Université de Namur
Faculté des Sciences
Département d'informatique

Report : Flagging suspicious hosts from DNS traces

Luyckx Marco 496283
Bouhnine Ayoub 500048

Professors: ROCHET Florentin

Course: Data analysis for cybersecurity ICYBM201

Delivered in November 2023

Academic year 2023-2024

Contents

1 Introduction **3**
- 1.1 How to run the project ? . . . 3

2 Explanations on how we approached the problem **4**
- 2.1 Our workflow . . . 4
- 2.2 Format of DNS queries . . . 5
- 2.3 Our assumptions and considerations . . . 6
- 2.4 Features selection . . . 7
  - 2.4.1 Aggregation/combination of features . . . 8

3 Background knowledge **13**
- 3.1 Slides from the course . . . 13
- 3.2 Articles . . . 14
- 3.3 Pitfalls . . . 15
- 3.4 Data collection and labelling . . . 15
  - 3.4.1 Sampling bias . . . 15
  - 3.4.2 Label inaccuracy . . . 16
- 3.5 System design and learning . . . 16
  - 3.5.1 Data snooping . . . 16
  - 3.5.2 Spurious correlations . . . 18
  - 3.5.3 Biased parameters selection . . . 18
- 3.6 Performance evaluation . . . 18
  - 3.6.1 Inappropriate baseline . . . 18
  - 3.6.2 Inappropriate performance measures . . . 19
  - 3.6.3 Base-rate fallacy . . . 19
- 3.7 Deployment and operation . . . 20
  - 3.7.1 Lab-only evaluation . . . 20
  - 3.7.2 Inappropriate threat model . . . 20
- 3.8 Summary of the pitfalls . . . 20

4 Algorithms and techniques used **21**
- 4.1 Our choice . . . 21
- 4.2 Baseline algorithms . . . 21
- 4.3 Other and more complex algorithms . . . 21

5 Discussion **22**
- 5.1 Definition of metrics used . . . 22
- 5.2 Eval2 : evaluation dataset . . . 24
- 5.3 Diagrams . . . 24
  - 5.3.1 Metrics depending on combination of features . . . 24
  - 5.3.2 Accuracy VS. false alarm rate . . . 28
  - 5.3.3 Comparison between the algorithms based on metrics . . . 29
  - 5.3.4 Visualisation of the data using t-SNE . . . 32

5.4 Threshold for distinguishing human+bot from bot 33
5.5 Critics about our methodology 34
5.6 Issues faced during the project 34

6 Conclusion **35**

7 Bibliography **36**

8 For us only **36**

1 Introduction

This project aims to develop a classifier that is capable of distinguishing between 3 different kinds of traffic : human, bot and a combination of human and bot. Based on labeled datasets of bots behaviour and human behaviour, we aim to train a model to classify each host into one of the 3 category. This report will outline the methodologies employed, the challenges faced and the results obtained. Additionally, a critical discussion of the result will be given regarding the common pitfalls for learning-based IDS.

1.1 How to run the project ?

Initially, the professor gave us 2 python files, namely `train.py` and `eval.py`. We have restructured the project to enhance its coherence and scalability.

In the following commands, '<algo>' can be replaced by 'decision_tree', 'logistic_regression', 'neural_networks', 'random_forest' or 'knn'.

The different commands that you can use are :

Training the model

Listing 1: Command for training the model

```
python3 train.py \
--webclients ../training_datasets/tcpdumps/webclients_tcpdump.txt \
--bots ../training_datasets/tcpdumps/bots_tcpdump.txt \
--algo <algo> \
--output ../trained_models/<algo>/trained_model_<algo>.pkl
```

where:

- `--webclients` specifies the path to a file containing data related to webclients;
- `--bots` specifies the path to a file containing data related to bots;
- `--algo` specifies the algorithm to use for the training process;
- `--output` defines the path where the trained model will be saved after the training process is completed. It needs to be in the Pickle format (`.pkl`).

Evaluating the model

Listing 2: Command for evaluating the model

```
python3 eval.py \
--trained_model ../trained_models/<algo>/trained_model_<algo>.pkl \
--dataset ../evaluation_datasets/tcpdumps/eval1_tcpdump.txt \
--output ../suspicious_hosts/suspicious_hosts.txt
```

where:

- `--trained_model` specifies the path to a pre-trained model which is in the `.pkl` format;

- `--dataset` specifies the path to the dataset that will be used for the evaluation;
- `--output` defines the path where the result of the evaluation will be saved. This file will contain a list of hosts considered suspicious by the trained model.

Training and evaluating simultaneously

Listing 3: Command for training and evaluating the model at the same time

```
python3 main.py \
--webclients ../training_datasets/tcpdumps/webclients_tcpdump.txt \
--bots ../training_datasets/tcpdumps/bots_tcpdump.txt \
--algo <algo> \
--trained_model ../trained_models/<algo>/trained_model_<algo>.pkl \
--dataset ../evaluation_datasets/tcpdumps/eval1_tcpdump.txt \
--output ../suspicious_hosts/suspicious_hosts.txt
```

The parameters for this command assume the same function as from the training and evaluation of the model.

This subsection only provides a concise guide. For more information and detailed instructions, refer to the main `README.md` file in the project's root directory.

2 Explanations on how we approached the problem

For everyone to start with the same data, the professor provided us with a set of datasets. We received two distinct training datasets : one containing traffic exclusively generated by bots, named `bots_tcpdump.txt`, and another containing traffic from human web clients, named as `webclients_tcpdump.txt`.

Also, we were given two evaluation datasets, each accompanied by a corresponding botlist. The first evaluation dataset, `eval1_tcpdump.txt`, consists of two distinct groups : bots and humans. These groups are guaranteed to be separate sets of hosts. In contrast, the second evaluation dataset, `eval2_tcpdump.txt`, is more complex, where certain hosts emit traffic from both humans and bots.

It's important to note that our machine learning models are exclusively **trained** using the **training datasets** and **evaluated** using the **evaluation datasets**. **There is no mixing of data between the two steps.**

2.1 Our workflow

1. **Understanding and exploring the data**
   - **Exploratory data analysis** : We explored the patterns in the datasets to understand them and identify differences between them. Some of the tasks included analyzing the average query length of the requests and responses, the average number of dots in a domain and some temporal patterns, etc...
2. **Data pre-processing**

- **Feature engineering** : Based on what we found on the exploration data phase, we created new features that we though they would be relevant for classification. See section features selection for more details.
- **Data cleaning** : We handled missing values, especially in cases where an ID is associated only with a response, indicating no corresponding request. In such cases, we removed the data. Additionally, we addressed situations where there were no responses after a request, considering the response as empty and filling it with zeros or 'None' as appropriate.

3. **Classifier selection**
   - **Baseline model** : We started with a simple classifier, such as logistic regression, to establish a baseline and understand the model's behavior and expected results.
   - **Other classifiers** : Then, we explored other classifiers, including decision trees, random forests, neural networks and KNN to compare their performance.
4. **Training and validation**
   - **Model training** : We trained the selected models using the training dataset.
   - **Model evaluation** : To validate the efficacity of our model, we performed a validation process using a dedicated testing set. This evaluation includes the computation of different metrics to measure various aspects of the performance. Among these metrics, we used the detection rate, the false alarm rate, the false positive rate and the true negative rate. Additionally, we used a range of different statistical measures such as accuracy, performance, recall, F1-score (and ROC curve). These metrics will provide an overview of model effectiveness from multiple perspectives, therefore, ensuring a robust assessment.
5. **Diagrams**
   - **Create diagrams** : To compare different metrics for each combination of features and classifiers, we generated different diagrams. This visualization will help in the interpretation and comparison of results.

2.2 Format of DNS queries

The first challenge we encountered involved understanding the format of the provided data, enabling us to identify and extract relevant features necessary for our machine learning models.

We started by examining the structure of a typical DNS query from the tcpdumps that were given. Consider the following line :

```
13:22:44.546969 IP unamur021.55771 > one.one.one.one.domain:  18712+ A? kumparan.com.
(30)
```

Breaking this down, we find :

1. **13:22:44.546969** : This is the timestamp at which the DNS query was logged, indicating the exact date the event occurred.
2. **IP** : Specifies that this log entry is concerning an IP packet.
3. **unamur021.55771** : This is the source of the DNS request. `unamur021` is the hostname of the device making the request and `55771` is the source port number from which the request is coming. Since port numbers are chosen at random, they should **not** be considered as features.
4. **>** : Indicates the direction of the traffic.
5. **one.one.one.one.domain** : This is the destination of the DNS request.
6. **18712+** : This is the query ID for this DNS request. The '+' indicates that this is a recursive query.
7. **A?** : This indicates the type of DNS record being requested.
8. **kumparan.com.** : This is the domain name for which the A record is being requested.
9. **(30)** : This is the length of the DNS request packet in bytes.

When it comes to the **response of a DNS query**, the format remains mostly identical. However, there are some key differences :

- **X/Y/Z** : This is a breakdown of the answer sections in the DNS response (in the code, this will be referred to as `counts`) :
  1. **X** : indicates there are X answers.
  2. **Y** : indicates there are Y authoritative nameservers.
  3. **Z** : indicates there are Z additional records.
- **A 104.18.130.231, A 104.18.129.231** : These are the answers to the A record query for the earlier example query `kumparan.com` and it means that the domain has at least two IPv4 addresses associated with it : `104.18.130.231` and `104.18.129.231`.

Understanding the format of these queries is important for the future feature selection. We must identify features that are relevant and impactful for distinguishing between human and bot traffic.

2.3 Our assumptions and considerations

To effectively address the problem, we made certain assumptions to guide our approach. Some of these will be explain in details later during the pitfalls :

- As stated in the project statement, we exclusively consider the `1.1.1.1` DNS resolver.
- Our analysis is confined to DNS captures and our approach is specifically tailored for DNS data.
- Some responses may appear without a preceding request, which we consider as unusual behavior.

- Label inaccuracy (P2) : We assume that the datasets have been correctly labeled and that the labels are reliable for training our models.
- **Our classifiers categorize both bots and instances of mixed bot and human behavior as 'bot'**. We believe that an infected computer, regardless of whether it is used by a human, should be classified as a 'bot' but with a certain threshold. Thus, non-infected instances are labeled as 'human'. For a visual representation, refer to the Figure 1.
- For our purposes, a **'session'** encompasses the entire trace. When we introduce new features based on aggregated data , a **'user-session'**, (it does not matter whether the user is a bot or human) spans from the initial request sent to the final response received, regardless of whether the user is a bot or a human.
- We observed that in the provided datasets, sometimes there were flags from TCP connections. However, we decided that these flags would not be taken into account during our analysis and parsing, as the focus of this project is to flag suspicious hosts based on **DNS traces**.

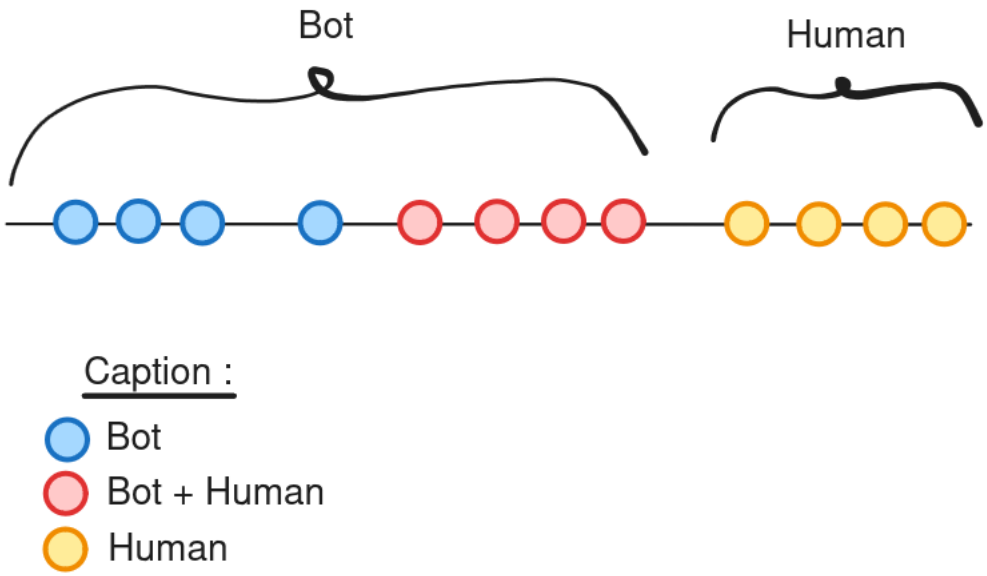

Figure 1: Simplified logistic regression for classification

2.4 Features selection

The development of an effective IDS based on ML requires careful consideration of the features that will be used for classification. Using only the raw features present in the datasets would not be optimal, as detecting bot **behavior** requires more than just examining **individual** queries (either requests or responses) in isolation.

Thus, to enhance the efficiency of our model, we opted to aggregate several raw features extracted during the data pre-processing phase. These aggregated features provide a more

accurate representation of host behavior.

To identify the distinguishing features, we examined the two training datasets by hand, aiming to find differences between human and bot traffic patterns. We found that:

- Human traffic tends to occur in bursts, concentrating activity within a few minutes and often on similar websites.
- Bot traffic, in contrast, is persistent yet sporadic. They perform requests at regular intervals (e.g., every 10 seconds) across a diverse range of websites.

2.4.1 Aggregation/combination of features

In our initial step, we begin by combining **each request-response pairs** based on their **ID**. If there are different requests-response pairs that have the same ID (it could be the case since we have a lot of data inside the dataset, and therefore there is a probability to have the same ID for two distinct pair of request-response), we simply **append to the end of the id a number** and each time we see again the same ID we **increment** this appended number. Once this completed, we aggregate **each pairs** to create a **behaviour** profile for **each host**. Following this aggregation, we then identify and extract useful features that will be explained below.

We created 3 different categories of features :

- **Features related to numbers**
  - **average number of dots in a domain** : Calculating the average number of dots in a domain queried by a host. Bots often query domains with **a single dot** (e.g., kumparan.com, fast.com), while web clients query **more complex domains**, including those queried by browsers (e.g., location.services.mozilla.com, services.addons.mozilla.org). The average number of dots serves as a distinguishing feature between bots and human clients.
  - **number of requests in a session** : Determining the number of requests in a session for each host. We define a user-session as the time between the initial request sent and the final response received. Web clients average around **1062** requests per session, whereas bots average approximately **22**. This very significant difference highlights the more active traffic of web clients compared to bots.
  - **number of unique domains** : Counting the number of unique domains queried by each host. We found that the bots query each time different domains compared to webclients who query in the most case several time usual domains. On average, web clients have around **275** unique domains, while bots have approximately **22**. Interestingly, the number of unique domain is almost the same as the number of requests in a session for the bots. Indeed, it confirms the assumption where almost all bots query each time different requests.
  - **average counts** : Computing the average number of answers received in a response for each host. This feature reveals that web clients receive an average of 1 **answer**

per request, while bots receive approximately 2 answers per request. Bots tend to request more answers than web clients for a single query.

- **Features related to time**
  - **average time for a session** : Determining the average time for a **user-session**. As a reminder, we defined a user-session as a the timing spent between the initial request sent by the host in the trace and the last response received by the host in the trace. The session duration for bots is approximately **875.78 seconds**, while for web clients, it is around **1113.61 seconds**.
  - **average time between requests** : Calculating the average time between each request for each host. We discovered that taking the average time between each request made could be a good feature. Indeed, the bots and webclient do not send requests at the same frequency. We therefore take all the difference between each request. We found that webclients send request every **2** seconds in average whereas bots send every **40**. We concluded that the traffic for a webclient is more active that the traffic of a bot.
  - **frequency of repeated requests in a short time frame** : Measuring the frequency of repeated requests for the same domain within a short time frame for each host. This feature helps identify if a host makes consecutive requests for the same domain within a short time. Bots query the same domain at a frequency of **0.0119** on average within a short time frame, while web clients query at a frequency of **0.9715**. It is coherent since we found that the majority of bots does query the same request twice since the number of request and unique domain is in average nearly equal as seen before.
- **Miscellaneous features**
  - **average of request length** : Calculating the average length of each request sent by each host. On average, request lengths for bots are approximately **30**, while web clients have an average request length of around **40**, which makes it a good distinguishing feature.
  - **average of response length** : Similar to request length. On average, response lengths for bots range between **60 and 70**, while response lengths for web clients **vary widely**.
  - **type of requests queried by hosts** : Categorizing the types of queries made by hosts (e.g., "A", "AAAA"). It can be a good feature to separate bot from webclients since, we noticed that in the training dataset, bots tend to query only "A" requests, while web clients exhibit more diverse query types.
  - **type of responses received by hosts** : Categorizing the types of responses received by hosts (e.g., "A", "AAAA", "NXDomain", "CNAME", "ServFail"). This feature highlights the different response types received by hosts and aids in distinguishing between bot and human behavior. Web clients receive a variety of response types, while bots receive limited types.

An important observation we made and one that we were careful not to fall into, concerns the handling of timestamps. It is very important to emphasize that we should **NOT** place excessive emphasis on **individual timestamp values**. Doing so would introduce a bias into our model :

1. **Timestamps for bot tcpdump**
   - First timestamp of the trace : 13:22:44.546969
   - Last timestamp of the trace :14:35:23.695938
2. **Timestamps for webclients tcpdump**
   - First timestamp of the trace : 11:44:12.890000
   - Last timestamp of the trace : 13:16:33.136055

Instead of focusing on individual timestamp values, we incorporated them as part of a broader context, such as a session or an aggregation between a request and a response. In some sense, we prioritized a session-level structure over a query-level one.

To prepare for our future machine learning algorithms, we will need some relevant combination of features. However, with a total of 11 features to consider, attempting an exhaustive search for the best combination would be testing **2,047 possibilities**. In practical terms, this computation would be extremely time-consuming, even on powerful commercial-grade computers.

To put this into perspective, on Marco's computer, a **single run takes approximately 25.5 seconds**. When multiplied by 2,047 (25.5 * 2,047), the total time required amounts to **52,198.5 seconds**, which translates to roughly 869,975 minutes or about **14.5 hours for a single algorithm**. Given that we have **5** algorithms to evaluate, this process would demand a cumulative time of **72.5 hours** just to compare and select the optimal feature set. Unfortunately, as students, we do not have that much resources.

Instead, we adopted a more efficient approach by grouping together features that **logically made sense to us**. This led us to identify 7 distinct feature groupings that were most suitable for our project :

1. **Time-related features** :
   - **Average time for a session**
   - **Average time between requests**
   - **Frequency of repeated requests in a short-time frame**

   Time-related features are interesting for distinguishing between bot and human traffic because they capture the behavioral cadence and rhythm of interactions with a network, which often differs between bots and human activities. Therefore, by understanding the temporal patterns unique to bots and humans, an IDS can be more effective in flagging suspicious activity and protecting the network against automated threats.
2. **Numbers-related features** :

- **Average number of dots in a domain**
- **Number of requests in a session**
- **Number of unique domains**
- **Average counts**

Numbers-related features are essential in understanding the structural and quantitative aspects of network interactions. They provide different measures regarding the kind of domains with which hosts interact, which can be crucial for identifying the systematic behavior of bots and human users. By analyzing these numerical features, an IDS could gain insights into the behavior and intentions behind the network traffic. A model trained on these features could, therefore, differentiate between bots and humans based on patterns like domain complexity, domain diversity and the nature of the DNS interactions.

3. **Miscellaneous features** :
   - **Average of request length**
   - **Average of response length**
   - **Type of requests queried by hosts**
   - **Type of responses received by hosts**

   Miscellaneous features provide a mixture of qualitative and quantitative data about the requests and responses. These features are interesting because they encapsulate both the content and context of network interactions, providing a deeper look into the communication patterns between hosts and the network. By understanding the typical lengths and types of requests and responses, as well as their variances, an IDS can more accurately characterize and classify the network traffic, leading to more effective identification of bot-related threats.

4. **High-level behavioral features**:
   - **Average number of dots in a domain** : Differentiates based on domain query patterns.
   - **Number of requests in a session**: Reflects the activity level of the host.
   - **Average time for a session** : Indicates the overall session duration and can reflect user engagement or bot operational patterns.
   - **Frequency of repeated requests in a short time frame** : Helps to distinguish bots that may repeatedly query the same domain in automation tasks.
   - **Type of requests queried by hosts** : May highlight the simplicity or complexity of queries, indicative of automated or human-driven behavior.

   This combination focuses on general patterns of host behavior, capturing both the complexity of their web activity and the intensity of their interactions with domains. Features like the average number of dots in a domain and the type of requests depicts

the complexity of user browsing patterns against the more straightforward patterns of bots. The number of requests in a session and the average session time provide a really good insight of user activity in a session, where more requests and longer times likely indicate human activity. The frequency of repeated requests within a short timeframe could give an idea of whether the same domain is repeated in a given timeframe, which highlight the difference of frequency between bots and humans.

5. **Domain interaction and traffic patterns features**:
   - **Number of unique domains** : Bots tend to query different domains, providing a clear behavioral distinction.
   - **Average counts** : Differentiates based on how many answers are typically requested, which could signify automated processes.
   - **Average time between requests** : Indicates the regularity or irregularity of traffic, which can be a telltale sign of automated querying.
   - **Average of request length** : This could serve to differentiate between the complexity of interactions.
   - **Type of responses received by hosts** : Varied responses might indicate a more human-like interaction with the network.

   This combination give some insight about the interaction between the host and the domains it contacts, alongside the traffic patterns. It captures the diversity and frequency of domain queries (number of unique domains and average time between requests) to distinguish between the broader exploration of domains from humans and bots. The average request length can indicate the level of detail in the queries. The type of responses indicates whether the host's behavior solicits a broad or limited set of responses.

6. **Response-based and session duration features**:
   - **Average of response length** : Suggests the complexity or nature of the information being exchanged.
   - **Average time for a session** : Prolonged sessions might be more human-like, whereas shorter could imply bot activities.
   - **Average time between requests** : Shorter times could imply automated processes and longer times more human behavior.
   - **Frequency of repeated requests in a short time frame** : High frequency may indicate bot behavior.

   This combination of feature focuses on the informational content and timing of sessions to discern bots from humans. Average response length and average time for a session are used to measure the depth and duration of interactions. The average time between requests offers insights into the rhythm of traffic, where irregular intervals might suggest human browsing, and regular ones could indicate bot operations. And, as explained before, the frequency of repeated requests within a short timeframe could help us in identifying the frequency between bots and humans wihtin a timeframe.

7. **Detailed request and response patterns features**:
    - **Type of requests queried by hosts** : Different query types might be indicative of automated or diversified human activity.
    - **Type of responses received by hosts** : A narrower range of responses might indicate bot activity.
    - **Average of request length** : Reflects the complexity of the queries made by the host.
    - **Average of response length** : Could indicate if a bot is programmed to carry out specific tasks that receive standard responses.

    This last combination is useful to capture the nuanced details of host interactions through the type of DNS requests and responses and their lengths. The diversity in the types of requests and responses provides a overview of a host's network behavior, distinguishing patterns of human users from those of bots. The average lengths of requests and responses add an additional layer of detail by providing insights about the length of queries.

For the future diagrams and comparisons of algorithms, we choose to stick with these combinations and also with a **baseline combination containing all the features** defined before.

3 Background knowledge

3.1 Slides from the course

Prior to the project, we had a whole lesson concerning IDS and about 2 critical aspects of IDS :

- **Effectiveness** : This dimension measures the accuracy with which an IDS **correctly** identifies intrusions within network traffic.
- **Security** : We discussed around the notion that high accuracy alone does not guarantee security. In particular, we emphasized the significance of **bayesian detection rates** and underscored the counter-intuitive nature of **the base-rate fallacy**. This fallacy suggests that even with a high overall accuracy, a system can still yield a considerable number of false positive alarms. Such a scenario is undesirable and we must be mindful of it.

Summing up the course insights, we arrived at a fundamental understanding that :

`"To be effective an IDS must achieve a high level of accuracy, but to be truly secure, it must also attain a high bayesian detection rate."`

3.2 Articles

In addition to the course materials, we had to read further research by examining 2 articles closely linked to the project :

1. **The base-rate fallacy and the difficulty of intrusion detection**[2] : This article introduce and mainly focuses on the base-rate fallacy, a non-intuitive concept in the field of probability that significantly impacts the effectiveness of IDS. The key objective for an IDS is not only to detect as many actual intrusions as possible but also to limit the number of false positives. The paper argues that the frequency of these false positives is a major limiting factor in an IDS's performance, with the base-rate fallacy playing a crucial role in this limitation. Essentially, the base-rate fallacy happens when someone is overlooking the base rate of the least frequent class in cases of class imbalance, therefore, it can distort the perceived effectiveness of a predictive model. In situations where the negative class is predominant, even a model with a low false-positive rate can produce a surprisingly large volume of false positives, undermining the model's apparent accuracy.

   We are going to develop in depth the base-rate fallacy and its implications in the next section.

2. **Dos and Don'ts of machine learning in computer security**[1] : This paper examines the application of machine learning in the field of computer security. The authors identify common pitfalls in the design, implementation and evaluation of learning-based security systems through a study of 30 papers from top-tier security conferences over the past decade. They find that these pitfalls are widespread and can lead to over-optimistic results, undermining the performance and practical deployment of ML in security tasks.

   The authors categorize ten common pitfalls into different stages of the ML workflow :

   - **Data collection and labeling** :
     - **Sampling bias (P1)**
     - **Label inaccuracy (P2)**
   - **System design and learning** :
     - **Data snooping (P3)**
     - **Spurious correlations (P4)**
     - **Biased parameter selection (P5)**
   - **Performance evaluation** :
     - **Inappropriate baseline (P6)**
     - **Inappropriate performance measures (P7)**
     - **Base rate fallacy (P8)**
   - **Deployment and operation** :

– **Lab-only evaluation (P9)**

– **Inappropriate threat model (P10)**

The paper emphasizes the importance of recognizing and mitigating these pitfalls to avoid biased results and over-optimistic conclusions. It provides actionable recommendations for researchers to support the avoidance of these pitfalls where possible.

3.3 Pitfalls

As stated earlier, from the second article[1], we learnt a lot about pitfalls in machine learning. Understanding these pitfalls is really important for our project since we need to take them into account during our analysis and it will serve as a shield against making erroneous assumptions.

In the next sections, we will dive deeper into the specifics of these pitfalls, how they apply to our project and what we did to identify them.

OSEF

for US, from the article, the most common is : P1 : 90% =¿ we got it P3 : 73% =¿ we got it P10: more than 50% P9 : more than 50% P7 : more than 50%

donc c'est surement les plus probable qu'on ait dans notre projet à nous aussi, donc faut faire gaffe

/ OSEF

3.4 Data collection and labelling

The first step for the design and development of a machine learning IDS is to obtain a representative dataset. From this stage, we need to consider the two following pitfalls : **Sampling bias** and **Label inaccuracy**.

3.4.1 Sampling bias

Sampling bias in the context of an IDS refers to a systematic error that occurs when the data used to train or test the IDS is not representative of the true population of network traffic or security events that the IDS will encounter in operations.

In our case, we used datasets provided by the professor, namely 'bots_tcpdump.txt' and 'webclients_tcpdump.txt'. The question we must ask ourselves is, "Do these datasets genuinely mirror real-world practices (which in our case are the dataset evaluations) ?" If there are not representative of the real-world distribution of human and bot DNS traffic, then our model might not perform well in practice.

Remarks :

- The limitation of the datasets provided lies in the fact that it has a limited number of DNS traces which do not really represent the true data distribution. Indeed, these traces are restricted to a specific range of hours (within a single day) and as a result,

they do not adequately represent the whole network traffic. While efforts can be made to mitigate this bias, it is practically impossible to entirely represent the whole network behavior across all possible scenarios. Since this was given by the teacher, we cannot reduce this issue.

- Additionally, the bots identified within the DNS trace demonstrate a specific pattern of DNS queries. Thus, if a different type of botnet occurs in the network, the IDS might not successfully detect these bots, as they would likely generate a distinct pattern of requests. As a result, the data would differ, potentially causing our classifier to miss the wider range of bot behaviors.

Very important remark : in the project statement, we are tasked with assessing our classifier on the **eval2** dataset, where hosts may behave like **bots at certain times** and **humans at others**. However, the training datasets we have been given only contain distinct instances of bot behavior and human behavior, without any overlap. It will have a significant impact on the performance of the model, more details will be given later.

3.4.2 Label inaccuracy

Label inaccuracy refers to the incorrectness of the labels assigned to data points in a dataset. Accurate labels are really important for classification machine learning algorithms since they rely on these labels to learn the relationship between the input data and the desired output. If it fails to do so, the overall performance of the model will be affected.

Given that the training datasets were provided by our professor, it is reasonable to assume that the ground-truth labels are accurate because if they are not, it would necessitate to relabel all the datasets which would be very challenging. Consequently, we are making the assumption that the professor has correctly labeled the datasets.

Remarks :

- If the labeled datasets were classified based on certain heuristics or assumptions, there might be mislabeling. For example, a host with unusual web requests that could be performed by a human might be labeled as a bot.
- Additionally, if the bots change their behaviour after deploying the IDS. This could introduce a bias known as label shift (for further details, see this article[1]). The classifier will not adapt to these changes which will experience performance decay in real scenarios.

3.5 System design and learning

With the data in hand, we can proceed to design and train a learning-based IDS. The goal is to process and use the meaningful features that are interesting for our project.

3.5.1 Data snooping

Data snooping is a phenomenon that occurs when a model is trained with information that is not available in practice. This typically happens when the test set is employed

for experimental purposes before the final evaluation. Data snooping can result in overly optimistic results, where the model appears to perform exceptionally well on its training data but fails to generalize effectively to new, unseen data.

For the sake of completeness, in the article titled "Dos and Don'ts of Machine Learning" [1], 3 types of data snooping are distinguished : **test snooping**, **temporal snooping** and **selective snooping**. The overview of these snooping types can be found in Table 8 of the article :

- **test snooping** :
  - Preparatory work : This aspect is not applicable to our project. We have ensured that the test set is used **ONLY** for the final model evaluation. No additional knowledge is gained from it during the learning process. Our feature selection was based only on the training datasets without any insights from the evaluation datasets.
  - K-fold cross-validation : Given our dedicated evaluation datasets, we do not use K-fold cross-validation in any of our algorithms.
  - Normalization : We have not used any normalization functions in our project, although we discuss this point as a potential improvement in the critics section.
  - Embeddings : Our neural network algorithm is trained only on training data and not on the training+test dataset.
- **temporal snooping** :
  - Time dependency : As stated earlier in the report : "Instead of focusing on individual timestamp values, we incorporated them as part of a broader context, such as a session or an aggregation between a request and a response. In some sense, we prioritized a session-level structure over a query-level one". Although, if bot behavior is specifically time-dependent (for example, bots that manifests at particular times of the day), our model will likely not be able to detect it since it doesn't consider individual timestamps for temporal analysis.
  - Aging datasets : In our case, this aspect is not applicable, as we use a provided dataset that is not publicly available and subject to aging.
- **selective snooping** :
  - Cherry-picking : We do not eliminate any data from our training datasets.
  - Survivorship bias : When parsing the datasets, we do not perform any data cleaning or filtering that could introduce survivorship bias.

In summary, while we have taken measures to prevent various forms of data snooping, we note that the only type of snooping that could potentially pose an issue for our project is **"time dependency"** due to our focus on session-level structures instead of individual timestamps.

3.5.2 Spurious correlations

Spurious correlations involve relying on apparent data relationships that occur "by chance". In the course of our project, we needed to select specific features that we believed were important for distinguishing between bot and human behavior. Then, we aggregated these features into sets that appeared meaningful for our objective of distinguishing between the two. We **intentionally exclude** features such as IP addresses or host to ensure that the model's selection process is not influenced by these factors.

3.5.3 Biased parameters selection

This pitfall involves selecting the right **hyperparameters** during training based on the testing dataset to achieve high accuracy. However, it's important to recognize that the IDS may behave differently in real-world situations.

In our project, as mentioned earlier, we initially did an aggregation of the most relevant features, guided by our assumptions and background knowledge. Then, we explored several combinations of these aggregated features to identify the model with the best performance. The model's performance is then assessed using the **test set**. However, we did not perform fine-tuning on hyperparameters. Most of our hyperparameters are left to the default ones because, in our case, knowing that we have several algorithms and that each of them has its own hyperparameters, we would have to perform a grid search on **all** the hyperparameters of **all** the algorithms, which would be very time-consuming and ressource demanding. We therefore decided to leave the hyperparameters to their default values.

In the case, we would had adjusted these settings, we need to be careful to perform the grid search on the training set and not on the test set. Indeed, if we perform the grid search on the test set, we would have a bias on the test set and the performance of the model would be overestimated.

3.6 Performance evaluation

The way that we evaluate our model can greatly influence the outcome and lead to misleading and biased results.

3.6.1 Inappropriate baseline

To show to what extent a novel method improves the state of the art, it is vital to compare it with previously proposed methods.

In our project, we decided to create 5 different models and see how they perform by comparing them side by side since there exists no universal algorithm that outperforms all the other. We will evaluate the different models not only on accuracy but also on precision, recall, F1-score and other relevant metrics such as detection rate, false alarm rate, etc.

Indeed, if we directly choose a complex learning method it will increase the chance of overfitting and will raise also other problems such as security, performance, etc. Thus, for a baseline model, we choose a straightforward algorithm namely `logistic regression`.

3.6.2 Inappropriate performance measures

When assessing a model, choosing the **right performance metric** is very important, especially for security-related applications. In our situation, where we aim to develop an IDS, relying on a single performance metric is inadequate, **particularly with imbalanced classes**. We have used accuracy (more details will be given in the next sections) as a measure, but this will not be ideal in our case as it does not accurately reflect the rate of true positive and false positive predictions. As a matter of fact, in our datasets, there are 5,474 entries for bots and 343,835 for web clients which clearly indicates imbalanced classes. As mentioned previously, we preferred metrics like **detection rate**, **false alarm rate**, **false negative**, **true negative**, **precision**, **recall** and **F1-score** that are more appropriate for our project.

3.6.3 Base-rate fallacy

The base-rate fallacy occurs when the prevalence of a class (in our case, the base rate of the negative class which are legitimate hosts) in the dataset is not correctly considered while evaluating the performance of the IDS. More precisely, if the class is predominant, even a very low false-positive rate can result in a surprisingly high numbers of false positive. This can lead to a biased perceptions of the system's effectiveness.

For example, in our dataset, the large majority of entries are legitimate webclients, as indicated by the 343,835 entries for `webclients_tcpdump.txt` compared to just 5,474 for `bots_tcpdump.txt`. In such a scenario, even if the IDS has a low false-positive rate, the scale of the volume of legitimate traffic can result in a significant number of false positives. This can be problematic as it could lead to unnecessary alarms.

Moreover, the high accuracy score that might be obtained in such an imbalanced dataset can be misleading. A model might achieve high accuracy by simply predicting the majority class (in our case webclients), but this doesn't mean it is effective in identifying actual threats (in our case bots).

The ideal IDS should balance the following:

- **Detection rate (true positive rate)** $(P(D|B))$: high probability of correctly identifying actual threats.
- **False alarm rate (false positive rate)** $(P(D|\neg B))$: low probability of mistakenly identifying legitimate traffic as a threat.
- **False negative rate** $(P(\neg D|B) = 1 - P(D|B))$: low probability of failing to detect actual threats.
- **True negative rate** $(P(\neg D|\neg B) = 1 - P(D|\neg B))$: high probability of correctly identifying legitimate traffic.

where:

- B and $\neg B$: bot or not bot behaviour
- D and $\neg D$: detection as bot or human by the IDS

In essence, an effective IDS must minimize both false positives and false negatives. This requires a good understanding of the dataset's characteristics and the selection of relevant metrics.

Considering the base-rate fallacy, it is important to use metrics like precision, recall and the F1-score in addition to accuracy. These metrics provide a more global view of the model's performance, especially in the context of imbalanced datasets.

3.7 Deployment and operation

This stage marks the final step where we deploy our solution to address the security problem in practical scenarios.

3.7.1 Lab-only evaluation

Our approach is susceptible to the "Lab-only evaluation" pitfall. The analysis is conducted and tested in a highly controlled environment, focusing only on DNS queries. In the real world, the landscape is far more diverse and unpredictable. Our evaluation relies on the provided datasets and real-world performance may exhibit variations. For instance, the "timestamps" feature is limited to a one-hour duration in our dataset. If bots exhibit different behavior patterns outside this timeframe, our IDS may struggle to classify them effectively.

3.7.2 Inappropriate threat model

Our learning-based algorithm is not vulnerable to certain attacks such as adversarial preprocessing or poisoning because we used a closed dataset provided by the professor.

Nevertheless, we did make certain assumptions, such as trusting the professor's datasets. If the initial datasets were tampered with or poisoned, there is little we can do and our model could be compromised. Additionally, assumptions that we made about distinguishing humans from bots, if incorrect, may affect our model's performance.

3.8 Summary of the pitfalls

This section provides a summary of the 10 pitfalls and if yes or no we fall into them

1. **Sampling bias** : **Yes**, we have this pitfall.
2. **Label inaccuracy** : **No**, we do not have this pitfall.
3. **Data snooping** : **Yes**, we have this pitfall.
4. **Spurious correlations** : **No**, we do not have this pitfall.
5. **Biased parameters selection** : **No**, we do not have this pitfall.
6. **Inappropriate baseline** : **No**, we do not have this pitfall.
7. **Inappropriate performance measures** : **No**, we do not have this pitfall.

8. **Base-rate fallacy** : **Yes**, we have this pitfall.
9. **Lab-only evaluation** : **Yes**, we have this pitfall.
10. **Inappropriate threat model** : **No**, we do not have this pitfall.

4 Algorithms and techniques used

In selecting the algorithms for this project, our initial step involved determining the most suitable intrusion detection strategies to implement. The seminal work on the base-rate fallacy by Axelsson [2] highlighted 3 major types of intrusion detection : **anomaly**, **signature** and **classical**.

4.1 Our choice

We opted to focus on **anomaly detection** as our primary approach. Anomaly detection is a technique that identifies deviations from expected behavior within a system or dataset. In the context of our project, it is particularly valuable because it allows us to differentiate between human-generated and bot-generated traffic without relying on predefined "signature" files. This is especially pertinent as we are not provided with such signature files in our dataset.

Anomaly detection, as a machine learning approach, is well-suited to handling cases where the behavior of bots may vary and evolve over time, making it challenging to create precise signatures for them. Instead, anomaly detection algorithms learn the baseline behavior of the network and can identify deviations from this norm. These deviations may indicate the presence of bots, which often exhibit patterns that deviate significantly from human traffic.

4.2 Baseline algorithms

From our machine learning course during the first year of our master, we chose to concentrate on the following algorithms :

Algorithm 1 : Logistic regression

Logistic regression is often the starting point for machine learning tasks, particularly in classification problems. We used it to establish our performance baseline, appreciating its simplicity and interpretability.

Algorithm 2 : Decision tree

Our choice of the decision tree algorithm was motivated by its simplicity, offering a simple yet effective method for classification tasks.

4.3 Other and more complex algorithms

Algorithm 3 : Random forest

After using decision trees, we moved on to random forests to improve performance beyond what a single decision tree can achieve. Based on our knowledge gained in machine learning courses from last year, random forests mitigate the over-fitting problems typically associated with decision trees.

Algorithm 4 : Neural networks

Neural networks represent a substantial increase in complexity over our previous models. We wanted to explore the performance of our model with a more advanced algorithm. Neural networks are known for their ability to understand complex patterns in data, functioning as black boxes.

Algorithm 5 : KNN

We chose the K-nearest neighbor (KNN) method because of its simplicity and straightforward implementation.

5 Discussion

"to be effective, the IDS must have high accuracy but to be secure, IDS must have high BDR"

5.1 Definition of metrics used

1. **Precision** : This metric answers the question : "Of all the instances the model labeled as a particular class, how many were actually that class" ? It's calculated as :

$$\text{Precision} = \frac{\text{True positives}}{\text{True positives} + \text{False positives}}$$

2. **Recall** : This metric answers the question : "Of all the actual instances of a particular class, how many did the model correctly identify"? It's calculated as :

$$\text{Recall} = \frac{\text{True positives}}{\text{True positives} + \text{False negatives}}$$

3. **F1-Score** : This metric provides a balance between precision and recall. It's particularly useful when the class distribution is uneven. It is the harmonic mean of precision and recall:

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

4. **Support** : This shows the number of actual instances for each class in the dataset that was used to compute these metrics.

5. **Accuracy** : This gives the overall proportion of predictions that were correct, across both classes. It's calculated as :

$$\text{Accuracy} = \frac{\text{True positives} + \text{True negatives}}{\text{True positives} + \text{False positives} + \text{False negatives} + \text{True negatives}}$$

6. **Macro avg** : This averages the unweighted mean per label. In the code, precision, recall and F1-score are each averaged, giving equal weight to both classes. It is particularly informative as there's a class imbalance, as it treats both classes equally.
7. **Weighted avg** : This averages the support-weighted mean per label. Metrics are calculated by considering the number of true instances for each label (support). This gives a weighted average which is more representative when there's a class imbalance.

"Accuracy is seeing frequent usage and conceptually it makes sense. However, formulaically, its reliance on TN in the numerator, giving it equal weight to TP, is invalid in cybersecurity due to the high volume of TNs relative to TPs and their nonexistent value. It's FPs that are a concern, in volume, while TNs are completely irrelevant. Essentially, the issue is a deviation between a theoretic assessment of the technique and a real-world assessment. From a purely theoretical point of view, accuracy appears to be a relevant and completely applicable metric. From the **real world**, cybersecurity point of view, **accuracy** is completely **irrelevant**. The problem being that accuracy assumes all samples are of **equal weight**. Since that isn't the case, **accuracy cannot be used**." [3]

For example, in the following case, we have an accuracy equals to 84.16%, at first glance, it seems to be a very good model, however, it is not the case. Indeed, if we take the number of true positive, we observe that it is equal to 2, in our case it is a very bad result because in a cybersecurity point of view, we need an IDS that detect as much as possible bots. In the presented case, we don't have this. It is biased because of the True Negative since, we have a lot more samples for humans, the accuracy is quite biased.

Listing 4: bash version

```
####
False alarm rate and detection rate...
Detection rate : 16.666666666666664 %
False alarm rate : 8.333333333333332 %
False negative rate : 83.33333333333334 %
True negative rate : 91.66666666666666 %
Accuracy : 84.16666666666667 %
####
Total host :  120
Number of true positive :  2
Number of false positive :  9
Number of false negative :  10
Number of true negative :  99
####
```

$$\text{Accuracy} = \frac{2+99}{2+9+10+99} = 0.8416666666666667$$

! A VOIR ! if we are more interested in detecting bots without caring at the false positive, we could use **recall**, if we are interested in the precision of the IDS to see if all the hosts

classified as bot are really bots, we can use precision ! A VOIR !

5.2 Eval2 : evaluation dataset

expliquer pourquoi on aura jamais des bons résultats sur eval2, parler qu'on a pas de datasets pour s'entrainer et vu que cest "l'entre-deux", on aura de toute facon des résultats de merde

+ trouver d'autres idées

5.3 Diagrams

For the next diagrams, we selected the `logistic regression` algorithm as a representative choice for generating diagrams due to its simplicity and effectiveness for binary classification tasks. Logistic regression serves as an illustrative example of how our approach works.

In the subsequent subsections, we will consistently present the same set of diagrams, each analyzed using two different evaluation datasets : `eval1` and `eval2`. It will allow us to do a comparison of the model's performance under varying conditions.

5.3.1 Metrics depending on combination of features

-¿ On a plusieurs combiniason, qd on a testé combi1 c based sur eval set parce qu'on a vu qu'il a une accuracy de 97,5%, mais mnt avec le code feature selecion, c testé sur training set ouais.

For eval1

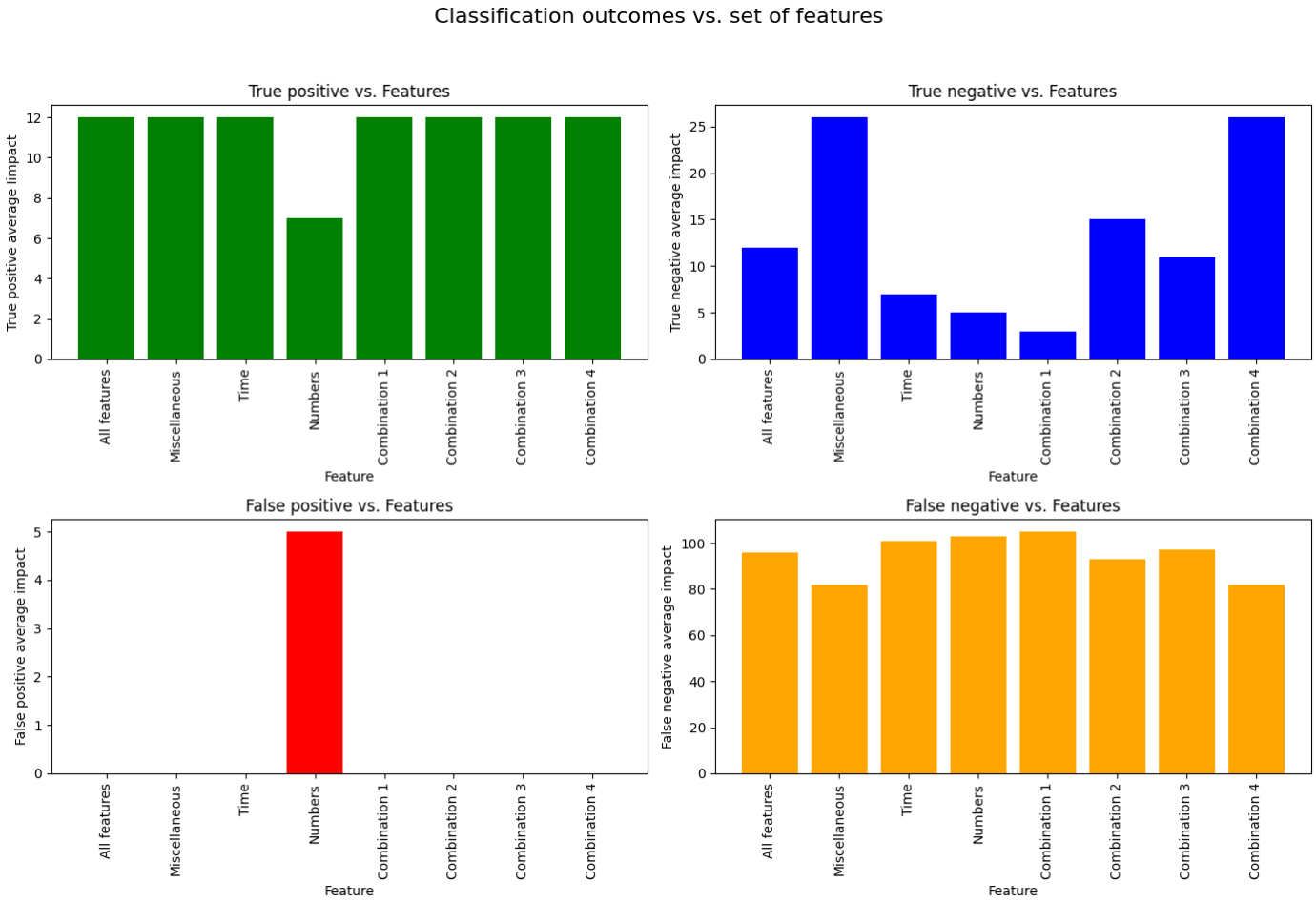

Figure 2: Classification outcomes vs. set of features on eval 1

FAIRE DES CRITIQUES SUR LES DIAGRAMS ICI

For eval2

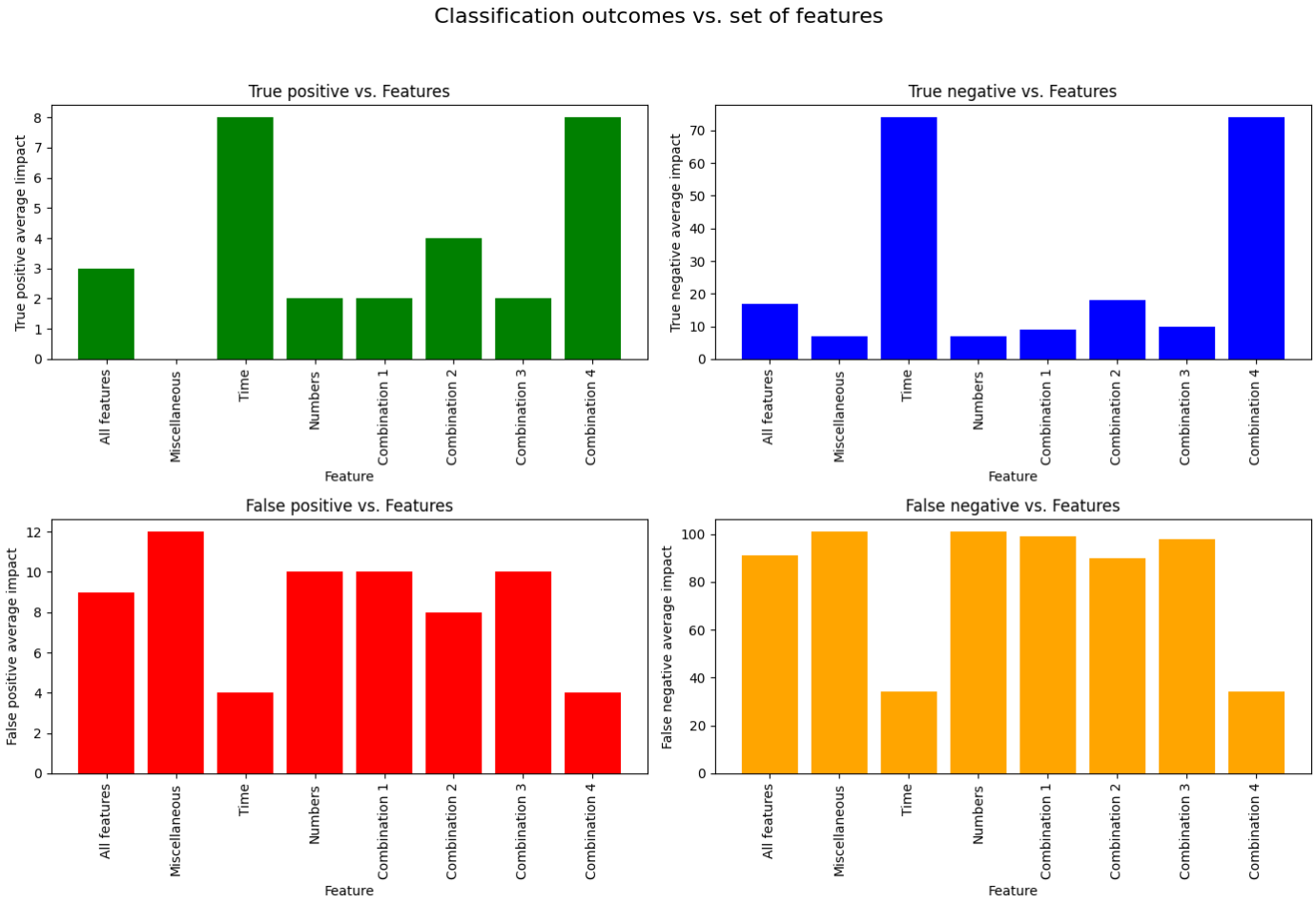

Figure 3: Classification outcomes vs. set of features on eval 2

THEY ARE NOT POURCENTAGES !!! FAIRE DES CRITIQUES SUR LES DIAGRAMS ICI

GEPETO :

From the report, it's clear that the model does a good job at identifying bots (recall of 99% for "bot"), but struggles significantly with identifying humans (recall of 2% for "human"). The overall accuracy of 57% might seem decent, but the low recall for the "human" class and its resulting F1-score show that the model's performance is imbalanced and needs improvement, especially in correctly identifying humans.

RESULTS GéPéTO !!

Certainly! The data you've provided shows the results of an Intrusion Detection System (IDS) that classifies hosts as either 'bot' or 'human' based on different combinations of features. Each combination of features results in a different performance of the classification model. Let's break down the terms and the results for each combination:

Results for Each Combination:

Combination 1: High-Level Behavioral Traits - **Detection Rate**: Perfect at 100- **False Alarm Rate**: Very low at approximately 2.78- **Accuracy**: Very high at 97.5- **Precision for 'bot'**: High at 0.8, indicating that when it predicts a host as a bot, it is correct 80- **Recall for 'bot'**: Perfect at 1.0, meaning it caught all bots. - Overall, this combination has excellent performance.

Combination 2: Domain Interaction and Traffic Patterns - **Detection Rate**: Still perfect at 100- **False Alarm Rate**: Higher at approximately 13.89- **Accuracy**: Lower at 87.5- **Precision for 'bot'**: Significantly lower at 0.444, meaning there are quite a few false positives. - **Recall for 'bot'**: Still perfect at 1.0. - This combination has a good detection rate but suffers from a higher rate of false alarms.

Combination 3: Response-Based and Session Duration Features - **Detection Rate**: Again, perfect at 100- **False Alarm Rate**: Moderate at approximately 10.19- **Accuracy**: Good at 90.83- **Precision for 'bot'**: Moderate at 0.522, better than Combination 2 but still with room for improvement. - **Recall for 'bot'**: Perfect at 1.0. - This combination offers a balanced performance with a moderate false alarm rate.

Combination 4: Detailed Request and Response Patterns - **Detection Rate**: Perfect at 100- **False Alarm Rate**: Quite high at 24.07- **Accuracy**: The lowest among the combinations at 78.33- **Precision for 'bot'**: Low at 0.316, indicating a lot of false positives relative to true positives. - **Recall for 'bot'**: Perfect at 1.0. - Although it detects all bots, this combination has the highest false alarm rate and the lowest overall accuracy.

Conclusion: - The first combination of features provides the best balance with the highest accuracy and a very low false alarm rate. - Combinations 2 and 3 provide a trade-off between the detection rate and false alarm rate. - Combination 4, while having a perfect detection rate, suffers significantly in terms of precision due to a high false alarm rate.

It's important for the security team to choose the right combination based on what's more critical for their context - catching every bot (high detection rate) or reducing false alarms (high precision). There's always a trade-off between detecting as many bots as possible and not misclassifying human users as bots.

Listing 5: bash version

```
####
False alarm rate and detection rate...
Detection rate : 100.0 %
False alarm rate : 11.11111111111111 %
False negative rate : 0.0 %
True negative rate : 88.88888888888889 %
####

Our own accuracy :
Accuracy : 90.0 %
```

-¿ we don't have any false negative ! our model detects successfully all the real bots. However, it misclassifies some human as bots, false positive rate = 11.11111111111111, hence the true negative rate = 88.88888888888889.

5.3.2 Accuracy VS. false alarm rate

Le but de ces diagrams est de montrer que l'accuracy n'est pas une valeur fiable

For eval1

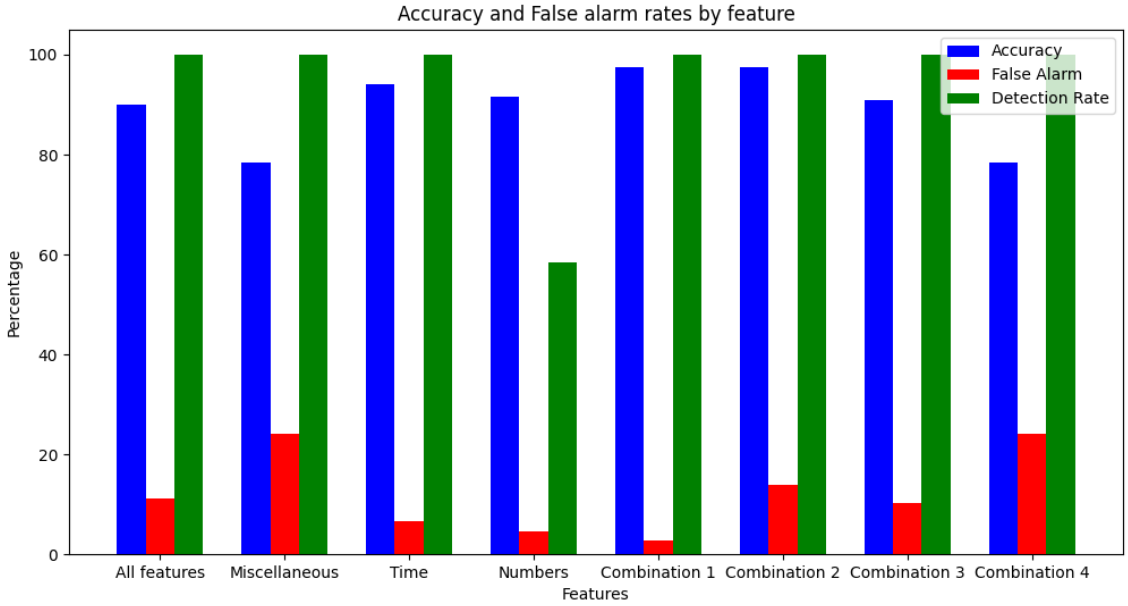

Figure 4: Accuracy and false alarm rate by features on eval 1

For eval2

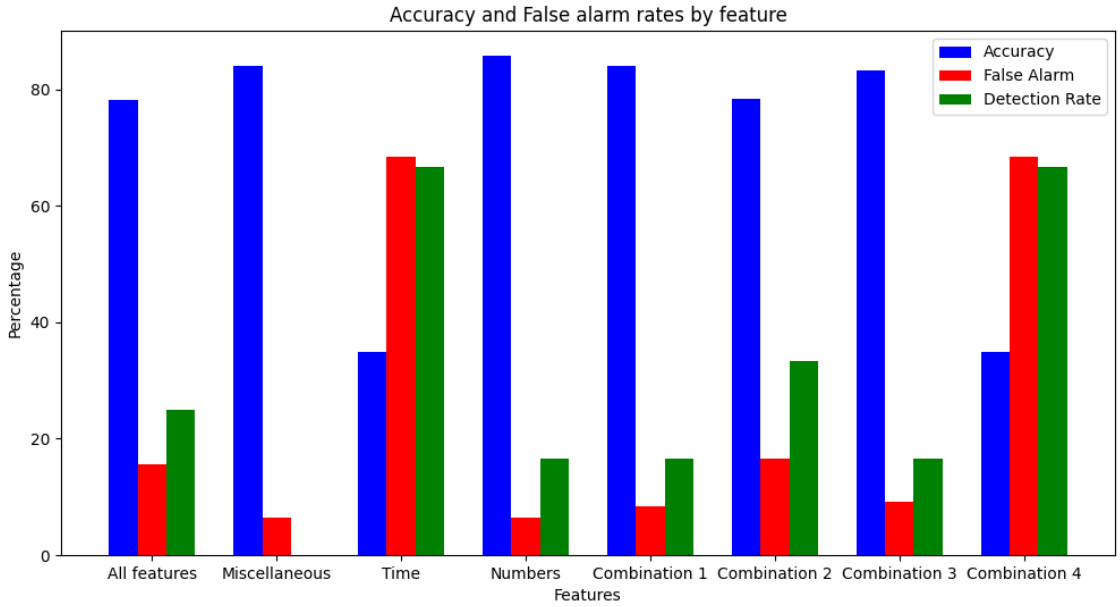

Figure 5: Accuracy and false alarm rate by features on eval 2

5.3.3 Comparison between the algorithms based on metrics

On devait choisir un combinaison pour faire notre diagram de comparaison entre les perfomances des algos entre eux, et on a choisi combi1 car ... (la plus logique selon nous, et la plus pertinente)

For eval1

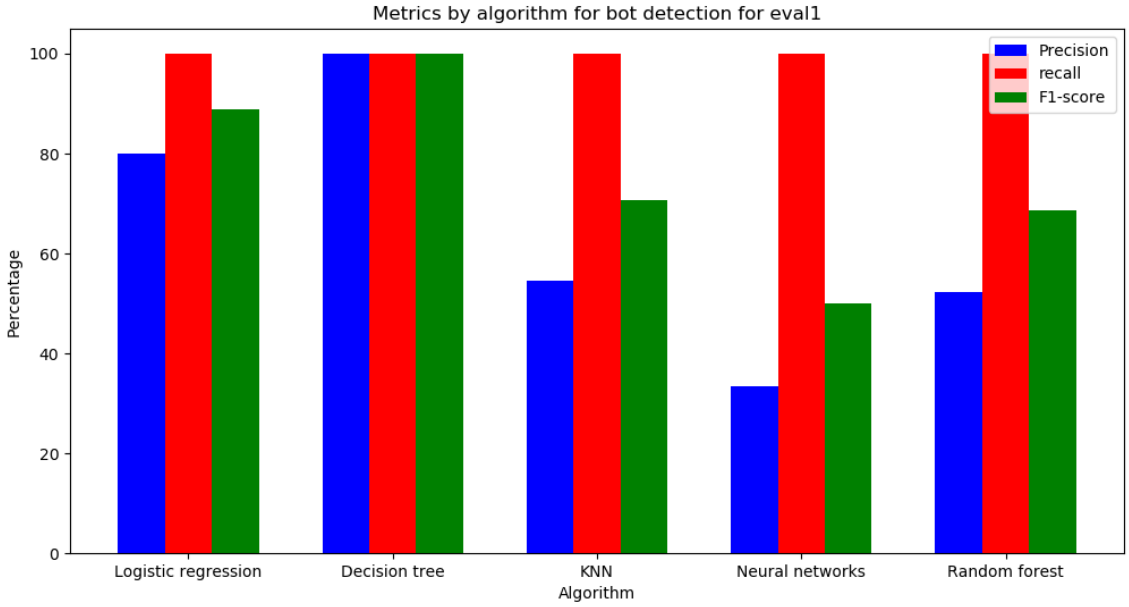

Figure 6: For bot

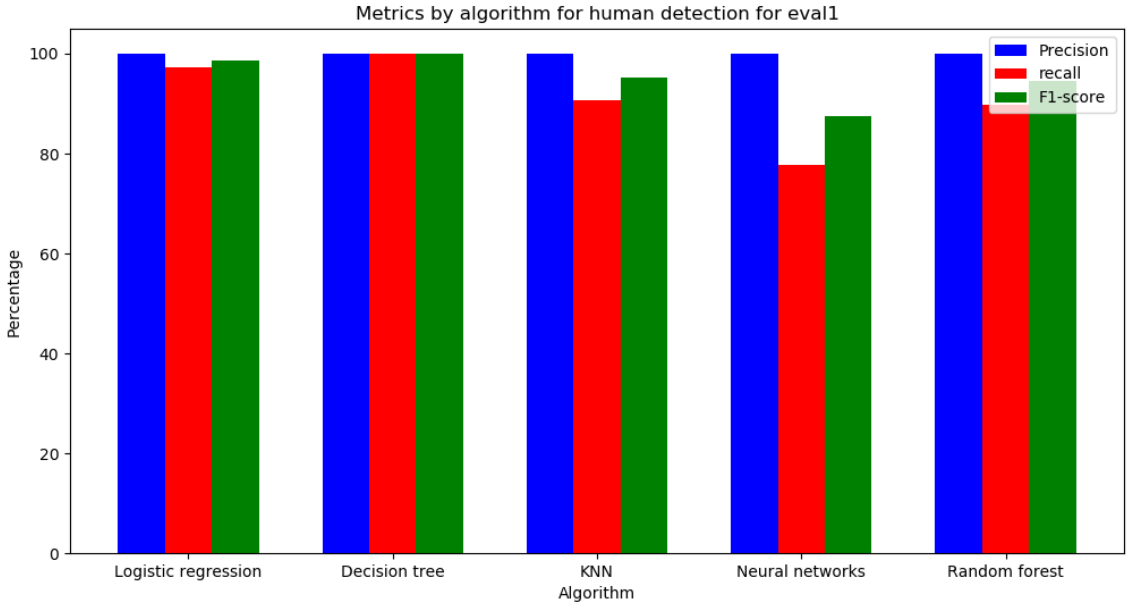

Figure 7: For human

For eval2

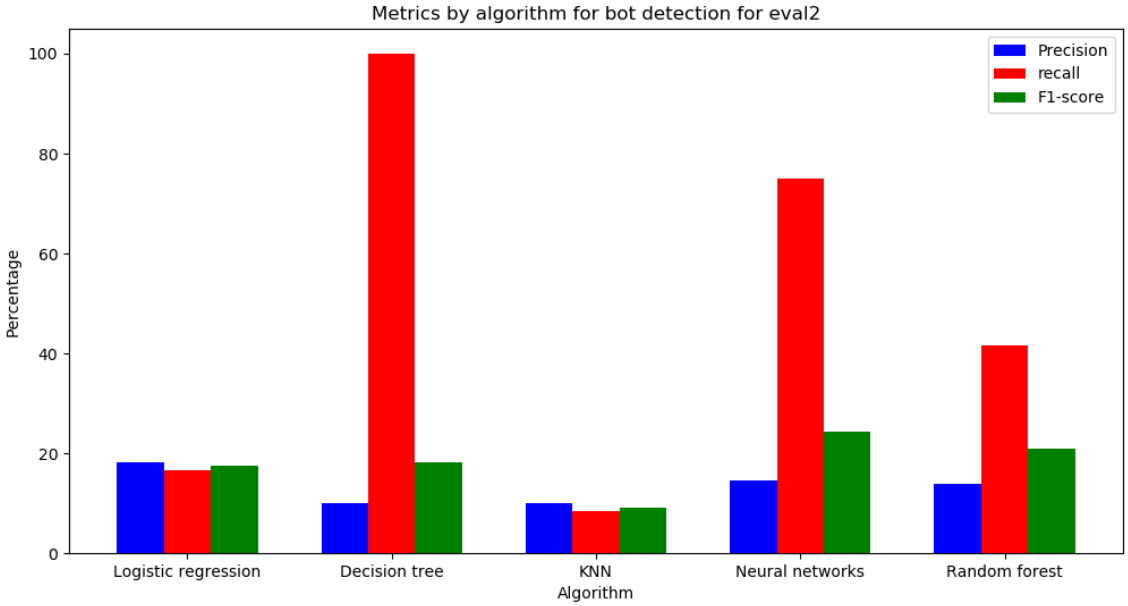

Figure 8: For bot

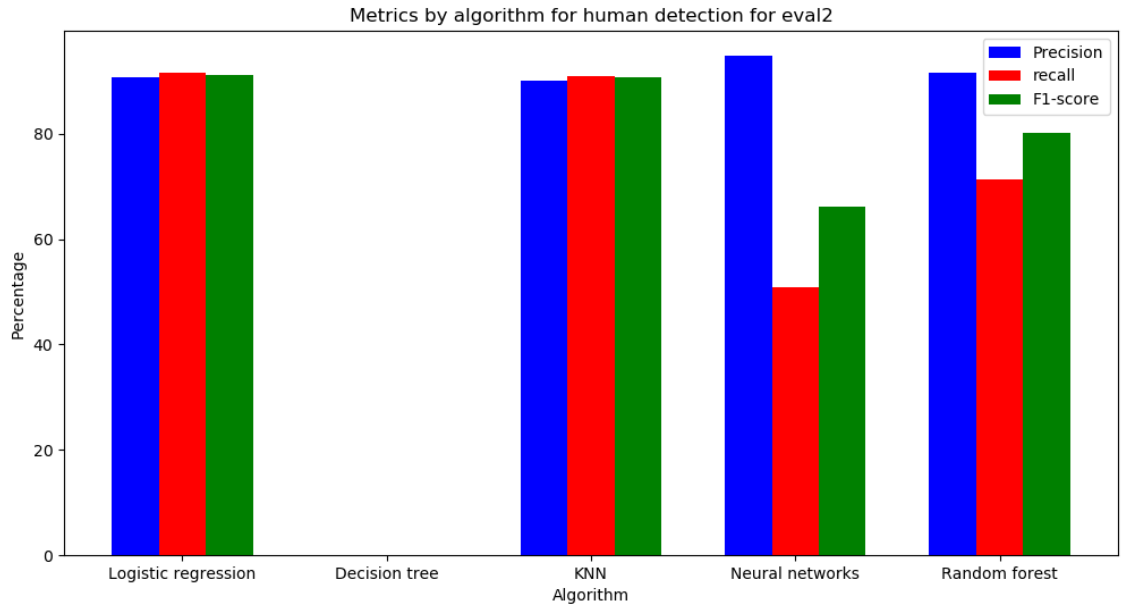

Figure 9: For human

TODO DISCUSS RESULTS HERE

5.3.4 Visualisation of the data using t-SNE

sur les différentes features

(pas de eval1 ou eval2, ici on est en preprocessing, ya pas d'algos qui interviennent)

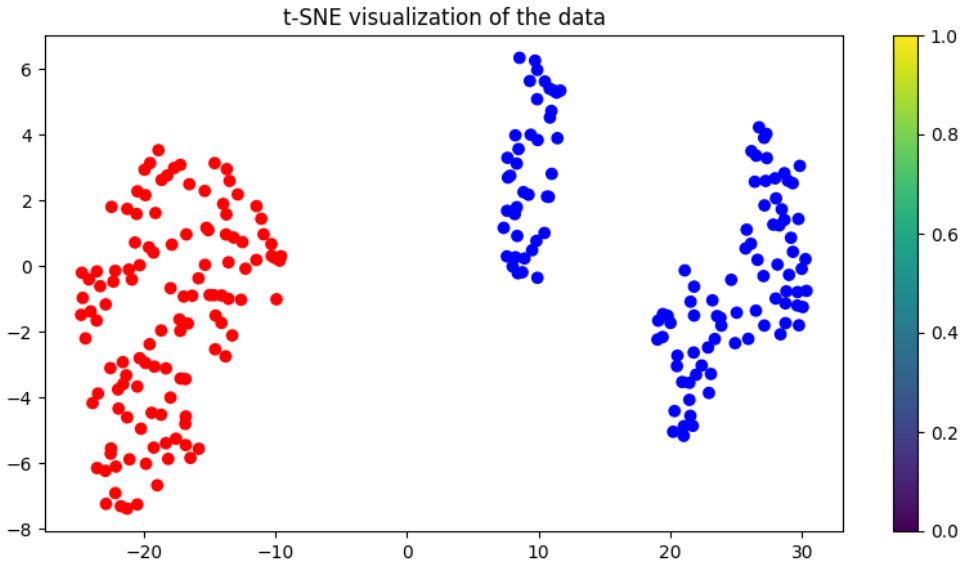

Figure 10: Visualization of training dataset for all features

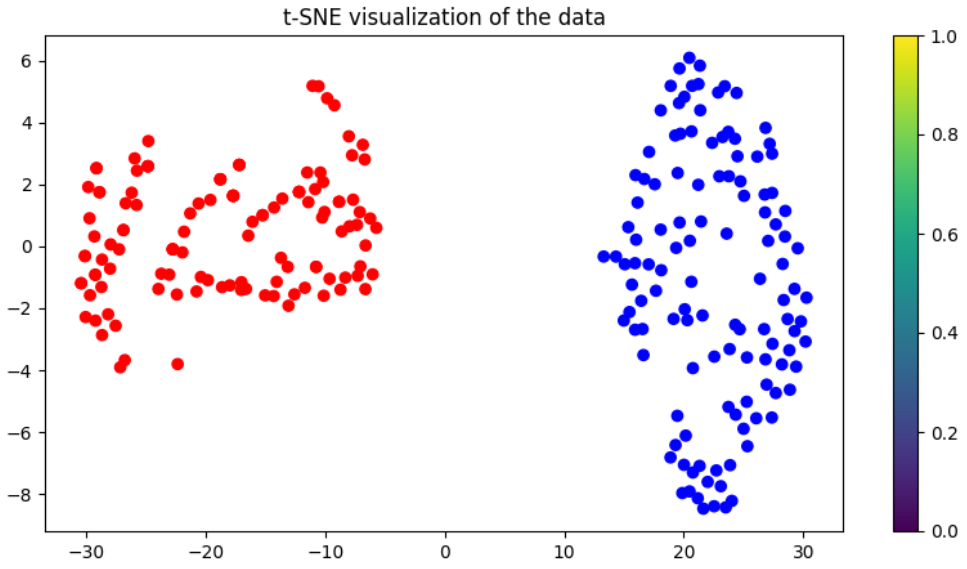

Figure 11: Visualization of training dataset for COMBI1

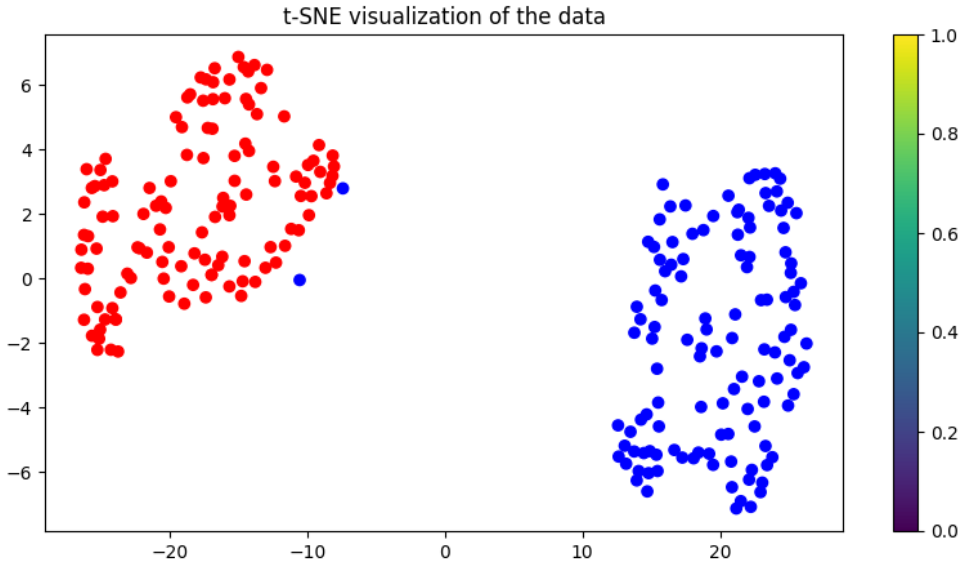

Figure 12: Visualization of training dataset for combinations of numbers

DISCUTER DES RESULTATS QU'ON A

pour numbers, les deux points bleus avec le groupe des rouges est normal car on a des faux négatifs donc ca démontre bien que ca marche

- F1-score =¿ aucune idée de ce que c'est mais ayoub a dit que c'est important

faire graphiques pour les autres configs c-a-d des changements et alternances de features qu'on a selectionné pour voir les changements sur l'accuracy du modele + voir avec appropriate baseline

expliquer les choix de features et comment elles influencent l'accuracy mais pas que ! (detection rate, false alarm rate, false negative and true negative)

5.4 Threshold for distinguishing human+bot from bot

Distinguishing between pure bot traffic and traffic generated by a combination of both bots and humans was a though task in the project. Our classifier is designed to perform binary classification, labeling data points as either bots or humans. However, it also provides a probability score indicating its confidence in each classification. To address the complexity of mixed behaviors, we introduced a post-processing step that involves comparing the absolute difference between the two probability values assigned by the classifier.

The threshold we set for this difference is at **0.5**. When the absolute difference between the probabilities for bot and human classifications exceeds this threshold, it signifies that the model is 'certain' about its decision. It suggests that the features' results are really

distinct, making it easy to definitively categorize the traffic as either purely bot or purely human.

Example for clarity

The classifier provides two probability values : the first indicating the likelihood that the host is a bot and the second indicating the likelihood that the host is human :

`unamur 25 : [0.71725976 0.28274024]`

In this case, the absolute difference between the two probabilities is smaller than the threshold of 0.5, indicating that the classifier is uncertain about its classification. This outcome suggests mixed behavior within the traffic data.

In contrast, in the following example :

`unamur 12 : human+bot : [0.99999 0.00001]`

Here, the absolute difference between the probabilities is significantly greater than the threshold, indicating a higher level of confidence in the classifier's decision. In this scenario, the model is more certain that the traffic is predominantly driven by a single behavior type.

5.5 Critics about our methodology

https://scikit-learn.org/stable/modules/feature_selection.html feature selection =¿ to select the best sets of features, we could have used SelectKBest, Model Accuracy: 1.0 Selected Feature Indices: [0 2 5 6 7] Selected Feature Names: ['average$_o f_r equest_l engt h_e ncoded$','$type_o f_r equests_q ueried_b y$

Those are the best sets of features for the **training**

but the problem is that it will overfit, it will give an excellent accuracy on the training datasets but on the evaluation datasets, the results were not that great

—-

improvement potentiel =¿ peut etre mettre ici qu'on fait pas de normalization sur les données + parler du tool que le frère d'ayoub nous avait présenté pour crée des fausses données

A TESTER, UTILISER 'SMOTE' =¿ intéressant pour augmenter le nombre de samples pour maybe éviter le base-rate fallacy

5.6 Issues faced during the project

- **Handling large volumes of data** : The datasets contained a massive volume of DNS traffic, making data preprocessing and feature selection a time-consuming process.
- **Parameter tuning** : The project involved an array of parameters that required fine-tuning. This included experimenting with various combinations of features, selecting appropriate machine learning algorithms and optimizing hyperparameters for these algorithms. Choosing the right balance between different parameters while maintaining acceptable performance metrics was a pretty hard process.

- **Feature selection** : We had a lot of features to choose from, each potentially contributing to the classification task. The challenge was to identify the most relevant features that could accurately differentiate bot behavior from human behavior.
- **Avoiding pitfalls** : The awareness of potential pitfalls in machine learning, such as sampling bias, data snooping and spurious correlations, added complexity to the project (but we recognize that it was necessary to have a robust classifier). It was essential to design and implement strategies to mitigate these pitfalls and ensure the robustness and validity of the machine learning model.
- **Evaluating model performance** : Determining the real effectiveness of the IDS model required careful consideration of performance metrics. Selecting the right evaluation criteria and ensuring that the model performs well in the evaluation datasets presented its own set of challenges.

6 Conclusion

During the project, we realised that developing an IDS based on machine learning is a really complex task. We tried to design and develop an IDS that not only achieves high accuracy but also ensures security, by relying on different metrics (TP, FP, TN, FN) in distinguishing bot traffic from human traffic. It has allowed us to develop a better understanding of the pitfalls, especially the base-rate fallacy and we tried to take measures to avoid them (whetever possible).

parler de bayesian etc aussi ici

Our key takeaways and reflections about this project include :

- **Complexity of ML IDS** : Creating an effective IDS using ML involves a lot of parameters, from feature selection and algorithm choice to hyperparameter tuning. Achieving a good balance between these parameters while maintaining performance metrics is pretty hard.
- **Pitfall awareness** : We gained a better understanding of the pitfalls associated with machine learning presented on this article[1], particularly in the context of IDS development.
- **Inherent data limitations** : By recognizing the constraints of the provided datasets, such as limited data volume and specific botnet behaviors, we were able to gain a better understanding of the results generated by the model. We addressed these limitations by adapting our analysis and modeling strategies.

In conclusion, this project has been both enjoyable and very challenging. We learnt a lot of complexities involved in machine-learning, it also enhanced our understanding of the pitfalls and we now have a better perspective on developing **secure** and **accurate** classifiers.

7 Bibliography

https://www.bibme.org/bibtex

References

[1] Daniel Arp et al. "Dos and Don'ts of Machine Learning in Computer Security". In: *CoRR* abs/2010.09470 (2020). arXiv: 2010.09470. URL: https://arxiv.org/abs/2010.09470.

[2] Stefan Axelsson. "The base-rate fallacy and the difficulty of intrusion detection". In: *ACM Transactions on Information and System Security* 3.3 (2000), pp. 186–205. DOI: 10.1145/357830.357849.

[3] Robert F Erbacher. "Base-rate fallacy redux and a deep dive review in cybersecurity". In: *arXiv preprint arXiv:2203.08801* (2022).

8 For us only

we are expected to provide visuals for the projects deadline : 8th of November. MAKE A requirements.txt at the end

dans le code, pour la list de feature, pas oublier de remettre avec COMBI1

vérifier que l'interface que le prof a demandé est l'interface correcte chez nous (avec les bons arguments et tout) FORMAT THE CODE faire un CTRL+F des TODOs du project et s'assurer qu'il ne reste plus rien faire une relecture complète du code et vérifier que tout est cohérent UN FOIS LE RAPPORT FINI, LE METTRE DANS LE GITHUB !!!!! A NE SURTOUT PAS OUBLIER