel('Keystroke samples')
   plt.ylabel('User')
   plt.gca().invert_yaxis()
   plt.savefig(figure_path1+'ks_raw_size.png')
   plt.savefig(figure_path2+'ks_raw_size.png')
   plt.show()

def type_of_keycode():
   path = 'C:/Zhaoyi_Fan/Dataset/Combo/training/samples/'
   users = os.listdir(path)
   types = []
   for user in users:
      temp = []
      with
          open(path+user+'/'+user+'_keystroke_train.csv','r',encoding='UTF-8')
          as f:
        data = np.array(pd.DataFrame(csv.reader(f)))[1:]
        for d in data:
           if d[3] not in temp:
              temp.append(d[3])
      types.append(len(temp))
   sorted_data = sorted(zip(users,types), key=lambda x: x[1])
   users,types = zip(*sorted_data)
   plt.figure(figsize=(10, 6))
   plt.barh(users, types, color='blue')
   plt.xlabel('Number of keycode categories')
   plt.ylabel('User')
   plt.gca().invert_yaxis()
   plt.savefig(figure_path1+'key_type.png')
   plt.savefig(figure_path2+'key_type.png')
   plt.show()


def Occurrence_of_frequently_used_keys_for_each_user():
   path = 'C:/Zhaoyi_Fan/Dataset/Combo/training/samples/'
   users = os.listdir(path)
   types = {}
   for user in users:
      temp = []
      if user not in types:
        types[user]={}
      with
          open(path+user+'/'+user+'_keystroke_train.csv','r',encoding='UTF-8')
          as f:
        data = np.array(pd.DataFrame(csv.reader(f)))[1:]
        for d in data:
            if d[3] not in types[user]:
              types[user][d[3]]=0
            types[user][d[3]]+=1

   df = pd.DataFrame(types).T
   df = df.dropna(axis=1)
   sorted_columns = df.sum(axis=0).sort_values(ascending=False).index
   df = df[sorted_columns]
   keys_sum = df.sum(axis=0)
   df.loc['Total'] = keys_sum
   for col in df.columns:
      if df[col].dtype == 'float64':
        df[col] = df[col].astype(int)
   for d in np.array(df):
      print (d)
   print (df)
   print (df.iloc[:, :10])
   pd.DataFrame(df).to_csv('KeyOccurrence.csv',header=True,index=True)


def ngraphs_check(n=2):
   path = path = 'C:/Zhaoyi_Fan/Dataset/Combo/training/samples/'

   users = os.listdir(path)
```

```
user_names=[]
user_profiles = {}

key_type = []
di = []
for user in users:
   name = user.split('.')[0]
   user_names.append(name)
   digraphs = []
   with
       open(path+user+'/'+user+'_keystroke_train.csv','r',encoding='UTF-8')
       as f:
      data = np.array(pd.DataFrame(csv.reader(f))).tolist()[1:]

      for i in range(len(data)-n+1):

         bad = False

         for j in range(n-1):
            if int(data[i+1+j][1])-int(data[i+j][1])>=1000 or
                int(data[i+j][2])-int(data[i+j][1])>=10000:
               bad = True
               break

         d = data[i][3]

         temps = [switch(data[i][3])]

         for k in range(n-1):
            temps.append(switch(data[i+1+k][3]))

         di_temp = ' '.join(temps)
         #di_temp = data[i][3]+'+'+data[i+1][3]
         if d!='' and d!='ArrowRight' and d!='ArrowLeft' and
             d!='ArrowDown' and d!='ArrowUp' and d!='keycode' and not
             bad:
         #int(data[i+1][1])-int(data[i][1]) <1000 and
             int(data[i][2])-int(data[i][1])<10000 and
             int(data[i+1][2])-int(data[i+1][1])<10000:
            digraphs.append(di_temp)
            di.append(di_temp)
      user_profiles[name]=digraphs
      #print (len(np.unique(digraphs)))

#total_di = np.unique(di)

alldi = di[:]
total_di = np.unique(di)

total_di_temp = total_di[:].tolist()
```

```
for key in total_di:
   for user in user_profiles:
      if key not in user_profiles[user]:
         total_di_temp.remove(key)
         break

total_di_shared = total_di_temp[:]
print (len(total_di_shared))

keysort = []
for key in total_di_shared:
   keysort.append([alldi.count(key),key])

keysort = sorted(keysort ,key=(lambda x:x[0]),reverse=True)
#print (keysort)

data_compare=[]
for user in user_profiles:
   data = user_profiles[user]
   data_temp = []
   for key in total_di_shared:
      data_temp.append([data.count(key),key])
   b = sorted(data_temp ,key=(lambda x:x[0]),reverse=True)
   data_compare.append(np.array(b).T[1].T.tolist()[:100])
   #print (user,b[:10])
   #print (user,np.array(b).T[1].T.tolist()[:10])

print (reduce(np.intersect1d,(data_compare)))
out = []
index = []
for key in keysort:
   line = []
   index.append(key[1])
   for user in user_profiles:
      t =
          '%.1f'%(100*user_profiles[user].count(key[1])/len(user_profiles[user]))+'%'
      line.append(user_profiles[user].count(key[1]))
   temp =[]
   line.reverse()
   line.append(np.sum(line))
   line.reverse()
   for i in range(len(line)):
      #temp.append('&')
      temp.append(line[i])

   #temp.append('\\\\')
   out.append(temp)
sums=[]
for i in np.array(out).T:
        #print (i)
        try:
            sums.append(np.sum(i.astype(int)))
        except:
            print ('aaa')
            #sums.append('&')
            #continue
    #sums[-1]='\\\\'
    out.append(sums)
    print (len(out))


    print (pd.DataFrame(out,index=index+['Total']))
    pd.DataFrame(out,index=index+['Total']).to_csv(str(n)+'graphs_ks.csv',header=True,index=True)


def segments():
    path = 'C:/Zhaoyi_Fan/Dataset/Combo/training/samples/'
    users = os.listdir(path)

    x,y,X,Y=[],[],[],[]
    TL = []
    users = os.listdir(path)
    names = []
    sumarry = []
    for user in users:
        file = path+user+'/'+user+'_keystroke_train.csv'
        name = user.split('_')[0]
        names.append(name)
        with open(file,'r',encoding='UTF-8') as f:
            data = np.array(pd.DataFrame(csv.reader(f))).tolist()
        out = []
        temp=[]
        the_data= []

        for i in data[1:]:
            if len(temp) == 0:
                temp.append(i)
            else:
                d=i[3]
                if int(i[2])-int(i[1])<10000 and int(i[2])-int(i[1])>0 and
                    d!='' and d!='ArrowRight' and d!='ArrowLeft' and
                    d!='ArrowDown' and d!='ArrowUp' and d!='keycode' and
                    'Volume' not in d:
                    if int(i[1])-int(temp[-1][1])<1000:
                        temp.append(i)

                    else:
                        out.append(temp)
                        X.append(temp)
                Y.append(name)
                #sublength = sublength+len(temp)
                temp=[]
                temp.append(i)
        sublength = [len(i) for i in out]
        TL.append(sublength)
        sumarry.append([len(out),'&',np.min(sublength),'&',np.max(sublength),'&',np.round(np.mean(sublen
        print
            (name,len(out),len(out),np.min(sublength),np.max(sublength),np.mean(sublength),np.std(sublen

    totallength = [len(i) for i in X]
    plt.figure(figsize=(10, 6))
    ax = sns.boxplot(data=TL)
    ax.set_xticklabels(names)
    plt.savefig(figure_path1+'segments.png')
    plt.savefig(figure_path2+'segments.png')

    plt.show()
    print (pd.DataFrame(sumarry,index=names))
    print
        ('Total:',len(X),len(Y),np.min(totallength),np.max(totallength),np.mean(totallength),np.std(tot


def classifier_select(n=4):

    train_path = 'C:/Zhaoyi_Fan/Dataset/Combo/training/samples/'

    users = os.listdir(train_path)

    #frequently used keys: k_keys
    f_keys = ['Space', 'Backspace', 'KeyI', 'KeyA', 'KeyN', 'KeyE',
        'KeyO', 'KeyU', 'KeyH', 'Enter', 'KeyG', 'KeyS', 'KeyC', 'KeyD',
        'KeyT', 'KeyL', 'KeyR', 'KeyY', 'KeyM', 'ShiftLeft', 'Digit1',
        'KeyB', 'Digit0', 'KeyZ', 'Digit2', 'KeyJ', 'KeyW', 'KeyX',
        'KeyP', 'KeyF', 'KeyV', 'Digit3', 'KeyK', 'Period', 'Digit9',
        'KeyQ', 'Digit4', 'Digit6', 'Digit7', 'Equal', 'Comma', 'Minus']
    x=[]
    y=[]
    X=[]
    Y=[]
    for user in users[:]:
        file = train_path+user+'/'+user+'_keystroke_train.csv'
        name = user.split('.csv')[0]
        with open(file,'r',encoding='UTF-8') as f:
            data = np.array(pd.DataFrame(csv.reader(f))).tolist()
        out = []
        temp=[]
        the_data= []
        for i in data[1:]:
```

```
      if len(temp) == 0:
         temp.append(i)
         #print (0,i[3])
      else:
         d=i[3]
         if int(i[2])-int(i[1])<10000 and int(i[2])-int(i[1])>0 and
             d!='' and d!='ArrowRight' and d!='ArrowLeft' and
             d!='ArrowDown' and d!='ArrowUp' and d!='keycode' and
             'Volume' not in d:

            if int(i[1])-int(temp[-1][1])<1000:
            #if int(i[1])-int(temp[-1][1])<1000 and d in f_keys:

               temp.append(i)
               #print ('<1000',i[3])
            else:
               out.append(temp)
               X.append(temp)
               Y.append(name)
               temp=[]
               temp.append(i)

print (len(X),len(Y))

labelencod=preprocessing.LabelEncoder().fit(Y)
Y = labelencod.transform(Y)
X = np.array(X,dtype=object)

#XGB--------------------------------------:
newh_xgb='C:/Zhaoyi_Fan/Dataset/Combo/training/ks_classifier'+str(n)+'/XGB/'
if not os.path.exists(newh_xgb):
   os.makedirs(newh_xgb)
cross_val = StratifiedKFold(n_splits=10)
f1_scores = []
acc_scores = []
pres_scores= []
recall_scores = []
cms = []
i=1
start_xgb = time.time()
print ('Xgb cross validate starts')
for train_index, test_index in cross_val.split(X,Y):

   X_train, X_test = X[train_index], X[test_index]
   Y_train, Y_test = Y[train_index], Y[test_index]
   #print (train_index)
   #print (test_index)

   x_train,x_test,y_train,y_test=[],[],[],[]
```

```
for d,name in zip(X_train,Y_train):
   #for d in out:
   if len(d)>n-1:
      for i in range(len(d)-n+1):
         boolean = True
         for boo in range(n-1):
            if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
               boolean=False
         if boolean:
            durations = []
            DD = []
            UU = []
            DU = []
            UD = []

            caps=[]
            shifts=[]

            for j in range(n):
               durations.append(int(d[i+j][2])-int(d[i+j][1]))
               caps.append(tobi(d[i+j][-1]))
               shifts.append(tobi(d[i+j][-2]))

            combo = list(combinations(np.arange(n),2))
            for c in combo:

               DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
               UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
               DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
               UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

            c = durations+DD+UU+DU+UD+caps+shifts

            my_feature=c[:]
            x_train.append(my_feature)
            #x.append([firstduration,secondduration,DDtime,UUtime,DUtime,UDtime])
            y_train.append(name)
               # temp2=my_feature[:]
               # temp2.append(name)

               # the_data.append(temp2)

   #pd.DataFrame(the_data).to_csv(feat_path+name+'trigraph.csv',header=0,index=False)

#print ('Training Data extraction completed',len(y_train),'samples
    involved')
unwanted=[]
#x = np.delete(x,unwanted,axis=1)
#print (np.unique(y))
#labelencod = preprocessing.LabelEncoder().fit(y_train)
```

```
#y = labelencod.transform(y)


for d,name in zip(X_test,Y_test):
   #for d in out:
   if len(d)>n-1:
      for i in range(len(d)-n+1):
         boolean = True
         for boo in range(n-1):
            if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
               boolean=False
         if boolean:
            durations = []
            DD = []
            UU = []
            DU = []
            UD = []

            caps=[]
            shifts=[]

            for j in range(n):
               durations.append(int(d[i+j][2])-int(d[i+j][1]))
               caps.append(tobi(d[i+j][-1]))
               shifts.append(tobi(d[i+j][-2]))

            combo = list(combinations(np.arange(n),2))
            for c in combo:

               DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
               UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
               DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
               UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

            c = durations+DD+UU+DU+UD+caps+shifts

            my_feature=c[:]
            x_test.append(my_feature)
            #x.append([firstduration,secondduration,DDtime,UUtime,DUtime,UDtime])
            y_test.append(name)
            # temp2=my_feature[:]
            # temp2.append(name)

            # the_data.append(temp2)

   #pd.DataFrame(the_data).to_csv(feat_path+name+'trigraph.csv',header=0,index=False)

#print ('Testing Data extraction completed',len(y_test),'samples
    involved')
   unwanted=[]

   x_train = np.array(x_train)
   y_train = np.array(y_train)
   x_test = np.array(x_test)
   y_test = np.array(y_test)


   clf_xgb =
       XGBClassifier(eval_metric='mlogloss',tree_method='gpu_hist',use_label_encoder=False)
   #clf_xgb = RandomForestClassifier()
   clf_xgb.fit(x_train,y_train)
   y_pred = clf_xgb.predict(x_test)
   f1 = f1_score(y_test,y_pred,average='macro',zero_division=0)
   acc = accuracy_score(y_test,y_pred)

   cm = confusion_matrix(y_test,y_pred)
   pres =
       precision_score(y_test,y_pred,average='macro',zero_division=0)
   recall =
       recall_score(y_test,y_pred,average='macro',zero_division=0)
   pres_scores.append(pres)
   recall_scores.append(recall)
   f1_scores.append(f1)
   acc_scores.append(acc)

   cms.append(cm.tolist())

   pd.DataFrame(cm).to_csv(newh_xgb+str(i)+'_cm.csv',header=0,index=False)
   i=i+1
result = np.sum(cms,axis=0)
t_used = time.time()-start_xgb
pd.DataFrame(result).to_csv(newh_xgb+str(11)+'_cm.csv',header=0,index=False)
print (np.mean(f1_scores),np.std(f1_scores,ddof=1),t_used)
pd.DataFrame([f1_scores,pres_scores,recall_scores,acc_scores]).to_csv(newh_xgb+str(12)+'_report.csv


print ('Xgb cross validate completed. Time used:',)


#random forest-----------------------------------:
newh_rf =
    'C:/Zhaoyi_Fan/Dataset/Combo/training/ks_classifier'+str(n)+'/RandomForest/'
if not os.path.exists(newh_rf):
   os.makedirs(newh_rf)

cross_val = StratifiedKFold(n_splits=10)
f1_scores = []
acc_scores = []
```

```
pres_scores= []
recall_scores = []
cms = []
i=1
start_rf = time.time()
print ('Random forest cross validate starts')
for train_index, test_index in cross_val.split(X,Y):
   X_train, X_test = X[train_index], X[test_index]
   Y_train, Y_test = Y[train_index], Y[test_index]

   x_train,x_test,y_train,y_test=[],[],[],[]

   for d,name in zip(X_train,Y_train):
      #for d in out:
      if len(d)>n-1:
         for i in range(len(d)-n+1):
            boolean = True
            for boo in range(n-1):
               if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
                  boolean=False
            if boolean:
               durations = []
               DD = []
               UU = []
               DU = []
               UD = []

               caps=[]
               shifts=[]

               for j in range(n):
                  durations.append(int(d[i+j][2])-int(d[i+j][1]))
                  caps.append(tobi(d[i+j][-1]))
                  shifts.append(tobi(d[i+j][-2]))

               combo = list(combinations(np.arange(n),2))
               for c in combo:

                  DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
                  UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
                  DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
                  UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

               c = durations+DD+UU+DU+UD+caps+shifts

               my_feature=c[:]
               x_train.append(my_feature)
               #x.append([firstduration,secondduration,DDtime,UUtime,DUtime,UDtime])
               y_train.append(name)
                  # temp2=my_feature[:]
            # temp2.append(name)

            # the_data.append(temp2)

   #pd.DataFrame(the_data).to_csv(feat_path+name+'trigraph.csv',header=0,index=False)

#print ('Training Data extraction completed',len(y_train),'samples
    involved')
unwanted=[]
#x = np.delete(x,unwanted,axis=1)
#print (np.unique(y))
#labelencod = preprocessing.LabelEncoder().fit(y_train)

#y = labelencod.transform(y)


for d,name in zip(X_test,Y_test):
   #for d in out:
   if len(d)>n-1:
      for i in range(len(d)-n+1):
         boolean = True
         for boo in range(n-1):
            if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
               boolean=False
         if boolean:
            durations = []
            DD = []
            UU = []
            DU = []
            UD = []

            caps=[]
            shifts=[]

            for j in range(n):
               durations.append(int(d[i+j][2])-int(d[i+j][1]))
               caps.append(tobi(d[i+j][-1]))
               shifts.append(tobi(d[i+j][-2]))

            combo = list(combinations(np.arange(n),2))
            for c in combo:

               DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
               UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
               DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
               UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

            c = durations+DD+UU+DU+UD+caps+shifts

            my_feature=c[:]
```

```
               x_test.append(my_feature)
               #x.append([firstduration,secondduration,DDtime,UUtime,DUtime,UDtime])
               y_test.append(name)
               # temp2=my_feature[:]
               # temp2.append(name)

               # the_data.append(temp2)

      #pd.DataFrame(the_data).to_csv(feat_path+name+'trigraph.csv',header=0,index=False)

   #print ('Testing Data extraction completed',len(y_test),'samples
       involved')

   unwanted=[]

   x_train = np.array(x_train).astype(float)
   y_train = np.array(y_train)
   x_test = np.array(x_test).astype(float)
   y_test = np.array(y_test)

   clf_rf = RandomForestClassifier()
   try:
      clf_rf.fit(x_train,y_train)
   except:
      for xx in x_train:
         if np.isnan(xx).any():
            print (xx)
      print (np.isnan(x_train).any())

   y_pred = clf_rf.predict(x_test)
   f1 = f1_score(y_test,y_pred,average='macro',zero_division=0)
   acc = accuracy_score(y_test,y_pred)
   cm = confusion_matrix(y_test,y_pred)
   pres =
       precision_score(y_test,y_pred,average='macro',zero_division=0)
   recall =
       recall_score(y_test,y_pred,average='macro',zero_division=0)
   pres_scores.append(pres)
   recall_scores.append(recall)

   f1_scores.append(f1)
   acc_scores.append(acc)
   cms.append(cm.tolist())

   pd.DataFrame(cm).to_csv(newh_rf+str(i)+'_cm.csv',header=0,index=False)
   i=i+1
result = np.sum(cms,axis=0)
pd.DataFrame(result).to_csv(newh_rf+str(11)+'_cm.csv',header=0,index=False)
pd.DataFrame([f1_scores,pres_scores,recall_scores,acc_scores]).to_csv(newh_rf+str(12)+'_report.csv'
print (np.mean(f1_scores),np.std(f1_scores,ddof=1))
print ('Random forest cross validate completed. Time
    used:',time.time()-start_rf,[f1_scores,pres_scores,recall_scores,acc_scores])

   #print (cm)


#SVM linearSVC OvR------------------------------:

newh_linearSVC =
    'C:/Zhaoyi_Fan/Dataset/Combo/training/ks_classifier'+str(n)+'/linearSVC/'
if not os.path.exists(newh_linearSVC):
   os.makedirs(newh_linearSVC)
cross_val = StratifiedKFold(n_splits=10)
f1_scores = []
acc_scores = []
pres_scores= []
recall_scores = []
cms = []
i=1
start_linearSVC = time.time()
print ('linearSVC cross validate starts')
for train_index, test_index in cross_val.split(X,Y):
   X_train, X_test = X[train_index], X[test_index]
   Y_train, Y_test = Y[train_index], Y[test_index]

   x_train,x_test,y_train,y_test=[],[],[],[]

   for d,name in zip(X_train,Y_train):
      #for d in out:
      if len(d)>n-1:
         for i in range(len(d)-n+1):
            boolean = True
            for boo in range(n-1):
               if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
                  boolean=False
            if boolean:
               durations = []
               DD = []
               UU = []
               DU = []
               UD = []

               caps=[]
               shifts=[]

               for j in range(n):
                  durations.append(int(d[i+j][2])-int(d[i+j][1]))
                  caps.append(tobi(d[i+j][-1]))
                  shifts.append(tobi(d[i+j][-2]))
               combo = list(combinations(np.arange(n),2))
               for c in combo:

                  DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
                  UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
                  DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
                  UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

               c = durations+DD+UU+DU+UD+caps+shifts

               my_feature=c[:]
               x_train.append(my_feature)
               #x.append([firstduration,secondduration,DDtime,UUtime,DUtime,UDtime])
               y_train.append(name)
                  # temp2=my_feature[:]
                  # temp2.append(name)

                  # the_data.append(temp2)

      #pd.DataFrame(the_data).to_csv(feat_path+name+’trigraph.csv’,header=0,index=False)

   #print (’Training Data extraction completed’,len(y_train),’samples
       involved’)
   unwanted=[]

   #y_train = labelencod.transform(y_train)

   for d,name in zip(X_test,Y_test):
      #for d in out:
      if len(d)>n-1:
         for i in range(len(d)-n+1):
            boolean = True
            for boo in range(n-1):
               if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
                  boolean=False
            if boolean:
               durations = []
               DD = []
               UU = []
               DU = []
               UD = []

               caps=[]
               shifts=[]

               for j in range(n):
                  durations.append(int(d[i+j][2])-int(d[i+j][1]))
                  caps.append(tobi(d[i+j][-1]))
                  shifts.append(tobi(d[i+j][-2]))
```

```
          combo = list(combinations(np.arange(n),2))
          for c in combo:

             DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
             UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
             DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
             UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

          c = durations+DD+UU+DU+UD+caps+shifts

          my_feature=c[:]
          x_test.append(my_feature)
          #x.append([firstduration,secondduration,DDtime,UUtime,DUtime,UDtime])
          y_test.append(name)
          # temp2=my_feature[:]
          # temp2.append(name)

          # the_data.append(temp2)

   #pd.DataFrame(the_data).to_csv(feat_path+name+'trigraph.csv',header=0,index=False)

#print ('Testing Data extraction completed',len(y_test),'samples
    involved')

unwanted=[]

x_train = np.array(x_train).astype(float)
y_train = np.array(y_train)
x_test = np.array(x_test).astype(float)
y_test = np.array(y_test)

x_train = preprocessing.StandardScaler().fit_transform(x_train)
x_test = preprocessing.StandardScaler().fit_transform(x_test)

#clf_linearSVC =
    XGBClassifier(eval_metric='mlogloss',tree_method='gpu_hist',use_label_encoder=False)
clf_linearSVC = LinearSVC()
clf_linearSVC.fit(x_train,y_train)
y_pred = clf_linearSVC.predict(x_test)
f1 = f1_score(y_test,y_pred,average='macro',zero_division=0)
acc = accuracy_score(y_test,y_pred)
cm = confusion_matrix(y_test,y_pred)
pres =
    precision_score(y_test,y_pred,average='macro',zero_division=0)
recall =
    recall_score(y_test,y_pred,average='macro',zero_division=0)
pres_scores.append(pres)
recall_scores.append(recall)
f1_scores.append(f1)
acc_scores.append(acc)
```

```
   cms.append(cm.tolist())

   pd.DataFrame(cm).to_csv(newh_linearSVC+str(i)+'_cm.csv',header=0,index=False)
   i=i+1
result = np.sum(cms,axis=0)
pd.DataFrame(result).to_csv(newh_linearSVC+str(11)+'_cm.csv',header=0,index=False)
pd.DataFrame([f1_scores,pres_scores,recall_scores,acc_scores]).to_csv(newh_linearSVC+str(12)+'_repo
print (np.mean(f1_scores),np.std(f1_scores,ddof=1))
print ('LinearSVC cross validate completed. Time
    used:',time.time()-start_linearSVC,[f1_scores,pres_scores,recall_scores,acc_scores])

#SVM SVC with linear OvO------------------------------:

newh_SVClin =
    'C:/Zhaoyi_Fan/Dataset/Combo/training/ks_classifier'+str(n)+'/SVClin/'
if not os.path.exists(newh_SVClin):
   os.makedirs(newh_SVClin)
cross_val = StratifiedKFold(n_splits=10)
f1_scores = []
acc_scores = []
pres_scores= []
recall_scores = []
cms = []
i=1
start_SVClin = time.time()
print ('SVClin cross validate starts')
for train_index, test_index in cross_val.split(X,Y):
   X_train, X_test = X[train_index], X[test_index]
   Y_train, Y_test = Y[train_index], Y[test_index]

   x_train,x_test,y_train,y_test=[],[],[],[]

   for d,name in zip(X_train,Y_train):
      #for d in out:
      if len(d)>n-1:
         for i in range(len(d)-n+1):
            boolean = True
            for boo in range(n-1):
               if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
                  boolean=False
            if boolean:
               durations = []
               DD = []
               UU = []
               DU = []
               UD = []

               caps=[]
               shifts=[]
            for j in range(n):
               durations.append(int(d[i+j][2])-int(d[i+j][1]))
               caps.append(tobi(d[i+j][-1]))
               shifts.append(tobi(d[i+j][-2]))

            combo = list(combinations(np.arange(n),2))
            for c in combo:

               DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
               UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
               DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
               UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

            c = durations+DD+UU+DU+UD+caps+shifts

            my_feature=c[:]
            x_train.append(my_feature)
            #x.append([firstduration,secondduration,DDtime,UUtime,DUtime,UDtime])
            y_train.append(name)
               # temp2=my_feature[:]
               # temp2.append(name)

               # the_data.append(temp2)

   #pd.DataFrame(the_data).to_csv(feat_path+name+'trigraph.csv',header=0,index=False)

#print ('Training Data extraction completed',len(y_train),'samples
    involved')
unwanted=[]
#x = np.delete(x,unwanted,axis=1)
#print (np.unique(y))
#labelencod = preprocessing.LabelEncoder().fit(y_train)

#y = labelencod.transform(y)

#y_train = labelencod.transform(y_train)

for d,name in zip(X_test,Y_test):
   #for d in out:
   if len(d)>n-1:
      for i in range(len(d)-n+1):
         boolean = True
         for boo in range(n-1):
            if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
               boolean=False
         if boolean:
            durations = []
            DD = []
            UU = []
            DU = []
```

```
          UD = []

          caps=[]
          shifts=[]

          for j in range(n):
             durations.append(int(d[i+j][2])-int(d[i+j][1]))
             caps.append(tobi(d[i+j][-1]))
             shifts.append(tobi(d[i+j][-2]))

          combo = list(combinations(np.arange(n),2))
          for c in combo:

             DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
             UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
             DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
             UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

          c = durations+DD+UU+DU+UD+caps+shifts

          my_feature=c[:]
          x_test.append(my_feature)
          #x.append([firstduration,secondduration,DDtime,UUtime,DUtime,UDtime])
          y_test.append(name)
          # temp2=my_feature[:]
          # temp2.append(name)

          # the_data.append(temp2)

   #pd.DataFrame(the_data).to_csv(feat_path+name+'trigraph.csv',header=0,index=False)

#print ('Testing Data extraction completed',len(y_test),'samples
    involved')

unwanted=[]

x_train = np.array(x_train).astype(float)
y_train = np.array(y_train)
x_test = np.array(x_test).astype(float)
y_test = np.array(y_test)

x_train = preprocessing.StandardScaler().fit_transform(x_train)
x_test = preprocessing.StandardScaler().fit_transform(x_test)

#clf_linear =
    XGBClassifier(eval_metric='mlogloss',tree_method='gpu_hist',use_label_encoder=False)
clf_SVClin = SVC(kernel='linear')
clf_SVClin.fit(x_train,y_train)
y_pred = clf_SVClin.predict(x_test)
f1 = f1_score(y_test,y_pred,average='macro',zero_division=0)
```

```
   acc = accuracy_score(y_test,y_pred)
   cm = confusion_matrix(y_test,y_pred)
   pres =
       precision_score(y_test,y_pred,average='macro',zero_division=0)
   recall =
       recall_score(y_test,y_pred,average='macro',zero_division=0)
   pres_scores.append(pres)
   recall_scores.append(recall)
   f1_scores.append(f1)
   acc_scores.append(acc)
   cms.append(cm.tolist())

   pd.DataFrame(cm).to_csv(newh_SVClin+str(i)+'_cm.csv',header=0,index=False)
   i=i+1
result = np.sum(cms,axis=0)
pd.DataFrame(result).to_csv(newh_SVClin+str(11)+'_cm.csv',header=0,index=False)
pd.DataFrame([f1_scores,pres_scores,recall_scores,acc_scores]).to_csv(newh_SVClin+str(12)+'_report.
print (np.mean(f1_scores),np.std(f1_scores,ddof=1))
print ('SVClin cross validate completed. Time
    used:',time.time()-start_SVClin,[f1_scores,pres_scores,recall_scores,acc_scores])

#SVM SVC with rbf OvO------------------------------:
newh_SVCrbf =
    'C:/Zhaoyi_Fan/Dataset/Combo/training/ks_classifier'+str(n)+'/SVCrbf/'
if not os.path.exists(newh_SVCrbf):
   os.makedirs(newh_SVCrbf)
cross_val = StratifiedKFold(n_splits=10)
f1_scores = []
acc_scores = []
pres_scores= []
recall_scores = []
cms = []
i=1
start_SVCrbf = time.time()
print ('SVCrbf cross validate starts')
for train_index, test_index in cross_val.split(X,Y):
   X_train, X_test = X[train_index], X[test_index]
   Y_train, Y_test = Y[train_index], Y[test_index]

   x_train,x_test,y_train,y_test=[],[],[],[]

   for d,name in zip(X_train,Y_train):
      #for d in out:
      if len(d)>n-1:
         for i in range(len(d)-n+1):
            boolean = True
            for boo in range(n-1):
               if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
                  boolean=False
            if boolean:
```

```
            durations = []
            DD = []
            UU = []
            DU = []
            UD = []

            caps=[]
            shifts=[]

            for j in range(n):
               durations.append(int(d[i+j][2])-int(d[i+j][1]))
               caps.append(tobi(d[i+j][-1]))
               shifts.append(tobi(d[i+j][-2]))

            combo = list(combinations(np.arange(n),2))
            for c in combo:

               DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
               UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
               DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
               UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

            c = durations+DD+UU+DU+UD+caps+shifts

            my_feature=c[:]
            x_train.append(my_feature)
            #x.append([firstduration,secondduration,DDtime,UUtime,DUtime,UDtime])
            y_train.append(name)
               # temp2=my_feature[:]
               # temp2.append(name)

               # the_data.append(temp2)

   #pd.DataFrame(the_data).to_csv(feat_path+name+'trigraph.csv',header=0,index=False)

#print ('Training Data extraction completed',len(y_train),'samples
    involved')
unwanted=[]
#x = np.delete(x,unwanted,axis=1)
#print (np.unique(y))
#labelencod = preprocessing.LabelEncoder().fit(y_train)

#y = labelencod.transform(y)
#x_train = np.delete(x_train,unwanted,axis=1)
#y_train = labelencod.transform(y_train)

for d,name in zip(X_test,Y_test):
   #for d in out:
   if len(d)>n-1:
      for i in range(len(d)-n+1):
        boolean = True
        for boo in range(n-1):
            if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
                boolean=False
        if boolean:
            durations = []
            DD = []
            UU = []
            DU = []
            UD = []

            caps=[]
            shifts=[]

            for j in range(n):
                durations.append(int(d[i+j][2])-int(d[i+j][1]))
                caps.append(tobi(d[i+j][-1]))
                shifts.append(tobi(d[i+j][-2]))

            combo = list(combinations(np.arange(n),2))
            for c in combo:

                DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
                UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
                DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
                UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

            c = durations+DD+UU+DU+UD+caps+shifts

            my_feature=c[:]
            x_test.append(my_feature)
            #x.append([firstduration,secondduration,DDtime,UUtime,DUtime,UDtime])
            y_test.append(name)
            # temp2=my_feature[:]
            # temp2.append(name)

            # the_data.append(temp2)

    #pd.DataFrame(the_data).to_csv(feat_path+name+'trigraph.csv',header=0,index=False)

#print ('Testing Data extraction completed',len(y_test),'samples
    involved')

unwanted=[]

x_train = np.array(x_train).astype(float)
y_train = np.array(y_train)
x_test = np.array(x_test).astype(float)
y_test = np.array(y_test)
```

```
      x_train = preprocessing.StandardScaler().fit_transform(x_train)
      x_test = preprocessing.StandardScaler().fit_transform(x_test)
      #y_test = labelencod.transform(y_test)
      #clf_linear =
          XGBClassifier(eval_metric='mlogloss',tree_method='gpu_hist',use_label_encoder=False)
      clf_SVCrbf = SVC(kernel='rbf')
      clf_SVCrbf.fit(x_train,y_train)
      y_pred = clf_SVCrbf.predict(x_test)
      f1 = f1_score(y_test,y_pred,average='macro',zero_division=0)
      acc = accuracy_score(y_test,y_pred)
      cm = confusion_matrix(y_test,y_pred)

      pres =
          precision_score(y_test,y_pred,average='macro',zero_division=0)
      recall =
          recall_score(y_test,y_pred,average='macro',zero_division=0)
      pres_scores.append(pres)
      recall_scores.append(recall)

      f1_scores.append(f1)
      acc_scores.append(acc)
      cms.append(cm.tolist())

      pd.DataFrame(cm).to_csv(newh_SVCrbf+str(i)+'_cm.csv',header=0,index=False)
      i=i+1
   result = np.sum(cms,axis=0)
   pd.DataFrame(result).to_csv(newh_SVCrbf+str(11)+'_cm.csv',header=0,index=False)
   pd.DataFrame([f1_scores,pres_scores,recall_scores,acc_scores]).to_csv(newh_SVCrbf+str(12)+'_report.
   print (np.mean(f1_scores),np.std(f1_scores,ddof=1))
   print ('SVCrbf cross validate completed. Time
       used:',time.time()-start_SVCrbf,[f1_scores,pres_scores,recall_scores,acc_scores])


def hyper_parameters(n=4):
   train_path = 'C:/Zhaoyi_Fan/Dataset/Combo/training/samples/'

   users = os.listdir(train_path)

   #frequently used keys: k_keys
   f_keys = ['Space', 'Backspace', 'KeyI', 'KeyA', 'KeyN', 'KeyE',
       'KeyO', 'KeyU', 'KeyH', 'Enter', 'KeyG', 'KeyS', 'KeyC', 'KeyD',
       'KeyT', 'KeyL', 'KeyR', 'KeyY', 'KeyM', 'ShiftLeft', 'Digit1',
       'KeyB', 'Digit0', 'KeyZ', 'Digit2', 'KeyJ', 'KeyW', 'KeyX',
       'KeyP', 'KeyF', 'KeyV', 'Digit3', 'KeyK', 'Period', 'Digit9',
       'KeyQ', 'Digit4', 'Digit6', 'Digit7', 'Equal', 'Comma', 'Minus']
   x=[]
   y=[]
   X=[]
   Y=[]
   for user in users[:]:
```

```
   file = train_path+user+'/'+user+'_keystroke_train.csv'
   name = user.split('.csv')[0]

   with open(file,'r',encoding='UTF-8') as f:
      data = np.array(pd.DataFrame(csv.reader(f))).tolist()
   out = []
   temp=[]
   the_data= []
   for i in data[1:]:

      if len(temp) == 0:
         temp.append(i)
         #print (0,i[3])
      else:
         d=i[3]
         if int(i[2])-int(i[1])<10000 and int(i[2])-int(i[1])>0 and
             d!='' and d!='ArrowRight' and d!='ArrowLeft' and
             d!='ArrowDown' and d!='ArrowUp' and d!='keycode' and
             'Volume' not in d:

            if int(i[1])-int(temp[-1][1])<1000:
            #if int(i[1])-int(temp[-1][1])<1000 and d in f_keys:

               temp.append(i)
               #print ('<1000',i[3])
            else:
               out.append(temp)
               X.append(temp)
               Y.append(name)
               temp=[]
               temp.append(i)

print (len(X),len(Y))

labelencod=preprocessing.LabelEncoder().fit(Y)
Y = labelencod.transform(Y)
X = np.array(X,dtype=object)

params =[]
#0.2 600,4,3,
for learning_rate in [0.1,0.2,0.3]:
   for n_estimators in [400,500,600]:
      for max_depth in [3,4,5,6]:
         for min_child_weight in [3,4,5,6]:
            cv_params = {
                     'learning_rate': learning_rate,
                     'n_estimators': n_estimators,
                     'max_depth': max_depth,
                     'min_child_weight': min_child_weight,
                     'objective': 'multi:softprob',
                  'use_label_encoder': False,
                  'eval_metric': 'mlogloss',
                  'tree_method': 'gpu_hist',
                    }
         params.append(cv_params)

final_report=[]

for cv_parmas in params:
   try:
      subpath =
          str(cv_parmas['learning_rate']*10)+'_'+str(cv_parmas['n_estimators'])+'_'+str(cv_parmas['r
   except:
      print (cv_parmas)

   #XGB----------------------------------------:
   newh_xgb =
        'C:/Zhaoyi_Fan/Dataset/Combo/training/ks_hyper_parameters_'+str(n)+'/XGB/GridSearchCV/'+subp
   if not os.path.exists(newh_xgb):
      os.makedirs(newh_xgb)

   cross_val = StratifiedKFold(n_splits=10)
   f1_scores = []
   acc_scores = []
   cms = []
   i=1
   start_xgb = time.time()
   print ('Xgb cross validate starts')
   for train_index, test_index in cross_val.split(X,Y):

      X_train, X_test = X[train_index], X[test_index]
      Y_train, Y_test = Y[train_index], Y[test_index]
      #print (train_index)
      #print (test_index)

      x_train,x_test,y_train,y_test=[],[],[],[]

      for d,name in zip(X_train,Y_train):
         #for d in out:
         if len(d)>n-1:
            for i in range(len(d)-n+1):
               boolean = True
               for boo in range(n-1):
                  if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
                     boolean=False
               if boolean:
                  durations = []
                  DD = []
                  UU = []
                  DU = []
                  UD = []

                  caps=[]
                  shifts=[]

                  for j in range(n):
                     durations.append(int(d[i+j][2])-int(d[i+j][1]))
                     caps.append(tobi(d[i+j][-1]))
                     shifts.append(tobi(d[i+j][-2]))

                  combo = list(combinations(np.arange(n),2))
                  for c in combo:

                     DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
                     UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
                     DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
                     UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

                  c = durations+DD+UU+DU+UD+caps+shifts

                  my_feature=c[:]
                  x_train.append(my_feature)
                  y_train.append(name)


      for d,name in zip(X_test,Y_test):
         #for d in out:
         if len(d)>n-1:
            for i in range(len(d)-n+1):
               boolean = True
               for boo in range(n-1):
                  if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
                     boolean=False
               if boolean:
                  durations = []
                  DD = []
                  UU = []
                  DU = []
                  UD = []

                  caps=[]
                  shifts=[]

                  for j in range(n):
                     durations.append(int(d[i+j][2])-int(d[i+j][1]))
                     caps.append(tobi(d[i+j][-1]))
                     shifts.append(tobi(d[i+j][-2]))

                  combo = list(combinations(np.arange(n),2))
                  for c in combo:
                          DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
                          UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
                          DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
                          UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

                      c = durations+DD+UU+DU+UD+caps+shifts

                      my_feature=c[:]
                      x_test.append(my_feature)
                      y_test.append(name)

              unwanted=[]

              x_train = np.array(x_train)
              y_train = np.array(y_train)
              x_test = np.array(x_test)
              y_test = np.array(y_test)


              clf_xgb = XGBClassifier(**cv_parmas)
              #clf_xgb = RandomForestClassifier()
              clf_xgb.fit(x_train,y_train)
              y_pred = clf_xgb.predict(x_test)
              f1 = f1_score(y_test,y_pred,average='macro',zero_division=0)
              acc = accuracy_score(y_test,y_pred)
              cm = confusion_matrix(y_test,y_pred)

              f1_scores.append(f1)
              acc_scores.append(acc)
              cms.append(cm.tolist())

              pd.DataFrame(cm).to_csv(newh_xgb+str(i)+'_cm.csv',header=0,index=False)
              i=i+1
          result = np.sum(cms,axis=0)
          pd.DataFrame(result).to_csv(newh_xgb+str(11)+'_cm.csv',header=0,index=False)
          pd.DataFrame([f1_scores,acc_scores]).to_csv(newh_xgb+str(12)+'_report.csv',header=0,index=False)
          f1_mean = np.mean(f1_scores)
          f1_std = np.std(f1_scores,ddof=1)
          print (np.mean(f1_scores),np.std(f1_scores,ddof=1))
          print ('Xgb cross validate completed. Time
              used:',np.round(time.time()-start_xgb,2),'[learning_rate:',cv_parmas['learning_rate'],']
              [n_estimators:',cv_parmas['n_estimators'],']
              [max_depth:',cv_parmas['max_depth'],']
              [min_child_weight:',cv_parmas['min_child_weight'],']')
          print ('---------------------')
          final_report.append([cv_parmas['learning_rate'],cv_parmas['n_estimators'],cv_parmas['max_depth']
      col =
          ['learning_rate','n_estimators','max_depth','min_child_weight','f1-macro-mean','f1-macro-std']
      pd.DataFrame(final_report,columns=col).to_csv('C:/Zhaoyi_Fan/Dataset/Combo/training/ks_hyper_parame
      def train_n_testvesion(n=4):
         path = 'C:/Zhaoyi_Fan/Dataset/Combo/training/samples/'
         feat_path = 'C:/Zhaoyi_Fan/Dataset/Combo/training/features/'
         dest =
             'C:/Zhaoyi_Fan/Dataset/Combo/training/results/Keystroke/'+str(n)+'_graphs/XGB/'
         if not os.path.exists(dest):
            os.makedirs(dest)
            print (dest,'created')
         newh = dest
         users = os.listdir(feat_path)

         #frequently used keys: k_keys
         f_keys = ['Space', 'Backspace', 'KeyI', 'KeyA', 'KeyN', 'KeyE',
             'KeyO', 'KeyU', 'KeyH', 'Enter', 'KeyG', 'KeyS', 'KeyC', 'KeyD',
             'KeyT', 'KeyL', 'KeyR', 'KeyY', 'KeyM', 'ShiftLeft', 'Digit1',
             'KeyB', 'Digit0', 'KeyZ', 'Digit2', 'KeyJ', 'KeyW', 'KeyX',
             'KeyP', 'KeyF', 'KeyV', 'Digit3', 'KeyK', 'Period', 'Digit9',
             'KeyQ', 'Digit4', 'Digit6', 'Digit7', 'Equal', 'Comma', 'Minus']
         x=[]
         y=[]
         X=[]
         Y=[]
         for user in users[:]:
            file = path+user+'/'+user+'_keystroke_train.csv'
            name = user.split('_')[0]
            with open(file,'r',encoding='UTF-8') as f:
               data = np.array(pd.DataFrame(csv.reader(f))).tolist()
            out = []
            temp=[]
            the_data= []
            for i in data[1:]:
               if len(temp) == 0:
                  temp.append(i)
                  #print (0,i[3])
               else:
                  d=i[3]
                  if int(i[2])-int(i[1])<10000 and int(i[2])-int(i[1])>0 and
                      d!='' and d!='ArrowRight' and d!='ArrowLeft' and
                      d!='ArrowDown' and d!='ArrowUp' and d!='keycode' and
                      'Volume' not in d:

                     if int(i[1])-int(temp[-1][1])<1000:
                     #if int(i[1])-int(temp[-1][1])<1000 and d in f_keys:

                        temp.append(i)
                        #print ('<1000',i[3])
                     else:
                        out.append(temp)
                      X.append(temp)
                      Y.append(name)
                      temp=[]
                      temp.append(i)

      print (len(X),len(Y))

      X_train,X_test,Y_train,Y_test =
          train_test_split(X,Y,test_size=0.2,stratify=Y,random_state=1)

      x_train=[]
      x_test=[]
      y_train=[]
      y_test=[]

      #for sample_set in
          [[X_train,Y_train,x_train,y_train],[X_test,Y_test,x_test,y_test]]:
      for d,name in zip(X_train,Y_train):
         #for d in out:
         if len(d)>n-1:
            for i in range(len(d)-n+1):
               boolean = True
               for boo in range(n-1):
                  if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
                     boolean=False
               if boolean:
                  durations = []
                  DD = []
                  UU = []
                  DU = []
                  UD = []

                  caps=[]
                  shifts=[]

                  for j in range(n):
                     durations.append(int(d[i+j][2])-int(d[i+j][1]))
                     caps.append(tobi(d[i+j][-1]))
                     shifts.append(tobi(d[i+j][-2]))

                  combo = list(combinations(np.arange(n),2))
                  for c in combo:

                     DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
                     UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
                     DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
                     UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

                  c = durations+DD+UU+DU+UD+caps+shifts
                  my_feature=c[:]
                  x_train.append(my_feature)
                  #x.append([firstduration,secondduration,DDtime,UUtime,DUtime,UDtime])
                  y_train.append(name)
                     # temp2=my_feature[:]
                     # temp2.append(name)

                     # the_data.append(temp2)

         #pd.DataFrame(the_data).to_csv(feat_path+name+'trigraph.csv',header=0,index=False)

      print ('Training Data extraction completed',len(y_train),'samples
           involved')
      unwanted=[]
      #x = np.delete(x,unwanted,axis=1)
      #print (np.unique(y))
      labelencod = preprocessing.LabelEncoder().fit(y_train)

      #y = labelencod.transform(y)
      x_train = np.delete(x_train,unwanted,axis=1)
      y_train = labelencod.transform(y_train)

      for d,name in zip(X_test,Y_test):
         #for d in out:
         if len(d)>n-1:
            for i in range(len(d)-n+1):
               boolean = True
               for boo in range(n-1):
                  if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
                     boolean=False
               if boolean:
                  durations = []
                  DD = []
                  UU = []
                  DU = []
                  UD = []

                  caps=[]
                  shifts=[]

                  for j in range(n):
                     durations.append(int(d[i+j][2])-int(d[i+j][1]))
                     caps.append(tobi(d[i+j][-1]))
                     shifts.append(tobi(d[i+j][-2]))

                  combo = list(combinations(np.arange(n),2))
                  for c in combo:

                     DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
                     UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
```

```
                DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
                UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

             c = durations+DD+UU+DU+UD+caps+shifts

             my_feature=c[:]
             x_test.append(my_feature)
             #x.append([firstduration,secondduration,DDtime,UUtime,DUtime,UDtime])
             y_test.append(name)
             # temp2=my_feature[:]
             # temp2.append(name)

             # the_data.append(temp2)

   #pd.DataFrame(the_data).to_csv(feat_path+name+'trigraph.csv',header=0,index=False)

print ('Testing Data extraction completed',len(y_test),'samples
    involved')
unwanted=[]

x_test = np.delete(x_test,unwanted,axis=1)
y_test = labelencod.transform(y_test)


pred_prob=[]
featurescore=[]

mychanges = []
theweights = []

try:
   print ('XGB fitting started')
   #cross_val = StratifiedKFold(n_splits=cvs)
   clf =
       XGBClassifier(objective='multi:softprob',use_label_encoder=False,
                eval_metric='mlogloss',tree_method='gpu_hist',
                learning_rate=0.3,n_estimators=600,
                max_depth = 5,min_child_weight=5,
                )
   #clf = lgb.LGBMClassifier()
   features_impts = []

   myweights,mychanges = myweight(y_train)
   print (myweights,mychanges)
   class_weights =
       class_weight.compute_sample_weight(myweights,y_train)
   print ('XGB fitting started',len(y_train), 'samples involved')
   clf.fit(x_train,y_train,sample_weight=class_weights)
   print ('XGB fitting completed')
   print ('XGB predicting started')
   pre_temp = clf.predict_proba(x_test)
   print ('XGB predicting completed')
   featurescore = clf.feature_importances_.tolist()

   pred_prob = np.array(pre_temp)

   labelset = np.unique(y_test)
   names = labelencod.inverse_transform(labelset)


except Exception as e:
   print (e)
   #print (folder)
   return 'bad'

myweights_total,mychanges=myweight(y)
theweights.append(myweights_total)
head = np.array(theweights[0].keys())
print (head)
#heads = labelencod.inverse_transform(head).tolist()
heads = names.tolist()
weights=pd.DataFrame(theweights)
weights = np.array(weights).tolist()
weights.insert(0,heads)
weights=pd.DataFrame(weights)

dic = {} #for computing the overall result
dic2 = {} #for computing each user's performance
#print (list(labelset))
for i in list(labelset):
   dic[i]=[]
   #dic2[i]=[]
for i in range(len(y_test)):
   dic[y_test[i]].append(pred_prob[i])
pred_probtemp = []
y_testtemp = []
for i in dic:
   #ypred=[]
   #ytrue=[]
   for j in dic[i]:
      pred_probtemp.append(j)
      y_testtemp.append(i)
      #ypred.append(j.tolist().index(j.max()))
      #ytrue.append(i)
   #dic2[names[i]]=[ypred,ytrue]

#everyone=[]


pred_prob = pred_probtemp[:]
```

```
y_test = y_testtemp[:]

probs =
    pd.concat([pd.DataFrame(pred_prob),pd.DataFrame(y_test)],axis=1)
probs.to_csv(dest+'ks_probabilities.csv',header=0,index=False)


outs = []
f1_micro=[]
f1_macro=[]

precision_micro=[]
precision_macro=[]

recall_micro=[]
recall_macro=[]
times = []

accuracy = []
rpt = []
f1s = []

for i in range(1,32,1):
   y_pred = []
   y_true = []
   for j in range(len(y_test)-i+1):
      if y_test[j]==y_test[j+i-1]:
         #y_pred_byprob=sum(np.array(pred_prob[j:j+i]),axis=0)
         y_pred_byprob=sum(np.array(pred_prob[j:j+i]),axis=0)
         y_pred.append(y_pred_byprob.tolist().index(y_pred_byprob.max()))
         y_true.append(y_test[j])
   #y_true = labelencod.inverse_transform(y_true)
   #y_pred = labelencod.inverse_transform(y_pred)

   outs.append([
         #'%.2f'%(f1_score(y_true,y_pred,average='micro')*100),
         float('%.3f'%(f1_score(y_true,y_pred,average='macro',zero_division=0))),
         float('%.3f'%(f1_score(y_true,y_pred,average='weighted',zero_division=0))),
         '-',
         #'%.2f'%(precision_score(y_true,y_pred,average='micro')*100),
         float('%.3f'%(precision_score(y_true,y_pred,average='macro',zero_division=0))),
         float('%.3f'%(precision_score(y_true,y_pred,average='weighted',zero_division=0))),
         '-',
         #'%.2f'%(recall_score(y_true,y_pred,average='micro')*100),
         float('%.3f'%(recall_score(y_true,y_pred,average='macro',zero_division=0))),
         float('%.3f'%(recall_score(y_true,y_pred,average='weighted',zero_division=0))),
         '-',
         float('%.3f'%(accuracy_score(y_true,y_pred))),
         ])
   f1s.append(f1_score(y_true,y_pred,average='macro',zero_division=0))
```

```
f1_micro.append(f1_score(y_true,y_pred,average='micro',zero_division=0)*100)
f1_macro.append(f1_score(y_true,y_pred,average='macro',zero_division=0)*100)
precision_micro.append(precision_score(y_true,y_pred,average='micro',zero_division=0)*100)
precision_macro.append(precision_score(y_true,y_pred,average='macro',zero_division=0)*100)
recall_micro.append(recall_score(y_true,y_pred,average='micro',zero_division=0)*100)
recall_macro.append(recall_score(y_true,y_pred,average='macro',zero_division=0)*100)
accuracy.append(accuracy_score(y_true,y_pred)*100)
times.append(i)
#newh = 'C:/Workspace/Python/mouse
    sets/Bogazici/browsing/combined_Train_LegalInternal/final_results_cv/3_timesIQR/0+/'+cata+'/

if i==1:
   if not os.path.exists(newh+str(i)+'/'):
      os.makedirs(newh+str(i)+'/')
   rpt = classification_report(y_true,y_pred,output_dict=True)
   #report=classification_report(y_true,y_pred,output_dict=True)
   pd.DataFrame(rpt).T.to_csv(newh+str(i)+'/'+str(i)+'ks_classification_report.csv',header=0,inde
   print (classification_report(y_true,y_pred))
   print (labelencod.inverse_transform(clf.classes_))

if (i+4)/5==int((i+4)/5):
   if not os.path.exists(newh+str(i)+'/'):
      os.makedirs(newh+str(i)+'/')
   cm_old = confusion_matrix(y_true, y_pred)
   cm = np.array(pd.DataFrame(cm_old)).tolist()
   cm_=[]
   for k in cm:
      temp=[]
      for j in k:
         temp.append(float('%.1f'%(100*j/sum(k))))
      cm_.append(temp)
   cm =mat(cm_.copy())

   report = classification_report(y_true,y_pred,output_dict=True)
   #report=classification_report(y_true,y_pred,output_dict=True)
   pd.DataFrame(report).T.to_csv(newh+str(i)+'/'+str(i)+'ks_classification_report.csv',header=0,i

   plt.figure(1,figsize=(10.24,8))
   ax1 = plt.axes()
   disp2 = ConfusionMatrixDisplay(confusion_matrix=cm)
   disp2.plot(cmap=plt.cm.Blues,include_values=False,ax=ax1)
   plt.savefig(newh+str(i)+'/'+str(i)+'ks_cm.png')
   plt.clf()

   plt.figure(3,figsize=(10.24,8))
   ax3 = plt.axes()
   disp3 =
       ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=heads)
   disp3.plot(cmap=plt.cm.Blues,include_values=True,ax=ax3)
   plt.savefig(newh+str(i)+'/'+str(i)+'ks_cm_newnew.png')
```

```
        plt.clf()

        plt.figure(4,figsize=(10.24,8))
        ax4 = plt.axes()
        disp4 =
            ConfusionMatrixDisplay(confusion_matrix=cm_old,display_labels=heads)
        disp4.plot(cmap=plt.cm.Blues,include_values=True,ax=ax4)
        plt.savefig(newh+str(i)+'/'+str(i)+'ks_cm_newnewnew.png')
        plt.clf()

        #np.set_printoptions(precision=2)
        cm_normalized = confusion_matrix(y_true,
            y_pred,normalize='true')
        cm_n = mat(np.round(cm_normalized,3))
        plt.figure(5,figsize=(10.24,8))
        ax5 = plt.axes()
        disp5 =
            ConfusionMatrixDisplay(confusion_matrix=cm_n,display_labels=heads)
        disp5.plot(cmap=plt.cm.Blues,include_values=True,ax=ax5)
        plt.savefig(newh+str(i)+'/'+str(i)+'ks_cm_newnewnew2.png')
        plt.clf()

        #np.set_printoptions(precision=2)
        cm_normalized2 = confusion_matrix(y_true,
            y_pred,normalize='pred')
        cm_n = mat(np.round(cm_normalized2,3))
        plt.figure(6,figsize=(10.24,8))
        ax6 = plt.axes()
        disp6 =
            ConfusionMatrixDisplay(confusion_matrix=cm_n,display_labels=heads)
        disp6.plot(cmap=plt.cm.Blues,include_values=True,ax=ax6)
        plt.savefig(newh+str(i)+'/'+str(i)+'ks_cm_newnewnew3.png')
        plt.clf()

print (pd.DataFrame(outs))


plt.figure(33,figsize=(10.24,8))
ax2 = plt.axes()
plot_importance(clf,ax=ax2,importance_type='gain')
plt.savefig(newh+'ks_FI.png')
plt.clf()

print (featurescore)

# plt.show()

plt.figure(34,figsize=(10.24,8))
#print (f1_macro,np.min(f1_macro))
#print
```

```
        (int(np.min([np.min(f1_macro),np.min(precision_macro),np.min(recall_macro),np.min(accuracy)])))
    myyticks = [i for i in range(0,100,5)]
    myxticks = [i for i in range(1,31,1)]
    plt.yticks(myyticks)
    #plt.xticks(myxticks)
    plt.plot(times,f1_macro,label='f1_macro')
    #plt.plot(times,f1_micro,label='f1_micro')
    plt.plot(times,precision_macro,label='precision_macro')
    #plt.plot(times,precision_micro,label='precision_micro')
    plt.plot(times,recall_macro,label='recall_macro')
    #plt.plot(times,recall_micro,label='recall_micro')
    plt.plot(times,accuracy,label='accuracy')
    plt.legend()
    plt.grid()

    pd.DataFrame(outs).to_csv(newh+'ks_30times.csv',header=0,index=False)


    #return f1s
    plt.savefig(newh+'ks_overall.png')
    plt.clf()
    # print (type(rpt).__name__)
    #plot_importance(clf)
    #plt.show()
    return [f1s,pd.DataFrame(outs),probs]


def testingset_basedon_features(n=4):
    train_path = 'C:/Zhaoyi_Fan/Dataset/Combo/training/features/'
    test_path = 'C:/Zhaoyi_Fan/Dataset/Combo/testing/features/'
    dest =
        'C:/Zhaoyi_Fan/Dataset/Combo/testing/results/'+str(n)+'_graphs/XGB/'
    if not os.path.exists(dest):
        os.makedirs(dest)
        print (dest,'created')

    newh = dest
    users = os.listdir(train_path)

    #frequently used keys: k_keys
    f_keys = ['Space', 'Backspace', 'KeyI', 'KeyA', 'KeyN', 'KeyE',
        'KeyO', 'KeyU', 'KeyH', 'Enter', 'KeyG', 'KeyS', 'KeyC', 'KeyD',
        'KeyT', 'KeyL', 'KeyR', 'KeyY', 'KeyM', 'ShiftLeft', 'Digit1',
        'KeyB', 'Digit0', 'KeyZ', 'Digit2', 'KeyJ', 'KeyW', 'KeyX',
        'KeyP', 'KeyF', 'KeyV', 'Digit3', 'KeyK', 'Period', 'Digit9',
        'KeyQ', 'Digit4', 'Digit6', 'Digit7', 'Equal', 'Comma', 'Minus']
    x_train=[]
    x_test=[]
    y_train=[]
    y_test=[]
```

```
unwanted=[]
for user in users[:]:
   #training set:
   file = train_path+user+'/'+user+'_keystroke_train_features.csv'
   with open(file,'r',encoding='UTF-8') as f:
      df = np.array(pd.DataFrame(csv.reader(f)).iloc[:,:-1])
      data = np.array(df,dtype=np.float64)
   for i in data:
      x_train.append(i)
      y_train.append(user)

   #testing set:
   file = test_path+user+'/'+user+'_keystroke_test_features.csv'
   with open(file,'r',encoding='UTF-8') as f:
      df = np.array(pd.DataFrame(csv.reader(f)).iloc[:,:-1])
      data = np.array(df,dtype=np.float64)
   for i in data:
      x_test.append(i)
      y_test.append(user)

print ('Training Data extraction completed',len(y_train),'samples
    involved')
print ('Testing Data extraction completed',len(y_test),'samples
    involved')
labelencod = preprocessing.LabelEncoder().fit(y_train)
x_train = np.delete(x_train,unwanted,axis=1)
x_test = np.delete(x_test,unwanted,axis=1)

y_train = labelencod.transform(y_train)
y_test = labelencod.transform(y_test)



pred_prob=[]
featurescore=[]

mychanges = []
theweights = []

try:
   print ('XGB fitting started')
   #cross_val = StratifiedKFold(n_splits=cvs)
   clf =
       XGBClassifier(objective='multi:softprob',use_label_encoder=False,
                eval_metric='mlogloss',tree_method='gpu_hist',
                learning_rate=0.3,n_estimators=600,
                max_depth = 5,min_child_weight=5,
                )
   #clf = lgb.LGBMClassifier()
   features_impts = []
```

```
    myweights,mychanges = myweight(y_train)
    print (myweights,mychanges)
    class_weights =
        class_weight.compute_sample_weight(myweights,y_train)
    print ('XGB fitting started',len(y_train), 'samples involved')
    clf.fit(x_train,y_train,sample_weight=class_weights)
    print ('XGB fitting completed')
    print ('XGB predicting started')
    pre_temp = clf.predict_proba(x_test)
    print ('XGB predicting completed')
    featurescore = clf.feature_importances_.tolist()

    pred_prob = np.array(pre_temp)

    labelset = np.unique(y_test)
    names = labelencod.inverse_transform(labelset)


except Exception as e:
    print (e)
    #print (folder)
    return 'bad'

# myweights_total,mychanges=myweight(y)
# theweights.append(myweights_total)
# head = np.array(theweights[0].keys())
# print (head)
# #heads = labelencod.inverse_transform(head).tolist()
heads = names.tolist()
# weights=pd.DataFrame(theweights)
# weights = np.array(weights).tolist()
# weights.insert(0,heads)
# weights=pd.DataFrame(weights)

dic = {} #for computing the overall result
dic2 = {} #for computing each user's performance
#print (list(labelset))
for i in list(labelset):
    dic[i]=[]
    #dic2[i]=[]
for i in range(len(y_test)):
    dic[y_test[i]].append(pred_prob[i])
pred_probtemp = []
y_testtemp = []
for i in dic:
    #ypred=[]
    #ytrue=[]
    for j in dic[i]:
        pred_probtemp.append(j)
        y_testtemp.append(i)
        #ypred.append(j.tolist().index(j.max()))
        #ytrue.append(i)
    #dic2[names[i]]=[ypred,ytrue]

#everyone=[]


pred_prob = pred_probtemp[:]
y_test = y_testtemp[:]

probs =
    pd.concat([pd.DataFrame(pred_prob),pd.DataFrame(y_test)],axis=1)
probs.to_csv(dest+'ks_test_probabilities_3.csv',header=0,index=False)


outs = []
f1_micro=[]
f1_macro=[]

precision_micro=[]
precision_macro=[]

recall_micro=[]
recall_macro=[]
times = []

accuracy = []
rpt = []
f1s = []

for i in range(1,32,1):
    y_pred = []
    y_true = []
    for j in range(len(y_test)-i+1):
        if y_test[j]==y_test[j+i-1]:
            #y_pred_byprob=sum(np.array(pred_prob[j:j+i]),axis=0)
            y_pred_byprob=sum(np.array(pred_prob[j:j+i]),axis=0)
            y_pred.append(y_pred_byprob.tolist().index(y_pred_byprob.max()))
            y_true.append(y_test[j])
    #y_true = labelencod.inverse_transform(y_true)
    #y_pred = labelencod.inverse_transform(y_pred)

    outs.append([
            #'%.2f'%(f1_score(y_true,y_pred,average='micro')*100),
            float('%.3f'%(f1_score(y_true,y_pred,average='macro',zero_division=0))),
            float('%.3f'%(f1_score(y_true,y_pred,average='weighted',zero_division=0))),
            '-',
            #'%.2f'%(precision_score(y_true,y_pred,average='micro')*100),
            float('%.3f'%(precision_score(y_true,y_pred,average='macro',zero_division=0))),
            float('%.3f'%(precision_score(y_true,y_pred,average='weighted',zero_division=0))),
            '-',
            #'%.2f'%(recall_score(y_true,y_pred,average='micro')*100),
            float('%.3f'%(recall_score(y_true,y_pred,average='macro',zero_division=0))),
            float('%.3f'%(recall_score(y_true,y_pred,average='weighted',zero_division=0))),
            '-',
            float('%.3f'%(accuracy_score(y_true,y_pred))),
            ])
    f1s.append(f1_score(y_true,y_pred,average='macro',zero_division=0)*100)
    f1_micro.append(f1_score(y_true,y_pred,average='micro',zero_division=0)*100)
    f1_macro.append(f1_score(y_true,y_pred,average='macro',zero_division=0)*100)
    precision_micro.append(precision_score(y_true,y_pred,average='micro',zero_division=0)*100)
    precision_macro.append(precision_score(y_true,y_pred,average='macro',zero_division=0)*100)
    recall_micro.append(recall_score(y_true,y_pred,average='micro',zero_division=0)*100)
    recall_macro.append(recall_score(y_true,y_pred,average='macro',zero_division=0)*100)
    accuracy.append(accuracy_score(y_true,y_pred)*100)
    times.append(i)

    if i==1:
        if not os.path.exists(newh+str(i)+'/'):
            os.makedirs(newh+str(i)+'/')
        rpt = classification_report(y_true,y_pred,output_dict=True)
        #report=classification_report(y_true,y_pred,output_dict=True)
        pd.DataFrame(rpt).T.to_csv(newh+str(i)+'/'+str(i)+'ks_test_classification_report.csv',header=(
        print (classification_report(y_true,y_pred))
        print (labelencod.inverse_transform(clf.classes_))

    if (i+4)/5==int((i+4)/5):
        if not os.path.exists(newh+str(i)+'/'):
            os.makedirs(newh+str(i)+'/')
        cm_old = confusion_matrix(y_true, y_pred)
        cm = np.array(pd.DataFrame(cm_old)).tolist()
        cm_=[]
        for k in cm:
            temp=[]
            for j in k:
                temp.append(float('%.1f'%(100*j/sum(k))))
            cm_.append(temp)
        cm =mat(cm_.copy())

        report = classification_report(y_true,y_pred,output_dict=True)
        #report=classification_report(y_true,y_pred,output_dict=True)
        pd.DataFrame(report).T.to_csv(newh+str(i)+'/'+str(i)+'ks_test__classification_report.csv',head

        plt.figure(1,figsize=(10.24,8))
        ax1 = plt.axes()
        disp2 = ConfusionMatrixDisplay(confusion_matrix=cm)
        disp2.plot(cmap=plt.cm.Blues,include_values=False,ax=ax1)
        plt.savefig(newh+str(i)+'/'+str(i)+'ks_test__cm.png')
        plt.clf()
```

```
          plt.figure(3,figsize=(10.24,8))
          ax3 = plt.axes()
          disp3 =
              ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=heads)
          disp3.plot(cmap=plt.cm.Blues,include_values=True,ax=ax3)
          plt.savefig(newh+str(i)+'/'+str(i)+'ks_test__cm_newnew.png')
          plt.clf()

          plt.figure(4,figsize=(10.24,8))
          ax4 = plt.axes()
          disp4 =
              ConfusionMatrixDisplay(confusion_matrix=cm_old,display_labels=heads)
          disp4.plot(cmap=plt.cm.Blues,include_values=True,ax=ax4)
          plt.savefig(newh+str(i)+'/'+str(i)+'ks_test__cm_newnewnew.png')
          plt.clf()

          #np.set_printoptions(precision=2)
          cm_normalized = confusion_matrix(y_true,
              y_pred,normalize='true')
          cm_n = mat(np.round(cm_normalized,3))
          plt.figure(5,figsize=(10.24,8))
          ax5 = plt.axes()
          disp5 =
              ConfusionMatrixDisplay(confusion_matrix=cm_n,display_labels=heads)
          disp5.plot(cmap=plt.cm.Blues,include_values=True,ax=ax5)
          plt.savefig(newh+str(i)+'/'+str(i)+'ks_test__cm_newnewnew2.png')
          plt.clf()

          #np.set_printoptions(precision=2)
          cm_normalized2 = confusion_matrix(y_true,
              y_pred,normalize='pred')
          cm_n = mat(np.round(cm_normalized2,3))
          plt.figure(6,figsize=(10.24,8))
          ax6 = plt.axes()
          disp6 =
              ConfusionMatrixDisplay(confusion_matrix=cm_n,display_labels=heads)
          disp6.plot(cmap=plt.cm.Blues,include_values=True,ax=ax6)
          plt.savefig(newh+str(i)+'/'+str(i)+'ks_test__cm_newnewnew3.png')
          plt.clf()

     print (pd.DataFrame(outs))


     plt.figure(33,figsize=(10.24,8))
     ax2 = plt.axes()
     plot_importance(clf,ax=ax2,importance_type='gain')
     plt.savefig(newh+'ks_test_FI.png')
     plt.clf()
```

```
print (featurescore)

# plt.show()

plt.figure(34,figsize=(10.24,8))
#print (f1_macro,np.min(f1_macro))
#print
    (int(np.min([np.min(f1_macro),np.min(precision_macro),np.min(recall_macro),np.min(accuracy)])))
myyticks = [i for i in range(0,100,5)]
myxticks = [i for i in range(1,31,1)]
plt.yticks(myyticks)
#plt.xticks(myxticks)
plt.plot(times,f1_macro,label='f1_macro')
#plt.plot(times,f1_micro,label='f1_micro')
plt.plot(times,precision_macro,label='precision_macro')
#plt.plot(times,precision_micro,label='precision_micro')
plt.plot(times,recall_macro,label='recall_macro')
#plt.plot(times,recall_micro,label='recall_micro')
plt.plot(times,accuracy,label='accuracy')
plt.legend()
plt.grid()

pd.DataFrame(outs).to_csv(newh+'ks_test_30times.csv',header=0,index=False)


#return f1s
plt.savefig(newh+'ks_test_overall.png')
plt.clf()

def n_graph_1to30(n=4):

  # for n_graphs in range(2,16,1):
  #  f1_scores = train_n_testvesion(n_graphs)[0]
  #
      pd.DataFrame(f1_scores).to_csv('C:/Zhaoyi_Fan/Dataset/Combo/training/results/Keystroke/'+str(n)
  #
      pd.DataFrame(f1_scores).to_csv('C:/Zhaoyi_Fan/Dataset/Combo/training/results/Keystroke/n_graphs

  path = 'C:/Zhaoyi_Fan/Dataset/Combo/training/results/Keystroke/'
  plt.figure(figsize=(10,6))
  for i in range(2,16,1):

    file = path+str(i)+'_graphs/XGB/ks_30times.csv'
    with open(file,'r',encoding='UTF-8') as f:
      df =
          np.array(pd.DataFrame(csv.reader(f)).iloc[:,1],dtype=np.float64).tolist()
      plt.plot(np.arange(1,32,1),df,label=str(i)+'_graphs')

  plt.xlabel('Samples number')
  plt.ylabel('F1-score')
```

```
    plt.title('Performance on various n-graphs and samples sizes')
    plt.legend()
    plt.grid(True)
    plt.savefig(figure_path1+'n_graphs_comparison.png')
    plt.savefig(figure_path2+'n_graphs_comparison.png')
    plt.show()

def get_testingset_size():
    path = 'C:/Zhaoyi_Fan/Dataset/Combo/testing/samples/'
    users = os.listdir(path)
    total =0
    for user in users:
        with
            open(path+user+'/'+user+'_keystroke_test.csv','r',encoding='UTF-8')
            as f:
           data = np.array(pd.DataFrame(csv.reader(f)))
           print (user,len(data))
           total += len(data)
    print ('total',total)


def extractfeatures_keystroke_example():
    path = 'C:/Zhaoyi_Fan/Dataset/'
    dest = path
    if not os.path.exists(dest):
        os.makedirs(dest)
        print (dest,'created')

    file = path+'keystroke_example.csv'
    name = 'keystroke_example'

    #frequently used keys: k_keys
    f_keys = ['Space', 'Backspace', 'KeyI', 'KeyA', 'KeyN', 'KeyE',
        'KeyO', 'KeyU', 'KeyH', 'Enter', 'KeyG', 'KeyS', 'KeyC', 'KeyD',
        'KeyT', 'KeyL', 'KeyR', 'KeyY', 'KeyM', 'ShiftLeft', 'Digit1',
        'KeyB', 'Digit0', 'KeyZ', 'Digit2', 'KeyJ', 'KeyW', 'KeyX',
        'KeyP', 'KeyF', 'KeyV', 'Digit3', 'KeyK', 'Period', 'Digit9',
        'KeyQ', 'Digit4', 'Digit6', 'Digit7', 'Equal', 'Comma', 'Minus']
    x=[]
    y=[]
    X=[]
    Y=[]

    with open(file,'r',encoding='UTF-8') as f:
        data = np.array(pd.DataFrame(csv.reader(f))).tolist()[1:]
    out = []
    temp=[]
    the_data= []
    for i in data[1:]:
```

```
   if len(temp) == 0:
      temp.append(i)
   else:
      d=i[3]
      if int(i[2])-int(i[1])<10000 and int(i[2])-int(i[1])>0 and
           d!='' and d!='ArrowRight' and d!='ArrowLeft' and
           d!='ArrowDown' and d!='ArrowUp' and d!='keycode' and
           'Volume' not in d:

         if int(i[1])-int(temp[-1][1])<1000:
            temp.append(i)
         else:
            out.append(temp)
            X.append(temp)
            Y.append(name)
            temp=[]
            temp.append(i)
n=4
for d,name in zip(X,Y):
   if len(d)>n-1:
      for i in range(len(d)-n+1):
         boolean = True
         for boo in range(n-1):
            if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
               boolean=False
         if boolean:
            durations = []
            DD = []
            UU = []
            DU = []
            UD = []

            caps=[]
            shifts=[]

            for j in range(n):
               durations.append(int(d[i+j][2])-int(d[i+j][1]))
               caps.append(tobi(d[i+j][-1]))
               shifts.append(tobi(d[i+j][-2]))

            combo = list(combinations(np.arange(n),2))
            for c in combo:
               DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
               UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
               DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
               UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

            c = durations+DD+UU+DU+UD+caps+shifts

            my_feature=c[:]
            the_data.append(my_feature+[name])
    pd.DataFrame(the_data).to_csv(dest+'keystroke_example_features.csv',header=0,index=False)

    print ('data extraction completed',len(the_data),'samples involved')



if __name__=='__main__':

    #extractfeatures_keystroke_example() #extract keystroke features from
        raw data

    testingset_basedon_features() #get the classicication result on
        testing set


    #extractfeatures_keystroke()
    #keystroke_raw_size()
    #type_of_keycode()
    #Occurrence_of_frequently_used_keys_for_each_user()
    #ngraphs_check(i)
    #segments()
    #classifier_select()
    #hyper_parameters() #[0.3,600,5,5] as the best parameters
    #train_n_testvesion()

    #n_graph_1to30()
    #get_testingset_size()
```

D.3.2 Python scripts for analysing mouse dynamics

List D.3.2 lists the Python scripts for mouse dynamics analysis used in Chapter 7.

```
import numpy as np
import sys
import csv
import pandas as pd
import math
from numpy import *
from time import time
import datetime
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn import svm
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.multiclass import OneVsOneClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
import random
import time
import sklearn
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import KFold,StratifiedKFold
from sklearn.metrics import make_scorer
from sklearn import linear_model
from sklearn.metrics import r2_score
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import f1_score,precision_score,recall_score
from sklearn.metrics import accuracy_score
from sklearn import tree
from xgboost import XGBClassifier,plot_importance,DMatrix,cv
from xgboost import XGBRegressor
from sklearn.ensemble import VotingClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
from sklearn.metrics import classification_report
from collections import Counter
import os
import string
from sklearn.utils import class_weight
from itertools import combinations


figure_path1 = 'C:/Zhaoyi_Fan/Figures/'
figure_path2 = 'C:/Zhaoyi_Fan/Latex/Figures/'


pd.set_option('display.max_rows',500)
pd.set_option('display.max_columns',500)
pd.set_option('display.width',1000)
pd.set_option('max_colwidth',1000)
np.set_printoptions(suppress=True)

f_names = ['points','gap','scurlpp','backward',
        'time','timepp','s/strai','widthmax_','width_min',
        'width-mean','width-std','vstrai','vcurl',
        'acc_mean','acc_std','angle_mean','angle_std',
        'squ/strai','squ/scurl','durantion','distance',
        'scurl','sstrai','acc_max','acc_min',
        'angle_min','angle_max',
        'acc_ep_mean','acc_ep_std','acc_ep_min','acc_ep_max',
        'v_c_ep_mean','v_c_ep_std','v_c_ep_min','v_c_ep_max',
        'ac_c_ep_mean','ac_c_ep_std','ac_c_ep_min','ac_c_ep_max',
        'v_seg_mean','v_seg_std','v_seg_min','v_seg_max',
        'width/sstrai','width/scurl','width/time',
        'b_relative','b_both',
        'sin','cos']

def myweight(y=arange(19)):
   myweights = {}
   for j in np.unique(y):
      myweights[j]=len(y)/(y.tolist().count(j)*len(np.unique(y)))
   try:
      for i in mychanges:
         myweights[i[0]]=myweights[i[0]]*i[1]
   except:
      mychanges=[]
   return myweights,mychanges

class Mouse():
   def __init__(self, file):
      def re(data):
         temp1 = []
         temp2 = []
         for i in data:
            a =
                (int(i[1][1])**2+int(i[1][2])**2)**0.5-(int(i[0][1])**2+int(i[0][2])**2)**0.5
            b = int(i[1][3])-int(i[0][3])
            if a<1 and b<2000:
               temp1.append(i)
               temp2.append(int(i[1][-1])-int(i[0][-1]))
         #print (temp2)
         me = np.mean(temp2)
         if len(temp2)!=0:
            me = np.mean(temp2)
            std = np.std(temp2,ddof=1)
         if len(temp2)==0:
            me = 0
            std = 0

         for i in temp1:
            if int(i[1][-1])-int(i[0][-1])-me>3*std:
               temp1.remove(i)
         return temp1

      try:
         with open(file,'r') as f:
            data = np.array(pd.DataFrame(csv.reader(f)))
      except:
         data = file

      content = []
      leftclick = []
      rightclick = []
      cmove=[]
      cleft=[]
      cright=[]
      cmiddle=[]
      cscroll=[]
      splt = []
      stemp = []
      lctemp = []
      rctemp=[]
      count1 = 0 #number of '1'--left click down
      count3 = 0

      for line in data:
         if line[0]=='TYPE':
            continue
         line =  [int(j) for j in line]
         if line[0]==0 or line[0]==1 or line[0]==2:  #click down up,
             move
            stemp.append(line)
            lctemp.append(line)
            if line[0]==1: #click down
               lctemp=[]
               lctemp.append(line)
               count1=count1+1
            if line[0]==2: #click up
               if count1==1:
                  leftclick.append(lctemp)
                  splt.append(stemp)
                  stemp=[]
                  lctemp=[]
                  count1=0
               else:
                  stemp=[]
                  lctemp=[]
                  count1=0

         if line[0] != 8:
            content.append(line)
         if line[0]==0:
            cmove.append(line)

         if line[0]==0 or line[0]==3 or line[0]==4:
            rctemp.append(line)
            if line[0]==3:
               rctemp=[]
               rctemp.append(line)
               count3=count3+1
            if line[0]==4:
               if count3==1:
                  rightclick.append(rctemp)
                  rctemp=[]
                  count3=0
               else:
                  rctemp=[]
                  count3=0

         if line[0]==1 or line[0]==2:
            if len(cleft)!=0 and line[0]==cleft[-1][0]:
               cleft.pop()
               cleft.append(line)
            else:
               cleft.append(line)
         if line[0]==3 or line[0]==4:
            if len(cright)!=0 and line[0]==cright[-1][0]:
               cright.pop()
               cright.append(line)
            else:
               cright.append(line)
         if line[0]==5 or line[0]==6:
            if len(cmiddle)!=0 and line[0]==cmiddle[-1][0]:
               cmiddle.pop()
               cmiddle.append(line)
            else:
               cmiddle.append(line)
```

```
            if line[0]==7:
                cscroll.append(line)
         self.splt=splt
         self.leftclick = leftclick
         self.rightclick = rightclick

         L=[]
         for i in range(0,len(cleft)-1,2):
            temp = [cleft[i],cleft[i+1]]
            L.append(temp)
         self.left = re(L)

         R=[]
         for i in range(0,len(cright)-1,2):
            temp = [cright[i],cright[i+1]]
            R.append(temp)
         self.right = re(R)

         M=[]
         for i in range(0,len(cmiddle)-1,2):
            temp = [cmiddle[i],cmiddle[i+1]]
            M.append(temp)
         self.middle = M
         self.rawdata = content
         self.move = cmove
         #self.left = cleft
         #self.right = cright
         #self.middle = cmiddle
         self.scroll = cscroll

   def fst(self):
      self.tem = self.rawdata[0]
      print (self.rawdata[:1000],self.tem)
      a = self.tem
      return a

def get_onemove_data_v1(da,ratio):
   datago=[]
   samples= []
   labels = []
   databyperson =[]
   p = da
   samplesbyperson = []
   labelsbyperson = []
   testlist=[]
   features = [] #features between neiboughs [interval,distance]
   spl = p[0].splt[:]
   co=0

   for point in spl:
```

```
duration = 0
distance = 0
i = point.copy()
x=[]
y=[]

i.reverse()

leftclick = []
data_sub = []
itemp = i[:]
itemp_ = itemp[:]
for j in itemp:
   if itemp[0][3]-j[3]>3000:
      itemp_ = itemp[:itemp.index(j)]
      break
itemp = itemp_[:]

for j in itemp:
   if j[0]==1:
      leftclick = itemp[:itemp.index(j)+1]
      i = itemp[itemp.index(j)+1:]
      break

if len(i)==0:
   continue

i_ = [i[0]]

j=1
while j<len(i):
   dis = (i_[-1][1]-i[j][1])**2+(i_[-1][2]-i[j][2])**2
   if dis == 0:
      for k in range(j,len(i),1):
         dis2 = (i_[-1][1]-i[k][1])**2+(i_[-1][2]-i[k][2])**2
         if dis2!=0:
            i_[-1] =
                 [i_[-1][0],i_[-1][1],i_[-1][2],(i_[-1][3]+i[k-1][3])/2]
            j=k
            #print (j)
            break
         elif k == len(i)-1:
            i_[-1] =
                 [i_[-1][0],i_[-1][1],i_[-1][2],(i_[-1][3]+i[k][3])/2]
            j+=1
            break
   else:
      i_.append(i[j])
      j+=1
```

```
i = i_

i.reverse()

if len(i)<6:
   continue

moveth=[]
tempi=i[:]
for j in range(len(tempi)-1):
   moveth.append(tempi[j+1][-1]-tempi[j][-1])
try:
   Q3 = np.percentile(moveth,75)
   Q1 = np.percentile(moveth,25)
   up = 2.5*Q3-1.5*Q1
   down = 2.5*Q1-1.5*Q3
except Exception as e:
   print (e,tempi)
   break
i.reverse()
itemp=i[:]
for j in range(len(itemp)-1):
   if ratio<3:
      if itemp[j][3]-itemp[j+1][3]>up*ratio or
          itemp[j][3]-itemp[j+1][3]<0:
         i = itemp[:j+1]
         break
      elif itemp[0][3]-itemp[j][3]>2000:
         i = itemp[:j]
         break
   elif ratio>=3:
      if up<100:
         if itemp[j][3]-itemp[j+1][3]>up*ratio or
             itemp[j][3]-itemp[j+1][3]<0:
            i = itemp[:j+1]
            break
         elif itemp[0][3]-itemp[j][3]>2000:
            i = itemp[:j]
            break
      elif up>=100:
         if itemp[j][3]-itemp[j+1][3]>up*2 or
             itemp[j][3]-itemp[j+1][3]<0:
            i = itemp[:j+1]
            break
         elif itemp[0][3]-itemp[j][3]>2000:
            i = itemp[:j]
            break

for j in range(len(i)-1):
   dis = ((i[j][1]-i[j+1][1])**2+(i[j][2]-i[j+1][2])**2)**0.5
   testlist.append([i[j][3]-i[j+1][3],dis])

try:
   if leftclick[0][0]==2 and leftclick[-1][0]==1:
      duration = leftclick[0][3]-leftclick[-1][3]
      distance = ((leftclick[0][1]-leftclick[-1][1])**2 +
           (leftclick[0][2]-leftclick[-1][2])**2)**0.5
except:
   continue

if duration>400:
   continue

if len(i)<6:
   continue

itemp = i[:]
distemp=[]

for j in range(len(itemp)):
   distemp.append(((itemp[j][1]-itemp[0][1])**2+(itemp[j][2]-itemp[0][2])**2)**0.5)

maxdist = distemp.index(np.max(distemp))
i = i[0:maxdist+1]

if len(i)<6:
   continue

i.reverse()


x=i[-1][1]-i[0][1] #x from start to end
y=i[-1][2]-i[0][2] #y from start to end

try:
   sin_coordinate = round(x/((x**2+y**2)**0.5),8)
   cos_coordinate = round(y/((x**2+y**2)**0.5),8)
except Exception as e:
   sin_coordinate = 0
   cos_coordinate = 0
   print (e)
   continue
#get square, width, speed,
squ = 0
vcurl = 0
vstrai = 0
s=0
v_=0
v0=0
sstrai = 0
```

```
scurl = 0
dlist = []
width = 0
time = 0
acc = []
angles=[]
angle_=0
v_each_seg= []
acc_eachpoint=[]
backward_relative = 0
backward_both = 0
backward_count = 0
isForward = True

if (((i[0][1]-i[-1][1] )**2+(i[0][2] -i[-1][2])**2)**0.5)==0:
   continue

for a in range(len(i)):
   L_=0
   d_=0
   t_=0

   maxd = 0
   mind = 0

   L = 0

   Xs=i[0][1]
   Ys=i[0][2]
   X0=i[a][1]
   Y0=i[a][2]
   X_=0
   Y_=0
   Xe=i[-1][1]
   Ye=i[-1][2]
   t0=i[a][3]
   try:
      d0 =
          abs(((X0-Xs)*(Ys-Ye)-(Xs-Xe)*(Y0-Ys))/(((Xs-Xe)**2+(Ys-Ye)**2)**0.5))
   except:
      print (i)
   d0 =
       abs(((X0-Xs)*(Ys-Ye)-(Xs-Xe)*(Y0-Ys))/(((Xs-Xe)**2+(Ys-Ye)**2)**0.5))
   L0 = (abs((Xs-X0)**2 + (Ys-Y0)**2 - d0**2))**0.5
   if (Xs-X0)**2 + (Ys-Y0)**2 >= d0**2:
      L0 = ((Xs-X0)**2 + (Ys-Y0)**2 - d0**2)**0.5
   elif (Xs-X0)**2 + (Ys-Y0)**2 < d0**2:
      L0 = -((abs((Xs-X0)**2 + (Ys-Y0)**2 - d0**2))**0.5)

   if a==0:
      L_=0
      d_=0
      L0=0
      L = L0-L_
      v_=0
      v0=0
      t_=0
      s0=0
      d0=0
      sq = (d_+d0)*L/2
      acc=[]
      dlist.append(d0)

   else:
      X_ = i[a-1][1]
      Y_ = i[a-1][2]
      t_ = i[a-1][3]
      d_ =
          abs(((X_-Xs)*(Ys-Ye)-(Xs-Xe)*(Y_-Ys))/(((Xs-Xe)**2+(Ys-Ye)**2)**0.5))
      L_ = (abs((Xs-X_)**2 + (Ys-Y_)**2 - d_**2)**0.5)
      if ((Xs-X_)**2 + (Ys-Y_)**2) >= d_**2:
          L_ = ((Xs-X_)**2 + (Ys-Y_)**2 - d_**2)**0.5
      elif ((Xs-X_)**2 + (Ys-Y_)**2)< d_**2:
          L_ = -((abs((Xs-X_)**2 + (Ys-Y_)**2 - d_**2)**0.5))
      L = L0-L_
      try:
          v0 = (((X0-X_)**2 + (Y0-Y_)**2)**0.5)/(t0-t_)
          v_each_seg.append(v0)
      except:
          print (i)
          print (p[1])
      acc_=(v0-v_)/(t0-t_)
      acc.append(acc_)
      v_=v0

      s0 = ((X0-X_)**2 + (Y0-Y_)**2)**0.5
      s = s+s0
      if(type(L_).__name__=='complex'):
          print (L_,L0)
          continue

      sq = abs((d_+d0)*L/2)

      dlist.append(((X0-Xs)*(Ys-Ye)-(Xs-Xe)*(Y0-Ys))/(((Xs-Xe)**2+(Ys-Ye)**2)**0.5))

      #angles
      if a<len(i)-1:

          X_N = i[a+1][1]
          Y_N = i[a+1][2]
          a1 = [X_-X0,Y_-Y0]
          a2 = [X0-X_N,Y0-Y_N]
          angle_ = math.acos((a1[0]*a2[0]+a1[1]*a2[1]) /
              (((a1[0]**2+a1[1]**2)*(a2[0]**2+a2[1]**2))**0.5))
          if angle_/np.pi>0.5:
             backward_relative=backward_relative+1
          angles.append(angle_)

          b1 = [X_N-X0,Y_N-Y0]
          b2 = [i[-1][1]-i[0][1],i[-1][2]-i[0][2]]
          angle2 = math.acos((b1[0]*b2[0]+b1[1]*b2[1]) /
              (((b1[0]**2+b1[1]**2)*(b2[0]**2+b2[1]**2))**0.5))

          if isForward:
             if angle2/np.pi>0.5:
                backward_count=backward_count+1
                if angle_/np.pi >0.5:
                   backward_both = backward_both+1
                isForward = False
          elif not isForward:
             if angle2/np.pi<0.5:
                backward_count=backward_count+1
                if angle_/np.pi>0.5:
                   backward_both = backward_both+1
                isForward = True

 squ = squ + sq

maxd = max(dlist)
mind = min(dlist)

if mind<0:
 width = maxd-mind
else:
 width=maxd

width_sum = np.sum(dlist)
time = i[-1][3]-i[0][3]
scurl = s
vcurl = s/time
sstrai = ((i[-1][1]-i[0][1])**2+(i[-1][2]-i[0][2])**2)**0.5

if sstrai==0: #if the path is a closed circle,delete this data
 continue

acc_eachpoint=np.diff(np.array(v_each_seg)).tolist()

vchaos_eachpoint = []
accchaos_eachpoint = []
for j in range(len(acc_eachpoint)):
 an = angles[j]
 ac = acc_eachpoint[j]
 ac2 = acc[j+1]-acc[j]

 temp = 0
 if an>0 and ac>0:
     temp = (an+1)*ac + (ac+1)*an
 elif an>0 and ac<0:
     temp = (an+1)*ac + (ac-1)*an
 elif an<0 and ac>0:
     temp = (an-1)*ac + (ac+1)*an
 elif an<0 and ac<0:
     temp = (an-1)*ac + (ac-1)*an
 elif an==0 and ac!=0:
     temp = ac
 elif an!=0 and ac==0:
     temp = an
 vchaos_eachpoint.append(temp)

 temp2=0
 if an>0 and ac2>0:
     temp2 = (an+1)*ac2 + (ac2+1)*an
 elif an>0 and ac2<0:
     temp2 = (an+1)*ac2 + (ac2-1)*an
 elif an<0 and ac2>0:
     temp2 = (an-1)*ac2 + (ac2+1)*an
 elif an<0 and ac2<0:
     temp2 = (an-1)*ac2 + (ac2-1)*an
 elif an==0 and ac2!=0:
     temp2 = ac2
 elif an!=0 and ac2==0:
     temp2 = an
 accchaos_eachpoint.append(temp2)

vstrai = sstrai/time
acc_mean = np.mean(acc)
acc_std = np.std(acc,ddof=1)
acc_min = np.min(acc)
acc_max = np.max(acc)

angle_mean = np.mean(angles)
angle_std = np.std(angles,ddof=1)
angle_min = np.min(angles)
angles_max = np.max(angles)

acc_eachpoint_mean = np.mean(acc_eachpoint)
acc_eachpoint_std = np.std(acc_eachpoint,ddof=1)
acc_eachpoint_min = np.min(acc_eachpoint)
acc_eachpoint_max = np.max(acc_eachpoint)
```

```
vchaos_eachpoint_mean = np.mean(vchaos_eachpoint)
vchaos_eachpoint_std = np.std(vchaos_eachpoint,ddof=1)
vchaos_eachpoint_min = np.min(vchaos_eachpoint)
vchaos_eachpoint_max = np.max(vchaos_eachpoint)

accchaos_eachpoint_mean = np.mean(accchaos_eachpoint)
accchaos_eachpoint_std = np.std(accchaos_eachpoint,ddof=1)
accchaos_eachpoint_min = np.min(accchaos_eachpoint)
accchaos_eachpoint_max = np.max(accchaos_eachpoint)

v_seg_mean = np.mean(v_each_seg)
v_seg_std = np.std(v_each_seg)
v_seg_max = np.max(v_each_seg)
v_seg_min = np.min(v_each_seg)

width_mean = np.mean(dlist)
width_std = np.std(dlist,ddof=1)
click_move_gap = (leftclick[-1][3]-i[-1][3])

#data_sub.append(zone)
data_sub.append(len(i))
data_sub.append(click_move_gap)
data_sub.append(scurl/(len(i)-1))
data_sub.append(backward_count)
#data_sub.append(backward_count/(len(i)-1))
data_sub.append(time)

data_sub.append(time/(len(i)-1))

data_sub.append(s/sstrai) #scurl/sstraight
#data_sub.append(width) #max width
data_sub.append(maxd)
data_sub.append(mind)
data_sub.append(width_mean)
data_sub.append(width_std)

data_sub.append(vstrai) #speed of straight away
data_sub.append(vcurl) #speed of actual move
data_sub.append(acc_mean)
data_sub.append(acc_std)

data_sub.append(angle_mean)
data_sub.append(angle_std)
#data_sub.append(max(angles))
data_sub.append(squ/sstrai)
data_sub.append(squ/scurl)
data_sub.append(duration) #mosue click
data_sub.append(distance) #mosue click
data_sub.append(scurl)
data_sub.append(sstrai)
```

```
      data_sub.append(acc_max)
      data_sub.append(acc_min)
      data_sub.append(angle_min)
      data_sub.append(angles_max)

      data_sub.append(acc_eachpoint_mean)
      data_sub.append(acc_eachpoint_std)
      data_sub.append(acc_eachpoint_min)
      data_sub.append(acc_eachpoint_max)

      data_sub.append(vchaos_eachpoint_mean)
      data_sub.append(vchaos_eachpoint_std)
      data_sub.append(vchaos_eachpoint_min)
      data_sub.append(vchaos_eachpoint_max)

      data_sub.append(accchaos_eachpoint_mean)
      data_sub.append(accchaos_eachpoint_std)
      data_sub.append(accchaos_eachpoint_min)
      data_sub.append(accchaos_eachpoint_max)

      data_sub.append(v_seg_mean)
      data_sub.append(v_seg_std)
      data_sub.append(v_seg_min)
      data_sub.append(v_seg_max)


      data_sub.append(width/sstrai)
      data_sub.append(width/scurl)
      data_sub.append(width/time)

      data_sub.append(backward_relative)
      data_sub.append(backward_both)

      data_sub.append(sin_coordinate)
      data_sub.append(cos_coordinate)

      #data_sub.append(angles_max - angle_min)

      co+=1

      features.append(data_sub)

   username = p[1].split('.')[0]
   for i in features:
      samples.append(i)
      labels.append(username)

   return [samples,labels]

def extractfeatures_mouse():
    path = 'C:/Zhaoyi_Fan/Dataset/Combo/'
    groups = ['training','testing']
    count=0
    for group in groups:
       newpath = path+group+'/'
       for IQR in range(1,6,1):
          if IQR!=3:
             continue
          finalpath = newpath
          users = os.listdir(finalpath+'samples/')
          for user in users:
             newhome = newpath+'features/'+user+'/'
             isExists=os.path.exists(newhome)
             if not isExists:
                os.makedirs(newhome)
                print (newhome,' created')
             files = os.listdir(finalpath+'samples/'+user+'/')
             for f in files:
                label = f.split('.csv')[0]
                if label.split('_')[1]=='keystroke':
                   continue
                file = [Mouse(finalpath+'samples/'+user+'/'+f),label]
                data = get_onemove_data_v1(file,IQR)
                r =
                    pd.concat([pd.DataFrame(data[0]),pd.DataFrame(data[1])],axis=1)
                r.to_csv(newhome+label+'_features.csv',header=0,index=False)


                print (f,'files
                    completed>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>')

def my_check(unwanted=[],cata='no_',cata2='nochange',src='',results=''):
    folder = src
    files = os.listdir(folder)
    x=[]
    y=[]
    for f in files[:]:
       user=f
       file = folder+user+'/'+user+'_mouse_train_features.csv'
       if f == 'user18.csv':
          continue
       with open(file,'r',encoding='UTF-8') as thefile:
          d = csv.reader(thefile)
          d = np.array(pd.DataFrame(d))[:]
          try:
             ss1 = float(d[0][0])
             ss2 = float(d[0][1])
          except:
             continue
          for i in d:
            temp = []
            if float(i[1])<1500 and float(i[3])/(float(i[0])-2)<0.2 and
                float(i[6])<2 and float(i[22])<2000 and float(i[20])<10
                and float(i[-5])/(float(i[0])-2)<0.2: # and
                float(i[3])/(float(i[0])-2)<0.3333:
               for j in i[:-1]:
                  temp.append(float(j))
               x.append(temp)
               y.append(f.split('_')[0])
print ('Data extraction completed',len(y),'samples involved')
try:
   x = np.delete(x,unwanted,axis=1)
   labelencod = preprocessing.LabelEncoder().fit(y)
   y = labelencod.transform(np.array(y))
   x_train,x_test,y_train,y_test =
       train_test_split(x,y,test_size=0.2,stratify=y,random_state=1)

except Exception as e:
   print (e)

pred_prob=[]
featurescore=[]

mychanges = []
theweights = []

try:
   print ('XGB fitting started')
   #cross_val = StratifiedKFold(n_splits=cvs)
   clf =
       XGBClassifier(objective='multi:softprob',use_label_encoder=False,
                 eval_metric='mlogloss',tree_method='gpu_hist',
                 learning_rate=0.3,n_estimators=600,
                 max_depth = 5,min_child_weight=3,
                 )
   features_impts = []

   myweights,mychanges = myweight(y_train)
   class_weights =
       class_weight.compute_sample_weight(myweights,y_train)
   print ('XGB fitting started',len(y_train), 'samples involved')
   clf.fit(x_train,y_train,sample_weight=class_weights)
   print ('XGB fitting completed')
   print ('XGB predicting started')
   pre_temp = clf.predict_proba(x_test)
   print ('XGB predicting completed')
   featurescore = clf.feature_importances_.tolist()

   pred_prob = np.array(pre_temp)
   labelset = np.unique(y_test)
   names = labelencod.inverse_transform(labelset)

except Exception as e:
   print (e)
   print (folder)
   return 'bad'

myweights_total,mychanges=myweight(y)
theweights.append(myweights_total)
head = np.array(theweights[0].keys())
print (head)
heads = names.tolist()
weights=pd.DataFrame(theweights)
weights = np.array(weights).tolist()
weights.insert(0,heads)
weights=pd.DataFrame(weights)

dic = {} #for computing the overall result
dic2 = {} #for computing each user's performance
for i in list(labelset):
   dic[i]=[]
for i in range(len(y_test)):
   dic[y_test[i]].append(pred_prob[i])
pred_probtemp = []
y_testtemp = []
for i in dic:
   for j in dic[i]:
      pred_probtemp.append(j)
      y_testtemp.append(i)
pred_prob = pred_probtemp[:]
y_test = y_testtemp[:]

outs = []
f1_micro=[]
f1_macro=[]

precision_micro=[]
precision_macro=[]

recall_micro=[]
recall_macro=[]
times = []

accuracy = []
rpt = []
for i in range(1,32,1):
   y_pred = []
   y_true = []
   for j in range(len(y_test)-i+1):
```

```
  if y_test[j]==y_test[j+i-1]:
    #y_pred_byprob=sum(np.array(pred_prob[j:j+i]),axis=0)
    y_pred_byprob=sum(np.array(pred_prob[j:j+i]),axis=0)
    y_pred.append(y_pred_byprob.tolist().index(y_pred_byprob.max()))
    y_true.append(y_test[j])
y_true = labelencod.inverse_transform(y_true)
y_pred = labelencod.inverse_transform(y_pred)

outs.append([
    #'%.2f'%(f1_score(y_true,y_pred,average='micro')*100),
    float('%.3f'%(f1_score(y_true,y_pred,average='macro',zero_division=0))),
    float('%.3f'%(f1_score(y_true,y_pred,average='weighted',zero_division=0))),
    '-',
    #'%.2f'%(precision_score(y_true,y_pred,average='micro')*100),
    float('%.3f'%(precision_score(y_true,y_pred,average='macro',zero_division=0))),
    float('%.3f'%(precision_score(y_true,y_pred,average='weighted',zero_division=0))),
    '-',
    #'%.2f'%(recall_score(y_true,y_pred,average='micro')*100),
    float('%.3f'%(recall_score(y_true,y_pred,average='macro',zero_division=0))),
    float('%.3f'%(recall_score(y_true,y_pred,average='weighted',zero_division=0))),
    '-',
    float('%.3f'%(accuracy_score(y_true,y_pred))),
    ])

f1_micro.append(f1_score(y_true,y_pred,average='micro',zero_division=0)*100)
f1_macro.append(f1_score(y_true,y_pred,average='macro',zero_division=0)*100)
precision_micro.append(precision_score(y_true,y_pred,average='micro',zero_division=0)*100)
precision_macro.append(precision_score(y_true,y_pred,average='macro',zero_division=0)*100)
recall_micro.append(recall_score(y_true,y_pred,average='micro',zero_division=0)*100)
recall_macro.append(recall_score(y_true,y_pred,average='macro',zero_division=0)*100)
accuracy.append(accuracy_score(y_true,y_pred)*100)
times.append(i)
newh = results+cata+'/'+cata2+'/'

if i==1:
  if not os.path.exists(newh+str(i)+'/'):
    os.makedirs(newh+str(i)+'/')
  rpt = classification_report(y_true,y_pred,output_dict=True)
  #report=classification_report(y_true,y_pred,output_dict=True)
  pd.DataFrame(rpt).T.to_csv(newh+str(i)+'/'+str(i)+'mouse_classification_report.csv',header=0,i
  print (classification_report(y_true,y_pred))

if (i+4)/5==int((i+4)/5):
  if not os.path.exists(newh+str(i)+'/'):
    os.makedirs(newh+str(i)+'/')
  cm_old = confusion_matrix(y_true, y_pred)

  pd.DataFrame(cm_old).to_csv(newh+str(i)+'/'+str(i)+'mouse_cm.csv',header=heads,index=True)
  cm = np.array(pd.DataFrame(cm_old)).tolist()
  cm_=[]
```

```
for k in cm:
   temp=[]
   for j in k:
      temp.append(float('%.1f'%(100*j/sum(k))))
   cm_.append(temp)
cm =mat(cm_.copy())

report = classification_report(y_true,y_pred,output_dict=True)
#report=classification_report(y_true,y_pred,output_dict=True)
pd.DataFrame(report).T.to_csv(newh+str(i)+'/'+str(i)+'mouse_classification_report.csv',header=


plt.figure(1,figsize=(10.24,8))
ax1 = plt.axes()
disp2 = ConfusionMatrixDisplay(confusion_matrix=cm)
disp2.plot(cmap=plt.cm.Blues,include_values=False,ax=ax1)
plt.savefig(newh+str(i)+'/'+str(i)+'mouse_cm.png')
plt.clf()

plt.figure(3,figsize=(10.24,8))
ax3 = plt.axes()
disp3 =
    ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=heads)
disp3.plot(cmap=plt.cm.Blues,include_values=True,ax=ax3)
plt.savefig(newh+str(i)+'/'+str(i)+'mouse_cm_newnew.png')
plt.clf()

plt.figure(4,figsize=(10.24,8))
ax4 = plt.axes()
disp4 =
    ConfusionMatrixDisplay(confusion_matrix=cm_old,display_labels=heads)
disp4.plot(cmap=plt.cm.Blues,include_values=True,ax=ax4)
plt.savefig(newh+str(i)+'/'+str(i)+'mouse_cm_newnewnew.png')
plt.clf()

#np.set_printoptions(precision=2)
cm_normalized = confusion_matrix(y_true,
    y_pred,normalize='true')
cm_n = mat(np.round(cm_normalized,3))
plt.figure(5,figsize=(10.24,8))
ax5 = plt.axes()
disp5 =
    ConfusionMatrixDisplay(confusion_matrix=cm_n,display_labels=heads)
disp5.plot(cmap=plt.cm.Blues,include_values=True,ax=ax5)
plt.savefig(newh+str(i)+'/'+str(i)+'mouse_cm_newnewnew2.png')
plt.clf()

#np.set_printoptions(precision=2)
cm_normalized2 = confusion_matrix(y_true,
    y_pred,normalize='pred')
```

```
        cm_n = mat(np.round(cm_normalized2,3))
        plt.figure(6,figsize=(10.24,8))
        ax6 = plt.axes()
        disp6 =
            ConfusionMatrixDisplay(confusion_matrix=cm_n,display_labels=heads)
        disp6.plot(cmap=plt.cm.Blues,include_values=True,ax=ax6)
        plt.savefig(newh+str(i)+'/'+str(i)+'mouse_cm_newnewnew3.png')
        plt.clf()

   print (pd.DataFrame(outs))

   out2 = []
   for i,j in enumerate(outs):
      out2.append(['&',j[0],'&',j[3],'&',j[6],'&',j[9],'\\\\'])
   print (pd.DataFrame(out2,index=np.arange(1,32,1)))




   plt.figure(2,figsize=(10.24,8))
   ax2 = plt.axes()
   plot_importance(clf,ax=ax2,importance_type='gain')
   plt.savefig(newh+'mouse_FI.png')
   plt.clf()

   print (featurescore)
   # plt.show()

   plt.figure(3,figsize=(10.24,8))
   myyticks = [i for i in range(0,100,5)]
   myxticks = [i for i in range(1,31,1)]
   plt.yticks(myyticks)
   plt.plot(times,f1_macro,label='f1_macro')
   plt.plot(times,precision_macro,label='precision_macro')
   plt.plot(times,recall_macro,label='recall_macro')
   plt.plot(times,accuracy,label='accuracy')
   plt.legend()
   plt.grid()
   plt.savefig(newh+'mouse_overall.png')
   plt.clf()
   return
       [rpt,outs,pred_prob,y_test,featurescore,np.delete(f_names,unwanted),weights]

def my_train(unwanted=[],src='',results=''):
   myweights,mychanges = myweight()
   cata2='nochange'
   if len(mychanges)>0:
      cata2=''
      for i in mychanges:
         temp = ''
```

```
            for j in i:
                temp = temp+str(j)+'_'
            cata2 = cata2+temp+'_'

    cata='no_'

    try:
        if len(unwanted)==0:
            cata = 'allfeatures'
        else:
            for i in unwanted:
                cata = cata+str(i)+'_'

        finalpath = src
        users = os.listdir(finalpath)

        resultpath = results+cata+'/'+cata2+'/'
        if not os.path.exists(resultpath):
            os.makedirs(resultpath)
            print (resultpath,'created')
        r = my_check(unwanted,cata,cata2,src,results)
        if r=='bad':
            print ('baaaaaaaaaaaaaaaad')
            return
        pd.DataFrame(r[0]).T.to_csv(resultpath+'classification_report.csv',header=0,index=True)
        pd.DataFrame(r[1]).to_csv(resultpath+'30times.csv',header=0,index=False)
        probs = pd.concat([pd.DataFrame(r[2]),pd.DataFrame(r[3])],axis=1)
        probs.to_csv(resultpath+'probabilities.csv',header=0,index=False)
        r[6].to_csv(resultpath+'sample_weight.csv',header=True,index=False)

        fscores=r[4][:]
        sorted_number = np.argsort(fscores)
        fnames = []
        scores = []
        for i in sorted_number:
            fnames.append(r[5][i]+'_'+str(i))
            scores.append(fscores[i])

        plt.figure(4,figsize=(10.24,8))
        plt.barh(fnames,scores)
        plt.grid()
        plt.savefig(resultpath+'myfeatures_importance.png')
        plt.clf()
    except Exception as e:
        print ('error occur in selecting best features')
        print (e)

def go(unwanted=[],src='',results=''):
    if not os.path.exists(results):
        os.makedirs(results)
        print (results,'created')
    my_train(unwanted,src,results)

def extractfeatures_mouse_nIQR():
    path = 'C:/Zhaoyi_Fan/Dataset/Combo/'
    groups = ['training','testing']
    count=0
    for group in groups:
        if group!='testing':
            continue
        newpath = path+group+'/'
        for IQR in range(1,6,1):
            # if IQR!=3:
            #  continue
            finalpath = newpath
            users = os.listdir(finalpath+'samples/')
            for user in users:
                newhome = newpath+'features2/'+str(IQR)+'_IQR/'
                isExists=os.path.exists(newhome)
                if not isExists:
                    os.makedirs(newhome)
                    print (newhome,' created')
                files = os.listdir(finalpath+'samples/'+user+'/')
                for f in files:
                    label = f.split('.csv')[0]
                    if label.split('_')[1]=='keystroke':
                        continue
                    file = [Mouse(finalpath+'samples/'+user+'/'+f),label]
                    data = get_onemove_data_v1(file,IQR)
                    r =
                        pd.concat([pd.DataFrame(data[0]),pd.DataFrame(data[1])],axis=1)
                    r.to_csv(newhome+label+'_'+str(IQR)+'_features.csv',header=0,index=False)


                    print (f,'files
                        completed>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>')

def get_actions_percent():
    path = 'C:/Zhaoyi_Fan/Dataset/Combo/testing/samples/'
    users = os.listdir(path)
    output = []
    col = ['user','orginal segements','1IQR','2IQR','3IQ3','4IQR','5IQR']
    for user in users:
        file = path+user+'/'+user+'_mouse_test.csv'
        mouse = Mouse(file)
        origin = len(mouse.splt)
        out = [user,origin]
        for i in range(5):
            #print (user,len(mouse.splt))
            with
            open('C:/Zhaoyi_Fan/Dataset/Combo/testing/features2/'+str(i+1)+'_IQR/'+user+'_mouse_test_
            as f:
           the_len = len(np.array(pd.DataFrame(csv.reader(f))))
         out.append(the_len)
      output.append(out)
   resultpath = 'C:/Zhaoyi_Fan/Dataset/Combo/testing/'
   if not os.path.exists(resultpath):
      os.makedirs(resultpath)
      print (resultpath,'created')
   pd.DataFrame(output).to_csv(resultpath+str(i+1)+'_IQR_mouse_action_percent_revised1_testing2_.csv',

def get_segments_figure2():
   file =
       'C:/Zhaoyi_Fan/Dataset/Combo/training/5_IQR_mouse_action_percent_revised1_testing_.csv'
   with open(file,'r') as f:
      data = np.array(pd.DataFrame(csv.reader(f)))[1:]

   plt.figure(figsize=(10,6))
   print (len(data))
   for i in range(len(data)):
      plt.subplot(5,4,i+1)
      plt.ylabel('Number of segments')
      user = i
      plt.title('user'+str(user))
      total = int(data[i][1])
      extracted1 = int(data[i][2])
      extracted2 = int(data[i][3])
      extracted3 = int(data[i][4])
      extracted4 = int(data[i][5])
      extracted5 = int(data[i][6])
      y = [total,extracted1,extracted2,extracted3,extracted4,extracted5]
      x = ['total','1 time','2 times','3 times','4 times','5times']
      for j,z in zip([2,3,4,5,6],y[1:]):
         plt.text(j-0.2,z+100,'%.0f'%(100*z/y[0])+'%')
      plt.bar([1,2,3,4,5,6],y,width=0.3,tick_label=x)
   #plt.legend()
   plt.show()

def get_segments_figure():
   file =
       'C:/Zhaoyi_Fan/Dataset/Combo/training/5_IQR_mouse_action_percent_revised1_testing_.csv'
   with open(file,'r') as f:
      df = pd.DataFrame(csv.reader(f))
      users = np.array(df.iloc[1:,0])
      total = np.array(df.iloc[1:,1],dtype = np.float64)
      extracted = np.array(df.iloc[1:,4],dtype = np.float64)
      abandoned = total - extracted

   plt.figure(figsize=(10,6))
```

```
    x = users
    y_ex = extracted
    y_ab = abandoned
    sums = total

    sorted_indices = np.argsort(sums)[::-1]
    users_sorted = np.array(users)[sorted_indices]
    y_ex_sorted = np.array(extracted)[sorted_indices]
    y_ab_sorted = np.array(abandoned)[sorted_indices]
    plt.barh(users_sorted, y_ex_sorted, label='Extracted segments',
         tick_label=users_sorted)
    plt.barh(users_sorted, y_ab_sorted, left=y_ex_sorted,
         label='Abandoned segments')

    plt.xlabel('Number of segments')
    plt.ylabel('User')
    for user, ex, ab in zip(users_sorted, y_ex_sorted, y_ab_sorted):
       if ab != 0:
          percent = (ex/ab)*100
       else:
          percent = 0
       plt.text(ex+ab, user, f"{int(percent)}%", va='center', ha='left')
    plt.tight_layout()
    plt.legend()
    plt.savefig(figure_path1+'Number_of_segments.png')
    plt.savefig(figure_path2+'Number_of_segments.png')
    plt.show()
    ex = np.sum(y_ex)
    ab = np.sum(y_ab)
    total = ex+ab
    print (len(x),total,ex,ex/total)

def classifier_select():
    folder = 'C:/Zhaoyi_Fan/Dataset/Combo/training/features/'
    newh_xgb='C:/Zhaoyi_Fan/Dataset/Combo/training/results/Mouse/XGB/'
    newh_rf =
         'C:/Zhaoyi_Fan/Dataset/Combo/training/results/Mouse/RandomForest/'
    newh_linearSVC =
         'C:/Zhaoyi_Fan/Dataset/Combo/training/results/Mouse/linearSVC/'
    newh_SVClin =
         'C:/Zhaoyi_Fan/Dataset/Combo/training/results/Mouse/SVClin/'
    newh_SVCrbf =
         'C:/Zhaoyi_Fan/Dataset/Combo/training/results/Mouse/SVCrbf/'

    if not os.path.exists(newh_xgb):
       os.makedirs(newh_xgb)
    if not os.path.exists(newh_rf):
       os.makedirs(newh_rf)
    if not os.path.exists(newh_linearSVC):
       os.makedirs(newh_linearSVC)
```

```
if not os.path.exists(newh_SVClin):
   os.makedirs(newh_SVClin)
if not os.path.exists(newh_SVCrbf):
   os.makedirs(newh_SVCrbf)

files = os.listdir(folder)
x=[]
y=[]
for f in files[:3]:
   the_file = folder+f+'/'+f+'_mouse_train_features.csv'
   with open(the_file,'r',encoding='UTF-8') as thefile:
      d = csv.reader(thefile)
      d = np.array(pd.DataFrame(d))[:100]
      try:
         ss1 = float(d[0][0])
         ss2 = float(d[0][1])
      except:
         continue
      #d = np.delete(d,unwanted,axis=1)
      for i in d[:]:
         temp = []
         if float(i[1])<1500 and float(i[3])/(float(i[0])-2)<0.2 and
             float(i[6])<2 and float(i[22])<2000 and float(i[20])<10
             and float(i[-5])/(float(i[0])-2)<0.2: # and
             float(i[3])/(float(i[0])-2)<0.3333:
            for j in i[:-1]:
               temp.append(float(j))
            x.append(temp)
            y.append(i[-1].split('_')[0])
print ('Data extraction completed',len(y),'samples involved')
try:
   x = np.array(x)

   labelencod = preprocessing.LabelEncoder().fit(y)

   y = labelencod.transform(np.array(y))

   #x_train,x_test,y_train,y_test =
       train_test_split(x,y,test_size=0.33,stratify=y,random_state=1)

except Exception as e:
   print (e)

#XGB--------------------------------------:

cross_val = StratifiedKFold(n_splits=5)
f1_scores = []
acc_scores = []
cms = []
i=1
```

```
start_xgb = time.time()
print ('Xgb cross validate starts')
for train_index, test_index in cross_val.split(x,y):
   x_train, x_test = x[train_index], x[test_index]
   y_train, y_test = y[train_index], y[test_index]
   clf_xgb =
       XGBClassifier(objective='multi:softprob',use_label_encoder=False,
                 eval_metric='mlogloss',tree_method='gpu_hist',
                 learning_rate=0.3,n_estimators=600,
                 max_depth = 5,min_child_weight=3,
                 )
   clf_xgb.fit(x_train,y_train)
   y_pred = clf_xgb.predict(x_test)
   f1 = f1_score(y_test,y_pred,average='macro',zero_division=0)
   acc = accuracy_score(y_test,y_pred)
   cm = confusion_matrix(y_test,y_pred)

   f1_scores.append(f1)
   acc_scores.append(acc)
   cms.append(cm.tolist())

   pd.DataFrame(cm).to_csv(newh_xgb+str(i)+'_cm.csv',header=0,index=False)
   i=i+1
result = np.sum(cms,axis=0)
pd.DataFrame(result).to_csv(newh_xgb+str(11)+'_cm.csv',header=0,index=False)
pd.DataFrame([f1_scores,acc_scores]).to_csv(newh_xgb+str(12)+'_report.csv',header=0,index=False)
print ('Xgb cross validate completed. Time
    used:',time.time()-start_xgb,np.mean(f1_scores))

#random forest-----------------------------------:

cross_val = StratifiedKFold(n_splits=5)
f1_scores = []
acc_scores = []
cms = []
i=1
start_rf = time.time()
print ('Random forest cross validate starts')
for train_index, test_index in cross_val.split(x,y):
   x_train, x_test = x[train_index], x[test_index]
   y_train, y_test = y[train_index], y[test_index]
   clf_rf = RandomForestClassifier()
   clf_rf.fit(x_train,y_train)
   y_pred = clf_rf.predict(x_test)
   f1 = f1_score(y_test,y_pred,average='macro',zero_division=0)
   acc = accuracy_score(y_test,y_pred)
   cm = confusion_matrix(y_test,y_pred)

   f1_scores.append(f1)
   acc_scores.append(acc)
   cms.append(cm.tolist())

   pd.DataFrame(cm).to_csv(newh_rf+str(i)+'_cm.csv',header=0,index=False)
   i=i+1
result = np.sum(cms,axis=0)
pd.DataFrame(result).to_csv(newh_rf+str(11)+'_cm.csv',header=0,index=False)
pd.DataFrame([f1_scores,acc_scores]).to_csv(newh_rf+str(12)+'_report.csv',header=0,index=False)
print ('Random forest cross validate completed. Time
    used:',time.time()-start_rf,np.mean(f1_scores))

   #print (cm)

scaler = preprocessing.StandardScaler().fit(x)
x = scaler.transform(x)
#SVM linearSVC OvR------------------------------:

cross_val = StratifiedKFold(n_splits=5)
f1_scores = []
acc_scores = []
cms = []
i=1
start_linearSVC = time.time()
print ('linearSVC cross validate starts')
for train_index, test_index in cross_val.split(x,y):
   x_train, x_test = x[train_index], x[test_index]
   y_train, y_test = y[train_index], y[test_index]
   #clf_linearSVC =
       XGBClassifier(eval_metric='mlogloss',tree_method='gpu_hist',use_label_encoder=False)
   clf_linearSVC = LinearSVC()
   clf_linearSVC.fit(x_train,y_train)
   y_pred = clf_linearSVC.predict(x_test)
   f1 = f1_score(y_test,y_pred,average='macro',zero_division=0)
   acc = accuracy_score(y_test,y_pred)
   cm = confusion_matrix(y_test,y_pred)

   f1_scores.append(f1)
   acc_scores.append(acc)
   cms.append(cm.tolist())

   pd.DataFrame(cm).to_csv(newh_linearSVC+str(i)+'_cm.csv',header=0,index=False)
   i=i+1
result = np.sum(cms,axis=0)
pd.DataFrame(result).to_csv(newh_linearSVC+str(11)+'_cm.csv',header=0,index=False)
pd.DataFrame([f1_scores,acc_scores]).to_csv(newh_linearSVC+str(12)+'_report.csv',header=0,index=Fal
print ('LinearSVC cross validate completed. Time
    used:',time.time()-start_linearSVC,np.mean(f1_scores))

#SVM SVC with linear OvO-------------------------------:

cross_val = StratifiedKFold(n_splits=5)
```

```
f1_scores = []
acc_scores = []
cms = []
i=1
start_SVClin = time.time()
print ('SVClin cross validate starts')
for train_index, test_index in cross_val.split(x,y):
   x_train, x_test = x[train_index], x[test_index]
   y_train, y_test = y[train_index], y[test_index]
   #clf_linear =
       XGBClassifier(eval_metric='mlogloss',tree_method='gpu_hist',use_label_encoder=False)
   clf_SVClin = SVC(kernel='linear')
   clf_SVClin.fit(x_train,y_train)
   y_pred = clf_SVClin.predict(x_test)
   f1 = f1_score(y_test,y_pred,average='macro',zero_division=0)
   acc = accuracy_score(y_test,y_pred)
   cm = confusion_matrix(y_test,y_pred)

   f1_scores.append(f1)
   acc_scores.append(acc)
   cms.append(cm.tolist())

   pd.DataFrame(cm).to_csv(newh_SVClin+str(i)+'_cm.csv',header=0,index=False)
   i=i+1
result = np.sum(cms,axis=0)
pd.DataFrame(result).to_csv(newh_SVClin+str(11)+'_cm.csv',header=0,index=False)
pd.DataFrame([f1_scores,acc_scores]).to_csv(newh_SVClin+str(12)+'_report.csv',header=0,index=False)
print ('SVClin cross validate completed. Time
    used:',time.time()-start_SVClin,np.mean(f1_scores))

#SVM SVC with rbf OvO------------------------------:

cross_val = StratifiedKFold(n_splits=5)
f1_scores = []
acc_scores = []
cms = []
i=1
start_SVCrbf = time.time()
print ('SVCrbf cross validate starts')
for train_index, test_index in cross_val.split(x,y):
   x_train, x_test = x[train_index], x[test_index]
   y_train, y_test = y[train_index], y[test_index]
   #clf_linear =
       XGBClassifier(eval_metric='mlogloss',tree_method='gpu_hist',use_label_encoder=False)
   clf_SVCrbf = SVC(kernel='rbf')
   clf_SVCrbf.fit(x_train,y_train)
   y_pred = clf_SVCrbf.predict(x_test)
   f1 = f1_score(y_test,y_pred,average='macro',zero_division=0)
   acc = accuracy_score(y_test,y_pred)
   cm = confusion_matrix(y_test,y_pred)
```

```
      f1_scores.append(f1)
      acc_scores.append(acc)
      cms.append(cm.tolist())

      pd.DataFrame(cm).to_csv(newh_SVCrbf+str(i)+'_cm.csv',header=0,index=False)
      i=i+1
   result = np.sum(cms,axis=0)
   pd.DataFrame(result).to_csv(newh_SVCrbf+str(11)+'_cm.csv',header=0,index=False)
   pd.DataFrame([f1_scores,acc_scores]).to_csv(newh_SVCrbf+str(12)+'_report.csv',header=0,index=False)
   print ('SVCrbf cross validate completed. Time
       used:',time.time()-start_SVCrbf,np.mean(f1_scores))

def XGB_check_test(unwanted=[],cata='no_',cata2='nochange'):
   #'C:/Zhaoyi_Fan/Dataset/Combo/training/features/'
   folder = 'C:/Zhaoyi_Fan/Dataset/Combo/training/features/'
   files = os.listdir(folder)
   x=[]
   y=[]
   for f in files[:]:
      #d = csv.reader(open(folder+f,'r'))
      the_file = folder+f+'/'+f+'_mouse_train_features.csv'
      if f == 'user18.csv':
         continue
      with open(the_file,'r',encoding='UTF-8') as thefile:

         d = csv.reader(thefile)
         d = np.array(pd.DataFrame(d))
         try:
            ss1 = float(d[0][0])
            ss2 = float(d[0][1])
         except:
            continue
         #d = np.delete(d,unwanted,axis=1)
         for i in d:
            temp = []
            if float(i[1])<1500 and float(i[3])/(float(i[0])-2)<0.2 and
                float(i[6])<2 and float(i[22])<2000 and float(i[20])<10
                and float(i[-5])/(float(i[0])-2)<0.2: # and
                float(i[3])/(float(i[0])-2)<0.3333:
               for j in i[:-1]:
                  temp.append(float(j))
               x.append(temp)
               y.append(i[-1].split('_')[0])
   print ('Training data extraction completed',len(y),'samples involved')
   try:
      x = np.delete(x,unwanted,axis=1)

      labelencod = preprocessing.LabelEncoder().fit(y)
```

```
    y = labelencod.transform(np.array(y))

    #x_train,x_test,y_train,y_test =
        train_test_split(x,y,test_size=0.33,stratify=y,random_state=1)
    x_train = x
    y_train = y

except Exception as e:
    print (e)

folder = 'C:/Zhaoyi_Fan/Dataset/Combo/testing/features/'
files = os.listdir(folder)
x=[]
y=[]
for f in files[:]:
    the_file = folder+f+'/'+f+'_mouse_test_features.csv'
    with open(the_file,'r') as thefile:
        d = csv.reader(thefile)
        d = np.array(pd.DataFrame(d))
        try:
            ss1 = float(d[0][0])
            ss2 = float(d[0][1])
        except:
            continue
        #d = np.delete(d,unwanted,axis=1)
        for i in d:
            temp = []
            if float(i[1])<1500 and float(i[3])/(float(i[0])-2)<0.2 and
                float(i[6])<2 and float(i[22])<2000 and float(i[20])<10
                and float(i[-5])/(float(i[0])-2)<0.2: # and
                float(i[3])/(float(i[0])-2)<0.3333:
                for j in i[:-1]:
                    temp.append(float(j))
                x.append(temp)
                y.append(i[-1].split('_')[0])
print ('Testing Data extraction completed',len(y),'samples involved')
try:
    x = np.delete(x,unwanted,axis=1)

    #labelencod = preprocessing.LabelEncoder().fit(y)

    y = labelencod.transform(np.array(y))

    #x_train,x_test,y_train,y_test =
        train_test_split(x,y,test_size=0.33,stratify=y,random_state=1)
    x_test = x
    y_test = y

except Exception as e:
    print (e)
```

```
pred_prob=[]
featurescore=[]

mychanges = []
theweights = []

try:
   print ('XGB fitting started')
   #cross_val = StratifiedKFold(n_splits=cvs)
   clf =
       XGBClassifier(objective='multi:softprob',use_label_encoder=False,
                eval_metric='mlogloss',tree_method='gpu_hist',
                learning_rate=0.3,n_estimators=600,
                max_depth = 5,min_child_weight=3,
                )

   features_impts = []

   myweights,mychanges = myweight(y_train)
   class_weights =
       class_weight.compute_sample_weight(myweights,y_train)
   print ('XGB fitting started',len(y_train), 'samples involved')
   clf.fit(x_train,y_train,sample_weight=class_weights)
   print ('XGB fitting completed')
   print ('XGB predicting started')
   pre_temp = clf.predict_proba(x_test)
   print ('XGB predicting completed')
   featurescore = clf.feature_importances_.tolist()

   pred_prob = np.array(pre_temp)

   labelset = np.unique(y_test)
   names = labelencod.inverse_transform(labelset)

except Exception as e:
   print (e)
   print (folder)
   return 'bad'

myweights_total,mychanges=myweight(y)
theweights.append(myweights_total)
head = np.array(theweights[0].keys())
print (head)
#heads = labelencod.inverse_transform(head).tolist()
heads = names.tolist()
weights=pd.DataFrame(theweights)
weights = np.array(weights).tolist()
weights.insert(0,heads)
```

```
weights=pd.DataFrame(weights)

dic = {} #for computing the overall result
dic2 = {} #for computing each user's performance
#print (list(labelset))
for i in list(labelset):
   dic[i]=[]
   #dic2[i]=[]
for i in range(len(y_test)):
   dic[y_test[i]].append(pred_prob[i])
pred_probtemp = []
y_testtemp = []
for i in dic:
   #ypred=[]
   #ytrue=[]
   for j in dic[i]:
      pred_probtemp.append(j)
      y_testtemp.append(i)
      #ypred.append(j.tolist().index(j.max()))
      #ytrue.append(i)
   #dic2[names[i]]=[ypred,ytrue]

#everyone=[]


pred_prob = pred_probtemp[:]
y_test = y_testtemp[:]


outs = []
f1_micro=[]
f1_macro=[]

precision_micro=[]
precision_macro=[]

recall_micro=[]
recall_macro=[]
times = []

accuracy = []
rpt = []
for i in range(1,32,1):
   y_pred = []
   y_true = []
   for j in range(len(y_test)-i+1):
      if y_test[j]==y_test[j+i-1]:
         #y_pred_byprob=sum(np.array(pred_prob[j:j+i]),axis=0)
         y_pred_byprob=sum(np.array(pred_prob[j:j+i]),axis=0)
         y_pred.append(y_pred_byprob.tolist().index(y_pred_byprob.max()))
         y_true.append(y_test[j])
   y_true = labelencod.inverse_transform(y_true)
   y_pred = labelencod.inverse_transform(y_pred)

   outs.append([
         #'%.2f'%(f1_score(y_true,y_pred,average='micro')*100),
         float('%.3f'%(f1_score(y_true,y_pred,average='macro',zero_division=0))),
         float('%.3f'%(f1_score(y_true,y_pred,average='weighted',zero_division=0))),
         '-',
         #'%.2f'%(precision_score(y_true,y_pred,average='micro')*100),
         float('%.3f'%(precision_score(y_true,y_pred,average='macro',zero_division=0))),
         float('%.3f'%(precision_score(y_true,y_pred,average='weighted',zero_division=0))),
         '-',
         #'%.2f'%(recall_score(y_true,y_pred,average='micro')*100),
         float('%.3f'%(recall_score(y_true,y_pred,average='macro',zero_division=0))),
         float('%.3f'%(recall_score(y_true,y_pred,average='weighted',zero_division=0))),
         '-',
         float('%.3f'%(accuracy_score(y_true,y_pred))),
         ])

   f1_micro.append(f1_score(y_true,y_pred,average='micro',zero_division=0)*100)
   f1_macro.append(f1_score(y_true,y_pred,average='macro',zero_division=0)*100)
   precision_micro.append(precision_score(y_true,y_pred,average='micro',zero_division=0)*100)
   precision_macro.append(precision_score(y_true,y_pred,average='macro',zero_division=0)*100)
   recall_micro.append(recall_score(y_true,y_pred,average='micro',zero_division=0)*100)
   recall_macro.append(recall_score(y_true,y_pred,average='macro',zero_division=0)*100)
   accuracy.append(accuracy_score(y_true,y_pred)*100)
   times.append(i)
   newh =
       'C:/Zhaoyi_Fan/Dataset/Combo/testing/results/Mouse/own/XGB/'+cata+'/'+cata2+'/'

   if i==1:
      if not os.path.exists(newh+str(i)+'/'):
         os.makedirs(newh+str(i)+'/')
      rpt = classification_report(y_true,y_pred,output_dict=True)
      #report=classification_report(y_true,y_pred,output_dict=True)
      pd.DataFrame(rpt).T.to_csv(newh+str(i)+'/'+str(i)+'mouse_classification_report.csv',header=0,i
      print (classification_report(y_true,y_pred))

   if (i+4)/5==int((i+4)/5):
      if not os.path.exists(newh+str(i)+'/'):
         os.makedirs(newh+str(i)+'/')
      cm_old = confusion_matrix(y_true, y_pred)

      pd.DataFrame(cm_old).to_csv(newh+str(i)+'/'+str(i)+'mouse_cm.csv',header=heads,index=True)
      cm = np.array(pd.DataFrame(cm_old)).tolist()
      cm_=[]
      for k in cm:
         temp=[]
         for j in k:
               temp.append(float('%.1f'%(100*j/sum(k))))
             cm_.append(temp)
           cm =mat(cm_.copy())

           report = classification_report(y_true,y_pred,output_dict=True)
           #report=classification_report(y_true,y_pred,output_dict=True)
           pd.DataFrame(report).T.to_csv(newh+str(i)+'/'+str(i)+'mouse_classification_report.csv',header=

           plt.figure(1,figsize=(10.24,8))
           ax1 = plt.axes()
           disp2 = ConfusionMatrixDisplay(confusion_matrix=cm)
           disp2.plot(cmap=plt.cm.Blues,include_values=False,ax=ax1)
           plt.savefig(newh+str(i)+'/'+str(i)+'mouse_cm.png')
           plt.clf()

           plt.figure(3,figsize=(10.24,8))
           ax3 = plt.axes()
           disp3 =
               ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=heads)
           disp3.plot(cmap=plt.cm.Blues,include_values=True,ax=ax3)
           plt.savefig(newh+str(i)+'/'+str(i)+'mouse_cm_newnew.png')
           plt.clf()

           plt.figure(4,figsize=(10.24,8))
           ax4 = plt.axes()
           disp4 =
               ConfusionMatrixDisplay(confusion_matrix=cm_old,display_labels=heads)
           disp4.plot(cmap=plt.cm.Blues,include_values=True,ax=ax4)
           plt.savefig(newh+str(i)+'/'+str(i)+'mouse_cm_newnewnew.png')
           plt.clf()

           np.set_printoptions(precision=2)
           cm_normalized = confusion_matrix(y_true,
               y_pred,normalize='true')
           cm_n = mat(np.round(cm_normalized,3))
           plt.figure(5,figsize=(10.24,8))
           ax5 = plt.axes()
           disp5 =
               ConfusionMatrixDisplay(confusion_matrix=cm_n,display_labels=heads)
           disp5.plot(cmap=plt.cm.Blues,include_values=True,ax=ax5)
           plt.savefig(newh+str(i)+'/'+str(i)+'mouse_cm_newnewnew2.png')
           plt.clf()

   print (pd.DataFrame(outs))
   out2 = []
   for i,j in enumerate(outs):
      out2.append(['&',j[0],'&',j[3],'&',j[6],'&',j[9],'\\\\'])
   print (pd.DataFrame(out2,index=np.arange(1,32,1)))
```

```
    plt.figure(2,figsize=(10.24,8))
    ax2 = plt.axes()
    plot_importance(clf,ax=ax2,importance_type='gain')
    plt.savefig(newh+'mouse_FI.png')
    plt.clf()

    print (featurescore)
    # plt.show()

    plt.figure(3,figsize=(10.24,8))
    #print (f1_macro,np.min(f1_macro))
    #print
        (int(np.min([np.min(f1_macro),np.min(precision_macro),np.min(recall_macro),np.min(accuracy)])))
    myyticks = [i for i in range(0,100,5)]
    myxticks = [i for i in range(1,31,1)]
    plt.yticks(myyticks)
    #plt.xticks(myxticks)
    plt.plot(times,f1_macro,label='f1_macro')
    #plt.plot(times,f1_micro,label='f1_micro')
    plt.plot(times,precision_macro,label='precision_macro')
    #plt.plot(times,precision_micro,label='precision_micro')
    plt.plot(times,recall_macro,label='recall_macro')
    #plt.plot(times,recall_micro,label='recall_micro')
    plt.plot(times,accuracy,label='accuracy')
    plt.legend()
    plt.grid()
    plt.savefig(newh+'mouse_overall.png')
    plt.clf()
    # print (type(rpt).__name__)
    #plot_importance(clf)
    #plt.show()
    return
        [rpt,outs,pred_prob,y_test,featurescore,np.delete(f_names,unwanted),weights]

def XGB_test(unwanted=[]):
    myweights,mychanges = myweight()
    cata2='nochange'
    if len(mychanges)>0:
        cata2=''
        for i in mychanges:
            temp = ''
            for j in i:
                temp = temp+str(j)+'_'
            cata2 = cata2+temp+'_'

    cata='no_'

    try:
      if len(unwanted)==0:
         cata = 'allfeatures'
      else:
         for i in unwanted:
            cata = cata+str(i)+'_'

      # finalpath =
          'C:/Workspace/Python/mouse_and_ks/forpapers/mouse/testing/'
      # users = os.listdir(finalpath)

      resultpath =
          'C:/Zhaoyi_Fan/Dataset/Combo/testing/results/Mouse/own/XGB/'+cata+'/'+cata2+'/'
      if not os.path.exists(resultpath):
         os.makedirs(resultpath)
         print (resultpath,'created')
      #r =
          checkontrain_cv(finalpath,n,[3,4,8,16,17,19,21,24],group,cvs,s)
      #r = checkontrain_cv(finalpath,n,[3,16,17,24],group,cvs,s)
      r = XGB_check_test(unwanted,cata,cata2)
      if r=='bad':
         print ('baaaaaaaaaaaaaaaad')
         return
      pd.DataFrame(r[0]).T.to_csv(resultpath+'mouse_classification_report.csv',header=0,index=True)
      pd.DataFrame(r[1]).to_csv(resultpath+'mouse_30times.csv',header=0,index=False)
      probs = pd.concat([pd.DataFrame(r[2]),pd.DataFrame(r[3])],axis=1)
      probs.to_csv(resultpath+'mouse_probabilities.csv',header=0,index=False)
      r[6].to_csv(resultpath+'mouse_sample_weight.csv',header=True,index=False)

      fscores=r[4][:]
      sorted_number = np.argsort(fscores)
      fnames = []
      scores = []
      for i in sorted_number:
         fnames.append(r[5][i]+'_'+str(i))
         scores.append(fscores[i])

      plt.figure(4,figsize=(10.24,8))
      plt.barh(fnames,scores)
      plt.grid()
      plt.savefig(resultpath+'myfeatures_importance.png')
      plt.clf()
   except Exception as e:
      print ('error occur in selecting best features')
      print (e)




if __name__=='__main__':
   XGB_test([2,5]) #get classification result on testing set.
```

```
#extractfeatures_mouse() #extract mouse features from raw data

#extractfeatures_mouse_nIQR()
#get_actions_percent()
#get_segments_figure2()
#get_segments_figure()
#classifier_select()
#go([2,5],'C:/Zhaoyi_Fan/Dataset/Combo/training/features/','C:/Zhaoyi_Fan/Dataset/Combo/training/re
```

D.3.3 Python scripts for combining keystroke and mouse dynamics

List D.3.3 lists the Python scripts for combining keystroke and mouse dynamics analysis used in Chapter 8.

```
import numpy as np
import sys
import csv
import pandas as pd
import math
from numpy import *
from time import time
import datetime
import matplotlib.pyplot as plt
from matplotlib import cm
from sklearn import preprocessing
from sklearn import svm
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.multiclass import OneVsOneClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
import random
import sklearn
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import KFold
from sklearn.metrics import make_scorer
from sklearn import linear_model
from sklearn.metrics import r2_score
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import f1_score,precision_score,recall_score
from sklearn.metrics import accuracy_score
from sklearn import tree
from xgboost import XGBClassifier,plot_importance,DMatrix,cv
from xgboost import XGBRegressor
from sklearn.ensemble import VotingClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
from sklearn.metrics import classification_report
from collections import Counter
```

```
import os
import string
from scipy.interpolate import griddata
from sklearn.utils import class_weight
from itertools import combinations

figure_path1 = 'C:/Zhaoyi_Fan/Figures/'
figure_path2 = 'C:/Zhaoyi_Fan/Latex/Figures/'

pd.set_option('display.max_rows',500)
pd.set_option('display.max_columns',500)
pd.set_option('display.width',1000)
pd.set_option('max_colwidth',1000)
np.set_printoptions(suppress=True)


# Mouse
f_names = ['points','gap','scurlpp','backward',
        'time','timepp','s/strai','widthmax_','width_min',
        'width-mean','width-std','vstrai','vcurl',
        'acc_mean','acc_std','angle_mean','angle_std',
        'squ/strai','squ/scurl','durantion','distance',
        'scurl','sstrai','acc_max','acc_min',
        'angle_min','angle_max',
        'acc_ep_mean','acc_ep_std','acc_ep_min','acc_ep_max',
        'v_c_ep_mean','v_c_ep_std','v_c_ep_min','v_c_ep_max',
        'ac_c_ep_mean','ac_c_ep_std','ac_c_ep_min','ac_c_ep_max',
        'v_seg_mean','v_seg_std','v_seg_min','v_seg_max',
        'width/sstrai','width/scurl','width/time',
        'b_relative','b_both',
        'sin','cos']

def myweight(y=arange(19)):
   myweights = {}
   for j in np.unique(y):
      myweights[j]=len(y)/(y.tolist().count(j)*len(np.unique(y)))
   try:

      for i in mychanges:
         myweights[i[0]]=myweights[i[0]]*i[1]
   except:
      mychanges=[]

   return myweights,mychanges

class Mouse():
   def __init__(self, file):
      def re(data):
         temp1 = []
         temp2 = []
```

```
  for i in data:
    a =
        (int(i[1][1])**2+int(i[1][2])**2)**0.5-(int(i[0][1])**2+int(i[0][2])**2)**0.5
    b = int(i[1][3])-int(i[0][3])
    if a<1 and b<2000:
      temp1.append(i)
      temp2.append(int(i[1][-1])-int(i[0][-1]))
  #print (temp2)
  me = np.mean(temp2)
  if len(temp2)!=0:
    #print ('!!!!')
    me = np.mean(temp2)
    std = np.std(temp2,ddof=1)
  if len(temp2)==0:
    #print ('??????')
    me = 0
    std = 0

  for i in temp1:
    if int(i[1][-1])-int(i[0][-1])-me>3*std:
      temp1.remove(i)
  return temp1
try:
  with open(file,'r') as f:
    data = np.array(pd.DataFrame(csv.reader(f)))
except:
  data = file

content = []
leftclick = []
rightclick = []
cmove=[]
cleft=[]
cright=[]
cmiddle=[]
cscroll=[]
splt = []
stemp = []
lctemp = []
rctemp=[]
count1 = 0 #number of '1'--left click down
count3 = 0

for line in data:
  if line[0]=='TYPE':
    continue
  line =  [int(j) for j in line]
  if line[0]==0 or line[0]==1 or line[0]==2:  #click down up,
     move
    stemp.append(line)
```

```
   lctemp.append(line)
   if line[0]==1: #click down
      lctemp=[]
      lctemp.append(line)
      count1=count1+1
   if line[0]==2: #click up
      if count1==1:
         leftclick.append(lctemp)
         splt.append(stemp)
         stemp=[]
         lctemp=[]
         count1=0
      else:
         stemp=[]
         lctemp=[]
         count1=0

if line[0] != 8:
   content.append(line)
if line[0]==0:
   cmove.append(line)

if line[0]==0 or line[0]==3 or line[0]==4:
   rctemp.append(line)
   if line[0]==3:
      rctemp=[]
      rctemp.append(line)
      count3=count3+1
   if line[0]==4:
      if count3==1:
         rightclick.append(rctemp)
         rctemp=[]
         count3=0
      else:
         rctemp=[]
         count3=0

if line[0]==1 or line[0]==2:
   if len(cleft)!=0 and line[0]==cleft[-1][0]:
      cleft.pop()
      cleft.append(line)
   else:
      cleft.append(line)
if line[0]==3 or line[0]==4:
   if len(cright)!=0 and line[0]==cright[-1][0]:
      cright.pop()
      cright.append(line)
   else:
      cright.append(line)
if line[0]==5 or line[0]==6:
```

```
            if len(cmiddle)!=0 and line[0]==cmiddle[-1][0]:
               cmiddle.pop()
               cmiddle.append(line)
            else:
               cmiddle.append(line)
         if line[0]==7:
            cscroll.append(line)
      self.splt=splt
      self.leftclick = leftclick
      self.rightclick = rightclick

      L=[]
      for i in range(0,len(cleft)-1,2):
         temp = [cleft[i],cleft[i+1]]
         L.append(temp)
      self.left = re(L)

      R=[]
      for i in range(0,len(cright)-1,2):
         temp = [cright[i],cright[i+1]]
         R.append(temp)
      self.right = re(R)

      M=[]
      for i in range(0,len(cmiddle)-1,2):
         temp = [cmiddle[i],cmiddle[i+1]]
         M.append(temp)
      self.middle = M
      self.rawdata = content
      self.move = cmove
      #self.left = cleft
      #self.right = cright
      #self.middle = cmiddle
      self.scroll = cscroll

   def fst(self):
      self.tem = self.rawdata[0]
      print (self.rawdata[:1000],self.tem)
      a = self.tem
      return a

def get_onemove_data_v1(da,ratio):
   datago=[]
   samples= []
   labels = []
   databyperson =[]
   p = da
   #print (p[1],'extraction started')
   samplesbyperson = []
   labelsbyperson = []
```

```
testlist=[]
features = [] #features between neiboughs [interval,distance]
spl = p[0].splt[:]
#print (len(spl))
co=0

for point in spl:
   duration = 0
   distance = 0
   i = point.copy()
   x=[]
   y=[]

   i.reverse()

   leftclick = []
   data_sub = []
   #get left click data
   #itp = i.copy()
   itemp = i[:]
   itemp_ = itemp[:]
   for j in itemp:
      if itemp[0][3]-j[3]>3000:
         itemp_ = itemp[:itemp.index(j)]
         break
   itemp = itemp_[:]

   for j in itemp:
      if j[0]==1:
         leftclick = itemp[:itemp.index(j)+1]
         i = itemp[itemp.index(j)+1:]
         break

   if len(i)==0:
      continue

   #i = itp

   i_ = [i[0]]

   j=1
   while j<len(i):
      #dis = i_[-1][0]-i[j][0]
      dis = (i_[-1][1]-i[j][1])**2+(i_[-1][2]-i[j][2])**2
      if dis == 0:
         for k in range(j,len(i),1):
            dis2 = (i_[-1][1]-i[k][1])**2+(i_[-1][2]-i[k][2])**2
            if dis2!=0:
               i_[-1] =
                    [i_[-1][0],i_[-1][1],i_[-1][2],(i_[-1][3]+i[k-1][3])/2]
```

```
          j=k
          #print (j)
          break
        elif k == len(i)-1:
          i_[-1] =
              [i_[-1][0],i_[-1][1],i_[-1][2],(i_[-1][3]+i[k][3])/2]
          j+=1
          break
   else:
     i_.append(i[j])
     j+=1

i = i_

i.reverse()

if len(i)<6:
   continue

moveth=[]
tempi=i[:]
for j in range(len(tempi)-1):
   moveth.append(tempi[j+1][-1]-tempi[j][-1])
try:
   Q3 = np.percentile(moveth,75)
   Q1 = np.percentile(moveth,25)
   up = 2.5*Q3-1.5*Q1
   down = 2.5*Q1-1.5*Q3
except Exception as e:
   print (e,tempi)
   break
#print (p[1],'Q1 Q3:',up,down)
i.reverse()
itemp=i[:]
for j in range(len(itemp)-1):
   if ratio<3:
     if itemp[j][3]-itemp[j+1][3]>up*ratio or
         itemp[j][3]-itemp[j+1][3]<0:
        i = itemp[:j+1]
        break
     elif itemp[0][3]-itemp[j][3]>2000:
        i = itemp[:j]
        break
   elif ratio>=3:
     if up<100:
        if itemp[j][3]-itemp[j+1][3]>up*ratio or
            itemp[j][3]-itemp[j+1][3]<0:
          i = itemp[:j+1]
          break
        elif itemp[0][3]-itemp[j][3]>2000:
            i = itemp[:j]
            break
      elif up>=100:
         if itemp[j][3]-itemp[j+1][3]>up*2 or
              itemp[j][3]-itemp[j+1][3]<0:
            i = itemp[:j+1]
            break
         elif itemp[0][3]-itemp[j][3]>2000:
            i = itemp[:j]
            break

# #set total moving time

# i=i2.copy()

#print (id(i),'Person:',datalist.index(p),len(i))

for j in range(len(i)-1):
   dis = ((i[j][1]-i[j+1][1])**2+(i[j][2]-i[j+1][2])**2)**0.5
   testlist.append([i[j][3]-i[j+1][3],dis])

try:
   if leftclick[0][0]==2 and leftclick[-1][0]==1:
      duration = leftclick[0][3]-leftclick[-1][3]
      distance = ((leftclick[0][1]-leftclick[-1][1])**2 +
           (leftclick[0][2]-leftclick[-1][2])**2)**0.5
except:
   continue

if duration>400:
   continue

# if distance>5:
#  continue

if len(i)<6:
   continue

itemp = i[:]
distemp=[]

for j in range(len(itemp)):
   distemp.append(((itemp[j][1]-itemp[0][1])**2+(itemp[j][2]-itemp[0][2])**2)**0.5)

maxdist = distemp.index(np.max(distemp))
i = i[0:maxdist+1]

if len(i)<6:
   continue
```

```
i.reverse()


x=i[-1][1]-i[0][1] #x from start to end
y=i[-1][2]-i[0][2] #y from start to end

try:
   sin_coordinate = round(x/((x**2+y**2)**0.5),8)
   cos_coordinate = round(y/((x**2+y**2)**0.5),8)
except Exception as e:
   sin_coordinate = 0
   cos_coordinate = 0
   print (e)
   continue
#get square, width, speed,
squ = 0
vcurl = 0
vstrai = 0
s=0
v_=0
v0=0
sstrai = 0
scurl = 0
dlist = []
width = 0
time = 0
acc = []
angles=[]
angle_=0
v_each_seg= []
acc_eachpoint=[]
backward_relative = 0
backward_both = 0
backward_count = 0
isForward = True

if (((i[0][1]-i[-1][1] )**2+(i[0][2] -i[-1][2])**2)**0.5)==0:
   continue


for a in range(len(i)):
   L_=0
   d_=0
   t_=0

   maxd = 0
   mind = 0

   L = 0
```

```
Xs=i[0][1]
Ys=i[0][2]
X0=i[a][1]
Y0=i[a][2]
X_=0
Y_=0
Xe=i[-1][1]
Ye=i[-1][2]
t0=i[a][3]
try:
   d0 =
        abs(((X0-Xs)*(Ys-Ye)-(Xs-Xe)*(Y0-Ys))/(((Xs-Xe)**2+(Ys-Ye)**2)**0.5))
except:
   print (i)
d0 =
    abs(((X0-Xs)*(Ys-Ye)-(Xs-Xe)*(Y0-Ys))/(((Xs-Xe)**2+(Ys-Ye)**2)**0.5))
L0 = (abs((Xs-X0)**2 + (Ys-Y0)**2 - d0**2))**0.5
if (Xs-X0)**2 + (Ys-Y0)**2 >= d0**2:
   L0 = ((Xs-X0)**2 + (Ys-Y0)**2 - d0**2)**0.5
elif (Xs-X0)**2 + (Ys-Y0)**2 < d0**2:
   L0 = -((abs((Xs-X0)**2 + (Ys-Y0)**2 - d0**2))**0.5)

if a==0:
   L_=0
   d_=0
   L0=0
   L = L0-L_
   v_=0
   v0=0
   t_=0
   s0=0
   d0=0
   sq = (d_+d0)*L/2
   acc=[]
   dlist.append(d0)
   #v_each.append(v0)
   #print (L,L0,L_,sq,v)
else:
   X_ = i[a-1][1]
   Y_ = i[a-1][2]
   t_ = i[a-1][3]
   d_ =
        abs(((X_-Xs)*(Ys-Ye)-(Xs-Xe)*(Y_-Ys))/(((Xs-Xe)**2+(Ys-Ye)**2)**0.5))
   L_ = (abs((Xs-X_)**2 + (Ys-Y_)**2 - d_**2)**0.5)
   if ((Xs-X_)**2 + (Ys-Y_)**2) >= d_**2:
      L_ = ((Xs-X_)**2 + (Ys-Y_)**2 - d_**2)**0.5
   elif ((Xs-X_)**2 + (Ys-Y_)**2)< d_**2:
      L_ = -((abs((Xs-X_)**2 + (Ys-Y_)**2 - d_**2)**0.5))
   L = L0-L_
   try:
```

```
   v0 = (((X0-X_)**2 + (Y0-Y_)**2)**0.5)/(t0-t_)
   v_each_seg.append(v0)
except:
   print (i)
   print (p[1])
acc_=(v0-v_)/(t0-t_)
acc.append(acc_)
v_=v0

s0 = ((X0-X_)**2 + (Y0-Y_)**2)**0.5
s = s+s0
if(type(L_).__name__=='complex'):
   print (L_,L0)
   continue

sq = abs((d_+d0)*L/2)

# if (type(sq).__name__=='complex'):
#  print ('!!!!!!!!!!',i[a],sq)
dlist.append(((X0-Xs)*(Ys-Ye)-(Xs-Xe)*(Y0-Ys))/(((Xs-Xe)**2+(Ys-Ye)**2)**0.5))
#print (L,L0,L_,sq,v)

#angles
if a<len(i)-1:
   # P0,    P_,     P_N
   # X0,Y0  X_ Y_   X_N Y_N
   X_N = i[a+1][1]
   Y_N = i[a+1][2]
   a1 = [X_-X0,Y_-Y0]
   a2 = [X0-X_N,Y0-Y_N]
   angle_ = math.acos((a1[0]*a2[0]+a1[1]*a2[1]) /
       (((a1[0]**2+a1[1]**2)*(a2[0]**2+a2[1]**2))**0.5))
   if angle_/np.pi>0.5:
      backward_relative=backward_relative+1
   angles.append(angle_)

   b1 = [X_N-X0,Y_N-Y0]
   b2 = [i[-1][1]-i[0][1],i[-1][2]-i[0][2]]
   angle2 = math.acos((b1[0]*b2[0]+b1[1]*b2[1]) /
       (((b1[0]**2+b1[1]**2)*(b2[0]**2+b2[1]**2))**0.5))

   if isForward:
      if angle2/np.pi>0.5:
         backward_count=backward_count+1
         if angle_/np.pi >0.5:
            backward_both = backward_both+1
         isForward = False
   elif not isForward:
      if angle2/np.pi<0.5:
         backward_count=backward_count+1
```

```
                if angle_/np.pi>0.5:
                    backward_both = backward_both+1
                isForward = True


    squ = squ + sq

maxd = max(dlist)
mind = min(dlist)

if mind<0:
    width = maxd-mind
else:
    width=maxd

width_sum = np.sum(dlist)
time = i[-1][3]-i[0][3]
scurl = s
vcurl = s/time
sstrai = ((i[-1][1]-i[0][1])**2+(i[-1][2]-i[0][2])**2)**0.5

if sstrai==0: #if the path is a closed circle,delete this data
    continue

acc_eachpoint=np.diff(np.array(v_each_seg)).tolist()

vchaos_eachpoint = []
accchaos_eachpoint = []
for j in range(len(acc_eachpoint)):
    an = angles[j]
    ac = acc_eachpoint[j]
    ac2 = acc[j+1]-acc[j]

    temp = 0
    if an>0 and ac>0:
        temp = (an+1)*ac + (ac+1)*an
    elif an>0 and ac<0:
        temp = (an+1)*ac + (ac-1)*an
    elif an<0 and ac>0:
        temp = (an-1)*ac + (ac+1)*an
    elif an<0 and ac<0:
        temp = (an-1)*ac + (ac-1)*an
    elif an==0 and ac!=0:
        temp = ac
    elif an!=0 and ac==0:
        temp = an
    vchaos_eachpoint.append(temp)

    temp2=0
    if an>0 and ac2>0:
        temp2 = (an+1)*ac2 + (ac2+1)*an
    elif an>0 and ac2<0:
        temp2 = (an+1)*ac2 + (ac2-1)*an
    elif an<0 and ac2>0:
        temp2 = (an-1)*ac2 + (ac2+1)*an
    elif an<0 and ac2<0:
        temp2 = (an-1)*ac2 + (ac2-1)*an
    elif an==0 and ac2!=0:
        temp2 = ac2
    elif an!=0 and ac2==0:
        temp2 = an
    accchaos_eachpoint.append(temp2)


vstrai = sstrai/time
acc_mean = np.mean(acc)
acc_std = np.std(acc,ddof=1)
acc_min = np.min(acc)
acc_max = np.max(acc)

angle_mean = np.mean(angles)
angle_std = np.std(angles,ddof=1)
angle_min = np.min(angles)
angles_max = np.max(angles)

acc_eachpoint_mean = np.mean(acc_eachpoint)
acc_eachpoint_std = np.std(acc_eachpoint,ddof=1)
acc_eachpoint_min = np.min(acc_eachpoint)
acc_eachpoint_max = np.max(acc_eachpoint)

vchaos_eachpoint_mean = np.mean(vchaos_eachpoint)
vchaos_eachpoint_std = np.std(vchaos_eachpoint,ddof=1)
vchaos_eachpoint_min = np.min(vchaos_eachpoint)
vchaos_eachpoint_max = np.max(vchaos_eachpoint)

accchaos_eachpoint_mean = np.mean(accchaos_eachpoint)
accchaos_eachpoint_std = np.std(accchaos_eachpoint,ddof=1)
accchaos_eachpoint_min = np.min(accchaos_eachpoint)
accchaos_eachpoint_max = np.max(accchaos_eachpoint)

v_seg_mean = np.mean(v_each_seg)
v_seg_std = np.std(v_each_seg)
v_seg_max = np.max(v_each_seg)
v_seg_min = np.min(v_each_seg)




width_mean = np.mean(dlist)
width_std = np.std(dlist,ddof=1)
```

```
click_move_gap = (leftclick[-1][3]-i[-1][3])

#data_sub.append(zone)
data_sub.append(len(i))
data_sub.append(click_move_gap)
data_sub.append(scurl/(len(i)-1))
data_sub.append(backward_count)
#data_sub.append(backward_count/(len(i)-1))
data_sub.append(time)

data_sub.append(time/(len(i)-1))

data_sub.append(s/sstrai) #scurl/sstraight
#data_sub.append(width) #max width
data_sub.append(maxd)
data_sub.append(mind)
data_sub.append(width_mean)
data_sub.append(width_std)

data_sub.append(vstrai) #speed of straight away
data_sub.append(vcurl) #speed of actual move
data_sub.append(acc_mean)
data_sub.append(acc_std)

data_sub.append(angle_mean)
data_sub.append(angle_std)
#data_sub.append(max(angles))
data_sub.append(squ/sstrai)
data_sub.append(squ/scurl)
data_sub.append(duration) #mosue click
data_sub.append(distance) #mosue click
data_sub.append(scurl)
data_sub.append(sstrai)
data_sub.append(acc_max)
data_sub.append(acc_min)
data_sub.append(angle_min)
data_sub.append(angles_max)

data_sub.append(acc_eachpoint_mean)
data_sub.append(acc_eachpoint_std)
data_sub.append(acc_eachpoint_min)
data_sub.append(acc_eachpoint_max)

data_sub.append(vchaos_eachpoint_mean)
data_sub.append(vchaos_eachpoint_std)
data_sub.append(vchaos_eachpoint_min)
data_sub.append(vchaos_eachpoint_max)

data_sub.append(accchaos_eachpoint_mean)
data_sub.append(accchaos_eachpoint_std)
```

```
            data_sub.append(accchaos_eachpoint_min)
            data_sub.append(accchaos_eachpoint_max)

            data_sub.append(v_seg_mean)
            data_sub.append(v_seg_std)
            data_sub.append(v_seg_min)
            data_sub.append(v_seg_max)


            data_sub.append(width/sstrai)
            data_sub.append(width/scurl)
            data_sub.append(width/time)

            data_sub.append(backward_relative)
            data_sub.append(backward_both)

            data_sub.append(sin_coordinate)
            data_sub.append(cos_coordinate)

            #data_sub.append(angles_max - angle_min)


            co+=1

            features.append(data_sub)


    username = p[1].split('.')[0]
    for i in features:
        samples.append(i)
        labels.append(username)


    return [samples,labels]


def extractfeatures_mouse():
    path = 'C:/Zhaoyi_Fan/Dataset/Combo/'
    groups = ['training','testing']
    count=0
    for group in groups:
        newpath = path+group+'/'
        for IQR in range(1,6,1):
            if IQR!=3:
                continue
            finalpath = newpath
            users = os.listdir(finalpath+'samples/')
            for user in users:
                newhome = newpath+'features/'+user+'/'
                isExists=os.path.exists(newhome)
            if not isExists:
               os.makedirs(newhome)
               print (newhome,' created')
            files = os.listdir(finalpath+'samples/'+user+'/')
            for f in files:
               label = f.split('.csv')[0]
               if label.split('_')[1]=='keystroke':
                  continue
               file = [Mouse(finalpath+'samples/'+user+'/'+f),label]
               data = get_onemove_data_v1(file,IQR)
               r =
                   pd.concat([pd.DataFrame(data[0]),pd.DataFrame(data[1])],axis=1)
               r.to_csv(newhome+label+'_features.csv',header=0,index=False)


               print (f,'files
                   completed>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>')

def my_check(unwanted=[],cata='no_',cata2='nochange',src='',results=''):
   folder = src
   files = os.listdir(folder)
   x=[]
   y=[]
   for f in files[:]:
      with open(folder+f,'r') as thefile:
         d = csv.reader(thefile)
         d = np.array(pd.DataFrame(d))[:]
         try:
            ss1 = float(d[0][0])
            ss2 = float(d[0][1])
         except:
            continue
         #d = np.delete(d,unwanted,axis=1)
         for i in d:
            temp = []
            if float(i[1])<1500 and float(i[3])/(float(i[0])-2)<0.2 and
                float(i[6])<2 and float(i[22])<2000 and float(i[20])<10
                and float(i[-5])/(float(i[0])-2)<0.2: # and
                float(i[3])/(float(i[0])-2)<0.3333:
               for j in i[:-1]:
                  temp.append(float(j))
               x.append(temp)
               #y.append(i[-1].split('.csv')[0])
               y.append(f.split('_')[0])
   print ('Data extraction completed',len(y),'samples involved')
   try:
      x = np.delete(x,unwanted,axis=1)

      labelencod = preprocessing.LabelEncoder().fit(y)
```

```
    y = labelencod.transform(np.array(y))

    x_train,x_test,y_train,y_test =
        train_test_split(x,y,test_size=0.2,stratify=y,random_state=1)

except Exception as e:
    print (e)


pred_prob=[]
featurescore=[]

mychanges = []
theweights = []

try:
    print ('XGB fitting started')
    #cross_val = StratifiedKFold(n_splits=cvs)
    clf =
        XGBClassifier(objective='multi:softprob',use_label_encoder=False,
                  eval_metric='mlogloss',tree_method='gpu_hist',
                  learning_rate=0.3,n_estimators=500,
                  max_depth = 4,min_child_weight=3,
                  )

    features_impts = []

    myweights,mychanges = myweight(y_train)
    class_weights =
        class_weight.compute_sample_weight(myweights,y_train)
    print ('XGB fitting started',len(y_train), 'samples involved')
    clf.fit(x_train,y_train,sample_weight=class_weights)
    print ('XGB fitting completed')
    print ('XGB predicting started')
    pre_temp = clf.predict_proba(x_test)
    print ('XGB predicting completed')
    featurescore = clf.feature_importances_.tolist()

    pred_prob = np.array(pre_temp)

    labelset = np.unique(y_test)
    names = labelencod.inverse_transform(labelset)

except Exception as e:
    print (e)
    print (folder)
    return 'bad'

myweights_total,mychanges=myweight(y)
theweights.append(myweights_total)
```

```
head = np.array(theweights[0].keys())
print (head)
#heads = labelencod.inverse_transform(head).tolist()
heads = names.tolist()
weights=pd.DataFrame(theweights)
weights = np.array(weights).tolist()
weights.insert(0,heads)
weights=pd.DataFrame(weights)

dic = {} #for computing the overall result
dic2 = {} #for computing each user's performance
#print (list(labelset))
for i in list(labelset):
   dic[i]=[]
   #dic2[i]=[]
for i in range(len(y_test)):
   dic[y_test[i]].append(pred_prob[i])
pred_probtemp = []
y_testtemp = []
for i in dic:
   #ypred=[]
   #ytrue=[]
   for j in dic[i]:
      pred_probtemp.append(j)
      y_testtemp.append(i)
      #ypred.append(j.tolist().index(j.max()))
      #ytrue.append(i)
   #dic2[names[i]]=[ypred,ytrue]

#everyone=[]


pred_prob = pred_probtemp[:]
y_test = y_testtemp[:]


outs = []
f1_micro=[]
f1_macro=[]

precision_micro=[]
precision_macro=[]

recall_micro=[]
recall_macro=[]
times = []

accuracy = []
rpt = []
for i in range(1,32,1):
```

```
y_pred = []
y_true = []
for j in range(len(y_test)-i+1):
   if y_test[j]==y_test[j+i-1]:
      #y_pred_byprob=sum(np.array(pred_prob[j:j+i]),axis=0)
      y_pred_byprob=sum(np.array(pred_prob[j:j+i]),axis=0)
      y_pred.append(y_pred_byprob.tolist().index(y_pred_byprob.max()))
      y_true.append(y_test[j])
y_true = labelencod.inverse_transform(y_true)
y_pred = labelencod.inverse_transform(y_pred)

outs.append([
      #'%.2f'%(f1_score(y_true,y_pred,average='micro')*100),
      float('%.2f'%(f1_score(y_true,y_pred,average='macro',zero_division=0)*100)),
      float('%.2f'%(f1_score(y_true,y_pred,average='weighted',zero_division=0)*100)),
      '-',
      #'%.2f'%(precision_score(y_true,y_pred,average='micro')*100),
      float('%.2f'%(precision_score(y_true,y_pred,average='macro',zero_division=0)*100)),
      float('%.2f'%(precision_score(y_true,y_pred,average='weighted',zero_division=0)*100)),
      '-',
      #'%.2f'%(recall_score(y_true,y_pred,average='micro')*100),
      float('%.2f'%(recall_score(y_true,y_pred,average='macro',zero_division=0)*100)),
      float('%.2f'%(recall_score(y_true,y_pred,average='weighted',zero_division=0)*100)),
      '-',
      float('%.2f'%(accuracy_score(y_true,y_pred)*100)),
      ])

f1_micro.append(f1_score(y_true,y_pred,average='micro',zero_division=0)*100)
f1_macro.append(f1_score(y_true,y_pred,average='macro',zero_division=0)*100)
precision_micro.append(precision_score(y_true,y_pred,average='micro',zero_division=0)*100)
precision_macro.append(precision_score(y_true,y_pred,average='macro',zero_division=0)*100)
recall_micro.append(recall_score(y_true,y_pred,average='micro',zero_division=0)*100)
recall_macro.append(recall_score(y_true,y_pred,average='macro',zero_division=0)*100)
accuracy.append(accuracy_score(y_true,y_pred)*100)
times.append(i)
newh = results+cata+'/'+cata2+'/'

if i==1:
   if not os.path.exists(newh+str(i)+'/'):
      os.makedirs(newh+str(i)+'/')
   rpt = classification_report(y_true,y_pred,output_dict=True)
   #report=classification_report(y_true,y_pred,output_dict=True)
   pd.DataFrame(rpt).T.to_csv(newh+str(i)+'/'+str(i)+'_classification_report.csv',header=0,index=
   print (classification_report(y_true,y_pred))

if (i+4)/5==int((i+4)/5):
   if not os.path.exists(newh+str(i)+'/'):
      os.makedirs(newh+str(i)+'/')
   cm_old = confusion_matrix(y_true, y_pred)
```

```
pd.DataFrame(cm_old).to_csv(newh+str(i)+'/'+str(i)+'_cm.csv',header=heads,index=True)
cm = np.array(pd.DataFrame(cm_old)).tolist()
cm_=[]
for k in cm:
   temp=[]
   for j in k:
      temp.append(float('%.1f'%(100*j/sum(k))))
   cm_.append(temp)
cm =mat(cm_.copy())

report = classification_report(y_true,y_pred,output_dict=True)
#report=classification_report(y_true,y_pred,output_dict=True)
pd.DataFrame(report).T.to_csv(newh+str(i)+'/'+str(i)+'_classification_report.csv',header=0,inc

plt.figure(1,figsize=(10.24,8))
ax1 = plt.axes()
disp2 = ConfusionMatrixDisplay(confusion_matrix=cm)
disp2.plot(cmap=plt.cm.Blues,include_values=False,ax=ax1)
plt.savefig(newh+str(i)+'/'+str(i)+'_cm.png')
plt.clf()

plt.figure(3,figsize=(10.24,8))
ax3 = plt.axes()
disp3 =
    ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=heads)
disp3.plot(cmap=plt.cm.Blues,include_values=True,ax=ax3)
plt.savefig(newh+str(i)+'/'+str(i)+'_cm_newnew.png')
plt.clf()

plt.figure(4,figsize=(10.24,8))
ax4 = plt.axes()
disp4 =
    ConfusionMatrixDisplay(confusion_matrix=cm_old,display_labels=heads)
disp4.plot(cmap=plt.cm.Blues,include_values=True,ax=ax4)
plt.savefig(newh+str(i)+'/'+str(i)+'_cm_newnewnew.png')
plt.clf()

#np.set_printoptions(precision=2)
cm_normalized = confusion_matrix(y_true,
    y_pred,normalize='true')
cm_n = mat(np.round(cm_normalized,3))
plt.figure(5,figsize=(10.24,8))
ax5 = plt.axes()
disp5 =
    ConfusionMatrixDisplay(confusion_matrix=cm_n,display_labels=heads)
disp5.plot(cmap=plt.cm.Blues,include_values=True,ax=ax5)
plt.savefig(newh+str(i)+'/'+str(i)+'_cm_newnewnew2.png')
plt.clf()

#np.set_printoptions(precision=2)
```

```
        cm_normalized2 = confusion_matrix(y_true,
            y_pred,normalize='pred')
        cm_n = mat(np.round(cm_normalized2,3))
        plt.figure(6,figsize=(10.24,8))
        ax6 = plt.axes()
        disp6 =
            ConfusionMatrixDisplay(confusion_matrix=cm_n,display_labels=heads)
        disp6.plot(cmap=plt.cm.Blues,include_values=True,ax=ax6)
        plt.savefig(newh+str(i)+'/'+str(i)+'_cm_newnewnew3.png')
        plt.clf()

   print (pd.DataFrame(outs))

   plt.figure(2,figsize=(10.24,8))
   ax2 = plt.axes()
   plot_importance(clf,ax=ax2,importance_type='gain')
   plt.savefig(newh+'/FI.png')
   plt.clf()

   print (featurescore)
   # plt.show()

   plt.figure(3,figsize=(10.24,8))
   #print (f1_macro,np.min(f1_macro))
   #print
       (int(np.min([np.min(f1_macro),np.min(precision_macro),np.min(recall_macro),np.min(accuracy)])))
   myyticks = [i for i in range(0,100,5)]
   myxticks = [i for i in range(1,31,1)]
   plt.yticks(myyticks)
   #plt.xticks(myxticks)
   plt.plot(times,f1_macro,label='f1_macro')
   #plt.plot(times,f1_micro,label='f1_micro')
   plt.plot(times,precision_macro,label='precision_macro')
   #plt.plot(times,precision_micro,label='precision_micro')
   plt.plot(times,recall_macro,label='recall_macro')
   #plt.plot(times,recall_micro,label='recall_micro')
   plt.plot(times,accuracy,label='accuracy')
   plt.legend()
   plt.grid()
   plt.savefig(newh+'overall.png')
   plt.clf()
   # print (type(rpt).__name__)
   #plot_importance(clf)
   #plt.show()
   return
       [rpt,outs,pred_prob,y_test,featurescore,np.delete(f_names,unwanted),weights]

def my_train(unwanted=[],src='',results=''):
   myweights,mychanges = myweight()
   cata2='nochange'
```

```
if len(mychanges)>0:
   cata2=''
   for i in mychanges:
      temp = ''
      for j in i:
         temp = temp+str(j)+'_'
      cata2 = cata2+temp+'_'

cata='no_'

try:
   if len(unwanted)==0:
      cata = 'allfeatures'
   else:
      for i in unwanted:
         cata = cata+str(i)+'_'

   finalpath = src
   users = os.listdir(finalpath)

   resultpath = results+cata+'/'+cata2+'/'
   if not os.path.exists(resultpath):
      os.makedirs(resultpath)
      print (resultpath,'created')
   #r =
       checkontrain_cv(finalpath,n,[3,4,8,16,17,19,21,24],group,cvs,s)
   #r = checkontrain_cv(finalpath,n,[3,16,17,24],group,cvs,s)
   r = my_check(unwanted,cata,cata2,src,results)
   if r=='bad':
      print ('baaaaaaaaaaaaaaaad')
      return
   pd.DataFrame(r[0]).T.to_csv(resultpath+'classification_report.csv',header=0,index=True)
   pd.DataFrame(r[1]).to_csv(resultpath+'30times.csv',header=0,index=False)
   probs = pd.concat([pd.DataFrame(r[2]),pd.DataFrame(r[3])],axis=1)
   probs.to_csv(resultpath+'probabilities.csv',header=0,index=False)
   r[6].to_csv(resultpath+'sample_weight.csv',header=True,index=False)

   fscores=r[4][:]
   sorted_number = np.argsort(fscores)
   fnames = []
   scores = []
   for i in sorted_number:
      fnames.append(r[5][i]+'_'+str(i))
      scores.append(fscores[i])

   plt.figure(4,figsize=(10.24,8))
   plt.barh(fnames,scores)
   plt.grid()
   plt.savefig(resultpath+'myfeatures_importance.png')
   plt.clf()
   except Exception as e:
      print ('error occur in selecting best features')
      print (e)


def go(unwanted=[],src='',results=''):
   if not os.path.exists(results):
      os.makedirs(results)
      print (results,'created')
   my_train(unwanted,src,results)


#Keystroke

def tobi(strr):
   if strr=='true':
      return 1
   else:
      return 0


def iskeyX(d):
   if d!='':
      if d[-1].isupper() and len(d)==4:
         return True

def switch(d):
   if d!='':
      if d[-1].isupper() and len(d)==4:
         return d[-1]
      elif d=='Backspace':
         return 'Back'
      else:
         return d
   else:
      return d

def is_bp(k):
   bad = ['Backspace','Arrow','keycode','Volume']
   for b in bad:
      if b in k[3]:
         return True
   return False

def train_n_testvesion(n=4):
   path = 'C:/Zhaoyi_Fan/Dataset/Combo/training/samples/'
   feat_path = 'C:/Zhaoyi_Fan/Dataset/Combo/training/features/'
   dest =
       'C:/Zhaoyi_Fan/Dataset/Combo/training/results/Keystroke/'+str(n)+'_graphs/XGB/'
   if not os.path.exists(dest):
```

```
   os.makedirs(dest)
   print (dest,'created')
newh = dest
users = os.listdir(feat_path)

#frequently used keys: k_keys
f_keys = ['Space', 'Backspace', 'KeyI', 'KeyA', 'KeyN', 'KeyE',
     'KeyO', 'KeyU', 'KeyH', 'Enter', 'KeyG', 'KeyS', 'KeyC', 'KeyD',
     'KeyT', 'KeyL', 'KeyR', 'KeyY', 'KeyM', 'ShiftLeft', 'Digit1',
     'KeyB', 'Digit0', 'KeyZ', 'Digit2', 'KeyJ', 'KeyW', 'KeyX',
     'KeyP', 'KeyF', 'KeyV', 'Digit3', 'KeyK', 'Period', 'Digit9',
     'KeyQ', 'Digit4', 'Digit6', 'Digit7', 'Equal', 'Comma', 'Minus']
x=[]
y=[]
X=[]
Y=[]
for user in users[:]:
   file = path+user+'/'+user+'_keystroke_train.csv'
   name = user.split('_')[0]
   with open(file,'r',encoding='UTF-8') as f:
      data = np.array(pd.DataFrame(csv.reader(f))).tolist()
   out = []
   temp=[]
   the_data= []
   for i in data[1:]:
      if len(temp) == 0:
         temp.append(i)
         #print (0,i[3])
      else:
         d=i[3]
         if int(i[2])-int(i[1])<10000 and int(i[2])-int(i[1])>0 and
              d!='' and d!='ArrowRight' and d!='ArrowLeft' and
              d!='ArrowDown' and d!='ArrowUp' and d!='keycode' and
              'Volume' not in d:

            if int(i[1])-int(temp[-1][1])<1000:
            #if int(i[1])-int(temp[-1][1])<1000 and d in f_keys:

               temp.append(i)
               #print ('<1000',i[3])
            else:
               out.append(temp)
               X.append(temp)
               Y.append(name)
               temp=[]
               temp.append(i)

print (len(X),len(Y))

X_train,X_test,Y_train,Y_test =
    train_test_split(X,Y,test_size=0.2,stratify=Y,random_state=1)

x_train=[]
x_test=[]
y_train=[]
y_test=[]

#for sample_set in
    [[X_train,Y_train,x_train,y_train],[X_test,Y_test,x_test,y_test]]:
for d,name in zip(X_train,Y_train):
   #for d in out:
   if len(d)>n-1:
      for i in range(len(d)-n+1):
         boolean = True
         for boo in range(n-1):
            if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
               boolean=False
         if boolean:
            durations = []
            DD = []
            UU = []
            DU = []
            UD = []

            caps=[]
            shifts=[]

            for j in range(n):
               durations.append(int(d[i+j][2])-int(d[i+j][1]))
               caps.append(tobi(d[i+j][-1]))
               shifts.append(tobi(d[i+j][-2]))

            combo = list(combinations(np.arange(n),2))
            for c in combo:

               DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
               UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
               DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
               UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

            c = durations+DD+UU+DU+UD+caps+shifts

            my_feature=c[:]
            x_train.append(my_feature)
            #x.append([firstduration,secondduration,DDtime,UUtime,DUtime,UDtime])
            y_train.append(name)
               # temp2=my_feature[:]
               # temp2.append(name)

               # the_data.append(temp2)
   #pd.DataFrame(the_data).to_csv(feat_path+name+'trigraph.csv',header=0,index=False)

print ('Training Data extraction completed',len(y_train),'samples
    involved')
unwanted=[]
#x = np.delete(x,unwanted,axis=1)
#print (np.unique(y))
labelencod = preprocessing.LabelEncoder().fit(y_train)

#y = labelencod.transform(y)
x_train = np.delete(x_train,unwanted,axis=1)
y_train = labelencod.transform(y_train)

for d,name in zip(X_test,Y_test):
   #for d in out:
   if len(d)>n-1:
      for i in range(len(d)-n+1):
         boolean = True
         for boo in range(n-1):
            if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
               boolean=False
         if boolean:
            durations = []
            DD = []
            UU = []
            DU = []
            UD = []

            caps=[]
            shifts=[]

            for j in range(n):
               durations.append(int(d[i+j][2])-int(d[i+j][1]))
               caps.append(tobi(d[i+j][-1]))
               shifts.append(tobi(d[i+j][-2]))

            combo = list(combinations(np.arange(n),2))
            for c in combo:

               DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
               UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
               DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
               UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

            c = durations+DD+UU+DU+UD+caps+shifts

            my_feature=c[:]
            x_test.append(my_feature)
            #x.append([firstduration,secondduration,DDtime,UUtime,DUtime,UDtime])
```

```
          y_test.append(name)
          # temp2=my_feature[:]
          # temp2.append(name)

          # the_data.append(temp2)

   #pd.DataFrame(the_data).to_csv(feat_path+name+'trigraph.csv',header=0,index=False)

print ('Testing Data extraction completed',len(y_test),'samples
    involved')
unwanted=[]

x_test = np.delete(x_test,unwanted,axis=1)
y_test = labelencod.transform(y_test)


pred_prob=[]
featurescore=[]

mychanges = []
theweights = []

try:
   print ('XGB fitting started')
   #cross_val = StratifiedKFold(n_splits=cvs)
   clf =
       XGBClassifier(objective='multi:softprob',use_label_encoder=False,
                eval_metric='mlogloss',tree_method='gpu_hist',
                learning_rate=0.3,n_estimators=600,
                max_depth = 5,min_child_weight=5,
                )
   #clf = lgb.LGBMClassifier()
   features_impts = []

   myweights,mychanges = myweight(y_train)
   print (myweights,mychanges)
   class_weights =
       class_weight.compute_sample_weight(myweights,y_train)
   print ('XGB fitting started',len(y_train), 'samples involved')
   clf.fit(x_train,y_train,sample_weight=class_weights)
   print ('XGB fitting completed')
   print ('XGB predicting started')
   pre_temp = clf.predict_proba(x_test)
   print ('XGB predicting completed')
   featurescore = clf.feature_importances_.tolist()

   pred_prob = np.array(pre_temp)

   labelset = np.unique(y_test)
   names = labelencod.inverse_transform(labelset)
```

```
except Exception as e:
   print (e)
   #print (folder)
   return 'bad'

myweights_total,mychanges=myweight(y)
theweights.append(myweights_total)
head = np.array(theweights[0].keys())
print (head)
#heads = labelencod.inverse_transform(head).tolist()
heads = names.tolist()
weights=pd.DataFrame(theweights)
weights = np.array(weights).tolist()
weights.insert(0,heads)
weights=pd.DataFrame(weights)

dic = {} #for computing the overall result
dic2 = {} #for computing each user's performance
#print (list(labelset))
for i in list(labelset):
   dic[i]=[]
   #dic2[i]=[]
for i in range(len(y_test)):
   dic[y_test[i]].append(pred_prob[i])
pred_probtemp = []
y_testtemp = []
for i in dic:
   #ypred=[]
   #ytrue=[]
   for j in dic[i]:
      pred_probtemp.append(j)
      y_testtemp.append(i)
      #ypred.append(j.tolist().index(j.max()))
      #ytrue.append(i)
   #dic2[names[i]]=[ypred,ytrue]

#everyone=[]


pred_prob = pred_probtemp[:]
y_test = y_testtemp[:]

probs =
    pd.concat([pd.DataFrame(pred_prob),pd.DataFrame(y_test)],axis=1)
probs.to_csv(dest+'ks_probabilities.csv',header=0,index=False)


outs = []
```

```
f1_micro=[]
f1_macro=[]

precision_micro=[]
precision_macro=[]

recall_micro=[]
recall_macro=[]
times = []

accuracy = []
rpt = []
f1s = []

for i in range(1,32,1):
   y_pred = []
   y_true = []
   for j in range(len(y_test)-i+1):
      if y_test[j]==y_test[j+i-1]:
         #y_pred_byprob=sum(np.array(pred_prob[j:j+i]),axis=0)
         y_pred_byprob=sum(np.array(pred_prob[j:j+i]),axis=0)
         y_pred.append(y_pred_byprob.tolist().index(y_pred_byprob.max()))
         y_true.append(y_test[j])
   #y_true = labelencod.inverse_transform(y_true)
   #y_pred = labelencod.inverse_transform(y_pred)

   outs.append([
         #'%.2f'%(f1_score(y_true,y_pred,average='micro')*100),
         float('%.3f'%(f1_score(y_true,y_pred,average='macro',zero_division=0))),
         float('%.3f'%(f1_score(y_true,y_pred,average='weighted',zero_division=0))),
         '-',
         #'%.2f'%(precision_score(y_true,y_pred,average='micro')*100),
         float('%.3f'%(precision_score(y_true,y_pred,average='macro',zero_division=0))),
         float('%.3f'%(precision_score(y_true,y_pred,average='weighted',zero_division=0))),
         '-',
         #'%.2f'%(recall_score(y_true,y_pred,average='micro')*100),
         float('%.3f'%(recall_score(y_true,y_pred,average='macro',zero_division=0))),
         float('%.3f'%(recall_score(y_true,y_pred,average='weighted',zero_division=0))),
         '-',
         float('%.3f'%(accuracy_score(y_true,y_pred))),
         ])
   f1s.append(f1_score(y_true,y_pred,average='macro',zero_division=0))
   f1_micro.append(f1_score(y_true,y_pred,average='micro',zero_division=0)*100)
   f1_macro.append(f1_score(y_true,y_pred,average='macro',zero_division=0)*100)
   precision_micro.append(precision_score(y_true,y_pred,average='micro',zero_division=0)*100)
   precision_macro.append(precision_score(y_true,y_pred,average='macro',zero_division=0)*100)
   recall_micro.append(recall_score(y_true,y_pred,average='micro',zero_division=0)*100)
   recall_macro.append(recall_score(y_true,y_pred,average='macro',zero_division=0)*100)
   accuracy.append(accuracy_score(y_true,y_pred)*100)
   times.append(i)
```

```
#newh = 'C:/Workspace/Python/mouse
    sets/Bogazici/browsing/combined_Train_LegalInternal/final_results_cv/3_timesIQR/0+/'+cata+'/

if i==1:
   if not os.path.exists(newh+str(i)+'/'):
      os.makedirs(newh+str(i)+'/')
   rpt = classification_report(y_true,y_pred,output_dict=True)
   #report=classification_report(y_true,y_pred,output_dict=True)
   pd.DataFrame(rpt).T.to_csv(newh+str(i)+'/'+str(i)+'ks_classification_report.csv',header=0,inde
   print (classification_report(y_true,y_pred))
   print (labelencod.inverse_transform(clf.classes_))

if (i+4)/5==int((i+4)/5):
   if not os.path.exists(newh+str(i)+'/'):
      os.makedirs(newh+str(i)+'/')
   cm_old = confusion_matrix(y_true, y_pred)
   cm = np.array(pd.DataFrame(cm_old)).tolist()
   cm_=[]
   for k in cm:
      temp=[]
      for j in k:
         temp.append(float('%.1f'%(100*j/sum(k))))
      cm_.append(temp)
   cm =mat(cm_.copy())

   report = classification_report(y_true,y_pred,output_dict=True)
   #report=classification_report(y_true,y_pred,output_dict=True)
   pd.DataFrame(report).T.to_csv(newh+str(i)+'/'+str(i)+'ks_classification_report.csv',header=0,i

   plt.figure(1,figsize=(10.24,8))
   ax1 = plt.axes()
   disp2 = ConfusionMatrixDisplay(confusion_matrix=cm)
   disp2.plot(cmap=plt.cm.Blues,include_values=False,ax=ax1)
   plt.savefig(newh+str(i)+'/'+str(i)+'ks_cm.png')
   plt.clf()

   plt.figure(3,figsize=(10.24,8))
   ax3 = plt.axes()
   disp3 =
       ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=heads)
   disp3.plot(cmap=plt.cm.Blues,include_values=True,ax=ax3)
   plt.savefig(newh+str(i)+'/'+str(i)+'ks_cm_newnew.png')
   plt.clf()

   plt.figure(4,figsize=(10.24,8))
   ax4 = plt.axes()
   disp4 =
       ConfusionMatrixDisplay(confusion_matrix=cm_old,display_labels=heads)
   disp4.plot(cmap=plt.cm.Blues,include_values=True,ax=ax4)
   plt.savefig(newh+str(i)+'/'+str(i)+'ks_cm_newnewnew.png')
```

```
        plt.clf()

        #np.set_printoptions(precision=2)
        cm_normalized = confusion_matrix(y_true,
            y_pred,normalize='true')
        cm_n = mat(np.round(cm_normalized,3))
        plt.figure(5,figsize=(10.24,8))
        ax5 = plt.axes()
        disp5 =
            ConfusionMatrixDisplay(confusion_matrix=cm_n,display_labels=heads)
        disp5.plot(cmap=plt.cm.Blues,include_values=True,ax=ax5)
        plt.savefig(newh+str(i)+'/'+str(i)+'ks_cm_newnewnew2.png')
        plt.clf()

        #np.set_printoptions(precision=2)
        cm_normalized2 = confusion_matrix(y_true,
            y_pred,normalize='pred')
        cm_n = mat(np.round(cm_normalized2,3))
        plt.figure(6,figsize=(10.24,8))
        ax6 = plt.axes()
        disp6 =
            ConfusionMatrixDisplay(confusion_matrix=cm_n,display_labels=heads)
        disp6.plot(cmap=plt.cm.Blues,include_values=True,ax=ax6)
        plt.savefig(newh+str(i)+'/'+str(i)+'ks_cm_newnewnew3.png')
        plt.clf()

print (pd.DataFrame(outs))


plt.figure(33,figsize=(10.24,8))
ax2 = plt.axes()
plot_importance(clf,ax=ax2,importance_type='gain')
plt.savefig(newh+'ks_FI.png')
plt.clf()

print (featurescore)

# plt.show()

plt.figure(34,figsize=(10.24,8))
#print (f1_macro,np.min(f1_macro))
#print
    (int(np.min([np.min(f1_macro),np.min(precision_macro),np.min(recall_macro),np.min(accuracy)])))
myyticks = [i for i in range(0,100,5)]
myxticks = [i for i in range(1,31,1)]
plt.yticks(myyticks)
#plt.xticks(myxticks)
plt.plot(times,f1_macro,label='f1_macro')
#plt.plot(times,f1_micro,label='f1_micro')
plt.plot(times,precision_macro,label='precision_macro')
```

```
    #plt.plot(times,precision_micro,label='precision_micro')
    plt.plot(times,recall_macro,label='recall_macro')
    #plt.plot(times,recall_micro,label='recall_micro')
    plt.plot(times,accuracy,label='accuracy')
    plt.legend()
    plt.grid()

    pd.DataFrame(outs).to_csv(newh+'ks_30times.csv',header=0,index=False)


    #return f1s
    plt.savefig(newh+'ks_overall.png')
    plt.clf()
    # print (type(rpt).__name__)
    #plot_importance(clf)
    #plt.show()
    return [f1s,pd.DataFrame(outs),probs]


def extractfeatures_keystroke():
    path = 'C:/Zhaoyi_Fan/Dataset/'
    dest = path
    if not os.path.exists(dest):
        os.makedirs(dest)
        print (dest,'created')

    file = path+'keystroke_example.csv'
    name = 'keystroke_example'

    #frequently used keys: k_keys
    f_keys = ['Space', 'Backspace', 'KeyI', 'KeyA', 'KeyN', 'KeyE',
        'KeyO', 'KeyU', 'KeyH', 'Enter', 'KeyG', 'KeyS', 'KeyC', 'KeyD',
        'KeyT', 'KeyL', 'KeyR', 'KeyY', 'KeyM', 'ShiftLeft', 'Digit1',
        'KeyB', 'Digit0', 'KeyZ', 'Digit2', 'KeyJ', 'KeyW', 'KeyX',
        'KeyP', 'KeyF', 'KeyV', 'Digit3', 'KeyK', 'Period', 'Digit9',
        'KeyQ', 'Digit4', 'Digit6', 'Digit7', 'Equal', 'Comma', 'Minus']
    x=[]
    y=[]
    X=[]
    Y=[]

    with open(file,'r',encoding='UTF-8') as f:
        data = np.array(pd.DataFrame(csv.reader(f))).tolist()[1:]
    out = []
    temp=[]
    the_data= []
    for i in data[1:]:

        if len(temp) == 0:
            temp.append(i)
```

```
   else:
      d=i[3]
      if int(i[2])-int(i[1])<10000 and int(i[2])-int(i[1])>0 and
          d!='' and d!='ArrowRight' and d!='ArrowLeft' and
          d!='ArrowDown' and d!='ArrowUp' and d!='keycode' and
          'Volume' not in d:

         if int(i[1])-int(temp[-1][1])<1000:
            temp.append(i)
         else:
            out.append(temp)
            X.append(temp)
            Y.append(name)
            temp=[]
            temp.append(i)
n=4
for d,name in zip(X,Y):
   if len(d)>n-1:
      for i in range(len(d)-n+1):
         boolean = True
         for boo in range(n-1):
            if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
               boolean=False
         if boolean:
            durations = []
            DD = []
            UU = []
            DU = []
            UD = []

            caps=[]
            shifts=[]

            for j in range(n):
               durations.append(int(d[i+j][2])-int(d[i+j][1]))
               caps.append(tobi(d[i+j][-1]))
               shifts.append(tobi(d[i+j][-2]))

            combo = list(combinations(np.arange(n),2))
            for c in combo:
               DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
               UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
               DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
               UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

            c = durations+DD+UU+DU+UD+caps+shifts

            my_feature=c[:]
            the_data.append(my_feature+[name])
pd.DataFrame(the_data).to_csv(dest+'keystroke_example_features.csv',header=0,index=False)
```

```
    print ('data extraction completed',len(the_data),'samples involved')


def combo_trainingset():
    path = 'C:/Zhaoyi_Fan/Dataset/Combo/training/'

    #keystroke################################################################

    #frequently used keys: k_keys
    f_keys = ['Space', 'Backspace', 'KeyI', 'KeyA', 'KeyN', 'KeyE',
         'KeyO', 'KeyU', 'KeyH', 'Enter', 'KeyG', 'KeyS', 'KeyC', 'KeyD',
         'KeyT', 'KeyL', 'KeyR', 'KeyY', 'KeyM', 'ShiftLeft', 'Digit1',
         'KeyB', 'Digit0', 'KeyZ', 'Digit2', 'KeyJ', 'KeyW', 'KeyX',
         'KeyP', 'KeyF', 'KeyV', 'Digit3', 'KeyK', 'Period', 'Digit9',
         'KeyQ', 'Digit4', 'Digit6', 'Digit7', 'Equal', 'Comma', 'Minus']
    x=[]
    y=[]
    X=[]
    Y=[]

    n=4
    dest = 'C:/Zhaoyi_Fan/Dataset/Combo/training/results'
    if not os.path.exists(dest):
        os.makedirs(dest)
        print (dest,'created')

    path_ks = path+'samples/'
    users = os.listdir(path_ks)
    for user in users[:]:
        file = path_ks+user+'/'+str(user)+'_keystroke_train.csv'
        name = user.split('_')[0]
        with open(file,'r',encoding='UTF-8') as f:
            data = np.array(pd.DataFrame(csv.reader(f))).tolist()
            out = []
            temp=[]
            the_data= []
            for i in data[1:]:

                if len(temp) == 0:
                    temp.append(i)
                    #print (0,i[3])
                else:
                    d=i[3]
                    if int(i[2])-int(i[1])<10000 and int(i[2])-int(i[1])>0 and
                         d!='' and d!='ArrowRight' and d!='ArrowLeft' and
                         d!='ArrowDown' and d!='ArrowUp' and d!='keycode' and
                         'Volume' not in d:

                        if int(i[1])-int(temp[-1][1])<1000:
```

```
          #if int(i[1])-int(temp[-1][1])<1000 and d in f_keys:

             temp.append(i)
             #print ('<1000',i[3])
          else:
             out.append(temp)
             X.append(temp)
             Y.append(name)
             temp=[]
             temp.append(i)
X_train,X_test,Y_train,Y_test =
    train_test_split(X,Y,test_size=0.2,stratify=Y,random_state=1)

x_train=[]
x_test=[]
y_train=[]
y_test=[]

for d,name in zip(X_train,Y_train):
   if len(d)>n-1:
      for i in range(len(d)-n+1):
         boolean = True
         for boo in range(n-1):
            if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
               boolean=False
         if boolean:
            durations = []
            DD = []
            UU = []
            DU = []
            UD = []

            caps=[]
            shifts=[]

            for j in range(n):
               durations.append(int(d[i+j][2])-int(d[i+j][1]))
               caps.append(tobi(d[i+j][-1]))
               shifts.append(tobi(d[i+j][-2]))

            combo = list(combinations(np.arange(n),2))
            for c in combo:

               DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
               UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
               DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
               UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

            c = durations+DD+UU+DU+UD+caps+shifts
            my_feature=c[:]
            x_train.append(my_feature)
            y_train.append(name)

print ('Keystroke Training Data extraction
    completed',len(y_train),'samples involved')
unwanted=[]
labelencod = preprocessing.LabelEncoder().fit(y_train)

x_train = np.delete(x_train,unwanted,axis=1)
y_train = labelencod.transform(y_train)

for d,name in zip(X_test,Y_test):
   if len(d)>n-1:
      for i in range(len(d)-n+1):
         boolean = True
         for boo in range(n-1):
            if int(d[i+1+boo][1])-int(d[i+boo][1])<=0:
               boolean=False
         if boolean:
            durations = []
            DD = []
            UU = []
            DU = []
            UD = []

            caps=[]
            shifts=[]

            for j in range(n):
               durations.append(int(d[i+j][2])-int(d[i+j][1]))
               caps.append(tobi(d[i+j][-1]))
               shifts.append(tobi(d[i+j][-2]))

            combo = list(combinations(np.arange(n),2))
            for c in combo:

               DD.append(int(d[c[1]+i][1])-int(d[c[0]+i][1]))
               UU.append(int(d[c[1]+i][2])-int(d[c[0]+i][2]))
               DU.append(int(d[c[1]+i][2])-int(d[c[0]+i][1]))
               UD.append(int(d[c[1]+i][1])-int(d[c[0]+i][2]))

            c = durations+DD+UU+DU+UD+caps+shifts

            my_feature=c[:]
            x_test.append(my_feature)
            y_test.append(name)

print ('Keystroke Testing Data extraction
    completed',len(y_test),'samples involved')
```

```
unwanted=[]

x_test = np.delete(x_test,unwanted,axis=1)
y_test = labelencod.transform(y_test)

pred_prob=[]
featurescore=[]

mychanges = []
theweights = []

try:
   print ('XGB fitting started')
   #cross_val = StratifiedKFold(n_splits=cvs)
   clf =
       XGBClassifier(objective='multi:softprob',use_label_encoder=False,
                eval_metric='mlogloss',tree_method='gpu_hist',
                learning_rate=0.2,n_estimators=600,
                max_depth = 4,min_child_weight=3,
                )
   #clf = lgb.LGBMClassifier()
   features_impts = []

   myweights,mychanges = myweight(y_train)
   #print (myweights,mychanges)
   class_weights =
       class_weight.compute_sample_weight(myweights,y_train)
   print ('XGB fitting started',len(y_train), 'samples involved')
   clf.fit(x_train,y_train,sample_weight=class_weights)
   print ('XGB fitting completed')
   print ('XGB predicting started')
   pre_temp = clf.predict_proba(x_test)
   print ('XGB predicting completed')
   featurescore = clf.feature_importances_.tolist()
   print ('ks:',clf.classes_)

   pred_prob = np.array(pre_temp)

   labelset = np.unique(y_test)
   names = labelencod.inverse_transform(labelset)


except Exception as e:
   print (e)
   #print (folder)
   return 'bad'

myweights_total,mychanges=myweight(y)
theweights.append(myweights_total)
head = np.array(theweights[0].keys())
```

```
print (head)
#heads = labelencod.inverse_transform(head).tolist()
heads = names.tolist()
weights=pd.DataFrame(theweights)
weights = np.array(weights).tolist()
weights.insert(0,heads)
weights=pd.DataFrame(weights)

dic_ks = {} #for computing the overall result
dic2 = {} #for computing each user's performance
#print (list(labelset))
labelset = sorted(list(labelset))
print (labelset)
for i in list(labelset):
   dic_ks[i]=[]
   #dic2[i]=[]
for i in range(len(y_test)):
   dic_ks[y_test[i]].append(pred_prob[i])
pred_probtemp = []
y_testtemp = []
for i in labelset:
   for j in dic_ks[i]:
      pred_probtemp.append(j)
      y_testtemp.append(i)

pred_prob_ks = pred_probtemp[:]
y_test_ks = y_testtemp[:]


#Mouse
    ######################################################################
path_mouse = path+'features/'
users = os.listdir(path_mouse)
x=[]
y=[]
for user in users[:]:
   file
       ='C:/Zhaoyi_Fan/Dataset/Combo/training/features/'+user+'/'+str(user)+'_mouse_train_features.
   name = user.split('_')[0]
   try:
      with open(file,'r',encoding='UTF-8') as f:
         d = np.array(pd.DataFrame(csv.reader(f))).tolist()
   except:
      print (file)

   with open(file,'r',encoding='UTF-8') as f:
      d = np.array(pd.DataFrame(csv.reader(f))).tolist()
      try:
         ss1 = float(d[0][0])
         ss2 = float(d[0][1])
      except:
         continue
      #d = np.delete(d,unwanted,axis=1)
      for i in d:
         temp = []
         if float(i[1])<1500 and float(i[3])/(float(i[0])-2)<0.2 and
             float(i[6])<2 and float(i[22])<2000 and float(i[20])<10
             and float(i[-5])/(float(i[0])-2)<0.2: # and
             float(i[3])/(float(i[0])-2)<0.3333:
            for j in i[:-1]:
               temp.append(float(j))
            x.append(temp)
            y.append(i[-1].split('_')[0])
print ('Mouse Data extraction completed',len(y),'samples involved')

try:
   x = np.delete(x,unwanted,axis=1)

   #labelencod = preprocessing.LabelEncoder().fit(y)

   y = labelencod.transform(np.array(y))

   x_train,x_test,y_train,y_test =
       train_test_split(x,y,test_size=0.2,stratify=y,random_state=1)

except Exception as e:
   print (e)

pred_prob=[]
featurescore=[]

mychanges = []
theweights = []

try:
   print ('XGB fitting started')
   #cross_val = StratifiedKFold(n_splits=cvs)
   clf =
       XGBClassifier(objective='multi:softprob',use_label_encoder=False,
                 eval_metric='mlogloss',tree_method='gpu_hist',
                 learning_rate=0.3,n_estimators=500,
                 max_depth = 4,min_child_weight=3,
                 )

   features_impts = []

   myweights,mychanges = myweight(y_train)
   class_weights =
       class_weight.compute_sample_weight(myweights,y_train)
   print ('XGB fitting started',len(y_train), 'samples involved')
```

```
        clf.fit(x_train,y_train,sample_weight=class_weights)
        print ('XGB fitting completed')
        print ('XGB predicting started')
        pre_temp = clf.predict_proba(x_test)
        print ('XGB predicting completed')
        featurescore = clf.feature_importances_.tolist()

        pred_prob = np.array(pre_temp)
        #print (clf.classes_)
        #return
        labelset = np.unique(y_test)
        names = labelencod.inverse_transform(labelset)

        y_pred2 = np.argmax(pred_prob, axis=1)

        y_pred2 = clf.classes_[y_pred2]


    except Exception as e:
        print (e)
        print (folder)
        return 'bad'

    #return
    myweights_total,mychanges=myweight(y)
    theweights.append(myweights_total)
    head = np.array(theweights[0].keys())
    #print (head)
    #heads = labelencod.inverse_transform(head).tolist()
    heads = names.tolist()
    weights=pd.DataFrame(theweights)
    weights = np.array(weights).tolist()
    weights.insert(0,heads)
    weights=pd.DataFrame(weights)

    dic_mouse = {} #for computing the overall result
    dic2 = {} #for computing each user's performance
    #print (list(labelset))

    for i in list(labelset):
        dic_mouse[i]=[]
        #dic2[i]=[]
    for i in range(len(y_test)):
        dic_mouse[y_test[i]].append(pred_prob[i])
    pred_probtemp = []
    y_testtemp = []
    for i in dic_mouse:
        for j in dic_mouse[i]:
            pred_probtemp.append(j)
            y_testtemp.append(i)
```

```
pred_prob_mouse = pred_probtemp[:]
y_test_mouse = y_testtemp[:]

outs = []
f1_micro=[]
f1_macro=[]

precision_micro=[]
precision_macro=[]

recall_micro=[]
recall_macro=[]
times = []

accuracy = []
rpt = []
f1s = []
print (clf.classes_)
print (len(y_test_mouse),len(y_test_ks))
ratio = len(y_test_ks)/len(y_test_mouse)

#return
for ks in range(1,30,1):
   for mo in range(1,30,1):
      y_pred,y_true = [],[]
      y_pr_k = []
      y_pr_m = []
      y_true_m = []
      y_true_k = []
      for cla in dic_mouse:
         prob_mouse = dic_mouse[cla]
         prob_ks = dic_ks[cla]
         ratio = len(prob_ks)/len(prob_mouse)

         if ratio>=1:
            for m in range(len(prob_mouse)-mo+1):
               y_pred_byprob_mouse =
                   sum(np.array(prob_mouse[m:m+mo]),axis=0)
               y_pr_m.append(y_pred_byprob_mouse.tolist().index(y_pred_byprob_mouse.max()))
               y_true_m.append(cla)
               for k in range(int(m*ratio),int(m*ratio+ratio),1):
                  if k+ks<len(prob_ks):
                     y_pred_byprob_ks =
                         sum(np.array(prob_ks[k:k+ks]),axis=0)
                     y_pr_k.append(y_pred_byprob_ks.tolist().index(y_pred_byprob_ks.max()))
                     y_true_k.append(cla)
                     y_pred_byprob =
                         np.array(y_pred_byprob_ks)+np.array(y_pred_byprob_mouse)
                     y_pred.append(y_pred_byprob.tolist().index(y_pred_byprob.max()))
                     y_true.append(cla)
         else:
            for k in range(len(prob_ks)-ks+1):
               y_pred_byprob_ks = sum(np.array(prob_ks[k:k+ks]),axis=0)
               y_pr_k.append(y_pred_byprob_ks.tolist().index(y_pred_byprob_ks.max()))
               y_true_k.append(cla)
               for m in range(int(k*ratio),int(k*ratio+ratio),1):
                  if m+mo<len(prob_mouse):
                     y_pred_byprob_mouse =
                         sum(np.array(prob_ks[m:m+mo]),axis=0)
                     y_pr_m.append(y_pred_byprob_mouse.tolist().index(y_pred_byprob_mouse.max()))
                     y_true_m.append(cla)

                     y_pred_byprob =
                         np.array(y_pred_byprob_ks)+np.array(y_pred_byprob_mouse)
                     y_pred.append(y_pred_byprob.tolist().index(y_pred_byprob.max()))
                     y_true.append(cla)


      outs.append([ks,mo,
         #'%.2f'%(f1_score(y_true,y_pred,average='micro')*100),
         float('%.2f'%(f1_score(y_true,y_pred,average='macro',zero_division=0)*100)),
         float('%.2f'%(f1_score(y_true,y_pred,average='weighted',zero_division=0)*100)),
         '-',
         #'%.2f'%(precision_score(y_true,y_pred,average='micro')*100),
         float('%.2f'%(precision_score(y_true,y_pred,average='macro',zero_division=0)*100)),
         float('%.2f'%(precision_score(y_true,y_pred,average='weighted',zero_division=0)*100)),
         '-',
         #'%.2f'%(recall_score(y_true,y_pred,average='micro')*100),
         float('%.2f'%(recall_score(y_true,y_pred,average='macro',zero_division=0)*100)),
         float('%.2f'%(recall_score(y_true,y_pred,average='weighted',zero_division=0)*100)),
         '-',
         float('%.2f'%(accuracy_score(y_true,y_pred)*100)),
         ])
      f1s.append(f1_score(y_true,y_pred,average='macro',zero_division=0))
      f1_micro.append(f1_score(y_true,y_pred,average='micro',zero_division=0)*100)
      f1_macro.append(f1_score(y_true,y_pred,average='macro',zero_division=0)*100)
      precision_micro.append(precision_score(y_true,y_pred,average='micro',zero_division=0)*100)
      precision_macro.append(precision_score(y_true,y_pred,average='macro',zero_division=0)*100)
      recall_micro.append(recall_score(y_true,y_pred,average='micro',zero_division=0)*100)
      recall_macro.append(recall_score(y_true,y_pred,average='macro',zero_division=0)*100)
      accuracy.append(accuracy_score(y_true,y_pred)*100)
      times.append([ks,mo])
      #newh = 'C:/Workspace/Python/mouse
          sets/Bogazici/browsing/combined_Train_LegalInternal/final_results_cv/3_timesIQR/0+/'+cata
      newh = 'C:/Zhaoyi_Fan/Dataset/Combo/training/results/combo/'
      # if not os.path.exists(newh+str(ks)+'_'+str(mo)+'/'):
      #     os.makedirs(newh+str(ks)+'_'+str(mo)+'/')
      if ks==mo==1:
         if not os.path.exists(newh+str(ks)+'_'+str(mo)+'/'):
               os.makedirs(newh+str(ks)+'_'+str(mo)+'/')
            rpt = classification_report(y_true,y_pred,output_dict=True)
            #report=classification_report(y_true,y_pred,output_dict=True)
            #pd.DataFrame(rpt).T.to_csv(newh+str(i)+'/'+str(i)+'_classification_report.csv',header=0,in
            print ('ks:',ks,'mouse:',mo)
            print (classification_report(y_true,y_pred))
            #print (labelencod.inverse_transform(clf.classes_))
            #print (classification_report(y_true_k,y_pr_k))
            #print (classification_report(y_true_m,y_pr_m))

         if (ks+2)/3==int((ks+2)/3) or (mo+2)/3==int((mo+2)/3):

            if not os.path.exists(newh+str(ks)+'_'+str(mo)+'/'):
               os.makedirs(newh+str(ks)+'_'+str(mo)+'/')

            pd.DataFrame(outs).to_csv(newh+str(ks)+'_'+str(mo)+'/'+str(ks)+'_'+str(mo)+'combo_30times_r
            cm_old = confusion_matrix(y_true, y_pred)
            cm = np.array(pd.DataFrame(cm_old)).tolist()
            cm_=[]
            for k in cm:
               temp=[]
               for j in k:
                  temp.append(float('%.1f'%(100*j/sum(k))))
               cm_.append(temp)
            cm =mat(cm_.copy())

            report = classification_report(y_true,y_pred,output_dict=True)
            #report=classification_report(y_true,y_pred,output_dict=True)
            pd.DataFrame(report).T.to_csv(newh+str(ks)+'_'+str(mo)+'/'+str(ks)+'_'+str(mo)+'_classifica

            plt.figure(1,figsize=(10.24,8))
            ax1 = plt.axes()
            disp2 = ConfusionMatrixDisplay(confusion_matrix=cm)
            disp2.plot(cmap=plt.cm.Blues,include_values=False,ax=ax1)
            plt.savefig(newh+str(ks)+'_'+str(mo)+'/'+str(ks)+'_'+str(mo)+'_cm.png')
            plt.clf()

            plt.figure(3,figsize=(10.24,8))
            ax3 = plt.axes()
            disp3 =
                ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=heads)
            disp3.plot(cmap=plt.cm.Blues,include_values=True,ax=ax3)
            plt.savefig(newh+str(ks)+'_'+str(mo)+'/'+str(ks)+'_'+str(mo)+'_cm_newnew.png')
            plt.clf()

            plt.figure(4,figsize=(10.24,8))
            ax4 = plt.axes()
            disp4 =
                ConfusionMatrixDisplay(confusion_matrix=cm_old,display_labels=heads)
            disp4.plot(cmap=plt.cm.Blues,include_values=True,ax=ax4)
```

```
            plt.savefig(newh+str(ks)+'_'+str(mo)+'/'+str(ks)+'_'+str(mo)+'_cm_newnewnew.png')
            plt.clf()

            #np.set_printoptions(precision=2)
            cm_normalized = confusion_matrix(y_true,
                y_pred,normalize='true')
            cm_n = mat(np.round(cm_normalized,3))
            plt.figure(5,figsize=(10.24,8))
            ax5 = plt.axes()
            disp5 =
                ConfusionMatrixDisplay(confusion_matrix=cm_n,display_labels=heads)
            disp5.plot(cmap=plt.cm.Blues,include_values=True,ax=ax5)
            plt.savefig(newh+str(ks)+'_'+str(mo)+'/'+str(ks)+'_'+str(mo)+'_cm_newnewnew2.png')
            plt.clf()

            #np.set_printoptions(precision=2)
            cm_normalized2 = confusion_matrix(y_true,
                y_pred,normalize='pred')
            cm_n = mat(np.round(cm_normalized2,3))
            plt.figure(6,figsize=(10.24,8))
            ax6 = plt.axes()
            disp6 =
                ConfusionMatrixDisplay(confusion_matrix=cm_n,display_labels=heads)
            disp6.plot(cmap=plt.cm.Blues,include_values=True,ax=ax6)
            plt.savefig(newh+str(ks)+'_'+str(mo)+'/'+str(ks)+'_'+str(mo)+'_cm_newnewnew3.png')
            plt.clf()

    print (pd.DataFrame(outs))
    pd.DataFrame(outs).to_csv(newh+'combo_train_30times_report.csv',index=False,header=False)


def combo_testing_set():

    #keystroke############################################################

    #frequently used keys: k_keys
    f_keys = ['Space', 'Backspace', 'KeyI', 'KeyA', 'KeyN', 'KeyE',
        'KeyO', 'KeyU', 'KeyH', 'Enter', 'KeyG', 'KeyS', 'KeyC', 'KeyD',
        'KeyT', 'KeyL', 'KeyR', 'KeyY', 'KeyM', 'ShiftLeft', 'Digit1',
        'KeyB', 'Digit0', 'KeyZ', 'Digit2', 'KeyJ', 'KeyW', 'KeyX',
        'KeyP', 'KeyF', 'KeyV', 'Digit3', 'KeyK', 'Period', 'Digit9',
        'KeyQ', 'Digit4', 'Digit6', 'Digit7', 'Equal', 'Comma', 'Minus']
    x=[]
    y=[]
    X=[]
    Y=[]
    x_train,y_train,x_test,y_test=[],[],[],[]

    n=4
```

```
newh = 'C:/Zhaoyi_Fan/Dataset/Combo/testing/results/combo/'
if not os.path.exists(newh):
   os.makedirs(newh)
   print (newh,'created')


training_path = 'C:/Zhaoyi_Fan/Dataset/Combo/training/features/'
testing_path = 'C:/Zhaoyi_Fan/Dataset/Combo/testing/features/'
test_path = testing_path
train_path = training_path
users = os.listdir(training_path)
for user in users:
   file = training_path+user+'/'+user+'_keystroke_train_features.csv'
   with open(file,'r',encoding='UTF-8') as f:
      df = pd.DataFrame(csv.reader(f)).iloc[:,:-1]
      data = np.array(df,dtype=np.float64)
      for i in data:
         x_train.append(i)
         y_train.append(user)

   file = testing_path+user+'/'+user+'_keystroke_test_features.csv'
   with open(file,'r',encoding='UTF-8') as f:
      df = np.array(pd.DataFrame(csv.reader(f)).iloc[:,:-1])
      data = np.array(df,dtype=np.float64)
   for i in data:
      x_test.append(i)
      y_test.append(user)


unwanted=[]
print ('Training Data extraction completed',len(y_train),'samples
    involved')
print ('Testing Data extraction completed',len(y_test),'samples
    involved')
labelencod = preprocessing.LabelEncoder().fit(y_train)
x_train = np.delete(x_train,unwanted,axis=1)
x_test = np.delete(x_test,unwanted,axis=1)

y_train = labelencod.transform(y_train)
y_test = labelencod.transform(y_test)


pred_prob=[]
featurescore=[]

mychanges = []
theweights = []

try:
   print ('XGB fitting started')
   #cross_val = StratifiedKFold(n_splits=cvs)
   clf =
       XGBClassifier(objective='multi:softprob',use_label_encoder=False,
                 eval_metric='mlogloss',tree_method='gpu_hist',
                 learning_rate=0.3,n_estimators=600,
                 max_depth = 5,min_child_weight=5,
                 )
   #clf = lgb.LGBMClassifier()
   features_impts = []

   myweights,mychanges = myweight(y_train)
   #print (myweights,mychanges)
   class_weights =
       class_weight.compute_sample_weight(myweights,y_train)
   print ('XGB fitting started',len(y_train), 'samples involved')
   clf.fit(x_train,y_train,sample_weight=class_weights)
   print ('XGB fitting completed')
   print ('XGB predicting started')
   pre_temp = clf.predict_proba(x_test)
   print ('XGB predicting completed')
   featurescore = clf.feature_importances_.tolist()
   print ('ks:',clf.classes_)

   pred_prob = np.array(pre_temp)

   labelset = np.unique(y_test)
   names = labelencod.inverse_transform(labelset)


except Exception as e:
   print (e)
   #print (folder)
   return 'bad'

myweights_total,mychanges=myweight(y)
theweights.append(myweights_total)
head = np.array(theweights[0].keys())
print (head)
#heads = labelencod.inverse_transform(head).tolist()
heads = names.tolist()
weights=pd.DataFrame(theweights)
weights = np.array(weights).tolist()
weights.insert(0,heads)
weights=pd.DataFrame(weights)

dic_ks = {} #for computing the overall result
dic2 = {} #for computing each user's performance
#print (list(labelset))
labelset = sorted(list(labelset))
print (labelset)
```

```
for i in list(labelset):
   dic_ks[i]=[]
   #dic2[i]=[]
for i in range(len(y_test)):
   dic_ks[y_test[i]].append(pred_prob[i])
pred_probtemp = []
y_testtemp = []
for i in labelset:
   for j in dic_ks[i]:
      pred_probtemp.append(j)
      y_testtemp.append(i)

pred_prob_ks = pred_probtemp[:]
y_test_ks = y_testtemp[:]


#Mouse
    #######################################################################

x=[]
y=[]
x_train,y_train,x_test,y_test=[],[],[],[]

training_path = 'C:/Zhaoyi_Fan/Dataset/Combo/training/features/'
testing_path = 'C:/Zhaoyi_Fan/Dataset/Combo/testing/features/'
users = os.listdir(training_path)
for user in users:
   file = training_path+user+'/'+user+'_mouse_train_features.csv'
   with open(file,'r',encoding='UTF-8') as f:
      df = pd.DataFrame(csv.reader(f)).iloc[:,:-1]
      data = np.array(df,dtype=np.float64)
      for i in data:
         x_train.append(i)
         y_train.append(user)

   file = test_path+user+'/'+user+'_mouse_test_features.csv'
   with open(file,'r',encoding='UTF-8') as f:
      df = np.array(pd.DataFrame(csv.reader(f)).iloc[:,:-1])
      data = np.array(df,dtype=np.float64)
   for i in data:
      x_test.append(i)
      y_test.append(user)

unwanted=[]
print ('Training Data extraction completed',len(y_train),'samples
    involved')
print ('Testing Data extraction completed',len(y_test),'samples
    involved')
labelencod = preprocessing.LabelEncoder().fit(y_train)
x_train = np.delete(x_train,unwanted,axis=1)
```

```
x_test = np.delete(x_test,unwanted,axis=1)

y_train = labelencod.transform(y_train)
y_test = labelencod.transform(y_test)

pred_prob=[]
featurescore=[]

mychanges = []
theweights = []

try:
   print ('XGB fitting started')
   #cross_val = StratifiedKFold(n_splits=cvs)
   clf =
       XGBClassifier(objective='multi:softprob',use_label_encoder=False,
                eval_metric='mlogloss',tree_method='gpu_hist',
                learning_rate=0.3,n_estimators=600,
                max_depth = 5,min_child_weight=3,
                )

   features_impts = []

   myweights,mychanges = myweight(y_train)
   class_weights =
       class_weight.compute_sample_weight(myweights,y_train)
   print ('XGB fitting started',len(y_train), 'samples involved')
   clf.fit(x_train,y_train,sample_weight=class_weights)
   print ('XGB fitting completed')
   print ('XGB predicting started')
   pre_temp = clf.predict_proba(x_test)
   print ('XGB predicting completed')
   featurescore = clf.feature_importances_.tolist()

   pred_prob = np.array(pre_temp)
   #print (clf.classes_)
   #return
   labelset = np.unique(y_test)
   names = labelencod.inverse_transform(labelset)

   y_pred2 = np.argmax(pred_prob, axis=1)

   y_pred2 = clf.classes_[y_pred2]

except Exception as e:
   print (e)
   print (folder)
   return 'bad'

#return
```

```
myweights_total,mychanges=myweight(y)
theweights.append(myweights_total)
head = np.array(theweights[0].keys())
#print (head)
#heads = labelencod.inverse_transform(head).tolist()
heads = names.tolist()
weights=pd.DataFrame(theweights)
weights = np.array(weights).tolist()
weights.insert(0,heads)
weights=pd.DataFrame(weights)

dic_mouse = {} #for computing the overall result
dic2 = {} #for computing each user's performance
#print (list(labelset))

for i in list(labelset):
   dic_mouse[i]=[]
   #dic2[i]=[]
for i in range(len(y_test)):
   dic_mouse[y_test[i]].append(pred_prob[i])
pred_probtemp = []
y_testtemp = []
for i in dic_mouse:
   for j in dic_mouse[i]:
      pred_probtemp.append(j)
      y_testtemp.append(i)

pred_prob_mouse = pred_probtemp[:]
y_test_mouse = y_testtemp[:]

outs = []
f1_micro=[]
f1_macro=[]

precision_micro=[]
precision_macro=[]

recall_micro=[]
recall_macro=[]
times = []

accuracy = []
rpt = []
f1s = []
print (clf.classes_)
print (len(y_test_mouse),len(y_test_ks))
ratio = len(y_test_ks)/len(y_test_mouse)

#return
for ks in range(1,30,1):
```

```
for mo in range(1,30,1):
   y_pred,y_true = [],[]
   y_pr_k = []
   y_pr_m = []
   y_true_m = []
   y_true_k = []
   for cla in dic_mouse:
      prob_mouse = dic_mouse[cla]
      prob_ks = dic_ks[cla]
      ratio = len(prob_ks)/len(prob_mouse)


      if ratio>=1:
         for m in range(len(prob_mouse)-mo+1):
            y_pred_byprob_mouse =
                 sum(np.array(prob_mouse[m:m+mo]),axis=0)
            y_pr_m.append(y_pred_byprob_mouse.tolist().index(y_pred_byprob_mouse.max()))
            y_true_m.append(cla)
            for k in range(int(m*ratio),int(m*ratio+ratio),1):
               if k+ks<len(prob_ks):
                  y_pred_byprob_ks =
                       sum(np.array(prob_ks[k:k+ks]),axis=0)
                  y_pr_k.append(y_pred_byprob_ks.tolist().index(y_pred_byprob_ks.max()))
                  y_true_k.append(cla)
                  y_pred_byprob =
                       np.array(y_pred_byprob_ks)+np.array(y_pred_byprob_mouse)
                  y_pred.append(y_pred_byprob.tolist().index(y_pred_byprob.max()))
                  y_true.append(cla)
      else:
         for k in range(len(prob_ks)-ks+1):
            y_pred_byprob_ks = sum(np.array(prob_ks[k:k+ks]),axis=0)
            y_pr_k.append(y_pred_byprob_ks.tolist().index(y_pred_byprob_ks.max()))
            y_true_k.append(cla)
            for m in range(int(k*ratio),int(k*ratio+ratio),1):
               if m+mo<len(prob_mouse):
                  y_pred_byprob_mouse =
                       sum(np.array(prob_ks[m:m+mo]),axis=0)
                  y_pr_m.append(y_pred_byprob_mouse.tolist().index(y_pred_byprob_mouse.max()))
                  y_true_m.append(cla)

                  y_pred_byprob =
                       np.array(y_pred_byprob_ks)+np.array(y_pred_byprob_mouse)
                  y_pred.append(y_pred_byprob.tolist().index(y_pred_byprob.max()))
                  y_true.append(cla)

   outs.append([ks,mo,
      #'%.2f'%(f1_score(y_true,y_pred,average='micro')*100),
      float('%.3f'%(f1_score(y_true,y_pred,average='macro',zero_division=0))),
      float('%.3f'%(f1_score(y_true,y_pred,average='weighted',zero_division=0))),
      '-',
   #'%.2f'%(precision_score(y_true,y_pred,average='micro')*100),
   float('%.3f'%(precision_score(y_true,y_pred,average='macro',zero_division=0))),
   float('%.3f'%(precision_score(y_true,y_pred,average='weighted',zero_division=0))),
   '-',
   #'%.2f'%(recall_score(y_true,y_pred,average='micro')*100),
   float('%.3f'%(recall_score(y_true,y_pred,average='macro',zero_division=0))),
   float('%.3f'%(recall_score(y_true,y_pred,average='weighted',zero_division=0))),
   '-',
   float('%.3f'%(accuracy_score(y_true,y_pred))),
   ])
f1s.append(f1_score(y_true,y_pred,average='macro',zero_division=0))
f1_micro.append(f1_score(y_true,y_pred,average='micro',zero_division=0)*100)
f1_macro.append(f1_score(y_true,y_pred,average='macro',zero_division=0)*100)
precision_micro.append(precision_score(y_true,y_pred,average='micro',zero_division=0)*100)
precision_macro.append(precision_score(y_true,y_pred,average='macro',zero_division=0)*100)
recall_micro.append(recall_score(y_true,y_pred,average='micro',zero_division=0)*100)
recall_macro.append(recall_score(y_true,y_pred,average='macro',zero_division=0)*100)
accuracy.append(accuracy_score(y_true,y_pred)*100)
times.append([ks,mo])

# if not os.path.exists(newh+str(ks)+'_'+str(mo)+'/'):
#    os.makedirs(newh+str(ks)+'_'+str(mo)+'/')
if ks==mo==1:
   if not os.path.exists(newh+str(ks)+'_'+str(mo)+'/'):
      os.makedirs(newh+str(ks)+'_'+str(mo)+'/')
   rpt = classification_report(y_true,y_pred,output_dict=True)
   #report=classification_report(y_true,y_pred,output_dict=True)
   #pd.DataFrame(rpt).T.to_csv(newh+str(i)+'/'+str(i)+'_classification_report.csv',header=0,in
   print ('ks:',ks,'mouse:',mo)
   print (classification_report(y_true,y_pred))
   #print (labelencod.inverse_transform(clf.classes_))
   #print (classification_report(y_true_k,y_pr_k))
   #print (classification_report(y_true_m,y_pr_m))

if (ks+2)/3==int((ks+2)/3) or (mo+2)/3==int((mo+2)/3):

   if not os.path.exists(newh+str(ks)+'_'+str(mo)+'/'):
      os.makedirs(newh+str(ks)+'_'+str(mo)+'/')

   pd.DataFrame(outs).to_csv(newh+str(ks)+'_'+str(mo)+'/'+str(ks)+'_'+str(mo)+'combo_30times_r
   cm_old = confusion_matrix(y_true, y_pred)
   cm = np.array(pd.DataFrame(cm_old)).tolist()
   cm_=[]
   for k in cm:
      temp=[]
      for j in k:
         temp.append(float('%.1f'%(100*j/sum(k))))
      cm_.append(temp)
   cm =mat(cm_.copy())
```

```
report = classification_report(y_true,y_pred,output_dict=True)
#report=classification_report(y_true,y_pred,output_dict=True)
pd.DataFrame(report).T.to_csv(newh+str(ks)+'_'+str(mo)+'/'+str(ks)+'_'+str(mo)+'combo_class

plt.figure(1,figsize=(10.24,8))
ax1 = plt.axes()
disp2 = ConfusionMatrixDisplay(confusion_matrix=cm)
disp2.plot(cmap=plt.cm.Blues,include_values=False,ax=ax1)
plt.savefig(newh+str(ks)+'_'+str(mo)+'/'+str(ks)+'_'+str(mo)+'combo_cm.png')
plt.clf()

plt.figure(3,figsize=(10.24,8))
ax3 = plt.axes()
disp3 =
    ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=heads)
disp3.plot(cmap=plt.cm.Blues,include_values=True,ax=ax3)
plt.savefig(newh+str(ks)+'_'+str(mo)+'/'+str(ks)+'_'+str(mo)+'combo_cm_newnew.png')
plt.clf()

plt.figure(4,figsize=(10.24,8))
ax4 = plt.axes()
disp4 =
    ConfusionMatrixDisplay(confusion_matrix=cm_old,display_labels=heads)
disp4.plot(cmap=plt.cm.Blues,include_values=True,ax=ax4)
plt.savefig(newh+str(ks)+'_'+str(mo)+'/'+str(ks)+'_'+str(mo)+'combo_cm_newnewnew.png')
plt.clf()

#np.set_printoptions(precision=2)
cm_normalized = confusion_matrix(y_true,
    y_pred,normalize='true')
cm_n = mat(np.round(cm_normalized,3))
plt.figure(5,figsize=(10.24,8))
ax5 = plt.axes()
disp5 =
    ConfusionMatrixDisplay(confusion_matrix=cm_n,display_labels=heads)
disp5.plot(cmap=plt.cm.Blues,include_values=True,ax=ax5)
plt.savefig(newh+str(ks)+'_'+str(mo)+'/'+str(ks)+'_'+str(mo)+'combo_cm_newnewnew2.png')
plt.clf()

#np.set_printoptions(precision=2)
cm_normalized2 = confusion_matrix(y_true,
    y_pred,normalize='pred')
cm_n = mat(np.round(cm_normalized2,3))
plt.figure(6,figsize=(10.24,8))
ax6 = plt.axes()
disp6 =
    ConfusionMatrixDisplay(confusion_matrix=cm_n,display_labels=heads)
disp6.plot(cmap=plt.cm.Blues,include_values=True,ax=ax6)
plt.savefig(newh+str(ks)+'_'+str(mo)+'/'+str(ks)+'_'+str(mo)+'combo_cm_newnewnew3.png')
plt.clf()
```

```
    print (pd.DataFrame(outs))
    pd.DataFrame(outs).to_csv(newh+'combo_test_30times_report.csv',index=False,header=False)


def draw_combo_testing():
    file =
        'C:/Zhaoyi_Fan/Dataset/Combo/training/results/Combo/combo_train_30times_report.csv'

    # or:

    # file =
        'C:/Zhaoyi_Fan/Dataset/Combo/testing/results/Combo/combo_test_30times_report.csv'
    with open(file,'r',encoding='UTF-8') as f:
        data = np.array(pd.DataFrame(csv.reader(f)))
        x = data[:,0].astype(float)
        y = data[:,1].astype(float)
        z = data[:,-1].astype(float)
    x_lin = np.linspace(min(x), max(x), len(np.unique(x)))
    y_lin = np.linspace(min(y), max(y), len(np.unique(y)))
    X, Y = np.meshgrid(x_lin, y_lin)
    points = np.column_stack((x, y))
    Z = griddata(points, z, (X, Y), method='linear')
    fig = plt.figure(figsize=(10, 6))
    ax = fig.add_subplot(111, projection='3d')
    norm = plt.Normalize(Z.min(), Z.max())
    colors = cm.jet(norm(Z))
    surf = ax.plot_surface(X, Y, Z, facecolors=colors, shade=False)
    mappable = cm.ScalarMappable(norm=norm, cmap=cm.jet)
    mappable.set_array(Z)
    plt.colorbar(mappable, ax=ax, label='Z')
    ax.set_xlabel('Number of keystroke samples')
    ax.set_ylabel('Number of mouse samples')
    ax.set_zlabel('F1-score')
    plt.savefig(figure_path1+'combo_test.png')
    plt.savefig(figure_path2+'combo_test.png')
    plt.show()



if __name__ == '__main__':
    #draw_combo_testing()

    # get the classicication result on testing set
    # it will cost a lot of time as it has 900 combination of keystroke
        and mouse:
    #combo_testing_set()


    #combo_trainingset()
```

```
  #extractfeatures_mouse()
  #extractfeatures_keystroke()
  #extractfeatures()
```

D.3.4 Python scripts for analysing mobile dynamics

List D.3.4 lists the Python scripts for mobile behaviour analysis used in Chapter 9.

```python
import sys
import numpy as np
import csv
import pandas as pd
import math
from numpy import *
import time
import datetime
import matplotlib.pyplot as plt
from sklearn import svm, linear_model, preprocessing
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.multiclass import OneVsOneClassifier, OneVsRestClassifier
from sklearn.svm import LinearSVC, SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
import random
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.metrics import make_scorer
from sklearn.metrics import r2_score
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import f1_score, precision_score, recall_score
from sklearn.metrics import accuracy_score
from sklearn import tree
from xgboost import XGBClassifier, plot_importance, DMatrix, cv
from xgboost import XGBRegressor
from sklearn.ensemble import VotingClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from collections import Counter
from sklearn.metrics import classification_report
from sklearn.utils import class_weight
import shutil
import os
from itertools import combinations
import gc
import scikitplot as skplt
import lightgbm as lgb
import seaborn as sns
from functools import reduce
```

```
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
pd.set_option('max_colwidth', 1000)
np.set_printoptions(suppress=True)

#Sample weights
def myweight(y=arange(19)):
   myweights = {}
   for j in np.unique(y):
      myweights[j]=len(y)/(y.tolist().count(j)*len(np.unique(y)))
   try:
      for i in mychanges:
         myweights[i[0]]=myweights[i[0]]*i[1]
   except:
      mychanges=[]
   return myweights, mychanges

#This function is for data pre-processing, main goal is to decode the
    data, as the author did not provide detailed information for their
    databet.
def decode_data():
   p = 'C:/Zhaoyi_Fan/Dataset/mmc/'
   dest = 'C:/Zhaoyi_Fan/Dataset/mmc_csv/'
   if not os.path.exists(dest):
      os.makedirs(dest)

   #Data file confirmed:
   d = np.load(p+ 'mmc1.npy', allow_pickle=True)
   print (len(d))
   print (len(d[1]))
   print (len(d[1][0]))

   if not os.path.exists(dest):
      os.makedirs(dest)
   files = os.listdir(p)
   people = []

   #Lable file confirmed:
   for f in files[4:]:
      d = np.load(p+ f, allow_pickle=True).astype(float)
      print (d)
      for i in d:
         people.append(i[0])
   print (np.unique(d[0][:, 0]), np.unique(d[0][:, 1]))
   print (len(d[0]))
   for i in range(len(d[0][0])):
      print (i, len(np.unique(d[0][:, i])))
   for i, j in zip(d[0][:, 0], d[0][:, 0]):
      if i==j:
         print (i)

   #Lable file matching confirmed:
   for f, l in zip(files[1:2], files[7:]):
      d = np.load(p+ f, allow_pickle=True).astype(float)
      label = np.load(p+ l, allow_pickle=True).astype(int)
      print (f, l)
      for the_d, the_label in zip(d, label):
         d0 = the_d
         label0 = the_label
         temp=[]
         for i in d0:
            if np.unique(i)[0]!=0:
               temp.append(i)
         print (len(temp), label0)

#Combine data with labels
def extract_mmc():
   combo = [[1, 7], [2, 8], [3, 5], [4, 6]]
   p = 'C:/Zhaoyi_Fan/Dataset/mmc/'
   dest = 'C:/Zhaoyi_Fan/Dataset/mmc_csv/'
   if not os.path.exists(dest):
      os.makedirs(dest)
   out=[]
   for c in combo:
      data = np.load(p+ 'mmc'+ str(c[0])+ '.npy',
          allow_pickle=True).astype(float)
      label = np.load(p+ 'mmc'+ str(c[1])+ '.npy',
          allow_pickle=True).astype(int)
      print ('mmc'+ str(c[0])+ '.npy', 'mmc'+ str(c[1])+ '.npy')


      user_data = []
      for the_d, the_label in zip(data, label):
         d0 = the_d
         label0 = the_label
         #temp=[]
         for i in d0:
            if np.unique(i)[0]!=0:
               temp=i[:].tolist()
               temp.append(label0[0])
               out.append(temp)
               user_data.append(temp)
      pd.DataFrame(user_data).to_csv(dest+
          'mmc'+str(c[0])+str(c[1])+'.csv', header=0, index=False)

   pd.DataFrame(out).to_csv(dest+ 'mmc_all.csv', header=0, index=False)
```

```
#Hyper-parameter tuning:
def para_cv(unwanted=[]):
  X, Y=[], []
  newh = 'C:/Zhaoyi_Fan/Dataset/mmc/results/testing/'
  if len(unwanted)==0:
    newh = newh
  else:
    t = ''
    for i in unwanted:
      t = t+ str(i)+ '_'
    newh = newh+ 'no'+ t+ '/'
  if not os.path.exists(newh):
    os.makedirs(newh)
  #train:
  for i in ['17', '28', '35']:
    file = 'C:/Zhaoyi_Fan/Dataset/mmc_csv/mmc'+ i+ '.csv'
    with open(file, 'r', encoding='UTF-8') as f:
      data =
          np.array(pd.DataFrame(csv.reader(f))).astype(float).tolist()
      for d in data:
        X.append(d[:-1])
        Y.append(d[-1])
  print ('extract train completed')
  labelencod = preprocessing.LabelEncoder().fit(Y)
  Y = labelencod.transform(Y)
  X = np.delete(X, unwanted, axis=1)
  params =[]
  #0.2 600, 4, 3,
  for learning_rate in [0.1, 0.2, 0.3]:
    for n_estimators in [400, 500, 600]:
      for max_depth in [3, 4, 5, 6]:
        for min_child_weight in [3, 4, 5]:
          cv_params = {
                'learning_rate': learning_rate,
                'n_estimators': n_estimators,
                'max_depth': max_depth,
                'min_child_weight': min_child_weight,
                'objective': 'multi:softprob',
                'use_label_encoder': False,
                'eval_metric': 'mlogloss',
                'tree_method': 'gpu_hist',
                  }
          params.append(cv_params)
  final_report=[]
  for cv_parmas in params:
    try:
      subpath = str(cv_parmas['learning_rate']*10)+ '_'+
          str(cv_parmas['n_estimators'])+ '_'+
          str(cv_parmas['max_depth'])+ '_'+
          str(cv_parmas['min_child_weight'])+ '/'
```

```
except:
   print (cv_parmas)
newh = 'C:/Zhaoyi_Fan/Dataset/mmc/results/training/cv/xgb/'
newh_xgb = newh+ subpath
if not os.path.exists(newh_xgb):
   os.makedirs(newh_xgb)
cross_val = StratifiedKFold(n_splits=5)
f1_scores = []
acc_scores = []
cms = []
i=1
start_xgb = time.time()
print ('Xgb cross validate starts')
for train_index, test_index in cross_val.split(X, Y):
   x_train, x_test = X[train_index], X[test_index]
   y_train, y_test = Y[train_index], Y[test_index]
   unwanted=[]
   x_train = np.array(x_train)
   y_train = np.array(y_train)
   x_test = np.array(x_test)
   y_test = np.array(y_test)
   clf_xgb = XGBClassifier(**cv_parmas)
   clf_xgb.fit(x_train, y_train)
   y_pred = clf_xgb.predict(x_test)
   f1 = f1_score(y_test, y_pred, average='macro', zero_division=0)
   acc = accuracy_score(y_test, y_pred)
   cm = confusion_matrix(y_test, y_pred)
   f1_scores.append(f1)
   acc_scores.append(acc)
   cms.append(cm.tolist())
   pd.DataFrame(cm).to_csv(newh_xgb+ str(i)+ '_cm.csv', header=0,
       index=False)
   i=i+ 1
result = np.sum(cms, axis=0)
pd.DataFrame(result).to_csv(newh_xgb+ str(11)+ '_cm.csv',
    header=0, index=False)
pd.DataFrame([f1_scores, acc_scores]).to_csv(newh_xgb+ str(12)+
    '_report.csv', header=0, index=False)
f1_mean = np.mean(f1_scores)
f1_std = np.std(f1_scores, ddof=1)
print (np.mean(f1_scores), np.std(f1_scores, ddof=1))
print ('Xgb cross validate completed. Time used:',
    np.round(time.time()-start_xgb, 2), '[learning_rate:',
    cv_parmas['learning_rate'], '] [n_estimators:',
    cv_parmas['n_estimators'], '] [max_depth:',
    cv_parmas['max_depth'], '] [min_child_weight:',
    cv_parmas['min_child_weight'], ']')
print ('----------------------')
final_report.append([cv_parmas['learning_rate'],
    cv_parmas['n_estimators'], cv_parmas['max_depth'],
            cv_parmas['min_child_weight'], f1_mean, f1_std])
    col = ['learning_rate', 'n_estimators', 'max_depth',
        'min_child_weight', 'f1-macro-mean', 'f1-macro-std']
    if not os.path.exists('C:/Zhaoyi_Fan/Dataset/mmc/'+
        'results/training/cv/xgb/GridSearchCV/'):
        os.makedirs('C:/Zhaoyi_Fan/Dataset/mmc/results/'+
            'training/cv/xgb/GridSearchCV/')
    pd.DataFrame(final_report, columns=col).to_csv('C:/Workspace/'+
        'Python/mmc/results/training/'+
        'cv/xgb/GridSearchCV/final_report2.csv', index=False)

#test on trained model:
def check_mmc_2(unwanted=[]):
    x_train, y_train, x_test, y_test=[], [], [], []
    newh = 'C:/Zhaoyi_Fan/Dataset/mmc/results/testing/0.2_600_6_3/'
    if len(unwanted)==0:
        newh = newh
    else:
        t = ''
        for i in unwanted:
            t = t+ str(i)+ '_'
        newh = newh+ 'no'+ t+ '/'
    if not os.path.exists(newh):
        os.makedirs(newh)
    dest = newh
    #train:
    for i in ['17', '28', '35']:
        file = 'C:/Zhaoyi_Fan/Dataset/mmc_csv/mmc'+ i+ '.csv'
        with open(file, 'r', encoding='UTF-8') as f:
            data =
                np.array(pd.DataFrame(csv.reader(f))).astype(float).tolist()
            for d in data:
                x_train.append(d[:-1])
                y_train.append(d[-1])
    print ('extract train completed')
    labelencod = preprocessing.LabelEncoder().fit(y_train)
    y_train = labelencod.transform(y_train)
    x_train = np.delete(x_train, unwanted, axis=1)
    #test:46
    for i in ['46']:
        file = 'C:/Zhaoyi_Fan/Dataset/mmc_csv/mmc'+ i+ '.csv'
        with open(file, 'r', encoding='UTF-8') as f:
            data =
                np.array(pd.DataFrame(csv.reader(f))).astype(float).tolist()
            for d in data:
                x_test.append(d[:-1])
                y_test.append(d[-1])
    y_test = labelencod.transform(y_test)
    x_test = np.delete(x_test, unwanted, axis=1)
    pred_prob, featurescore, mychanges, theweights=[], [], [], []
```

```
try:
   print ('XGB fitting started')
   clf = XGBClassifier(objective='multi:softprob',
       use_label_encoder=False,
                 eval_metric='mlogloss', tree_method='gpu_hist',
                 learning_rate=0.2, n_estimators=600,
                 max_depth = 6, min_child_weight=3,
                 )
   features_impts = []
   myweights, mychanges = myweight(y_train)
   class_weights = class_weight.compute_sample_weight(myweights,
       y_train)
   print ('XGB fitting started', len(y_train), 'samples involved')
   clf.fit(x_train, y_train, sample_weight=class_weights)
   print ('XGB fitting completed')
   print ('XGB predicting started')
   pre_temp = clf.predict_proba(x_test)
   print ('XGB predicting completed')
   featurescore = clf.feature_importances_.tolist()
   pred_prob = np.array(pre_temp)
   labelset = np.unique(y_test)
   names = labelencod.inverse_transform(labelset)
except Exception as e:
   print ('eeor', e)
   return 'bad'
myweights_total, mychanges=myweight(y_train)
theweights.append(myweights_total)
heads = names.tolist()
weights=pd.DataFrame(theweights)
weights = np.array(weights).tolist()
weights.insert(0, heads)
weights=pd.DataFrame(weights)
dic = {} #for computing the overall result
dic2 = {} #for computing each user's performance
for i in list(labelset):
   dic[i]=[]
for i in range(len(y_test)):
   dic[y_test[i]].append(pred_prob[i])
pred_probtemp = []
y_testtemp = []
for i in dic:
   for j in dic[i]:
      pred_probtemp.append(j)
      y_testtemp.append(i)

pred_prob = pred_probtemp[:]
y_test = y_testtemp[:]

probs = pd.concat([pd.DataFrame(pred_prob), pd.DataFrame(y_test)],
    axis=1)
```

```
probs.to_csv(dest+ 'probabilities_3.csv', header=0, index=False)
outs, f1_micro, f1_macro = [], [], []
precision_micro, precision_macro, recall_micro, recall_macro=[], [],
    [], []
times, accuracy, rpt = [], [], []
for i in range(1, 32, 1):
   y_pred = []
   y_true = []
   for j in range(len(y_test)-i+ 1):
      if y_test[j]==y_test[j+ i-1]:
         y_pred_byprob=sum(np.array(pred_prob[j:j+ i]), axis=0)
         y_pred.append(y_pred_byprob.tolist().index(y_pred_byprob.max()))
         y_true.append(y_test[j])
   outs.append([
         float('%.2f'%(f1_score(y_true, y_pred, average='macro',
             zero_division=0)*100)),
         float('%.2f'%(f1_score(y_true, y_pred, average='weighted',
             zero_division=0)*100)),
         '-',
         float('%.2f'%(precision_score(y_true, y_pred,
             average='macro', zero_division=0)*100)),
         float('%.2f'%(precision_score(y_true, y_pred,
             average='weighted', zero_division=0)*100)),
         '-',
         float('%.2f'%(recall_score(y_true, y_pred, average='macro',
             zero_division=0)*100)),
         float('%.2f'%(recall_score(y_true, y_pred,
             average='weighted', zero_division=0)*100)),
         '-',
         float('%.2f'%(accuracy_score(y_true, y_pred)*100)),
         ])
   #f1s.append(f1_score(y_true, y_pred, average='macro',
       zero_division=0))
   f1_micro.append(f1_score(y_true, y_pred, average='micro',
       zero_division=0)*100)
   f1_macro.append(f1_score(y_true, y_pred, average='macro',
       zero_division=0)*100)
   precision_micro.append(precision_score(y_true, y_pred,
       average='micro', zero_division=0)*100)
   precision_macro.append(precision_score(y_true, y_pred,
       average='macro', zero_division=0)*100)
   recall_micro.append(recall_score(y_true, y_pred, average='micro',
       zero_division=0)*100)
   recall_macro.append(recall_score(y_true, y_pred, average='macro',
       zero_division=0)*100)
   accuracy.append(accuracy_score(y_true, y_pred)*100)
   times.append(i)
   if i==1:
      if not os.path.exists(newh+ str(i)+ '/'):
         os.makedirs(newh+ str(i)+ '/')
```

```
        rpt = classification_report(y_true, y_pred, output_dict=True)
        pd.DataFrame(rpt).T.to_csv(newh+ str(i)+ '/'+ str(i)+
            '_classification_report.csv', header=0, index=True)
        print (classification_report(y_true, y_pred))
        print (labelencod.inverse_transform(clf.classes_))

    if (i+ 4)/5==int((i+ 4)/5):
        if not os.path.exists(newh+ str(i)+ '/'):
            os.makedirs(newh+ str(i)+ '/')
        cm_old = confusion_matrix(y_true, y_pred)
        cm = np.array(pd.DataFrame(cm_old)).tolist()
        cm_=[]
        for k in cm:
            temp=[]
            for j in k:
                temp.append(float('%.1f'%(100*j/sum(k))))
            cm_.append(temp)
        cm =mat(cm_.copy())

        report = classification_report(y_true, y_pred, output_dict=True)
        pd.DataFrame(report).T.to_csv(newh+ str(i)+ '/'+ str(i)+
            '_classification_report.csv', header=0, index=True)

        plt.figure(1, figsize=(10.24, 8))
        ax1 = plt.axes()
        disp2 = ConfusionMatrixDisplay(confusion_matrix=cm)
        disp2.plot(cmap=plt.cm.Blues, include_values=False, ax=ax1)
        plt.savefig(newh+ str(i)+ '/'+ str(i)+ '_cm.png')
        plt.clf()

        #np.set_printoptions(precision=2)
        cm_normalized2 = confusion_matrix(y_true, y_pred,
            normalize='pred')
        cm_n = mat(np.round(cm_normalized2, 3))
        plt.figure(6, figsize=(10.24, 8))
        ax6 = plt.axes()
        disp6 = ConfusionMatrixDisplay(confusion_matrix=cm_n,
            display_labels=heads)
        disp6.plot(cmap=plt.cm.Blues, include_values=True, ax=ax6)
        plt.savefig(newh+ str(i)+ '/'+ str(i)+ '_cm_newnewnew3.png')
        plt.clf()
print (pd.DataFrame(outs))
plt.figure(33, figsize=(10.24, 8))
ax2 = plt.axes()
plot_importance(clf, ax=ax2, importance_type='gain')
plt.savefig(newh+ '/FI.png')
plt.clf()
print (featurescore)
plt.figure(34, figsize=(10.24, 8))
myyticks = [i for i in range(0, 100, 5)]
```

```
    myxticks = [i for i in range(1, 31, 1)]
    plt.yticks(myyticks)
    plt.plot(times, f1_macro, label='f1_macro')
    plt.plot(times, precision_macro, label='precision_macro')
    plt.plot(times, recall_macro, label='recall_macro')
    plt.plot(times, accuracy, label='accuracy')
    plt.legend()
    plt.grid()
    pd.DataFrame(outs).to_csv(newh+ '30times.csv', header=0, index=False)
    plt.savefig(newh+ 'overall.png')
    plt.clf()

if __name__=='__main__':
    print ('let\'s go')
    extract_mmc()
    check_mmc_2()

    #para_cv()
    #check_mmc_2()
    #decode_data()
    #extract_mmc()
```