Politecnico di Torino

Master of Science in Communications and Computer Networks Engineering

Master's Degree Thesis

Estimating QoE from QoS in real-time traffic: A Machine Learning approach

Supervisor:
Prof. Martino Trevisan

Co-Supervisors:
Prof. Michela Meo
Dena Markudova

Candidate:
Ymer Gurra

DECEMBER 2021

Table of Contents

List of Figures

List of Tables

Acknowledgements

First of all, I would like to express my deepest graditute to my suppervizor **Prof**. Martino Trevisan, as well as co-supervizors **Prof**. Michela Meo and Dena Markudova, for their precious guidance and support throughout the whole process. All of them have been very friendly and ready to asnwer every question of mine, without any hesitation or exhaustion. I have benefitted a lot from their extraordinary experiences and knowledge so that I am pretty sure that what I have learnt from them during this period will help me a lot in my carrier in the future.

My sincere thanks also goes to all other members of Smartdata@PoliTO research group, for their valuable ideas and collaboration during the weekly meetings.

Last but not least, I want to thank my family for their unconditional support and loving, by encouraging me in every step of my Master's degree, from the very beginning, until the end of it.

Chapter 1

Introduction

Real-Time traffic has been continuously growing and it has gained an important role in people's life. With the recent Covid-19 situation and the rise of working from home and the remote social interactions, RTC traffic has reached its peak so far and RTC-based applications such as WebRTC, WebEx, Jitsi etc., has become vital nowadays. For this reasons, it is extremely important for ISP providers and network operators to guarantee Quality of Experience for end users in all network conditions.
Unfortunately, because end-to-end encryption is becoming more common, as a result of increased video streaming content over HTTPS and QUIC, ISPs cannot directly observe video quality metrics such as startup delay, and video resolution from the video streaming protocol. The end-to-end encryption of the video streams thus presents ISPs with the challenge of inferring video quality metrics solely from properties of the network traffic that are directly observable [14].

Today, all useful QoE-aware mechanisms for traffic management, app delivery adaptation, and user experience problem diagnosis are stymied by a basic limitation: determining a user's QoE for a particular app requires software on the user's device that is capable of measuring QoE metrics for that app, and reports information to the entity (OS, ISP, or application server) implementing the QoE-aware mechanism. This limitation stems from several reasons:

a. It is impossible to write one software, which if installed on a user's device, can measure the user's QoE for any arbitrary app.
b. A lack of API to communicate QoE. There typically does not exist an interface via which an app's client software can relay measured QoE information to other entities that can make use of this information, such as the user's OS or ISP.
c. Third-party clients. It can also be challenging for an app's own servers to discover user-perceived QoE because users often access apps via client software not developed by the app provider [13].

Because of those limitations, the only possible solutions for gauging the QoE of video streaming is to rely on network level features obtained from the encrypted video traffic traces, or from independent network measurement tools executed outside the video application data plane.
Here is the point where this thesis mainly focuses on, considering the QoS values observable in clear by network devices, we try to map those values into QoE metrics which's knowledge is quite valuable in ISP providers and Network operators point of view. Learning and predicting Quality of Experience as a valuable metric for flow and network control operations is crucial for ISPs. For instance, they can plan upgrades of part of the network if users from a specific area continuously

report low QoE, they can update the path of certain flows with lower QoE, or apply different policies at the routing-scheduling levels [16].
To summarize, the methodology which we use to reach the goal of this thesis, which is building a model able to map the QoS metrics into QoE, is the one via a Machine Learning approach. As we will also see in the following sections, we have some ground truth values (representing some of the QoE metrics which we aim to predict) collected via application logs and a various fields of dataset, collected via network sniffing tools such as Wireshark, which will be used to train the model via different ML algorithms. Once we train the model, we test it through our testing dataset and evaluate its performance. So basically, we predict QoE metrics from QoS features since these last ones, unlike QoE metrics, are objective and measurable in the vantage points of the network (i.e. Router devices etc.).

1.1 Thesis Objective

This thesis was developed as part of the collaboration between the research group "SmartData@Polito" and Cisco Systems Inc. This partnership about this project, aims to close the gap between QoS and QoE, and creating to network operators the possibility to improve the network performance according to the quality of experience perceived at the end user side.
The knowledge of QoE metrics by the network providers is extremely important because they can accordingly allocate various resources in case there is a network issue. However, it is a widely known truth that not every problem is networking related, where it may be application problems as well, and as a consequence the knowledge of QoE metrics helps in quick troubleshooting of the problem. Saying that, the main objective of this thesis is to define these crucial QoE metrics from the network traffic analysis and develop some approaches that could close this gap between the two. Building a predictive model consists of many steps and the path being followed is like this:

a. **Problem definition**: First of all, we must define a problem so that we can look for possible solutions. This step answers the following questions: "What is the scenario which the data are collected from?", "What kind of problems will this approach solve?", "What is the proper methodology to be followed up?, etc". As understood, this step is problem specific and it is focused on possible solutions of the introduced problem.

b. **Data collection**: After the problem definition and methodology description, the data collection process and tools used for this purpose need to be defined. A clear overview on the data partitioning and their role on the defined model, need to be clarified.

c. **Data analyzation and cleaning:** This step focuses on a detailed analyzation of the data to be used and a correct pre-process of the them before being used for the

experimentation. It's well-known that data are messy and they have various problems such as containing outliers, missing data etc. Therefore, it's crucial to analyze them and take the necessary precautions by handling them correctly.

d. **Model construction:** From now on, we start to develop the approaches and create the model which could solve the problem defined at the beginning. Here it's the step where we decide the fields to feed up the model and the outputs we need to get.

e. **Application of the Machine Learning algorithms and techniques**: After all the analyzations and the theory part held so far, from now on it's time to put them into practice. This step focuses on a numerous number of experiments done by combining various algorithms and techniques and finally evaluate the model's performance for each of them. So basically, we can say that this part is results-oriented.

1.2 Thesis Outline

In this section we give a short overview on how the thesis is organized and structured.

Chapter 1, also the chapter we are actually in, clarifies the goal of this study, its usage in real-life etc. In this chapter, we introduce the problem and possible solutions to it. Basically, this is the introduction part of this study where we become familiar with the objectives of this work.

Chapter 2, also called the background chapter, introduces the main theoretical concepts necessary to understand the problem formulation and all the steps towards the solution we are providing.

Chapter 3, which treats the literature review section, provides a brief overview on most important researches done so far in the related field. It presents various methodologies studied by different researchers and a discussion which one fits to our problem and which not.

Chapter 4, is the part where the problem introduction and methodology used are explained in details. It gives a clear scenario about the dataset types, collection methods etc. It also provides an explanation of some concepts regarding some important additional methods applied for model performance improvement.

Chapter 5, is the chapter where the results obtained after all the experimentation process regarding the model's performance are present. Together with Chapter 4, they are the most important parts of this thesis since they state the whole process followed from the beginning until the solution of the problem.

Chapter 6, here we test the applicability of this model to other scenarios, more precisely to a cloud gaming scenario. Throughout this part we give a brief introduction about the type of dataset being used and then we present and comment the results obtained from a various number of experiments.

Chapter 7, it concludes our thesis with some important summarization and gives some important indications about possible future works.

Chapter 2

Background

This section aims at providing a brief explanation on basic concepts and clarifying the most important topics, strongly related to this study, such as RTC, RTP, QoS & QoE etc.

2.1. RTC

Real-time communications (RTC) is a term used to refer to any live telecommunications in which all users can exchange information instantly or with negligible latency or transmission delays. RTC data and messages, unlike applications such as email and voicemail which always involve some form of data storage between the source and the destination, are not stored between transmission and reception. RTC is generally a peer-to-peer, rather than broadcasting or multicasting, transmission. Real-time communications can take place in half-duplex (where Data transmission can happen in both directions on a single carrier or circuit but not at the same time) or full-duplex (where data transmission can occur in both directions simultaneously on a single carrier or circuits).

2.1.1. Importance and Evolution of RTC

In 1915, the debut of a new transcontinental telephone line meant that, for the first time in history, users separated by more than 3,000 miles could hold an interactive conversation as though they were in the same room. Later in the 20th century, high-speed internet, mobile telephony and smart devices further revolutionized real-time communications, enabling instant messaging (IM), Internet Protocol (IP) telephony, video calling, video conferencing and more.

The COVID-19 global pandemic dramatically underscored the importance of modern RTC tools, which enabled many organizations to remain functional and productive even while accommodating unanticipated and potentially long-term work-from-home requirements [8][9].

2.1.2. RTC Examples

Real-time communication tools and applications are many and varied, ranging from old-school telephony to cloud communications services. They include the following: Fixed Line telephony, VoIP, teleconferencing, video calling, automatic-live meeting transcription etc.

On the other hand, some of the RTC products vendors are: Cisco Webex, Zoom meeting, Jitsi etc.

2.1.3. WebRTC

WebRTC is an open-source standard which provides real-time communication capabilities to your application that works on top of an open standard. It supports video, voice, and generic data to be sent between peers, allowing developers to build powerful voice and video communication solutions. The technology is available on all modern browsers as well as on native clients for all major platforms. The technologies behind WebRTC are implemented as an open web standard and available as regular JavaScript APIs in all major browsers. The WebRTC is supported by Apple, Google, Microsoft and Mozilla, amongst others [26].

As anticipated above, major components of WebRTC include several JavaScript APIs:

- **getUserMedia** acquires the audio and video media (e.g., by accessing a device's camera and microphone)
- **RTCPeerConnection** enables audio and video communication between peers. It performs signal processing, codec handling, peer-to-peer communication, security, and bandwidth management.
- **RTCDataChannel** allows bidirectional communication of arbitrary data between peers. It uses the same API as WebSockets and has very low latency.

Although initially developed for web browsers, WebRTC has applications for non-browser devices, including mobile platforms and **IoT** devices. Examples include browser-based VoIP telephony, also called cloud phones or web phones, which allow calls to be made and received from within a web browser, replacing the requirement to download and install a softphone [27].

2.2. RTP Basics

As it is widely known, TCP which is a reliable protocol cannot be used in real-time communications due to its slowness (caused by overhead) and unavailability in multicast and broadcast communications. For this reason, automatically UDP is the most suitable alternative for real-time multicast traffic, which doesn't have TCP's limitations since it's connectionless protocol (out of order packets) and low overhead protocol. However, there is a huge limitation of UDP called unreliability, since it does not compensate for packet losses and jitter of the packets, neither flow nor congestion control etc. This is the case where RTP comes on the scene and becomes crucial for such kind of data traffic.

RTP is placed between layer-4 and layer-5 of OSI/ISO Model and is not an independent protocol since it is used on top of UDP. It is composed of 2 different protocols:

RTP: deals with multimedia data transport (even ports of UDP)

RTCP: Provides control and monitoring services (odd ports of UDP)

Real-Time Transport Protocols have the following available functions such as payload identifications, sequence number management, timestamp management, monitoring and

performance analysis, and participant identifications. RTP provides sequence numbers in the RTP header to detect out-of-order delivery. However, it does not provide QoS support and does not guarantee in order delivery [1].

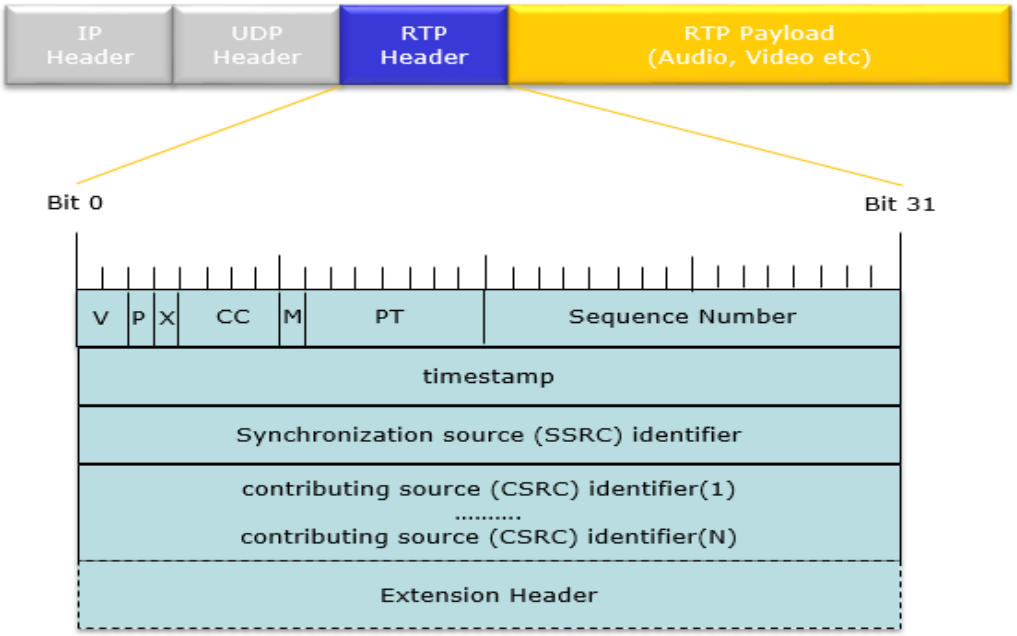


Figure 2.1: *RTP Header*

From the **figure 2.2** we can see the fields which are part of the RTP header which's meaning are as below:

- **Version (V):** it reports the version of RTP (2 bits)
- **Padding (P)**: it indicates if there is padding at the end of the RTP packet not included in the payload (1 bit)
- **Extension (X)**: it indicates if there is an **extension header** between the header and the payload of the RTP packet (1 bit);
- **CC (CSRC Count)**: Indicates the number of CSRC identifiers that follow the fixed header.
- **M (Marker)**: This is the field intended for marking a special event (e.g, frame boundary). The exact meaning of this marker varies with a profile.
- **Payload Type (PT)**: it identifies the format of the RTP payload and determines its interpretation by the application (7 bits);
- **Sequence Number:** it is used by the receiver to detect packet loss and to reorder packets. It is incremented by one for each RTP packet sent (16 bits);
- **Timestamp**: timing information used to allow the receiver to synchronize and reproduce the received media flow (32 bits);
- **Synchronization Source Identifier (SSRC)**: it uniquely identifies the synchronization source within the RTP session (32 bits). SSRC is not the source IP address but it has

meaning only in the APP layer so the further the packet goes, from different coding schemes it passes, it is identified from this field in that particular coding scheme.

- **Contributing Source Identifier (CSRC)**: list of the contributing sources for the payload contained in this packet (from 0 to 15 items, 32 bits each) [2]. So, it's an array of 0 to 15 CSRC elements identifying the contributing sources for the payload contained in a particular packet. The number of identifiers is given by the CC field. If there are more than 15 contributing sources, only 15 may be identified. CSRC identifiers are inserted by mixers, using the SSRC identifiers of contributing sources.

On the other hand, RTCP uses ACKs, not for checking if a packet is delivered or not, but to get feedback about how it is going. So every user is sending periodic reports (so **called sender and receiver reports**) which includes TX and RX statistics. The RTCP traffic must be limited (i.e. to 5% of whole data) and only 25% of it is used for **SENDER REPORT** whereas the remaining 75% of it is for RX report. The period of transmitting RTCP packets for the transmitter is given by the formula below:

$$T_{SR} = \frac{Num_{senders}}{0.25 * 0.05 * Session_{rate}} * RTCP_{pkt_{size}} \quad (1)$$

The formula in (1) above defines how frequently a sender report should be sent, and it strongly depends on the number of senders. So, the higher the number of senders, the less frequently it is sent (note that the formula is in terms of period which is the opposite of the frequency). So basically, RTP tries to overcome the limitations of UDP and make the communication as reliable as TCP and as fast as UDP.

2.3. QoS & QoE

Quality of Service (QoS) refers to the capability of a network to provide better service to some flows over some others. It refers to traffic prioritization and resource reservation control mechanisms rather than the achieved service quality. So, it is a tool which helps in managing the packets loss, delay and jitter on the network infrastructure by marking different flows based on the priority of each of them. QoS has a various number of definitions where Cisco defines it as *"the capability of a network to provide better service to selected network traffic over various technologies, including Frame Relay, Asynchronous Transfer Mode (ATM), Ethernet and 802.1 networks, SONET, and IP-routed networks that may use any or all of these underlying technologies"*. On the other hand, the International Telecommunication Union (ITU) defines QoS

as *"Totality of characteristics of a telecommunications service that bear on its ability to satisfy stated and implied needs of the user of the service"*.
As a common conclusion of those definitions, we can say that fundamentally QoS aims at providing better service to some flows with respect to some others. QoS uses the following tools to reach its goals:

1. Classification: Identify and mark the flows accordingly
2. Congestion Management: Queue and treat the flows in priority
3. Congestion Avoidance: Prevents a queue by filling up
4. Shaping/Policing: Limit the bandwidth a flow uses, etc.

Regarding the QoS measurement, it is done in the network devices by exploiting the DPI (Deep Packet Inspection). So by using the metrics such as delay, jitter, packet loss etc., via packets' header inspection, network devices can quantitatively measure the quality of service (QoS).- However, the limitation of QoS remains the strong cryptography used between packets transfer. Nowadays, the widely usage of protocols such as SSL, VPNs etc., have made it nearly impossible the inspection of packets, travelling around the network, by third parties.

Quality of Experience (QoE) on the other hand, is used to measure customer needs and try to provide services according to their needs. QoE is users' evaluation of the achieved data about the network and services provided by the network, so it's a subjective evaluation. Similar to QoS, QoE can be defined in different ways where the most important one is the one made by ITU-T which defines it as "*the overall acceptability of an application or service, as perceived subjectively by the end-user*". In a more general perspective "*QoE is defined as a measurement of customer satisfaction or customer performance dependent on objective or subjective measure of using any service or product*" [7].

QoE aims at taking into consideration every factor that contributes to a user's perceived quality of a system or service, including system, human and contextual factors. Saying that, in order to measure QoE, human ratings can be used. MOS (Mean Opinion Score) is a widely used technique for QoE measurement, relating to a specific media type. MOS evaluates the quality of a system or service by defining some levels (generally from 0 which represent the worst case, up to 5 representing the best case). Although originally Mean Opinion Scores were derived from surveys of observers, today a MOS is often produced by an *Objective Measurement Method* approximating a human evaluation.
After having introduced both concepts, now we will have a brief comparison between two. Even though QoE and QoS might seem as the same thing or very similar with each other, actually they differ from each other by a lot and this can be obviously spotted out during a video-conference. The video transmission might technically hit quality of service level agreement minimums over the span of the call, but the overall experience may be not the at the desired level.
QoS focuses more on network characteristics such as latency, jitter, packet loss etc., so basically it focuses on Network parameters point of view without caring about the end user experience.

Whereas QoE looks at the impact of the network behavior on the end user (the end user's overall happiness or frustration with the network service experience). Another important difference between two is that QoS is measured on the devices operating in vantage points of the network such as routers, by exploiting the packets' headers which are in clear. On the other hand, QoE is measured on the end devices of the network using methods explained in the previous paragraph. Therefore, QoE is a subjective metric whereas QoS is objective.
As a conclusion, the limitation of QoS is that it doesn't account for the relationship between the end user and the technology. QoS, alone, simply cannot factor for end user satisfaction, including the effect of large audio or video level variations.

2.4 Useful Tools

In this section, we introduce some of the widely used tools in this research such as Retina, Wireshark and Jupyter Notebook.

2.4.1 Retina

Retina which stands for "Real-Time Analyzer", is a command-line tool exclusively written in Python language and it is developed by SmartData@PoliTO research group of Politecnico Di Torino, group in which I am pursuing this thesis research. As the name clearly indicates, Retina produces a rich log of statistics on observed streams. It is highly configurable and gives the possibility to choose the types of statistics to output, the desired temporal aggregation (ex. per-second statistics) as well as many other parameters. If the packet capture comes along with application logs, it can match the data and enrich its output with application and QoE-related statistics. Retina uses multiprocessing if given more than one captures to process. Retina helps not only in putting the data in a very structured way but also in plotting those statistics to make a better analyzation. Figure 2.2 gives us a clear picture of how Retina works, what are the inputs it takes and the outputs it produces:

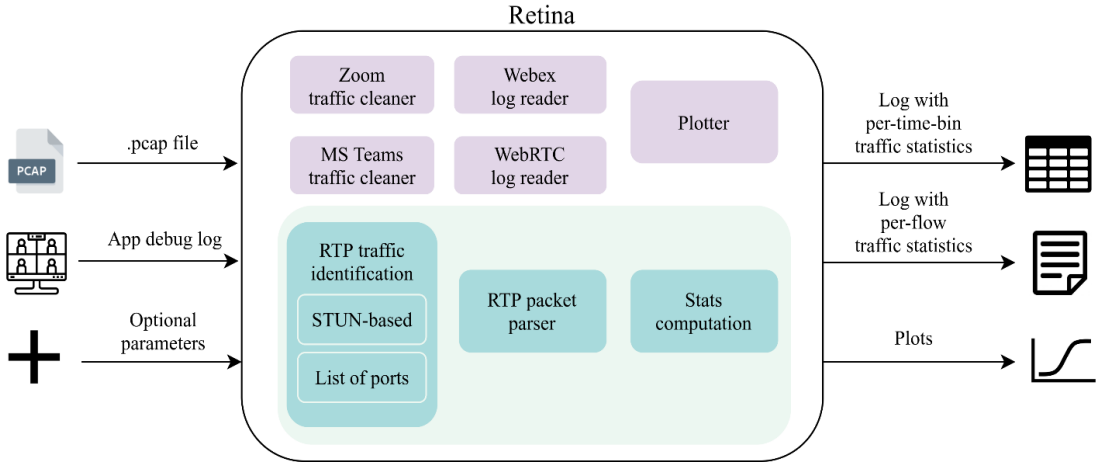


Figure 2.2: *Retina working process [3]*

As it is seen, it takes one or more RTP traffic pcap files as input and outputs various statistic logs and plots. This tool was extremely useful for my research to structure the collected data and merge them with the application logs in order to prepare it to be read and processed accurately from the Jupiter Notebook web-application tool. In order to learn more about this tool, you can go to the link in [3] and find some more detailed information illustrated with examples.

2.4.2 Wireshark

Wireshark is a network protocol analyzer, or an application that captures packets from a network connection and it is one of the most widely used packet sniffers nowadays. Like all the other packet sniffing tools, it has the following three main functions: packet capture, filtering and visualization. In the figure 2.3 we show how a Wireshark interface looks like via a print screen in a randomly captured traffic:

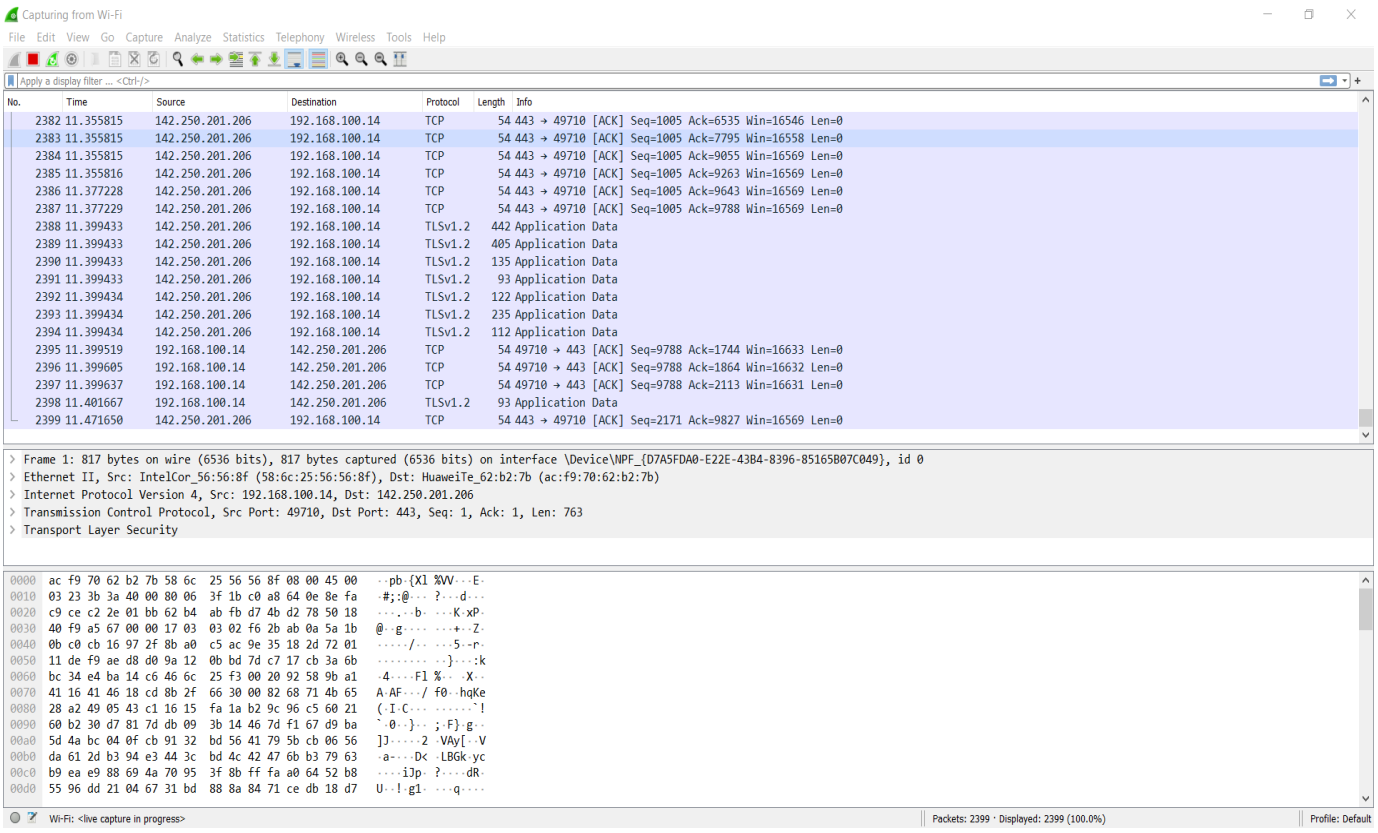


Figure 2.3: *Example of Wireshark interface [4]*

Wireshark has many uses, including troubleshooting networks that have performance issues. Cybersecurity professionals often use Wireshark to trace connections, view the contents of suspect network transactions and identify bursts of network traffic. Basically, Wireshark is quite important

in traffic analyzing and it has been crucial in collecting the huge amount of data used in this research.

2.4.3 Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more [5]. It is very practical with ML algorithms and not only, due to the fact that it contains various libraries so that it simplifies the implementation of those algorithms. Basically, it is the web application where I run all the codes written in Python language used for this research and its interface looks like in the Figure 2.4

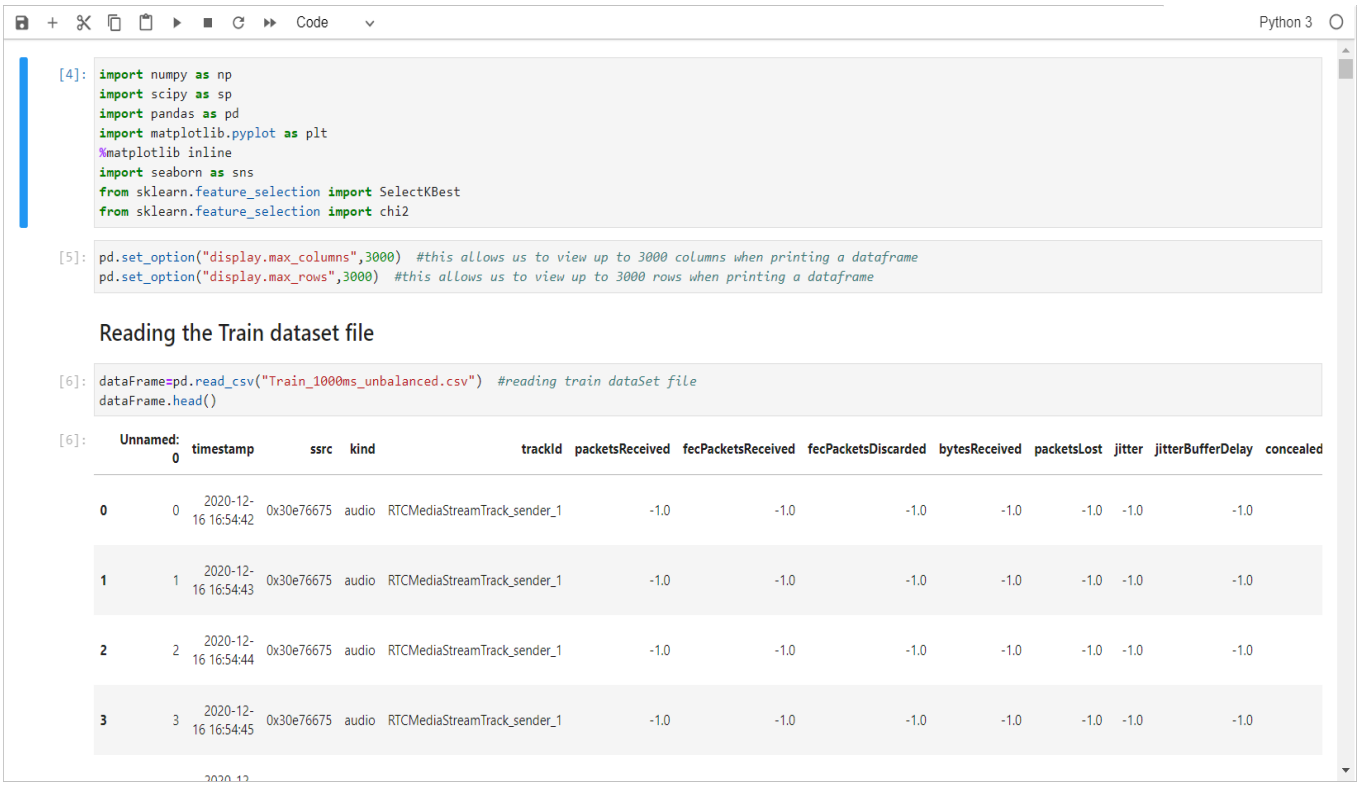


Figure 2.4: *Jupyter Notebook example visualization*

2.5. Machine Learning

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy. Machine learning algorithms use historical data as input to predict new output values. ML is used almost in any research field nowadays, as it simplifies the analysis of data offering intelligent insight. The fields where ML is heavily used are Prediction, Image recognition, Speech recognition, Medical diagnosis, Financial industry and trading etc.

2.5.1. Machine Learning types

Machine Learning can be classified into 3 main categories:

Supervised Machine Learning

In this type of machine learning there is a set of inputs which are used to feed the model with and a set of target values or output data, used as a reference in order to fit the model together with the input dataset. So basically, there are some labels used as a target so the model tries to find the correlation between input data and those target values. This type of ML is useful for binary classification, regression, ensembling etc.

Unsupervised Machine Learning

This type of machine learning involves algorithms that train on unlabeled data. The algorithm scans through data sets looking for any meaningful connection. The data that algorithms train on as well as the predictions or recommendations they output are predetermined [24]. This type of ML is used for clustering, anomaly detection, association mining etc.

Semi-Supervised Machine Learning

Semi-supervised learning offers a happy medium between supervised and unsupervised learning. During training, it uses a smaller labeled data set to guide classification and feature extraction from a larger, unlabeled data set. Semi-supervised learning can solve the problem of having not enough labeled data (or not being able to afford to label enough data) to train a supervised learning algorithm [23].

2.5.2. Introduction to ML algorithms

In this section, a brief overview on the algorithms used in this research will be given. There will be introduced the main goals for each of them, and the methodology they use to make the calculations.

a. Linear Regression

This algorithm is used to estimate real, continuous values such as age, cost, weight etc. Here, the relationship between independent and dependent variables is established by fitting a best line. This

best fit line is known as regression line and represented by a linear equation Y= a *X + b. In this equation:

- Y – Dependent variable
- a – Slope
- X – Independent variable
- b – intercept

These coefficients “a” and “b” are derived based on minimizing the sum of squared difference of distance between data points and regression line [25].
In the figure 2.5, it is shown an example of this best line and the real values around it:

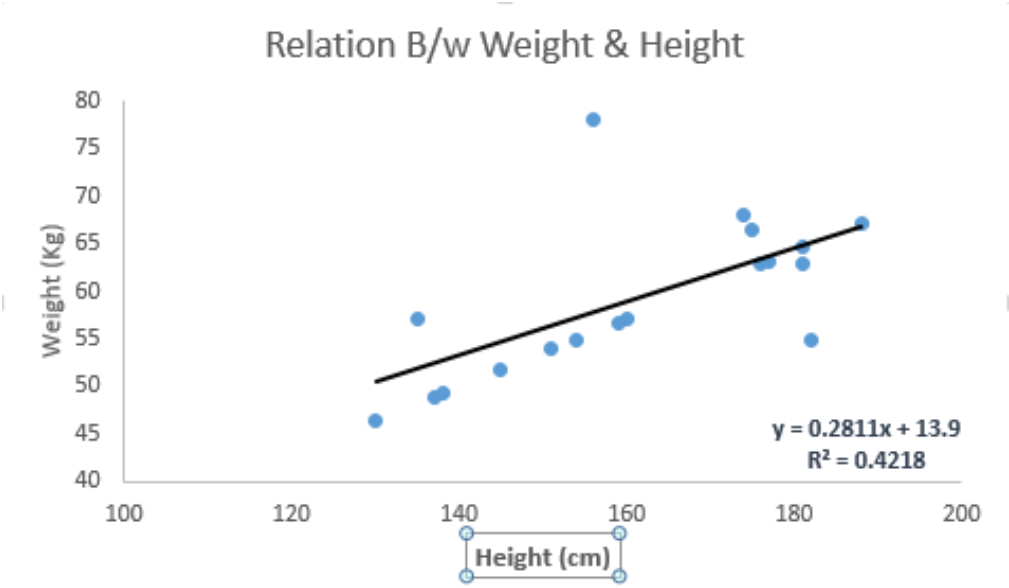


Figure 2.5: *The example of best line in Linear Regression [25]*

As it is seen from the figure 2.5, this algorithm makes the predictions based on that best line by comparing the value with the points in the line.

b. Logistic regression

Even though from the name it may seem as a regression algorithm which predicts continuous values, logistic regression is nothing else but the classification algorithm used to predict discrete values such as the occurance of an event or not, true or false, which group an object is part of (where the number of groups is finite) etc.

c. Decision Tree

This is one of the most frequently used supervised learning algorithms which works for both discrete and continuous data prediction. In the figure below this algorithm is explained with a very

simple example of the decision of a group to play a particular sport or not depending on weather conditions. So, it takes the different conditions and dependently splits them into different decision groups and looks at the occurrence of playing under those conditions or not. Then depending on the majority the model decides what to predict.

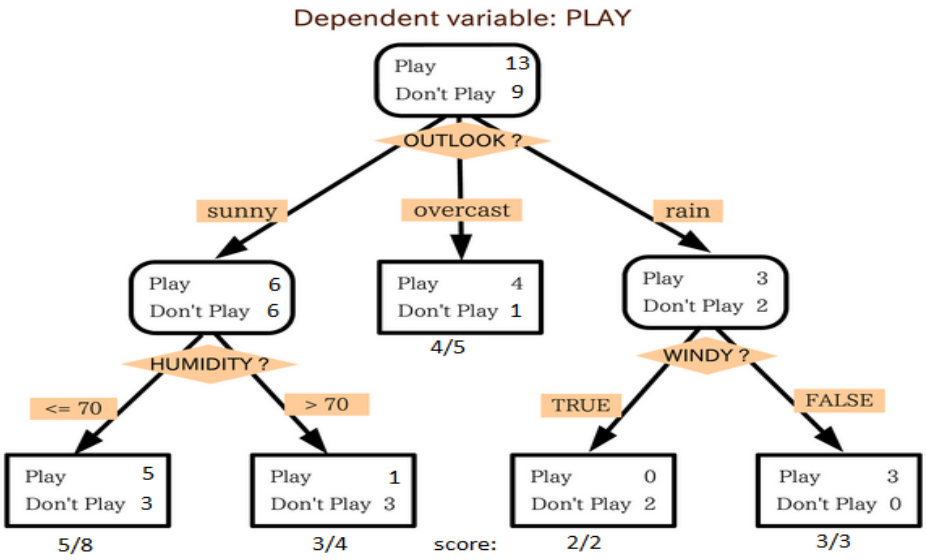


Figure 2.6: *The working principle of Decision Tree [25]*

d. Random Forest

This one is a collection of decision tree algorithms therefore it is expected to perform better than the decision tree. This is because the random forest is the mean of multiple decision trees.
Basically, those are the most useful algorithms which are worth to mention about. Then if we go deeply in machine learning we can find also other types such as: Naïve Bayes, kNN, k-means etc.

2.6 Machine Learning performance evaluation

When implementing ML algorithms it is important to make some evaluations of the model. In order to do so, we use various metrics where the most common ones used in this research are confusion matrix, f-1 score, Mean Absolute Error (MAE), Mean Squared Error (MSE) etc. In this section we introduce the currently mentioned metrics for ML model evaluation and give a brief idea on which one fits better to which.

2.6.1 Confusion Matrix

Confusion Matrix as the name suggests gives us a matrix as output and describes the complete performance of the model. It is commonly used in classification problems and it gives the exact number of the true and false predictions. Basically there are 4 important terms or possible alternatives when using this kind of metrics:

I. True Positives: The cases in which we predicted YES and the actual output was also YES.
II. True Negatives: The cases in which we predicted NO and the actual output was NO.
III. False Positives: The cases in which we predicted YES and the actual output was NO.
IV. False Negatives: The cases in which we predicted NO and the actual output was YES.

In the Table 2.1, it is shown an example of how the Confusion Matrix look like.

Table 2.1: *Confusion Matrix example [6]*

n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	50	10
Actual: YES	5	100

Note that Confusion Matrix forms the basis for other type of metrics.

2.6.2 F1 Score

F1 Score is the Harmonic Mean between precision and recall. Therefore, firstly we need to define the precision and recall which are as following:

- ***Precision***: It is the number of correct positive results divided by the number of positive results predicted by the classifier:

$$\text{Precision} = \frac{TruePositives}{\text{TruePositives} + FalsePositives} \tag{2}$$

- ***Recall***: It is the number of correct positive results divided by the number of ***all*** relevant samples (all samples that should have been identified as positive):

$$\text{Recall} = \frac{True\ Positives}{\text{TruePositives} + FalseNegatives} \tag{3}$$

On the other hand, F1 Score tries to find the balance between precision and recall. The range for F1 Score is [0, 1]. It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances). The greater the F1 Score, the better is the performance of our model. Mathematically, it can be expressed as [6]:

$$\text{F1} = 2 * \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{4}$$

2.6.3 Mean Absolute Error

Mean Absolute Error is the average of the difference between the Original Values and the Predicted Values. It gives us the measure of how far the predictions were from the actual output. However, they don't give us any idea of the direction of the error i.e. whether we are under predicting the data or over predicting the data [6].

2.6.4 Mean Squared Error

Mean Squared Error(MSE) is quite similar to Mean Absolute Error, the only difference being that MSE takes the average of the **square** of the difference between the original values and the predicted values. The advantage of MSE being that it is easier to compute the gradient, whereas Mean Absolute Error requires complicated linear programming tools to compute the gradient. As, we take square of the error, the effect of larger errors become more pronounced then smaller error, hence the model can now focus more on the larger errors [6].

Chapter 3

Literature Review

The aim of this chapter is to present a variety of researches regarding the QoE inference through QoS, studied and implemented so far by different researchers in this field. Our goal on the other hand, is to analyze and adapt the most suitable one to our dataset.

3.1. QoE Inference

As anticipated in the previous chapter, QoS and QoE might seem as the same thing but in reality they are quite different from each other so that it has been very difficult to find a strong correlation between two. QoS metrics do not directly translate to customer experience, which is more qualitative than quantitative [10]. For instance, QoE is the quality perceived in the end user which is directly connected to events such as video freezing, voice and video delay etc., whereas QoS is everything that can be measured by network devices in the middle of the transmission. Examples of QoS metrics are: bandwidth, packet loss, throughput etc. This means that not necessarily having a high bandwidth guarantees high quality at the end user, since there may be other problems with other metrics and not any of them has the same impact on QoE. Saying that, our aim is to provide an approach which takes into account all these QoS metrics and translates them to QoE. However, before diving into our approach, let's first have a look at some similar researches done related to this topic.

The study in [11] aims at infering QoE without having to look at metrics generated at application side, however this requires a software on the user's device that is capable of measuring QoE metrics for that app and reports this information to the entity implementing the QoE-aware mechanisms. The experiment here is based on 2 scenarios, one of them takes into account video conference apps (so RTCApp) and the other one the video streaming scenario. In this study, an algorithm called: "**Adaptive Sampling of QoS metric space"** was introduced. Basically, the authors of this paper vary some of the QoS metrics while the others remain fixed, then they check the QoE metric observed. The traffic shaping using tc is done at the WiFi Access Point.
They envision that an application maintains an offline network QoS-to-QoE mapping and keeps track of its network QoS and QoE metrics during run time. In this way the QoE metrics are observed from those modules in the end users' devices for each QoS parameter combination. Then, the authors of this paper argue that they can map the QoE metrics (such as *frame_rate* and *end-to-end video delay* for video conferencing) to a limited number of QoE classes, in this way, they can selectively increase the sampling of QoS metric space close to the borders instead of increasing (or decreasing) a metric which if changes would have no effect to the QoE at all.
In the figure 3.1, it is clearly shown how this algorithm performs, it shows that even if you decrease the packet loss at the lowest level still without increasing the bandwidth the fps will be the same.

The main idea of this algorithm is to increase the parameter where it is needed (where it affects the QoE the most).

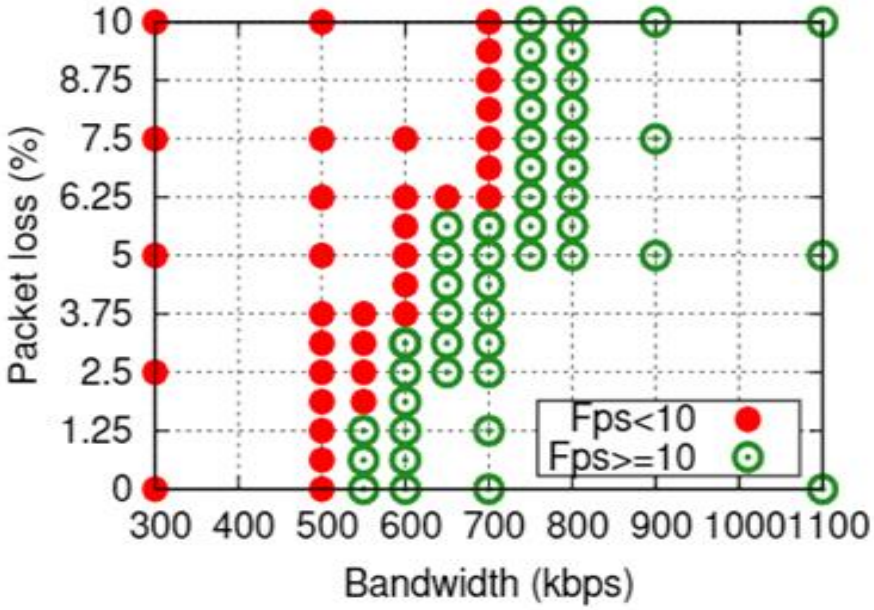


Figure 3.1: *Example of algorithm's working principle [11]*

In the paper named "**Analysis of the effect of QoS on Video Conferencing QoE**" in [10], there is an attempt to correlate video conferencing QoE with network QoS. The scenario introduced is as follows: two persons will conduct a video call and send their camera feed to each other over a manipulated network condition. Network manipulation is implemented with **Netm** and **Wondershape**r tools in Linux. This experiment is run with Lime tool in Linux which is a Docker image which creates client applications using Docker Engine that connects to a video conferencing server and joins a video call. As a method it uses a tool called **Video Quality Analyzer (VQA)** which is used to measure the quality of a video by comparing the original one sent at the beginning in the TX side with the one received at the RX side. **Video Quality Analyzer (VQA)** is a very powerful software solution to **precisely measure and analyze the video quality perceived by end-users.** It produces quality scores which are highly correlated with the human judgment of visual data quality.

Then they obtain some DMOS values which represent the QoE perceived in the RX side, by comparing the video sent from the TX (which is recorded) and compare it with the one received at the RX side in order to produce some DMOS values. Then it is possible to analyze and reveal the correlation between QoS and QoE metrics which will help us to create a model based on the QoS parameters. For instance, if jitter is above a particular value and bandwidth is below another particular value then it will have a particular effect, or if we increase only one of the QoS parameters then we'll have a better QoE and so on. Basically, the idea is to clearly identify the QoS metrics which affect the QoE most and in this way, ISPs can improve the quality.

According to [12], the QoE estimation researches are mainly divided in 2 classes:

a) **Session Modelling based Approach (Deep packet inspection)**: This method looks at packet headers in clear but it works well only for unencrypted traffic (HTTP). However, nowadays most of the traffic is in HTTPS (encrypted), so as a bconsequence this method is obsolete.
b) **ML-based Approach**: This method infers QoE by correlating the network observable QoS metrics with ground truth QoE, by using ML algorithms. However, this method has some limitations:
    - ML based method requires ground truth QoE metrics for initial training which are not generally available.
    - Different video services use different service design parameters thus one common method cannot be generalized.
    - These approaches give a categorical QoE estimation which may not be true for active QoE based traffic etc,.

Due to the problems mentioned above for both classes, a method named eMIMIC which is similar to **session modelling based approach** but works for encrypted traffic was targeted by the authors of this paper so that they invented eMIMIC.
The network scenario they are looking at in this paper is HAS (HTTP-based Adaptive Streaming), so there is no real time communication traffic but still holds for QoE estimation and then it can be adapted to RTC.
The methodology used in [12] is of type heuristic. They have been introduced some formulas as seen below:

AVERAGE BIT-RATE:

$$BR = \frac{\sum_{i=1}^{N} Q_i}{N}$$

$\boldsymbol{Q_i}$ → Estimated Bit-rate

N → Total no. of chunks

REBUFFEREING RATIO:

$$RR = \frac{\sum_{i=1}^{N} b_k}{N*L + \sum_{i=1}^{N} b_k}$$

$\boldsymbol{b_k}$ → Rebuff time between 2 consecutive chunks

L → Chunk duration

BITRATE SWICTHES:

$$\boldsymbol{SPM} = \frac{\sum_{i=2}^{N} \mathrm{I}\,(Q_i != Q_{i-1})*60}{N*L}$$

N → Switches per minute

STARTUP TIME:

$$ST = T_{loading} + \mathrm{TTNC} + T_{decode}$$

TTNC → Time taken to download min number of chunks to start the playback

So they measure the QoE metrics according to formulas above and finally evaluate the obtained estimations by comparing them with the ground truth QoE metrics. They built an automated browser-based framework that streams video sessions of a video service in a web browser under emulated network conditions and collects packet traces, HTTP traces and ground truth video QoE metrics. They use Java implementation of a popular browser automation framework, known as Selenium1. The HTTP logs of encrypted sessions are collected using a trusted proxy, BrowserMob proxy2, that is easy to integrate with Selenium. They also use TShark3 for capturing packet-level network traffic and Linux Traffic Control (tc) to emulate different network conditions.
So, to conclude, they use all these programs to extract all the QoE metrics from the video sessions and they compare them with the obtained ones from eMIMIC formulas previously introduced. After the comparison of the results they observe a really good performance which overpasses the ML based method as well.
As a conclusion we can say that eMIMIC is a relevant design because of the following reasons:

- Works well on encrypted traffic
- Minimally depend on ground truth QOE
- Generalizes across different services
- Provide quantitative measurements etc.

On the other hand, the paper in [15] aims to close a significant gap between QoS and QoE in home networks by proposing a framework for inferring QoE from remotely collected network QoS metrics. It focuses on video services (e.g., YouTube application) as the main contributor and generator of indoor network traffic. The method used is of **Predictive modelling** where a predictor for multiple QoE classes given the Application and network QoS metrics where app QoS metrics are remotely accessible from access points based on industry adopted standards (i.e., TR-181). This enables operators to infer specific QoE metrics using remotely collected passive network measurement with no knowledge of application-specific parameters. To bridge the gap between the network QoS and users' QoE, a two-step process is undertaken. First, the application QoS is mapped to QoE classes using known models, and second, the mapping between the network QoS and QoE classes is devised. *Network QoS and derived QoE classes make the training dataset* **(e.g., ground truth)** *upon which the mapping function F{·} is learned.* Once the mapping function is obtained, the operators can infer the QoE of an end user upon receiving network QoS indicators and without knowledge of the application QoS metrics, so we needed app QoS only at the beginning to create a model which will help in mapping directly from Network App. However, it is worth mentioning that inferring QoE from QoS measurements is feasible per application basis. This means that, If one wants to infer the QoE of another application, a corresponding mapping function should be derived following the proposed conceptual framework. So to summarize, they used 2 types of QoS metrics: App and Network, where APP is used to directly map the APP QoS to QoE with the aid of some known models and then use the Network QoS only for the mapping since there are Network QoS metrics which can be viewed always from ISP providers and we need

to have a map between them and QoE classes to improve the quality of the perceived videos. In the table 3.1 below there is a summary of some of the studies which have similar goals with ours:

Table 3.1: *Summary of some similar studies*

Publication (title)	Application	QoS Parameters	QoE metrics	Technique	Ground Truth QoE
QoE inference without App Control [11]	***AppRTC*** and ***on-demand video streaming***	***Bandwidth*** and ***Loss Rate***	***Frame rate*** and ***end-to-end delay*** (for AppRTC), ***Bitrate*** and ***rebuffering freq***. (for on-demand video streaming)	***Decision tree*** generated via from Adaptive sampling algor.	YES
Analysis of the effect of QoS on Video Conferencing QoE [10]	**Video call** (so **RTC**)	**Packet Loss**, **Jitter** and **Bandwidth.**	**DMOS**	Correlation Analysis	YES
eMIMIC: Estimating HTTP-based Video QoE Metrics from Encrypted Network Traffic [12]	**HTTP-based Adaptive Streaming**	**Average Bitrate**, **Rebuffering Ratio**, **Bitrate Switches** and **Start-up time**.	**Chunk duration**, **decoding time** and **Loading time.**	Correlation Analysis through formulas	YES
Quality of Experience Inference for Video Services in Home WiFi Networks [15]	**On-demand video services**	**Network QoS** (error rate, dropped pkt rate, channel utilization etc.) and **Application QoS** (buffer delay, rebuffering freq., buffer ratio etc.)	**QoE classes** as function of QoS parameters, **MOS**.	Machine Learning techniques, Correlation analysis.	NO
From Network Traffic Measurements to QoE for Internet Video [30]	**On-demand video services (Youtube)**	**Downlink BW, Uplink BW, RTT, Packet Loss rate, Jitter**.	**MOS**	Unsupervised ML, **Random Forest**	NO

Apart from the above mentioned papers, the study we have been trying to take as a reference is the one in [16], which is a paper released by Cisco, and it has the main goal to characterize the relationship between most relevant QoE indicators for RTC video-conferencing applications and QoS parameters for online inference, as the title of this paper implies. In this paper, the focus is on Video Conferencing Apps, and 3 are metrics to define the QoE used:

A) **Resolution**: Defined as the size in pixels of the received video frames.
B) **Smoothness**: The number of frames per second displayed to the user.
C) **Concealment**: The time spent by the decoder rendering frames that are partially received.

The method being used to observe the correlation between those 3 QoE parameters and QoS ones in this study is via Machine Learning algorithms.
Given the fact that this thesis is part of a collaboration with Cisco, we mainly rely on this methodology but we adapt it to our problem and dataset.

Chapter 4

Methodology

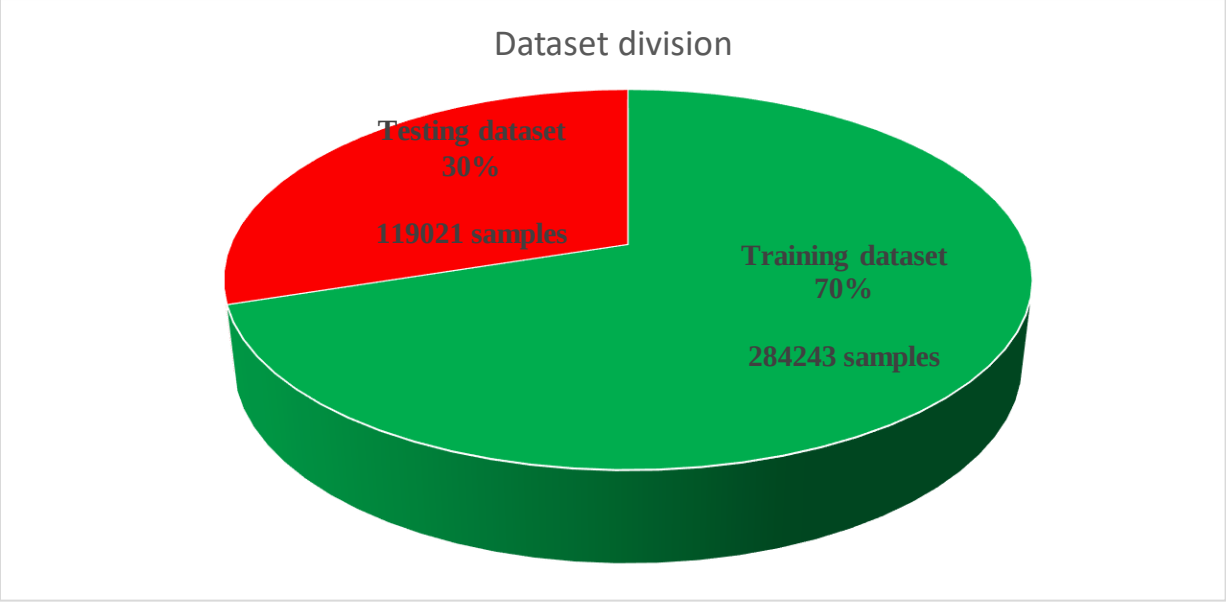
The purpose of this chapter is to explain the methodology used in this research, including a brief overview on the network scenario, QoE metrics and statistics about the most important QoS features used to define the QoE, such as bitrate, interarrival time etc. Then we describe the ML methodology with different algorithms and methods used to improve our model.

4.1. Network scenario and dataset description

Prior to the analyzation of model's performance, it's worth mentioning the type of meeting, application and tools used for data collection. Starting from the type of dataset, we can say that those data were collected from the meetings conducted between some people, where there is a presenter who shares the screen and others who ask questions and discuss a particular topic. Therefore, it is obvious that the type of traffic is WebRTC where the meeting was held via the "Jitsi" application. The data collection was done through application logs for target features as well as via a packet sniffer and analyzer tool such as Wireshark for training ones. Both tools were introduced at the second section of this report.

After having introduced the methodology of the data collection, now let's dive into the contents of the dataset. The Table 4.1 gives us a clear picture about the exact numbers of the dataset partitioning which will be used to train the model:

Figure 4.1: *Visualization of the dataset division*



- Total dataset samples = 403264

Translating that into words, we can say that the number of pcaps in our dataset is 50 where the total number of samples is 403264. Among them, 284243 of the samples are used for training and 119021 for testing. It's very important to note that the pcaps used for training and test dataset are of different type, which makes the problem a realistic one. The total time of the call is around 34 hours which is a considerable amount for collecting enough data.

Apart from the training dataset which are collected in the network vantage points through Wireshark, there is also another type of dataset collected from application logs which are measured at the end user side and some of them will be used as ground truth metrics. In the Figure 4.3, we show some pictures and graphs representing the fields extracted from the application logs.

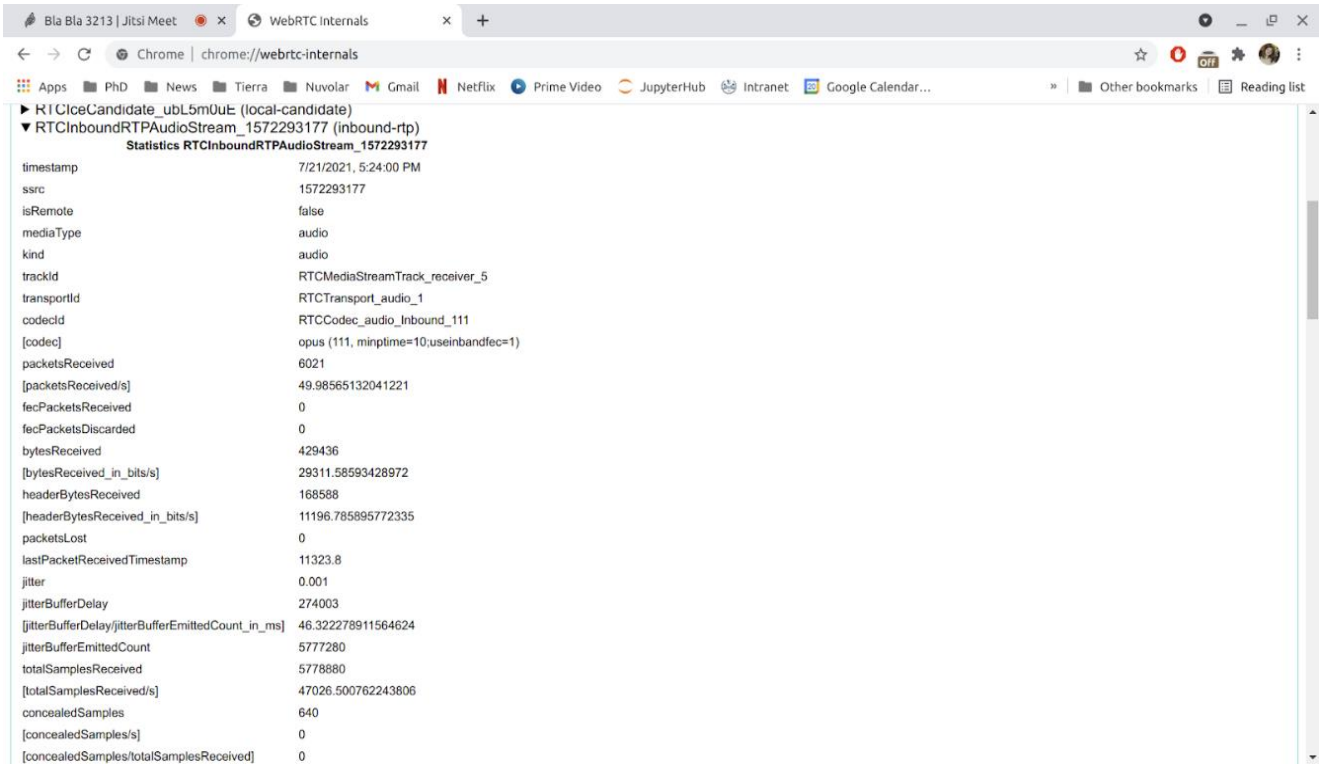


Figure 4.2: *Some of the fields extracted from Application logs*

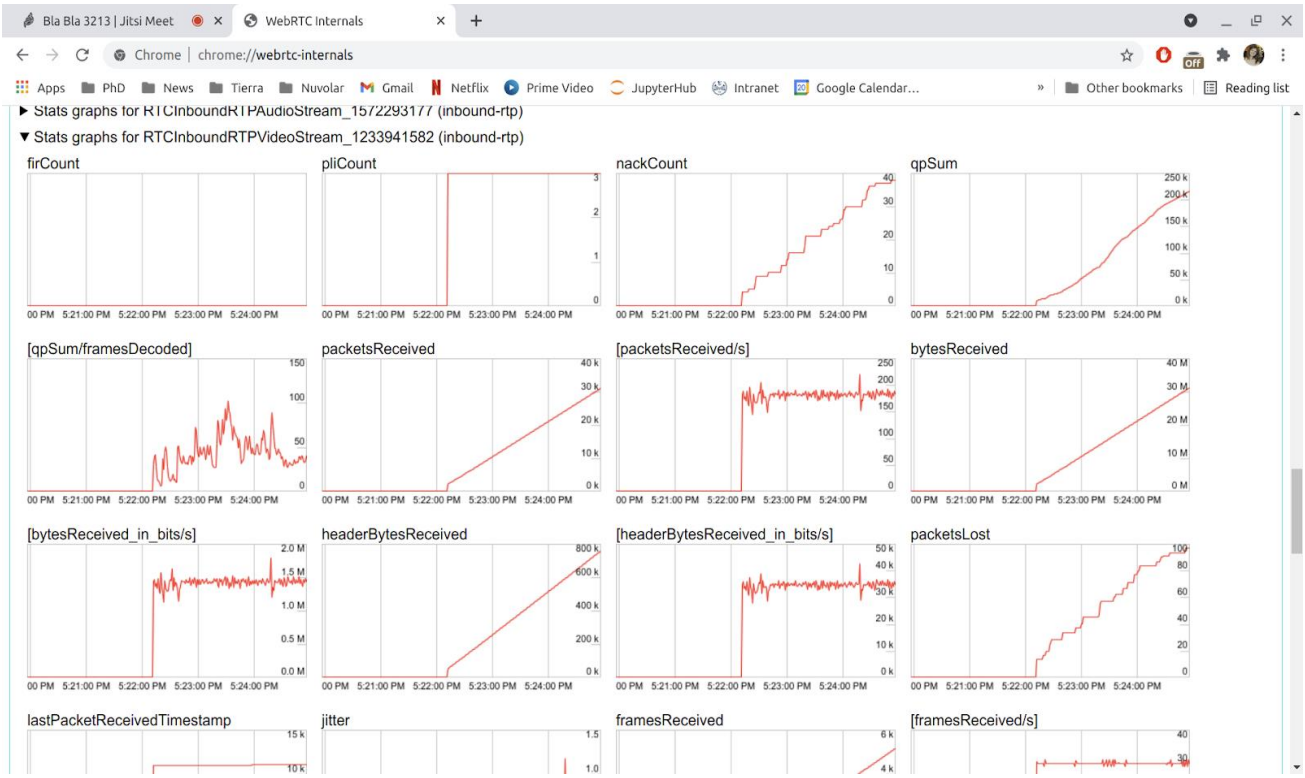


Figure 4.3: *Graphical distribution of some of the input features' values*

As anticipated, in the Figures 4.2 and 4.3, only the fields collected from Application logs are shown, which means that those data can be used as ground truth but not as a training dataset. The reason behind this scenario is that the network devices in the vantage points of the network (i.e. routers) cannot see everything due to the applied encryptions. However, since the encryptions are mostly end-to-end, the end user is able to decrypt packets, therefore from the logs we can see what are the perceived results. As introduced at the beginning, this is our main goal, by using the headers which are in clear and which can be understood by the network devices, we try to build some approaches using ML algorithms and compare them to the corresponding metrics collected from the application logs.

Saying that, features calculated from the Wireshark tool, so those in which the network vantage point devices such as routers can see in clear, are listed in the table 4.1 below. These fields can be grouped into 4 main categories:

a) **Inter-arrival**: The time between 2 consecutively arrived packets.
b) **Packet length (UDP):** The length of UDP packets.
c) **Inter packet length**: The distance between consecutive packets.
d) **RTP inter-arrival**: The time between 2 consecutive packets in real time communication.

The **figure 4.4** reflects 2 important observations, firstly we can see the fields which are part of the main 4 categories introduced in the previous paragraph and secondly it is clearly shown the relationship that these fields have with each other. The relationship between input training features is extremely important to be observed due to the fact that the higher the correlation between them, the lower it will be the performance. Therefore, by having a clear overview on this relationship among them, we can implement various feature selection methods which consequently will improve the performance.

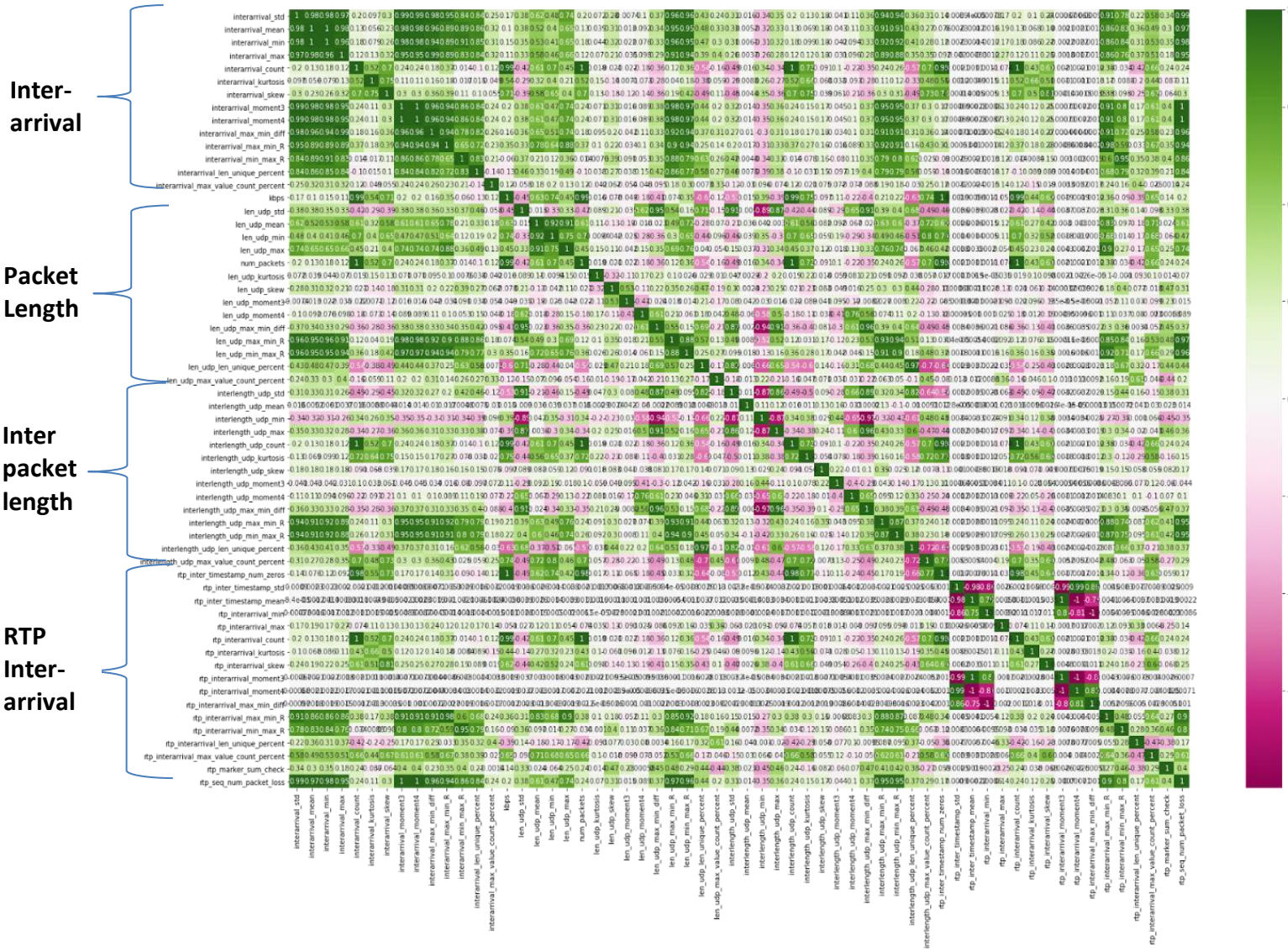


Figure 4.4: *Correlation between input features*

The main observation from the heat map in the Figure 4.4 is that some of the fields belonging to the same main groups have a strong correlation with each other (i.e.'interarrival_std' and 'interarrival_max' fields, 'len_udp_max_min_R' and 'len_udp_min_max_R' fields, and so on). However, in general there is not that much correlation between other features, especially between those part of different categories, which is what we want because as anticipated, the higher the correlation among the training data, the worse the prediction results will be due to the low diversity of the training dataset. This simply means that training the model with different data makes it easier to make predictions for the ones never seen before due to the high diversity, whereas when you teach the model with less various data it will be too difficult to analyze new ones. More details related to the training fields used in this experimentation can be found using the link of Retina in [3]. Saying that, let's now have a look at the ground truth values or the so called "target fields".

4.2. Ground Truth Overview

As anticipated in the previous chapter, we took as a reference the paper in [16] released by Cisco. Recalling what we discussed previously, in [16] they rely on the following 3 QoE metrics to be predicted which are "*resolution*", "*smoothness*" and "*concealment*". Since this research is held in collaboration with Cisco and we're trying to adapt it according to our dataset, we found out that in our dataset the equivalent features of those 3 metrics are "***framesPerSecond***" representing *smoothness*, "***frameWidth***" representing *resolution* and "***concealmentEvents***" and "***framesDropped***" representing the *concealment for audio and video* respectively.

The reason why it is decided to use three metrics instead of one is because each one alone is not enough to characterize the overall video QoE. For instance, a user may receive a video at the maximum resolution but just a few frames per second. In this case, looking only at the video resolution we may say that the QoE is high, while in reality it is not.

Saying that, the expectations are to have a tight correlation between resolution and bandwidth due to the fact that the wider the bandwidth the higher the number of frames per second to be displayed in the screen. On the other hand, resolution may be negatively impacted by packet inter-arrival time which means that the higher the interarrival time the lower the resolution. What we can say in prior is that we expect it to be strongly correlated to loss rate which is obviously meant from definition of the concealment itself.

In the following plots, in the Figures 4.5, 4.6, 4.7 and 4.8, we show the Cumulative Distribution Functions (CDF) of those ground truth fields' values.

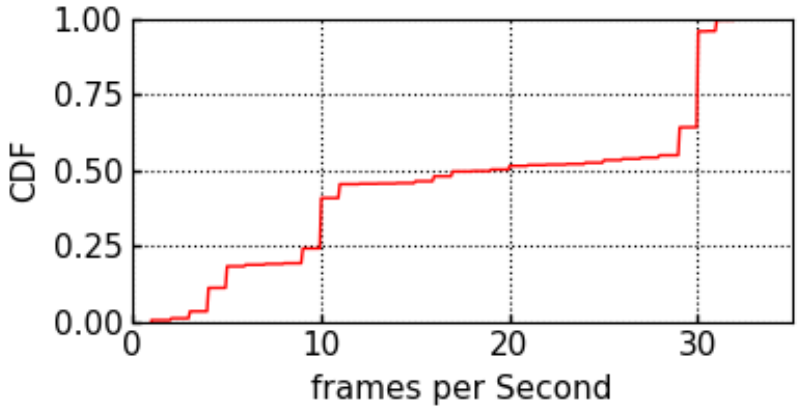


Figure 4.5: *CDF for "Frames per Second" field*

Figure 4.5 gives us a clear picture on how the values for the "*resolution*" metric are distributed. From this plot we observe that 25% of the values of this dataset are below 10 fps whereas almost 20% of the whole data is equal to 10 fps. Then there is a variation of data between 12 fps until 30 fps. Finally, we notice that 30 fps is the most common value with almost 30% of the whole data. Basically, this is a clear overview regarding our first target's values distribution which corresponds to framesPerSecond field representing the resolution metric. Now let's have a look on the next target field which's values distribution is shown in the figure 4.6 below.

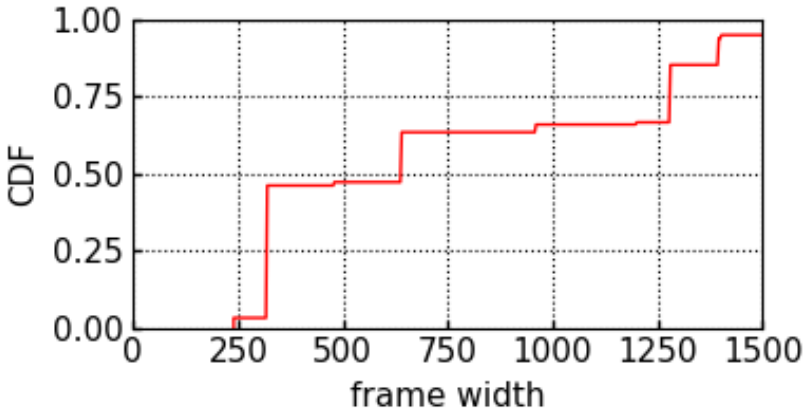


Figure 4.6: *CDF for "frame width" field*

Regarding the CDF for "frameWidth" field or smoothness metric, the figure 4.6 tells us that the most common value with more than 40% of frequency is 320 pixels one. This is followed by the value 640 with around 15% of frequency and so on. The most noticeable thing in this graph is that almost 50% of the values are less than 620 pixels and the remaining part belong to all the others. Again, this graph helped us in understanding what kind of values are we going to expect when predicting that field.

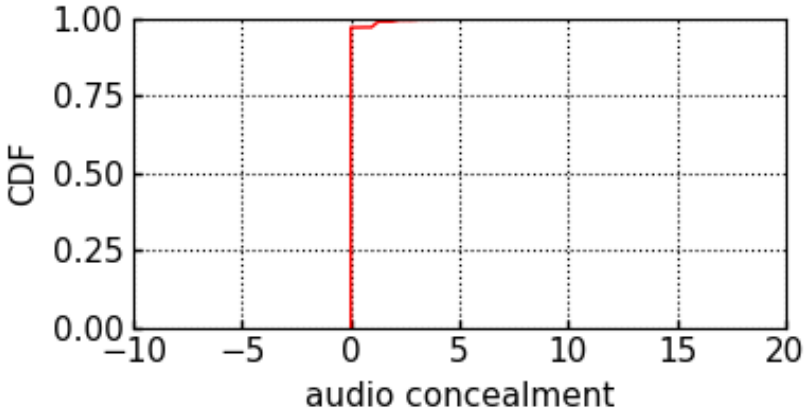


Figure 4.7: *CDF for audio concealment (or "concealmentEvents" field)*

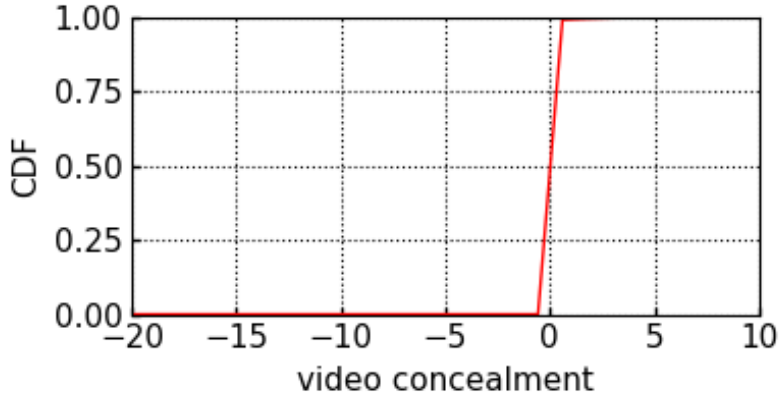


Figure 4.8: *CDF for video concealment (or "framesDropped" field)*

Finally, figures 4.7 and 4.8 represent the CDF distribution for the "*framesDropped*" and "*concealmentEvents*" fields respectively. The first thing we need to know here is that initially the values of both these fields are in a cumulative way, which means they are summation of previous values. For this reason, what we need to do in order to find the correct values for both fields is to extract the subsequent values of each field (part of the same flow) from each other and only then we'll have the exact values of concealments for each field. After having done this, we can view the exact values from the CDF plots shown in the figure 4.7 and 4.8.

The first thing we notice in common from both graphs is that the major part of the field is occupied by 0s (zeros), which represent the NO CONCEALMENT case. Then we have the other values representing the case where concealment is present and they indicate the exact number of the concealments for each flow. The strange thing initially was the presence of the negatives when we ectract the subsequent values, however we found out that this happens due to the duplicated packets.

So to conclude, we had a detailed look at the distribution of each feature which will be used as target fields for our model. The analyzation of them is quite important in order to have a clear idea on what to expect and comment the results when necessary. In the next chapter, we'll keep analyzing the results obtained from the experiments done using these features and correlate them with the model's behavior.

4.3. Implementation Techniques

After having introduced the dataset in the previous paragraphs, it's time to explain the methodology used to process these data in order to obtain the best possible results. In this section, differently from the introduction on ML algorithms made at the beginning, a brief overview about some of the methods used to improve the model's accuracy are given.

4.3.1. Overview

Firstly, it was started by using simple algorithms such as Decision Tree and Random Forest without any additional method for performance improvement. Later on, with the aim to improve the performance, some feature selection methods such as: information gain, correlation coefficient, Annova f-value etc., were applied. Moreover, outliers were removed and combined with the previously mentioned feature selection methods in order to obtain more accurate results which will be explained in detail in the next chapter. Saying that, in the following section, a brief overview of the recently mentioned methods about feature selection and outlier removal is given.

4.3.2. Missing value

Both training and testing dataset generally contain some missing values regarding particular fields and this is one of the many problems in Machine Learning. The reasons behind that can of different types such as:

1. Data extraction: When gathering the data there may be problems with the extraction process.
2. Data collection: These errors occur at time of data collection and are harder to correct. They can be categorized in four types:

- **Missing completely at random:** This is a case when the probability of missing a variable is the same for all observations.
- **Missing at random:** This is a case when a variable is missing at random and the missing ratio varies for different values / levels of other input variables.

- **Missing that depends on unobserved predictors:** This is a case when the missing values are not random and are related to the unobserved input variable.

Missing that depends on the missing value itself: This is a case when the probability of missing value is directly correlated with missing value itself [22].

The negative effects of missing data are not to be underrated since they can reduce the power / fit of a model or can lead to a biased model because we have not analysed the behavior and relationship with other variables correctly. It can lead to wrong predictions or classification.

The treatment of missing values can be done in different ways:

1) **Deletion**: This is also of two types: List wise and Pairwise. List wise is the case when the whole list corresponding to a missing value is deleted. So if a value in a particular field is missing then the whole fields regarding this line in the dataset will be deleted and as a consequence this list will be missed. This method has the disadvantage of information loss which reduces the power of the prediction. On the other hand, there is pairwise deletion which is nothing else but removing only the value which is missing and keeping the other fields regarding this list. This has the advantage of not losing any information however the main disadvantage is the usage of different sample sizes for different variables.

2) **Mean/ Mode/ Median Imputation:** This method is one of the most used ones and is nothing else but the substitution of missing values with the estimated ones. Those imputations may be of two different types:
   a) **Generalized imputation**: we calculate the mean or median for all non-missing values of that variable (in other lists) then replace missing value with mean or median.
   b) **Similar case imputation**: This is the case of categorizing the data in some groups according to a particular field and calculating the mean or median for these groups and then replacing the missing values with the mean or median of the group it is part of.

3) **Prediction model:** Prediction model is one of the sophisticated methods for handling missing data. Here, we create a predictive model to estimate values that will substitute the missing data. In this case, we divide our data set into two sets: One set with no missing values for the variable and another one with missing values. First data set becomes the training data set of the model while the second data set with missing values is the test data set and the variable with missing values is treated as the target variable. Next, we create a model to predict target variables based on other attributes of the training data set and populate missing values of the test data set.We can use regression, ANOVA, Logistic regression and various modeling techniques to perform this. There are 2 drawbacks for this approach:
   a) The model estimated values are usually more well-behaved than the true values

b) If there are no relationships with attributes in the data set and the attribute with missing values, then the model will not be precise for estimating missing values [22].

4) **NN Imputation:** In this method of imputation, the missing values of an attribute are imputed using the given number of attributes that are most similar to the attribute whose values are missing. The similarity of these values is determined using a distance function.

Advantages:

- k-nearest neighbour can predict both qualitative & quantitative attributes
- Creation of predictive model for each attribute with missing data is not required
- Attributes with multiple missing values can be easily teat
- Correlation structure of the data taken into consideration

Disadvantages:

- KNN algorithm is very time-consuming in analyzing large database. It searches through all the dataset looking for the most similar instances.
- Choice of k-value is very critical. Higher value of k would include attributes which are significantly different from what we need whereas lower value of k implies missing out of significant attributes [22].

4.3.3. Outlier Detection and Treatment

Before proceeding with outlier detection or how to treat then, let's first see what an outlier is. According to Wikipedia, in statistics, an **outlier** is a data point that differs significantly from other observations. An outlier may be due to variability in the measurement or it may indicate experimental error; the latter are sometimes excluded from the data set. An outlier can cause serious problems in statistical analyses. In a graphical point of view, outliers can be defined as below:

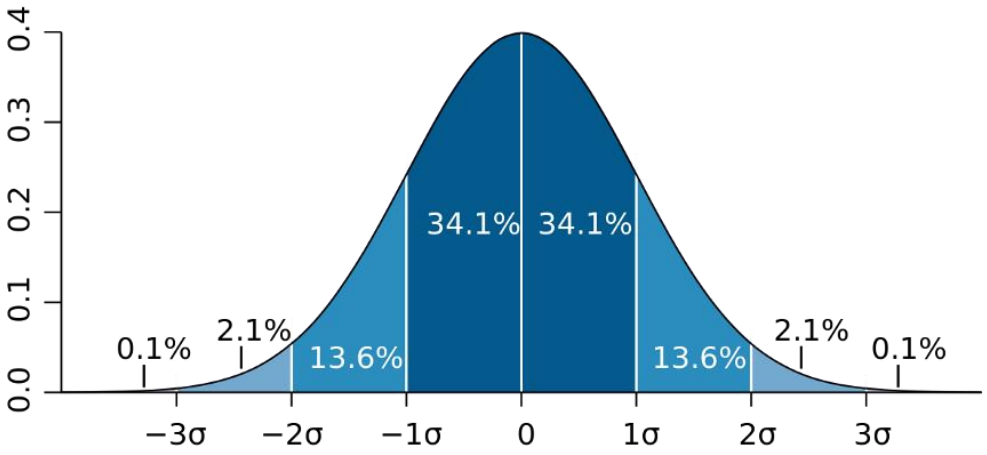


Figure 4.9. *Outlier definition in a normally distributed model [29]*

From the figure 4.9. we see a normally distributed model with mean 0 and standard deviation 1. We perfectly know that 99.7% of the data points are placed between -3 and 3 standard deviation away from the mean. Here is the point where we can define the outliers. Every data point which do not fall inside that range is considered as an outlier. Then the identification process is done using different techniques where the most famous ones are:

A. Identifying outliers with interquartile range (IQR)

IQR is one of the most commonly used techniques for outlier detection. It functions based on the following assumption that the data distribution is skewed which means that the left and the right part of the graph are not symmetrical, unlike the normal distribution. Saying that, in summary it works as in the following: the interquartile range (IQR) is a measure of statistical dispersion and is calculated as the difference between the 75th and 25th percentiles. It is represented by the formula $IQR = Q3 - Q1$. Then by using these IQR scores we can detect the outliers (for more details see the coding part).

B. Identifying outliers via z-score

Differently from the previously explained IQR method, this one makes the assumption that the data follow a normal (Gaussian) distribution. This is easy to check with the *skewness value*, which explains the extent to which the data is normally distributed. Ideally, the skewness value should be between -1 and +1, and any major deviation from this range indicates the presence of extreme values or the so called outliers.

C. Identifying outliers with Visualization

Apart from the quantitative methods above, outliers can also be viewed with plots. The plot types used may be “Box Plot”, “Histogram”, “Scatterplot” etc. [17]

One of the best visualization methods to detect the outliers is via Box Plot. The idea of this method is as below:

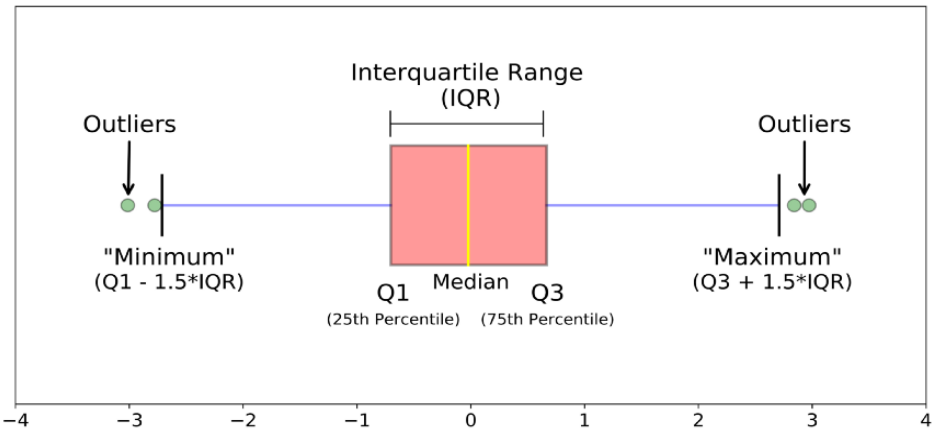


Figure 4.10: *Outlier detection via BoxPlot [29]*

As we see, the figure 4.10. has got many components and they can be explained as below:

- **Q1:** 25% of the data is below this data point.
- **Median:** The central value of the data set. It can also be represented as Q2. 50% of the data is below this data point.
- **Q3:** 75% of the data is below this data point.
- **Minimum:** The data point with the smallest value in the data set that isn't an outlier.
- **Maximum:** The data point with the biggest value in the data set that isn't an outlier.
- **IQR:** Represents all the values between Q1 and Q3.

Once we have identified these outliers, the next problem to be solved is the way how to handle them. There are mainly three techniques to do so:

- **Flooring and Capping**: This outlier handling technique is nothing else but the replacement of the outliers which are lower than the minimum allowed value with that minimum one and the maximum outliers with the maximum allowed value. This is one among the suggested techniques.

- **Trimming**: This method stands for removing all the available outliers and not replacing them at all. One of the reason this technique is not recommended is because it leads to a decrease of the training dataset which as a consequence then decreases the performance of the model. Although, this is something to be observed once it is implemented in our dataset (see the next chapter).

- **Replacement with Mean, Median, mode** etc.: In this technique, we replace the extreme values with the mode value, also we can use median or mean value but it is advised not to use the mean values because it is highly susceptible to outliers.

4.3.4. Feature Selection Methods

Machine learning works on a simple rule – if you put garbage in, you will only get garbage to come out. By garbage here, it means noise in data. In order to escape from these "garbage" we need not use every feature at your disposal for creating an algorithm. We can assist our algorithm by feeding in only those features that are really important [18].

For this reason, in order to improve the accuracy of the model, one of the most effective methods to be used is feature selection. **Feature selection techniques** help us in finding the smallest set of features which produces the significant model fit. Feature selection has the following objectives:

a) It **eliminates irrelevant and noisy features** by keeping the ones with minimum redundancy and maximum relevance to the target variable.
b) It **reduces the computational time and complexity** of training and testing a classifier, so it results in more cost-effective models.
c) It **improves learning algorithms' performance**, avoids overfitting, and helps to create better general models.

There are various feature selection methods which can be categorized into 3 main groups:

1. ***Filter methods***

Figure 4.11: *Filter methods steps [18]*

Filter methods are generally used as a preprocessing step. The selection of features is independent of any machine learning algorithms. Instead, features are selected on the basis of their scores in various statistical tests for their correlation.

Table 4.1: *Feature / Response correlation [18]*

Feature / Response	Continuous	Categorical
Continuous	Pearson's Correlation	LDA
Categorical	Annova f-value	Chi-Square

2. ***Wrapper Methods***

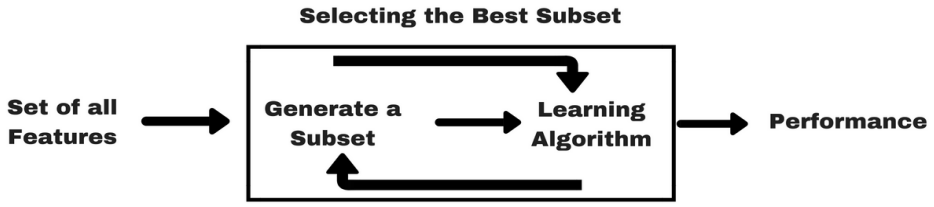


Figure 4.12: *Wrapper methods steps [18]*

In wrapper methods we try to use a subset of features and train a model using them. As we see from the figure above based on selection of subset we decide to add or remove features from the subset depending on the performance (which one is the best among them).
Some common examples of wrapper methods are:

- **Forward Selection:** Forward selection is an iterative method in which we start with having no feature in the model. In each iteration, we keep adding the feature which best improves our model till an addition of a new variable does not improve the performance of the model.
- **Backward Elimination:** In backward elimination, we start with all the features and removes the least significant feature at each iteration which improves the performance of the model. We repeat this until no improvement is observed on removal of features.
- **Recursive Feature elimination:** It is a greedy optimization algorithm which aims to find the best performing feature subset. It repeatedly creates models and keeps aside the best or the worst performing feature at each iteration. It constructs the next model with the left features until all the features are exhausted. It then ranks the features based on the order of their elimination.

The main differences between filter and wrapper methods for feature selection are:

- Filter methods measure the relevance of features by their correlation with dependent variable while wrapper methods measure the usefulness of a subset of feature by actually training a model on it.
- Filter methods are much faster compared to wrapper methods as they do not involve training the models. On the other hand, wrapper methods are computationally very expensive as well.
- Filter methods use statistical methods for evaluation of a subset of features while wrapper methods use cross validation.
- Filter methods might fail to find the best subset of features in many occasions but wrapper methods can always provide the best subset of features.
- Using the subset of features from the wrapper methods make the model more prone to overfitting as compared to using subset of features from the filter methods.

3. ***Embedded Methods:***

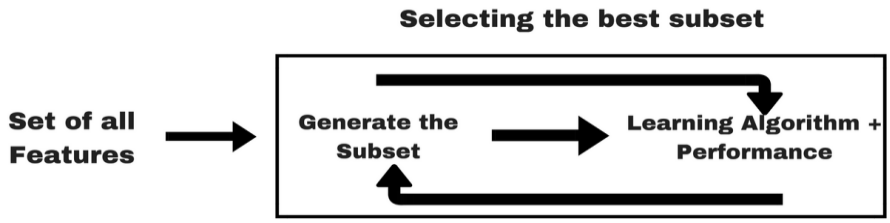


Figure 4.13: *Embedded methods steps [18]*

Embedded methods combine the qualities of filter and wrapper methods. It's implemented by algorithms that have their own built-in feature selection methods.

Some of the most popular examples of these methods are **LASSO** and **RIDGE** regression which have inbuilt penalization functions to reduce overfitting.

- Lasso regression performs L1 regularization which adds a penalty equivalent to absolute value of the magnitude of coefficients.
- Ridge regression performs L2 regularization which adds a penalty equivalent to the square of the magnitude of coefficients [18][20].

After having introduced the main feature selection methods, the following question may be asked: "What is the best feature selection method?". The answer to this is very simple, there is no best feature selection method. Just like there is no best set of input variables or best machine learning algorithm, at least not universally. Instead, it must discover what works best for a specific problem using careful systematic experimentation. The methodology to be used is by trying a range of

different models fit on different subsets of features chosen via different statistical measures and discover what works best for your specific problem.

4.3.5. Learning Curve

It's obvious that when we built a model, we want it to perform as accurate as possible with lowest possible errors. We also know that the main sources of errors are variance and bias. Bias is the assumption made by the model to predict the target values, so to approximate it as much as possible whereas variance is the difference between target function estimations when applied to different data, so how much it changes.

4.3.5.1. Bias – Variance Trade-Off

Saying that, our main goal should be to find the trade-off between those 2 metrics, bias and variance. Obviously, the ideal scenario would be having low bias and low variance. However, we are aware that low-biased method captures most of the differences (even the minor ones) between the different training sets. Therefore, the estimation function *varies* a lot as we change training sets, and this indicates high variance.

The less biased a method, the greater its ability to fit data well. The greater this ability, the higher the variance. Hence, the lower the bias, the greater the variance which means that finding a trade-off will not be trivial. The reverse also holds: the greater the bias, the lower the variance. A high-bias method builds simplistic models that generally don't fit well training data. As we change training sets, the estimation models we get from a high-bias algorithm are, generally, not very different from one another [28].

Usually what we want is having a low variance to avoid building an overly complex model. Such a model fits almost perfectly all the data points in the training set. Training data, however, generally contains noise and is only a sample from a much larger population. An overly complex model captures that noise. And when tested on *out-of-sample* data, the performance is usually poor. That's because the model learns the *sample* training data too well. It knows a lot about something and little about anything else. This is also called *overfitting* in machine learning language. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models ability to generalize. The opposite of overfitting is underfitting which is refers to a model that can neither model the training data nor generalize to new data. An underfit model is not suitable as it'll have a poor performance on the training data.

So to summarize, as anticipated a trade-off between bias and variance is needed. For this purpose, we may use learning curves.

4.3.5.2. Introduction to Learning Curves

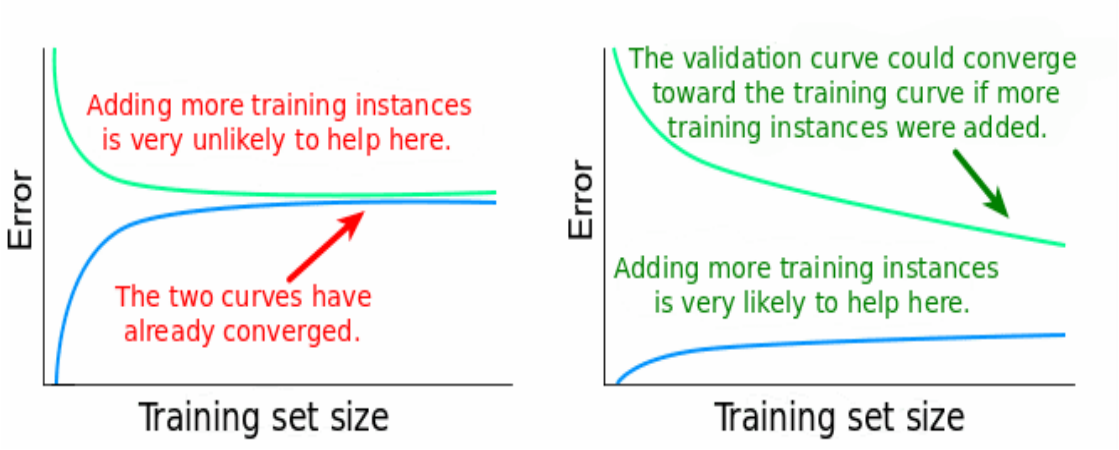


Figure 4.14 *Learning curve idea [28]*

A learning curve shows how error changes as the training set size increases. Learning curve can help us in understanding whether adding more data would help in performance increase or not. This is perfectly explained in the graphs below:

So the message aimed to be given from the first graph in **Figure 4.11** is obviously not to lose time in adding more training data after the point where training score error (blue curve) and validation score error (green curve) are so close to each other. Whereas, in the second graph it is told to add as many training data as possible until the two curves merge (converge) in one point.
Now let's have a look at how we can define bias and variance from the learning curve. The general idea is as following: If the training error is very low, it means that the training data is fitted very well by the estimated model. So it means that it has *low* bias with respect to that set of data. If the training error is high, it means that the training data is not fitted well enough by the estimated model and it has *high* bias with respect to that set of data. The graphs below clearly present what we just explained:

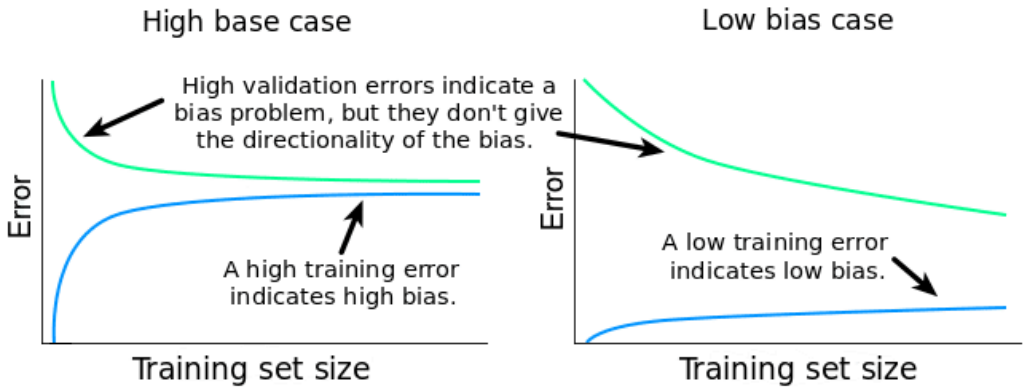


Figure 4.15. *Bias definition in Learning Curves [28]*

Regarding the variance, its diagnosis can be done in two ways:

- By examining the gap between the validation learning curve and training learning curve.
- By examining the training error: its value and its evolution as the training set sizes increase.

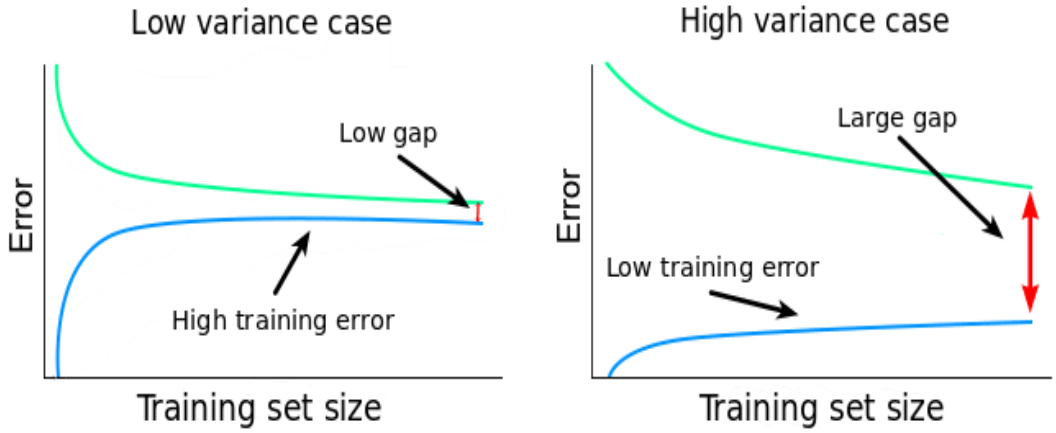


Figure 4.16. *Variance diagnosis in Learning Curves [28]*

Obviously ***gap=validation error – training error*** .

So the bigger the difference between the two errors, the bigger the gap. The bigger the gap, the bigger the variance. In our case, the gap is very narrow, so we can safely conclude that the variance is low. High *training* MSE scores are also a quick way to detect low variance. If the variance of a learning algorithm is low, then the algorithm will come up with simplistic and similar models as we change the

training sets. Because the models are overly simplified, they cannot even fit the training data well (they *underfit* the data). So we should expect high training MSEs. Hence, high training MSEs can be used as indicators of low variance.

Last but not least, we can ask a question to ourselves like: “What is the best learning curve”? The answer to this question is that this is not possible, neither in practice nor in theory due to a parameter called “irreducible error”. It’s not that important to go very deeply into details but just knowing that there’s no way to know the true value of the irreducible error based on the data we have. For this reason, there is no perfect learning curve but by defining the important parameters such as bias, variance etc., from it we can get close to a good result. Now we’ll implement it in our dataset and the related results will be explained in the next chapter when introducing also other results.

Chapter 5

Results

In this section, the best results obtained from the implemented ML algorithms as well as the implementation of the methods described in the previous chapters are presented. Before going deeply into the predictions for each target field separately, first let's have a look at a common issue effecting all the 3 target metrics, called outliers. As introduced in the previous chapter, outliers are data points that differ significantly from other observations and need to be identified in order to treat them as required, otherwise they will negatively affect our model. In order to identify them, we have used the 3rd method described in the previous chapter, via visualization, more precisely via Box Plot. The figure 5.1 shows the boxplots representing the distributions of some features (having the same magnitude) where the values above the maximum and below the minimum are called outliers.

From that figure in the next page, we notice a significant number of outliers, especially for the features "*interarrival_mean*" and "*interlength_udp_max_min_R*" as well as the case where no outliers are present (i.e. "*len_udp_min_max_R*" feature). Saying that, our expectations are towards having a negative impact of the outliers on our model and once they are handled well (removed or substituted with the correct values of each field), the model's performance is expected to be improved. This process of handling the outliers is the so called "cleaning of the dataset" process, introduced in the step-3 of our methodology, in Chapter-1.

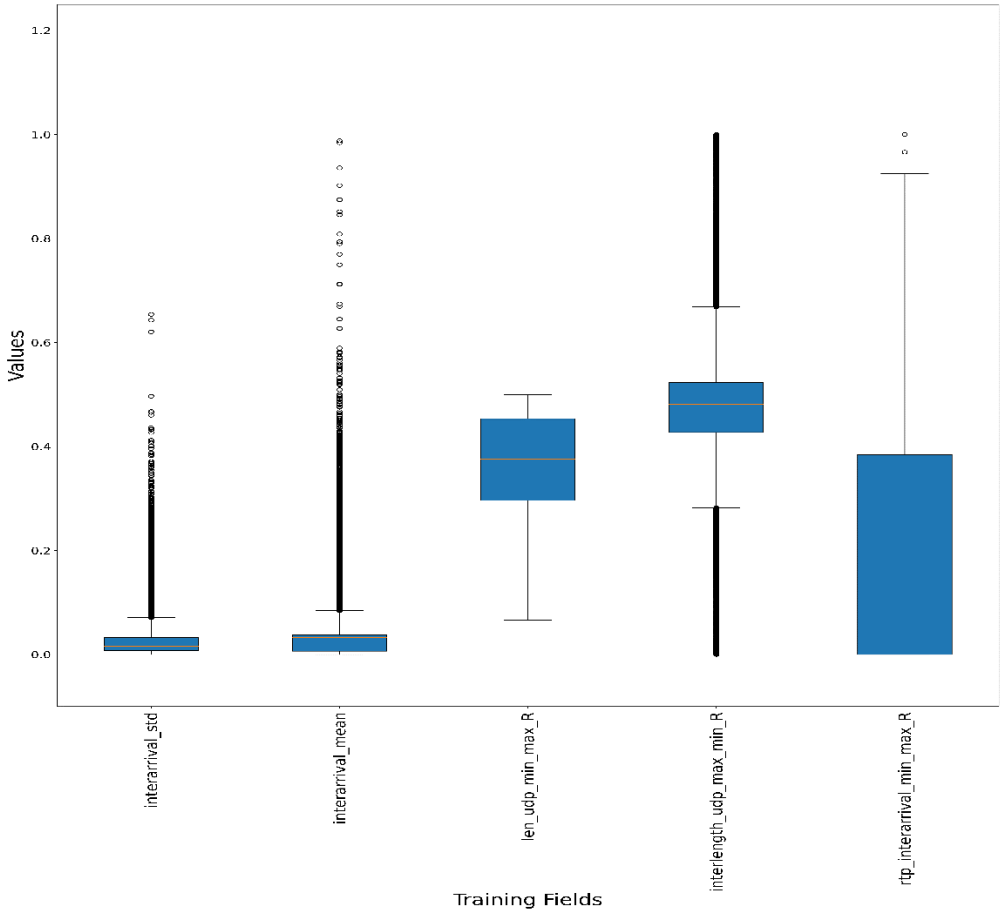


Figure 5.1: *Box Plot view of some training fields*

5.1 Resolution Prediction

Now let's have a look at all the 3 target metrics one-by-one, starting from the resolution. As anticipated, the resolution in our dataset is represented by the field "*frameWidth*" in our dataset which is simply the "size in pixels of the received video frames" and this is one of the 3 QoE metrics which we will try to predict. As for the training dataset, at the beginning, all the QoS metrics (the 60 fields introduced in the Figure 4.4) were used. These are features measured from the vantage points of the network (so routers, switches etc.,) using the Wireshark tool, as anticipated. Saying that, the technique used for this prediction is the one via Classification. If we go back to the Figure 4.7, where the distribution of the "frameWidth" fied is shown, we notice that there are many classes, and trying to predict them as they are, would for sure, not give the results we expect to obtain. The reasons behind this scenario are many, first of all the data are unclean, which means that there are classes which appear in training dataset and not in the testing one and vice-versa, and this happens because we use different pcaps in this experimentation study and the training and test sets are randomly chosen. Another issue we would face in case we were using all the classes of the "*frameWidth*" field as they are, is the variety in support (representation) for each class, where the ones with higher support would have the highest predictability and the ones with the lowest would be nearly unpredictable.
Saying that, the solutions presented for this problem are two, either droping the classes with lower supports by defining a threshold, or grouping adjecent classes together and form some new ones. In the following, I will be presenting the results regarding both solutions and we'll discuss why the one is better than the other.

The first trial was about removing the classes and the data corresponding to "*frameWidth*" field with less than 1000 samples (supports). In this way, only the classes above this threshold exceeding this amount of samples will contribute and as consequence it is expected to increase the model's performance. The table 5.1 below, somehow reflects our expectations:

Table 5.1: *Results after removing values below a thresholder*

Algorithm	Feature Selection method	Outliers processed	Result (F-1 score)
Decision Tree	NO	NO	0.78
Random Forest	NO	NO	0.79
XG Boost	NO	NO	0.77

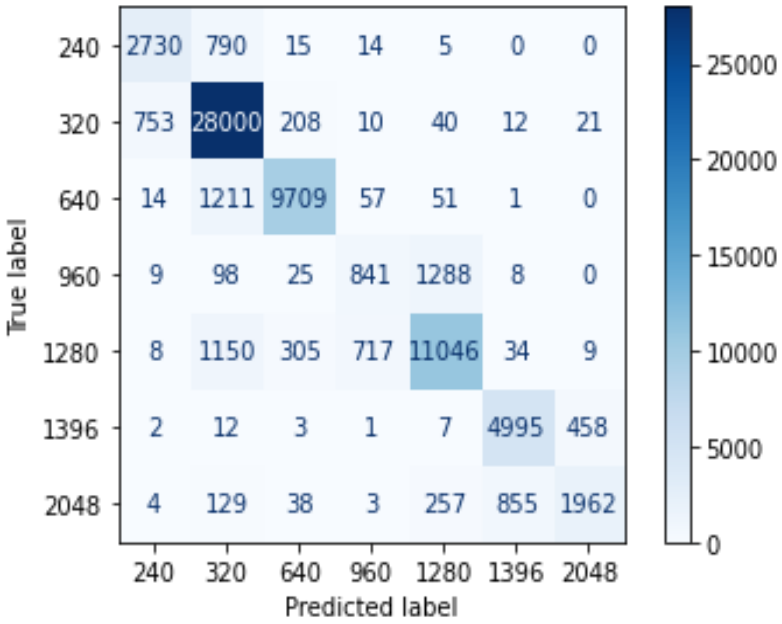


Figure 5.2: *Confusion matrix for Decision Tree method*

Despite those results, we are still far away from our expectations and this happened because even though we removed the classes below a threshold, there are classes which have a huge number of support compared to those near to 1000. The figure 5.2 clearly corfims that, where the class of 320-pixels has the highest number of support (more than 28000 samples) and as a cosequence is the best predicted one, unlike 960-pixels one which has the lowest support (around 2000) among all and as a result it is the worst predicted one.

Therefore the next step was to try another method, the one which groups the adjacent resolution types with each other and form new classes which balance the model. The classes are grouped together and re-named as in the table 5.2:

Table 5.2: *New classes after the grouping process*

Existing Classes Groups	New Formed Classes
240 - 480	1
480 - 960	2
960 - 1280	3
> 1280	4

So, with this grouping method we have created a kind of MOS classes between 1-4, which means that the values to be predicted will no more be those in the left of the table but those in the right, so from 1 to 4. Saying that, once having trained the model with this new dataset, the following results have been obtained:

Table 5.3: *Results after the grouping process*

Algorithm	Feature Selection method	Outliers removed	Result (F-1 score)
Decision Tree	NO	NO	0.90
Random Forest	NO	NO	0.91
XGBoost	NO	NO	0.91

Finally, according to the table 5.3, we started to see some promising results which we expect to improve even more when using additional performance improvement techniques such as feature selection or outlier processing etc.

Before implementing any method in order to increase the model's performance, we would like to see the level of bias and variance in our model by using "Learning Curve". The learning curve function in python exploits the training dataset sizes defined in prior in order to build its curve alongside the graph. So, the training dataset sizes were defined as following: [1, 1000, 10000, 35000, 75000, 100000]. Based on those training sizes the learning curve function will split the data into training and validation, and will issue some outputs such as accuracy for classification and MSE for regression which will then be plotted in a graph. The graph in the Figure 5.3 shows the first results regarding Learning Curve obtained using Decision Tree Classifier to fit the model:

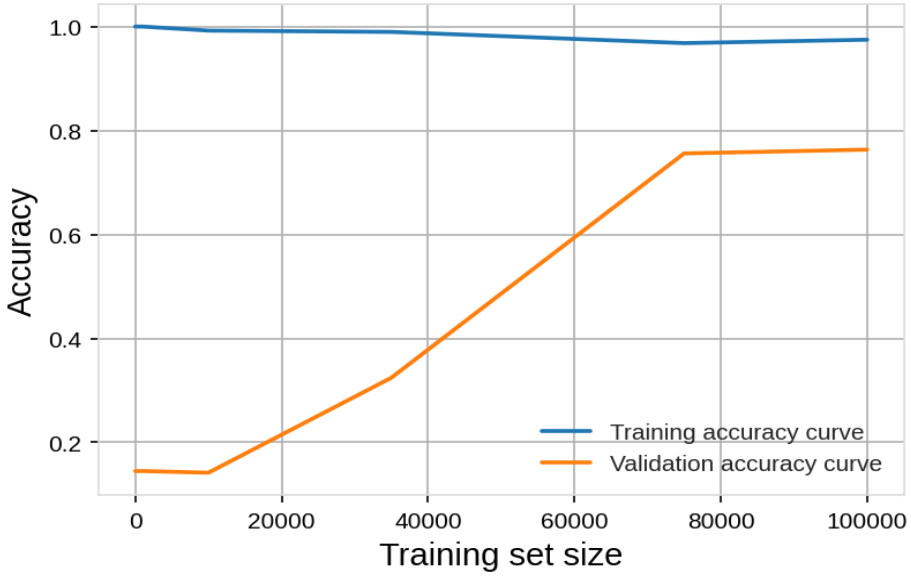


Figure 5.3: *Learning curve obtained for DecisionTreeClassifier*

The Figure 5.3 explicitly shows the behavior of the training and validation scores (in this case "accuracy") among different training set sizes. Recalling what we explained in the previous

chapter when introducing this topic, we said that the bias is the line defining the training error which in this case seems to be near to 1. According to the knowledge acquired so far, this indicates a low bias which is one of our desired parameters we are looking at. Then regarding the variance, one of the methods was to look at the gap between training and validation score lines. The gap between those lines in the figure 5.3. is not so small, which means the variance is a bit high. However, another observation is that after 75000 indexes of training dataset the lines are developing in parallel which tells us a very important conclusion which is that the increase of the number of dataset most probably would not lead to any increase in the accuracy. Therefore, other methods such as feature selection and/or outliers processing etc., should be taken into consideration instead.

From now on, our goal is to increase the accuracy as much as possible by using different methods combined with different algorithms. As anticipated, there isn't any best method or algorithm in machine learning but it all depends on the type of dataset and the model it will be used.

Saying that, the first trials were towards identifying and treating the outliers or basically cleaning the data. As anticipated, outliers play a significant role in the model's accuracy improvement and treating them wisely could help in the improvement of model's performance. Note that the method used for outlier processing is called IQR (Inter Quartile Range) and after their identification, the *flooring and capping* method was used for their treatment.

Beside the outlier process, it was also used the technique of feature selection in combination with that. In the table 5.4 below, we present some of the best results obtained by using the feature selection methods such as Extra Tree, Information Gain, Correlation Coefficient and Annova f-value, combined with outlier processing.

Table 5.4: *Results regarding resolution prediction*

Algorithm	Training class (grouping or reducing)	Feature Selection method	Outliers processed	Results (F-1 score)
Random Forest	grouping	InfoGain	YES	0.91
Random Forest	grouping	Annova f-value	YES	0.91
XGBoost	grouping	Annova f-value	YES	0.91

Basically these are the most important results obtained for this metric. Unfortunately, we didn't notice any significant improvement regarding the resolution prediction when applying feature selection and processing the outliers. For this reason, let's have a look at the confusion matrix

regarding the best result which is for the combination of the techniques in the first line in the table 5.4 above.

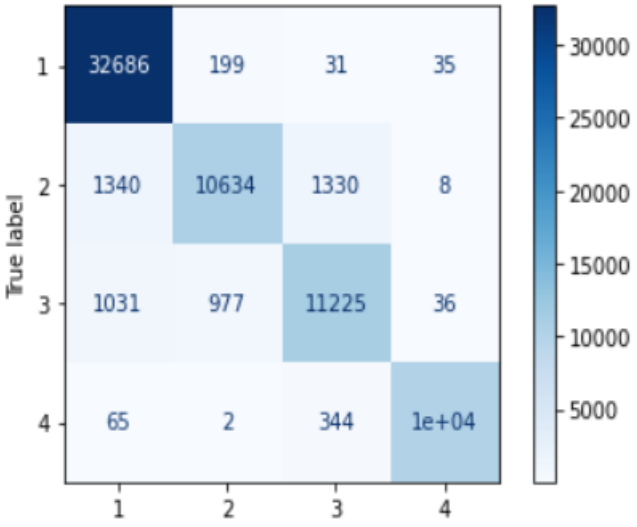


Figure 5.4: *Confusion matrix for resolution prediction*

The figure 5.4 gives us a clearer scenario of how the classes are predicted and where are the mistakes made by the model. Class "1" has the highest number of the supports which makes it well predictable due to the large number of data used for training, as a consequence we achieve a very good f-1 score value , around 0.96. Then the 2nd and the 3rd classes have almost the same number of support and most errors are made towards the 1st class which is somehow expected because it has the number of support as almost all other 3 classes combined together, and this makes it kind of object to overfitting.

Moreover, the 4th class has a better predictability with around 0.98 of f-1 score where most of the confusions here are with the previous adjacent class which is class-3. Despite that, this class has the highest predictability and the reason behind this is because this class, which contains resolution values starting from 1280 pixels and higher, represents screensharing in the communication session. So this is the main reason why this class is much more difficult to be confused with other data having lower values.

Therefore, as a brief summary about this section, we can conclude that the best results were obtained when using "Random Forest" ML algorithm combined with "InfoGain" or "Annova f-value" feature selection methods and outliers being detected and treated via "*IQR"* and *"Flooring and Capping*" methods respectively. Then it follows up the one via "XGBoost" and so on.

5.2 Smoothness Prediction

Now, let's have a look at our 2nd target called "Smoothness". By smoothness it is meant the number of frames per second which clearly directs us to use the field "framesPerSecond" in our dataset. Same as we did in the previous section for the resolution, at the beginning all possible fields of QoS metrics which could be measured in the vantage points were used, except one field called "rtp_marker_sum_check" which indicates the frames per second itself. Unlike the "classification" technique used to predict the resolution, here we use "regression". So by training our model with the data from those fields and as a target the field indicating the **frames per second** extracted from the logs of the application, we can now present the very first results obtained firstly without any additional performance improvement method of feature selection or outlier processing.

Table 5.5: *First results without any feature selection or outlier treatment method*

Algorithm	Feature Selection method	Outliers processed	Score (R-squared/MAE)
Decision Tree	NO	NO	R^2= 0.84 / MAE = 1.65
Random Forest	NO	NO	R^2= 0.86 / MAE = 1.43

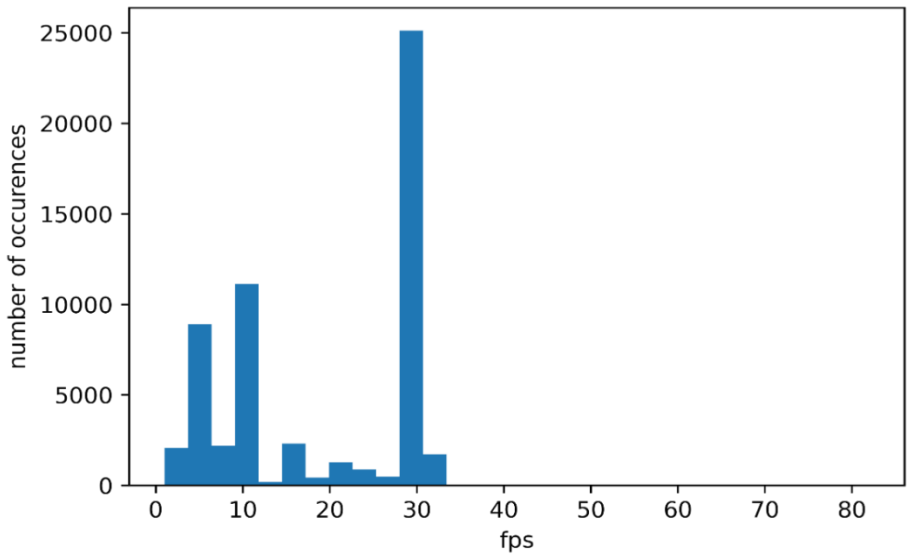


Figure 5.5: *The distribution of "Frames per Second" field in the testing dataset*

By simply applying only the most useful two algorithms such as Decision Tree and Random Forest we obtain some promising results which are Mean Absolute Error of 1.65 and 1.43 respectively. With respect to the distribution of the "framesPerSecond" field's values (shown in the **figure 5.5**),

the mean absolute error score is not that high, given that the most frequent samples oscillate around 30. Then, the second most frequent "framesPerSecond" value is 10 which may be a bit risky with an MAE value of 1.43 (the best case among two). However, the expectations are towards having a considerable improvement when applying feature selection methods and/or removing outliers.

Before that, similarly to what was done in resolution prediction, we'll implement learning curve in order to see if any addition of training set would be useful or we can directly move to other methods such as feature selection etc. Again, the idea followed is the same as the one implemented in resolution prediction, where we define 6 different training sizes in an increasing order. The results obtained are as following:

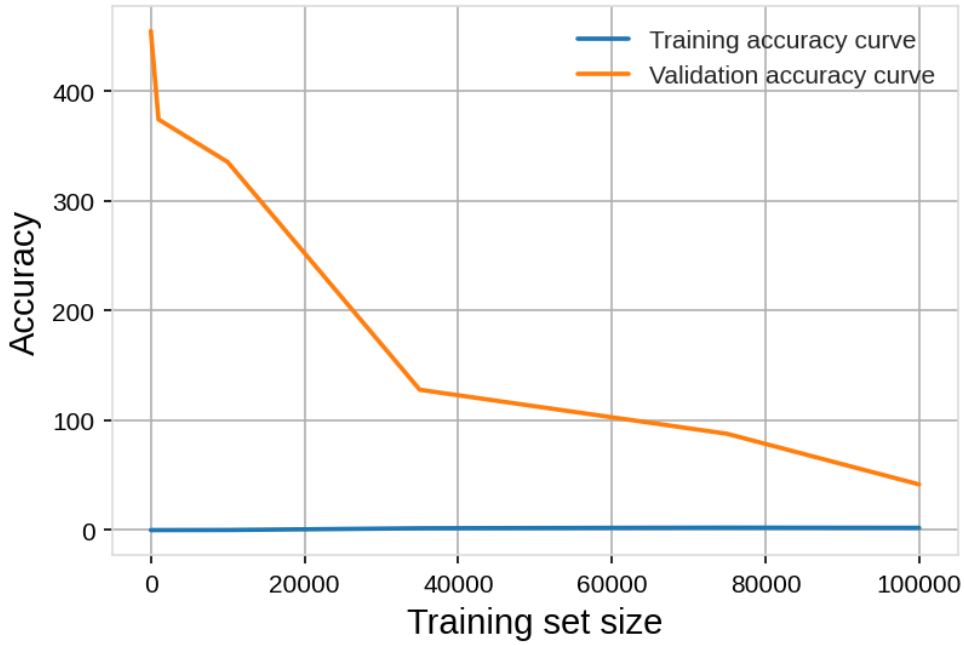


Figure 5.6: *Learning curve obtained for DecisionTreeRegressor*

Differently from the curves in resolution prediction, here the reference scoring parameters will no longer be "accuracy" but "negative MSE" which is flipped to positive by being multiplied with minus (-) sign since the required parameter is the positive MSE. Saying that, the results are explicitly shown in the **Figure 5.6.** The first observation is the decrease of validation error which is quite expected due to the fact that training set size is continuously increasing and more data we have for training the model, the better it will be trained and the less errors will be when predicting never seen data. From the training score, we observe a very low error which clearly indicates how low the bias is. Regarding the variance which is represented by the gap between 2 curves (validation and training errors) we can say that it keeps lowering sharply until it reaches a considerable number of training set. However, I would say that the increase of the training set size

would narrow the gap even more but not at that point they become so close to each other. For this reason, it would be more effective looking at other methods such as feature selection rather than trying with more training data as it would lead to time wasting.

Before that, we tried to clean the data by identifying and processing the outliers with IQR technique and treat them using "flooring and capping" method

Later on, there were applied different feature selection methods such as Pearson correlation, Coefficient correlation, Information gain, Annova f-value etc,. Then, a combination with and without outliers was made in order to see the effects and the differences between each other. The table below summarizes this situation by showing the algorithms and feature selection methods performing the best among them:

Table 5.6: *Results after the outliers processing*

Algorithm	Feature Selection method	Outliers removed	Score (R-squared/MAE)
Decision Tree	Annova f-value	YES	R^2= 0.84 / MAE = 1.64
Random Forest	Pearson	YES	R^2= 0.865 / MAE = 1.43
Random Forest	Correlation Coefficient	YES	R^2= 0.865 / MAE = 1.43

Indeed, there is a very slight improvement compared to the case where the outlier removal and feature selection techniques were not used. Obviously we see that Random Forest algorithm combined with the Correlation Coefficient feature selection method and the right treatment of outliers gives the best results so far with a Mean Absolute Error equal to 1.431 and Mean Squared Error 14.99. Referring to the **figure 5.6.,** where the values mainly vary around 30 we observe some promising result.

5.3. Concealment Prediction

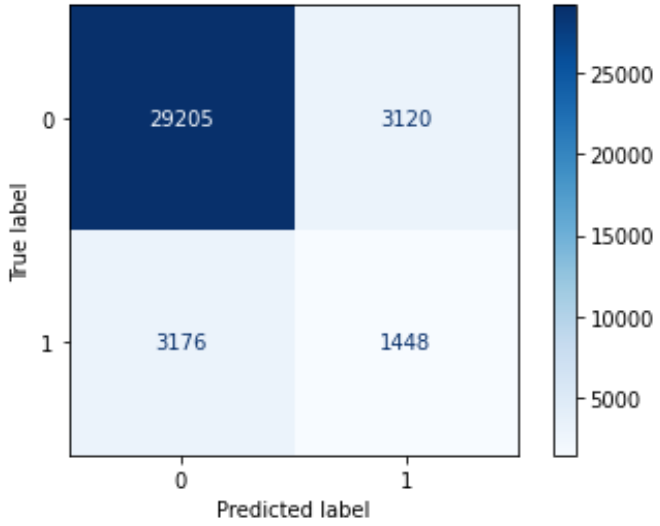
Last but not least, let's have a look at our 3rd target, called "concealment". As previously introduced, concealment is the time spent by the decoder rendering frames that are partially received, so it has a strong relationship with packet losses. At the same time, this feature is the most difficult one to be predicted among the three, due to its uncertainty. Because of its high correlation with packet losses, for video concealment the field used as a target in our dataset is the "framesDropped" feature whereas on the other hand, for audio concealment there is used the "concealmentEvents" feature. Apart from the prediction difficulties, also the procedure of data preprocessing is more complicated compared to the previous fields' one. Initially, we needed to take some actions such as distinctions among video and audio concealment values, extracting the contiguous values of the target fields from each other since they were in a cumulative form and so on. Finally, once the data were correctly organized, it was started with training the model. This problem is more similar to our 1st prediction problem regarding resolution, because also here we use classification to predict whether there is any concealment or not. Actually we would like to predict the exact value of the concealment but this is extremely difficult and it can be think of it as a future work.

Saying this, the following results are giving us a first overview on how the results for audio and video concealments look like:

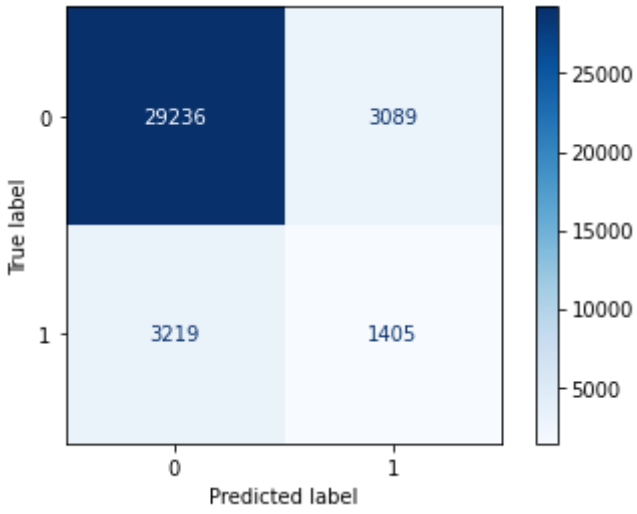
Table 5.7: *Results regarding audio and video concealments*

Concealment Type	Algorithm used	F-1 Score
Audio Concealment	Decision Tree	0.61
Audio Concealment	Random Forest	0.61
Video Concealment	Decision Tree	0.58
Video Concealment	Random Forest	0.58

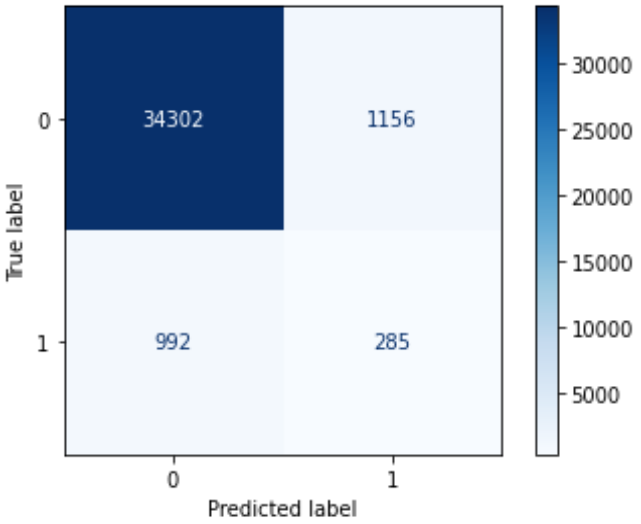
AUDIO CONCEALMENT (Decision Tree):



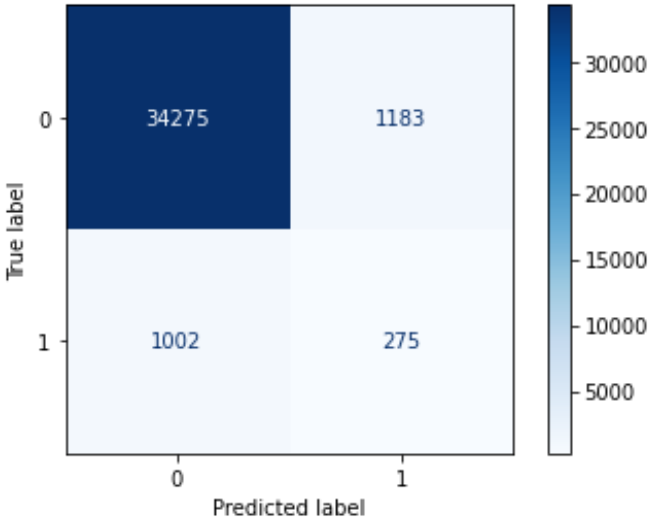
AUDIO CONCEALMENT (Random Forest):



VIDEO CONCEALMENT (Decision Tree):



VIDEO CONCEALMENT (Random Forest):



First of all, note that here we have only 2 classes: 0 (zero) and 1 (one). 0 (zero) represents the NO CONCEALMENT case whereas 1 (one) the opposite. Saying that, what we observe from those results is that due to the low number of data representation for the case where concealments are present, the performance is low which is what it is expected in those cases. So the main reason of obtaining such low results is due to the difference of the support numbers between 2 classes.

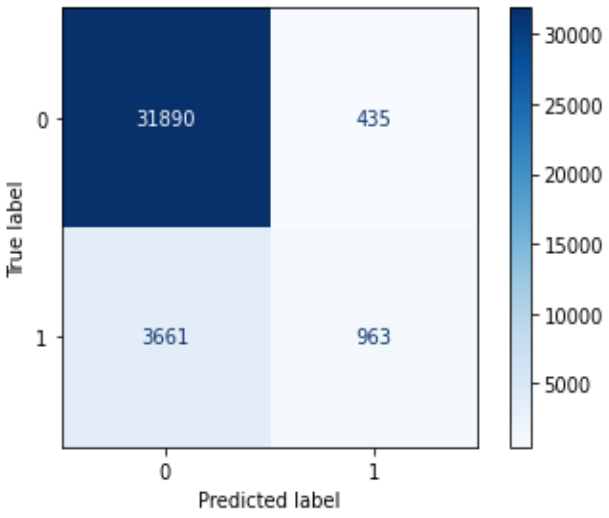
Later on, aiming to improve the performance, same as we did for the previous 2 targets, we tried to use different feature selection and outlier treatment methods and below we present the best results obtained for each method combination:

Table 5.8: *Results regarding audio and video concealments when additional methods applied*

Concealment Type	Algorithm	Outlier processing meth.	Feature selection meth.	F-1 Score
Audio Concealment	Random Forest	IQR/flooring and capping	Correlation Coeff.	0.63
Audio Concealment	Random Forest	IQR/flooring and capping	Annova f-value	0.64
Video Concealment	Decision Tree	IQR/flooring and capping	Annova f-value	0.6
Video Concealment	Decision Tree	IQR/flooring and capping	Information Gain	0.58

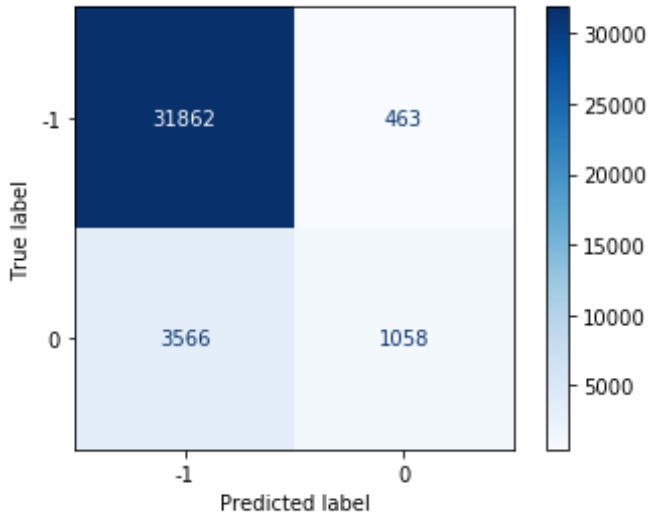
AUDIO CONCEALMENT:

- Random Forest
- Correlation Coeff.



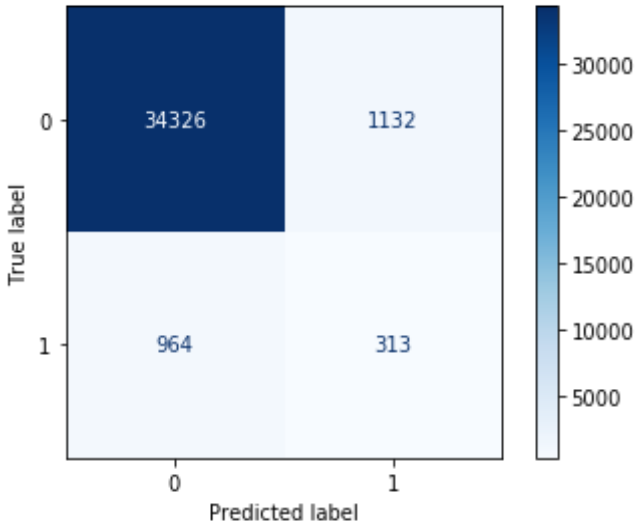
AUDIOCONCEALMENT:

- Random Forest
- Annova f-value



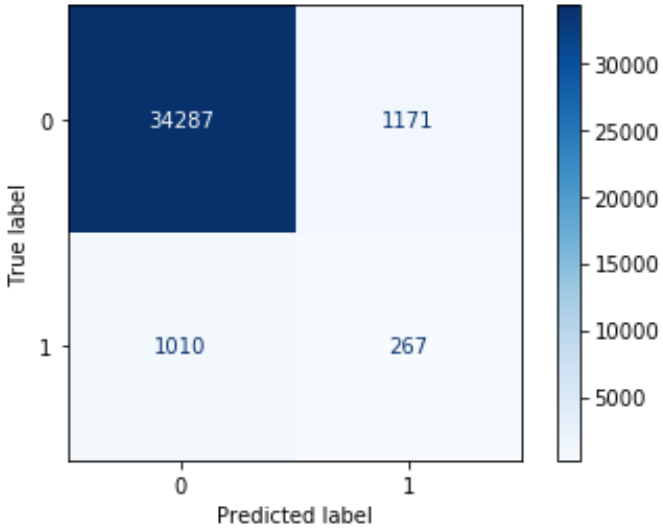
VIDEO CONCEALMENT:

- Decision Tree
- Annova f-value



VIDEO CONCEALMENT:

- Decision Tree
- Info Gain



Basically those are the best results obtained for audio and video concealment prediction. We observe a low improvement with respect to the previous case where no additional methods were applied. However, we can briefly conclude that the low support for concealment case with respect to the no concealment case, is the main factor leading to such a low predictability.

Chapter 6

Cloud Gaming Scenario

In this chapter, we explain the implementation of the same approach to some other data type collected from another RTC application. In particular, the scenario which the data are collected from is of Cloud Gaming. The applications on the other hand are “Stadia” and “GeForce Now”. The main goal here is to evaluate the performance of the methodology developed for online meeting applications such as Jitsi or WebEx to other application types, so that the method is not specific to a particular application but it can have a wide range of usage.

6.1 Dataset description

In this section, we give a brief overview on the distribution of the fields since the training and testing fields used for this experimentation remain the same. So, as a quick recap we have 3 target values as following: “framesPerSecond” representing the resolution, “frameWidth” representing the smoothness and “conceamentEvents” and “framesDropped” representing audio and video concealment respectively. On the other hand, the training fields are as same as those introduced in the previous chapter, where they were grouped into 4 main categories: “**Inter-arrival**”, “**Packet length (UDP)”, “Inter packet length**”, “**RTP inter-arrival”**. Regarding the amount of the collected traffic it’s worth mentioning that there are a 13 hours of traffic and 196 pcaps in this dataset.
After this brief introduction, now let’s have a look at the distribution of the target fields:

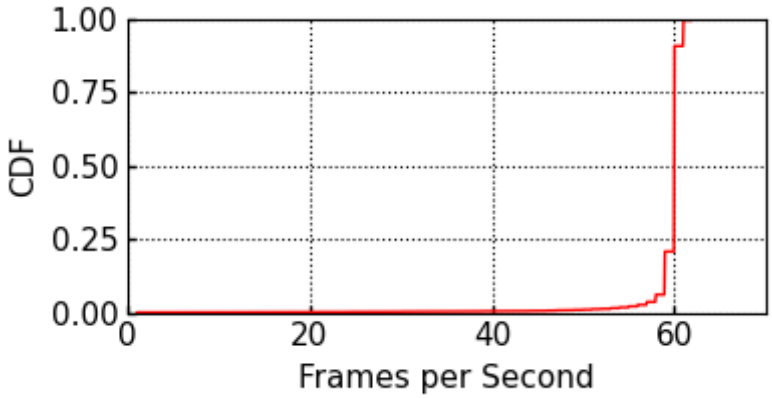


Figure 6.1: *“Frames per Second” field distribution*

Figure 6.1 gives us a clear picture of the distribution of the values in the framesPerSecond field. The most noticeable information here is that the most frequent values are those between 57 and 61. Also, this distribution tells us that the highest occurance is around 67 fps.

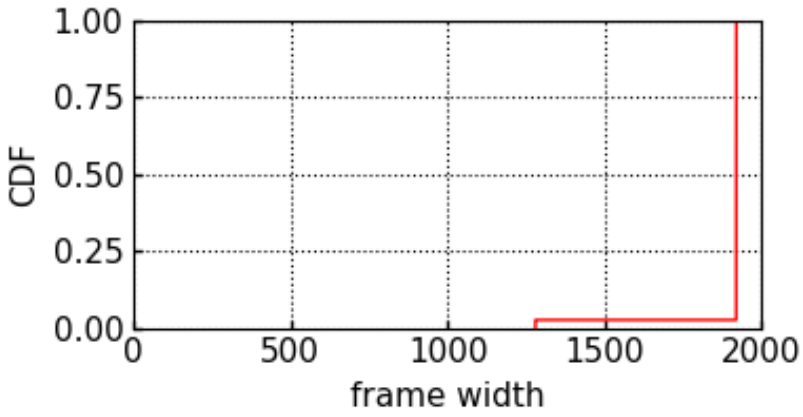


Figure 6.2: *"frame width" field distribution*

On the other hand, the second graph regarding the frameWidth field distribution in the Figure 6.2, simply indicates that we have only 2 classes of the framewidth, 1280 and 1920 pixels. The most frequent value seems to be the 1920 pixels one with around 97% of the whole data in this field. Basically, here we expect a pretty high prediction for the 1920 pixels class due to the simple fact that it has more data to be trained compared to the 1280 pixels one.

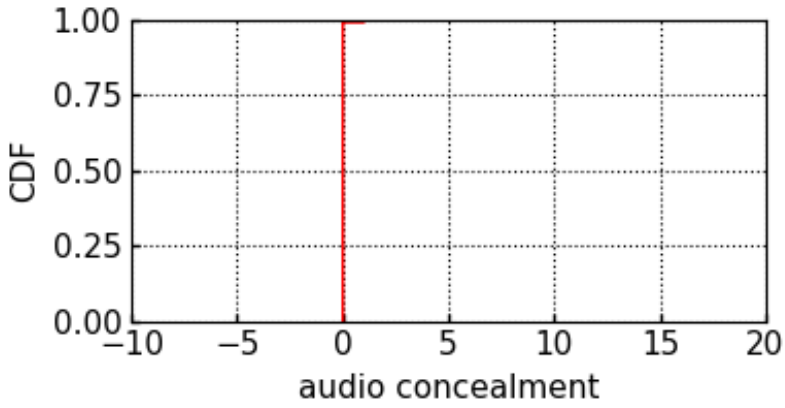


Figure 6.3: *"concealmentEvents" field (or audio concealment) distribution*

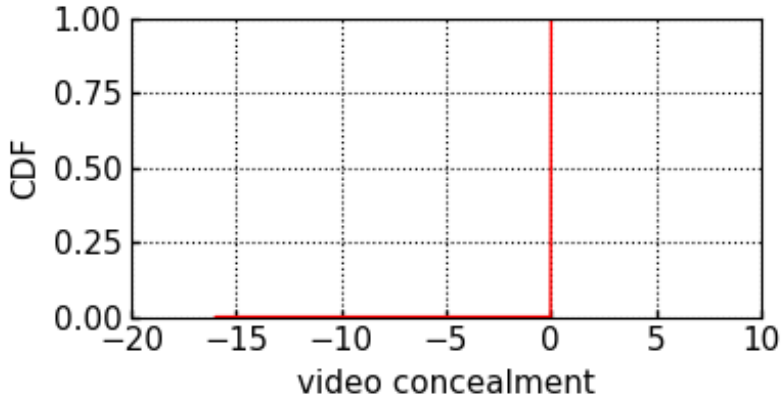


Figure 6.4: *"framesDropped" field (or video concealment) distribution*

Finally, regarding the distribution of the values in the concealments fields, we have shown it in the Figures 6.3 and 6.4. Note that, same as in the previous scenario, also here these values are in cumulative form and we needed to extract subsequent ones (part of the same flow), one from the other. However, the plots in the Figures 6.3 and 6.4 represent the exact values' distribution and not the cumulative form.

The common thing we notice in both graphs is that the major part of the dataset is composed by 0 (zero) class which at the same time represents the NO CONCEALMENT case whereas all other values represent the case where concealment is present and they have a very low number of support. Saying that, the expectations are towards having a high predictability of the class representing NO CONCEALMENT case since the number of support for this class is extremely high compared to the other class. This scenario for sure effects the overall f-1 score since for the other class we'll have low predictability. After having introduced the dataset and the target fields in detail, now let's go into the next section where we present the results obtained by this model.

6.2 Results

In this section, we simply present and comment the results for each target field.

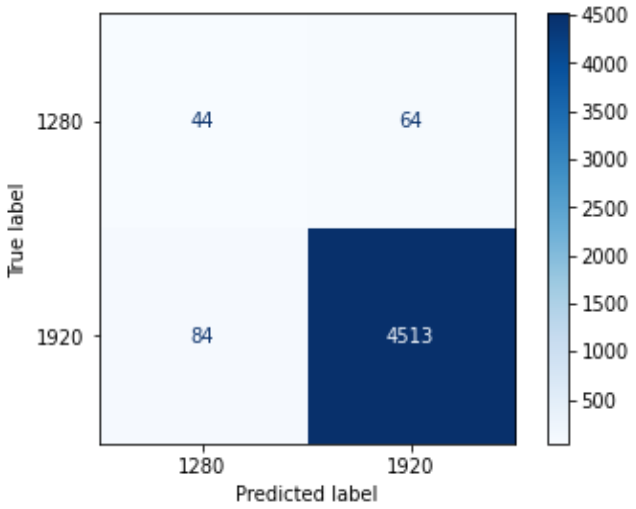
6.2.1 Resolution prediction

As anticipated, the resolution is represented by the field "frameWidth" and from the distribution graph we saw that we have only 2 classes. The technique used here is via Classification whereas the best results obtained for this prediction are shown below:

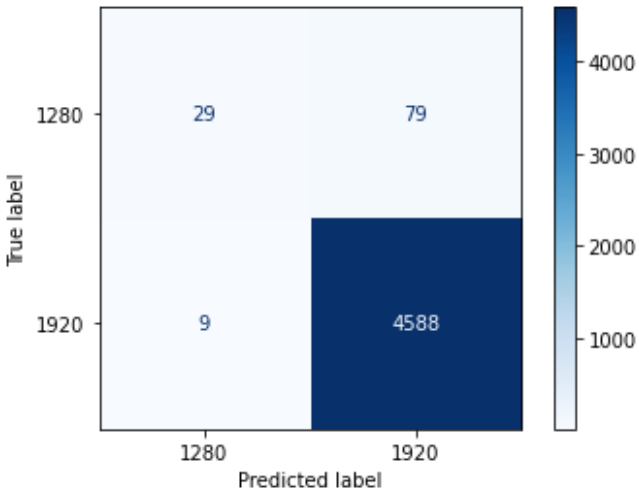
Table 6.1: *Results regarding resolution prediction for cloud gaming*

Algorithm	Feature selection method	Result (f-1 score)
Decision Tree	Correlation Coefficient	0.68
Random Forest	Annova f-value	0.69

Decision Tree with Corr. Coefficient method:



Random Forest with Annova f-value



According to these results, as expected, we observe a very high predictability for 1920 class due to the higher number of support and a low predictability for the 1280 pixels one. Basically, the huge gap between training datasets of each class is the main reason why we observe such results and if we increase the number of data, in particular for 1280 pixels class, we expect a significant improvement in our model's performance.

6.2.2 Smoothness prediction

Our second target is smoothness, which is represented by "framesPerSecond" field in our dataset. Unlike the previous target's prediction, here we use Regression method and the best results obtained for each algorithm are described the table 6.2:

Table 6.2: *Results regarding the smoothness prediction for cloud gaming*

Algorithm	Feature Selection method	Outliers processing meth.	Score (MAE/MSE)
Random Forest	Not applied	Not applied	MAE = 0.79 / MSE = 4.69
Random Forest	Annova f-value	IQR / flooring and capping	MAE = 0.79 / MSE = 4.74
Decision Tree	Not applied	Not applied	MAE = 1.02 / MSE = 10.64

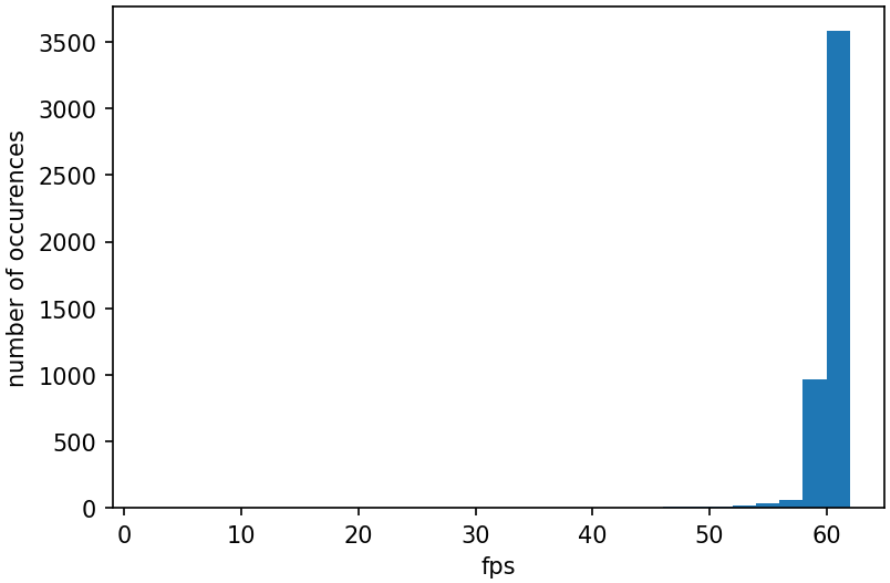


Figure 6.5: *"Frames per Second" field distribution in the bar graph*

Among the best results in the table 6.2 we have obtained a Mean Absolute Error of 0.79 which seems a very promising result compared to the distribution of the values illustrated in the Figure 6.5. This means that, having a majority of the values of 60, a mean absolute error of 0.79 is considered a pretty good result. So to conclude, for this target field our model has a satisfying performance.

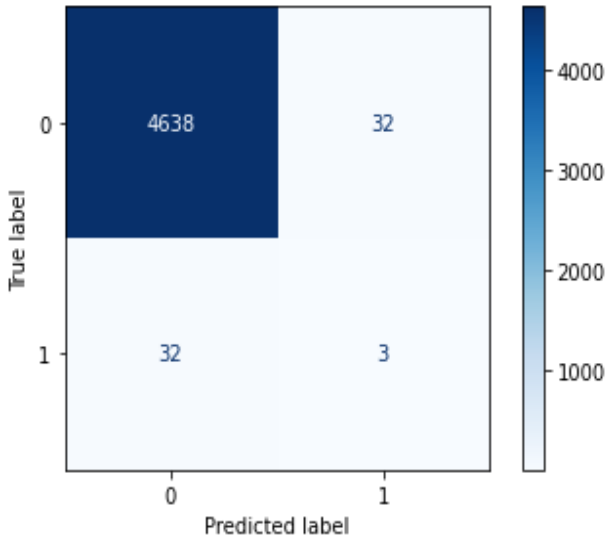
6.2.3 Concealment Prediction

Last but not least, in this section, we'll evaluate the model's performance when predicting the concealment of audio and video, represented by the "framesDropped" and "concealmentEvents" fields respectively. The method used here is similar to the one used in resolution prediction, so via classification. As classes of this field, we need to predict whether there is concealment or not, so only 2 classes. Saying that, we have obtained the results for audio and video concealment respectively as following:

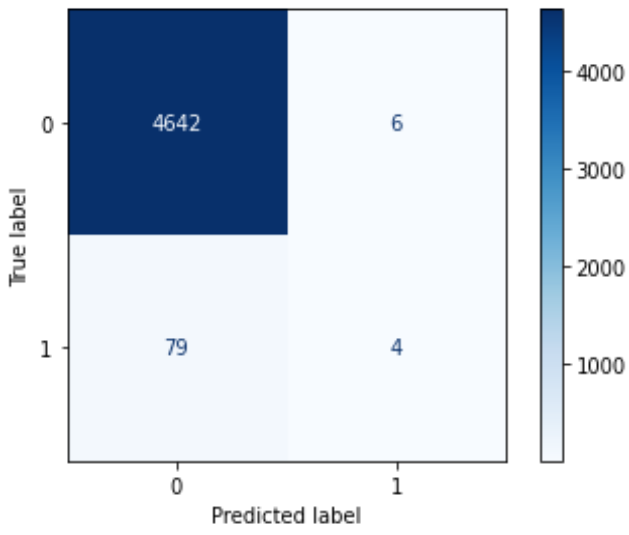
Table 6.3: *Results regarding the concealment prediction for cloud gaming*

Prediction type	Algorithm	Feature selection method	Result (f-1 score)
Audio	Decision Tree	Not used	0.54
Video	Random Forest	Annova f-value	0.54

- Video concealment, Decision Tree algorithm



Audio concealment, Random Forest with Annova f-value



Basically what we observe is quite expectable, because there is a huge gap between the number of supports for the two classes which makes the one with lower support having a bad predictability (f-1 score). The best way to improve it is the increase the number of support and balance it, solution that would lead to a better performance of the model. Although, what we got here is a f-1 score of

0.99 for the case when there is no concealment and 0.1 for the case when they are present. So, this indicates that the model's performance is better than a flipping coin scenario.

As a conclusion, we can say that the approach developed for meeting application is performing more or less similarly in another RTC scenario, except for some cases where we had also a lack of data. Therefore, probably defining some new QoE metrics would be one solution to this model because of the fact that we have only 2 resolution classes in total here, which is too less, as well as the number of frames dropped in this gaming scenario is too low.

Chapter 7

Conclusions

During the tough times of the Covid-19 pandemic, the world was somehow forced to move towards smart working and the amount of online meetings were by far increased compared to the period pre-pandemic. Almost after 2 years from the start of the pandemic, many companies are still applying the smart working (remote work) which clearly indicates that the world is now trying to limit as much as possible the working in presence when not neccesary. Saying that, the importance to have a high quality experince during the online meetings does not need any explanation. However, almost anyone of us, have had bad experiences temporarily during online meetings throughout this period and this is exactly what this study focusses on.

This work aims at predicting the quality of experience (QoE) metrics received by the end user through the quality of service (QoS) features which are measured in the network vantage points. So the main idea is to be able to use the QoS metrics which are measured from the network devices and through them predict what is the quality percived in the user side. Creating a model which is able to make such predictions is very important for ISP providers since by knowing the QoE in the user side they may change some QoS parameters which as a consequence would have a positive impact on the QoE as well. That's the reason why finding a correlation between the two is extremely important for maintaing a high quality experince in the online meetings.

The methodology used to create such a model in this study, is by exploiting a Machine Learning approach. On the other hand, the metrics used as target features to train the model are the following QoE metrics, extracted from the application logs: "*Resolution*", "*Smoothness*" and "*Concealment*". Moreover, the techniques used for predicting each of these targets were different, as well as the number of experimentations where I tried different ML algorithms with different performance improving methods. It's worth mentioning that the pcaps used for training and test sets during the experimentations are of different types which makes the problem realistic, since if they were the same the problem would have been trivial.

In conclusion, we managed to build a model via Machine Learning approach with promissing results. This model was also tested with another scenario different from the one of online meeting, which is a cloud gaming scenario and it came out that it performs fine there as well. However, for that last one we suggest to define some other target metrics since the ones defined for online meeting have different characteristics from those which may appear in another scenario such as cloud gaming. Despite that, the model created can be called successful and the application of the above mentioned suggestions would increase the usability of this model in the related fields.

7.1 Future Work

Despite having a couple of studies in this field, there are some challenges that can be considered for the future:

- Trying to make a further analyzation and **develop a mathematical relationship between QoE and QoS metrics**. The reason is that a Machine Learning approach always will be object to errors (never reach 100% accuracy), whereas finding a mathematical relationship would be much more effective for this purpose.

- Regarding concealments, in this study we focus on predicting only the presence of them, whereas as a future work it would be great to **predict the exact values of the concealments**, even though it is a really difficult problem.

- **Extend the application of this model** to other similar fields. Even though we tried for cloud gaming scenario in this study, we realized that some changes need to be made, that's why when extending the model, it must be very-well analyzed the type of scenario with its characteristics and use the correct target features when training the model.

BIBLIOGRAPHY

[1] Andrea Bianco - TNG Group - Politecnico Di Torino, https://www.telematica.polito.it/app/uploads/2018/12/rtp-rtcp-rstp_6.pdf

[2] IMS/SIP – RTP / RTCP, http://www.sharetechnote.com/html/IMS_SIP_RTP_RTCP.html

[3] GitHub - GianlucaPoliTo/Retina: RTP tool analyzer, https://github.com/GianlucaPoliTo/Retina

[4] CompTIA, What is Wireshark and how it is used ? , https://www.comptia.org/content/articles/what-is-wireshark-and-how-to-use-it

[5] Jupyter manual, https://jupyter.org/

[6] Aditya Mishra, Metrics to evaluate your Machine Learning Algorithm, https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234, Feb 2018

[7] Quality of Experience (QoE) and Quality of Service (QoS) in UAV Systems, University of Plymouth, Faculty of Science and Engineering. https://pearl.plymouth.ac.uk/bitstream/handle/10026.1/13197/QoE%20and%20QoE%20in%20UAV%20Systems%20final.pdf?sequence=1&isAllowed=y

[8] Techopedia, Real-Time Communications, https://www.techopedia.com/definition/24426/real-time-communications-rtc

[9] TechTarget, Real-Time Communications, https://searchunifiedcommunications.techtarget.com/definition/real-time-communications

[10] N. Rao, A. Maleki, F. Chen, W. Chen, C. Zhang, N. Kaur, A. Haque, "**Analysis of the Effect of QoS on Video Conferencing QoE**", Department of Computer Science, Western University London, Ontario N6A 5B7, Canada

[11] Ashkan Nikravesh, David Ke Hong, Qi Alfred Chen, Harsha V. Madhyastha, Z. Morley Mao, "**QoE Inference Without Application Control**", University of Michigan

[12] Tarun Mangla, Emir Halepovic, Mostafa Ammar, Ellen Zegura, "**eMIMIC: Estimating HTTP-based Video QoE Metrics from Encrypted Network Traffic**", Georgia Institute of Technology, AT&T Labs - Research.

[13] Ashkan Nikravesh,Qi Alfred Chen, Scott Haseley, Xiao Zhu, Geoffrey Challen, Z. Morley Mao,"**QoE Inference and Improvement Without End-Host Control**", University of Michigan, University of Illinois at Urbana-Champaign

[14] Paul Schmitt◦, Francesco Bronzino∗, Sara Ayoubi∗, Guilherme Martins◦, Renata Teixeira∗, and Nick Feamster◦, **"Inferring Streaming Video Quality from Encrypted Traffic: Practical Models and Deployment Experience**", ◦Princeton University ∗Inria, Paris

[15] Amir Ligata, Erma Perenda, Haris Gacanin, "**Quality of Experience Inference for Video Services in Home WiFi Networks**".

[16] Giovanna Carofiglio, Giulio Grassi, Enrico Loparco, Luca Muscariello, Michele Papalini, Jacques Samain, Cisco Systems, Inc, "**Characterizing the relationship between application QoE and network QoS for real-time services".**

[17] Pluralsight, Cleaning up data from the outliers, https://www.pluralsight.com/guides/cleaning-up-data-from-outliers

[18] Analytics Vidya, Introduction to Feature Selection methods with an example, https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variables/

[19] Dimitris Effrosynidis, Feature selection for Machine Learning: 3 categories and 12 methods, https://towardsdatascience.com/feature-selection-for-machine-learning-3-categories-and-12-methods-6a4403f86543

[20] Sayak Paul, Beginner guide to feature selection in Python, https://www.datacamp.com/community/tutorials/feature-selection-python , Jan 2020.

[21] Sunil Ray (Analytics Vidya), 8 proven ways for improving the "Accuracy" of a Machine Learning Model, https://www.analyticsvidhya.com/blog/2015/12/improve-machine-learning-results/ , Dec 2015.

[22] Sunil Ray (Analytics Vidya), A comprehensive guide to data exploration, https://www.analyticsvidhya.com/blog/2016/01/guide-data-exploration/#one, Jan 2016.

[23] IBM Vloud Education, Machine Learning, https://www.ibm.com/cloud/learn/machine-learning

[24] Ed Burns (TechTarget), Machine Learning,
https://searchenterpriseai.techtarget.com/definition/machine-learning-ML

[25] Sunil Ray (Analytics Vidya), Commonly used Machine Learning Algorithms,
https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/

[26] WebRTC, https://webrtc.org/

[27] Wikipedia, WebRTC, https://en.wikipedia.org/wiki/WebRTC

[28] Dataquest, Tutorial: Learning curves for Machine Learning algorithms,
https://www.dataquest.io/blog/learning-curves-machine-learning/ , Jan 2018

[29] Unai Lopez Ansoleaga, How to detect, handle and visualize outliers,
https://towardsdatascience.com/how-to-detect-handle-and-visualize-outliers-ad0b74af4af7 , Jun 2021

[30] Muhammad Jawad Khokhar, Thibaut Ehlinger, Chadi Barakat muhammad-jawad.khokhar@inria.fr , thibaut.ehlinger@inria.fr , chadi.barakat@inria.fr,
"**From Network Traffic Measurements to QoE for Internet Video**", Universite C´ ote d'Azur, Inria, France