POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

IoT Forensics Made Easy Extracting Information from Amazon Echo's Network Traffic with Feature Sniffer

Tesi di Laurea Magistrale in
Computer Science Engineering -
Ingegneria Informatica

Author: **Marco Marini**

Student ID: 953281
Advisor: Prof. Alessandro Enrico Cesare Redondi
Co-advisors: Fabio Palmese
Academic Year: 2022-2023

Abstract

In the last few years, with the increasing use in everyday life of IoT devices, more attention has been posed to the security and privacy aspect. Ensuring and verifying the reliability of these devices is a key objective for IoT Forensics. Different papers have been published to demonstrate the possibility of extracting information from encrypted network traffic using Machine Learning techniques.
However, there are two main problems when using learning algorithms on those kind of data: first of all, a real, reliable ground truth is rarely available secondly, it is not immediate to find out and extract the useful features needed to feed the final model. For these reasons, the researchers of Politecnico di Milano have developed a tool named **Feature-Sniffer** (FS) that helps to extract relevant features, obtained by grouping packets into time windows, while sniffing the internet traffic directly from the access point.
The aim of this thesis is to test Feature Sniffer's capability to extract information out of the encrypted traffic produced by an Amazon Echo device (Alexa) while the user interacts with it performing common home assistant activities. We will show that this is possible (even though it depends from the task and with some limitations) and we will also train a model able to identify, with almost zero error, when an interaction with Alexa is happening. Having a tool like that makes building the training dataset way more easier than ever, opening the doors to a multitude of new research activities.

Keywords: Feature-Sniffer, IoT, IoT Forensics, Alexa, Machine Learning.

Abstract in lingua italiana

Negli ultimi anni, con la crescita esponenziale dell'utilizzo di dispositivi IoT nella vita di tutti i giorni, più attenzione è stata posta anche sull'aspetto della sicurezza e della privacy. Assicurare e verificare l'affidabilità di questi dispositivi è un requisito fondamentale per l'IoT Forensics. Sono stati pubblicati numerosi articoli che dimostrano la possibilità di estrarre informazioni utilizzando algoritmi di Machine Learning allenati sul traffico dati cifrato delle interazioni con questi dispositivi.

Ci sono però due problematiche principali quando si pratica Machine Learning su questo tipo di dati: in primis, quasi mai abbiamo un vero ground truth (un dataset pulito, privo di errori e rumore) e in secondo luogo è veramente difficile e macchinoso individuare ed estrarre quali dati possano essere utili per allenare tali algoritmi.

Per questo motivo i ricercatori del Politecnico hanno sviluppato uno strumento, chiamato **Feature-Sniffer** (FS), in grado di estrarre dei dati rilevanti provenienti dal traffico di rete prodotto dai dispositivi IoT durante il loro funzionamento, ottenuti raggruppando i pacchetti in finestre temporali ed estraendo dei valori statistici.

Lo scopo di questa tesi è quello di verificare se, utilizzando FS, sia effettivamente possibile estrarre informazioni dal traffico dati generato da delle interazioni con un dispositivo Amazon Echo (Alexa).

Vedremo come ciò non solo sia possibile (anche se con qualche limitazione, dipendentemente dal tipo di informazione che si vuole estrarre), ma alleneremo anche anche un algoritmo in grado di identificare, con errore quasi nullo, quando sta avvenendo un'interazione con Alexa.

Avere un tool come questo a disposizione permetterà di semplificare di molto il processo di creazione del training dataset d'ora in poi, spianando la strada per una moltitudine di nuove ricerche.

Parole chiave: Feature-Sniffer, IoT, IoT Forense, Alexa, Machine Learning.

Contents

1 | Introduction

1.1. Overview

There are well over 13 billion connected IoT devices around the globe and it's expected they will be 25.4 billion by 2030 [4].
And, if these numbers are not enough, to understand how this technology is so deep-rooted in our everyday life, just think of the fact that there are almost as many connected IoT devices as there are people worldwide.
What makes these devices so popular is the fact that they usually don't need any computational power, just collect data and send them to remote servers that will make the "hard work" for them. This kind of design allows scalability, computational speed, and most of all, affordability.
But, if from one side this is a strong point, on the other hand, we are lacking in terms of privacy and security. Our data and information are sent to remote servers and this data exchange is not always attack-proof. And, even if it would, the user almost always does not know how its data is managed, used, and preserved.
Seen the enormous quantity of private and sensible information IoT devices exchange every day, it is essential to test them and try to find vulnerabilities.
One of the main kinds of approaches that are being studied in these years relies on performing advanced machine learning techniques on sniffed encrypted network traffic between the device and the remote server.
Developing a wide knowledge of IoT devices and how to extract information from them is also the main objective of IoT Forensics, which is a branch of Digital forensics that has the goal of identifying and extracting digital information from devices belonging to the Internet of Things field, using a forensically sound and legally acceptable process [12].
For this reason, being able to extract relevant data out of encrypted traffic may reveal to be an incredibly powerful tool in the hands of a Forensic analyst.
In this thesis, we will explore this type of technique on the world's most used IoT device for the smart home: Amazon Echo.

1.2. Thesis Objective

It is already been proved that information can be extracted by performing machine learning techniques on encrypted network traffic. However, in this thesis we want to move a step forward in this direction. We want to make the whole process, from the data extraction part and ground truth creation up to the learning process, quick and simple.
To obtain this result, a tool, developed by the researchers of Politecnico di Milano, named **Feature Sniffer** (FS), will be used. Feature Sniffer allows to collect, in real time, features from a network traffic exchange. Nevertheless, to do this, we loose granularity of information and potentially introduce noise in the training dataset.
In the next chapters we will try to understand if Feature Sniffer could be effectively used to collect training features able to produce consistent results when used on learning models. We will also develop a machine learning model able to isolate the windows of traffic containing an interaction with the Amazon Echo device (Alexa). The combination of this last model applied after Feature Sniffer is going to introduce the possibility of having a clean and precise training dataset of interactions ready in almost zero time while sniffing the traffic.

1.3. Thesis Outline

This section is used to summarize the content of the thesis chapters, in order to ease the reader in understanding the key logical steps and focal points of this research work.

- **Chapter 1** introduces the thematic area of this work with its correlated issues, our purpose and some of the instruments we have used.
- **Chapter 2** illustrates the state of the art for machine learning applied to network traffic. Here we will see what kind of researches have been developed on this topic and with which results. This knowledge, will be helpful to understand the starting point and how our work could distinguish itself from the existing ones, bringing additional value to the scientific community.
- **Chapter 3** presents the steps followed to collect the data used to train the models along with some insights on our dataset composition. We will also introduce the logic behind a fundamental tool for this research: Feature Sniffer.
- **Chapter 4** is the core of the thesis, it illustrates the main problem we had to address and the various solutions we figured out to isolate the ground truth from our data. It will deeply go through the algorithm developed and how we have tested its output.
- **Chapter 5** shows all the machine learning steps performed on the training dataset, produced in the previous chapter, for all the test cases. In this chapter we will also discuss and reason about the results obtained and how or if they could be improved.
- **Chapter 6** finally summarizes what we have obtained giving also some hints on possible future developments.

2 | State Of The Art

In 2010 IoT devices were only the 9% of the total number of tech devices around the globe, right now, in 2023, with the 66% of sector dominance, the amazing spread of IoT technology is unquestionable.

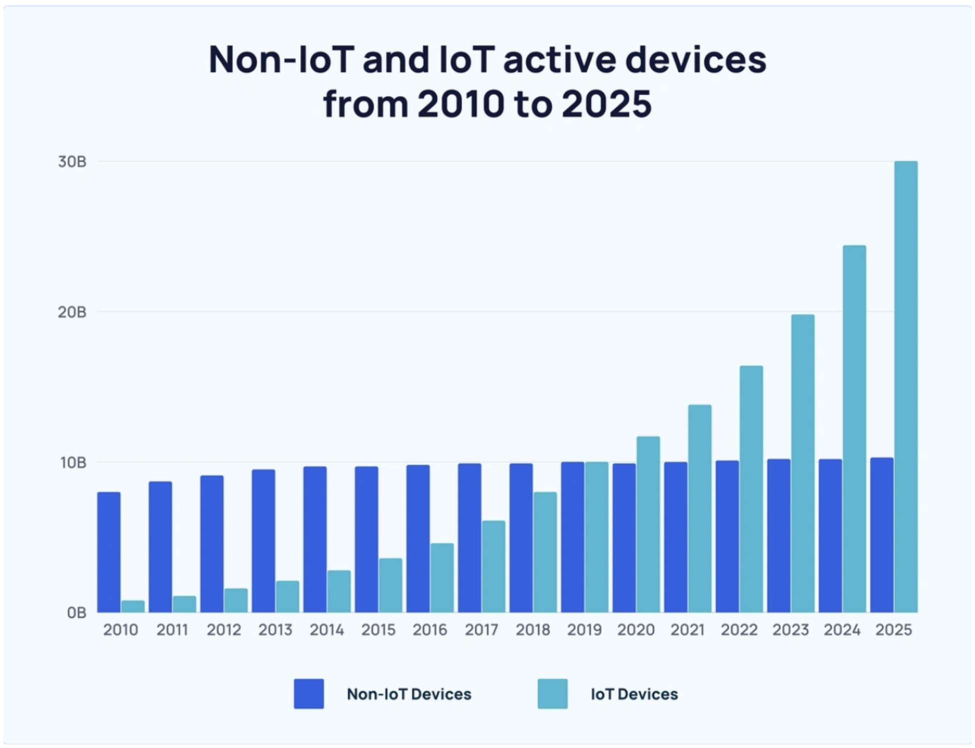


Figure 2.1: This figure shows the composition of technological devices divided in IoT and non-IoT devices with a forecast for 2025 [4]

The quantity of sensible and private data that goes through these devices is enormous. Payments, contacts, information about our interests and house settings are just some examples. Due to IoT design, these data just travel along the internet and can be sniffed by malicious agents. Of course, most of the time, the exchange of information between IoT devices and servers is encrypted; but are we really sure that encryption is a warranty of protection?

In the last few years has been in fact discovered that, with the right features, applying machine learning on encrypted data can bring amazing results.
Even the biggest companies lately are experimenting the use of machine learning on encrypted information in order to make their recommendation systems work on anonymized and protected data.
Going back to the IoT world, the problems addressed by researchers are the following:

1. If we supposed a malicious agent able to sniff the packets exchange between an IoT device and the server, would it be possible to extract meaningful information from that encrypted conversation?

2. If the answer to the previous question is yes, which kind of information could we extract and how?

3. Is there a way to prevent that kind of attack from happening?

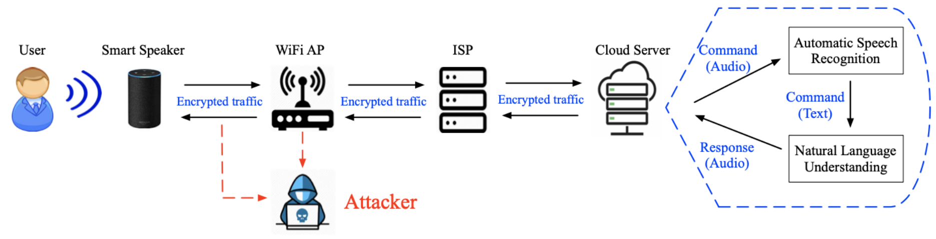


Figure 2.2: This figure exemplify the situation described above. Image taken from [13]

Whereas in this thesis we will work with the Amazon Echo's vocal assistant, during this chapter we are going to focus on researches made on the same or similar products.
All the works we will see have found a positive answer to question number 1 showing that it is actually possible to obtain information from an encrypted traffic exchange and that this system is not attack-proof. Now, the focal point becomes discovering what kind of data could be extracted and how to use it to perform an effective attack.
In particular, [13] and [6], in their papers have been shown that a neural network, built on top of the network traffic exchange of an Amazon Echo device, can score up to a 92% of accuracy while trying to infer the exact question asked to Alexa. Always in [13], same results are obtained also on Google Home. The authors of [5] investigated the possibility to guess, using a classification model, the person who is asking the question to Alexa. The results, with an 80% accuracy on binary classification and a 30% on classification between 12 speakers, are not astonishing, but for sure prove the possibility to infer information not only on the object of the conversation but also on who is speaking.
Finally, [1] studied the possibility of classifying if the question towards Alexa is generated

by an human voice or by a text-to-speech (tts) bot. Reaching an AUC value of around 0.95 for Random Forest and Gradient Boosting algorithms.
Speaking about possible mitigation of this attack, [13] and [6] have proposed to use adaptive padding. With adaptive padding basically "dummy" packets (packets carrying only a portion of the interaction filled with other useless data) are implemented in order to make all the packets the same size, so that the encrypted data results equal from the outside, bringing also the latency to zero. This approach helps in preventing these types of machine learning techniques from being effective, but it has the drawback of slowing down the whole communication between the device and the server, penalizing the customer experience. Moreover, transmitting useless data has a non-trivial energy cost, especially in battery-operated IoT devices. These are probably the reasons why big companies are not yet adopting this strategy.

All the research works we have seen have proven that machine learning can produce great results when applied on encrypted data traffic, because we are exploiting informative features such as packet lengths, inter-arrival times, number of packets sent, and so on. Nevertheless, collecting those features in a fast and easy way is difficult for two main reasons: first, great and clean datasets of network traffic captures do not exist, all the researchers mentioned above had to build their own dataset. To build a dataset like that, it is necessary to generate interactions with the home assistant, and to generate an interaction it is needed to have a voice which is asking questions. Since finding a modest number of human voices is quite hard, many research works (in particular [13], [6]) used artificial voices. This helps in obtaining really large training sets but, on the other hand, may produce different results with respect to a real-life situation, because a robot will always ask the same question with the same voice tone and inflection, while, a human voice may introduce noise leading to a less precise outcome.
The second reason that makes the whole process of feature extraction tough, is the number of steps that have to be performed before obtaining the data that will feed the various machine learning models. First of all, the traffic exchange has to be recorded through some software, which will produce files (usually pcap) containing all the relevant information of the network traffic. Then, it is necessary to write some code in order to compute the features of interest out of these files (some examples could be the number of packets sent or the inter arrival time). Finally, has to be isolated only the data recorded when an interaction with Alexa was happening, but, since packet payloads are encrypted, it could be really difficult to understand whether a packet exchange produced by the device is due to a real interaction with an user or to a standard protocol communication with the server.

For the moment, let's leave aside this last topic (which will be addressed further on) and suppose that our dataset is perfect and the model too. We can assume that one of the objectives might be to eavesdrop the traffic from the victim's router, in order to monitor when and which Alexa commands are launched. The main problem here, as said before, is manipulating and converting the data sniffed so that could be used as input for our trained model, applying the steps described above inevitably causes a significant delay between what the user is doing and when we will know what he did, making the procedure way less effective.
This is an open problem issued in the most cases where machine learning is applied on an encrypted traffic exchange.
Anyway, it is important to underline that the possible use cases do not necessarily relate to a malicious scenario. In fact, there are several applications of ML used on network exchange besides from the hacking and spying sphere.
For example [9], [10], [7] and [11] all studied different ways of predicting the future traffic flow by studying the live packets exchange in order to optimize the resource allocation server side to reduce the possibilities of network congestion.
The different fields of application of this technique are many and range from optimization problems up to the IoT Forensic world. Anyway, the great part of the use cases require a trained algorithm able to produce a quick and fast prediction. For this reason, the authors of [8] have recently developed a tool named Feature-Sniffer (for simplicity we will call it FS from now on). The aim of the tool is to collect in real-time the desired features directly from the access point while recording the live traffic. By doing this, FS outputs a file (usually `csv`) containing all the chosen features directly computed without the need of using a heavy `pcap` file. Having access to a tool like FS is an extra boost in an IoT Forensic analysis because it speeds up the feature collection phase opening the doors to an almost instant prediction on the traffic recorded. Unfortunately, FS introduces some limitations by design (that we will study further on in a dedicated section). The aim of this thesis will be trying to understand if FS could be effectively used to ease the process of information extraction from a recorded interaction with Alexa, we will also try to understand the limits of this tool and, instead, in which tasks it can bring great results. Furthermore, will be analyzed how to detect, with extremely high precision, when an interaction with the Amazon Echo device is happening. Combining FS ability to extract features in real-time with a model able to detect, with extreme precision, the traffic windows where interactions are happening, will allow to obtain a clean and precise dataset of interactions in almost zero time. How to build the training set and collect clean features won't be a problem anymore, moving the focus only on which information we can extract from those data.

3 | Data Collection

In this chapter, will be shown what kind of data we needed for the experiment, how they have been collected and created the final dataset. In addition, we will go through the tools and the hardware used with a dedicated section to describe the main logic behind Feature Sniffer.

3.1. Collecting The Voices

To test FS we had to build a training dataset straight from zero, so let's have a look at how we built it and why.
We wanted to check if it would be possible to infer from the traffic exchange the following pieces of information:

- The presence of an interaction with Alexa.
- The genre of the speaker (bot or human)
- The language of who is talking (Italian or English)

Since the variety of classification tasks, the dataset needed to be almost equally distributed between robots and humans, English and Italian, males and females. We decided to give more relevance to the Italian sample because, as the voices were mainly of Italian people, we thought that speaking another language could have brought to an alteration of the voice's fluency causing the data to be less natural. In addition, collecting vocal recordings in a different language revealed to be a quite hard feat.
To have the voices always available we decided first, to record them, in order to be able to reproduce them as many times as we wanted in a successive phase. First of all, we wrote down a collection of 20 questions in Italian and 20 questions in English. Different questions have been used for the two languages because, in the mid of the work, we were able to recover some already existing voice recordings from Arensi's work [1] so, when we collected English voices, we decided to reuse its questions to have a larger dataset. Anyway, since Alexa gives different answers depending on the language we didn't considered having same questions in both the languages a crucial requirement.

Questions Recorded

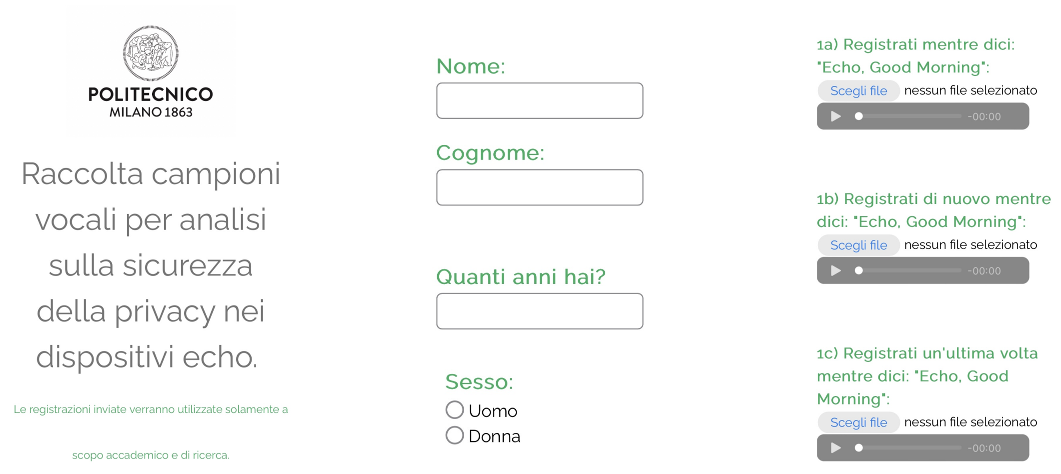
	Italian	English
Q1	Alexa, come stai?	Echo, Good Morning
Q2	Alexa, quando è stata distrutta Pompei?	Echo, Help.
Q3	Alexa, quanti anni ha la regina Elisabetta?	Echo, How hot is the sun?
Q4	Alexa, chi ha vinto Sanremo?	Echo, How many days are in September?
Q5	Alexa, quanti anni ha Gianni Morandi?	Echo, How many days in a year?
Q6	Alexa, cosa sta succedendo in Ucraina?	Echo, How many days until Christmas?
Q7	Alexa, quanto manca all'estate?	Echo, How many seconds are in a year?
Q8	Alexa, quando è Pasqua?	Echo, How much does an elephant weigh?
Q9	Alexa quando è la festa della mamma?	Echo, How old are you?
Q10	Alexa, quando gioca la Juve?	Echo, How old is Serena Williams?
Q11	Alexa, che tempo fa?	Echo, How tall is the Empire State Building?
Q12	Alexa, autodistruzione!	Echo, Pick a number?
Q13	Alexa, quando morirò?	Echo, Surprise me.
Q14	Alexa, sei innamorata?	Echo, Talk like a pirate.
Q15	Alexa, come fa il cane?	Echo, Tell me a fun fact
Q16	Alexa, batti il 5!	Echo, Tell me a joke
Q17	Alexa grazie.	Echo, Tell me a Star Wars joke.
Q18	Alexa, qual è il tuo cibo preferito?	Echo, Tell me some good news.
Q19	Alexa, quanto sei alta?	Echo, Tell me something weird.
Q20	Alexa, mi presti dei soldi?	Echo, Translate good morning to Spanish.

Table 3.1: List of all the questions recorded in the two languages

In order to ease the collection of the voice samples, we developed with HTML, Flask and Python, a simple web form using pythonanywhere[1]. This form could be opened from both mobile phones and personal computers and contained the list of the questions with a button to start a vocal recording and also the possibility to listen again the audios recorded. On the back-end, the recordings were sent to an online archive where each audio was saved as "**question_number** + **name_of_the_person** + **age** + **unique_id** + **.wav**".
For each question three different recordings were requested. In this way, even if one or

[1]To know more about pythonanywhere and how it works consult this link: pythonanywhere.

two recordings revealed to be unusable (for example a misspelling of the word "Alexa" or an audio cut), we had an higher probability to have at least one working audio sample.


POLITECNICO MILANO 1863

Raccolta campioni vocali per analisi sulla sicurezza della privacy nei dispositivi echo.

Le registrazioni inviate verranno utilizzate solamente a scopo accademico e di ricerca.

Nome:

Cognome:

Quanti anni hai?

Sesso:
○ Uomo
○ Donna

1a) Registrati mentre dici: "Echo, Good Morning":
Scegli file nessun file selezionato
-00:00

1b) Registrati di nuovo mentre dici: "Echo, Good Morning":
Scegli file nessun file selezionato
-00:00

1c) Registrati un'ultima volta mentre dici: "Echo, Good Morning":
Scegli file nessun file selezionato
-00:00

(a) Introductory part. (b) Registry part. (c) Recording part.

Figure 3.1: Insight of the website's content.

All the questions in the form were set as mandatory so each user had to send a total of 60 recordings (3 multiplied by 20 questions). Thanks to the website we have been able to collect the voices of 22 people for the Italian questions and 9 for the English ones (as already stated before, it has been a bit tough to collect the English sample). This brought us a total of 1860 recordings. Considering that the work in [1] obtained great results with a total of 400 samples, we were satisfied with our human-dataset dimension.
For what concerns robot voices, we used three different text-to-speech (tts) APIs: Google-tts, pyttsx3 (based on python) and Amazon Polly (see [2]). Google and Python tts produce more mechanical voices, easy to identify as robotic, while Amazon Polly can produce either less realistic voices or super realistic voices using the flag "**neural**". In the Italian samples we used both the flags while, in the English one, we used only neural voices. Polly can also produce a wide range of different voices (Men, Women, kids) and English accents (American, British, Australian and so on), while python and google APIs offer only a man and woman voice. Since Polly offers many English different voices, but way less Italian voices we used Google and Python tts to increase the Italian sample of robotic voices. We also noticed that, using the default settings, questions asked by google and python tts are not understood by the Amazon Echo, anyway, slowing down the voice speed, the device correctly replied to all the questions.
To sum up, in the end we got 15 different English artificial voices and 8 Italians, for a total of 1380 recordings.

The following tables contain a final recap of the voices collected:

Human Voices Dataset

	Italian	English	Total
MEN	14	5	19
WOMEN	8	4	12
TOTAL	22	9	31
RECORDINGS	1320	540	1860

Table 3.2: Human dataset sample distribution.

Robot Voices Dataset

	Italian	English	Total
GoogleTTS	1	0	1
PythonTTS	2	0	2
Amazon Polly	5	15	20
TOTAL	8	15	23
RECORDINGS	480	900	1380

Table 3.3: Robot dataset sample distribution.

Final Dataset

	Number of Recordings
Human	1860
Robot	1380
Italian	1800
English	1440
TOTAL	3240

Table 3.4: Final dataset dimensions.

Amazon Polly Voices

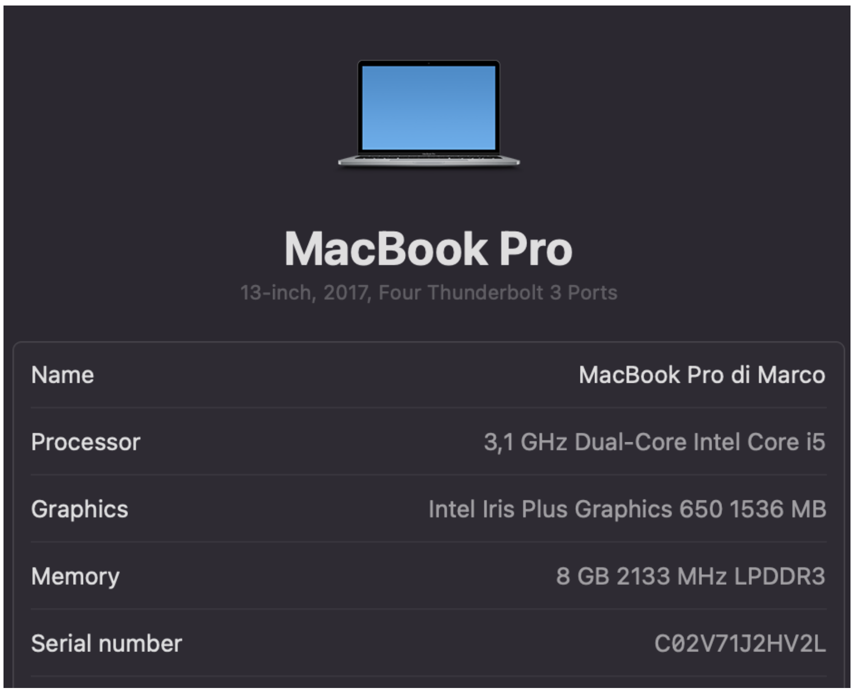
	Normal	Neural
Italian	Bianca Carla Giorgio	Bianca
English	-	Amy Aria Ayanda Brian Emma Ivy (kid) Joanna Joey Justin (kid) Kendra Kevin (kid) Kimberly Matthew Olivia Salli

Table 3.5: Amazon Polly voices used.

3.2. Dataset Construction

After the collection phase we examined all the recordings collected to see first, if they were correct, secondly, if Alexa was able to answer to them. During this step we removed all the recordings which were not triggering the Echo (around the 1-2% of the dataset). However, since we asked for three audios per question, we always had at least one recording which worked fine. In case of need, we duplicated the correct recordings in order to substitute the wrong ones. This meant that some audio samples were used two or three times when building the final dataset, anyway we verified that even the same recording can produce different traffic exchanges so, using the same samples more times doesn't necessarily mean having duplicates in the final dataset. Regarding robot audios, to insert some differences between repetitions of the same question, we have varied the speed at

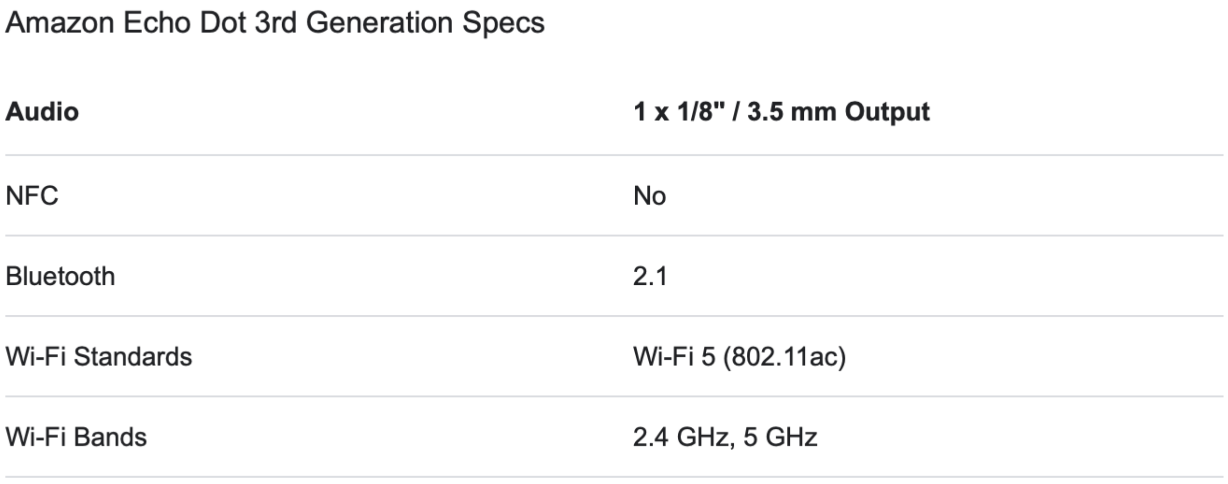
which the robot was talking.
After this intermediate skimming, we were ready to reproduce our voices and record the traffic exchange between the Echo and the server. To do so, we set up an Access Point with FS installed and running, then we connected our Amazon Echo to the WiFi network. To generate the interactions using pre-recorded voices, we exploited a Python script in a laptop to reproduce one audio sample every minute using the laptop speakers. The one-minute pause is needed for Alexa to elaborate and reply to the questions, since some of them produced really short replies, while others needed up to 40 seconds.
Of course, we changed the language settings of the Echo when switching from Italian to English.
Finally, we created, with FS, a single file for each human/bot, in order to be able to organize better the next data-processing phase; we ended up with 54 **csv** files, containing the traffic features extracted by Feature-Sniffer. We also used Wireshark to record the traffic exchange, for the sake of comparing the size of files produced using pcaps with respect to FS output.
To give some specifications about the hardware, we used a MacBook Pro with macOS Ventura installed and an Amazon Echo Dot of third generation, while the Access Point used for the feature collection is the Linksys WRT3200ACM.


MacBook Pro
13-inch, 2017, Four Thunderbolt 3 Ports

Name	MacBook Pro di Marco
Processor	3,1 GHz Dual-Core Intel Core i5
Graphics	Intel Iris Plus Graphics 650 1536 MB
Memory	8 GB 2133 MHz LPDDR3
Serial number	C02V71J2HV2L

Figure 3.2: Specifications of the laptop used.

(a)



Amazon Echo Dot 3rd Generation Specs

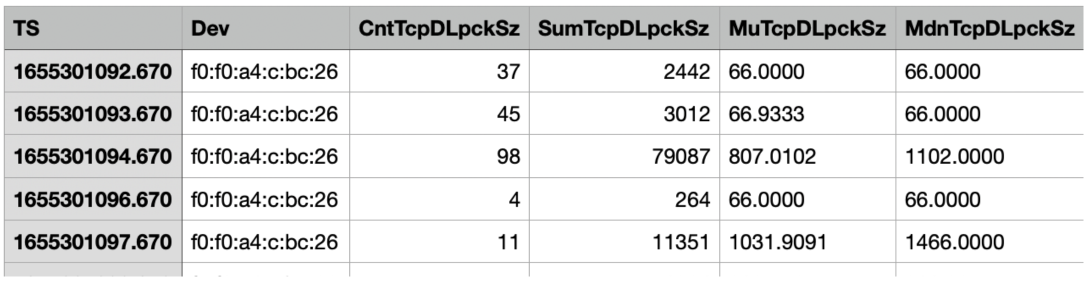
Audio	**1 x 1/8" / 3.5 mm Output**
NFC	No
Bluetooth	2.1
Wi-Fi Standards	Wi-Fi 5 (802.11ac)
Wi-Fi Bands	2.4 GHz, 5 GHz

(b)

Figure 3.3: Specifications of the Echo Dot used.

3.3. Feature-Sniffer Overview

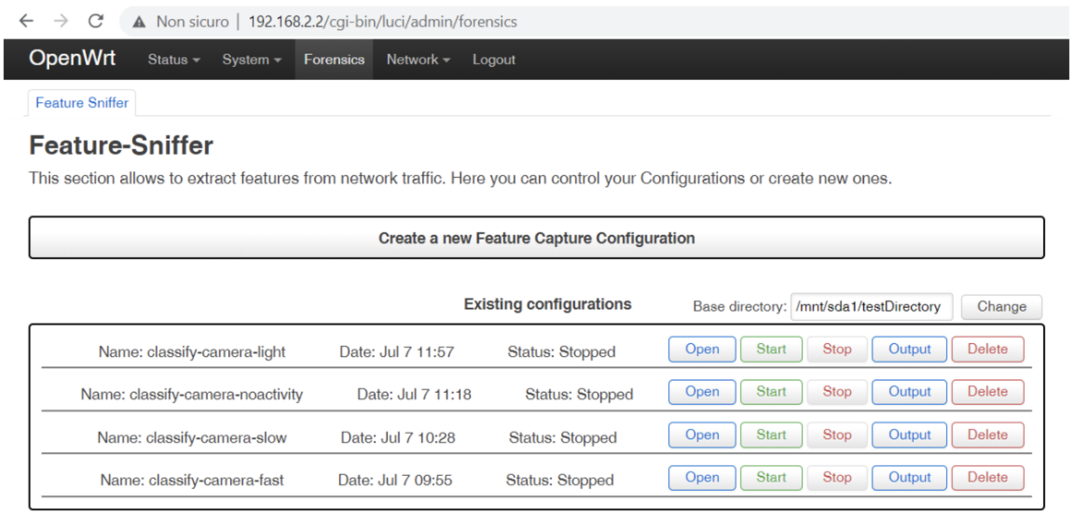
Feature-Sniffer, as better explained in its presentation paper [8], is an add-on for OpenWrt-based access points that allows to easily perform online traffic feature extraction, avoiding to store large pcap files. We are basically speaking of a software with an ad-hoc GUI, that can be installed on access points to have a live extraction of network traffic features. FS produces csv files containing, in the columns, the features chosen and in the rows the time windows. To be more precise, before starting the collection process, it is possible to choose which features to extract, which time window to consider and, optionally, if we want to filter this piece of information only for one or more devices identified by the MAC address. The time window is necessary because it tells FS how to compute the value of the features: for example, if we are using a time window of 1 second, the feature `"average_packet_length"` will compute the average length of the traffic packets every one second and if we record the traffic for 10 seconds, we will obtain 10 rows in our final csv which will contain 10 values for each feature. Figure 3.4 shows a piece of the csv produced by FS with a window of 1 second. The first row contains the name of some features (number of TCP packets in down-link, the sum of their size, their average and so on). While `TS`, shows the timestamp at which it is started the recording of each window.



TS	Dev	CntTcpDLpckSz	SumTcpDLpckSz	MuTcpDLpckSz	MdnTcpDLpckSz
1655301092.670	f0:f0:a4:c:bc:26	37	2442	66.0000	66.0000
1655301093.670	f0:f0:a4:c:bc:26	45	3012	66.9333	66.0000
1655301094.670	f0:f0:a4:c:bc:26	98	79087	807.0102	1102.0000
1655301096.670	f0:f0:a4:c:bc:26	4	264	66.0000	66.0000
1655301097.670	f0:f0:a4:c:bc:26	11	11351	1031.9091	1466.0000

Figure 3.4: A piece of the csv produced by FS.

According to the window's length, we are basically adding more details (small window) to our dataset or more generalization (larger window), but we are also producing heavier (small window) or lighter (large window) files.
For our experiment, we used two different time windows: 0.5 seconds and 1 second. We selected all the available features, since it's always better to evaluate in a second phase which are the relevant ones, we also filtered the data for the Amazon Echo device only, using its MAC address.
It is also worth mentioning that, if all the features for a time window have value zero, that row is not produced by Feature-Sniffer in the final csv.



Feature-Sniffer

This section allows to extract features from network traffic. Here you can control your Configurations or create new ones.

Create a new Feature Capture Configuration

Existing configurations

Base directory: /mnt/sda1/testDirectory Change

Name: classify-camera-light	Date: Jul 7 11:57	Status: Stopped	Open Start Stop Output Delete
Name: classify-camera-noactivity	Date: Jul 7 11:18	Status: Stopped	Open Start Stop Output Delete
Name: classify-camera-slow	Date: Jul 7 10:28	Status: Stopped	Open Start Stop Output Delete
Name: classify-camera-fast	Date: Jul 7 09:55	Status: Stopped	Open Start Stop Output Delete

Figure 3.5: Feature Sniffer GUI homepage. Image taken from [8].

Using FS, has been produced a total of 150MB of csv files while, the same traffic analysis obtained with Wireshark brought us to have more than 1.2GB of pcaps. The size reduction of files produced might be game changing when working on very large datasets, especially, when performing IoT forensics, is not unusual that might be necessary hours or days of traffic sniffing and this can produce even Terabytes of data due to pcap files. Anyway, the main advantage of FS is for sure the possibility to have ready-to-use features computed on the spot. No need to build ad-hoc scripts and wait their computational time, all we need is produced by Feature-Sniffer and at our fingertips. This characteristic of FS particularly shines when we already have the model trained and we want to use it: if we had to pass through pcap files, produce the features and fit the model this might make us loose the ability to have a real-time prediction. Instead, thanks to FS, we can skip the first two steps feeding our model with data computed on-the-fly.
On the other side, FS brings some limitations: even though the number of features available (see figure 3.6) is large, it is still finite and does not give us room to build our own features. In addition, the time window, if not adequately selected may add noise to the data and lead to worse results.
It will be the main objective of this thesis to further investigate the limits and the advantages of FS, in order to understand for which tasks it shows to be a fundamental tool and when it might be necessary to explore other roads.

Feature	#	Σ	μ	$\mu_{1/2}$	M	σ^2	σ	K
TCP DL Packet Len	✓	✓	✓	✓	✓	✓	✓	✓
TCP UL Packet Len	✓	✓	✓	✓	✓	✓	✓	✓
TCP Packet Len	✓	✓	✓	✓	✓	✓	✓	✓
UDP DL Packet Len	✓	✓	✓	✓	✓	✓	✓	✓
UDP UL Packet Len	✓	✓	✓	✓	✓	✓	✓	✓
UDP Packet Len	✓	✓	✓	✓	✓	✓	✓	✓
DL Packet Len	✓	✓	✓	✓	✓	✓	✓	✓
UL Packet Len	✓	✓	✓	✓	✓	✓	✓	✓
Packet Len	✓	✓	✓	✓	✓	✓	✓	✓
TCP DL Payload Len		✓	✓	✓	✓	✓	✓	✓
TCP UL Payload Len		✓	✓	✓	✓	✓	✓	✓
TCP Payload Len		✓	✓	✓	✓	✓	✓	✓
UDP DL Payload Len		✓	✓	✓	✓	✓	✓	✓
UDP UL Payload Len		✓	✓	✓	✓	✓	✓	✓
UDP Payload Len		✓	✓	✓	✓	✓	✓	✓
DL Payload Len		✓	✓	✓	✓	✓	✓	✓
UL Payload Len		✓	✓	✓	✓	✓	✓	✓
Payload Len		✓	✓	✓	✓	✓	✓	✓
TCP DL Inter. Time			✓		✓	✓	✓	✓
TCP UL Inter. Time			✓		✓	✓	✓	✓
TCP Inter. Time			✓		✓	✓	✓	✓
UDP DL Inter. Time			✓		✓	✓	✓	✓
UDP UL Inter. Time			✓		✓	✓	✓	✓
UDP Inter. Time			✓		✓	✓	✓	✓
DL Inter. Time			✓		✓	✓	✓	✓
UL Inter. Time			✓		✓	✓	✓	✓
Inter. Time			✓		✓	✓	✓	✓

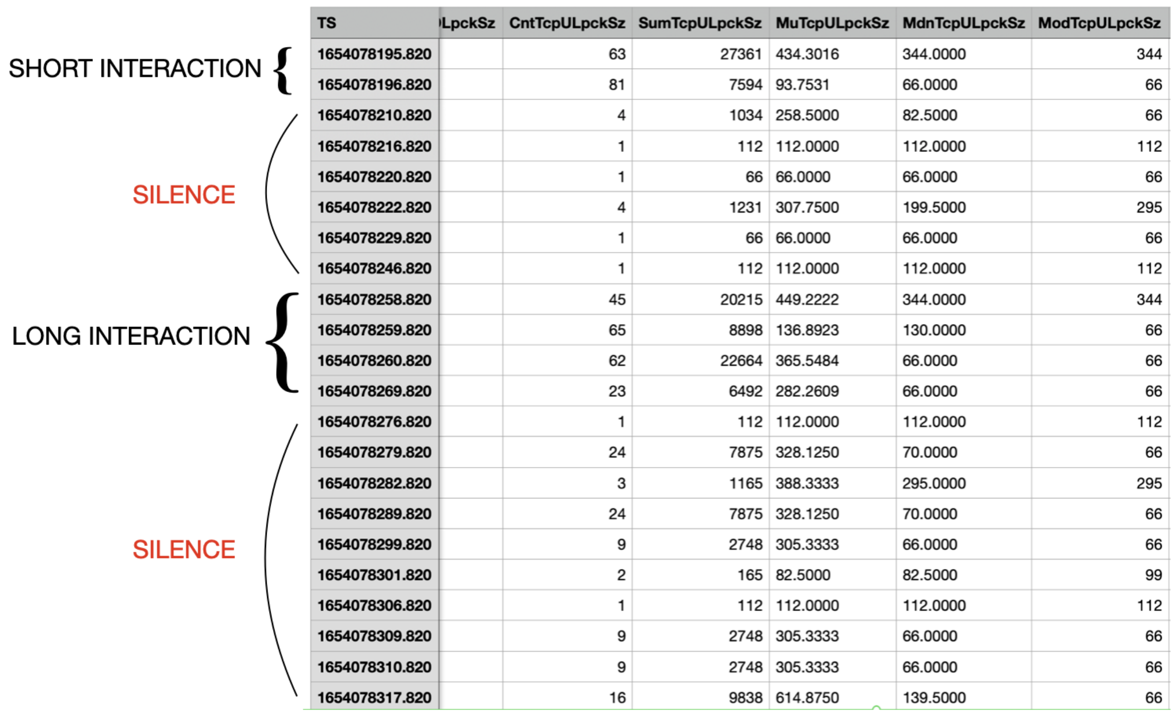
Table 3.6: List of Available Features in Feature-Sniffer

4 | Data Preprocessing

In this chapter, we will reach the heart of the thesis, showing how we managed to extract a ground truth of Alexa interactions ready to be used. We will go through the logic behind the algorithm, how we tested it, and the different challenges we had to face in the whole process.

4.1. The Ground Truth Problem

After the data collection, we end up with a set of csv files containing the needed data. However, such files also contained several potentially useless pieces of information that could be filtered out.
Remark that our purpose is to classify interactions of the users with Alexa and discover what can we infer from these interactions, only by looking at their encrypted traffic. To do so, we needed only the traffic data related to when the interaction is happening. Instead, our dataset was composed of a csv file for each user with each csv containing one interaction with Alexa, a one-minute pause of silence (delay added to let alexa complete the requested task), the next interaction and so on again. So, we actually had a large number of useless windows between two interactions. One may think that when there aren't questions asked to Alexa no network traffic is produced, but this is not true at all. There are many reasons why Alexa might want to reach its servers even though no question is asked, for example to install a software update, to send a keep alive or some reports regarding the user's activity, we can't know. FS removes by default the time windows where all the features computed have value equal to zero. Nevertheless, in our case this happened rarely, even though Alexa was not triggered, we observed that many windows were produced anyway (Figure 4.1 helps in understanding the concept).


SHORT INTERACTION
SILENCE
LONG INTERACTION
SILENCE

TS	LpckSz	CntTcpULpckSz	SumTcpULpckSz	MuTcpULpckSz	MdnTcpULpckSz	ModTcpULpckSz
1654078195.820		63	27361	434.3016	344.0000	344
1654078196.820		81	7594	93.7531	66.0000	66
1654078210.820		4	1034	258.5000	82.5000	66
1654078216.820		1	112	112.0000	112.0000	112
1654078220.820		1	66	66.0000	66.0000	66
1654078222.820		4	1231	307.7500	199.5000	295
1654078229.820		1	66	66.0000	66.0000	66
1654078246.820		1	112	112.0000	112.0000	112
1654078258.820		45	20215	449.2222	344.0000	344
1654078259.820		65	8898	136.8923	130.0000	66
1654078260.820		62	22664	365.5484	66.0000	66
1654078269.820		23	6492	282.2609	66.0000	66
1654078276.820		1	112	112.0000	112.0000	112
1654078279.820		24	7875	328.1250	70.0000	66
1654078282.820		3	1165	388.3333	295.0000	295
1654078289.820		24	7875	328.1250	70.0000	66
1654078299.820		9	2748	305.3333	66.0000	66
1654078301.820		2	165	82.5000	82.5000	99
1654078306.820		1	112	112.0000	112.0000	112
1654078309.820		9	2748	305.3333	66.0000	66
1654078310.820		9	2748	305.3333	66.0000	66
1654078317.820		16	9838	614.8750	139.5000	66

Figure 4.1: A piece of a csv output file produced by FS. We have highlighted the windows corresponding to an interaction and the ones corresponding to silence. By looking at the TS column one can indeed notice that between an interaction and the following one there is a one-minute pause. As shown in the figure, the dimension of interactions may vary significantly and also the traffic produced during silence intervals. Our objective is finding a way to understand when there is silence and remove those windows from the dataset.

To recap, the main problem is understanding when a window produced by FS is related to an interaction with the device and when it is instead a silent window. Having a clean ground truth, without noise, is essential when performing machine learning analysis. For this reason, the interactions detected had to be as precise as possible. This challenge, which may look easy to solve, has been instead quite complex and we are going to see why in the next sections.

4.2. Possible Solutions

We figured out two possible approaches to isolate the interaction windows:

- **First approach:** since we know that between the reproduction of one audio and the next one there are 60 seconds, we can somehow "hardcode" the position of interaction windows by looking at the timestamp (TS column). This may be done

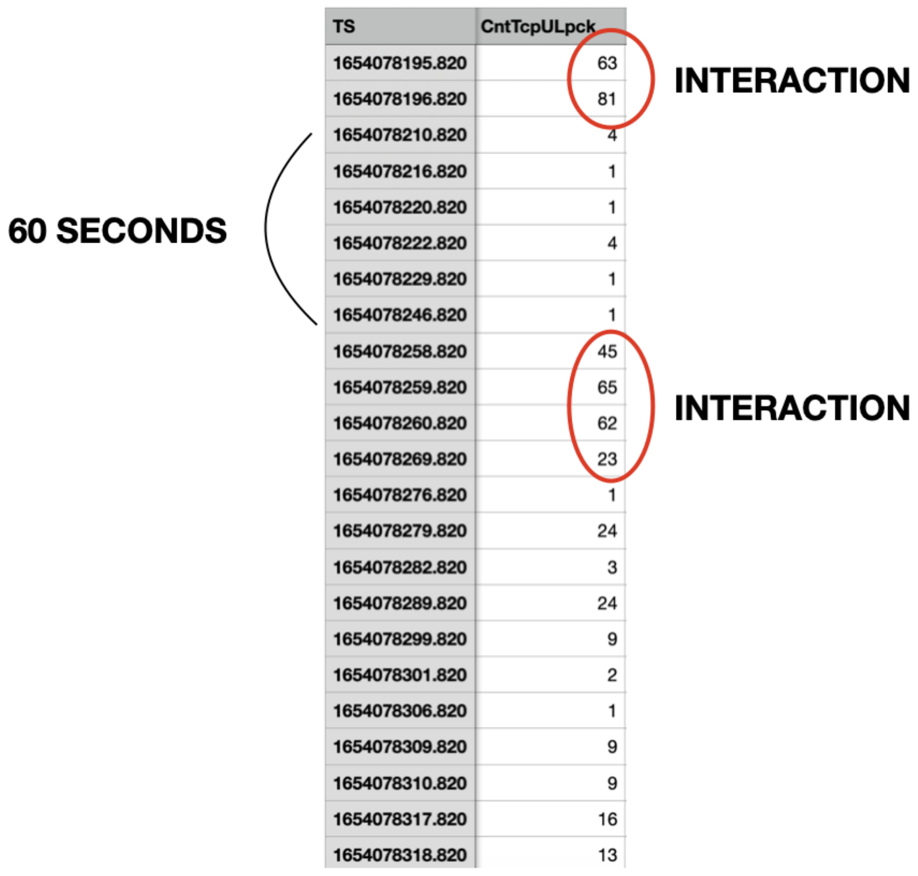
by retrieving the length of the different audios in order to keep the windows corresponding then, skip the next 60 windows (supposing we are using a window of 1 second, if we are using 0.5 seconds the numbers are doubled), and take the successive interaction windows by looking at the length of the next audio.
This first approach might look fine, but brings some issues. First of all, we are not sure that Alexa understood all the questions during audios reproduction. With this method, we are not sure if some traffic exchange is happening and we would probably add noise or silence, classified as interaction, to the final dataset. Secondly, we know the audio duration but we don't know when the user started talking in that audio. For example, we have received an audio sample of 5 seconds in which the user started talking after the first second and stopped at the third. Therefore, yes we are blindly taking all the interactions, but we are introducing a relevant noise to our dataset, even though the interactions are all part of it. In addition, we have to consider that the traffic produced by Alexa is not always perfectly aligned with the start and the end of the question and there may be a delay, due to communication protocols and physical lag, that we are not considering with this approach.

- **Second approach:** this method is based on finding a feature that exceeds or stays below a specific value (threshold) when an interaction happens. By using this threshold, we should be able to precisely identify all the interactions. The main challenges of this approach are first in finding the correct threshold and feature just by looking at the CSV files. Secondly, with Feature-Sniffer, we computed values within a range of time, this means that if an interaction starts at the end of the window we may find a lower (or higher, depending on the feature) value with respect to the threshold defined. So, the risk here is to find the windows corresponding only to the mid part of the interaction, ignoring the start and the end where values might not respect the threshold chosen. Another consideration to make is that the perfect threshold does not exist, thus we will always have a percentage of windows classified as interactions but which are not (false positives).

4.3. Guessing The Silence Threshold

Having seen the drawbacks of the first approach, we decided to explore the second one to understand if it could lead us to the desired result. First of all, we had to spot the right feature to use. In [5] the authors used the inter-arrival time between uplink packets to understand if someone is speaking to the device. If the inter-arrival time is lower than a chosen value then, the portion of traffic is classified as an interaction.

Anyway, this approach couldn't be applied to our use case because, due to Feature-Sniffer, we don't have the inter-arrival time between each packet but we only have the average inter-arrival time, computed for each time window of fixed duration (1 second and 0.5 seconds for our experiment). Moreover, if no uplink packet is sent in a time window, FS forces the feature for that window to be zero. Moreover, just by analyzing our CSV files, we didn't notice any particular pattern linking the average uplink inter-arrival time with the presence of an interaction.
We instead noticed that the **number of TCP packets sent in uplink (CntTcpULpck)** could behave as a really interesting feature to localize interactions. It sounds in fact reasonable to believe that, when we interact with the Echo, the device will send a greater number of packets to the server than it usually does.


INTERACTION
60 SECONDS
INTERACTION

TS	CntTcpULpck
1654078195.820	63
1654078196.820	81
1654078210.820	4
1654078216.820	1
1654078220.820	1
1654078222.820	4
1654078229.820	1
1654078246.820	1
1654078258.820	45
1654078259.820	65
1654078260.820	62
1654078269.820	23
1654078276.820	1
1654078279.820	24
1654078282.820	3
1654078289.820	24
1654078299.820	9
1654078301.820	2
1654078306.820	1
1654078309.820	9
1654078310.820	9
1654078317.820	16
1654078318.820	13

Figure 4.2: This is a piece of one CSV where we isolated the threshold column to show how the value of this particular feature increases when an interaction is happening. A double check on this assumption is always given by the time between the end of the previous interaction and the beginning of the next one, which is always close to 60 seconds.

Now we spotted a feature that can act as an indicator for interactions, the next step is finding the right value to use as a threshold. To do so, we recorded, always using FS, four hours of traffic where nobody talked to Alexa. We then analyzed the CSV produced to study the upper bound of the selected feature when no one is interacting with the device. To perform this analysis a simple python script has been used, it printed each value of the feature with its occurrences (number of times it was present in the CSV) plus the complementary of its CDF (Cumulative Distribution Function) value. The complementary of the CDF value of x is the probability of having (in that CSV) another value greater than x. We used the CDF to understand the probability of having windows of silence wrongly detected as interactions by using a determined threshold (probability of having false positives). Of course, the lower the probability the better the results, but if we choose a value for the threshold which is too high, we have the opposite risk of misclassifying interactions as silence (false negatives).

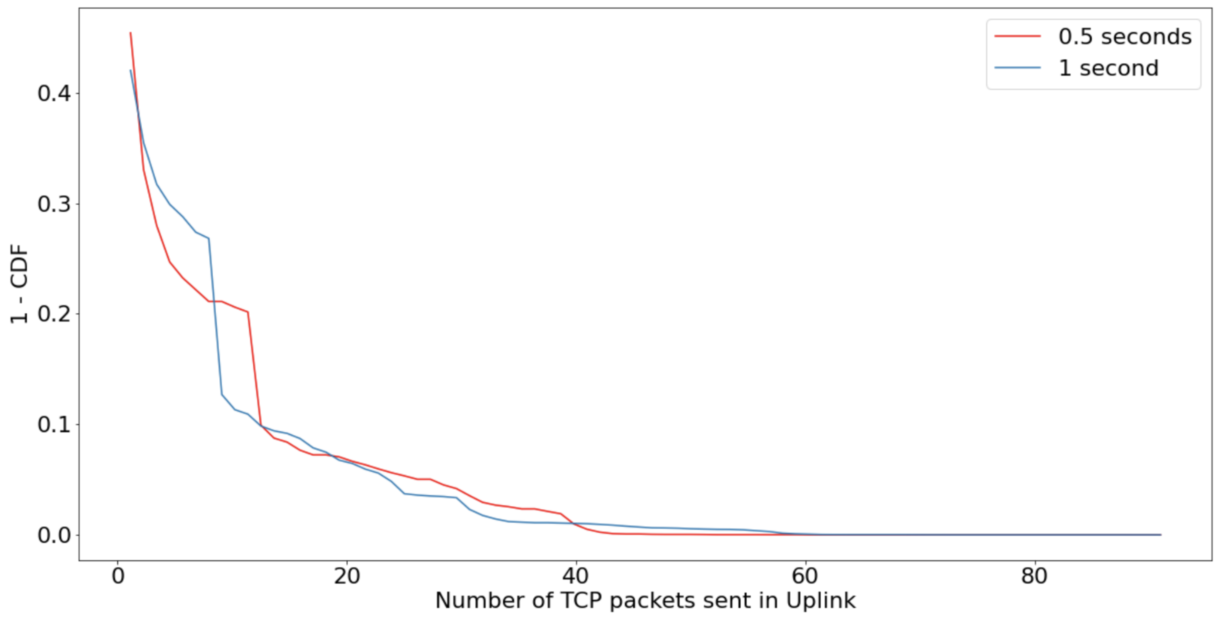


Figure 4.3: Comparison of the complementary of the CDF for the silence CSV using a 1-second window and 0.5-second window

From figure 4.3, we can see that there is a probability of less than 8% of sending more than 20 packets in a "silence" window. One can also notice that there are some differences between 0.5 and 1-second windows, meaning that different threshold values should be used when using different time window durations.
After some fine-tuning we find out that the best threshold value for a window of 1 second is of 40 (with a probability of misclassifying silence of 1%) while, for a window of 0.5 seconds, the value is 22 (with a probability of misclassifying silence of 4.1%). Even if

their cumulative distribution functions are similar, there is a strong gap between the two thresholds. In fact, the selected threshold does not depend on the silence values (which do not differ that much) but on the interaction values. Working with the 0.5-second window the interaction values are halved and so, to correctly spot them, we had to lower the silence threshold, otherwise we would have skipped some useful data classifying interactions as silence.

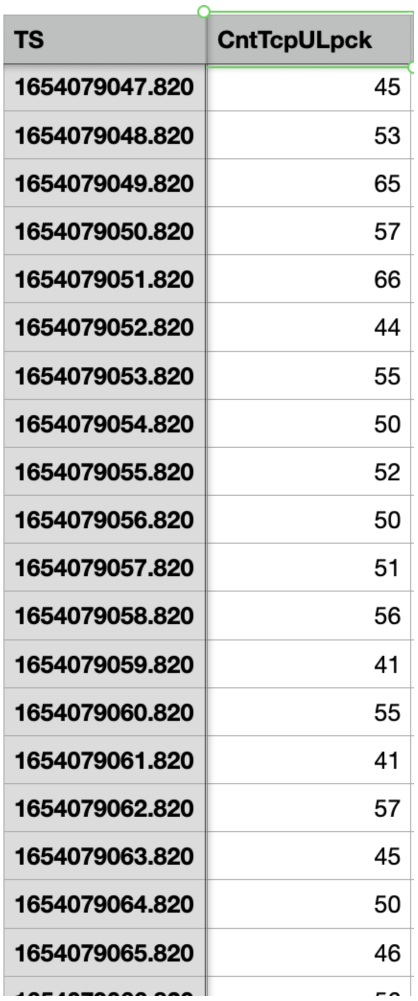
4.4. Cleaning

Having chosen the threshold feature and a range of values to try, we started to isolate interactions to understand which value worked better. To isolate interactions we considered consecutive windows (considering the timestamp) over the threshold as a single interaction. A python script did this for us, producing a CSV made of two columns containing the starting and ending timestamp of each interaction. After that, this CSV has been used to analyze the performance of this approach by counting the interactions, their length, and their frequency.
After the first attempts, we noticed that we detected a larger number of interactions than expected. So, with this approach, we were including also some false positive windows. Anyway, the most interesting thing we observed was the fact that some interactions resulted really long, around one minute or even more. The presence of such long interactions for sure could not happen with the audio samples we recorded. With a further investigation, by analyzing the source CSV using the timestamp of such interactions, we discovered that, at least one time per user (more or less one time every 60 questions), Alexa started sending a very long uplink flow to the server. By only inspecting the traffic it is not possible to know why this happens and what is contained in those packets, since the traffic is encrypted. Our hypothesis is that, after being triggered many times, the Amazon Echo sends some "reports" to the server, maybe for statistical customer analysis or advanced machine learning purposes. Anyway, the thing is that these "reports" (we will call the long interactions as reports from now on) are introducing a great noise to our dataset. First, because they alter the results of our script which is recognizing them as interactions, and secondly because they overlapped also with correct interactions, bringing all the values of most of the uplink features highly above the average. For these reasons, we decided to consider them as pure noise and remove them from the dataset. In order to remove the reports we use a properly written python script that uses the following parameters:

- **Report Threshold**: a threshold value used to identify the report windows.

- **Window Length**: length of the window set in FS, 0.5 or 1 second.
- **Max Length of Interaction**: the maximum time length that an interaction could have.

The idea is to parse our files and remove all the consecutive windows that have "CntTcpULpck" over **Report Threshold** and a length over **Window Length**. To compute **window length** we just took the longest audio recorded, there was no need to be precise since reports are consistently longer than our audio samples. To choose the Report Threshold instead, we performed some fine-tuning operations: we used a value a bit lower than the Silence Threshold since reports are long, but that does not have really high values. In the end, the ideal values of the Report Threshold revealed to be 30 for 1-second windows and 15 for 0.5-second windows. Anyway, these values don't have to be that precise: what matters to detect reports is finding a long list of consecutive windows with medium values of uplink packets.



TS	CntTcpULpck
1654079047.820	45
1654079048.820	53
1654079049.820	65
1654079050.820	57
1654079051.820	66
1654079052.820	44
1654079053.820	55
1654079054.820	50
1654079055.820	52
1654079056.820	50
1654079057.820	51
1654079058.820	56
1654079059.820	41
1654079060.820	55
1654079061.820	41
1654079062.820	57
1654079063.820	45
1654079064.820	50
1654079065.820	46

Figure 4.4: Example of how a piece of a report looks like.

After some tries, the script successfully removed all the reports. We estimated that in this cleaning procedure around 3% of our dataset has been lost bringing us from a total of 3240 interactions to 3143.

4.5. Interaction Detection

We have seen how to clean the data from reports and two possible approaches to isolate interactions after the data cleaning part. However, both the approaches inevitably add noise to our ending dataset and, in order to have a clean training set, we had to avoid this. For this reason, we couldn't adopt any of the two. This section presents in detail the final algorithm used to spot the interactions and create a dataset of interactions as clean as possible.

4.5.1. The Idea

Both the approaches presented in the previous section are not wrong, but simply incomplete. For what concerns the first one, with the "hard-coding" of timestamps (ts) we basically have an approximate location of all the interactions but the precision at which interactions are spotted (starting ts and ending ts) brings many false positives in the dataset. On the other hand, with the threshold approach, we have quite good precision in spotting the exact windows of an interaction, but we also get a significant number of other windows as false positives (fake interactions).
In our final solution we have thus decided to use both methodologies, exploiting the threshold to have knowledge of when the interaction starts and when it finishes combining the fact that our questions are asked every sixty seconds. By doing so, we had a range of time between one question and another where each interaction detected by the threshold algorithm could be ignored since we know it is for sure a silent period.
This is the idea behind the final algorithm, obviously the logic is way more complex and we will explore it deeply in the next subsections.

4.5.2. Hyper-parameters

The Interactions Detection Algorithm (IDA) has been coded and tested in Python using PyCharm. IDA takes some parameters as inputs, necessaries to regulate the metrics used to detect interactions. We set up some tests to control the results of IDA and fine-tuned the parameters to find the perfect values which allowed us to exactly spot all the interactions with maximum precision.

All the hyper parameters used are listed here:

- **window** (`float`): the length of the window produced by FS. If the window value changes, the other parameters should be adapted too.
- **column_threshold** (`string`): the feature used to consider the threshold. In our case, we always used the total number of TCP packets sent in uplink in a window time (CntTcpULpck).
- **threshold** (`integer`): if the feature is above the threshold, the window is considered as part of an interaction.
- **secondary_threshold** (`integer`): a secondary threshold used to correctly spot the beginning and end of an interaction, we will explain how this parameter works later.
- **use_secondary_threshold** (`Boolean`): a simple control variable to check whether using the secondary threshold or not.
- **average_time_between_interactions** (`integer`): the time between one question and the next one (in seconds).
- **average_audio_length** (`integer`): the average length of our questions recorded.
- **interval** (`integer`): a tolerance range inside which the next interaction could happen, we will explore how this parameter works later.
- **continuity_value** (`integer`): a value that specifies how many consecutive values under the threshold we may find inside an interaction.

After a fine-tuning phase, the best hyper-parameters values to isolate interactions using the two considered window values revealed to be the followings:

Final hyper-parameters values

	0.5 sec window	**1 sec window**
threshold	22	40
secondary_threshold	15	30
avg_time_btw_interactions	60	60
avg_audio_length	6-7	6-7
interval	10	7
continuity_value	1	1

Table 4.1: Hyperparameters optimized values.

4.5.3. The Algorithm

This section has the purpose of explaining in detail the logic of the proposed algorithm (IDA). We divided the problem of detecting interactions into two sub-problems equally important. The first sub-problem consists in correctly identifying where are located all the interactions, in other words, finding the approximate position of each interaction. This means that if we have sixty interactions we must spot exactly sixty different groups of windows corresponding to those interactions. The second sub-problem comes right after the first, focusing on the single interaction, since we have a range of windows where we know is located an interaction, the challenge becomes correctly understanding the exact windows at which the interaction starts and ends. We can translate these two sub-problems into "quantity" and "quality". If we want to perform machine learning on this kind of dataset, it is fundamental to have only real interactions (quantity) in the training dataset, but it is also equally important to have interactions spotted with precision and without noise (quality). Of course, it is desirable to have a bigger number of interactions detected, but it might be worth having a smaller, but cleaner, number of interactions. To make an example, we could have designed an algorithm that counts 60 seconds and then takes the next 10 windows classifying them as an interaction. For sure this algorithm correctly spots and takes all the interactions, but also adds to them a lot of windows where no traffic occurs, altering the cleanness of our dataset.
Let's see now how IDA faces the first sub-problem: interaction spotting. Before going on, a clarification has to be made, it is true that we reproduced an audio sample every sixty seconds, but this does not necessarily mean that there must be exactly sixty seconds between an interaction (seen in traffic windows) and the successive one. This is due to three possible reasons: first, since we collected voices through an online form we could not supervise and standardize recordings and so, a sample recorded may contain one or more seconds of silence before the question for Alexa is effectively pronounced. Having, for example, two seconds of silence before the question brings to 62 seconds of distance between two interactions. Secondly, there may be a little delay, even one second, between the speaker's question and the traffic produced by the Echo. Finally, some questions are not understood by Alexa. If the "Alexa" (or "Echo") word is not understood by the device no traffic trace is produced and this causes a gap of even more than 120 seconds between interactions. This also happens when we consider interactions deleted from the dataset when performing the reports cleaning phase. With these considerations in mind, it becomes clear that the sixty seconds can be used only as an indicator and that the interactions are not equally distributed over time. However, the difference is of just a few seconds: if an interaction is detected after 30 seconds from the previous one, one of the

two (or maybe both) must be a false positive. Therefore we implemented IDA using the parameter **interval** (in seconds) which gives a range of time where interactions can be spotted using the threshold. For example, with a window of one second, we found out that the optimal value of the interval was 7 seconds. This means that, as we detected an interaction, the next one can fall after (60 - 7, 60 + 7) seconds. We used an interval a little bigger than needed in order to be sure that all interactions are considered and so that we do not discard a correct interaction because of the too-small interval value. If we are not in the range generated by the **interval** and **average_time_between_interactions** parameters, then all the windows over the threshold are discarded. If we are instead inside that range of time, a group of windows over the threshold is considered an interaction.

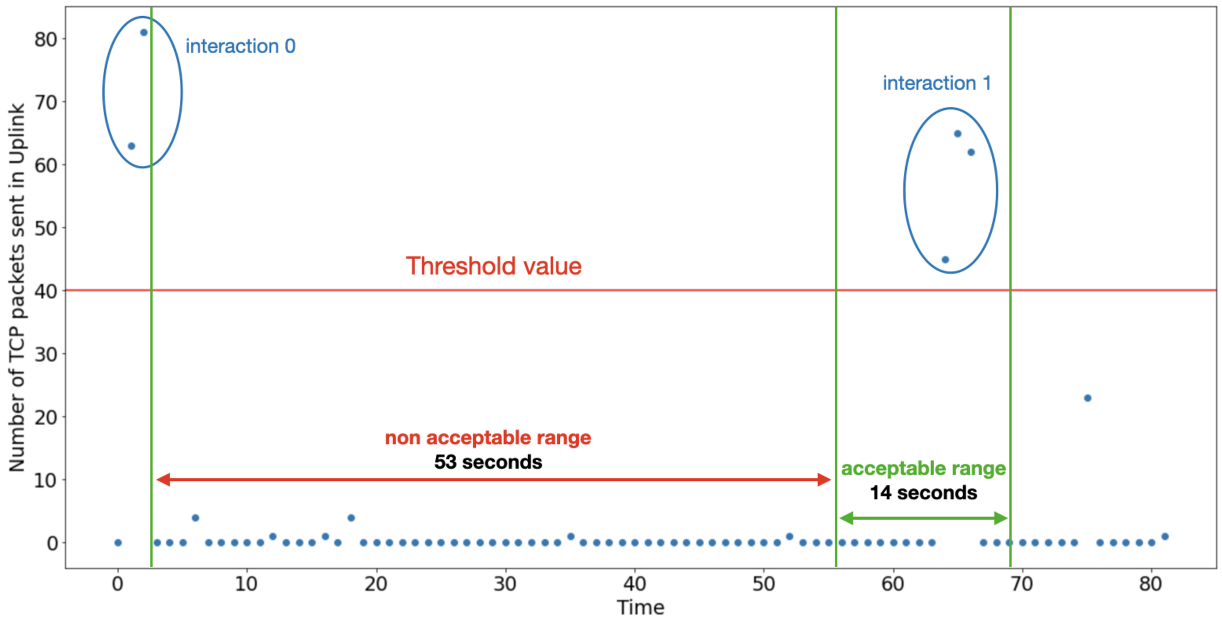


Figure 4.5: In this example we have represented the number of uplink packets in a time range of around 80 seconds. Each blue point represents a one-second window. The red line represents our threshold value which indicates "dots" that can be considered interaction windows. IDA uses parameters "interval" (7 seconds) and "average_time_between_interactions" (60 seconds) to compute, after each interaction, an "acceptable range" of fall-in for the next interaction. In this example, we had really low values in the non-acceptable range. However, even if we had windows way over the threshold, those windows would have been discarded, since located inside the non-acceptable area. Thanks to this logic we are able to greatly reduce the number of false positive interactions.

It can also happen that, due to different reasons already explained above, a question may be skipped, causing a time gap between consecutive interactions of more than 120 seconds. In this case, IDA computes the acceptable range considering also the parameter **avg_audio_length**. Here we have to take into account a timer of 60 seconds, the length of the question that did not trigger Alexa (or caused the report), and another 60 seconds timer to correctly spot the successive interaction. Depending on the speaker and the question, all the interactions have different lengths; to set up the correct value of **avg_audio_length** we started from the average length of our audios and then proceeded by trial and error to find the value which lead to the best result. This is also the main reason why we have a high value of the **interval** parameter: because it helps to compensate the variability of audio sample lengths.

Now that we have seen how IDA spots the interactions, we focus on how all the windows of the interaction are chosen. The threshold value is a great indicator to understand whether a window is part of an interaction or not. However, we are interested in finding all the windows which contain even also few packets of the interaction. Due to the mechanism of Feature-Sniffer, it may happen that one interaction starts at the end of a window and ends at the beginning of another one; this causes FS to produce windows, which are part of the interaction, with a number of uplink packets lower than the threshold value. So the risk here is that IDA only considers the core of the interaction, but not the extreme windows. In addition to this, it can also happen that the interaction may have some intermediate windows a little lower than the threshold value. Given so, if we want to augment the precision of our dataset we have to take into account also those two cases in order to have a dataset as precise as possible. Usually, as far as we have seen, the threshold-feature's values between windows in an interaction follow a sort of Gaussian trend, starting low, increasing in the mid and ending low, anyway it doesn't always happen that way; in the following figures, we reported all the different types of trends that we considered when we designed the logic to identify the interactions.

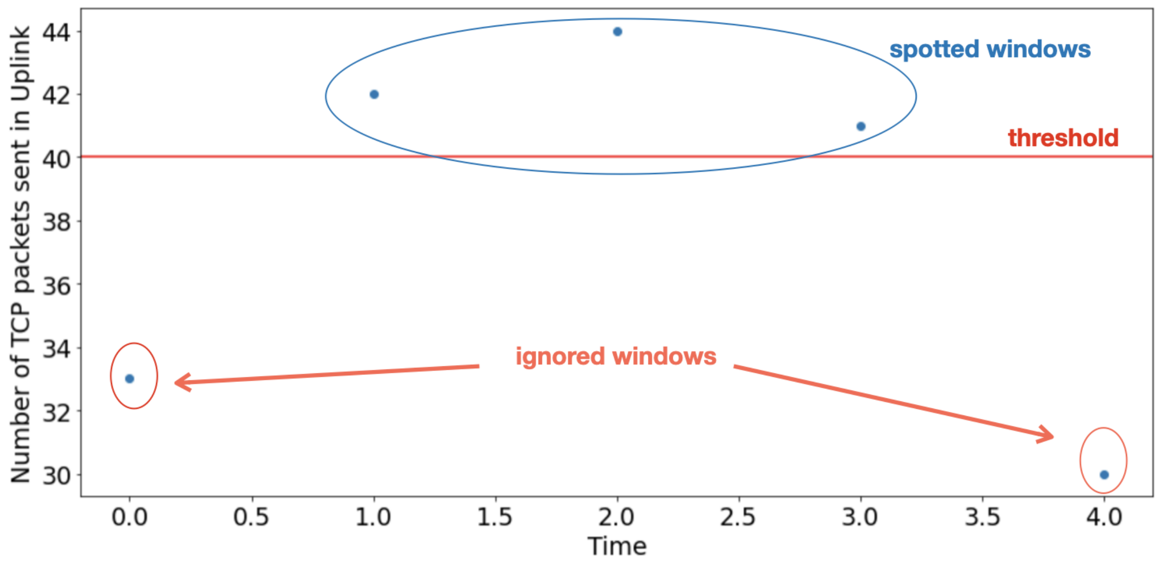


Figure 4.6: This is the most standard type of interaction, the start and the end of the question for Alexa are, respectively, at the end of the window and at the beginning, causing the first and last windows of the interaction to have a value lower than the threshold.

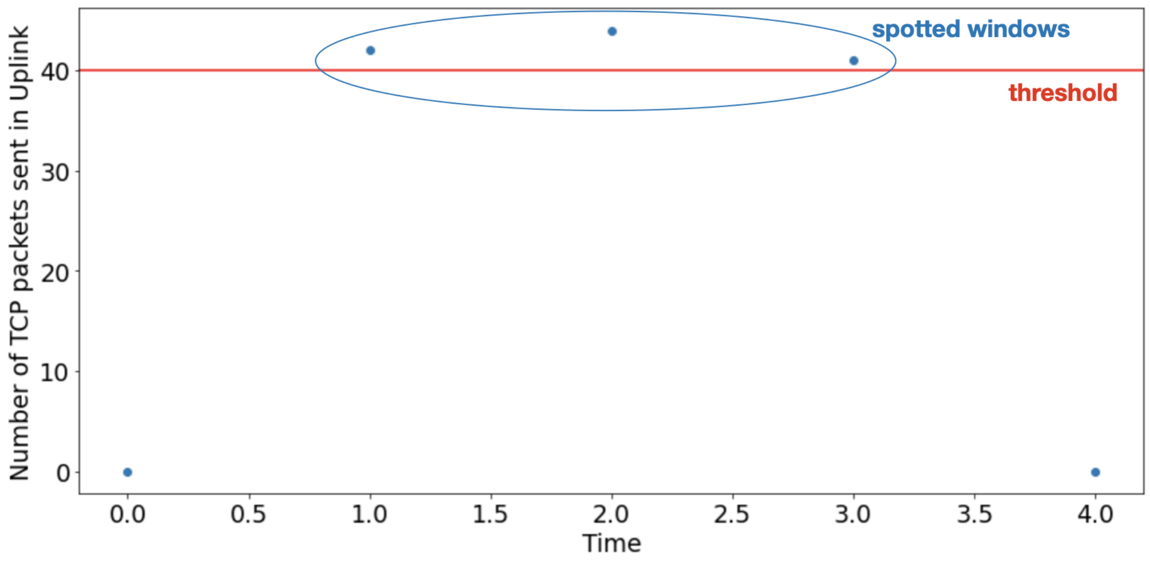


Figure 4.7: This is the other most common type of interaction, the traffic produced by the speaker's question coincides with the start of the window and ends with the window, so all the windows are spotted correctly just by looking at the threshold value.

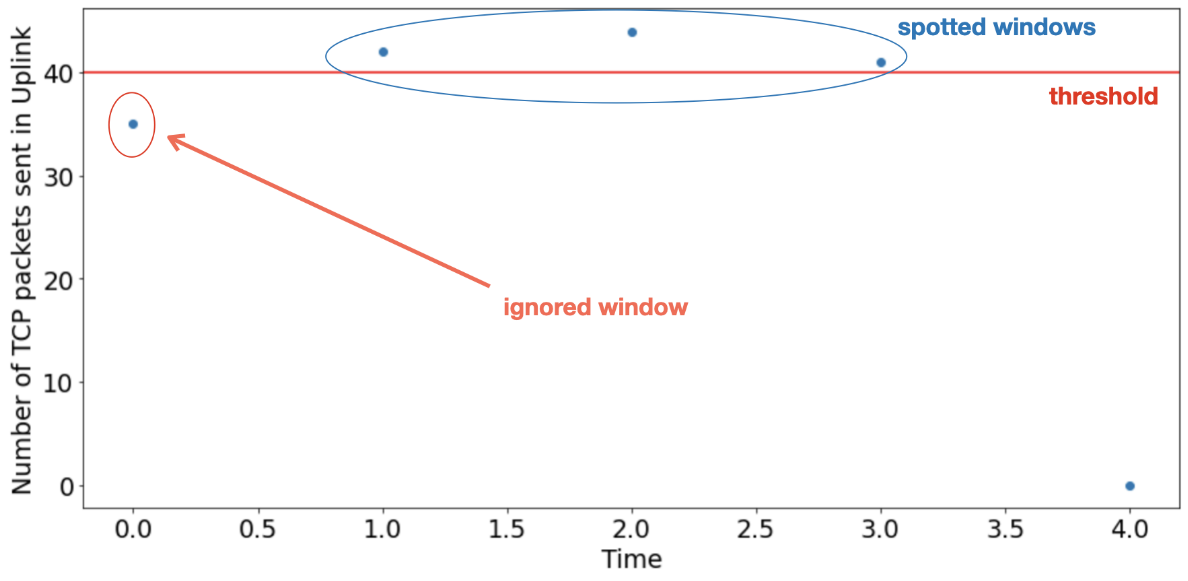


Figure 4.8: Similar to the first type of interaction, but in this case it is ignored only the first window because the last window has a value higher than the threshold.

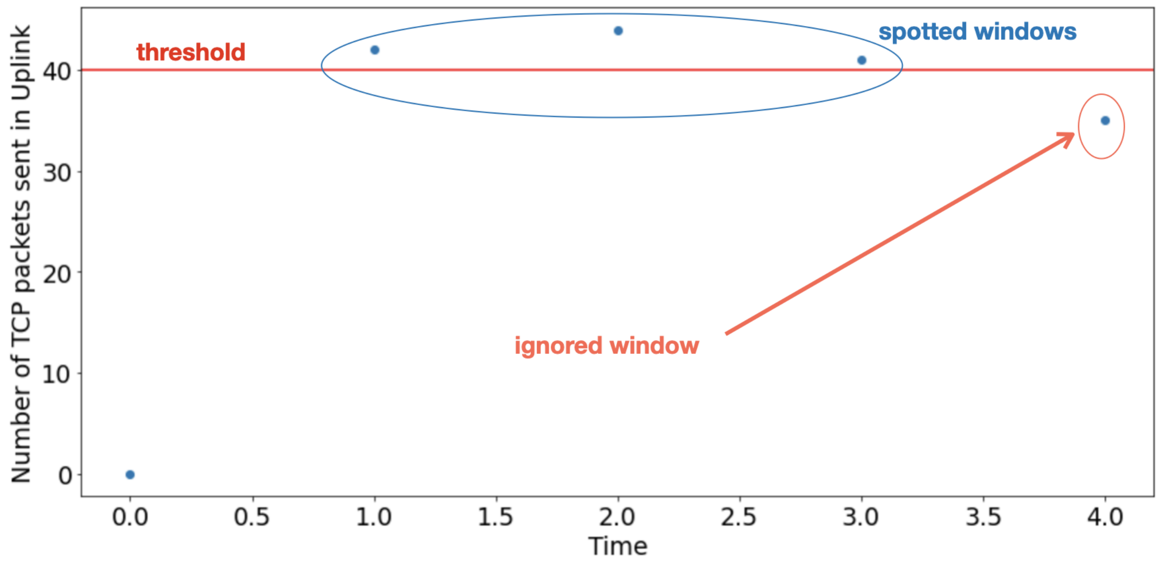


Figure 4.9: Like the previous type, but in this case the interaction starts aligned with the window but ends at the beginning of the last window, so the last part of the interaction is ignored.

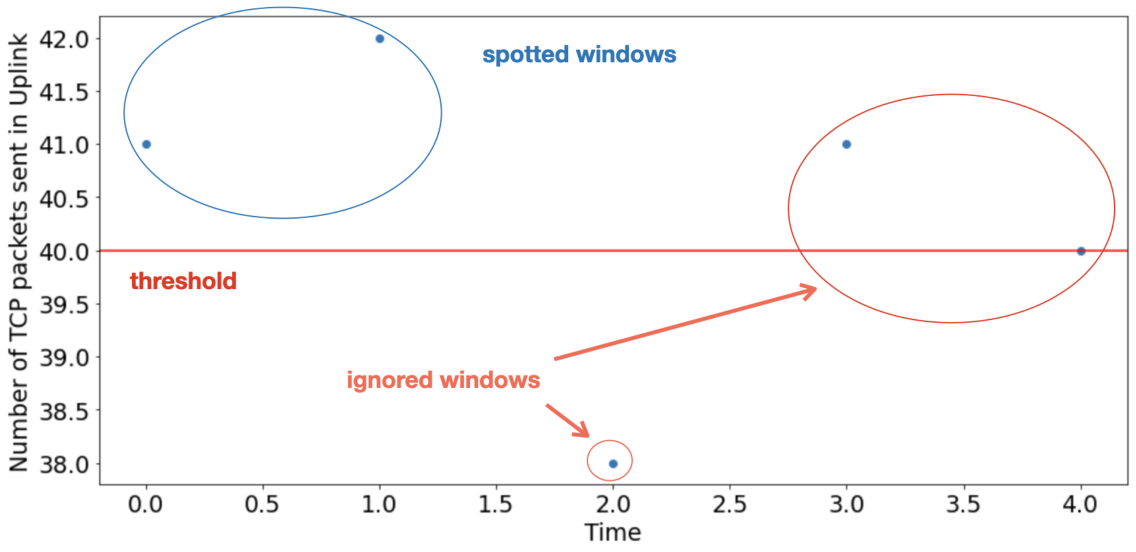


Figure 4.10: This is the rarest type of interaction. In this case, all the windows belong to the same interaction, however, in the mid, one of them has a value slightly lower than the threshold. This causes IDA to consider as part of the interaction only the first two windows, ignoring the third. Note that the fourth and the fifth windows are not taken even though they are above the threshold because IDA considers the interaction stopped when met the first window under the threshold. So, after the third window, the non-acceptable range is recomputed and the fourth and fifth windows fall in this range being discarded by the algorithm.

In all the graphs above we have seen the possible shapes of interactions, these are the most common cases but of course, there may be other combinations of the listed types. For example, we can find interactions presenting the peculiarities of Figure 4.10 combined with 4.6. All these interactions (except for the one reported in Figure 4.7) have in common the fact that they are not captured by IDA in their entirety, but some windows are ignored. To correct this problem and increase the precision of IDA, we decided to adopt another parameter: **secondary_threshold**. The secondary threshold introduces a value lower than the threshold which is used, after an interaction detection, to check whether we may have lost some windows at the beginning and at the end of it. Basically, IDA checks first if the windows are in the acceptable range. Then, it looks for consecutive windows over the threshold: after a group of consecutive windows is detected and classified as an interaction, the windows before and after the group detected are examined. We are verifying whether those two windows have a value higher than the secondary threshold: if the answer is positive, then those windows are considered as part of the interaction, otherwise, they are discarded. Similarly to the threshold, the secondary threshold value is been established by fine-tuning and comparing the results obtained.

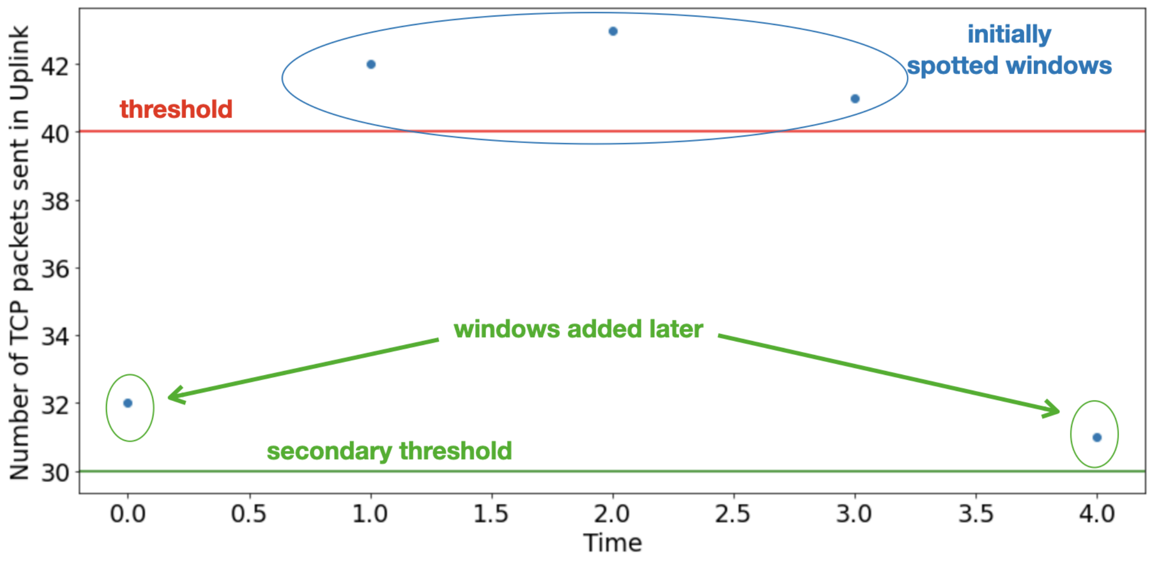


Figure 4.11: An example explaining the logic behind secondary threshold utilization. After the three windows over the threshold are spotted and classified as interaction windows, the ends of the interaction are compared with the secondary threshold. The beginning of the interaction shows a value of 32, and the concluding window has a value of 31. Both the values are below the threshold, but over the secondary threshold, so the windows are classified as part of the interaction.

A further consideration has to be made for the interaction type shown in 4.10. It happens in fact for some interactions to have, in the middle of the interaction, windows with a value slightly lower than the threshold. In those cases, it is used always the secondary threshold, anyway the risk here is that IDA spots the middle window as the end of the interaction not considering the last windows.

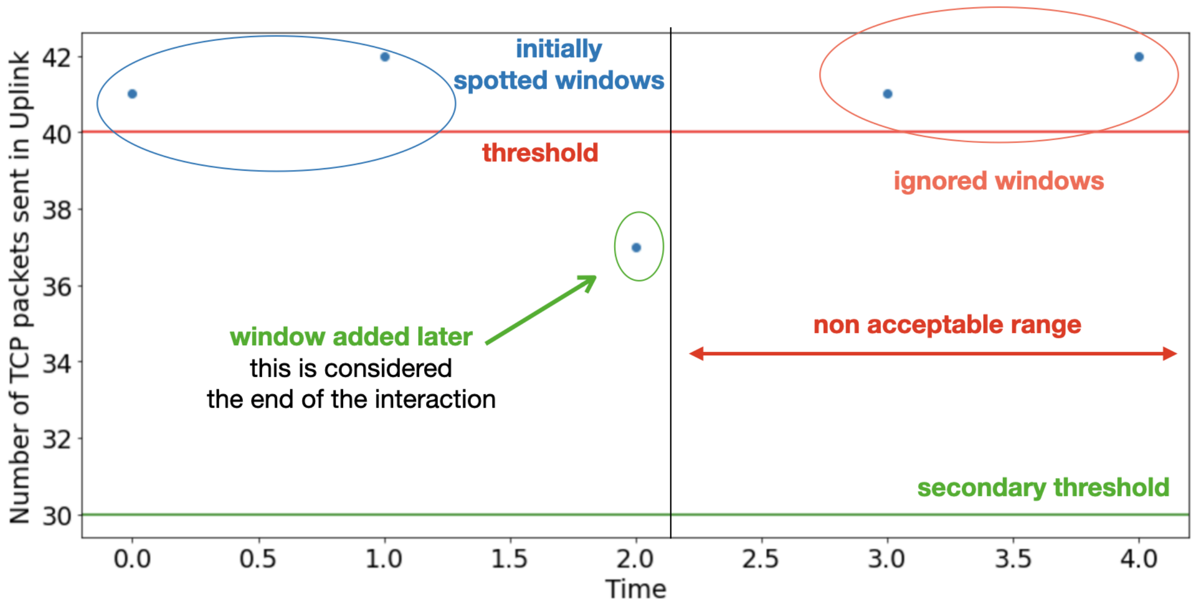


Figure 4.12: Figure showing the problem described in the precedent page.

To avoid this from happening we introduced the last parameter: **continuity_value**. This value regulates the maximum number of consecutive windows which can be lower than the threshold in the mid of an interaction. Essentially, while IDA parses windows in the interaction, if a value lower than the threshold is met, before classifying it as the end of the interaction the successive window is observed: if this window is over the threshold too, then the mid window is considered as inside the interaction (if greater than the secondary threshold) and IDA keeps parsing the windows. The continuity value depends on the threshold value, on the length of the window, and on the other parameters. We experimented that, for a 1-second window, we may find at maximum one consecutive window below the threshold in the middle of an interaction. By setting up a higher threshold value or a lower window length this value needs to be higher. The parameter values we used bring a really precise final dataset. However, this does not mean that other combinations of values might not bring to a similar or even better result. It is all matter of fine-tuning, and choosing coherent hyper-parameters values.

4.5.4. Testing

Testing has been a fundamental step while developing IDA. The problem of not having a verified ground truth makes it impossible to tell whether IDA works perfectly or not. However, we exploited some characteristics of our dataset in order to build metrics to evaluate IDA's results. The first metric used is the number of interactions detected by

the algorithm. We had a total of 3240 interactions in our dataset, this number is reduced to 3143 after report cleaning. We can assume also that a small percentage of interactions may be lost due to badly recorded audio samples (i.e. misspelling the activation word) or not understood questions by the device. Therefore, we can conclude that the total number of interactions that IDA should detect has to be 3143 or, more presumably, a bit lower. This is a first good metric to use to understand if we are taking false positive interactions or whether we are missing some of them. In our case, since we had a quite large dataset, it's preferable to miss some interaction, but with the certainty of having a clean dataset. A second metric to use is the length of the interaction spotted, an interaction too long is probably a report not cleaned from the dataset while, an interaction too short (one or two windows only), might be an outlier window wrongly detected as an interaction. Finally, a really useful metric is the time between the interactions detected, we know we are asking one question to Alexa every sixty seconds, so we are expecting a time between interactions of around one minute (excluding when a question is missed, in those cases the time is, more or less, multiple of 60 seconds). As seen in the explanation of IDA's logic we already use the time between interactions to compute the acceptable range, however, our acceptable interval is quite large (14 seconds) and, for this reason, it may happen to have more values above the threshold in this interval causing a misclassification of interactions. For obvious reasons, we could not examine thousands of interactions by hand to spot errors in the algorithm. Therefore we coded a script, still in Python, to parse the timestamps of the interactions detected and highlight "weird" interactions, probably identified by error. This algorithm just checked the length of the interactions, highlighting really small ones, and also checked the distance between them. If two interactions are too close (let's say 54 seconds distance with respect to the expected 60 seconds), then the interactions timestamps are printed with a little alert to explain the warning. The same happens if the interactions are too distant. Thanks to this script, we were able to reduce the number of interactions to examine manually.
However, at first runs, we noticed that the number of "weird" interactions detected by the test script was really high. After an inspection of some of the interactions highlighted, we discovered that this was caused by outliers windows which fell at the extremes of the acceptable range causing the real interaction to be ignored and shift of the successively computed acceptable ranges.

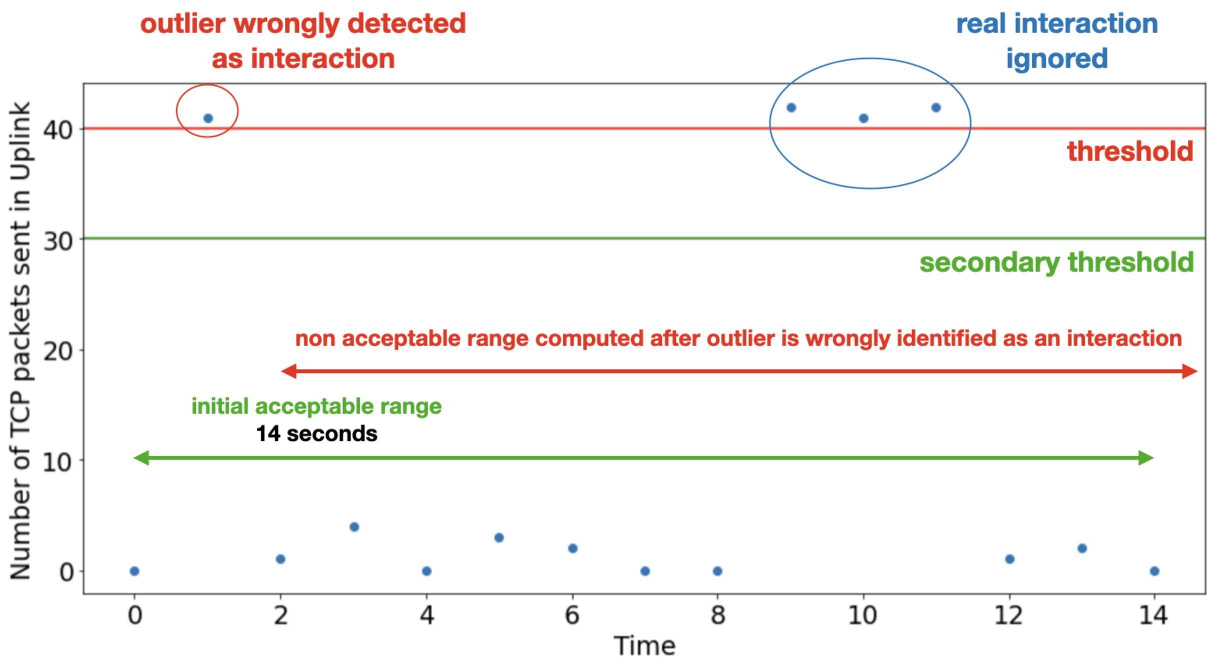


Figure 4.13: This figure exemplifies the problem just described. If an outlier window falls in the acceptable range before the interaction windows, then IDA wrongly identifies the outlier as the interaction, discarding the real one. This also causes a wrong alignment of the newly computed acceptable range which can cause the exclusion of successive correct interactions.

To correct and avoid this kind of error we added a few lines to IDA's code in order to discard one-window interactions, since outliers are mainly isolated windows. After this little adjustment and hyper-parameters fine-tuning, no more warnings were outputted by the test script, meaning that IDA was perfectly doing its job.
In the end, in its final version, IDA detected a total of 3075 interactions, missing 165 interactions with respect to the starting 3143. This number sounds reasonable to us. The 165 missing interactions are probably mainly questions that did not trigger Alexa and partially might be interactions not spotted by IDA. Assuming they are all not-triggered interactions, then we have an average of 3 interactions per user without an answer, which makes sense considering each user asked sixty questions.

Final Dataset with IDA

	Number of Interactions
After collection phase	3240
After report cleaning	3143
Final dataset (Detected by IDA)	3075

Table 4.2: Final dataset with IDA.

4.5.5. Computing the features

Once we were sure that the interactions are correctly identified by the algorithm, the next step has been computing the features to use in the machine learning process. FS automatically computes features for each time window: these features are statistical measures of the traffic that occurred in that range of time. We have for example the average number of TCP uplink packets, the total number, the variance, and so on. However, to perform classification tasks based on interactions (e.g. speaker recognition) we don't need the features related to the single window, but to the whole interaction. If an interaction is composed of multiple windows, then their features have to be combined in order to obtain features related to the entire interaction that will be used in the learning phase. This can be considered the weak point of using Feature-Sniffer to perform this kind of machine-learning task. In fact, we not only have less informative features, since averaged in a one-second window but, moreover, we have to reduce the information contained in them too, in order to compute the final features for the interaction. Anyway, this is not a problem for all the features. For example, if we want the number of uplink packets in the whole interaction, it is sufficient to sum the number of packets sent in all the windows of the interaction and we are not losing information. If the number of samples is the same for each window, also the average of averages does not cause a loss of information. More critical instead are the features regarding the variance, the mode, or the median, because are impossible to mediate without losing information. Features related to packet inter-arrival time particularly suffer this loss of information too.
To compute the final features we wrote a dictionary in python containing all the features (from Feature-Sniffer) that we wanted to bring in the final dataset as keys, and, as values, some keywords related to the statistical measures that we wanted to use to produce the final feature. So, while computing the features, IDA could look up the dictionary to know how to combine the values of the windows.

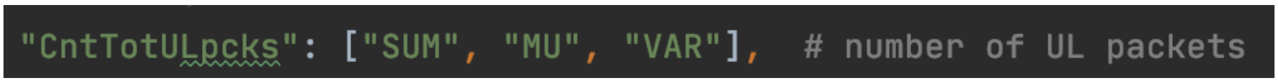
Here is the legend of the keyword used:

Features legend

	operation
"MU"	average
"SUM"	summation
"STD"	standard deviation
"VAR"	variance

Table 4.3: Features' legend

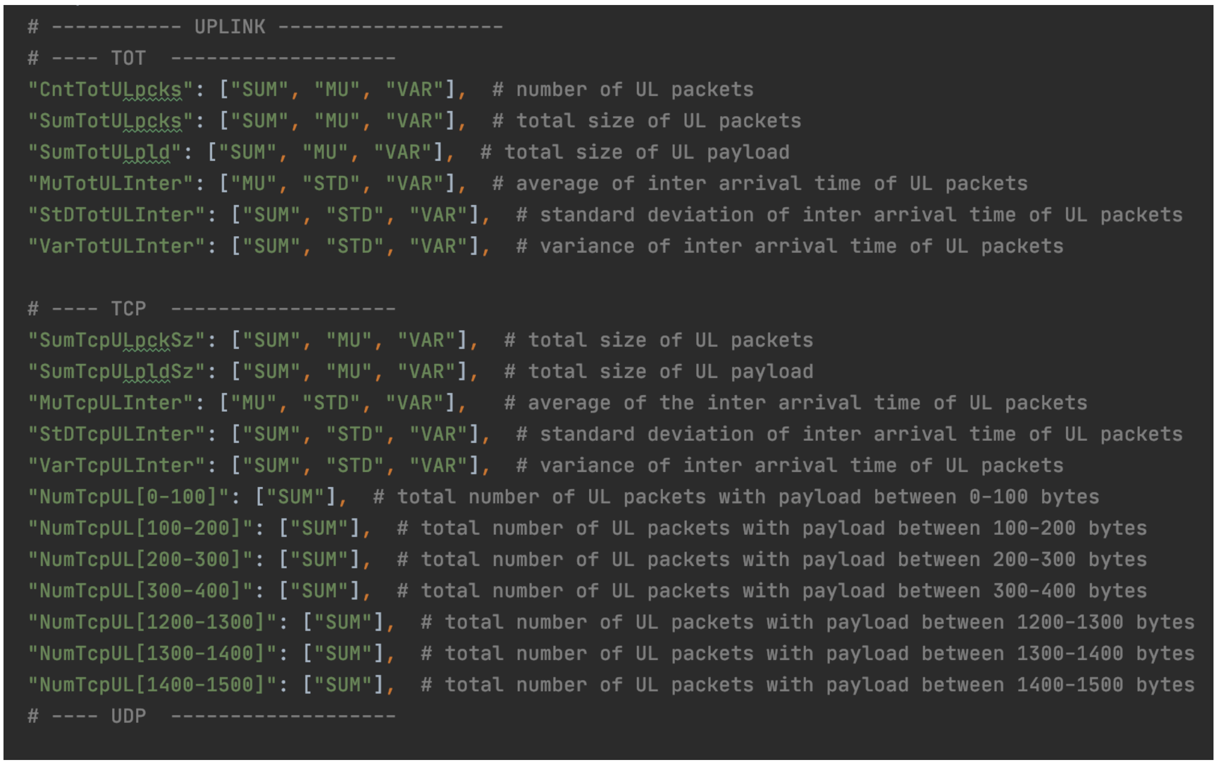
Let's see an example of feature creation by looking at this line in the dictionary.



```
"CntTotULpcks": ["SUM", "MU", "VAR"],  # number of UL packets
```

Figure 4.14: A line of features dictionary.

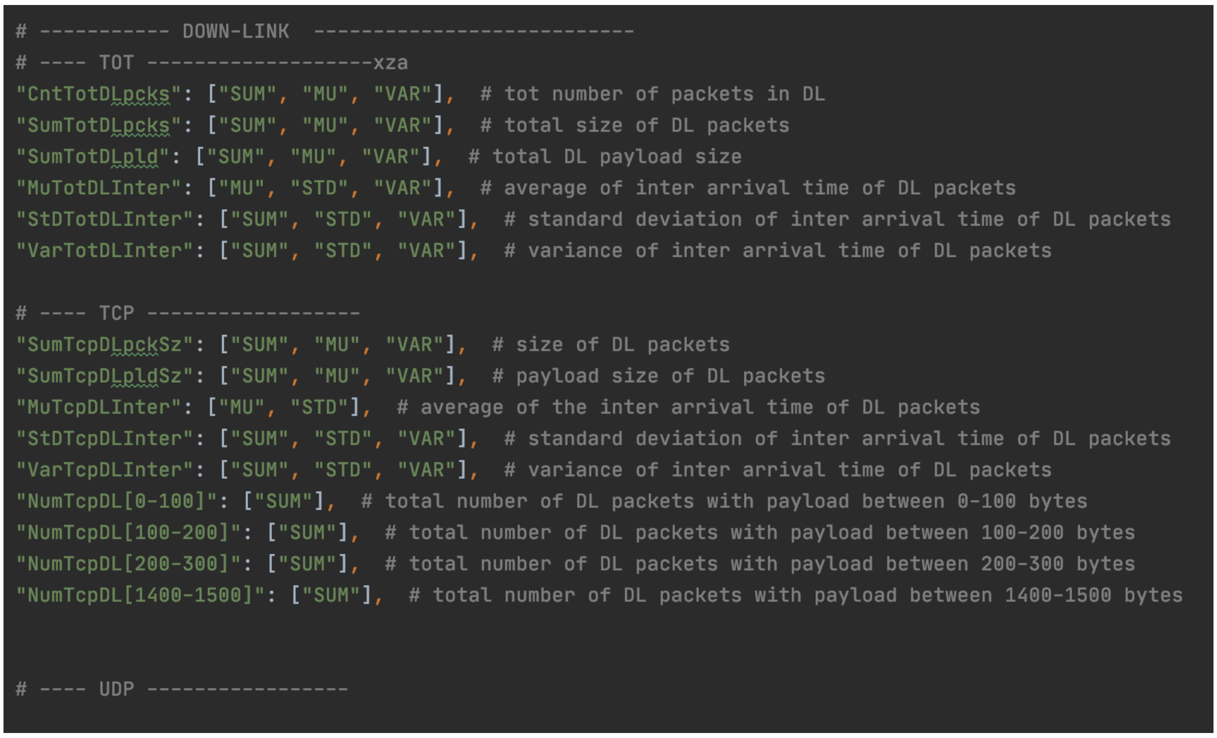
The key indicates the feature column produced by Feature-Sniffer, in this case the total number of uplink packets sent. The values of the dictionary instead tell how to combine feature values, here we are summing the windows, averaging them and computing their variance. So, in the end, we are producing three different features out of one. We decided to use this method in order to maintain as much information as possible.
Since they can always be discarded later, during the feature selection phase, it is useful to produce more features as possible. We divided the features produced in: "uplink", "downlink" and "total". Uplink features contain information regarding only packets sent from the Echo device to the server, downlink features contain packets from the server to the Echo, while the "total" features consider all the traffic of the device. In addition, each category contains features divided per TCP traffic only, UDP traffic, or both TCP and UDP. Finally, we have a section containing other features.
In the following pages we reported the whole set of IDA features dictionary with a comment for each feature describing its content.



```
# ----------- UPLINK -------------------
# ---- TOT  -------------------
"CntTotULpcks": ["SUM", "MU", "VAR"],  # number of UL packets
"SumTotULpcks": ["SUM", "MU", "VAR"],  # total size of UL packets
"SumTotULpld": ["SUM", "MU", "VAR"],  # total size of UL payload
"MuTotULInter": ["MU", "STD", "VAR"],  # average of inter arrival time of UL packets
"StDTotULInter": ["SUM", "STD", "VAR"],  # standard deviation of inter arrival time of UL packets
"VarTotULInter": ["SUM", "STD", "VAR"],  # variance of inter arrival time of UL packets

# ---- TCP  -------------------
"SumTcpULpckSz": ["SUM", "MU", "VAR"],  # total size of UL packets
"SumTcpULpldSz": ["SUM", "MU", "VAR"],  # total size of UL payload
"MuTcpULInter": ["MU", "STD", "VAR"],   # average of the inter arrival time of UL packets
"StDTcpULInter": ["SUM", "STD", "VAR"],  # standard deviation of inter arrival time of UL packets
"VarTcpULInter": ["SUM", "STD", "VAR"],  # variance of inter arrival time of UL packets
"NumTcpUL[0-100]": ["SUM"],  # total number of UL packets with payload between 0-100 bytes
"NumTcpUL[100-200]": ["SUM"],  # total number of UL packets with payload between 100-200 bytes
"NumTcpUL[200-300]": ["SUM"],  # total number of UL packets with payload between 200-300 bytes
"NumTcpUL[300-400]": ["SUM"],  # total number of UL packets with payload between 300-400 bytes
"NumTcpUL[1200-1300]": ["SUM"],  # total number of UL packets with payload between 1200-1300 bytes
"NumTcpUL[1300-1400]": ["SUM"],  # total number of UL packets with payload between 1300-1400 bytes
"NumTcpUL[1400-1500]": ["SUM"],  # total number of UL packets with payload between 1400-1500 bytes
# ---- UDP  -------------------
```

Figure 4.15: Uplink dictionary.



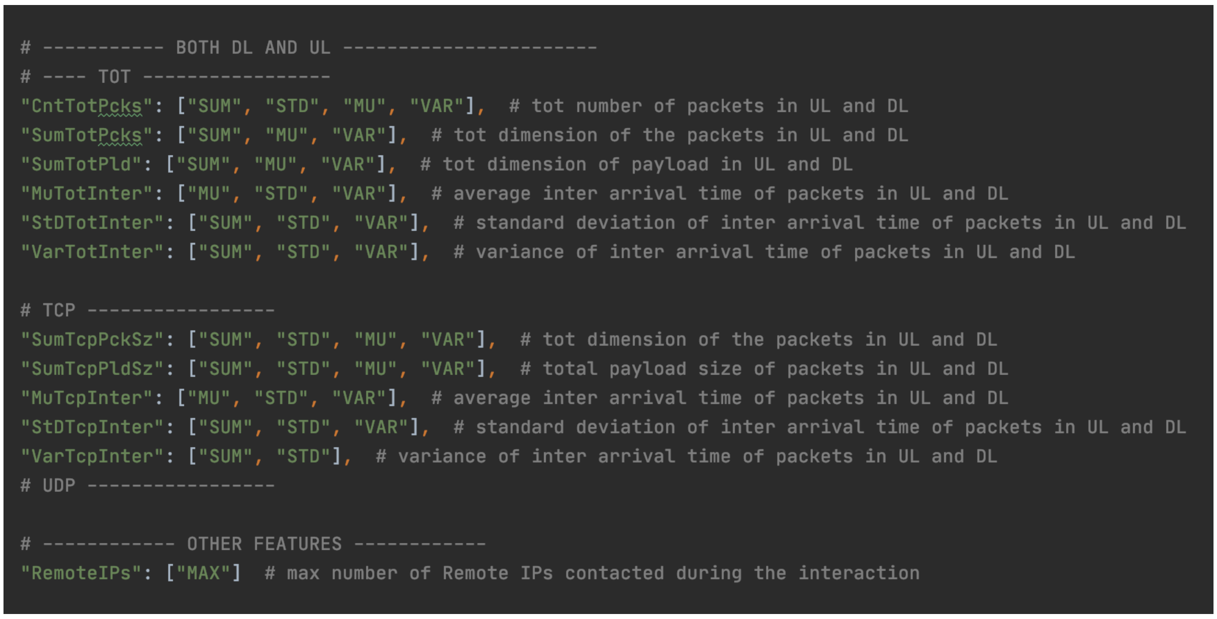
```
# ----------- DOWN-LINK  ---------------------------
# ---- TOT -------------------xza
"CntTotDLpcks": ["SUM", "MU", "VAR"],  # tot number of packets in DL
"SumTotDLpcks": ["SUM", "MU", "VAR"],  # total size of DL packets
"SumTotDLpld": ["SUM", "MU", "VAR"],  # total DL payload size
"MuTotDLInter": ["MU", "STD", "VAR"],  # average of inter arrival time of DL packets
"StDTotDLInter": ["SUM", "STD", "VAR"],  # standard deviation of inter arrival time of DL packets
"VarTotDLInter": ["SUM", "STD", "VAR"],  # variance of inter arrival time of DL packets

# ---- TCP ------------------
"SumTcpDLpckSz": ["SUM", "MU", "VAR"],  # size of DL packets
"SumTcpDLpldSz": ["SUM", "MU", "VAR"],  # payload size of DL packets
"MuTcpDLInter": ["MU", "STD"],  # average of the inter arrival time of DL packets
"StDTcpDLInter": ["SUM", "STD", "VAR"],  # standard deviation of inter arrival time of DL packets
"VarTcpDLInter": ["SUM", "STD", "VAR"],  # variance of inter arrival time of DL packets
"NumTcpDL[0-100]": ["SUM"],  # total number of DL packets with payload between 0-100 bytes
"NumTcpDL[100-200]": ["SUM"],  # total number of DL packets with payload between 100-200 bytes
"NumTcpDL[200-300]": ["SUM"],  # total number of DL packets with payload between 200-300 bytes
"NumTcpDL[1400-1500]": ["SUM"],  # total number of DL packets with payload between 1400-1500 bytes



# ---- UDP -----------------
```

Figure 4.16: Downlink dictionary.



```
# ---------- BOTH DL AND UL ----------------------
# ---- TOT ----------------
"CntTotPcks": ["SUM", "STD", "MU", "VAR"],  # tot number of packets in UL and DL
"SumTotPcks": ["SUM", "MU", "VAR"],  # tot dimension of the packets in UL and DL
"SumTotPld": ["SUM", "MU", "VAR"],  # tot dimension of payload in UL and DL
"MuTotInter": ["MU", "STD", "VAR"],  # average inter arrival time of packets in UL and DL
"StDTotInter": ["SUM", "STD", "VAR"],  # standard deviation of inter arrival time of packets in UL and DL
"VarTotInter": ["SUM", "STD", "VAR"],  # variance of inter arrival time of packets in UL and DL

# TCP ----------------
"SumTcpPckSz": ["SUM", "STD", "MU", "VAR"],  # tot dimension of the packets in UL and DL
"SumTcpPldSz": ["SUM", "STD", "MU", "VAR"],  # total payload size of packets in UL and DL
"MuTcpInter": ["MU", "STD", "VAR"],  # average inter arrival time of packets in UL and DL
"StDTcpInter": ["SUM", "STD", "VAR"],  # standard deviation of inter arrival time of packets in UL and DL
"VarTcpInter": ["SUM", "STD"],  # variance of inter arrival time of packets in UL and DL
# UDP ----------------

# ----------- OTHER FEATURES -----------
"RemoteIPs": ["MAX"]  # max number of Remote IPs contacted during the interaction
```

Figure 4.17: Features computed considering both uplink and downlink.

Note that we have not taken any feature related to UDP traffic. This is simply because UDP traffic is not informative for our learning tasks, since the Amazon Echo device traffic is transmitted in TCP packets.

4.5.6. The Window Problem

Before concluding the chapter it is worth to make a consideration regarding the window length. In our experiment, we worked with two different sizes: one second and half a second. We first developed and optimized IDA for the window of 1 second and then moved on with the 0.5. However, while fine-tuning the hyperparameters for the smaller window, we noticed that it becomes way more difficult to spot interactions as we decrease the window's size. One may think that, if we halve the value of the window, also the values of the hyperparameters have to be halved but it does not work like that. Let's assume, for example, that a normal traffic communication between the Amazon Echo and its servers lasts 0.25 seconds, then the first 0.25 seconds of an interaction might be indistinguishable from a normal traffic exchange; we can differentiate interactions from normal traffic produced by the device because interactions last longer and so we are producing more packets in a single window, while a normal traffic exchange just stops at the beginning of the window. But if we reduce the window to 0.25, it becomes really hard to spot the differences between one single window of interaction and a window of normal protocol traffic. This is just to say that the functioning of IDA, regarding this research, is strongly related to the size of the window chosen. For windows larger than one second the

algorithm should still work fine, while for smaller windows it might be needed a rework of the algorithm.
In the end, if we lower the time window, we are getting more details of the traffic, but we are also facing the risk of misclassifying interactions and we have to average more values (windows) when producing the final features, losing a great part of the information we tried to gain.

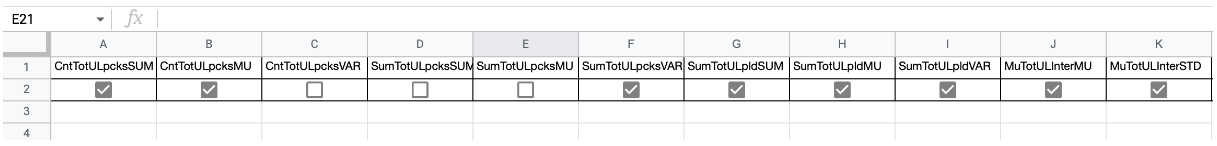
5 | Learning Process and Experimental Results

In this chapter, we are going to see the final results of our experiment. We remind here that our purpose is to test the usability of FS in a scenario in which we apply machine learning to encrypted traffic. We want to understand what are the limits of this tool and for what tasks it is instead an essential resource. We used the interactions, laboriously collected, to explore three different classification tasks:

- **TASK 1**: identify windows containing an interaction by directly applying machine learning on CSV files produced from Feature-Sniffer. We developed IDA with this task in mind. It is indeed important to highlight that IDA works mainly because we are imposing the condition of having a question reproduced every sixty seconds. This condition will be no more available in a real-life scenario, or when the final models will be deployed and used. For this reason, it is important to understand if the windows we have struggled to collect can be used to train a model with the purpose of spotting interactions without using timing or other constraints.
- **TASK 2**: distinguishing the genre of the speaker between a human or a bot. This task focuses mostly on the uplink traffic flow since, regardless of the speaker's genre, Alexa will always reply with the same downlink traffic to the same question. So, exploring this task gives us information about the ability of FS to extract relevant data related to the inter-arrival rate of packets and uplink flow.
- **TASK 3**: distinguishing the language of the speaker between English and Italian. This task allows us to have insights into FS ability to collect information regarding the content of the traffic. An English question corresponds to an English response, this implies a difference, between Italian and English in the uplink and downlink flow of the Echo traffic and not only in the speech frequency, as in the case above.

5.1. Machine Learning environment

A little technical note to be precise regarding the software we used. For all the learning tasks, we performed the feature and data analysis in the coding environment provided by Google Colaboratory, using their notebooks. Notebooks were written in python and, for the machine learning models we used pandas to manipulate the dataframes and scikit-learn to fit the models and compute our predictions. We linked the various notebooks to a personal drive where we uploaded the training data. In this drive has been also uploaded a google sheet with the list of all the features with a simple flag under each feature, this sheet is processed by the notebook code in order to know which features to use when training the models. We thought this was an easy and comfortable way to store feature configurations with the results they produced.


E21

	A	B	C	D	E	F	G	H	I	J	K
1	CntTotULpcksSUM	CntTotULpcksMU	CntTotULpcksVAR	SumTotULpcksSUM	SumTotULpcksMU	SumTotULpcksVAR	SumTotULpldSUM	SumTotULpldMU	SumTotULpldVAR	MuTotULInterMU	MuTotULInterSTD
2	☑	☑	☐	☐	☐	☑	☑	☑	☑	☑	☑
3											
4											

Figure 5.1: This is part of the sheet we used to tell the model which features to consider, by flagging the checkbox below the feature we are telling the notebook to consider it.

5.2. Interaction Detection Task

The aim of the first task is to obtain a model able to create a dataset of interactions, as close as possible to ground truth, right out of Feature-Sniffer's output, by classifying a single window as part of an interaction or not. In this case, the dataset we are using is quite different from the next tasks, because we are not managing interactions as input, but single windows of traffic. Basically, we are using the files produced by Feature-Sniffer with a label, obtained thanks to IDA, telling the model whether a window is part of an interaction or not.

5.2.1. Preparation and Feature Selection

For this task, the features of the starting dataset are different from the ones that will be used in successive cases. In fact, we have features related to the single window and no more averaged on a set of windows that corresponds to an interaction. With respect to this consideration, we have way less information loss than before. On this dataset it has not been necessary to perform any particular transformation like normalization or so on, we simply have removed the reports which we have talked about in section 4.4. We removed them because reports are made of windows that look like interactions but that instead are not and we didn't want these windows to introduce noise in our dataset. The idea was, in fact, to build an algorithm able to isolate the interactions, as precisely as possible, after a first report cleaning phase (see Figure 5.2).

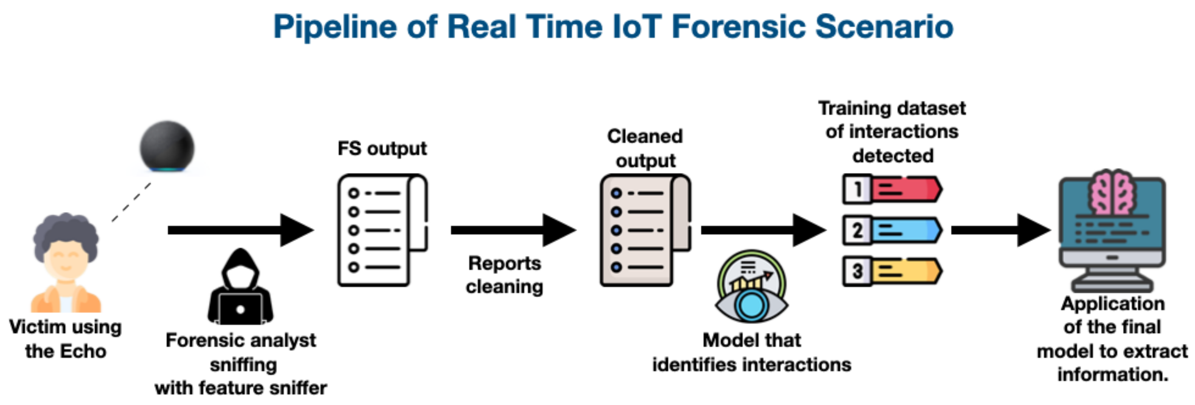

Pipeline of Real Time IoT Forensic Scenario
Victim using the Echo
Forensic analyst sniffing with feature sniffer
FS output
Reports cleaning
Cleaned output
Model that identifies interactions
Training dataset of interactions detected
1
2
3
Application of the final model to extract information.

Figure 5.2: Steps performed by an hypothetical forensic analyst in a real time scenario. In this scenario our model would be used in a data pre-processing phase to identify and isolate the interactions on which will be applied the final model.

For this task, we don't have to perform any particular feature analysis, since the first subset of features we choose gives at first try an excellent result. With respect to the other tasks, fewer features have been used. We selected the most relevant features, as follows: features regarding the number of packets sent in uplink (a higher number of uplink packets may indicate that someone is speaking with Alexa), the inter-arrival time of the packets (for the same reason), but also features related to the payload of the packets, even in downlink (a large content in downlink probably identifies an answer to a user question). Here is the list of all the features used to train the model:

- **Downlink:**
  - Number of TCP packets received
  - Sum of the TCP packets' payload received
  - Average size of a TCP packet
  - Mode of the size of a TCP packet
  - Variance of the TCP size packets
  - Average inter-arrival time of TCP packets

  - Variance of the inter-arrival time of TCP packets
  - Number of TCP packets with payload between 100 and 200
  - Number of TCP packets with payload between 200 and 300
  - Number of TCP packets with payload between 300 and 400
  - Number of TCP packets with payload between 1400 and 1500

- **Uplink:**
  - Number of TCP packets received
  - Sum of the TCP packets' payload received
  - Average size of a TCP packet
  - Mode of the size of a TCP packet
  - Variance of the TCP size packets
  - Average inter-arrival time of TCP packets
  - Variance of the inter-arrival time of TCP packets
  - Number of TCP packets with payload between 0 and 100
  - Number of TCP packets with payload between 100 and 200
  - Number of TCP packets with payload between 200 and 300
  - Number of TCP packets with payload between 300 and 400
  - Number of TCP packets with payload between 1200 and 1300
  - Number of TCP packets with payload between 1300 and 1400
  - Number of TCP packets with payload between 1400 and 1500

One thing to consider is that using this dataset, the number of windows that are labeled as non-interaction is way more than the ones which are part of interactions. This happens because there is one minute of silence between different interactions, causing the generation of a lot of silence windows. In order to avoid an unbalanced training set, which might lead the model to predict only silence, we removed part of the silence windows. In the end, we obtained a training set of 13968 windows, half of which were part of interactions and half of them correspond to silent periods. The validation set was composed of 3491 windows and the test set of 3081.

5.2.2. Models Tried and Results

For all the classification tasks we compared and tried the following classification models:

- Decision Tree
- Random Forest
- Gradient Boosting
- Naive Bayes
- K-Nearest Neighbor
- Support Vector Machine

For this task, we obtained fantastic results. Using windows of one-second duration, all the selected models reached an accuracy over the 95%; the best results have been achieved by Random Forest and Gradient Boosting with an accuracy of 99% and an AUC (Area Under the ROC Curve, see section [3]) value of 0.999, close to perfection. For what concerns the 0.5-second dataset, the results were very similar, with Random Forest and Gradient Boosting always performing better with an accuracy of 99.8% on the test set and an AUC of 0.99.

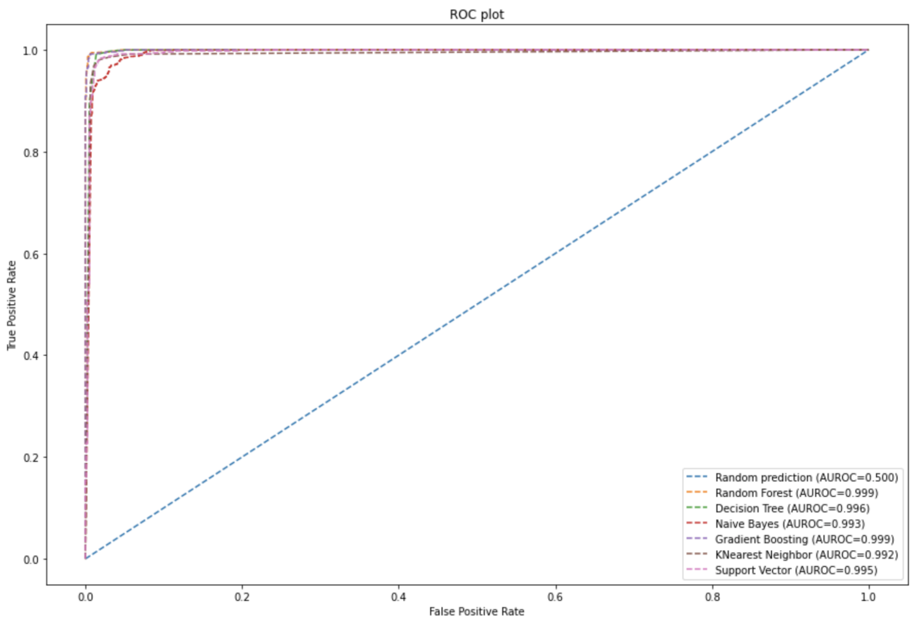


Figure 5.3: ROC curve for all the models we have tried in the classification task with the one-second dataset. The best models are Gradient Boosting and Random Forest. However, all the models obtained very good results in this task.

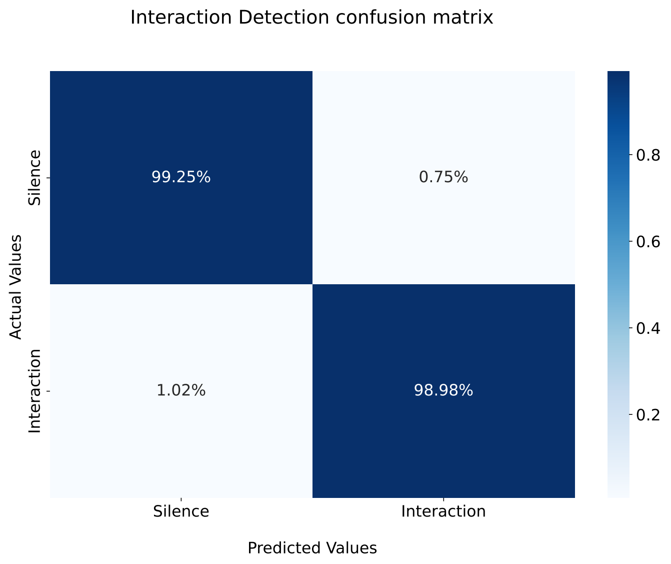


Figure 5.4: Confusion matrix of the Gradient Boosting algorithm for the one-second dataset, false negatives and positives are equally distributed in this case and represents a really small percentage of the total prediction. The predictions are almost perfect.

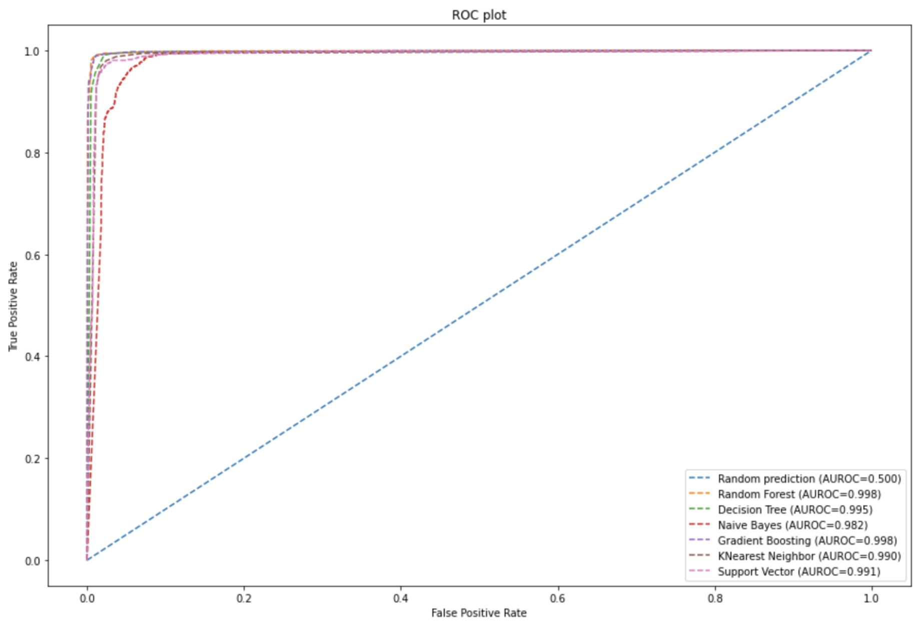


Figure 5.5: ROC curve for all the models we have tried in the classification task with the 0.5-second dataset. The best models are Random Forest and Gradient Boosting but, as before, all the models performed pretty well.

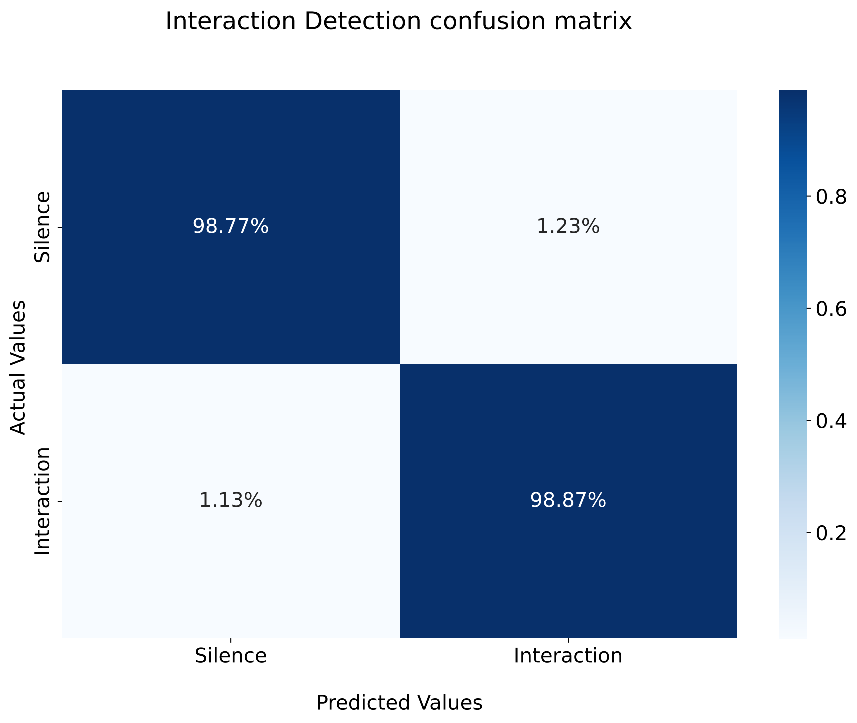


Figure 5.6: Confusion matrix of the Random Forest algorithm for the 0.5-second dataset. We have similar results with respect to the 1-second scenario.

We also have tried to train all the models feeding them with only the single feature used in IDA: the number of packets sent in uplink. Even though only one feature is used, the results are still very good, the best model is Random Forest with an accuracy of 95% on the test set and an AUROC value of 0.98. We can therefore conclude that it is the most important feature to use in this task. Of course, by using the complete set of features as shown above, the overall final result almost reaches perfection.

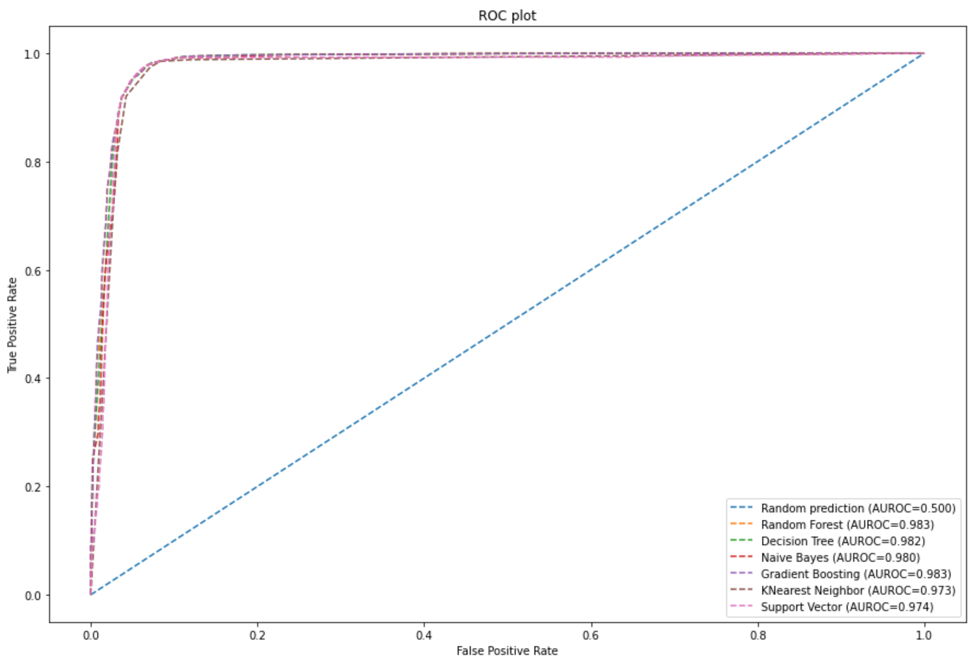


Figure 5.7: ROC curve for all the models we have tried in the classification task with the 0.5-second dataset. All the models performed well even though only one feature has been used.

One consideration that is worth making is that yes, these results are very good, anyway, it is sufficient one miss-classified window to ignore a whole interaction or to wrongly consider silence as an interaction. For this reason, our model needs to be as precise as possible. We made some experiments by running the model on the test set to see if the classification produced was correct and the answer has been positive. The very few errors we have found were mainly related to some outlier windows of silence, with anomalously high values, which were classified as interactions. Anyway, a large part of the interactions present in the test set were correctly spotted. Basically, the great part of the errors caused the addition of interactions or simply did not consider some of the extremes of an interaction, missing its start or its end. Missing the extremes of an interaction is way less critical than missing its center because this causes the interaction to be split into two different interactions. While, by missing its ends, we are simply losing a little piece of information and this is acceptable. We also have run our model on the silence dataset that we had used in section 4.3, containing only silence windows. The aim of this test was to have an indicator of how many false positives our model could produce. This test showed that, on a total of 16.622 silence windows, only 202 (1.04%) were classified as interactions.

5.2.3. Can We Ignore The Reports?

The results discussed in the previous section are great, however we were not completely satisfied with the results of the test regarding the outliers and the fact that, before applying the model in a real-time scenario, we had to filter out the reports. So, we decided to investigate if the model could learn how to not consider report windows. To answer this question we have tried to train a model on a one-second window dataset similar to the previous one, but containing report windows labeled as silent. The models produced by this training showed a similar performance with respect to the ones trained on the dataset without reports. In this case, the best models remained Gradient Boosting and Random Forest, but with a slightly lower accuracy 98.9% and the same AUROC value of 0.999. So, in the end, we lost almost nothing in terms of performance.

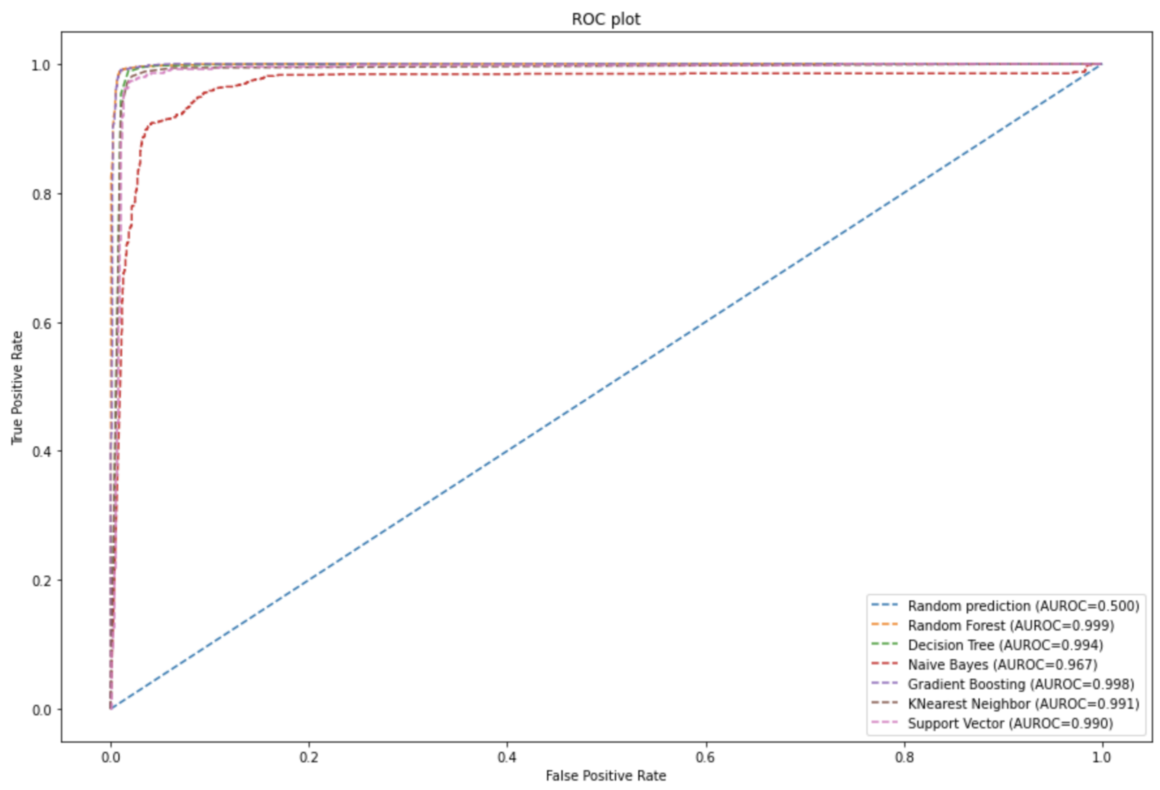


Figure 5.8: ROC curve for all the models we have tried in the classification task with the one-second dataset using the reports. The best models are Gradient Boosting and Random Forest. However, all the models performed well. AUROC values are almost the same with respect to 5.3. We only lost 1% of accuracy score.

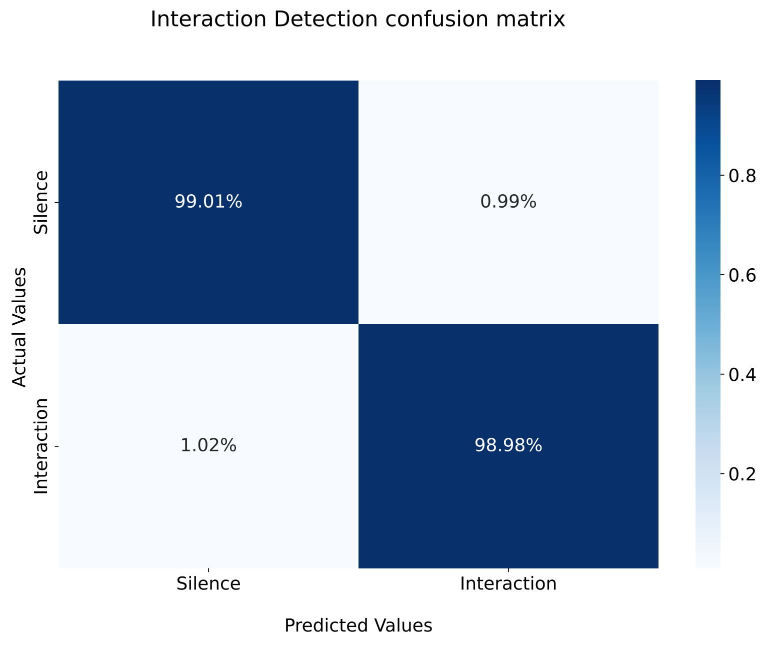


Figure 5.9: Confusion matrix of the Gradient Boosting algorithm for the one-second dataset using the reports. False negatives and positives are equally distributed in this case and represent a really small percentage of the total prediction. The predictions are almost perfect.

We then performed the outlier test on the silence dataset which, this time, revealed only 27 false positives out of 16622 (0.16%). This number, compared with the one produced by the previous model, highlights a great improvement in avoiding outliers.
We also tested the model on test data to see how reports windows were classified and, even if a small percentage was classified as interaction, the great majority of the reports was correctly classified as silence. In the end, the utilization of this model in a real-time scenario could be optimal because removes the reports cleaning phase out of the pipeline.

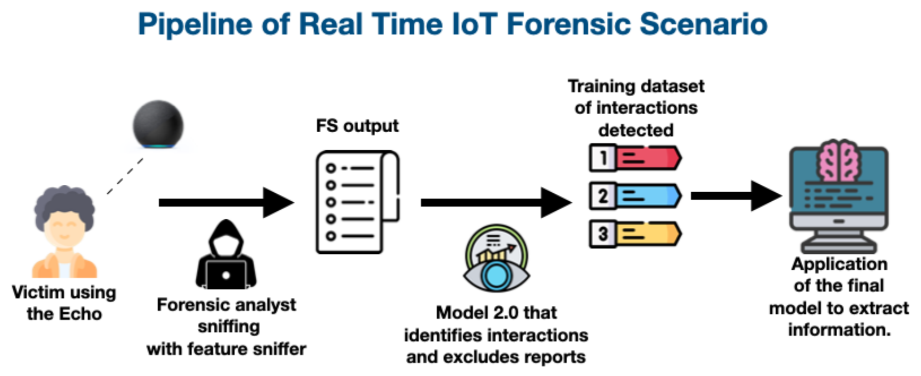

Pipeline of Real Time IoT Forensic Scenario
Victim using the Echo
Forensic analyst sniffing with feature sniffer
FS output
Model 2.0 that identifies interactions and excludes reports
Training dataset of interactions detected
1
2
3
Application of the final model to extract information.

Figure 5.10: New forensic analyst's pipeline with the use of this new model.

However, if we prefer precision with respect to the speed, for example, if we don't have the need to extract information in real-time, but we can do it at a later date, the optimal solution might be to perform reports cleaning and then apply the first model, since got a higher accuracy. It all depends on the needs.

5.3. Human or Bot Task

When facing this task it is important to compare our work with the one from Arensi ([1]). In the previous research, the author tested the same task but without using Feature Sniffer. He worked with four human voices, each person recorded 100 questions and four robot voices were generated with Amazon Polly. Unfortunately, the author does not specify if the bot voices were neural (really realistic) or standard (not realistic). Anyhow, in the end, Arensi got really good results, obtaining a model able to distinguish a human interaction from a robot-generated one, with an AUROC value of 0.90.

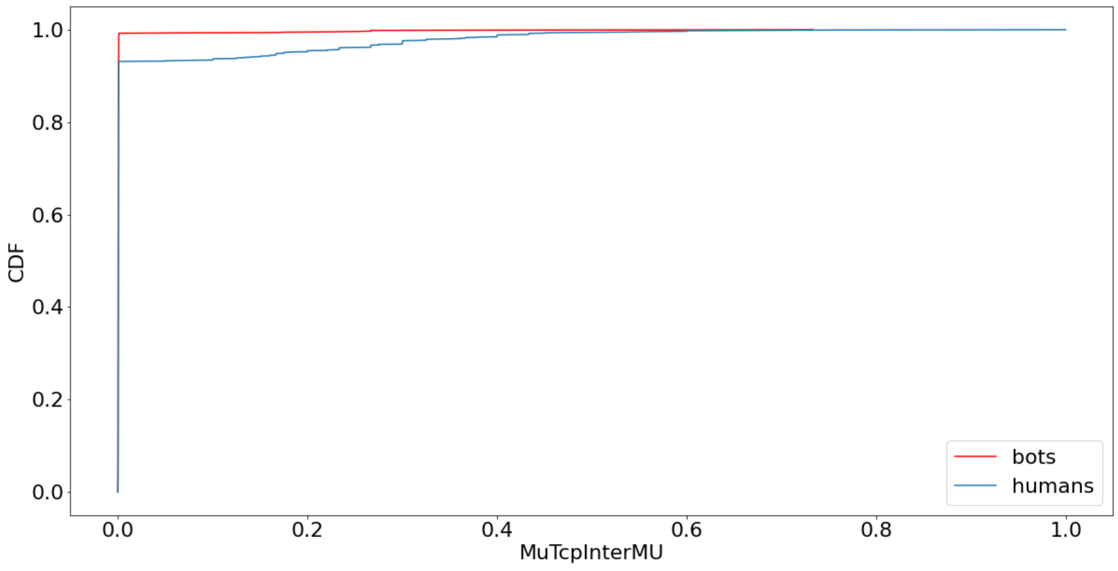
5.3.1. Preparation and Feature Selection

The first step, when developing a machine learning model, is understanding which features might be relevant for our use case and which could be discarded. Since Arensi got great results, we started with the features he used in his research (see Figure 5.11). Note that we have circled in red the features we could not use because FS can't produce them. Arensi used PCAP files and a script to extract data from those. By doing so, he could differentiate between packets of large dimension and small dimension, he could also compute time intervals between uplink and downlink and so on. We are speaking of milliseconds: in FS, with a window of one second, the downlink traffic starts always in the same window as the uplink traffic. This is the most important drawback of using FS: a more limited feature choice, on the other hand, we didn't have to build a script to compute all the features since the tool does that for us. Anyhow, we could still use almost half of the features used from Arensi, it is important to notice that the features **pack_ratio** and **bit_ratio** are not computed by default from FS, however, with the data available, we could easily compute them by ourselves. Another feature not computed by FS is **burst_t_len**, which is the time length of the interaction, we substituted that feature with the number of windows in the interaction. For example, assuming a window of 1 second, an interaction composed of 5 windows is considered 5 seconds in length. This feature is not too precise, since an interaction can end (and start) at the beginning, or in the middle, of a window, altering the real value of this feature, but it was the best indicator of the interaction's length we had. Of course, by lowering the window, this feature becomes more precise.

IDA outputs 115 features, so it is important to analyze them to understand which are relevant to our use case. Performing feature selection in classification problems is not always easy. Usually, it is necessary to consult subject matter experts, and even with their help, trial and error is often the main approach. In our case, it has been particularly difficult to spot the crucial features since the logic behind the reception of the audio, from the Echo's side, and the production of the TCP traffic is a black box, whose functioning is known only by Amazon engineers. We assumed that different voices, having each one a unique cadence, may produce differences in the uplink payload and in the inter-arrival rate. For this reason, we thought that features such as the average inter-arrival rate or the variance of the inter-arrival might be quite informative for this task. To verify this theory, we plotted the CDF for each feature of the robot class (0) and of the human class (1). By doing this, we can see if, for some features, bots produce values different from humans. If this happens, then we can probably conclude that the feature might be relevant since it shows a different behavior depending on the class.
The result of this analysis partially confirmed our hypothesis. Generally, all the features showed similar distributions between humans and robots: inter-arrival time features showed more variability for humans with respect to bots, also the bit ratio highlighted an interesting difference.

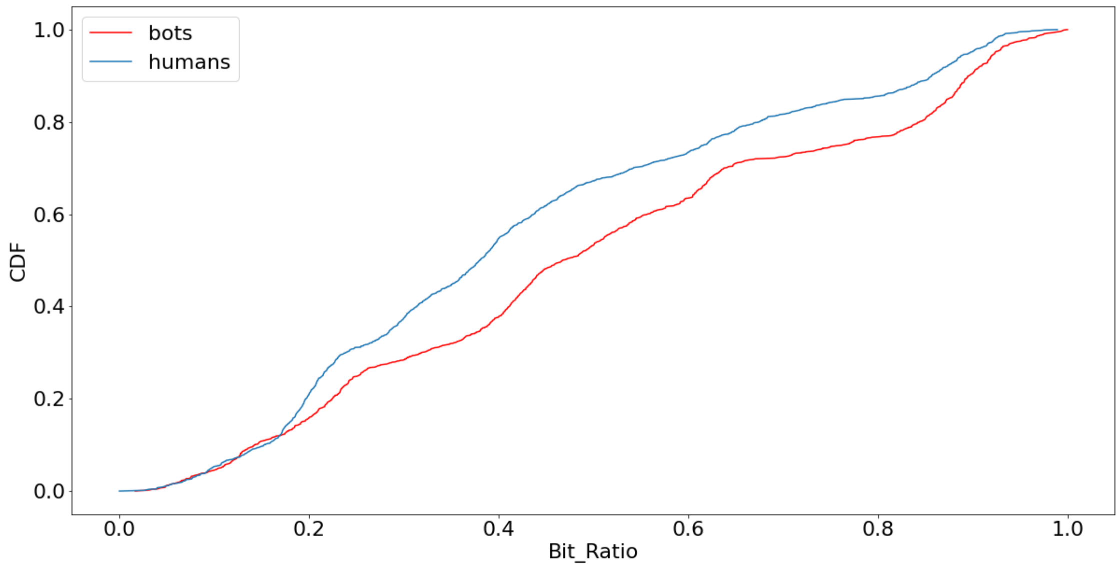
Parameter	Description
question	question asked to Alexa
name	name of the speaker of the question
sex	male or female
genre	bot or human
max_out_ip	most contacted IP address during the request
len_out_ip	number of IP addresses contacted during the request
burst_t_len	total time length of the packet transmission
pack_ratio	ratio of outgoing packets over the total num of packets
bit_ratio	ratio of outgoing packets dimension over the total dimension of packets
n_packets	length of the packet transmission stream
sum_packets	total dimension of the packet I/O exchange
mu_packets	μ of the total dimension of the packet I/O exchange
std_packets	σ of the total μ dimension of the packet I/O exchange
mu_packets_in	μ of the dimension of the incoming packets from the server
std_packets_in	σ of the μ dimension of the incoming packets from the server
mu_packets_out	μ of the dimension of the outgoing packets to the server
std_packets_out	σ of the μ dimension of the outgoing packets to the server
mu_t_s_in	μ of the inter-time among the incoming packets of small dimensions
std_t_s_in	σ of the μ inter-time among the incoming packets of small dimensions
mu_t_h_in	μ of the inter-time among the incoming packets of big dimensions
std_t_h_in	σ of the μ inter-time among the outgoing packets of small dimensions
mu_t_s_out	μ of the inter-time among the incoming packets of small dimensions
std_t_s_out	σ of the μ inter-time among the outgoing packets of small dimensions
mu_t_h_out	μ of the inter-time among the outgoing packets of big dimensions
std_t_h_out	σ of the μ inter-time among the outgoing packets of small dimensions
delta_io	Δ_{io} of the server response
delta_in	Δ_{in} of the incoming communication
delta_out	Δ_{out} of the outgoing communication

Figure 5.11: Here the list of features used in [1]



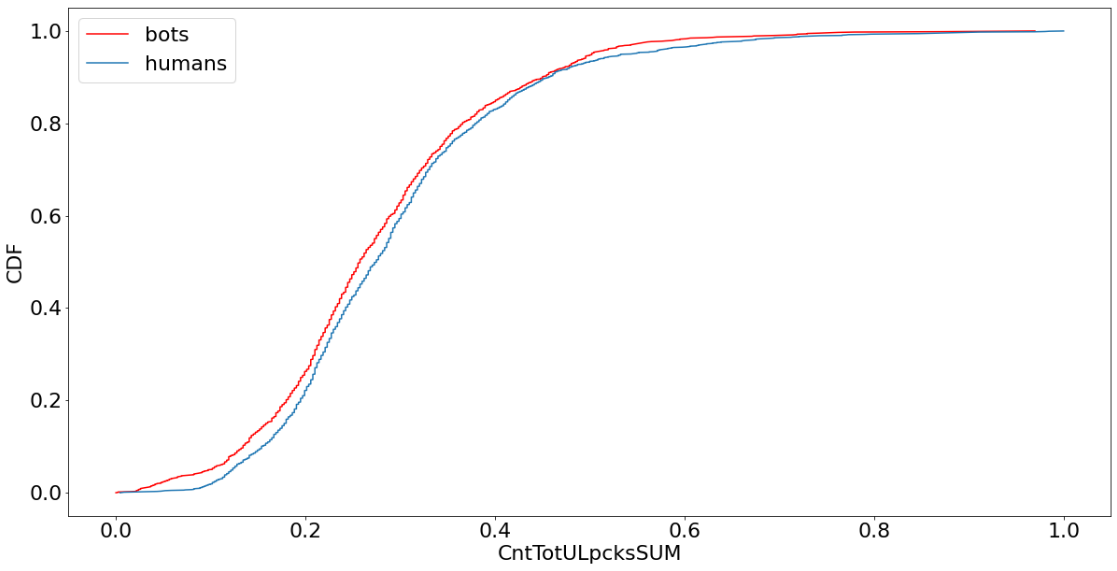
(a)

Figure 5.12: This graph shows, respectively, the distribution of values for bots and for humans of the average packets inter-arrival time. From the CDF distribution can be observed that, both for uplink and downlink, inter-arrival values of robots tend to be more constant and small while, for humans, there is a little higher variability.



(a)

Figure 5.13: This graph shows the distribution for bots and humans of the bit ratio feature. This is the feature that showed the most pronounced differences.



(a)

Figure 5.14: This graph shows the distribution for bots and humans of the number of packets sent in uplink. This feature, like many others, showed a similar distribution for both classes.

After feature analysis, we shuffled our data and produced the training set, made of 2092 interactions (around 69% of the total dataset), the validation set (16% of the dataset for a total of 523 interactions) and the test set (461 interactions). Note that the data are not evenly split between humans (57% of total interactions) and bots (43%). We have trained the final models also on a balanced set (50-50) by removing some of the human samples, however, we didn't detect any particular change in the model performances, so we preferred to work with an unbalanced but larger set of data.

5.3.2. Models Tried and Results

Regarding the bot-human classification task, we did not get amazing results; focusing on the one-second window dataset, our best model revealed to be Gradient Boosting, with 200 estimators, reaching an accuracy of around 74% and an AUROC of 0.79. We tried different feature configurations to improve the model performance: using only inter-arrival features led to a bad result (around 63-64% accuracy). We have noticed that, by adding other uplink and downlink features, the accuracy increased. In the end, the best result was obtained with all the features produced by IDA, in fact, Gradient Boosting already performs a sort of built-in feature selection using estimators and, for this reason, it may be worth having as many features as possible as long as they don't introduce wrong

information, noise or errors.
Moving now to the 0.5-second window dataset, we have not found particular differences in terms of performance, the best model for this dataset revealed to be Random Forest, with 250 estimators, with an accuracy of 73% and an AUROC value of 0.78. We would have expected a slightly better performance for the 0.5-second window dataset, since we are retrieving a higher level of detail from the traffic, but however, we observed the opposite. This is probably due to the fact that, to compute the final features, we are using statistical indicators, to average all the windows features, losing the level of detail we had gained by lowering the window size. Even for the 0.5-second dataset, all the features have been used to feed the model. We also have tried to perform a min-max normalization on the features' values, with the purpose of having the same order of magnitude for all the features; this procedure brought only a really small improvement in terms of accuracy and AUROC (we are speaking of a 1% difference in the performance of the normalized dataset versus the non-normalized one).

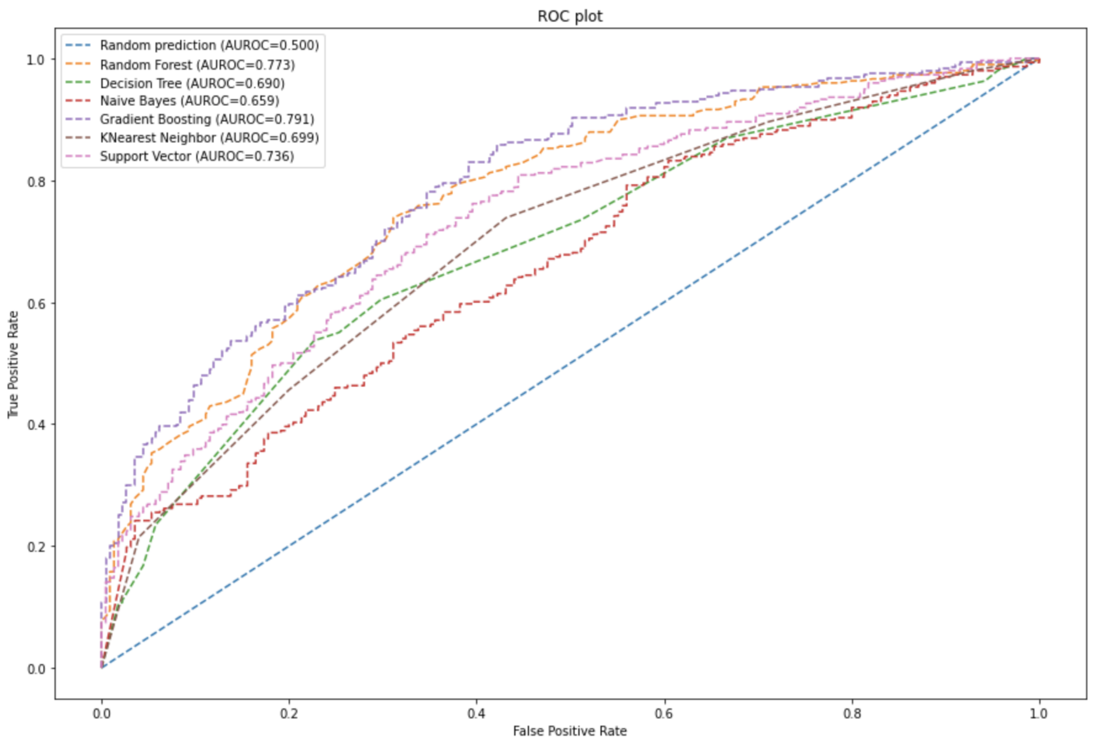


Figure 5.15: ROC curve for all the models we have tried in the classification task with the one-second dataset. The best model is Gradient Boosting, while the worst is Naive Bayes.

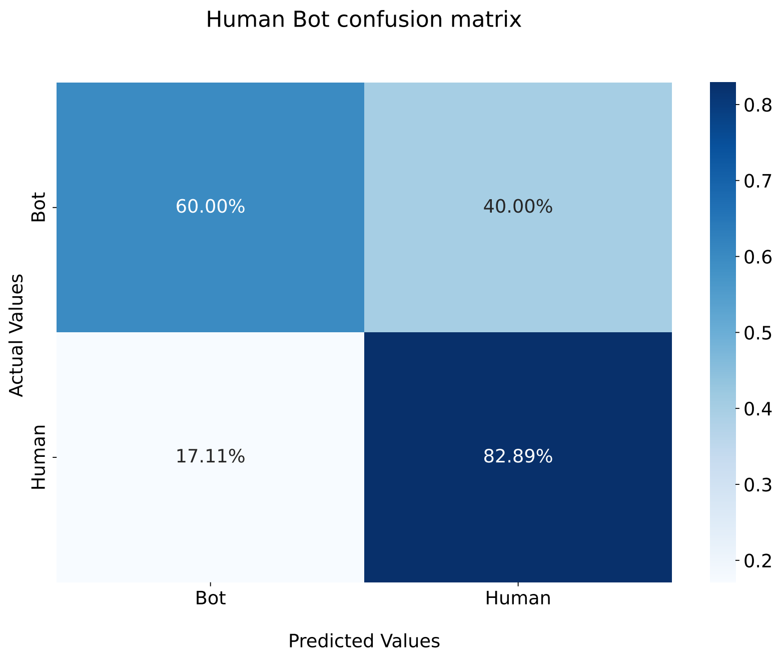


Figure 5.16: Confusion matrix of the Gradient Boosting algorithm for the one second dataset

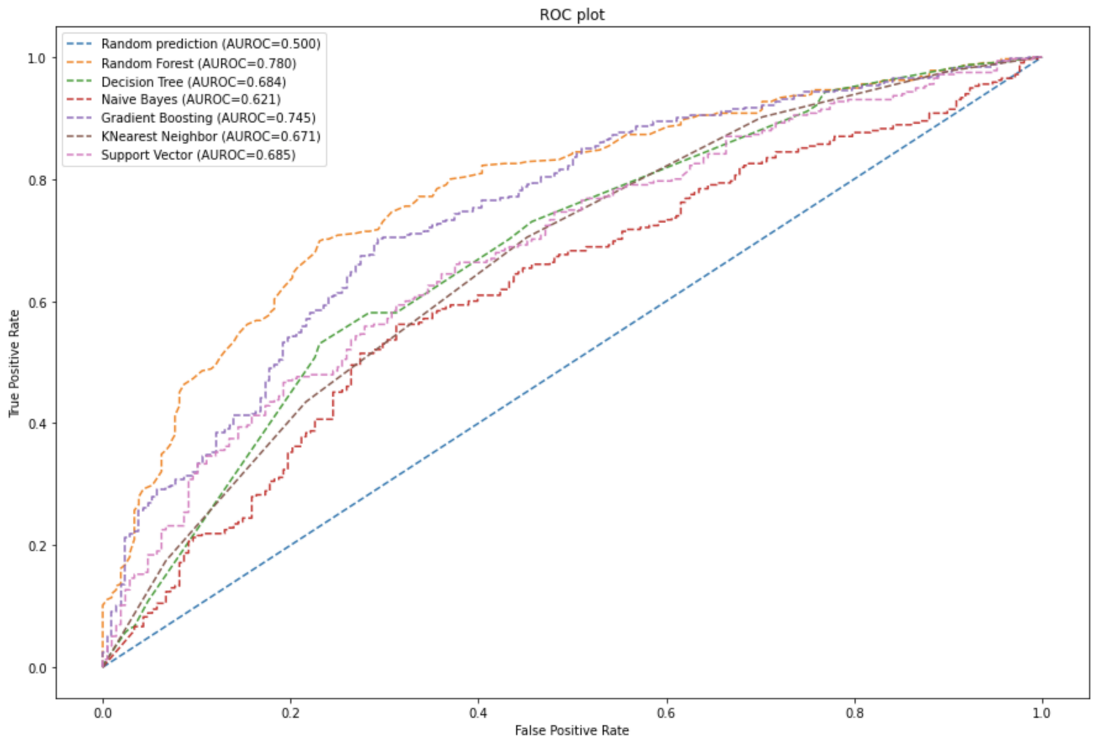


Figure 5.17: ROC curve for all the models we have tried in the classification task with the 0.5-second dataset. The best model this time is Random Forest.

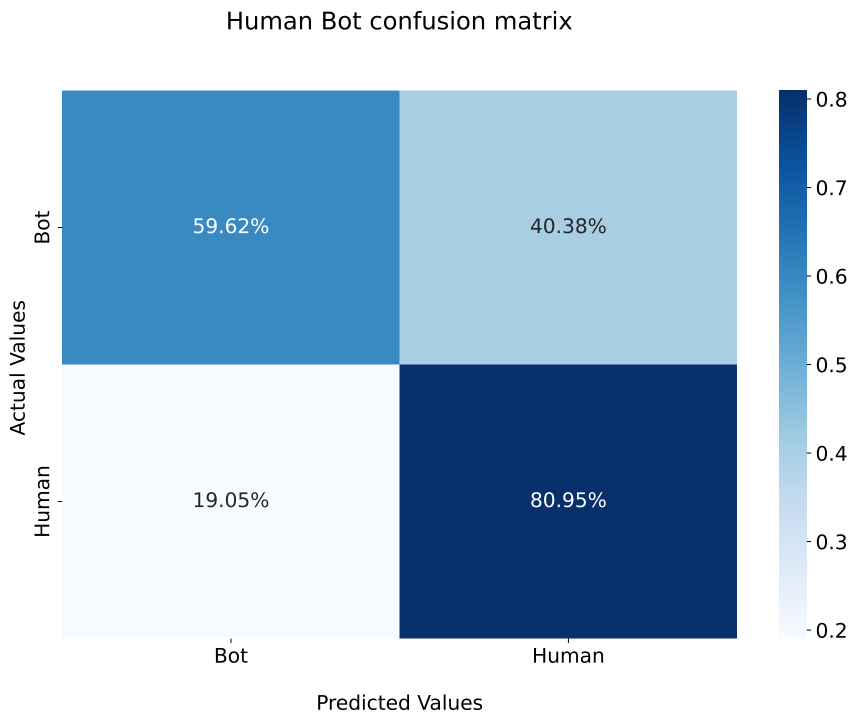


Figure 5.18: Confusion matrix of the Random Forest algorithm for the 0.5-second dataset.

Before moving on, is worth mentioning that, for sake of curiosity we have also tried to perform a classification task between men and women, by trying to predict the sex of the speaker from the produced traffic. However, as one might have supposed, we got a model which predicted the sex of the speaker with accuracy near to random guessing. We can therefore state that there are no particular differences between the traffic produced by a man's question with respect to the one produced by a woman. Of course, two people can produce different traffic traces, but no pattern distinguishes men's interactions from women's ones.

5.4. English or Italian Task

Let's now step into the language classification task. In this case, there isn't a similar research that we can use to compare our results, however, there are case studies that analyzed the possibility to classify the question made by the user (see [6]). These studies, most of the time, reported quite good results. For this reason, regarding this task, which was more related to the content of the interaction rather than to the speaker, we were expecting a better performance.

5.4.1. Preparation and Feature Selection

Regarding this classification problem, the process of feature selection and analysis has been similar to the one of the previous task. In this case, we are supposed to measure a not-so-relevant difference between inter-arrival rate data of the two languages, like in the previous analysis. Instead, we imagined finding some differences between uplink and, especially, downlink payload data. By performing the same analysis of 5.3.1 our assumptions have been confirmed. We found, for almost all the features, more relevant differences with respect to the genre classification. Below can be found the graphs of some of the features which showed the most pronounced differences.

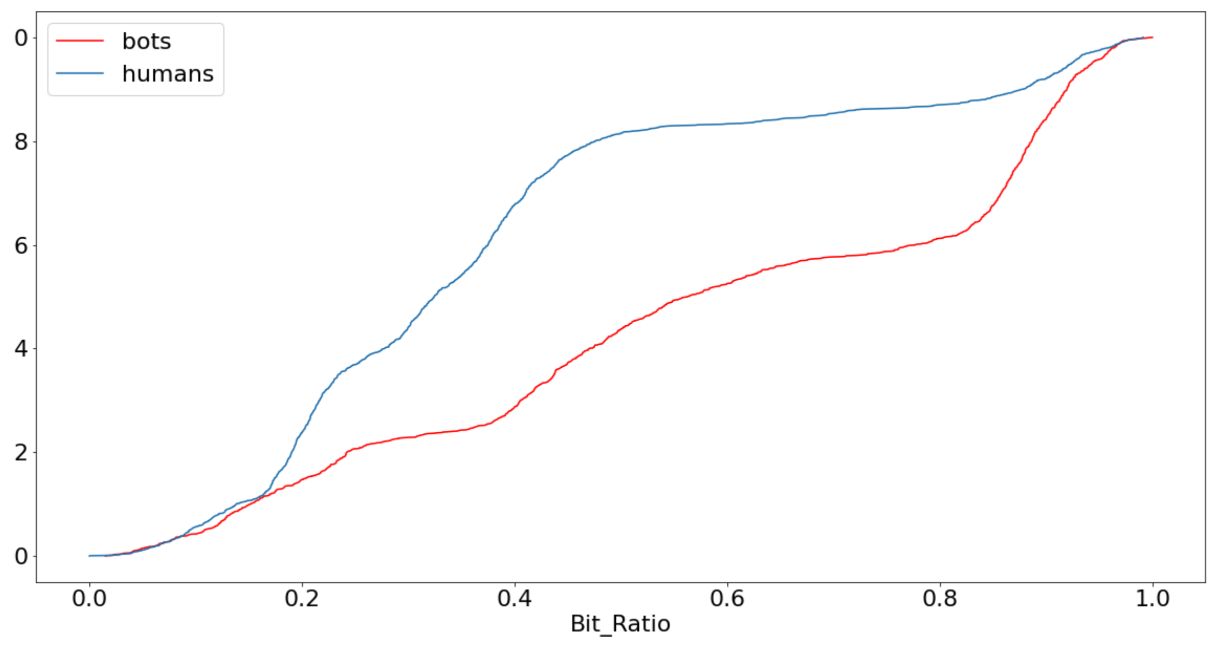


Figure 5.19: The CDF distributions of the payload size of packets received in downlink.

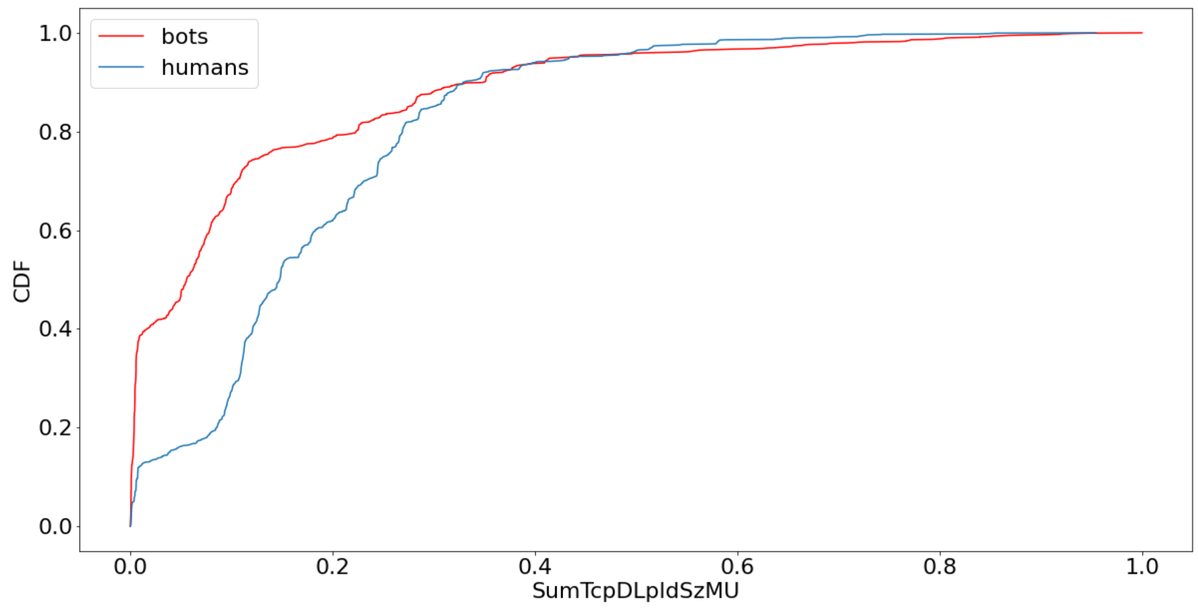


Figure 5.20: The CDF distributions of the downlink payload size.

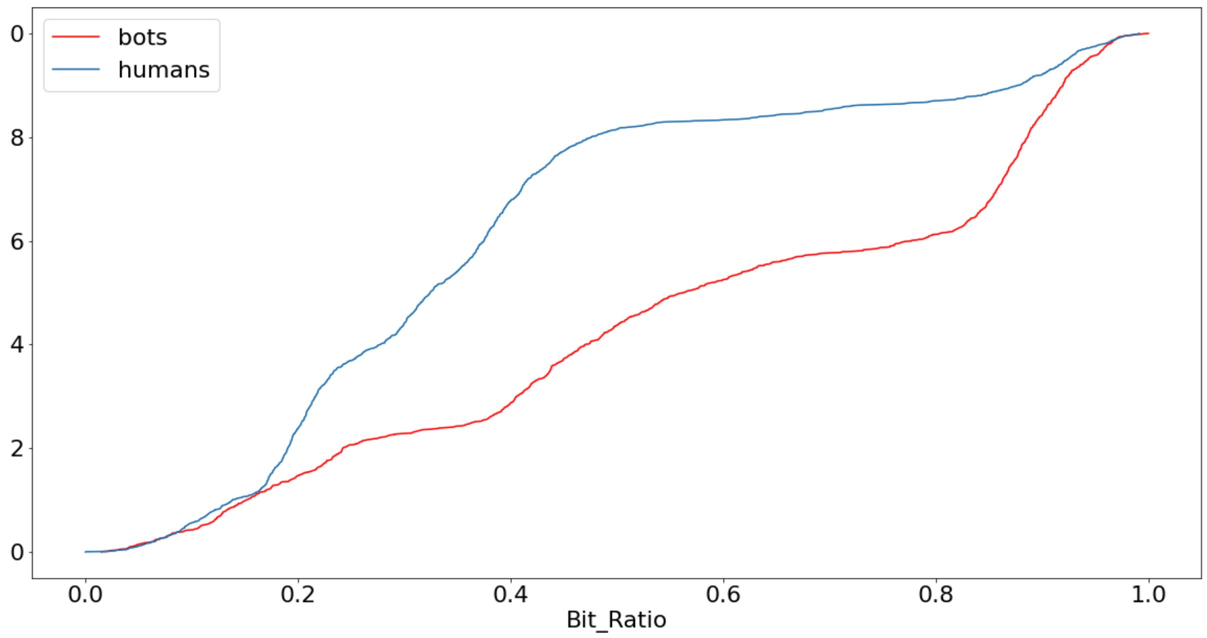


Figure 5.21: The CDF distributions of the bit ratio feature.

5.4.2. Models Tried and Results

The results obtained for this task are considerably better with respect to the human or bot classification. For the 0.5-second window dataset, our best model has been Random Forest which got an accuracy of 85% and an AUROC of 0.92. To train this model we have used a subset of the features produced by IDA, however, by using all the features, the final results do not change that much, but simply the best model becomes Gradient Boosting with the same AUROC and with a slightly lower accuracy value (0.84). Very similar results have been found for the one-second window dataset.

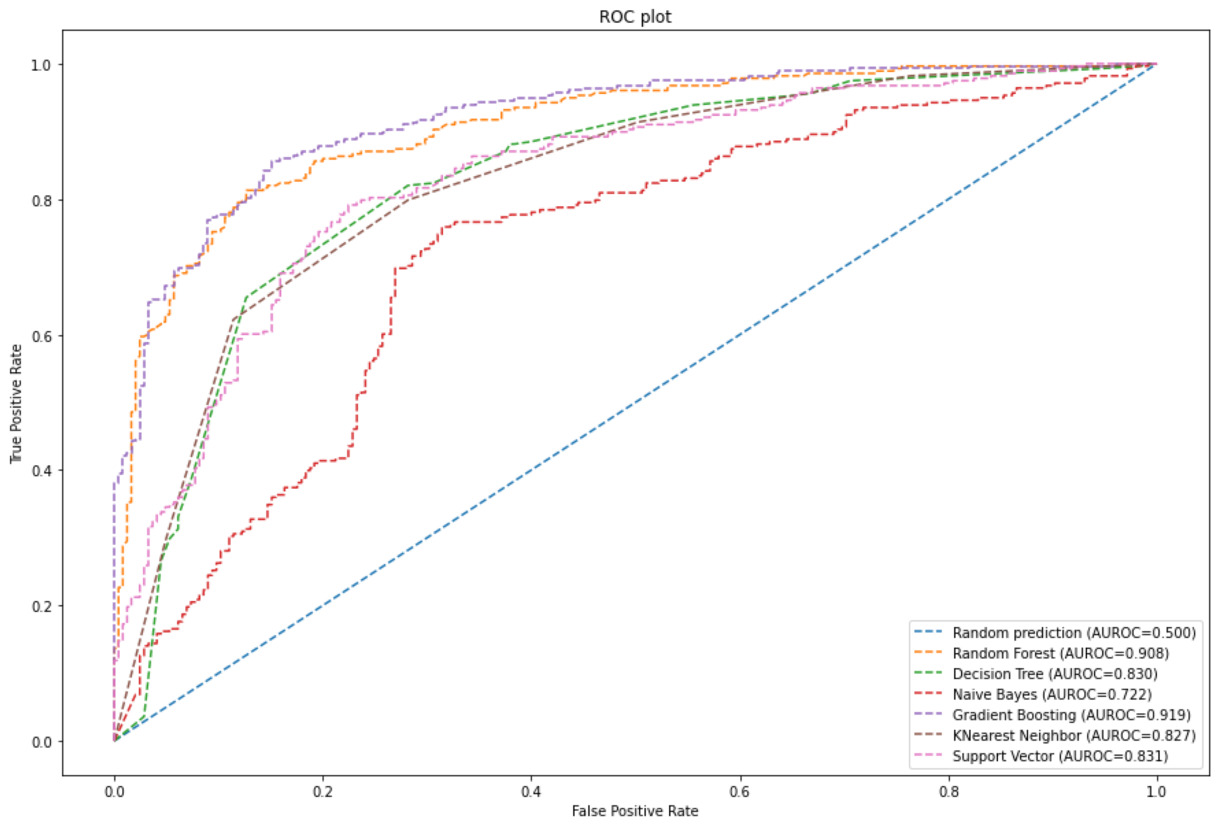


Figure 5.22: ROC curve for all the models we have tried in the classification task with the one-second dataset. The best model is Gradient Boosting, while the worst is Naive Bayes. These are the performances obtained when using all the features.

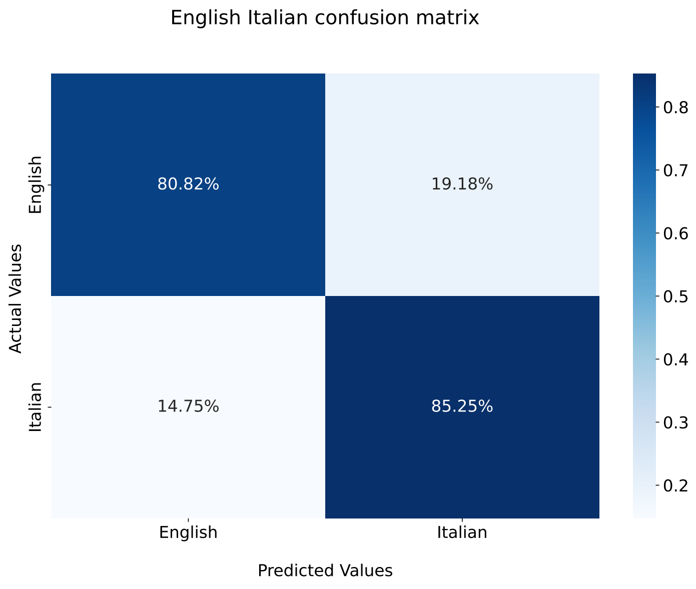


Figure 5.23: Confusion matrix of the Gradient Boosting algorithm for the one-second dataset, false negatives and positives are equally distributed in this case and represent a small percentage of the total predictions.

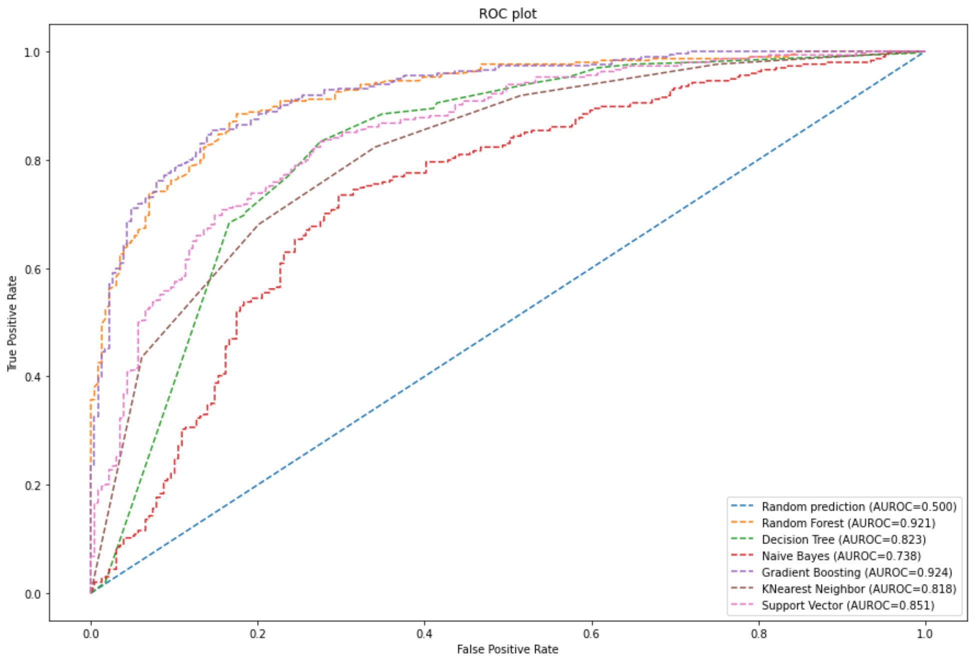


Figure 5.24: ROC curve for all the models we have tried in the classification task with the 0.5-second dataset. This performance is obtained by using a subset of all the features. The best model this time is Random Forest in terms of accuracy (85%), while, in terms of AUROC, is slightly better Gradient Boosting. The worst remains Naive Bayes.

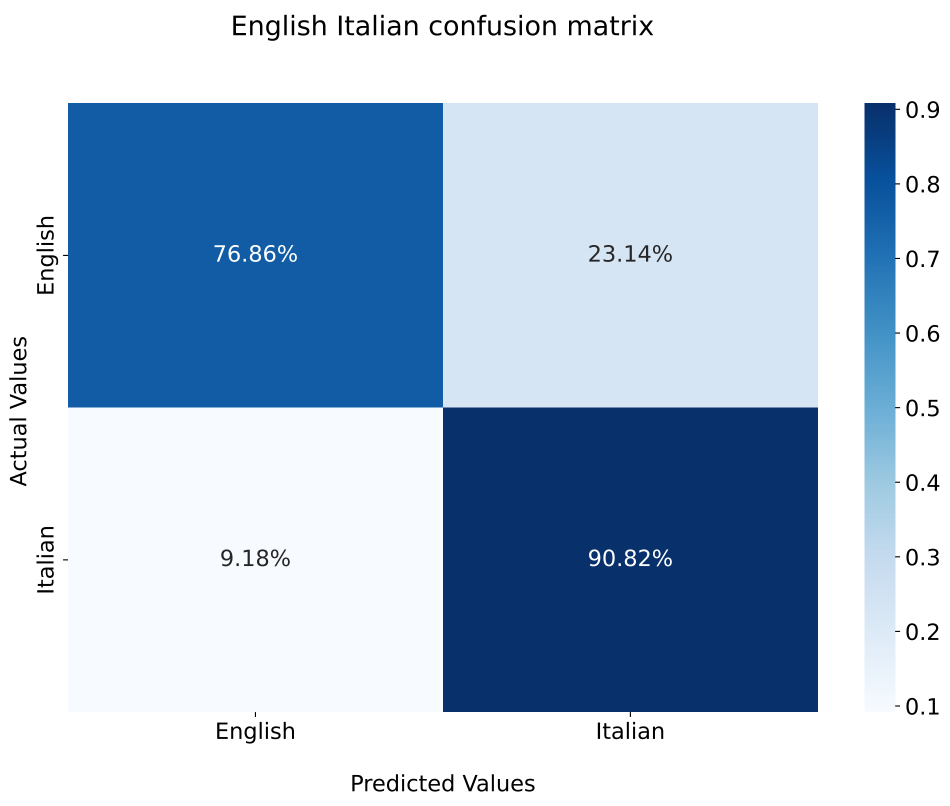


Figure 5.25: Confusion matrix of the Random Forest algorithm for the 0.5-second dataset. We have similar proportions with respect to the 1-second scenario.

5.5. Final Considerations

Before moving to the final considerations, let's recap the results we have obtained:

	Window	Best Model	Accuracy	AUROC	FP	FN
Bot-Human	1 sec	Gradient Boosting	74%	0.791	9.75%	17.21%
Bot-Human	0.5 sec	Random Forest	73%	0.78	13.58%	17.59%
Eng-Ita	1 sec	Gradient Boosting	84%	0.919	7.84%	8.99%
Eng-Ita	0.5 sec	Random Forest	85%	0.921	6.31%	9.75%
Interaction	1 sec	Random Forest Gradient Boosting	99.8%	0.999	0.49%	0.39%
Interaction	0.5 sec	Random Forest Gradient Boosting	99.8%	0.999	0.49%	0.39%
Interaction Reports	1 sec	Random Forest Gradient Boosting	98.8%	0.999	0.52%	0.49%

Table 5.1: Results of all the models tried, FP and FN stands for False Positives and False Negatives.

By having a look at the table above it's clear that the task where we obtained the worst performances is the genre detection task (Human-Bot). Anyway, the overall result is not that bad, it is far distant from random guessing proving that some information regarding this task can be extracted. However, if we compare our results with the ones obtained by Arensi [1], our AUROC value is lower than 0.11 with respect to the one he has obtained. Is this actually due to the use of Feature Sniffer? Before answering the question let's highlight the differences between our experiment and his one. First of all, his training set contained only 4 human speakers and each speaker recorded 100 questions, while our training set was composed of the voices of 31 different humans and each one asked only 60 questions. This means that our dataset, being more realistic, shows a larger variability of voices and, therefore, of voice patterns with respect to his one. We also collected voices from people of different ages and sex, while he recorded the voices of 4 people of the same age. Being our data more variable, it becomes also more difficult for a model to learn a precise schema. Another point is that we don't know if he used neural voices produced by Amazon Polly. Neural voices are so realistic that even a human could find it difficult

to classify them as robotic voices, also for this reason, our performances are worse. In the end, it is true that by using FS we couldn't use some important features, but this is not necessarily the reason why we got a lower performance. All the considerations above take part in the final result. What can be inferred from this experiment is that FS allows the extraction of pieces of information about the speaker's identity, but it might introduce some limitations if features regarding the inter-arrival time are relevant to the use case. Anyway, even if our dataset could be considered large enough, to train models on certain tasks it could be still too small. Probably, with a larger dataset with more questions per person, the performance of the model could be relatively better.
Way higher performances have been obtained instead, in the language classification task. In this case, the AUROC reaches really good values and the accuracy is good, even if not over 90%. Also in this case it's worth highlighting that we worked with data containing 40 different questions and that these questions often provide different answers from the device. This means that there is great variability in this aspect too. This makes it difficult for the model to learn a pattern between English and Italian and, for this reason, we consider the performance obtained a success. We can conclude that, regarding the capability to extract information on the content of the traffic, FS revealed to be a great tool and can be used with good results to perform IoT forensic tasks.
With respect to the interaction detection task there is not much more to say, the results are extremely promising, and combining the model obtained after FS may simplify the work of the IoT forensic/Data Scientist a lot. The choice between the model with/without reports depends mainly on the objective of the task. If time is available and precision is a key requirement, then it would be better to perform report cleaning manually, on the other hand, if we want to actuate an on-the-spot data extraction from a sniffed flow of packets, or if we don't want to lose time because precision is not so essential, the second model is the one to go for. Of course, the difference between the two models is minimal (1% accuracy on the final prediction).

6 | Conclusions and Future Developments

We have come to the end, in these pages, a practical application of Feature Sniffer has been explored and optimized. We highlighted that FS can be used with success to perform machine learning on an encrypted traffic exchange, even though it presents some limitations. In particular, FS use is more promising when related to the analysis of the traffic exchange content rather than the speaker's identity. The major advantages of this tool are the possibility to extract in a quick and on-demand way the features necessary to perform machine learning, the reduced size of the files produced (CSV files are generally way lighter than PCAP files), and the capability to perform live feature extraction. In addition, we have developed a model that can be integrated with FS to automate the creation of a clean and precise dataset of interactions with an Amazon speaker. Thanks to this model, future research on Amazon Echo devices' traffic could be focused only on the machine learning part, ignoring the training dataset construction and feature extraction problems. A future development that we, unfortunately, didn't have the time to explore is the comparison between the traffic produced by Alexa with the ones produced by other smart speakers like Google Home or Siri (Apple). It could be really interesting to understand if our interaction detection model might be applied, with the same success, to the traffic produced by other vocal assistants, or if it should be necessary to train another model. Also, it would be interesting to perform again our experiment, but with a larger/smaller window size, to understand which is the optimal length for each task. Other features might be added in the future to FS, like, for example, the time delta between the first uplink packet and the first downlink one. Finally, in the language and genre detection tasks, we averaged the different windows to produce the interactions feature's values: it might be worth exploring how the results would have been changed if, instead, in the training data all the windows' values were inserted one by one, so if an interaction is composed of 3 windows, we have 3 different values of inter-arrival time, one for each window. In the end, there are tons of possible developments in this field of research and we hope that our work will pave the way for them.

"*If a machine is expected to be infallible, it cannot also be intelligent*"
Alan Turing

Bibliography

[1] A. Arensi. Machine learning classification of amazon echo dot users via encrypted voice traffic. 2021.

[2] AWS. Amazon polly documentation, 2023. URL `https://docs.aws.amazon.com/polly/latest/dg/what-is.html`.

[3] V. Cortez. Understanding the roc curve, 2020. URL `https://towardsdatascience.com/understanding-the-roc-curve-in-three-visual-steps-795b1399481c`.

[4] J. Howarth. 80+ amazing iot statistics, 2022. URL `https://explodingtopics.com/blog/iot-stats`.

[5] T. Kalin, K. Stone, and T. Camp. Amaze: Recognizing speakers with amazon's echo dot device. In *2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 494–503. IEEE, 2019.

[6] S. Kennedy, H. Li, C. Wang, H. Liu, B. Wang, and W. Sun. I can hear your alexa: Voice command fingerprinting on smart home speakers. In *2019 IEEE Conference on Communications and Network Security (CNS)*, pages 232–240. IEEE, 2019.

[7] F. Loh, F. Wamser, C. Moldovan, B. Zeidler, D. Tsilimantos, S. Valentin, and T. Hoßfeld. Is the uplink enough? estimating video stalls from encrypted network traffic. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2020.

[8] F. Palmese, A. E. Redondi, and M. Cesana. Feature-sniffer: Enabling iot forensics in openwrt based wi-fi access points. *arXiv preprint arXiv:2302.06991*, 2023.

[9] A. Pimpinella, A. E. Redondi, F. Loh, and M. Seufert. Machine-learning based prediction of next http request arrival time in adaptive video streaming. In *2021 17th International Conference on Network and Service Management (CNSM)*, pages 558–564. IEEE, 2021.

[10] V. Sciancalepore, K. Samdanis, X. Costa-Perez, D. Bega, M. Gramaglia, and A. Banchs. Mobile traffic forecasting for maximizing 5g network slicing resource uti-

lization. In *IEEE INFOCOM 2017-IEEE conference on computer communications*, pages 1–9. IEEE, 2017.

[11] M. Seufert, P. Casas, N. Wehner, L. Gang, and K. Li. Stream-based machine learning for real-time qoe analysis of encrypted video streaming traffic. In *2019 22nd Conference on innovation in clouds, internet and networks and workshops (ICIN)*, pages 76–81. IEEE, 2019.

[12] M. Stoyanova, Y. Nikoloudakis, S. Panagiotakis, E. Pallis, and E. K. Markakis. A survey on the internet of things (iot) forensics: challenges, approaches, and open issues. *IEEE Communications Surveys & Tutorials*, 22(2):1191–1221, 2020.

[13] C. Wang, S. Kennedy, H. Li, K. Hudson, G. Atluri, X. Wei, W. Sun, and B. Wang. Fingerprinting encrypted voice traffic on smart speakers with deep learning. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 254–265, 2020.

List of Figures

List of Tables

List of Abbreviations

- **IoT** Internet of Things
- **ML** Machine Learning
- **TTS** text-to-speech
- **API** Application Programming Interface
- **TCP** Transmission Control Protocol
- **TP** True Positive
- **FP** False Positive
- **TN** True Negative
- **FN** False Negative
- **ROC** Receiving Operator Characteristic
- **AUROC** Area Under the ROC
- **CDF** Cumulative Distribution Function
- **IP** Internet Protocol
- **MAC** Media Access Control
- **GB** Gradient Boosting
- **RF** Random Forest
- **K-NN** K - Nearest Neighbors
- **IDA** Interaction Detection Algorithm
- **FS** Feature Sniffer

Acknowledgements

Dire grazie non è mai stato il mio forte, ma in questo caso è giusto dedicare qualche riga a chi mi è stato vicino e mi ha accompagnato in questi anni di studi.
Prima di tutto è d'obbligo ringraziare le persone senza le quali questo lavoro non sarebbe stato possibile, il professore Redondi ed il professore Cesana, che mi hanno permesso di lavorare su una tematica così interessante, un ringraziamento sentito va anche all'ingegnere Fabio Palmese che, con la sua infinita disponibilità e gentilezza, è venuto incontro ai miei terribili orari di studente lavoratore. Impossibile non menzionare anche tutte le persone che hanno gentilmente prestato il loro tempo e la loro voce per la creazione del dataset.
Infine, un pensiero va ai miei amici ed alla mia famiglia che mi hanno sempre supportato e sopportato ed, a Caterina, la cui compagnia ha illuminato le mie giornate di questi ultimi quattro anni.