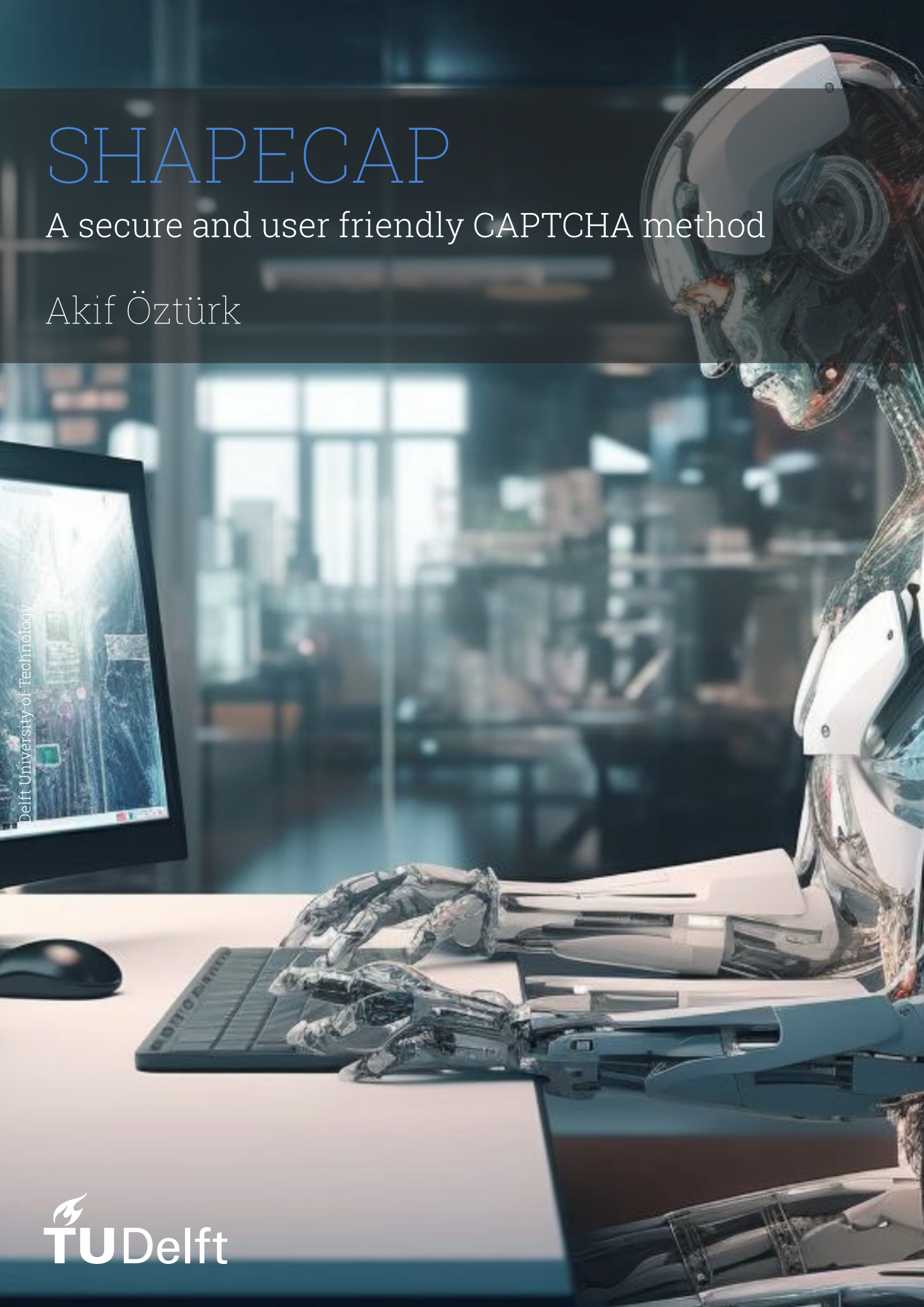
SHAPECAP

A secure and user friendly CAPTCHA method

Akif Öztürk



Delft University of Technology

TU Delft

SHAPECAP

A secure and user friendly CAPTCHA method

Akif Öztürk

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Wednesday October 16, 2024 at 13:00 CEST.

Student number:	5602173		
Project duration:	February 12, 2024 – October 16, 2024		
Thesis committee:	Prof. Mauro Conti	Full Professor	TU Delft, supervisor
	Prof. Georgios Smaragdakis,	Full Professor	TU Delft
	Prof. Fenia Aivaloglu,	Assistant Professor	TU Delft
	Riccardo Spolaor	External member	Daily supervisor

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

In the name of Allah, the Most Gracious, the Most Merciful.

I want to thank my father for making me the man I am today and always pushing me to the end, even at times when I was about to give up. I want to thank my mother for always supporting me and taking care of me till this age. Furthermore, I want to thank the rest of my family and close friends for entertaining me and making this journey fun. It would not have been possible without you.

I want to thank my supervisors Mauro Conti and Riccardo Spolaor for giving me the chance to work on this thesis project. Even though Mauro was not always physically present in Delft, he provided amazing insights on my research through online video calls. An even greater thank you to Riccardo, who joined the meetings with an 8-hour timezone difference and always had time to schedule an extra meeting when I needed his opinion or insights. Finally, I want to thank the remaining members of my thesis committee for providing feedback on my report and joining me for my upcoming thesis defence.

My inspiration for this thesis started during a lecture given by professor Conti. In this lecture, he briefly mentioned one of his own works: CAPTCHASTAR. It was a novel CAPTCHA method that used a dissorted image of stars, and I found it quite interesting. Later in the course, we received an assignment to contribute to any scientific article. I chose the CAPTCHASTAR paper and identified an attacking method that could potentially outperform the one described in it. Professor Conti confirmed my method, revealing that a previous paper had successfully defeated the CAPTCHASTAR challenge using a similar attacking method. Since then, I've realised how easily most CAPTCHAs can be broken and started thinking of new methods that could provide more security. However, every time I thought of a new method, a powerful attack was present. One day, I started thinking completely out of the box and had an idea that totally turned around the CAPTCHA challenge. We should make the challenge so hard for the user that it has to fail it in order to pass it. Let humans show their weakness in the challenge and use the fact that attackers complete the challenges "perfectly" without errors.

Akif Öztürk
Delft, October 2024

Abstract

CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart) have been in use for a long time on the web to block bots from accessing services. Many Different types of CAPTCHAs exist in various shapes and forms. As traditional CAPTCHAs became increasingly susceptible to attacks, mainly with the rise of artificial intelligence, there were efforts to enhance their complexity. As a side effect, this also increased the difficulty for legitimate users. Then attackers improved their methods, and the CAPTCHAs were broken once again, while the regular user keeps getting harder challenges. Over the years, this cycle has continued, leading to today's CAPTCHAs, which are not only insecure but also difficult to solve for regular users, defeating the purpose of having a CAPTCHA in the first place. What makes AI attacks so successful in CAPTHCA systems is their ability to perform the given task with high accuracy. AI bots are now faster and more successful in solving CAPTCHAs than humans.

In this paper, we propose SHAPECAP (Shape Analysis and Precision Exploiting CAPTCHA), a novel interactive CAPTCHA system that aims for a user-friendly solution that is also secure against AI-backed attackers. The design of our solution involves a canvas on which shapes are moving randomly. The user can choose a specific shape at the start of the challenge and use the mouse pointer to follow it. There will be small variations of the chosen shape in order to confuse the user. If the user follows the correct shape, they will complete the challenge faster, while following the variation shapes will take longer. Based on the mouse movement data collected throughout the challenge, we can confirm whether the user is a human or a bot. Our aim is to find a pattern that shows humans are more likely to follow irregular shapes. We exploit the precision of machines and use it against them while capitalising on the natural tendency of humans to make errors.

Contents

1

Introduction

1.1. Background

CAPTCHAs, or “Completely Automated Public Turing tests to tell Computers and Humans Apart,” play a crucial role in modern cybersecurity. By distinguishing between human users and automated bots, they safeguard online services from various forms of malicious activity.

The most important value of CAPTCHAs is their ability to prevent automated attacks. Malicious actors often use bots to perform various types of attacks, such as spamming, data theft, scraping sensitive information, and launching Distributed Denial of Service (DDoS) attacks. Online services can significantly reduce the risk of automated attacks by requiring users to complete a CAPTCHA before proceeding.

CAPTCHAs are useful for spam protection. Automated bots can flood comment sections, forums, and live chats with unwanted content, worsening the user experience and damaging the reputation of a website. CAPTCHAs act as a barrier, allowing only verified humans to submit content.

In addition to spam protection, CAPTCHAs are also useful in protecting user accounts and personal information. Many online communication services, such as email providers and social media platforms, require a personal account to function properly. CAPTCHAs are used in the registration process to prevent bots from creating fake accounts. Malicious actors can use these fake accounts to spread misinformation, carry out phishing attacks, and send spam emails.

Furthermore, CAPTCHAs are essential for securing online transactions. E-commerce websites and online banking services use CAPTCHAs to verify that transactions are initiated by humans. This helps prevent fraudulent activities, such as automated ticket scalping and unauthorised access to financial accounts.

Furthermore, CAPTCHAs improve web application security by preventing brute force attacks. An attacker uses automated scripts in a brute force attack to repeatedly guess login credentials until they find the correct combination. By implementing CAPTCHAs into the login process, websites can block these automated attempts and provide a secure authentication system for their regular users.

Although CAPTCHAs enhance online security, they must be designed with user experience in mind. Poorly designed CAPTCHAs can frustrate users and result in a negative impression of the website. Therefore, it is essential to find a balance between security and usability for the user. To establish this, modern CAPTCHAs are developed in different ways, such as image-based, video-based, math-based, interaction-based, and many more. All these techniques aim to improve security while maintaining user friendliness.

1.2. Problem statement

Despite their widespread use and importance in cybersecurity, current CAPTCHA methods are not secure against most types of attacks, especially when confronted with modern machine learning (ML) attacks. Traditional CAPTCHAs were initially designed to create challenges that were hard to solve for machines but easy for humans. However, advancements in ML have increased in such a way that challenges can be solved by machines as well. Consequently, it became increasingly difficult for CAPTCHAs to effectively distinguish between human users and automated bots. This created a need to increase the challenge complexity, resulting in CAPTCHAs that are now difficult to solve for humans. Meanwhile, technology keeps advancing, enabling machines to solve even the newly complex CAPTCHAs.

Text-based CAPTCHAs, for example, use distorted text that humans could easily read but computers struggled to interpret. Figure 1.1 shows an example of a text-based captcha called GIMPY. At the time, the distorted text used in these CAPTCHAs was considered unrecognisable by Optical Character Readers (OCR), making it an effective method for distinguishing between human users and automated bots. Modern OCR systems are able to recognise and decode distorted text with high precision. Many text-based CAPTCHAs are no longer effective, as bots can now solve these challenges with a success rate comparable to that of human users. Once again, the need to create more complex distortions to counteract OCR advancements often results in CAPTCHAs that are difficult for humans to solve, leading to a negative user experience.

Figure 1.1: Gimpy, a text-based CAPTCHA

Image-based CAPTCHAs, such as those used in Google's reCAPTCHA v2, also face important challenges. These CAPTCHAs require users to identify objects within images. Figure 1.2 shows an example of Google's image-based CAPTCHA challenge. Advancements in computer vision and object detection algorithms have made it possible for bots to solve these challenges with high accuracy, threatening the security of these CAPTCHAs. Once again, increasing the complexity of these CAPTCHAs can frustrate users, leading to a negative user experience.

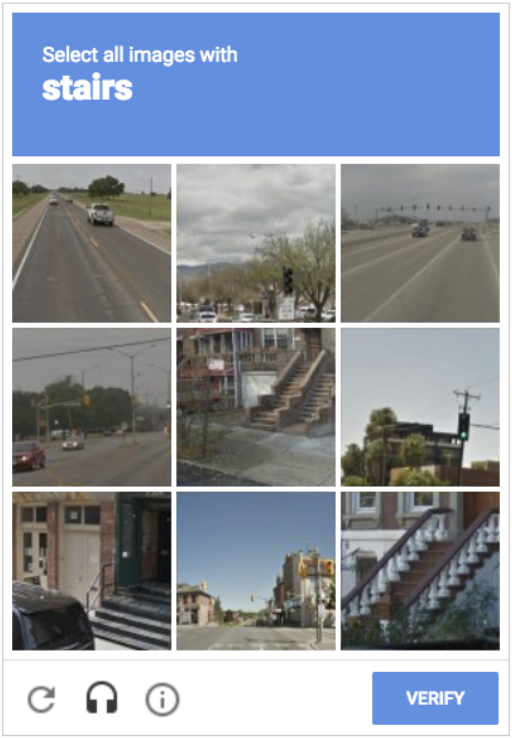


Figure 1.2: Google ReCAPTCHA v2, an image-based CAPTCHA

The rapid advancements in ML and deep learning have exposed significant limitations in traditional CAPTCHA methods. Most CAPTCHAs methods are increasingly vulnerable to sophisticated ML attacks, which can solve these challenges with high accuracy. Additionally, the need to enhance security often leads to more complex CAPTCHAs, negatively impacting the user experience. Therefore, there is a need for innovative CAPTCHA designs that can effectively balance complexity and usability for human users.

1.3. Objective

Artificial intelligence (AI) has impacted many fields by providing unmatched levels of accuracy, consistency, and speed. ML models, when trained properly, can easily identify patterns and make decisions with surprising accuracy, often outperforming humans. These properties allow for AI and ML to be highly effective in solving complex problems and automating tasks. However, this presents a challenge for modern CAPTCHA systems, as modern AI and ML attacks can easily bypass many of the challenges designed to prevent automated attacks.

In contrast, especially when faced with a complex task, humans are prone to make errors. Unlike machines, humans are not precise and may struggle with tasks that require prolonged attention and accuracy. While these errors can be seen as a limitation, they also provide a unique chance to distinguish between human users and automated bots. Understanding and using humans' natural mistake tendencies allows us to develop CAPTCHA systems that are more secure against AI and ML attacks.

Building on these insights, the goal of this research is to explore the development of new CAPTCHA methods that capitalise on the precision of AI and use the natural tendency of humans to make errors in complex tasks. However, a new challenge that arises with this approach is to prevent AI from intentionally "worsening" its performance in order to mimic human behaviour. We must consider this when designing new challenges. We must clearly distinguish between genuine human errors and artificial errors caused by AI.

1.4. Structure

This paper has the following structure: In Section 2, we provide a comprehensive review of relevant previous research on CAPTCHA methods, focussing on their evolution, types, and attacks against them. In Section 3, we present a detailed description of our proposed CAPTCHA method, including its design principles, implementation details, and the security parameters that can be adjusted to optimise performance. A technical overview of the implementation, deployment and data collection of the CAPTCHA system is given in 4. The data analysis and feature extraction processes are elaborated in Section 5, followed by the development, evaluation, and results of our machine learning models in Section 6. In Section 7, we interpret the findings, discuss the strengths and limitations of our method, and suggest areas for future research. Finally, in Section 8, we summarise the research, highlighting the broader implications of our work.

2

Literature Review

In this section, we discuss relevant previous research on the topic. Concretely, we will look at the various state-of-the-art CAPTCHAs to date, as well as the various attacks on these CAPTCHAs. Finally, we introduce the idea of geometric shape regularity, which are the building blocks of our CAPTCHA challenge.

2.1. Evolution of CAPTCHA types and attacks

Traditional CAPTCHAs started simple in a text-based format. The invention of various attacks, such as machine learning attacks, segmentation attacks, and human solver relay attacks, has resulted in many types of CAPTCHAs trying to mitigate these attacks. Many of these types can be solved by bots with a high percentage of success [13]. The advancement of machine learning is the cause of this high percentage. Attackers make use of deep learning, neural networks, and support vector machines to break the CAPTCHA challenge [17].

2.1.1. Text-based CAPTCHAs

Text-based CAPTCHAs were the first CAPTCHAs created. ReCAPTCHA (2008) [1] is the most known 2D text-based CAPTCHA. Figure 2.1 shows an example of the ReCAPTCHA challenge. It consists of an image of a word that is distorted and has a contrasting background. Text-based CAPTCHAs have been compromised through various methods [7, 18, 4]. One of them by Goodfellow et al. [8] breaks reCAPTCHA with an accuracy of 99.8% on their hardest category using deep convolutional neural networks. Attempts to improve text-based CAPTCHAS through 3D images (e.g., Imsamai and Phimoltares[15]) or animation have not resisted being broken either.

Figure 2.1: ReCAPTCHA (2008)

2.1.2. Image-based CAPTCHAs

Image-based CAPTCHAs exist in many shapes and forms, such as clickable images (e.g., Implicit CAPTCHA (2005) [3]), sliding images (e.g., What's Up CAPTCHA? (2009) [11]), selecting images (e.g., Google's No CAPTCHA reCAPTCHA (2014) [9]), and interactive image-based CAPTCHAs (e.g., CAPTCHaStar (2015) [5]). Figure 2.2 shows an example of the CAPTCHaStar challenge. This variety in types has not stopped many of these CAPTCHAs from being broken as well through various means, from random guessing and relay attacks to machine learning techniques [19, 12, 14].

Figure 2.2: CAPTCHaStar Challenge

2.1.3. Behaviour-based CAPTCHAs

In recent years, behaviour-based CAPTCHAs have gained increased interest. Behaviour-based CAPTCHAs make use of the behaviour of the user, such as mouse clicks and metadata on the user, to determine to a certain degree the validity of the traffic. Notable mentions of behaviour-based CAPTCHAs are Google's No CAPTCHA reCAPTCHA (2014) [9] and Invisible reCAPTCHA (2017) [10]. Figure 2.3 shows an example of the No CAPTCHA reCAPTCHA challenge, which just seems like a simple clickable box. According to Guerar et al. [13], this increased interest is the consequence of conventional CAPTCHAs being broken. Behaviour-based CAPTCHAs have not been resistant to attacks either. Simulations of human behaviour have been successful [19] in breaking several currently used CAPTCHAs from Tencent, Facebook, and Google. Akrout et al. broke No CAPTCHA reCAPTCHA using machine learning techniques [2]. Invisible reCAPTCHA is not broken yet but is feared to be vulnerable with the newest generation of bots being able to simulate traffic [13, 17]. There are many more types of CAPTCHAs available, but we wanted to highlight the most important ones.

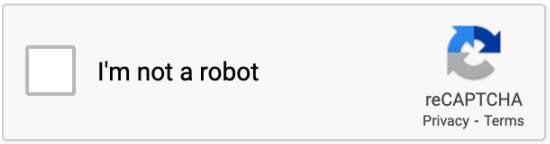


Figure 2.3: No CAPTCHA reCAPTCHA Challenge

2.2. Geometric Shape Regularity and Human Singularity

The paper "Sensitivity to geometric shape regularity in humans and baboons" by Sablé-Meyer et al.[16] presents important findings that could contribute to the design of new CAPTCHA methods. The study demonstrates that all human groups, regardless of age, culture, or education, show an intuition of geometric regularity, like right angles, parallelism, and symmetry. Machine learning models have not been able to replicate this geometric intuition of humans, while the models were able to successfully replicate the behaviour of the baboons on the same experiment. This suggests that automated bots that do not have this sensitivity to geometric regularities may struggle with tasks designed around these principles. The fact that this intuition is present in varying human groups regardless of age, culture, or education makes the CAPTCHA more user-friendly and accessible to everyone. Using these findings, our aim is to develop a user-friendly and

secure CAPTCHA method.

3

ShapeCAP overview

3.1. Design Concept

3.1.1. Shape Selection

At the start of the challenge, the user is presented with three quadrilateral shapes to choose from. This selection is randomised, and there are a total of nine different shapes available. A quadrilateral shape is any polygon with four sides and four corners, where the interior angles of a quadrilateral add up to 360 degrees. Common examples of quadrilaterals include squares, rectangles, trapezoids, and parallelograms. The ”perfect” regular quadrilateral is a square because it holds all properties of parallelism, equal sides, equal angles, and right angles. Each property that a shape loses makes it less regular. Figure 3.1 shows an example of all shapes ordered from regular to irregular. According to the geometric regularity effect, humans detect variations easier on more regular shapes [16]. The user selects one of these shapes, from now on referred to as the ”chosen shape,” to proceed to the next stage. The decision to offer the user a choice instead of just giving the user shape is made so that we can already get a first indication of the humanness of the user. At this stage, we expect human users to select more regular shapes, but a machine can be programmed to take this into consideration as well. So the shape choice is considered an initial indication rather than a definitive decision. Figure 3.2 displays a snapshot of the shape-choise stage.

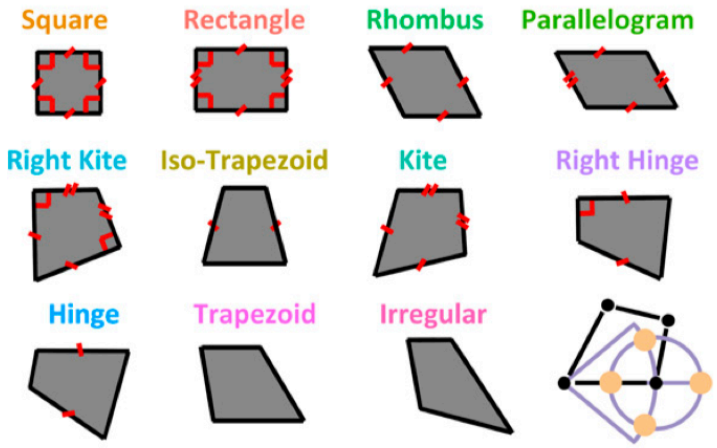


Figure 3.1: Quadrilateral shapes ordered according to geometrical regularies [16]

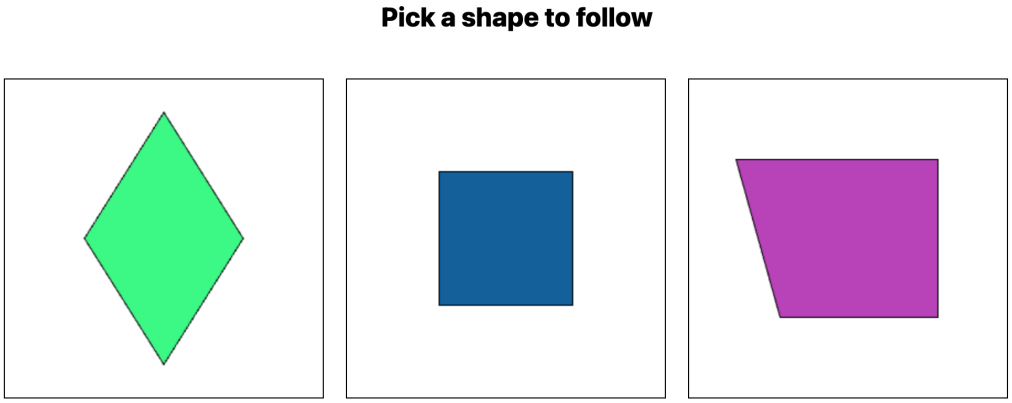


Figure 3.2: Shape choise stage

3.1.2. Shape Variation

In the next stage, the actual challenge will start so the remaining necessary shapes are created. First, a certain number of variation shapes are generated from the chosen shape. A variation is defined as a shape in which three points coincide with the chosen shape, but the fourth point is slightly offset, resulting in a slight variation of the original shape. In Figure 3.3, we can see the difference between the chosen shape and a variation shape. The bottom left corner of the variation is slightly moved to the right, resulting in an inclined left edge. The degree of offset is determined by the radius of a circle drawn with its centroid at the fourth point of the chosen shape. Then, a random point on this circle is chosen as the new fourth point. The radius, and therefore the degree of offset, can be tuned to make variations easier or harder to detect in the challenge.

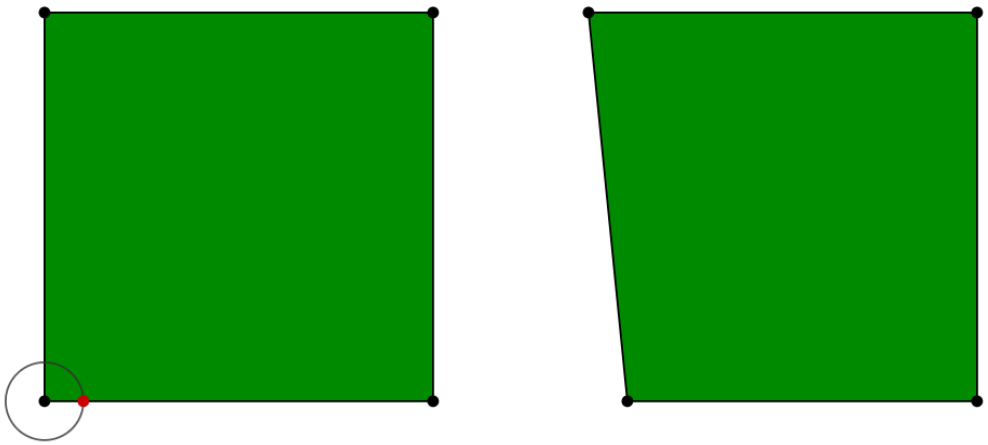


Figure 3.3: Normal shape and a possible variation where the bottom left corner is slightly pulled to the right

3.1.3. Object Shape

Finally, an obstacle shape is created. The obstacle shape is larger than the other shapes, has a fixed position, and is coloured black. The obstacle shape hides any shapes passing through it, making the shapes invisible to the user. This adds another element of complexity to the challenge. The main goal of the obstacle is to lure the human users to make more errors. Even if the user is following the right chosen shape from the start, once it passes through the obstacle and becomes invisible, the user must re-assess the situation and choose the right shape once it comes out of the obstacle. An even greater confusion is created when multiple shapes enter and leave the obstacle at the same time.

3.1.4. Movement and Tracking

A snapshot of the start of this stage with all shapes (chosen, variation, obstacle) drawn is shown in Figure 3.4. For demonstration purposes, we coloured the variation shapes with a slightly lighter green colour than the chosen shape in order to identify the chosen shape. In the actual challenge, all shapes will have the same colour as can be seen in a snapshot of this stage from the user perspective in Figure 3.5.

Once all shapes are created, the original shape and its variations start moving across the canvas. We use Simplex noise to calculate the movement path of each shape. Simplex noise is a method for constructing an n-dimensional noise function. The reason we use Simplex noise is due to its characteristic of producing "fixed" randomness [REF]. Truly random movements can create unexpected paths where shapes move unpredictably, which can be disorienting and frustrating for the user. Simplex noise, on the other hand, generates a well-defined and continuous gradient across the canvas. This results in more natural, smooth movements of the shapes, improving the user experience while maintaining the challenge's complexity. By using Simplex noise, we ensure that the movement of the shapes is unpredictable enough to pose a challenge but also consistent enough to be trackable, finding a balance between difficulty and user-friendliness. To better understand the concept of simplex noise, the trajectories of the shapes are illustrated in Figure 3.4. For demonstration purposes, we opted for a smaller number of points (n = 8) to keep the canvas clean and provide a basic understanding of simplex noise. In the actual challenge, we set this number to n = 100, which means each shape traverses through 100 points in the canvas. Once a shape reaches the final point, it will continue from the first point, creating a loop. The number of points is considered a security parameter and can be tuned accordingly in order to extend or shorten the movement paths of the shapes. This will be covered in more detail in 3.2.

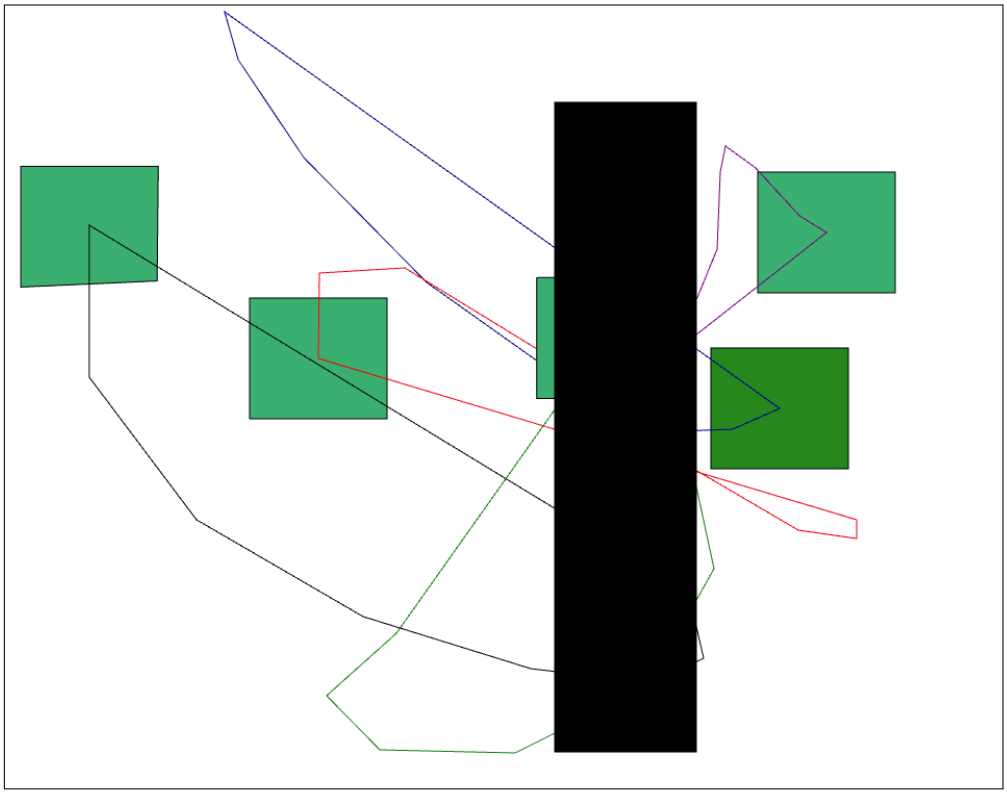


Figure 3.4: Challenge with hints

3.1.5. Scoring System

The user's task is to follow the chosen shape with their cursor or finger (if on a mobile device). The reward for following the chosen shape is that the user completes the challenge faster by collecting more points per second. Conversely, if the user follows a variation shape, they are penalised by collecting fewer points per second and completing the challenge slower. The user passes the challenge once they reach a threshold score. It is important to note that the user can still pass the challenge by following a variation or by switching between different shapes during the challenge, albeit at a slower pace.

The server receives the user's mouse movement data and the challenge seed upon completion of the challenge. Using the seed, the server can reconstruct the entire challenge scenario, including all shapes, their variations, obstacles, and moving patterns. This data, together with the mouse movement data, serves as the input for a machine-learning model. The model processes this input and generates an output that classifies the user as either a human or a bot. After this stage, the challenge is completed. Users that are classified as humans can continue with their process, while users classified as bots are denied access or allowed to repeat the challenge.

3.2. Security parameters

In order to design a good CAPTCHA, there should be randomness and a uniform distribution in all parameters of the challenge[6]. For some parameters, we did not achieve this and instead left them at a fixed value. For others, we have established randomness within a range of values. We set a minimum and maximum value, then randomly generate a value within this range. These values determine the uniqueness of each challenge. For certain parameters, a higher value can make the challenge more difficult or easy. Therefore, we can tune the range of values to find an optimal balance between challenge difficulty and user friendliness. The following is an explanation of the parameters and their impact on the challenges.

- **NShapes:** The number of initial shapes to choose from.
- **NVariations:** The challenge displays a certain number of variations. More variations will make the challenge more difficult, while fewer variations will make it easier. The number of variations must be at least one.
- **VariationRadius:** When creating a variation, the radius corresponds to the 4th point. A larger radius means that the 4th point has a larger offset, making the variations easier to detect. A smaller radius will make variations harder to detect.
- **Speed:** Determines the speed at which the shapes move from one point to the other. Larger speeds will make the challenge harder to complete while slower speeds can give the user more time to detect and follow the right shapes.
- **ShapeSize:** Sets the sizes of the shapes. Larger shapes can be easier to follow, while also giving a higher margin for error. Smaller shapes will be harder to follow precisely. This also applies to the object shape, where a larger object can make the challenge harder and a smaller object will make the challenge easier.
- **Path points:** This determines the amount of points that are present in the path of the shape. If the amount of points is less than the shape will enter a loop faster, making the movements more predictable. We aim to pick a high number such that the challenge is completed before the shapes enter a loop so that the movement cannot be predicted from previous knowledge. However, observing the user behaviour when the shapes are looping could be another interesting topic.
- **MaxScore** The maximum value needed to be reached in order to complete the challenge. A higher max score will result in longer challenges, while a lower max score can create faster challenges.
- **IncrementScore** The score gained by following the chosen/variation shapes. Following the chosen shape should reward users and result in more score progression

compared to following variations. Incrementing the score by larger amounts will result in completing the challenge faster, while retrieving lower scores will result in slower challenges.

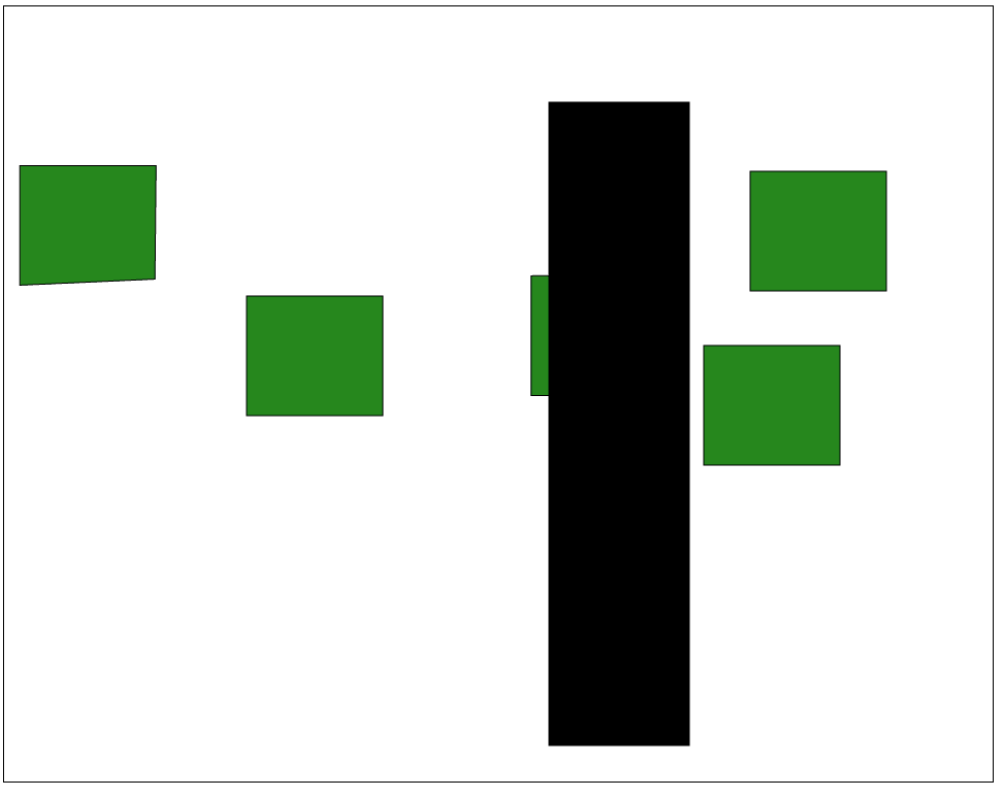


Figure 3.5: Complete Challenge from the user's perspective

4

Implementation

4.1. Technical Details

The CAPTCHA challenge is designed using a technology stack including React, Tailwind CSS, and TypeScript. The main logic of the CAPTCHA is coded using the HTML5 canvas. This is where all the shapes are drawn and animated. A seeded pseudorandom number generator (PRNG) is used to generate a unique challenge for each user. The numbers generated with the PRNG determine the sizes, speed, quantity, positions, and paths of the shapes. The use of a seeded PRNG ensures that each challenge can be accurately recreated on the server side using the same number generator. We will provide a general overview of the most important functions, but a full source code is publicly available on github. [1].

One of the most important classes is the quadrilateral shape class. The quadrilateral class contains all the properties and methods to create, move, and draw a shape on the canvas. All shape classes extend from this class to create their own specific quadrilateral shape. In the main class for the app logic, these shape classes are then used to create new shapes, variations, and obstacles accordingly. Each shape moves simultaneously to its next point in the path, which is also contained in the shape class. The score is tracked by comparing the position of the mouse pointer and the position of the shapes on the canvas. Points are added based on what type of shape is followed. There is a single score counter that is incremented, and when it reaches the threshold value, the challenge is completed. The user is presented with a final screen where it can see the time of completion and an option to restart the challenge.

When the challenge is completed, the mouse movements, challenge seed, and some client-side data are sent to the server. The server is able to recreate the challenge from the seed and extract the necessary data to use for the model. This data includes the used device (mobile/desktop), completion time, shape difficulty level, and shape data. The shape data is created by comparing the mouse movement to the movement of the shapes. This leads to the determination of a fixed number of data points, such as n = 100. Then we divide the max score to complete the challenge by this number, so for example, if the max score is 5000 and n is 100, we get an interval of 50. At every 50th point, we collect information about the mouse position. The information contains which kind of shape the user is following and the absolute distance from the centroid of the closest shape. A distinction is made between the following shape being behind the obstacle or not. There is also the possibility of following nothing behind the object. This creates for the following options: chosenShape, variationShape, chosenObject, variationObject, and Object. Note that following nothing outside the object, so on the canvas, is not considered as this option does not gain any points. As there are no points gained, the user cannot

[1] https://github.com/akifozturk61/shape-captcha

pass the interval score and will not reach the measurement point. Finally, all 100 data points are put in an array and stored in the database, so it looks as follows: [[type0, distance0],..., [type99, distance99]].

4.2. Deployment

To prevent unexpected attacking methods on the CAPTHCA, we try to minimise the information that is sent between the client and server applications and include only essential information.When the challenge is completed, only the mouse movements, challenge seed, and client-side data such as completion time, device type, and shape difficulty are sent to the server. The server is able to recreate the challenge from the seed and extract the necessary movement data to use for the model. However, during the development stage, there were some troubles with the PRNG we use from Alea. The same PRNG on different programming languages was giving different random numbers using the same seed. For this reason, the server side was not able to recreate the same challenge using the seed. Because this was only a prototype implementation, we neglected the server side completely and calculated all the necessary data on the client side, sending it straight to our database. It has to be noted that during actual development of the CAPTCHA system, if the PRNG's are implemented correctly, there would be no problem in sending only the seed between the client and server, and the resulting dataset would be the same. So we decided to not waste more time and resources on this matter.

In order to collect the human data from friends and family, we needed the CAPTCHA application to be available online. We used Vercel [2] to deploy a web server that was directly connected with the github project. Every time new code is pushed, Vercel automatically triggers a new deployment with a unique url. As of writing this, the CAPTCHA challenge is still up and running [3]. This URL was shared with friends and family to access and perform the CAPTCHA challenge. After completion of the challenge, the necessary data was sent to the database.

The database is deployed on Supabase [4]. Supabase is an open-source platform that provides tools and services for building web and mobile applications. We used the Postgres database in order to store and retrieve our data.

4.3. Data Collection

Given the uniqueness of the challenge, there was no pre-existing human or bot data available to train our model. Due to this, we had to gather our own data. Sharing the challenge with family and friends was successful in achieving this for the human data. Each participant was allowed to complete the challenge multiple times, thereby expanding our database. The challenge was shared with approximately 50 individuals, resulting in 649 data entries. No compensation was provided to the participants. Before starting the challenge, participants are shown an informed consent page where the task, data privacy, risk, and participation of the experiment are explained in detail. The user has to agree with this form in order to proceed to the challenge.

To store the user data, an application was made to the TuDelft Human Research Ethics (HREC). The application contained the informed consent page, a dataplan, and a detailed checklist for human research. The application was approved, and the challenge was shared with participants to collect the human data.

To simulate the bot data, we considered different strategies an attacker might employ to complete the challenge. It has to be noted that this process does include some bias, as the model will contain only data in the way we designed the bot. An attacker who has different perspectives and methods of designing the bot might create situations that are

[2] https://vercel.com
[3] https://shape-captcha.vercel.app/
[4] https://supabase.com

not recognised by the model. We assumed the attacker would try to solve the challenge in the most efficient ways possible and created our bots accordingly.

The first strategy that we used was a ”perfect” bot. That is a bot that distinguishes the right shape right from the beginning and keeps following it until the end. During the challenge, the shape is followed perfectly, keeping the pointer at the centroid of the shape. Our assumption is that this ”perfect” bot is remarkably also the easiest method for the model to detect that the user is a bot. That is because a human user will most likely never be able to follow the right shape with that kind of accuracy. Following this assumption, we created another bot that still keeps track of the right shape. However, this time we introduced some jitter with the mouse movements, such that the pointer is not always at the centroid but will move around a bit while following the right shape. This is already a better attack, as it is mimicking human behaviour more accurately. Then, we also considered that following the right shape for the whole duration of the challenge might not always represent human behaviour. So we designed new bots that will occasionally follow wrong shapes, switch between wrong and right shapes, or only follow wrong shapes. Here, we also used both ”perfect” and jittered following methods. The bot resulted in 585 data entries, bringing the total database up to $649 + 585 = 1234$ total entries.

5

Data Analysis and Feature Extraction

5.1. Preprocessing

We explain the preprocessing steps taken on the collected dataset to use for training and testing the ML models. The same steps are applied when the model receives data from future challenges in order to classify the user as a human or bot. An example of the original database with the column names and values is shown in Table 5.1.

The first step is to convert the shapeData column into a time series so that we can use the tsfresh library and extract relevant features from it. For each challenge, there are exactly 100 timeframes on which information about the mouse movement is stored. At each timeframe, two values are measured: type and distance. The type value represents the type of shape that the mouse pointer was following at that timeframe. There are a total of 5 possible types.

- **chosenShape:** The user is following the chosen shape.
- **chosenObstacle:** The user is following the chosen shape, which happens to be inside the obstacle.
- **variationShape:** The user is following a variation shape.
- **variationObstacle:** The user is following a variation shape, which happens to be inside of the obstacle.
- **Obstacle:** The user is inside the obstacle, but not following any shape.

The distance value represents the absolute distance from the mouse pointer to the nearest shape's centroid. In order to convert these values to a time series, we split the information in each element to a separate row and gave it a timestamp: t0, t1,..., t99. An example of the splitting data for challenge (id=1) is given in Table 5.2. The seed column is removed as it is not necessary any more.

Finally, we apply label encoding to the categorical data columns, such as device, type, and label. We use the standard LabelEncoder from the sklearn library. After all preprocessing steps, an example of the final dataset is given in Table 5.3.

id	seed	device	shapeData	time	shapeDifficulty	label
1	279143102	desktop	[(”chosenShape”, 55), ... , (”variationShape”, 45)]	24.55	4	human
2	455625862	desktop	[(”chosenShape”, 0), ... , (”chosenShape”, 0)]	11.32	9	bot
3	975294772	mobile	[(”variationShape”, 23), ... , (”variationObstacle”, 21)]	25.10	1	human

Table 5.1: Example of the original dataset

id	t	device	time	shapeDifficulty	type	distance	label
1	t0	desktop	24.55	4	chosenShape	55	human
1	t...	desktop	24.55	4	...	...	human
1	t99	desktop	25.55	4	variationShape	45	human
...	...	...	...	...	...	...	...

Table 5.2: Example dataset after converting to time series

id	t	device	time	shapeDifficulty	type	distance	label
1	t0	0	24.55	4	0	55	0
1	t...	0	24.55	4	...	...	0
1	t99	0	25.55	4	2	45	0
2	10	0	25.10	1	0	0	1
...	...	...	...	...	...	...	...

Table 5.3: Example dataset after preprocessing

5.2. Feature Extraction

Feature extraction is an important step in the machine learning process. Selecting the right features can have a significant effect on the accuracy of the model. By transforming the raw data into meaningful features, we allow the model to focus on the most relevant parts of the data. This speeds up the training process and results in more reliable models.

We use tsfresh to obtain a set of features from the mouse movement data collected during the challenge. More specifically, these are the type and distance columns we prepared in the previous step. Tsfresh is a Python package that automates the feature extraction process on time series data. It extracts a large number of features, such as statistical, time, and frequency features. There are several modes in which the feature extraction can be performed. The first option, full features, generates all possible features from the time series, resulting in 1567 features for our dataset. However, these features also contain NaN values, and some of them are irrelevant. To refine this, we used the impute method from the tsfresh library to replace all NaN values with the median value. Then we used the select_features() method from the tsfresh library, which checks the relevance of all features and returns a reduced feature set containing only relevant features. With this step, we managed to reduce our feature set to 804 features, which is still a lot. The second option was using the efficient settings to extract the features, but this was resulting in 1558 features, which was almost as much as using the full set, so we did not proceed with this method. For the last option, we used the minimal option, which produced a feature set of only 20 features. Considering our limited data size, we expect the minimal option to be the most effective feature set. Additionally, we included the shape difficulty, time, and device type as features, bringing the total feature set to 24 features plus the label in our dataset.

6

Machine Learning Models

All of the models in this chapter are implemented using Sklearn. Sklearn is an open source machine learning library for Python. It provides simple and efficient tools for machine learning and predictive data analysis. Built on top of NumPy, SciPy, and Matplotlib, it is designed to be easy to use. All operations of training, testing, and fitting the models are done on a MacBook Pro (13-inch, M1, 2020) with an 8-core GPU, a 16-core neural engine, and 16GB of memory.

6.1. Model Selection

The task of the machine learning model is to classify that the given data is generated by a human user or an automated bot. Finding the right model that fits the dataset and achieves high accuracy is a complex task. Different models provide different insights and advantages over others. We have chosen to implement three different models that focus on a different aspect of the dataset. The decision tree provides interpretability, the neural network captures complex patterns, and the one-class SVM excels in anomaly detection. A more detailed overview of the models is given below.

- **Decision Tree:** A decision tree model uses a tree-like structure to make decisions based on the input data. Each node in the tree acts as a decision point, where one of the features of the dataset is evaluated. The branches of the node represent the possible outcomes of the decision and lead to other nodes or to a final decision represented by the leaf node. Decision trees are intuitive and simple to visualise, which makes them useful for understanding which features are more important when making a decision.
- **Neural Network:** Neural networks are inspired by the human brain. A network model is built consisting of interconnected layers of nodes, comparable with neurones in the human brain. Each node processes the data that it gets as an input and passes the result as an output to the next layer. This process is determined by a weight value that is present between the nodes across different layers. The weights control how much influence the input data has on the output of that node. A network typically starts with one input layer, then comes one or more hidden layers, and finally it has an output layer. Neural networks are powerful tools for recognising complex patterns in large datasets.
- **One-Class SVM:** A one-class SVM model is a type of SVM model used for anomaly detection. It is used to train the model with only a single class containing the normal data and then aims to identify whether new data belongs to the same class or are anomalies. One-class SVMs are an effective tool for anomaly detection or when the dataset only has data available from a single class for training.

6.2. Training and Testing

Training and testing data are an essential component of the machine learning process. The complete dataset is split, usually in an 80/20 ratio, into a training and testing set, respectively. The primary goal of the training phase is to learn patterns and relationships in the dataset that separate the human users data from the automated bots data. During the training phase, the model receives a labelled training set on which it can adjust its model parameters to minimise the error between predictions and actual labels. Each model has a different method to achieve this. For example, the decision tree is trained by recursively splitting the dataset on feature values, creating a tree-like structure. The neural network is trained by a process called forward and backward propagations, during which the weights of each connection between nodes are adjusted to find the optimal setting. The one-class SVM is unique in a way that it only trains on one-class data. It learns to find a boundary that includes most of the human class and excludes the outliers. This boundary can then be used to identify anomalies in the dataset. If the boundary is too low, human data can be seen as anomalies. Picking the boundary too high can result in anomalies being classified as human users. It is important to find the right balance in order to create an accurate classifier. It is of utmost importance for all models that during the training phase the model can never receive any form of the testing data.

The testing phase is then used to evaluate the performance of the model on unseen data. The model makes predictions on the testing set, and the results are compared against the actual labels in the testing set. Various metrics can be computed to evaluate the results of the model. These metrics provide an insight into the strengths and weaknesses of the model. More detailed information on these metrics will be given in Section 6.3.

Finally, it has to be noted that we had to remove the ”device type” feature for the training of the Decision Tree and Neural Network models because we were not able to perform and collect data from bot attacks on mobile devices. This means that during training, the models would never learn about bots on mobile devices, and therefore, when predicting future data, if it contained mobile data, it would always be classified as human. In the case of the one-class SVM model, this is not relevant because only the human class is trained. Everything else is considered not-human, which can be anything, including mobile bots.

6.3. Results

6.3.1. Terminology

Firstly, there are some common terms that we need to understand in order to evaluate machine learning models.

- **True Positive (TP):** This occurs when the model predicts the class as positive and it is actually positive. For example, predict a human, and it is a human.
- **False Positive (FP):** This occurs when the model predicts the class as positive but it is actually negative. For example, predict a human, but it is a bot.
- **True Negative (TN):** This occurs when the model predicts the class as negative and it is actually negative. For example, predict a bot, and it is a bot.
- **False Negative (FN):** This occurs when the model predicts the class as negative but it is actually positive. For example, predicta bot, but it is a human.

To evaluate the performance of machine learning models, there are several metrics that use these terms. A detailed explanation of each metric is given below.

- **Confusion Matrix:** This is just a representation of each of the terms explained above in a matrix format. An example confusion matrix is given in Figure 6.1

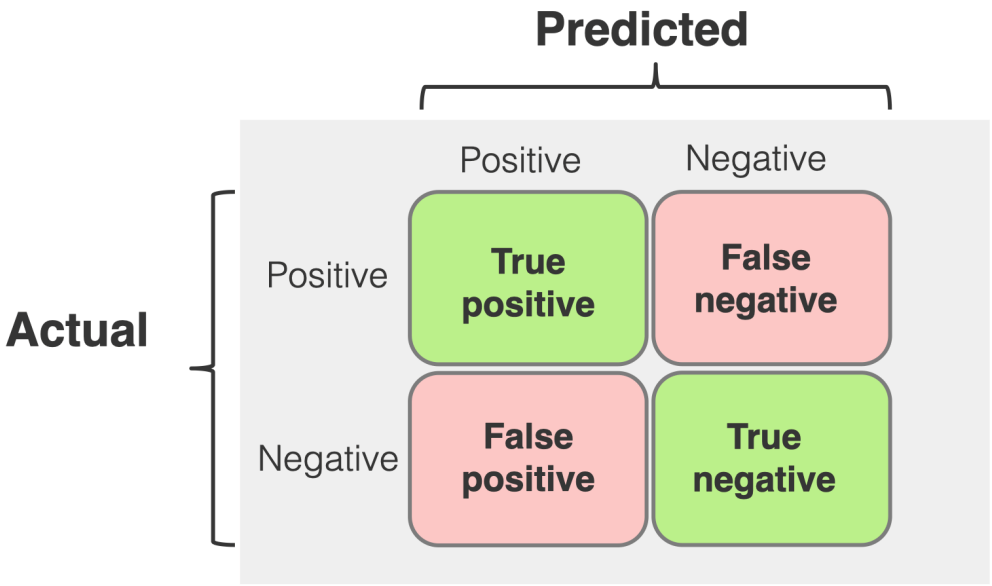


Figure 6.1: Example Confusion Matrix

- **Accuracy:** Accuracy is the ratio of all true instances to the total number of instances. It is the most commonly used metric and is used to indicate the overall effectiveness of the model. However, it can be a misleading metric in imbalanced datasets. Say we have data of 100 users, and only 5 of them are automated bots. Predicting all users as humans would give an accuracy of 95%, which can be seen as a high accuracy, but the model did nothing else than predicting a single class for everything.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

- **Precision:** Precision is the ratio of true positives to the total number of positives. It indicates how many times the model is actually right when it claims it is right. High precision means fewer false positives.

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

- **Recall** Recall is the ratio of true positives to the total actual positives. The actual positives are when a human is classified as a human, or a bot as a bot. It indicates how many times the model makes the right classification. High recall means fewer false negatives.

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

- **F1 Score** The F1 score is the harmonic mean of precision and recall. Both values contribute to the final F1 score, so a higher score for both results in a higher F1 score. However, due to the product in the denominator, if one of the two is a low value, the final F1 score goes down significantly.

$$\text{F1 Score} = 2 * \frac{precision * recall}{precision + recall}$$

- **Classification report** A classification report includes the precision, recall, and f1-score for each class in the dataset. It also includes a ”support” column, which is just

the number of occurrences of the class in the dataset. Furthermore, it includes a macro and a weighted average. The macro average is the unweighted mean across all classes. It treats all classes equally, regardless of their frequency. The weighted average is the mean across all classes, weighted by the number of instances in each class. This method accounts for class inbalance by giving more weight to the classes that contain more instances.

- **ROC Curve and AUC** Receiver Operating Characteristic (ROC) is a graph plot of the model's performance across various threshold values. It plots the true positive rate (TPR) against the false positive rate (FPR). The Area Under the Curve (AUC) is just the area under the curve. A higher AUC means better performance. An example ROC plot is shown in Figure 6.2. We aim to find the threshold value that leads us closer to the top left corner.

$$\text{TPR = Recall = } = \frac{TP}{TP + FN}$$

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

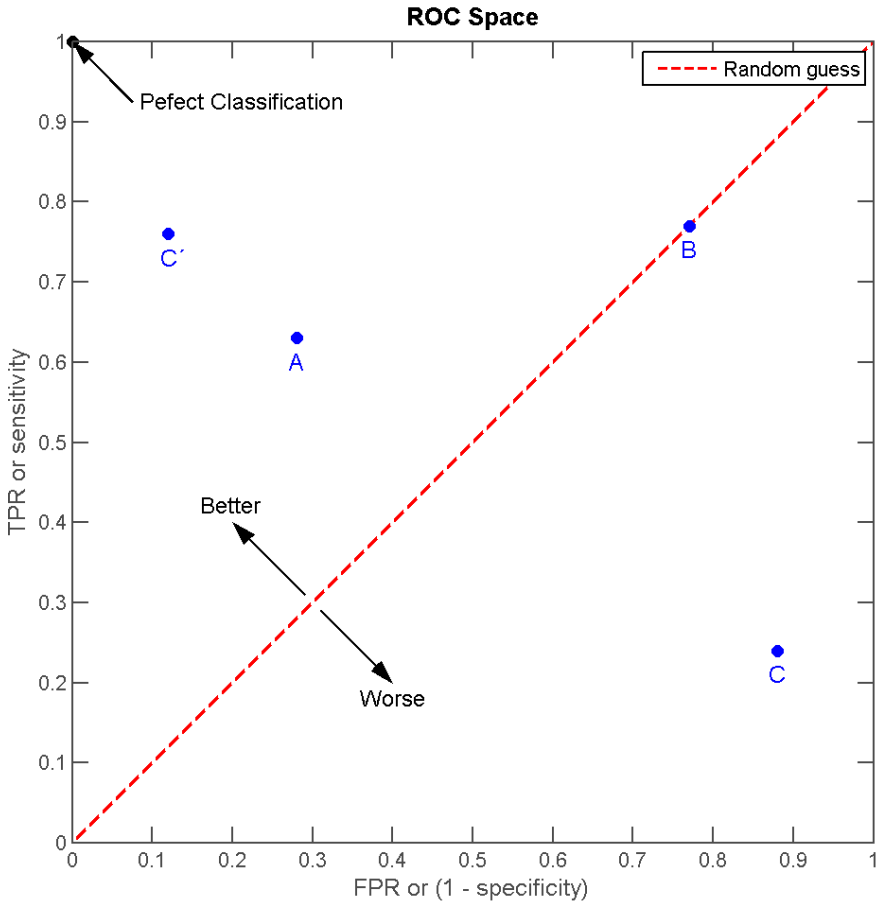


Figure 6.2: Example ROC Plot

6.3.2. Decision Tree

For the decision tree, we use the DecisionTreeClassifier from sklearn. A full list of the parameter values used is provided in Table 6.1.

Parameter	Value
random_state	42
max_depth	5
min_samples_split	10
min_samples_leaf	5

Table 6.1: Parameters for DecisionTreeClassifier

The model is then fit with the training data and training labels, after which we can use it to predict new data. The new predictions are done with the testing data, and we retrieve the predicted labels. Combined with the actual labels, we can then evaluate the performance using the confusion matrix, classification report, feature importance values, and a visualisation of the decision tree. Because the visualisation of the tree was too large to fit on one page, we have separated the tree into 3 parts. All parts of the decision tree can be found in Appendix A. The first part is the root of the decision tree, which can be seen in Figure A.1. Then we can see the left-hand side of the decision tree in Figure A.2 and the right-hand side in Figure A.3.

Confusion matrix

The confusion matrix for the decision tree model is shown in Figure 6.3. From the 138 actual human instances, the model has correctly classified 123, while 15 instances were wrongly classified as bots. On the other hand, out of the 109 actual bot instances, the model has correctly classified 88, with 21 wrongly classified as humans. We can see a satisfactory performance of classification in both classes with a small number of false negatives for both classes.

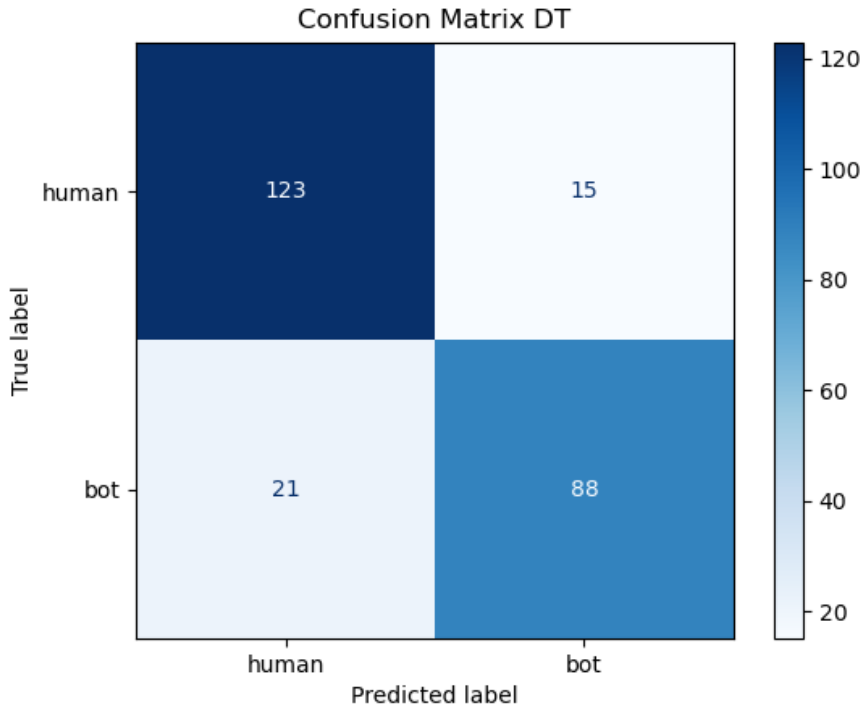


Figure 6.3: Decision Tree Confusion Matrix

Classification report

The classification report for the decision tree model is shown in Table 6.2. The overall accuracy of the model is 0.85, so 85% of the total instances are classified correctly. We observe an equal precision score and small differences in the recall and f1 scores. This results in both the macro and weighted averages being consistent at 0.85, showing us an equal performance between the two classes without any bias to one or another.

	precision	recall	f1-score	support
human	0.85	0.89	0.87	138
bot	0.85	0.81	0.83	109
accuracy			0.85	247
macro avg	0.85	0.85	0.85	247
weighted avg	0.85	0.85	0.85	247

Table 6.2: Classification Report Decision Tree

ROC Curve

The ROC curve for the decision tree model is shown in Figure 6.4. The shape of the curve rises towards the top left corner, indicating an increase in TPR with a higher FPR. Together with the high AUC value of 0.92, it indicates that the model has good performance in classifying the data. The model provides a high level of accuracy and reliability for our CAPTCHA system.

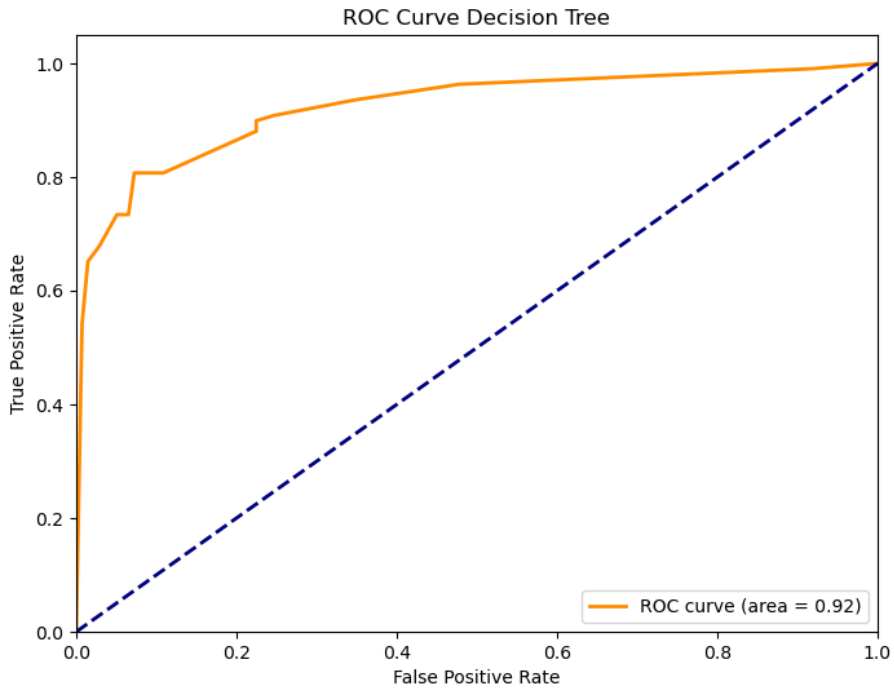


Figure 6.4: Decision Tree ROC Plot

Feature Importance

The feature importance table for the decision tree model is shown in Table 6.3. We can see a clear winner in terms of importance in the distance__median value. This is the median value computed by tsfresh on the distance feature. Following up are the

type__variance, time, and type__median values. Type__variance is the variance value computed by tsfresh on the type feature; likewise, likewise type__median is the computed median value. The time feature is the completion time of the challenge. Other features in the table are low-importance, and those with 0.0 importance are not shown. One thing to note is the low importance of the shapeDifficulty feature. We think this is because we only stored the chosen feature's difficulty rating when calculating this feature. With this method, some information is lost about the user's choice. Instead, we should have stored the difficulty of the other shape options as well. Then it would be possible to compare the difficulty of the chosen shape to the other available shapes to get a better insight into the user preference for shape types. This could have resulted in a higher importance value for the shapeDifficulty feature.

Feature	Importance
distance__median	0.572825
type__variance	0.152930
time	0.117984
type__mean	0.116901
distance__standard_deviation	0.014987
distance__sum_values	0.010002
distance__variance	0.007468
distance__mean	0.003414
shapeDifficulty	0.002392
type__standard_deviation	0.001097

Table 6.3: Feature Importance Decision Tree

6.3.3. Neural Network

For the neural network, we use the MLPClassifier from sklearn. A full list of the parameter values used is provided in Table 6.4

Parameter	Value
hidden_layer_sizes	(64, 32)
activation	'relu'
solver	'adam'
alpha	0.0001
batch_size	'auto'
learning_rate	'constant'
learning_rate_init	0.001
max_iter	200
shuffle	True
random_state	42
tol	0.0001
early_stopping	True
validation_fraction	0.2

Table 6.4: Parameters for Neural Network

Confusion matrix

The confusion matrix for the neural network model is shown in Figure 6.5. From the 130 actual human instances, the model has correctly classified 117, while 13 instances were wrongly classified as bots. On the other hand, out of the 117 actual bot instances, the model has correctly classified 99, with 18 wrongly classified as humans. We can see a satisfactory performance of classification in both classes with a small number of false negatives for both classes. In comparison with the decision tree, both models are equal in performance based on the confusion matrix.

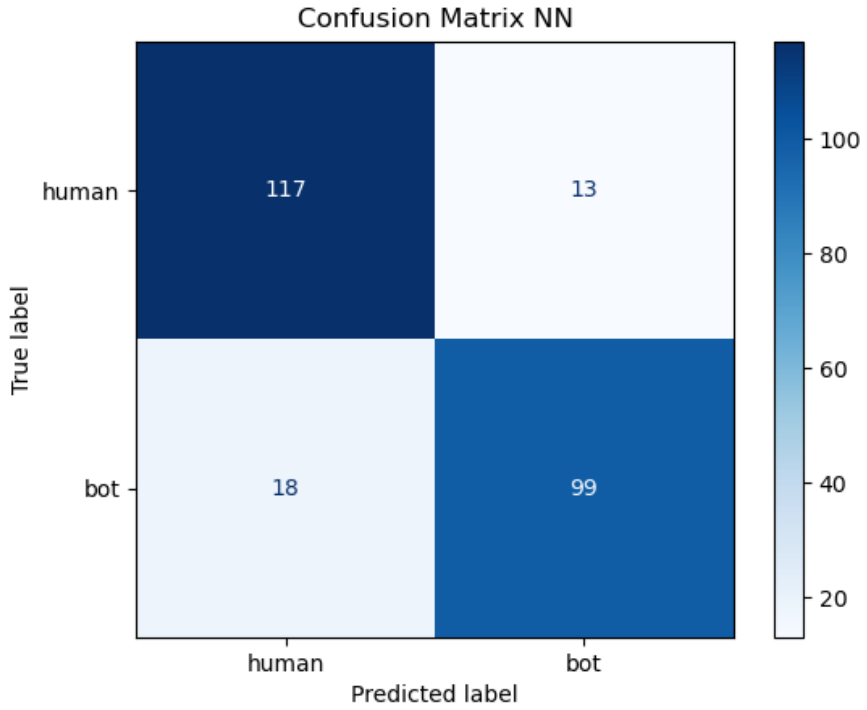


Figure 6.5: Neural Network Confusion Matrix

Classification report

The classification report for the neural network model is shown in Table 6.5. The overall accuracy of the model is 0.87, so 87% of the total instances are classified correctly. We again observe an almost equal precision score and small differences in the recall and f1 scores. This results in both the macro and weighted averages being consistent at 0.87, showing us an equal performance between the two classes without any bias to one or another. Just like the confusion matrix, the classification reports of both the decision tree and neural network are similar in performance.

	precision	recall	f1-score	support
human	0.87	0.90	0.88	130
bot	0.88	0.85	0.86	117
accuracy			0.87	247
macro avg	0.88	0.87	0.87	247
weighted avg	0.87	0.87	0.87	247

Table 6.5: Classification Report Neural Network

ROC Curve

The ROC curve for the neural network model is shown in Figure 6.6. The shape of the curve rises towards the top left corner, indicating an increase in TPR with a higher FPR. Together with the high AUC value of 0.95, it indicates that the model has good performance in classifying the data. Compared with the ROC plot for the decision tree, we can see the neural network reaches the top left corner much faster, and it also comes closer to the 1.0 value. This is also represented in the higher AUC value of 0.95 for the neural network. We can conclude that the neural network provides a high level of accuracy and reliability for our CAPTCHA system.

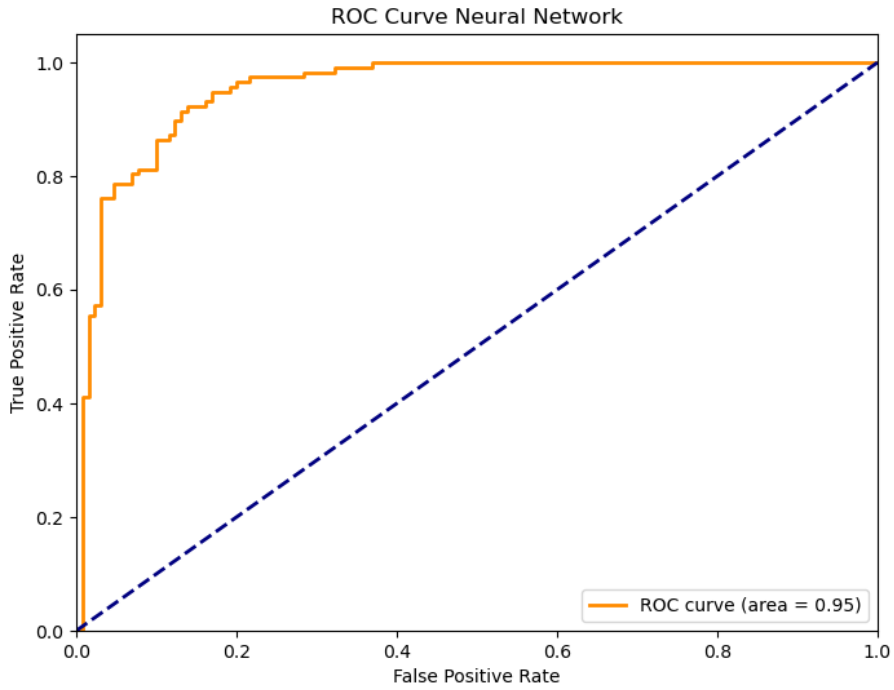


Figure 6.6: Neural Network ROC Plot

6.3.4. One-Class SVM

For the One-Class classifier, we use the OneClassSVM from sklearn. A list of the parameter value used is provided in Table 6.6

Parameter	Value
gamma	'auto'

Table 6.6: Parameters for One-Class SVM

Confusion matrix

The confusion matrix for the one-class SVM model is shown in Figure 6.7. From the 585 actual human instances, the model has correctly classified 489, while 96 instances were wrongly classified as bots. On the other hand, out of the 130 actual not-human instances, the model has correctly classified 72, with 58 wrongly classified as humans. We can see a clear difference in performance compared to the decision tree and neural network confusion matrices. Firstly, the one-class SVM has access to much more training instances than the other two models. This is because the model only trains on the human class, neglecting any bot data. This results in a much higher true positive rate for the human class. But the false negative and false positive rates have also increased compared to the other models. The true negative is the only value that has remained similar.

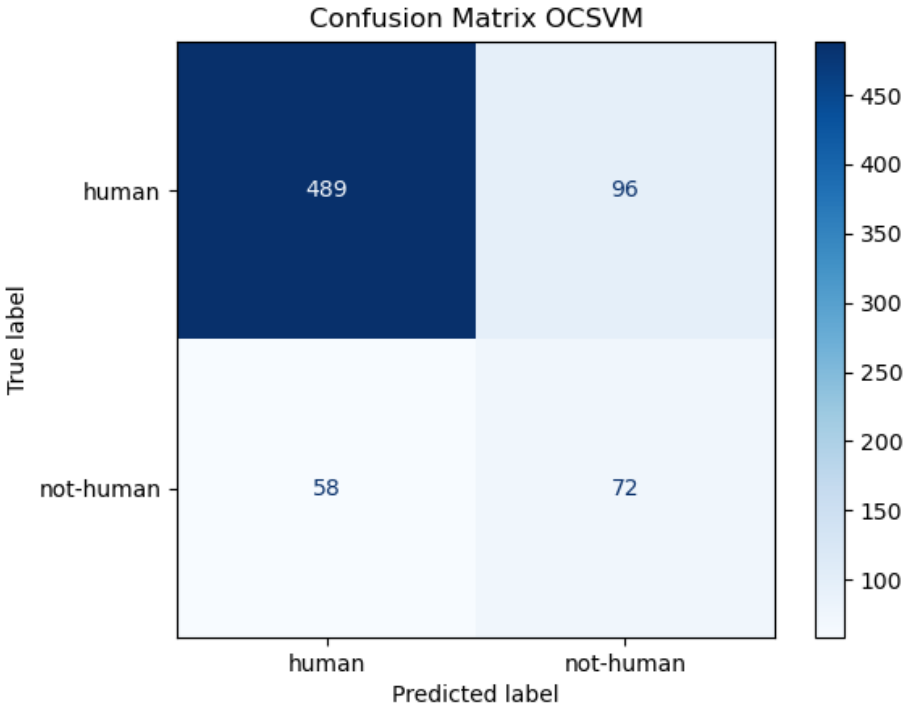


Figure 6.7: One-Class SVM Confusion Matrix

Classification report

The classification report for the one-class SVM model is shown in Table 6.7. The overall accuracy of the model is 0.78, so 78% of the total instances are classified correctly. However, this time we notice a substantial difference between the performances of both classes. While the human class performs similar in terms of performance to the decision tree and neural network models, the not-human class performs really badly. Precision score is decreased by almost 50%, recall by 35% and f1-score by 44%. We also notice a great difference in support numbers, indicating an imbalance in the dataset. This is further confirmed by the higher weighted average scores compared to the macro average. The macro average treats both classes equally, while the weighted average takes into account the high number of instances in the human class.

	precision	recall	f1-score	support
human	0.89	0.84	0.86	585
not-human	0.43	0.55	0.48	130
accuracy			0.78	715
macro avg	0.66	0.69	0.67	715
weighted avg	0.81	0.78	0.79	715

Table 6.7: Classification Report One-Class SVM

ROC Curve

The ROC curve for the one-class SVM model is shown in Figure 6.8. The curve is still above the diagonal line representing random performance, so the model is effective to some extent but not highly reliable. The performance is moderate, which is also confirmed by the moderate value of the AUC = 0.69. There is room for improvement, and the model could benefit from more training data to provide better performance. Compared to the

decision tree and neural network models, the one-class SVM is not as reliable and does not provide a satisfactory accuracy.

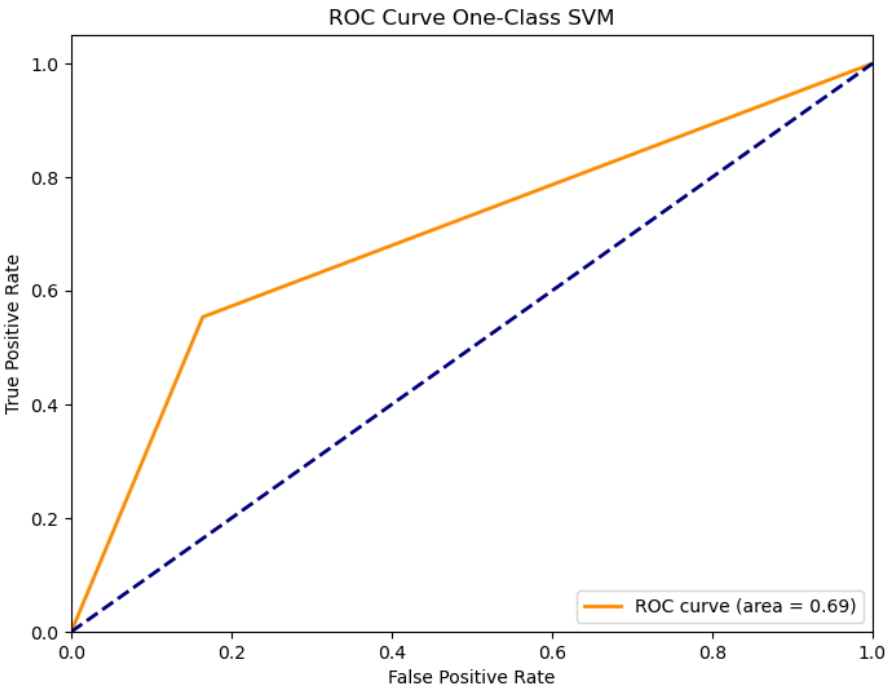


Figure 6.8: One-Class SVM ROC Plot

7

Discussion

7.1. Interpretation of Results

Comparing the performance of the three machine learning models, we can clearly see that the decision tree and neural network models perform much better than the one-class SVM model. Both models provide high accuracy and effectively distinguish between human and bot data. The decision tree provides high interpretability by giving an overview of the feature importance in the decision-making process. The neural network shines with its ability to capture the complex patterns in the dataset.

However, it is important to notice that while these models show promising results, the dataset used for the training and testing of the models is not fully representative of real-world scenarios. The bot data only includes data from attacks that we performed ourselves. In the real world, attackers can use different attacking methods, generating data that the model has not seen before. This limitation of both models to classify new, unseen attacks is a serious concern.

Despite its relatively lower performance, the one-class SVM model offers a unique advantage in addressing this issue. Because this model is trained exclusively on human data, the model considers any outliers as not-human. If the outlier boundary is chosen with enough precision, this can include future attacker data that the model has not seen before. This characteristic of the one-class SVM model provides a layer of security against novel attack methods that the other two models do not provide.

7.2. Strengths and Limitations

Strengths

The main strength of our CAPTCHA system is the positive user experience. While adding complexity to traditional CAPTCHAs can have a negative effect on the user experience, our method benefits from the added complexity. Increased complexity leads to more human errors, which are rewarded in our CAPTCHA design. This makes the challenge more engaging and less frustrating for the regular users. The task of the CAPTCHA is relatively simple, and the user is not expected to change its behaviour, even if the challenge becomes more complex.

Another significant strength of the CAPTCHA comes from the one-class SVM model. While the performance might not have been the most optimal, it provides a future proof classification method that, once trained optimally, can identify future attacker data that it has not seen before.

Weaknesses

The main weakness of the CAPTCHA system lies in its dataset. The current dataset used for training and testing is not a fully representation of real-world scenarios. The bot data only includes data from attacking methods that we implemented ourselves, which does not cover all possible attacker scenarios out there. There is a need for more data collection and refinement of our machine learning models.

Another weakness is the introduction of new attacking methods. While our CAPTCHA system provides novel security measures against modern AI attacks, it also opens up new attacking methods. Attackers can deliberately weaken their methods or introduce computation errors to mimic human behaviour. This creates a new challenge in recognising real human errors from machine-generated errors.

7.3. Future Work

Because of the resource and time limitations of the thesis project, we were unable to research a number of points that we put here as future work. First, we provided the main building blocks of a novel CAPTCHA system that uses machine precision and human errors to make an accurate classification. However, we have only explored one implementation of this idea. The building blocks can be used to produce many more implementations that exploit this idea in a more effective way.

Next, we did not have enough time to do a full user experience study. Because the complete CAPTCHA system needs working classifiers to deploy it to real users, we had to first collect our own dataset and train our models. Together with a full implementation of the CAPTCHA system itself, we had no time left to do a user study. While in theory the user experience should be satisfactory with an accurate classifier model, we do not have scientific evidence to prove this.

Lastly, for both the security parameters in the challenge and the machine learning models in the classification, a limited amount of testing is performed to find optimal parameters. Some values have been chosen that ”feel” well and provide pleasing results, but further research needs to be done to find the optimal parameters that find the balance between optimal security and user experience.

8

Conclusion

In this thesis, we explored novel methods to design secure CAPTCHA systems. We focused on providing a user-friendly experience by returning the simplicity of traditional CAPTCHAs. The core of the challenge involves moving shapes, which is something recognised by all sorts of human types. Furthermore, the user is rewarded for making errors, proving their humanness. While increasing the complexity of traditional CAPTCHAs was having a negative effect on the user experience, our CAPTCHA system becomes more user-friendly when the complexity is increased. However, all of this has introduced a new attacking method: worsening the attacker side to also make mistakes. The challenge then becomes to identify real human mistakes from machine-generated ones.

We successfully implemented a fully operational prototype on the client side. The decision-making process is reliant on the machine learning classifiers. However, due to the limited amount of training data available, these classifiers are not ready to use yet. Further research, data collection, model refinement, and attacking methods need to be explored in order to achieve a final working end product.

With technology evolving rapidly, modern CAPTCHAs are faced with difficult challenges in maintaining both security and user-friendliness. With the rise of AI and technology that keeps improving, traditional CAPTCHAs are more vulnerable to new attacking methods. CAPTCHAs that are secure today might not hold up against the threats of tomorrow. This dynamic environment shows the difficulty in designing CAPTCHAs that are secure against attacks while still providing a seamless user experience.

Our work introduces a unique perspective in CAPTCHA development that leverages the precision of machines and the natural tendency of humans to make errors. This approach improves the security of CAPTCHA systems while providing a user-friendly experience that is more engaging and less frustrating. By exploring this perspective, we open the door for future advancements in this field.

References

[1] Luis von Ahn et al. “reCAPTCHA: Human-Based Character Recognition via Web Security Measures”. In: *Science* 321.5895 (2008), pp. 1465–1468. DOI: 10.1126/science.1160379. eprint: https://www.science.org/doi/pdf/10.1126/science.1160379. URL: https://www.science.org/doi/abs/10.1126/science.1160379.

[2] Ismail Akrout, Amal Feriani, and Mohamed Akrout. “Hacking google recaptcha v3 using reinforcement learning”. In: *arXiv preprint arXiv:1903.01003* (2019).

[3] Henry Baird and Jon Bentley. *Implicit CAPTCHAs*. 2005. DOI: 10.1117/12.590944.

[4] Jun Chen et al. “A Survey on Breaking Technique of Text-Based CAPTCHA”. In: *Security and Communication Networks* 2017 (2017), pp. 1–15. DOI: 10.1155/2017/6898617.

[5] Mauro Conti, Claudio Guarisco, and Riccardo Spolaor. “CAPTCHaStar! A Novel CAPTCHA Based on Interactive Shape Discovery”. In: *Applied Cryptography and Network Security*. Ed. by Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider. Cham: Springer International Publishing, 2016, pp. 611–628. ISBN: 978-3-319-39555-5.

[6] Nghia Trong Dinh and Vinh Truong Hoang. “Recent advances of Captcha security analysis: a short literature review”. In: *Procedia Computer Science* 218 (2023). International Conference on Machine Learning and Data Engineering, pp. 2550–2562. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2023.01.229. URL: https://www.sciencedirect.com/science/article/pii/S1877050923002296.

[7] Haichang Gao et al. “A Simple Generic Attack on Text Captchas”. In: *Network and Distributed System Security Symposium*. 2016. URL: https://api.semanticscholar.org/CorpusID:12024381.

[8] Ian J. Goodfellow et al. *Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks*. 2014. arXiv: 1312.6082 [cs.CV].

[9] Google. *Are you a robot? Introducing “No CAPTCHA reCAPTCHA”*. 2014. URL: https://developers.google.com/search/blog/2014/12/are-you-robot-introducing-no-captcha (visited on 01/03/2024).

[10] Google. *Invisible reCAPTCHA”*. 2017. URL: https://developers.google.com/recaptcha/docs/invisible (visited on 01/03/2024).

[11] Rich Gossweiler, Maryam Kamvar, and Shumeet Baluja. “What’s up CAPTCHA? A CAPTCHA Based on Image Orientation”. In: *Proceedings of the 18th International Conference on World Wide Web*. WWW ’09. Madrid, Spain: Association for Computing Machinery, 2009, pp. 841–850. ISBN: 9781605584874. DOI: 10.1145/1526709.1526822. URL: https://doi.org/10.1145/1526709.1526822.

[12] Thomas Gougeon and Patrick Lacharme. “A Simple Attack on CaptchaStar”. In: *International Conference on Information Systems Security and Privacy*. 2018. URL: https://api.semanticscholar.org/CorpusID:198365671.

[13] Meriem Guerar et al. “Gotta CAPTCHA ’Em All: A Survey of 20 Years of the Human-or-Computer Dilemma”. In: *ACM Comput. Surv.* 54.9 (2021). ISSN: 0360-0300. DOI: 10.1145/3477142. URL: https://doi.org/10.1145/3477142.

[14] Carlos Javier Hernández-Castro, David F. Barrero, and Maria Dolores R-Moreno. “Breaking CaptchaStar Using the BASECASS Methodology”. In: *ACM Transactions on Internet Technology* 23 (2022), pp. 1–12. URL: https://api.semanticscholar.org/CorpusID:251980517.

[15] Montree Imsamai and Suphakant Phimoltares. “3D CAPTCHA: A Next Generation of the CAPTCHA”. In: *2010 International Conference on Information Science and Applications*. 2010, pp. 1–8. DOI: 10.1109/ICISA.2010.5480258.

[16] Mathias Sablé-Meyer et al. “Sensitivity to geometric shape regularity in humans and baboons: A putative signature of human singularity”. In: *Proceedings of the National Academy of Sciences* 118 (2021), e2023123118. DOI: 10.1073/pnas.2023123118.

[17] N. Tariq et al. *CAPTCHA Types and Breaking Techniques: Design Issues, Challenges, and Future Research Directions*. 2023. arXiv: 2307.10239 [cs.CR].

[18] Guixin Ye et al. “Yet Another Text Captcha Solver: A Generative Adversarial Network Based Approach”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’18. Toronto, Canada: Association for Computing Machinery, 2018, pp. 332–348. ISBN: 9781450356930. DOI: 10.1145/3243734.3243754. URL: https://doi.org/10.1145/3243734.3243754.

[19] Binbin Zhao et al. “Towards Evaluating the Security of Real-World Deployed Image CAPTCHAs”. In: *Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security*. AISec ’18. Toronto, Canada: Association for Computing Machinery, 2018, pp. 85–96. ISBN: 9781450360043. DOI: 10.1145/3270101.3270104. URL: https://doi.org/10.1145/3270101.3270104.

A

Appendix A

Decision Tree

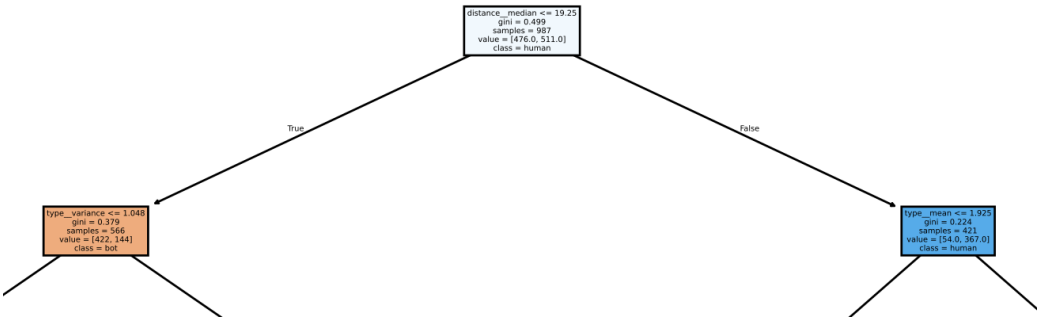


Figure A.1: Root of the decision tree.

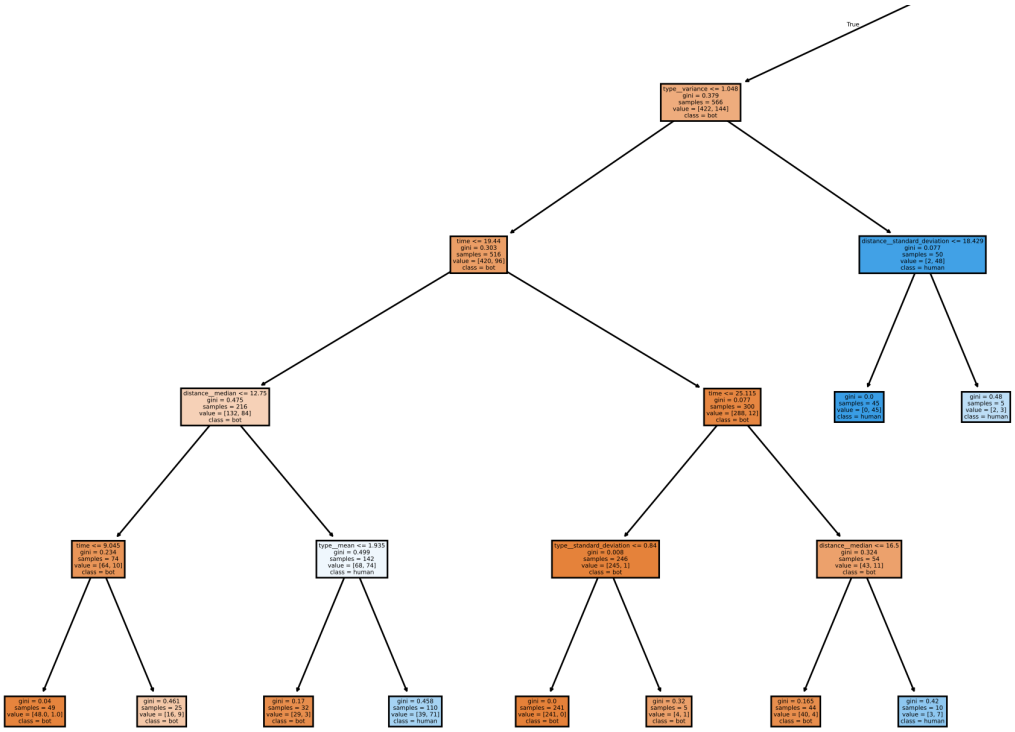


Figure A.2: Left-hand side of the decision tree.

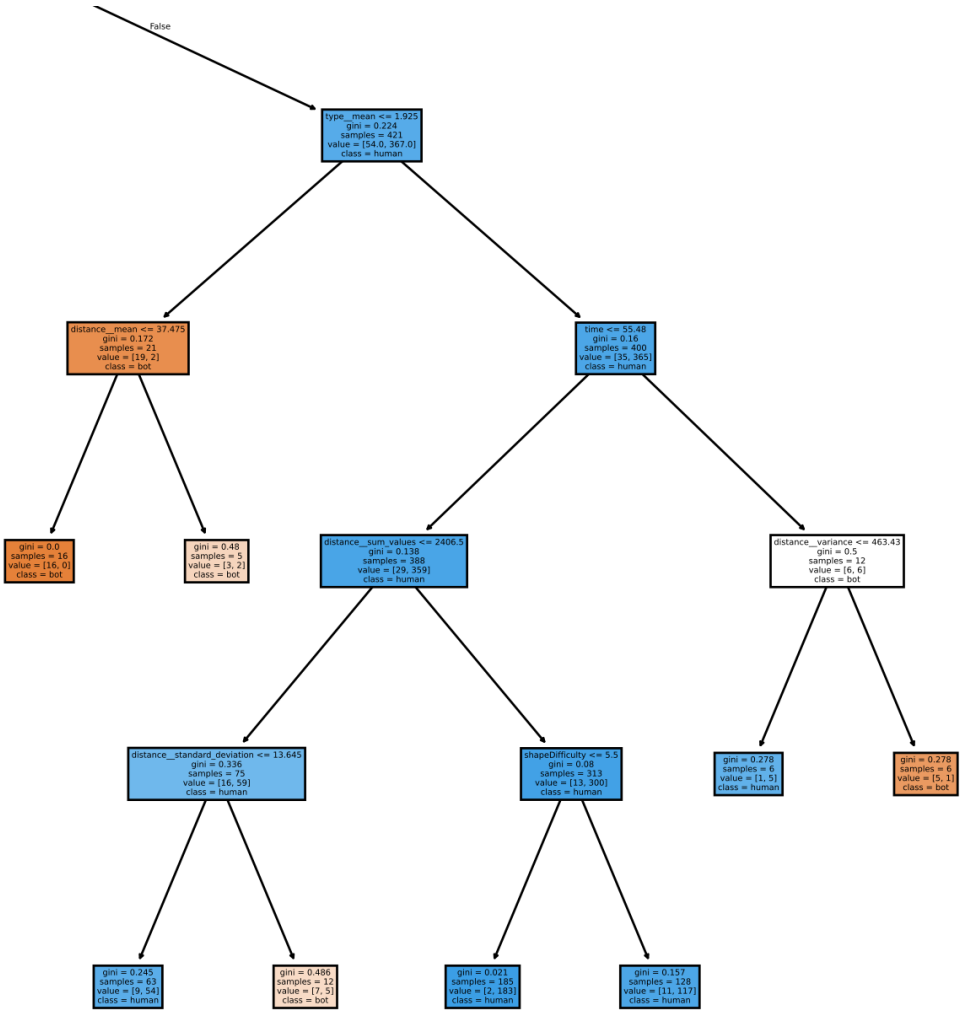


Figure A.3: Right-hand side of the decision tree.