ISSN 1688-2806

Universidad de la República
Facultad de Ingeniería

Audio-based Classroom Activity Detection for Primary School Lessons

Thesis submitted to Facultad de Ingeniería de la Universidad de la República by

Braulio Ríos

in partial fulfillment of the requirements for the degree of Master in Data Science and Machine Learning.

Thesis Directors

Pablo Cancela	Universidad de la República
Germán Capdehourat	Ceibal

Thesis Tribunal

Martín Rocamora	Universitat Pompeu Fabra
Guillermo Carbajal	Universidad de la República
Andrés Ferraro	McGill University

Academic Director

Pablo Cancela	Universidad de la República

Montevideo
Wednesday 27th September, 2023

Audio-based Classroom Activity Detection for Primary School Lessons, Braulio Ríos.

ISSN 1688-2806

This thesis was prepared in LaTeX using the iietesis class (v1.1) (adapted and translated).
Total pages: 108.
Compiled Wednesday 27th September, 2023.
`http://iie.fing.edu.uy/`

Acknowledgments / Agradecimientos

This is the only section that is not written in English, since it is intended to be read by people who speak Spanish.

Quiero comenzar por agradecer a mis tutores, Pablo Cancela y Germán Capdehourat, quienes no sólo me dieron incontables horas de su tiempo, toda su experiencia y apoyo durante todo el desarrollo de este trabajo, sino que también fueron quienes me motivaron a retomarlo luego de un intento frustrado. Además, lideran el proyecto del cual este trabajo forma parte apuntando a la mayor excelencia, pero también generando un excelente ambiente de intercambio, colaboración y trabajo en equipo entre todos los que participamos.

A mis otros compañeros en este proyecto, Emilio Martínez y Diego Silvera, que siempre participaron activamente aportando ideas y opiniones durante las reuniones de trabajo, enriqueciendo el proceso y el resultado. Todo esto además de tener un rol protagónico durante la etapa de etiquetado y curado del dataset que se usa en este trabajo, haciendo de este proceso -en general tedioso-, algo mucho más ameno y hasta memorable.

A Gonzalo Marín, por insistir en dar el paso para comenzar este viaje y motivarme siempre, además de realizar algunos de los contactos iniciales con mis tutores. No sólo fue un ejemplo a seguir en lo académico, sino también en muchos otros sentidos y fue un gran apoyo constante durante todo este camino.

A los miembros del tribunal: Martín Rocamora, Guillermo Carbajal y Andrés Ferraro, por sus comentarios y preguntas que evidenciaron un muy detallado y cuidadoso análisis del trabajo, resultando en correcciones que mejoraron el resultado final.

A la ANII (Agencia Nacional de Investigación e Innovación), por permitirme dedicarle mi tiempo en forma casi exclusiva a este trabajo durante un amplio período de tiempo, gracias a la financiación mediante una beca de maestría.

A Ceibal, por proveer los datos, facilitarnos las reuniones y a menudo las instalaciones para que el resultado de este trabajo tenga un sentido práctico además de académico.

A la Facultad de Ingeniería de la Universidad de la República, en la cual todo este trabajo se llevó a cabo, además de toda la formación académica necesaria. En particular, cabe un especial agradecimiento al Grupo de Procesamiento de Audio del Instituto de Ingeniería Eléctrica (IIE).

A mi familia, mis amigos de la vida, y a los tantos buenos que hice siendo compañeros de estudio o de trabajo. Son la razón de mi felicidad y por lo tanto quienes le dan sentido a todo esto.

Abstract

Classroom Activity Detection (CAD) is a challenging task, especially for primary school lessons, where student participation is fragmented, short, and often concurrent with teacher speech and background noise. This thesis proposes and evaluates three CAD models: two based on supervised audio classification (trained on a proprietary dataset that was annotated for this work), and one based on unsupervised diarization.

These models are assessed through the visualization of the estimated label density, rather than typical CAD segment visualizations. This approach proves to be more effective in dealing with the highly fragmented segments observed in this specific use case. The main metric to compare these models is the correlation coefficient between estimated and ground-truth label densities.

The density and correlation are used to evaluate the accuracy of the models in capturing the temporal distribution of the different classroom activities. Complimentary to that, another metric that is also used is the error in the total time estimated for each label (e.g., estimated Teacher Talking Time or TTT).

The supervised models, based on an LSTM neural network and a decision tree classifier, achieve similar classification performance, outperforming the unsupervised diarization pipeline. Even a small amount of training data is enough for the supervised models to achieve the performance of the diarization system, and they generalize well to previously unseen voices.

The unsupervised diarization model does not require training data for this particular task, but its performance is not as good as the supervised models to detect the teacher's voice. Additionally, it cannot distinguish properly between the labels "single student" and "group work".

Overall, the supervised CAD models proposed in this thesis demonstrate promising results for primary school lessons, even with limited training data. These models could be used to develop valuable tools to support classroom observation and evaluation.

Contents

Chapter 1

Introduction

1.1 Motivation

The design and development of classroom activities in primary school lessons play an important role in shaping students' learning experiences. With a growing focus on student-centered learning techniques, understanding these activities has become essential to improve learning outcomes, highlighting the need for thorough observation and evaluation procedures. However, traditional approaches often rely entirely on manual assessment by trained observers, which is time-consuming and impractical for large-scale evaluations. Therefore, the need for automated analysis tools to assist in this task has become increasingly important.

This work is carried out as part of a larger research project in partnership with *Ceibal*[1], a public institution aimed at advancing education through digital inclusion and innovation. The objective of the project is to create valuable tools to aid in the analysis of teaching practices, and this particular work focuses on the Classroom Activity Detection (CAD) module, as will be described below. The lessons to be analyzed are conducted by remote teachers using the Ceibal platform, which enables them to connect with primary school students in different regions of Uruguay. Ceibal's evaluation primarily centers on the remote teachers who use their platform.

Currently, the evaluation process consists of human evaluators meticulously watching and listening to recorded videos from virtual lessons multiple times to identify key moments, such as student participation, teacher speaking, and group work, following some guidelines and evaluation forms. This entirely manual approach is resource-intensive and requires significant effort and time from the evaluators. Consequently, only a fraction of all virtual lessons can be evaluated, severely limiting the scope of feedback and potentially overlooking crucial insights to enhance teaching methodologies.

In some courses, the entire evaluation of a full lesson may take one week per evaluator, using only manual assessment and considering some back and forth with the teachers, to provide them feedback. This limitation results in a small, random fraction of the total teaching time being observed. In fact, the current goal is to evaluate only one class per teacher per year, which represents a minimal fraction of the total teaching time throughout the year. Although different courses may have specific evaluation guidelines to follow, it has been recognized (through meetings with the people involved in the process) that assessing the level of student engagement, instances of group work, and comparing them with the Teaching Talking Time (time the teacher spends speaking) are generally crucial aspects.

[1]`https://ceibal.edu.uy/`

The CAD module developed in this work aims to automatically analyze audio recordings and identify key moments within the classroom, such as student participation, teacher instructions, and group interactions. This result should be visualized as a function of time, to understand the development of the lesson and provide an overview that facilitates the search and replay of specific sections. Additionally, the system should also provide global metrics such as *Teacher Talking Time* and the fraction of time spent in relation to student-centered activities.

It is important to emphasize that this work does not aim to automatically evaluate any aspect of the activities or directly replace any part of the evaluators' task. Instead, the aim is to provide them with support to make the process less repetitive and more efficient. Manual evaluation involves watching multiple replays of the recorded video and taking notes about different aspects to observe (the specific technique may vary based on the evaluator), resulting in an inevitably slow and hardly uniform process. Combining insights provided by the automatic analysis tool and the evaluators' criteria, the process should improve significantly in all these aspects.

The implementation of an automated analysis tool not only has the potential to reduce the time and effort required, but also allows for a more uniform evaluation criteria and provides objective feedback for educators to refine their teaching methodologies. Finally, due to the reduced time required to get an overview of a number of lessons, it enables the design of guidelines to select the lessons to observe, and also facilitates the identification of patterns and trends that may otherwise go unnoticed, thereby enabling data-driven decisions and promoting continuous pedagogical improvement.

1.1.1 Challenges and particularities

Although there is a vast literature on speech processing and classification, and even some particular on the topic of Classroom Activity Detection (CAD), to the best of our knowledge, there are no works that solve the problem presented here out-of-the-box. The main differences from some systems found so far are the following:

- Systems oriented to the analysis of college-level courses are not adequate, since the classroom dynamics are completely different from primary school lessons. In the latter case, student participation is very spontaneous and usually short, with conversations between teacher and student where some interventions may not exceed one second in duration. In fact, children rarely speak for more than a few seconds without being interrupted. In other words, the conversations are very fragmented and there are frequent overlaps between the teacher and students' voices.
- There is a need to perform well on children's voices. Although this distinction might seem an advantage to separate the voices of teachers from students easier (adults and children), there is also great variability within the voices of children, with some being even in lower pitch than the teacher's voice. Except for some audio detection models aimed specifically at children's voice classification (which does not match entirely the problem presented here), most models are pre-trained over datasets where there are only -or a vast majority of- adult voices. This leads to unpredictable behavior when faced with out-of-distribution samples such as children's voices, potentially in an overlapping and noisy environment.
- In the case of student activity observed in this data, clear individual participations are not frequent, and instead it is more common for many students to respond simultaneously to teacher questions. When providing simple answers, it may be the case that many children respond synchronously (in a chorus-like voice), but in general the responses consist of multiple voices overlapping, saying slightly different things, or hardly intelligible words.
- Group work instances bring up another set of challenges, since there might be long time intervals where there are no student or teacher interventions. Instead, a background whispering can be heard with varying intensity, eventually some voices might stand out and short

conversations with the teacher arise (queries and clarifications), but also long silences and all kind of sudden noises are observed, depending on the motivation and structure of the class.

For this work, the aim is to create a system that is capable of finding and distinguishing classroom activities (i.e., different kinds of student participation) and some global statistics, like total time estimates and percentages, in a way that is robust to the high variability observed among recordings, in terms of noise levels, audio quality, particular set-ups, and video-conference systems used.

Exact temporal precision is not a priority in this case (as might be the case in standard diarization systems, for example), which was also proven to be a hardly feasible task, given the observed amount of overlapping and high fragmentation between voices. From conversations with the evaluators, it was derived that providing time-based statistics with some localization over the audio (but not necessarily high temporal precision and resolution), combined with some global quantities (e.g: teacher talking time vs. students) would be a more effective approach to help them in their task.

1.2 Objectives

The main goal of this thesis work is to understand and evaluate audio processing and machine learning techniques to distinguish the teacher and student voices, enabling the detection of the key moments and activities in a recorded lesson, such as situations with high student participation or group work.

It is appealing to compare the performance of a diarization system (referred to as the *unsupervised* approach, since it does not require specific fitting to the problem's data) versus a *supervised* classification model, trained over a small dataset that will be manually annotated for this work. Both approaches shall be evaluated over the same test data.

Aside of providing a tool to assist the evaluators on their task, this enquiry is aimed to provide some answers to the following questions:

- What useful information can be extracted by using an out-of-the-box diarization system over the classroom audio?
- What is the human effort required and the best annotation criteria to create training and evaluation datasets for our task?
- What is a good evaluation criteria to compare diarization and audio classification models, in a way that relates to the value provided for the intended users of the system?
- Is the supervised approach able to generalize to new recorded lessons? How much human-annotated data is needed to make it comparable to the unsupervised approach?

1.3 Summary of Results

This section aims to provide an overview of the main results that were obtained throughout this work, according to the objectives presented above. All these results are discussed in more detail in the following sections.

As a general result, three different models of Classroom Activity Detection (CAD) were developed, specifically tailored for primary school lessons. All of these models were evaluated and compared, highlighting their advantages and weaknesses.

An end-to-end CAD system was developed, capable of processing any recorded lesson with any of the implemented models, generating an output video with the detection results. An example capture of the output can be observed in Figure 1.1. This video interface is intended as a tool to

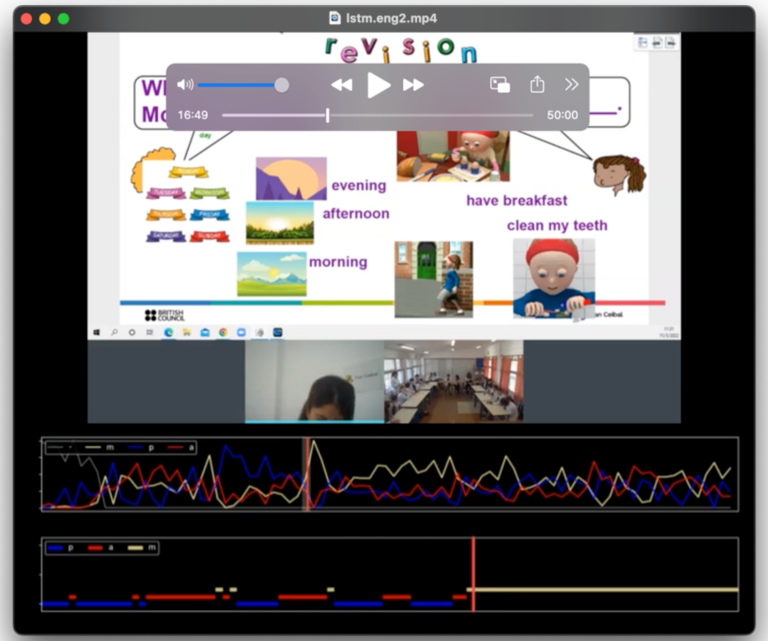


Figure 1.1: Capture of an example output video using our CAD system. It shows the original recorded lesson on top of the resulting plots. The user can easily navigate through the lesson, and the red cursors over the plots will mark the current position, relative to each track. The upper track covers the entire lesson and is considered the main result from the system. The curves represent the label densities as a function of time, showing the amount of teacher/student/groupwork activity (respectively in blue/red/yellow). The lower track plots the detected segments in a zoomed window of 30 seconds, showing the high level of fragmentation in the interventions.

assist evaluators, who can have an overview of the lesson and easily navigate through it. Other global metrics such as Teacher Talking Time are also calculated, and could be provided in the form of a PDF report.

To enable this research, a custom audio dataset was annotated, allowing us to train, evaluate, and compare the models. Although the dataset is not publicly available, we provide detailed descriptions of the annotation protocol, labeling criteria, and the challenges faced during the process. The data quality that results after following this protocol is assessed by comparing redundant human annotations in selected audio fragments.

The implemented CAD models consist of two supervised approaches (which require a portion of the annotated dataset for training) and one unsupervised model (which does not require training data). All these models were evaluated and compared on the data fraction that is not used for training.

To better understand the supervised models' performance and generalization capabilities, we designed a data partitioning approach to create balanced audio splits and groups, in which the models can be trained and evaluated separately. This enables us to understand how the model performance varies across lessons, how it improves with more training data, and how it generalizes to new lessons.

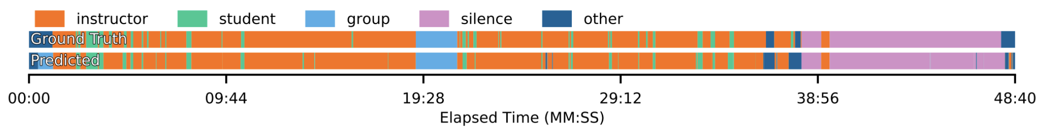


Figure 1.2: Example of a typical visualization of results from other CAD systems, tailored for college-level lessons. Source: [41]

A simple heuristic is proposed to implement the unsupervised model, converting diarization results into CAD outputs. We achieve accurate identification of the teachers' voice, but showing limitations in distinguishing student interventions and multiple background voices. We also compare the supervised model with increasing training data, to this unsupervised approach.

Finally, a new visualization approach was designed to show and compare CAD results specifically for the case of primary school lessons. Its usefulness can be appreciated by comparing Figure 1.1 (our system, designed for primary school lessons) with Figure 1.2 (another system, tailored for college-level lessons). The former shows detected segments only in a 30-second window (lower track with "zoomed" results), while the latter shows the detected segments in the whole 50-minute lesson. The interventions in our case are significantly more fragmented than those of college-level courses. In fact, showing the results in the usual CAD style (i.e., all detected segments for the usual 45-minute primary school lessons) turns out to be a very cluttered and confusing visualization for our use case, hardly useful in practical terms. Therefore, we introduce a method based on label density estimation (upper graph in Figure 1.1) and measure its performance by calculating the correlation coefficient with the reference density. This evaluation method is discussed in detail and compared to other usual metrics to justify its convenience.

1.4 Considered approaches

At the time of carrying out this work, no open datasets could be found aimed at detecting the kinds of classroom activities and scenarios that were required in this case. However, Ceibal provided several hours and instances of recorded lessons. Initially, some of these needed to be annotated in order to evaluate the performance of the implemented system, but it was also decided to spend some more annotation effort in order to create an initial training set for a supervised system.

Under these conditions, two CAD approaches were proposed: *Diarization* and *Classification*.

1.4.1 Why Diarization?

Audio diarization is the task of automatically detecting the speakers in a conversation and the exact intervals where they spoke. This must be achieved without knowing beforehand the number of speakers and without prior samples of their voices. The great advantage of these systems is that once trained, they can run on any audio with unknown speakers. That is, if we find a pre-trained diarization system, we can run and evaluate it on our lessons without needing any training data.

In order to achieve its task, a diarization system must extract audio features whose values depend primarily on the particular voice speakers, and are in turn invariant to the different words and intonations with which each person may speak. These features allow identifying audio segments that belong to the same speakers.

Once the system is trained, it must work with voices that were not present in the training set. Given a conversation, it makes the decision about which segments are similar enough to assign them to the same speaker. This is why understanding the functioning of a system with these characteristics seems very relevant to solving the problem of this work, where the main objective is to distinguish the voice of the teacher from those of the students.

Although the teacher's voice would not be known directly from the diarization output, in practice it has been observed that it is always the most frequent in primary school lessons. Therefore, if the diarization output is accurate enough, the speaker with the longest accumulated duration can be assigned to the teacher and the remaining ones should correspond to students in general.

However, there are limitations and potentially serious problems with this approach. The most important one is that depending on any heuristic -taking the predominant voice and assigning it to the teacher- usually leads to important errors when unforeseen conditions appear. For example, if the teacher is detected as two different speakers for some reason (e.g., due to a change in the video conference setup), only a fraction of its voice will be correctly identified, while the rest would be erroneously assigned to student participation. This would produce important errors in the estimations or even swapped labels.

Furthermore, estimations could be unpredictable during group work or when many students speak with a certain degree of overlap. This is, in fact, the most common scenario (as noted in Section 1.1.1), since it is uncommon for students to participate for long periods of time without interruptions. Although this is not the usual case in the datasets used to train diarization systems, it is very common in primary classroom audios.

1.4.2 Supervised Classification Models

A CAD system can also be implemented as a time-based classification model, where each time interval must be assigned a predefined label, such as teacher's voice, student's voice, groupwork, among other possibilities. This is the most common approach in the CAD literature.

The main disadvantage of this approach is the need to manually create a training set. The problem is that deep learning systems generally require a huge amount of labeled data in order to achieve good results, and there are limited resources for such a task in this work.

In any case, the idea of training these models is to have an estimate of the amount of data needed to have comparable results with the diarization system, and understand how the available data impacts the predictions (on lessons with known/unknown teachers).

This approach also aims to determine the appropriate audio features for this task. While there are numerous audio classification systems designed for diverse purposes, the uniqueness of this problem lies in distinguishing between adults' and children's voices in an environment with echoing, overlapping, and very fragmented voices, making it a distinctive domain. Therefore, it is essential to invest time in evaluating which audio features are relevant and suitable to address this specific challenge.

1.5 Related Work

The literature on Classroom Activity Detection (CAD) has seen significant advances in recent years, since it has become an area of interest in education research. The increasing interest in student-centered learning techniques promotes the analysis of classroom activities to maximize learning outcomes. Traditional approaches rely on manual annotation by trained observers, which is time consuming and impractical for large-scale evaluations. Therefore, there is a growing need for automated solutions in this field.

The detection of classroom activities has been performed in a wide range of scenarios, but most of the existing work is intended to automate the analysis of college-level courses. Wang et al. (2014) proposed training a random forest model using the outputs of modified wearable devices to predict activities such as "teacher lecturing", "whole class discussion" and "student group work" [16].

Owens et al. (2017) developed the DART system (Decibel Analysis for Research in Teaching), which used decision trees over signal amplitude statistics such as sound intensity and its variance, to predict the amount of time spent on single voice activities (e.g., lecture), multiple voice (e.g., peer-to-peer discussions), and no voice activities (other sounds) [26].

Cosbey et al. (2019) improved upon DART by employing a Gated Recurrent Unit (GRU) neural network on top of Mel-filterbank features [29]. They used the same three labels as DART but reported significant reductions in error rates compared to it, showcasing the potential of neural networks in CAD.

Another line of research from Li et al. (2020) focused on distinguishing between teacher and student speaker roles using siamese networks and an attentional prediction mechanism, on top of features that can be trained using representation learning [36].

More recently, Slyman et al. (2021) [41] compare many model architectures including recurrent and time-delay networks on top of different audio features such as Mel-filterbank and modern self-supervised embeddings, to perform CAD with different label granularity levels. At a fine-grained level, they define 9 labels, which include 4 instructor modes (lecture, asking/answering questions, or announcement) and 2 student modes (asking/answering question). Otherwise, they collapse these categories into 5 or 4 labels, which allows comparison with previously existing systems. A result from this particular work is shown in Figure 1.2.

It can be seen that the literature in CAD encompasses a range of approaches, including portable and wearable audio recording devices, model architectures based on deep neural networks including recurrent, siamese and attention-based mechanisms, and making use of many different audio features and embeddings aimed at speech processing.

Although previous work demonstrated many advances in this field, challenges persist in accurately recognizing speaker roles in classrooms. Particularly for primary school lessons, the challenge of fast turn-taking (i.e., fragmented interventions) and high overlap is such an issue that the standard way of showing CAD results as a timeline bar with different colors (as in Figure 1.2) is not suitable to compare predictions against reference ground truth labels. Some segments are so short that they may not even be visible on a time scale that covers the entire lesson.

Furthermore, our CAD model needs to work under particular conditions, including high levels of noise, low audio quality, and the diverse set-ups and video-conference systems used. In addition, primary school lessons are outside the regular domain of systems focused on college-level lessons, where students are basically adults. The developed solution must be robust to all these varying conditions and the results must be evaluated and compared in a way that is related to the value of the information that they provide to the users.

Chapter 2

Methodological Foundations

In this chapter, we discuss some of the theoretical underpinnings that are needed to understand the different CAD models implemented in this work.

We begin with the essential elements of audio and speech processing, from the vocal tract model to a recap of some important concepts and tools of time-frequency analysis, such as the Short-Time Fourier Transform (STFT) and Mel-Frequency Cepstral Coefficients (MFCC). These techniques are examined in detail, as they are the basis for capturing the distinguishing features of speech and audio, in general. Other audio features based on statistics of the time-frequency components of audio signals are also described, among others, spectral mean, bandwidth, and contrast.

Next, we delve into the machine learning classification models employed in our supervised approaches. We start with Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM), a particular type of architecture that is well-known for sequence classification tasks. Then we also discuss Classification and Regression Trees (CART) models and in particular the widely used XGBoost classifier. It serves as an alternative for comparison, being a robust and versatile tool in machine learning that is not specifically tailored for sequence classification, but can be adapted for it.

Furthermore, we provide an extensive overview of a typical diarization pipeline, which is the competing *unsupervised* approach that we want to compare to the supervised models mentioned above. We briefly explain the main concepts and metrics used. Then, a diagram of the pipeline is outlined and the main blocks are described, including speaker embedding extraction and clustering, describing some example algorithms that are typically used for each task.

Building upon these fundamental techniques, we lay the groundwork for the subsequent chapters, where we dive into the implementation, experimentation, and evaluation of our approach.

2.1 Audio Features for Speech Processing

The human voice is generated by the vocal tract, which includes the larynx, pharynx, oral cavity, and nasal cavity. To produce voiced sounds, air is pushed out of our lungs and passes through the vocal folds in the larynx (see Figure 2.1), causing them to vibrate and produce an impulse train that is the basis of the voiced sound synthesis process. This train can be seen as the output T_p of the excitation generator in Figure 2.2.

This sound then travels through the vocal tract, where it is shaped by the different structures,

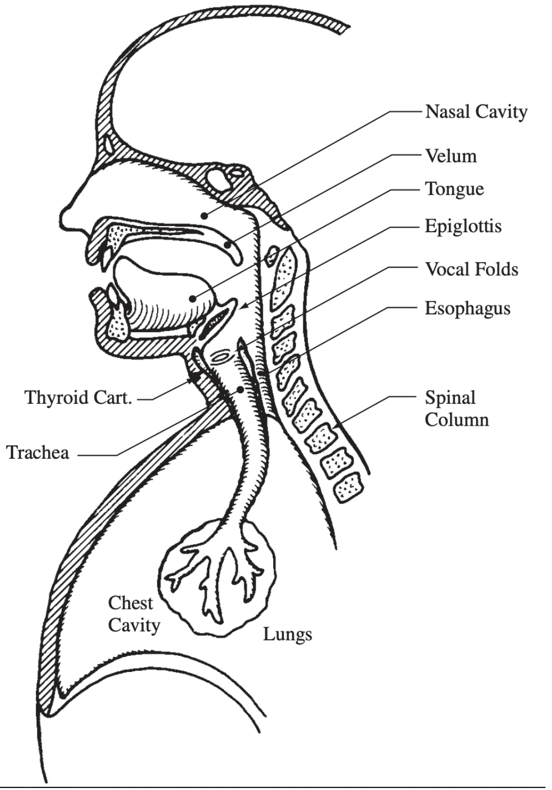


Figure 2.1: Cut of the vocal tract (sagittal), trachea and lungs (Source: [13])

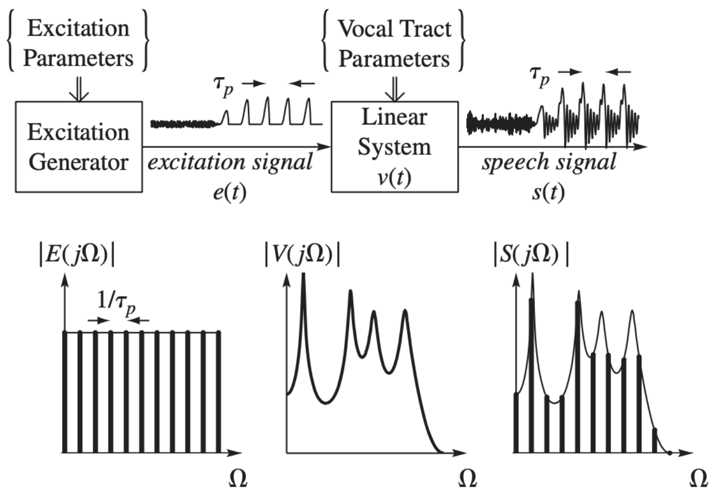


Figure 2.2: Diagram of the signals involved in speech synthesis, as they are output from the different components. These signals can be visualized in the time (up) and frequency (down) domains (Source: [13])

creating specific resonances, and producing unique speech sounds. The vocal tract can basically be modeled as a linear system that shapes the amplitude of the different frequency components, as shown at the bottom of 2.2, where the frequency response of each part of the synthesis is placed below the corresponding block. Mathematically, the frequency representation of an impulse train is another impulse train but in the frequency domain (as shown at the left of the figure), which is then filtered by the represented linear system, and results in a series of frequency peaks (or harmonics) with different magnitude that is obtained by multiplying both functions in the frequency domain (shown at the bottom right of the figure).

The frequency of the impulse train, which is the rate at which the vocal folds vibrate and determine the pitch of the voice, is called the fundamental frequency (F0). This is the lowest frequency and also determines the harmonics positions, since they are multiples of F0.

F0 varies according to the speaker, gender and emotional state and can be used to distinguish between different speakers.

However, F0 alone is not enough to identify speakers as different voices can have similar pitch values. Therefore, speech analysis also involves formant analysis, which refers to the resonances of the vocal tract that shape the sound of the voice. These formants cannot be observed directly (that is, the exact shape of $V(j\Omega)$ in Figure 2.2), but they can be estimated from the final result in $S(j\Omega)$, with a resolution that depends on the fundamental frequency value, since the spacing between peaks is determined by it.

Formants are determined by the size and shape of the vocal tract and are characterized by their center frequency, bandwidth, and amplitude. By analyzing the formants of different speakers, unique patterns that are specific to each individual can be identified. This information can be used to detect and differentiate between different speakers' voices, even in complex acoustic environments.

It is important to note that all these parameters are not fixed, but they change in time as the speech signal is produced. However, if the observed time interval is small enough, they can be considered to be approximately constant in these segments. Time-frequency analysis is particularly useful in this context, as it enables us to analyze speech signals in both the time and frequency domains. By applying techniques such as the Short-Time Fourier Transform (STFT), features can be extracted from speech signals that reveal their spectral and temporal characteristics, so that formants and F0 can be approximated and visualized as a function of time.

Finally, it is worth mentioning that in speech there are also unvoiced sounds (e.g., consonants "f", "s"), which do not involve activity in the vocal folds at all. These sounds have a frequency spectrum that is basically white noise reshaped by the vocal cavity filters. These differences are described in more detail in the next section, showing examples of voiced and unvoiced sounds in Figure 2.3.

2.1.1 STFT for speech signals

Vowels and some consonants like *m* and *n*, which involve the vibration of the vocal folds, will exhibit a harmonic spectrum, which means that there is a fundamental frequency and multiples of it, with different frequency envelopes. These are distinguished as *voiced* sounds, as opposed to the mentioned *unvoiced* sounds.

This can be better visualized by observing the spectrogram of some examples. In Figure 2.3 there are two consecutive vowels (a and e), and one consonant at the end (f).

The horizontal lines observed are precisely the frequency peaks due to harmonics (the bottom-most line being F0), so in the first two vowels, it can be seen that F0 is similar, but the intensity of the harmonics varies because of the different formant frequencies. At the end, the non-voiced sound of the consonant is visualized as a non-harmonic spectrum (no lines corresponding to F0 multiples), which instead shows a flat response over a wide frequency range, similar to filtered white noise.

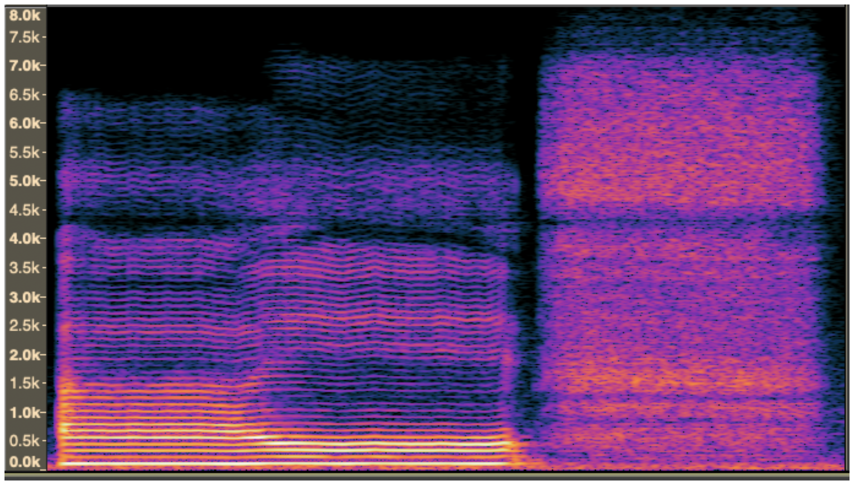


Figure 2.3: Spectrogram of two voiced sounds (vowels *a*, *e*) with harmonic spectrum, and an unvoiced consonant at the end (f). Horizontal axis represents time, vertical axis shows frequency, and lighter colors mean higher signal intensity at the given time-frequency point

Another important point to observe from this figure is the fact that most of the signal power for voiced sounds is in the harmonics below 4 kHz, which means that it should be possible to distinguish voices by looking at this restricted bandwidth.

Windowing and the *Uncertainty Principle*

One of the most fundamental parameters to choose in any time-frequency transformation (like the STFT), is the size of the sliding window that is evaluated at each time interval, on top of which the Fast Fourier Transform (FFT) is calculated.

The range of good values for this parameter depends on the kind of signals under analysis and the intended application. Conceptually, note that if the windows are too short, they will not cover enough cycles of the signal (mainly in the lower frequency components) to calculate its frequency components precisely (i.e: low frequency resolution), which is visualized as a spectrogram with a lot of uncertainty due to wide frequency bands.

On the other hand, if the window is made much larger so as to include many cycles, two problems arise. First, the temporal resolution is lower, because now the location of the sliding window belongs to a wider time interval (e.g: if there is a sudden frequency change, the spectrogram will react slowly, so the exact time of the event is less clear). And second, the voice signal may change its frequency components widely in the time duration of a large window, leading to a mixture where the frequency components are not well-defined because they represent a signal where the frequencies are not nearly constant.

This *uncertainty principle* (which is an adaptation of the same Quantum Mechanics principle for signals) actually has a precise mathematical definition and an equation to calculate the relation between time and frequency uncertainties. In fact, as both quantities cannot be minimized at the same time, considering the particular example of a pulse with uncorrelated time frequency (e.g., symmetric in time), pulse width T and bandwidth B (defined as the time and frequency standard deviations, respectively), the following inequality holds (see [6]):

$$T.B \geq \frac{1}{2}. \tag{2.1}$$

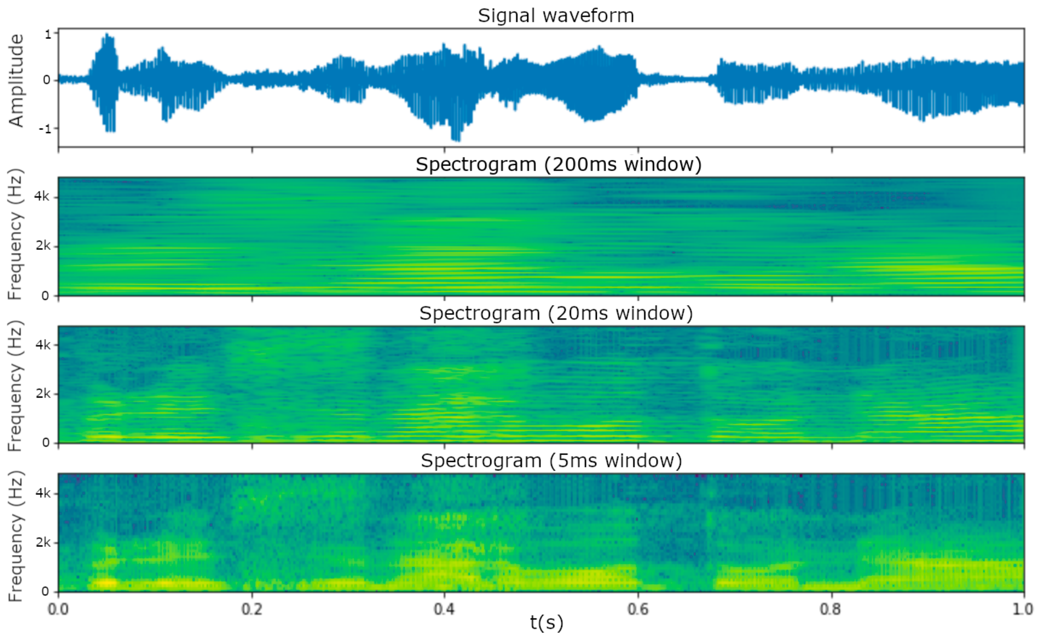


Figure 2.4: Spectrograms of the same audio signal, using different sliding window lengths (see plot titles).

This means that, regardless of its shape, it is not possible to create a pulse without some uncertainty in its time position or in frequency. Some edge cases to illustrate this: the Dirac delta ($\delta(t)$), has a precise instantaneous position (pulse width/variance $T = 0$), but infinite bandwidth ($B = \infty$); and an infinite sine wave has a precise frequency (bandwidth $B = 0$), but it is impossible to know its exact location (pulse width $T = \infty$).

That said, the shape of the pulse does make a difference in the actual uncertainty that the signal has, since this inequality only imposes a lower bound on the product. There is a family of complex Gaussian pulses that are optimal in time-frequency localization, in the sense that they reach the equality value in this equation. This is part of the theoretical framework behind the so-called Gabor filters, which have recently been used by Google to develop an audio front-end for feature extraction called LEAF, similar to STFT or Mel-filterbanks but with learnable parameters that are fitted to the training data [42]. This is an alternative that could be evaluated in the future and include in audio classification models.

The uncertainty principle is also useful in practice when the STFT of a signal is extracted, since the FFT is calculated at each frame after the windowing process, which basically converts the signal into a short pulse with the duration of the window, at each step. This windowing process has the effect of increasing the bandwidth of the frequency peaks (since the time location is decreased, the frequency uncertainty increases). That is why speech signals are a challenge: a good frequency resolution is needed from frequencies around 100 Hz (period of 10 ms, so the window must be chosen above that), but at the same time the signal properties change rapidly as people can pronounce numerous phonemes in a fraction of a second.

The effect of this parameter on a speech signal can be clearly visualized in Figure 2.4, where in the top example there is a long window size of 200 ms, hence the frequency bands in the spectrogram are quite narrow (good frequency resolution), but the time resolution is very bad compared to the cases below, and the changes in the waveform are not clearly delimited in the spectrogram.

In the opposite extreme, there is a short window size of 5 ms at the bottom, where it is observed

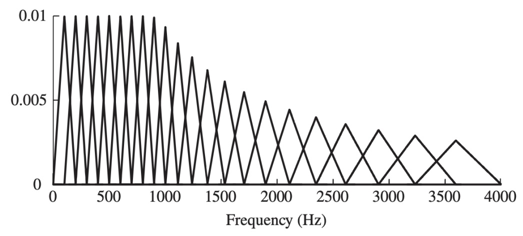


Figure 2.5: Triangular filters used to calculate the Mel-filterbank features. The center and width of each filter are evenly sized in the Mel scale (Source: [13])

that the frequency bands of the harmonics are no longer distinguished, but all the time transients of the signal are clearly delimited.

The image in the center, with a window size of 20 ms shows a good balance between the frequency and time resolutions. For speech processing, the window sizes are generally between 20 ms and 30 ms.

2.1.2 Mel-filterbanks

These features are widely used, and they produce a spectrogram that is basically a transformed version of the STFT-based spectrogram.

The goal of Mel-filterbanks (named after the pitch measurement unit *mel*) is to mimic the human ear's sensitivity to different frequencies. The human ear is more sensitive to frequencies in the range of speech and music, and less sensitive to frequencies outside that range. Also, at higher frequencies, larger changes are needed for the ear to perceive them, so the filters in this transformation are also aimed at grouping together the frequencies that would not be perceived as different by a human, producing a spectrogram that reflects better the human perception.

By using Mel-filterbanks, the important frequency components of an audio signal can be captured, while reducing the noise introduced by frequencies outside that range.

Mel-filterbanks work by dividing the frequency spectrum of an audio signal into a set of overlapping triangular filters, as shown in Figure 2.5. Each filter has a center frequency that corresponds to a specific Mel frequency, and the width of the filter is determined by the critical bandwidth at that point (the minimum change that the human ear would perceive).

The filters are evenly spaced on a Mel frequency scale, which is a non-linear scale that maps the frequency spectrum of sound to the Mel scale, which is usually approximated by the Equation 2.2 (see [13]):

$$m = 1127.01048 \ln\{1 + f/700\}. \tag{2.2}$$

The output of each filter is the energy of the signal within that filter's frequency range. This energy is typically computed using a discrete cosine transform (DCT). The result is a set of filterbank outputs that represent the energy distribution of the input signal across the Mel frequency scale. The number of filters to use (and hence the frequency range to cover) can be changed, and for speech processing applications, the minimum number that should be used is 24 filters, which covers the most relevant range of human voice, that is $0 - 4$ kHz.

2.1.3 MFCC: Mel-filterbank Cepstral Coefficients

The notion of cepstrum was introduced in a landmark paper published in 1963 titled "The Quefrency Analysis of Time Series for Echoes: Cepstrum, Pseudo-Autocovariance, Cross-Cepstrum,

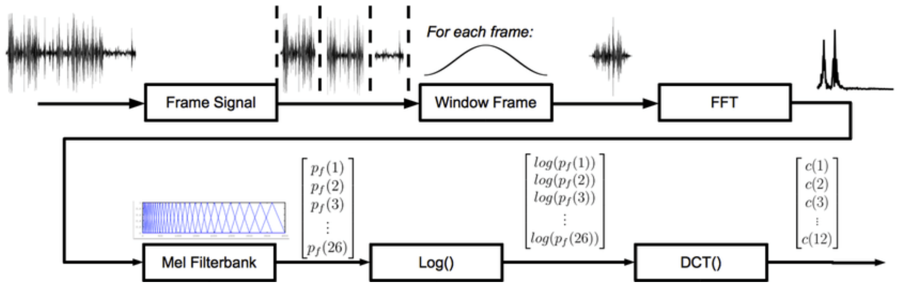

Frame Signal
For each frame:
Window Frame
FFT
Mel Filterbank
Log()
DCT()
$p_f(1)$ $p_f(2)$ $p_f(3)$ ⋮ $p_f(26)$
$log(p_f(1))$ $log(p_f(2))$ $log(p_f(3))$ ⋮ $log(p_f(26))$
$c(1)$ $c(2)$ $c(3)$ ⋮ $c(12)$

Figure 2.6: Complete diagram of the MFCC calculation from the signal waveform, including the FFT transform over the windowed frames (which is the STFT), the Mel-filterbank transformation, the log operation and finally the discrete cosine transform. (Source: [19])

and Saphe Cracking." The term "cepstrum" is derived from "spectrum" spelled quite randomly, just as "quefrency" derives from "frequency" and "lifter" from "filter".

The reason for introducing these funny names is that this technique involves taking the inverse Fourier transform of the logarithm of the power spectrum of a signal, and it contains information about the periodicities and repetitions in the log spectrogram. Since there are two Fourier transforms involved (one is inverse but over the logarithm of the magnitude), the new signal cannot be interpreted in units of time or frequency; hence the name *quefrency* was introduced.

In 1980, these techniques were adapted particularly for speech processing in [2], by performing *cepstral* analysis on top of the Mel log-spectrogram, giving rise to the so-called *MFCC: Mel-filterbank Cepstral Coefficients*. The exact formula to calculate these coefficients can be found in Eq. 2.3, where instead of the inverse Fourier, the *DCT: Discrete Cosine Transform* is used.

$$c[n] = \frac{1}{R}\sum_{r=1}^{R} \log(p_f[r]).\cos\left(\frac{2\pi}{R}\,(r+0.5)\,.n\right). \tag{2.3}$$

Where $p_f[r]$ are the Mel bands (using a total of R bands), as shown in Figure 2.6, where there is a complete diagram of this calculation from the signal waveform.

Basically, the log operation over the Mel spectrum will enhance small peaks, which usually correspond to higher-order harmonics, making it easier to observe the periodicity of the signal spectrum. Then, by performing the cosine transform, the signal is transformed into the quefrency domain, where the lower quefrency coefficients will capture the slowly varying spectral envelope of the signal, which is information about the frequency formants. In these coefficients, the many periodic peaks of the harmonics (multiples of F0) are filtered out, the short distance between them is not relevant, and only their modulation envelope is captured.

The information about F0 is instead encoded in the higher-order coefficients or high quefrencies, since its responsible for the rapid periodicities observed in the harmonic spectrum.

2.1.4 Other audio features

There are other audio features that were used or evaluated in this work, some of them are just statistics on top of the spectrogram, and further information can be found on the documentation of the *librosa* library [48]. In this section, a brief description and an explanation of how they could be relevant for speech processing is provided.

Spectral Flatness

Spectral flatness is a measure of the level of distribution of energy across the frequency spectrum of an audio signal. This is also referred as the tonality coefficient, and quantifies how much noise-like a sound is, as opposed to being tone-like. In speech recognition, this means that it can be useful to distinguish voiced and non-voiced sounds, as was explained in the previous sections.

A high spectral flatness (closer to 1) indicates that the spectrum is similar to white noise, which can also be interpreted as a signal with high randomness [11]. In these cases, the energy is distributed uniformly across the frequency range, as was observed for the consonant sound in Figure 2.3.

Spectral Mean

Spectral mean or centroid is a measure of the central value of the frequency spectrum. It is computed by first normalizing the magnitude spectrogram as a density distribution and then taking its average value.

This means that a speaker with a high-pitched voice may have a higher spectral mean than a speaker with a lower-pitched voice. This is not equal to the actual pitch, which is measured from the fundamental frequency, but instead it can be more robust to overtone or undertone errors due to detecting higher-order harmonics instead of the actual F0.

Because of this property, the spectral mean can also be used to distinguish between male and female speakers, as male speakers typically have a lower spectral mean than female speakers, and this is also the case with adult and child voices, which is very useful for Classroom Activity Detection.

Spectral Bandwidth

Spectral bandwidth is a measure of the spread of the frequency spectrum of an audio signal. It is computed by taking the standard deviation of the power spectrum, where the mean is the spectral mean. The spectral bandwidth captures the range of frequencies present in the signal.

It is useful for speaker recognition because it can capture the timbre of the speaker's voice. For example, a speaker with a nasal voice may have a narrower spectral bandwidth than a speaker with a more resonant voice.

Spectral Contrast

The spectral contrast is not a scalar value, but a vector. Each frame of the spectrogram is divided into a few sub-bands. These should not be too narrow bands, as they need to include peaks and valleys. For example, the default value of *librosa* is 7 bands. For each subband, the contrast is estimated by comparing the mean energy in the top quantile (peak energy) to that of the bottom quantile (valley energy).

High contrast values generally correspond to clear narrowband signals, whereas low contrast values correspond to noise in that frequency band.

Different speakers may have different spectral contrast patterns across the frequency bands, and so this feature vector can be useful for our classification systems.

Signal Power

This is a measure of the energy of the audio signal, in each windowed frame. It can be computed by adding the squares of the signal samples and dividing by the number of samples, which provides an estimate of the loudness of the signal at each time interval.

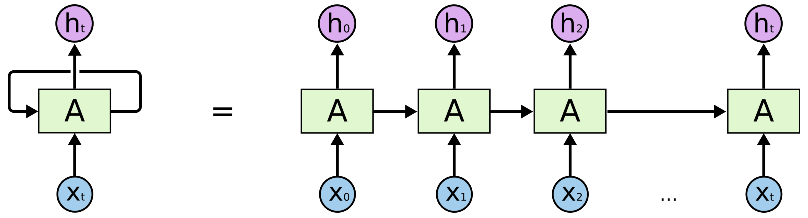


Figure 2.7: Basic architecture of a Recurrent Neural Network (RNN), showing that it contains a recurrent loop (left) that can be also visualized as an unrolled sequence with multiple copies of the same network, with the same parameters but different states and inputs at each step (right). Source: [18]

It can also be calculated for each frame directly from the spectrogram (using the Parseval theorem, which states that the energy in the frequency domain must be equal to the energy in the time domain), and this is the preferred method since the signal is already windowed.

In this work, the power is estimated by calculating the RMS (Root Mean Square) value first (which is available in *librosa*), and then converted to Decibel scale, using the audition threshold ($20\mu P$ - RMS sound pressure in pascals) as reference:

$$P(t)_{dB} = 20\log(RMS(t)/20 \times 10^{-6}). \tag{2.4}$$

It's important to normalize the average audio loudness of each recording during training and inference, since the important feature is the relative loudness of the speaker's voice at any given time, and it should not depend on the volume at which the recording was configured.

Zero Crossing Rate (ZCR)

ZCR is a straightforward measurement on the time domain: it counts how often the audio signal crosses the zero level. The number of times the audio signal changes sign is counted and divided by the duration of the frame interval.

Although it may lead to unintuitive results for signals with noise or with a complex mix of frequencies, it is often considered for speech recognition applications due to its simplicity [13].

2.2 Recurrent Neural Networks and the LSTM

Recurrent Neural Networks (RNNs) are a specialized architecture designed to handle sequential data, where the order of input elements is significant. Their main advantage is their ability to retain a hidden state that preserves information about past inputs in the sequence. This serves as a form of memory, enabling RNNs to capture dependencies and patterns over time. This is achieved by using loops in their architecture (as in the left side of Figure 2.7), allowing the network to process each element of the input sequence while updating the hidden state with information from the previous steps.

The loops can also be visualized in its "unrolled" form, as a series of interconnected copies of the network, one for each time step (as in the right side of Figure 2.7). The unrolled visualization helps to understand the flow of gradients through time and how they affect the updates of the network's parameters. It becomes evident that the gradients at each time step depend not only on the current step but also on the gradients from subsequent ones, which represent the influence of future time steps on the current one. Considering this interdependence of gradients across time, it is possible to understand that long-term dependencies can lead to issues known as vanishing or exploding gradients in conventional RNNs, and the need for architectures that allow better control on the flow of information through the network, like the LSTM.

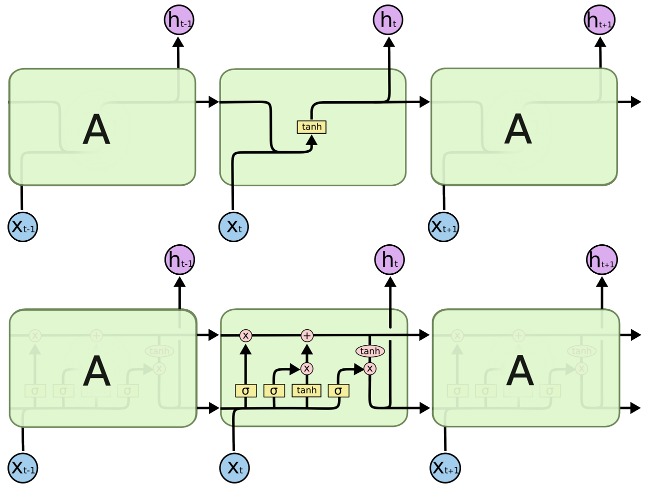


Figure 2.8: Unrolled diagrams of recurrent networks. The input is a sequence $x_{1..T}$ and the output is usually extracted only at the last step as h_T, the final state after the whole sequence has been processed. **Top:** Standard RNN network, consisting of a single block where each state is calculated as $h_t = tanh(W_x x_t + W_h h_{t-1} + b)$. **Bottom:** LSTM network, with all the gate mechanisms to control the information flow to the memory cells, and avoid the exploding/vanishing gradients problem. Source: [18]

The basic architecture of the LSTM was proposed in 1997 by Schmidthuber & Hochreiter [8], and was later improved by the same authors in 2000 [9] with the introduction of the forget gate. In subsequent years, researchers have proposed various modifications and extensions to the LSTM architecture. A comprehensive examination of various architectures was conducted in 2015 by researchers at Google, accompanied by an empirical evaluation of the components to assess their impact on performance [17]. It is also worth mentioning that a great conceptual explanation including a walkthrough on how this architecture works has been published by Cristopher Olah on his personal blog [18].

As mentioned, one of the main problems of conventional RNN architectures (top image in Figure 2.8) is known as the exploding or vanishing gradients, which can arise in deep neural networks when backpropagating errors through many layers. In RNNs, this is especially problematic because of the Backpropagation Through Time (BPTT) process: the gradient flows through the network for multiple sequence steps because of its recurrent nature. As a consequence, if the sequence is too long, the training process becomes unstable.

In the LSTM architecture (bottom image in Figure 2.8), this problem of divergent gradients is mitigated by using a gating mechanism that controls the information flow through the network, allowing the gradients to be backpropagated through many time steps without being significantly amplified or attenuated, which makes the training process stabilize even for long sequence lengths.

2.2.1 How LSTMs work

For this work, the LSTM in use is defined by the following set of equations [15], which have a correspondence with the elements in the diagram at the bottom of Figure 2.8 from left to right:

$$\begin{aligned}
f_t &= \sigma(W_{if}x_t + W_{hf}h_{t-1} + b_f) \\
i_t &= \sigma(W_{ii}x_t + W_{hi}h_{t-1} + b_i) \\
g_t &= \tanh(W_{ig}x_t + W_{hg}h_{t-1} + b_g) \\
o_t &= \sigma(W_{io}x_t + W_{ho}h_{t-1} + b_o) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t).
\end{aligned} \tag{2.5}$$

Here, all the W_{nm}, b_m parameters are learnable matrices or biases, respectively. The symbol $\odot$ means the element-wise product (i.e., Hadamard product).

Let us first identify all the $\sigma(\cdot)$ blocks in Figure 2.8 and also in the equations (left to right in the figure is mapped as top to bottom in the equations). They are called the *forget gate* f_t, *input gate* i_t and *output gate* o_t. The symbol represents the standard sigmoid function, which maps any input value to the range $[0, 1]$, hence the name *gate*: by multiplying its output by a vector, it controls how much information flows through the gate (that is why all gates are followed by multiplication blocks).

Next, observe that the lower arrow lane in the LSTM figure, takes its input from the output h_{t-1} of the previous step. In each cell, h_t is calculated from the upper lane after passing through a $\tanh(\cdot)$ block. The upper lane represents c_t in the equations, which is the cell state or memory. The memory value passes through $\tanh(\cdot)$ to ensure that it is in range $[-1, 1]$ and then through the output gate o_t, to produce the output h_t.

At this point, the role of g_t can also be clarified: this is the new candidate value for updating the cell memory. In the figure, it is the output of the yellow tanh block, which applies linear transformations and adds biases to the sequence input x_t and the previous output h_{t-1}. The new candidate to update the memory value must pass through the *input gate* i_t, and then be added to the upper lane c_t, which represents the memory as mentioned before, to update it.

The memory state from the previous step c_{t-1} can be preserved if $f_t \simeq 1$ (first yellow block σ) or totally reset if $f_t = 0$, hence the name *forget gate*.

This LSTM definition has been used successfully for a wide range of tasks including speech recognition and audio classification, but also for any kind of problem involving inference over sequences of feature vectors (like time series forecasting, natural language processing, video processing, etc.). There are also many variants proposed for the LSTM and conventional RNN, such as the more recent *GRU: Gated Recurrent Unit* which is simpler and achieves similar results to LSTM in some tasks, but was not implemented in this work.

2.2.2 Using LSTMs for audio classification

LSTMs can be used for a variety of audio classification tasks, including speaker recognition, music genre classification, and detection of sound events [15], [41].

To use an LSTM for audio classification, we first convert the audio signal into a sequence of feature vectors as seen in Section 2.1.

These feature vectors are then fed into the LSTM one at a time, along with the previous hidden state. After processing the entire sequence, the final hidden state is fed to a *fully connected* layer (i.e. a learnable linear transformation) that maps the hidden state dimension to the number of *classes* or *labels* in the classification problem. Finally, these numbers are translated into a probability distribution over the possible output classes by using a softmax layer.

It is usual to stack LSTMs in multiple *layers*, by using the output sequence h_t from one network as the input of another network.

The number of layers and the hidden dimension (i.e., the dimensions of c_t and h_t) are important parameters that affect the performance of the LSTM. Increasing the number of layers and the hidden dimension can improve the LSTM's ability to capture complex features and relationships in the audio signal but may also increase the risk of overfitting.

2.3 XGBoost: Gradient Boosted Trees

Another family of supervised classification models that is evaluated in this work is based on *Decision Trees* instead of *Recurrent Neural Networks* like the LSTM. These models are fundamentally different in their working principles, and are based on CART (Classification and Regression Trees) algorithms introduced in 1984 by Leo Breiman et al. [3].

In particular, XGBoost (eXtreme Gradient Boosting) [20] is an advanced and powerful machine learning algorithm that combines CART with the principles of gradient boosting introduced by Jerome Friedman in 2001 [10]. It is known for its efficiency, accuracy, and scalability and has been widely used in various machine learning competitions and real-world applications.

There are two important reasons for studying this particular model. On one side, we check the LSTM's performance against another state-of-the-art classifier that works with a different principle, and on the other side we have an alternative that does not require any special hardware to run. It can be trained on CPU and does not depend on a GPU like the LSTM does -the latter is orders of magnitude slower without it-.

2.3.1 Gradient Boosting and CART

To understand how XGBoost works, let us break down its key components and the steps involved:

- CART is a decision tree algorithm that can be used for both classification and regression tasks. It recursively partitions the input data on the basis of certain feature thresholds, creating a binary tree structure. Each internal node represents a decision based on a characteristic and each leaf node holds a prediction value.
- Gradient boosting is an ensemble learning method that combines multiple weak prediction models (typically decision trees) to create a strong predictive model. It builds the model in an iterative manner, sequentially adding new weak predictors to correct the errors made by the previous ones.

XGBoost employs gradient boosting as its foundational framework, using decision trees for week predictors.

2.3.2 How XGBoost works

XGBoost starts by defining an objective function that needs to be optimized during the training process. This means choosing the model parameters θ that minimize the objective function $\mathcal{L}(\theta)$. The function consists of two parts: a loss function $L(\theta)$ that increases with a greater difference between predicted and actual values, and a regularization term $\Omega(\theta)$ that controls the complexity of the model to prevent overfitting [20]:

$$\mathcal{L}(\theta) = L(\theta) + \Omega(\theta). \tag{2.6}$$

During the training process, XGBoost builds the ensemble of weak predictors in an iterative manner. It starts with an initial prediction (usually the average of the target values) and calculates the

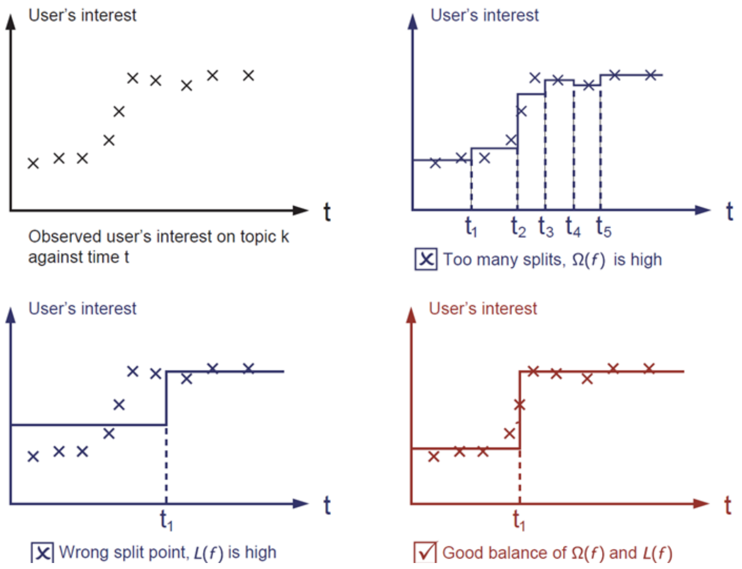


Figure 2.9: Examples to illustrate the regularization concept of an XGBoost model. The data points to fit are user's interest on a topic as a function of time (**top left**). If the complexity of the model is too high (i.e. overfitting the data), the regularization term $\Omega(f)$ is high, hence the objective function would not be at a minimum (**top right**). If the tree split parameters are not optimal, then $L(f)$ is high, and again the objective function would not be optimal (**bottom left**). When the regularization parameters are properly selected and the objective function is minimized, the resulting predictions should be as observed in the **bottom right** case. Source: [20]

gradient of the loss function with respect to the current prediction. Then, it fits a new decision tree aiming to improve the prediction at that step, which can be calculated as the sum of all the weak learners so far:

Assume that we are training on a dataset of samples with features and target values x_i, y_i. At the iteration step t, it creates a new weak learner tree f_t, to improve the predictions from the previous training step $\hat{y}_i^{t-1}$.

$$\hat{y}_i^t = \sum_{k=1}^{t} f_i(x_i) = \hat{y}_i^{t-1} + f_t(x_i)$$

Only the parameters of f_t are adjusted in each training step t, based on the gradient of the loss function. The loss $L^t(\theta)$ from the above equation is the sum of the sample losses $l(.)$ for this step (which could be the squared error or logistic loss, for example). So, given a set with n samples, we add up:

$$\mathcal{L}^t = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{t-1} + f_t(x_i)) + \Omega(f_t).$$

Here, the regularization term for the new tree is $\Omega(f_t) = \gamma T + \frac{1}{2}\lambda||\omega||^2$. In turn, γ, λ are scalar parameters of the model, T is the number of leaves, and ω are the weights of the tree f_t. This term controls the complexity of the model and prevents overfitting. It penalizes the complexity of the trees (through the variable T) and also encourages simpler or even sparse solutions (through L2 regularization in this case, but L1 can be used instead).

The optimization of the objective function $\mathcal{L}^t$ in each step is then performed using some approximations and the gradient of this function with respect to the parameters of the new tree f_t

[20]. This means calculating the best split points for each feature in the newest tree.

The process continues until a stopping criterion is met, such as reaching a maximum tree depth or the improvement in the loss function falls below a threshold.

For example, using the *early stopping* technique, the predictions are evaluated in a separate validation set, containing samples that are not used to calculate the loss gradient and update the parameters during training. When the error in this separate subset is no longer decreasing for a certain number of rounds, the training process is stopped regardless of the fact that $\mathcal{L}^t$ may continue decreasing with more iterations. The validation set is used to test the generalization capacity of the model, so the early stopping method avoids overfitting the training set.

The regularization parameters γ, λ, which also improve generalization (e.g. if the gap between training and validation errors seems too large, they should be increased), can be found by training several variants in a hyperparameter tuning procedure and picking the one with the lowest validation error at the end.

Figure 2.9 shows examples of different balances in the optimization function $\mathcal{L}$, depending on the regularization term $\Omega(f)$ and the training loss value $L(f)$.

In summary, XGBoost combines the techniques of gradient boosting and CART to build an ensemble of decision trees, optimizing an objective function through gradient-based optimization.

2.3.3 XGBoost for Audio Classification

The same audio features that can be used to train an LSTM model can be also used as input to the XGBoost algorithm. XGBoost will learn to classify audio samples based on the features provided by iteratively constructing a set of decision trees, each focused on different aspects of the feature space. The ensemble of these decision trees forms the final XGBoost model, which can be used to predict the audio labels for each feature vector.

It is important to note that models based on decision trees do not have a built-in mechanism to classify sequences of data. In other words, each feature vector that is classified is treated as completely independent of other neighboring samples.

In order to leverage the temporal sequence nature of the audio classification problem, a *Context Window* must be implemented externally. Therefore, when classifying a given sliding window, for example of 20 ms duration, the features extracted for this particular sample can be concatenated with the feature vectors of neighboring samples. For example, using a context window of 5 samples before/after, suppose that there are 20 audio features extracted per sliding window, then the concatenated vector to classify with the boosted tree model will have a dimension of $(5 + 1 + 5) \times 20 = 220$ input features.

One notable aspect of XGBoost is its ability to provide insight into the importance of features. By keeping track of how often and how much each feature is used in the construction of the trees, XGBoost assigns importance scores to the features. This information can be valuable for understanding the underlying patterns in the data and selecting the relevant features.

2.4 Diarization Overview

The goal of diarization is to detect speaker changes in an audio recording and to identify which speech segments correspond to the same speaker (see Figure 2.10), answering the question of who spoke when.

This problem is different from transcription because it focuses on segmentation of the audio and identification of different speakers. Transcription consists of detecting what was said in each part and turning it into text.

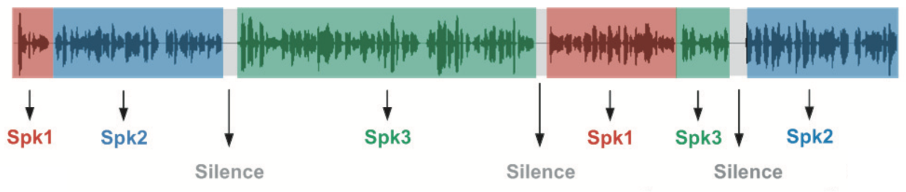


Figure 2.10: Example of audio diarization with 3 speakers [14]

While diarization can be used to complement transcription, to know who said each part, it's important to note that from the point of view of the audio representations (or *embeddings*) that are useful to solve them, they are almost opposite problems:

- For diarization, an ideal audio representation would allow separating different voices, being invariant to the particular words and phonemes of the segment to encode.
- For transcription, the ideal representation would allow one to identify words, being invariant to the particular voice of the speaker.

This means that these representation vectors or *embeddings* for diarization or speaker identification applications are generally very different from those used for text transcription.

It is important to keep in mind this clarification, since recently there have been significant advances in transcription models that use deep learning and large volumes of data, like OpenAI's Whisper model [45] or the models in use by popular platforms like YouTube, to automatically generate subtitles for the videos, which show impressively accurate results.

This may intuitively generate the idea that diarization is simply a sub-problem of transcription that should be easily solved in comparison. However, while diarization systems have also seen great improvements by using deep learning and large amounts of data, it is necessary to understand that advances in each of these problems do not generally transfer directly to each other, considering that the challenges involved, and even the training data used in each case, are generally different. For transcription tasks, even though the manual annotation effort is much higher in general, there are some huge preannotated data sources ready to use, such as audiobooks and movie subtitles.

For diarization, a popular dataset used mainly to evaluate models is *Voxconverse* [34], which was obtained from videos that were processed with a face recognition system, to automatically label audio segments that match the detected speakers. It consists of approximately 20hs of data for development, and 43hs intended for testing. Other popular datasets are *CALLHOME (1997)* [7], which consists mainly of phone recordings of american speakers, and the *DIHARD* dataset [32], which is a compilation of different audio sources with a variety of settings.

2.4.1 Applications, similar problems, and challenges

Audio diarization is often used as a pre-processing step for other applications, or as an end in itself.

For example, in the case of Acoustic Speech Recognition (ASR) for audio-to-text transcription, as mentioned earlier, a diarization or speaker change detection stage is usually performed beforehand to improve context information when performing the speech recognition.

In turn, the result of diarization can be useful in itself, for example, to automatically analyze recorded customer service phone calls, patient-doctor interviews, or classroom activity. The use case that will be emphasized in this work is the automatic analysis of recorded virtual classes to estimate the intervals in which the teacher or students speak.

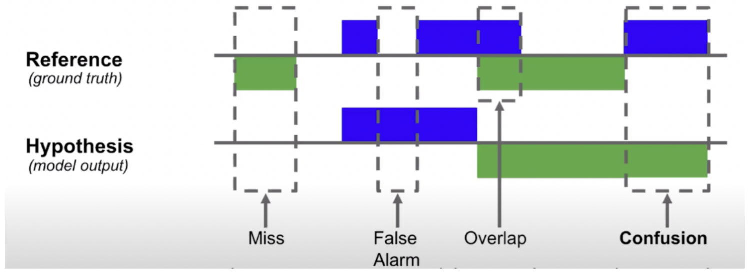


Figure 2.11: Errors that a diarization system can introduce, showing an example with two speakers (green and blue). Source: [28]

A subproblem of diarization is speaker change detection, which, together with voice/speech activity detection (VAD or SAD), allows for speaker segmentation. These can be considered as previous steps because they do not require identifying whether the voices after each change or in each segment had previously participated in the conversation, but rather they are sufficient to delimit the voice segments that potentially belong to different speakers.

Another problem that could be considered a subproblem is speaker verification. In this case, it is required to verify whether two given audio segments belong to the same speaker. The audio representations used to solve this problem can distinguish voices from each other, and therefore can be used as part of a diarization system. However, speaker verification is intended to analyze short audio segments, which contain only one speaker, and it is not important to do any kind of temporal delimitation of the sequence.

A diarization system is basically the combination of the two previous tasks: first delimiting voice segments, and second knowing which ones belong to the same speakers, or to an entirely new speaker who may not have previously participated in the conversation (the number of participants is not known a priori). For this reason, diarization competitions and challenges usually have different tracks, where some focus specifically on solving one of the previous subproblems.

For example, in the *DIHARD 2018* challenge [27], there is the first track where audio segmentation is already provided, and the task is to detect which speaker each segment belongs to, and the second track is for complete diarization. In the *VoxSRC-21* challenge [38], there are the first three tracks for speaker verification, and only the fourth is for complete diarization.

2.4.2 Diarization metrics: DER, JER, EER

The most widely used metric for speaker diarization is the *DER: Diarization Error Rate*, defined in Equation 2.7 as the sum of False Alarm, Miss, and Confusion errors divided by the total number of speech segments in the reference signal ([24], [27]). Figure 2.11 illustrates the types of errors that a diarization system can make in a two-speaker scenario.

$$DER = \frac{\text{False Alarm} + \text{Miss} + \text{Confusion}}{\text{Total}}. \tag{2.7}$$

The overlap component, which represents speech segments in which multiple speakers are talking simultaneously, is usually discarded because including these segments and adding them to the numerator can result in a DER greater than 1 [28]. Furthermore, when audio segmentation is known and provided as input to the diarization system, as is the case in some evaluation challenges mentioned before, only the *Confusion* component is used to calculate the error, since *False Alarm* and *Miss* correspond to segmentation errors, specifically related to the Voice Activity Detection (VAD) process.

Note that speaker IDs in the reference and those detected by the system generally do not coincide. For instance, speakers labeled 1, 2, and 3 in the reference may correspond to those detected by the system, but assigned IDs 2, 4, and 1, respectively. Mapping speakers is not trivial, as it requires assigning speakers whose segments coincide to the greatest extent. Note that following the order of appeareance would not be robust, since one single false speaker detection would shift all subsequent speaker numbers. In order to match the speaker numbers whose segments overlap the most, the Hungarian algorithm [1] for optimal assignment is often used, and that is the case in the open source library *pyannote.metrics* [24], which is widely adopted in industry and academia. This is not a DER-specific problem, but a general one in any metric used to compare these speaker detections with references, as the ID mapping cannot be known a priori.

As a secondary metric for speaker diarization, *JER: Jaccard Error Rate* (introduced in [32]) is often used. It is based on the popular Intersection over Union (IoU) metric, also known as the Jaccard index, which is popular in object detection tasks. In this case, the same principle is applied to the speech segments detected by the system and those in the reference. After optimal mapping, each reference speaker ID is paired with a detected ID, and the temporal Intersection over Union (IoU) is measured for all these speaker segments. The JER for that speaker is then $JER_i = 1 - IoU_i$. Here the subindex i is the speaker number. This value is then averaged over all reference speakers.

The intersection of the segments, which is the interval where they overlap, corresponds to $I = \text{Total} - \text{False Alarm} - \text{Miss}$ (discarding the overlap and not using the Confusion component in this case, as each IoU is calculated between only one detected speaker and one reference speaker after the optimal assignment). The union corresponds to $U = \text{Total}$, and therefore:

$$JER_i = 1 - \frac{I}{U} = \frac{\text{False Alarm} + \text{Miss}}{\text{Total}}$$

Finally, it is worth mentioning another metric known as *EER: Equal Error Rate*, which sometimes appears in some parts of diarization system publications. Although it is not directly applicable to speaker diarization but only to speaker verification tasks, it can be used to evaluate the quality of the voice embeddings (the audio representations mentioned before), which are vectors that should be close together when the voices are similar and distant otherwise.

To measure this in an interpretable way, the rate of *true positives* and *false positives* are calculated, using these vectors with different distance thresholds for the classification task (when the distance between two vectors is less than the threshold, it is considered that they belong to the same speaker). However, to obtain a metric that does not depend on the chosen threshold parameter, one option is to use the area under the ROC curve, or a simpler option is to find the point where the rate of *true positives* and *false positives* coincide, and this coinciding rate value is the EER.

2.4.3 Diarization pipeline

In the following sections, we detail the function of each block of a typical diarization system, with some optional components and different possible configurations, as shown in Figure 2.12.

Audio Features

The first block is feature extraction, which basically means extracting vectors at periodic time intervals, with meaningful values to understand aspects of the signal, according to downstream tasks in the pipeline.

Typically, the waveform of the signal is not directly used, although there are some deep learning models that operate directly at that level. Most likely, the signal is converted into a sequence of feature vectors (see Section 2.1), which could be some statistics of each segment (such as *ZCR:*

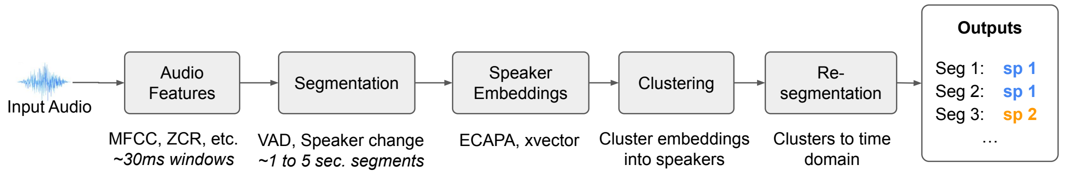


Figure 2.12: Components diagram of a typical diarization system.

Zero Crossing Rate, signal power, amplitude, etc.), or a time-frequency representation such as the spectrogram or Mel frequency bands, or the mentioned cepstral coefficients *MFCC*, which can be useful to reduce the dimensionality of the features.

As mentioned in Section 2.1, these local features are extracted over sliding windows that usually have a duration of between $[20, 30]$ ms for voice processing applications, and an overlap of at least 50% in most cases.

The tradeoff to select an appropriate window size in the case of a time-frequency representation is given by the *uncertainty principle* for signal processing (also mentioned in Section 2.1), which opposes frequency resolution with temporal resolution [6]. This implies that if the windows are much shorter than 20ms, they will not cover enough cycles of the voice signal (mainly in the low frequency components), as to calculate its frequency components precisely (i.e: low frequency resolution), which is visualized as a spectrogram with a lot of uncertainty due to wide frequency bands.

Although the mathematical formulation of this principle only applies to time-frequency transformations, this trade-off must be considered at least qualitatively with any features extracted over sliding windows, as they are approximations that assume that the signal parameters are approximately constant in that interval.

Voice Activity Detection, Overlapping, and Speaker Changes

The second block of Figure 2.12 (*VAD: Voice Activity Detector*) detects voice activity to avoid processing segments where other types of sound such as noise, music, etc., are present. Although not common, some systems do not use a specialized VAD module, but instead calculate embeddings for all segments and eventually discard some by evaluating them a posteriori.

After the VAD, there are further variations in different systems. There may be an overlapping detection module to ignore these segments or process them in some particular way. There may also be an optional segmentation module that detects potential speaker changes (speaker change detection). Other systems do not use any of these optional blocks and instead calculate speaker embeddings for all voice-detected segments, in some cases detecting speaker changes or overlapping with some postprocessing technique (the overlapping problem is often simply ignored).

A very interesting approach that covers the functionality of all the above blocks is the model used by *pyannote.audio*, performing VAD, speaker change detection and speaker overlap detection all at once [37]. This is achieved by training a local speaker classification model with a permutation-invariant loss, which outputs the probability of 3 different speakers restricted to short time intervals (around 3 seconds), taking advantage of the fact that it is very rare to have more than 3 different voices in such a short period of time. This approach will be discussed further.

Speaker Embeddings

The *speaker embeddings* block depicted in Figure 2.12 plays a crucial role in the transformation of audio features at that stage. Its primary objective is to generate a vector representation that remains relatively constant for a specific speaker, regardless of the language sounds and words that they produce. This vector should ideally change only when the speaker changes. In essence, it should exhibit invariance to variations in language, remaining close to any other voice segment uttered by the same individual. In contrast, it should ensure substantial dissimilarity from voice fragments spoken by different speakers, even if they are saying the same words.

As mentioned earlier, this fundamental distinction sets apart voice-to-text transcription systems, where the focus lies on achieving invariance to speaker voices and maximizing distinction between different words.

The segments or sliding windows over which these embeddings are calculated usually last between 0.5 and 3 seconds, and there is a trade-off between using short windows to have good temporal resolution by quickly detecting speaker changes, or using longer windows to generate a good context of a few seconds of audio that allow proper voice characterization and differentiation.

Intuitively, this can be compared to the time it takes a human to distinguish a certain voice. This tradeoff may not be confused with the *uncertainty principle* for signals mentioned before, although there is some resemblance in the discussion. In general these embeddings are not calculated directly over the waveform domain, but instead over a sequence of vectors with the higher-level features extracted in the first block.

For example, in [28], an LSTM receives as input a sequence with a span of 240 ms, where each element of the sequence is a feature vector of dimension 40 representing the logarithmic Mel-filterbank energies extracted from frames of 25 ms, with a step of 10 ms. The output is an embedding that they call *d-vector*, and the neural network is trained using a contrastive loss, so that these embeddings are close to others from the same speaker and far away from other different voices.

Another example using more modern attention mechanisms and Time-Delay Neural Networks (TDNN) is shown in [35], where the input features are vectors containing 80 MFCCs, also extracted over 25 ms frames with 10 ms step. This model known as *ECAPA-TDNN*, is widely used as one of the state-of-the-art embedding models from many speech recognition applications, including diarization (for example, in the *SpeechBrain* library [40]).

Clustering

In the *clustering* stage, all the embedding vectors are taken for each audio segment, and an attempt is made to group them together to generate clusters that contain segments from the same speaker, whether they are consecutive in time or not. Ideally, the number of clusters should match the number of people in the conversation.

For this stage, the most widely used clustering algorithms are *Agglomerative Clustering* and *Spectral Clustering.*

Agglomerative clustering is the “bottom-up” approach to the more general hierarchical clustering [23]. The process begins by considering each data point as its own individual cluster. Then, based on a specified similarity measure, the algorithm progressively merges the closest clusters together to form larger clusters. This merging process continues iteratively until all data points are combined into a single large cluster or until some specified criteria are met.

The sequence of cluster mergers leads to a tree-like representation of the hierarchical clustering process called *dendogram*, showing for example the number of clusters at each similarity level. There is some flexibility to choose the linkage criterion, which is the method used to compute the similarity between clusters during the merging process.

For example, the *centroid* linkage method uses the distance between cluster centroids as the similarity measure, while the *single* linkage uses the minimum distance between data points in each pair of clusters to determine its similarity. There are many other methods, and this adaptability allows the algorithm to be tailored to specific datasets, accommodating different types of data and cluster shapes effectively. Since it does not assume any particular shape for the clusters, it

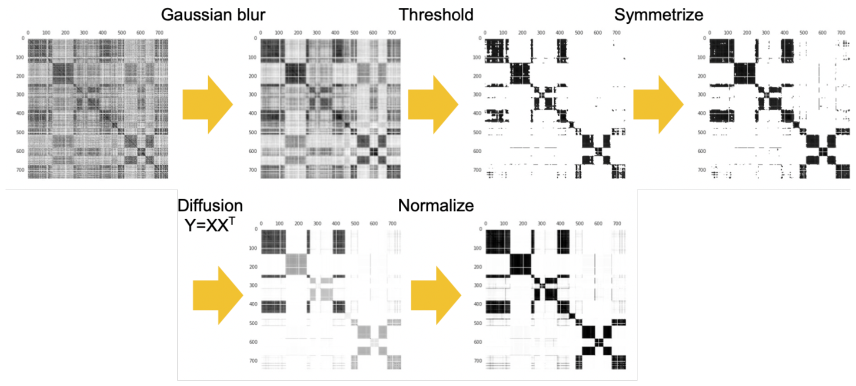


Figure 2.13: Embeddings affinity matrix and refinements applied to regularize it before spectral clustering. The original affinity matrix is shown in the top-left corner, and it can be seen that after the final normalization step (bottom-right), the dark blocks are more clearly defined. Each of these blocks represent zones of high affinity (or low distance) between consecutive embeddings, which indicate that they belong to the same speaker. Source: [28]

is applicable to a wide range of data distributions, making it very versatile and suitable for the high-dimension vectors often used as speaker embeddings.

However, aggomerative clustering does not leverage any information about the temporal position of embeddings. For example, there would be no special consideration for two samples corresponding to consecutive audio segments, and they would have exactly the same probability to be merged into the same cluster as two distant samples. Some diarization models make the assumption that, in most conversations, each speaker will talk for a couple seconds (several consecutive frames) before giving way to another person. Therefore, they impose a structural bias on the clustering algorithm to favor grouping together consecutive samples.

Spectral clustering allows for the implementation of this structural bias, by applying some refinements to the embeddings' affinity matrix (Figure 2.13). In this matrix, the dot product between all embeddings is calculated, where the row and column numbers represent the position of each embedding (i.e., consecutive samples lead to embeddings in consecutive rows and columns). Each cell in the matrix represents the dot product between two embeddings, with dark pixels meaning high affinity (high dot product).

It is worth noting that well-defined segments that belong to the same speaker lead to well-delimited dark blocks in the matrix, since they are areas of high similarity between contiguous embeddings.

Spectral clustering consists of finding lower-dimensional embeddings that still reproduce this affinity pattern between embeddings. The name comes from the fact that in graph theory, this affinity matrix represents the connectivity between the graph nodes (in this case, each node would be a sample's embedding) and the clusters or "node communities" are identified by analyzing the "spectrum" of the matrix (i.e., the eigenvalues and eigenvectors of the matrix in graph theory).

However, before performing the spectral decomposition of the affinity matrix, some refinements are applied to it, in order to improve the definition of the dark blocks that represent each speaker's intervention. The regularized matrix is less prone to spurious patterns that would generate intermittent changes in the speaker, and this is assumed to be a positive thing in most conversations, as mentioned before.

After the refinements, the matrix is diagonzalized using the eigenvalues. Since the affinity matrix is symmetric (the dot product is commutative, so $A_{ij} = Aji$), this decomposition is of the form:

$$S = P \cdot D \cdot P^T, \tag{2.8}$$

where D is a diagonal matrix with the eigenvalues and the rows of P (or columns of P^T) are the eigenvectors associated with those eigenvalues. If we assume that $S \in \mathcal{M}^{n \times n}$ (where n is the number of audio segments on which the embeddings were computed), the diagonalization also results in three matrices $n \times n$.

However, only a subset $m < n$ of the eigenvalues can be used, and the corresponding rows and columns can be eliminated from P to obtain $P_r \in \mathcal{M}^{n \times m}$, $P_r^T \in \mathcal{M}^{m \times n}$, and $D_r \in \mathcal{M}^{m \times m}$ such that, when multiplied, the resulting matrix S_r still has dimension n, and becomes increasingly similar to S as $n \longrightarrow m$. By filtering a certain number m of eigenvalues, an approximation of S (as good as desired) is obtained, but more importantly, a matrix $P_r \in \mathcal{M}^{n \times m}$ is obtained, where each of the n audio segments corresponds to a row in the matrix, which can be seen as a new embedding for the segment, with reduced dimension m.

Finally, a clustering algorithm such as *k-means* is used to group embeddings with reduced dimension. The essential aspect of this spectral smoothing and clustering procedure is that a structural bias is introduced into the system, favoring embeddings close in time to belong to the same cluster (same speaker), and at the same time filtering out noisy detections (spurious dark blocks in the affinity matrix).

A widely recommended source of consultation on this topic is “A Tutorial on Spectral Clustering” by Ulrike von Luxburg [12].

At first hand, it is not clear that introducing this temporal structural bias in the system would lead to better results in the case of the diarization system to develop in this work, because most frequently, interventions from students are actually very short in time. A structural bias that favors longer interventions could lead to undetected student participation.

Resegmentation

After the embeddings are assigned to different clusters that correspond to each speaker, the problem is almost solved. If there is no overlapping between the segments on which the embeddings were calculated, the resegmentation process would only consist of positioning each cluster number into the corresponding embedding segment.

However, in most cases, the embeddings do have overlapping, and hence there are multiple clusters to choose from at any given instant. There must be some sort of aggregation method to decide which cluster to choose from. In the open source diarization pipeline available from *pyannote.audio* [33], this is implemented by creating a separate channel for each cluster: the level of the signal in each channel is an integer that counts the number of overlapping segments that were assigned to this cluster. Then the signal with maximum value is chosen and its corresponding cluster is assigned to that interval. All in all, this is basically a vectorized implementation of a majority voting mechanism, between all the overlapping clusters at a given timestamp.

The result of this stage is a sequence of time-bounded segments with the assigned speaker number as the label. By adding up the duration of each speaker’s segments and then selecting the one with the largest sum, the teacher can be assigned to this particular speaker, while all the other segments can be assigned to the generic student label. This process will be discussed further in the implementation section.

2.5 Chapter summary

In this chapter, we established the methodological foundations to understand the upcoming sections. Further practical details about how the specific models were implemented, will be provided in following sections.

So far, we explored crucial aspects of speech processing, including the vocal tract model, Short-Time Fourier Transform (STFT), and Mel-Frequency Cepstral Coefficients (MFCC). These audio features enable us to capture distinct characteristics of adult and children's voices in challenging audio environments.

Furthermore, we introduced the main machine learning models used in this work: XGBoost, a versatile classifier, and LSTM, suitable for sequence classification tasks.

Additionally, we discussed the diarization pipeline, on top of which we intend to develop an unsupervised classroom activity detection system. We provided some details about the most usual metrics used (which are not necessarily the same as will be used in this work, since this is not a diarization system), and described some example components of the pipeline, like embedding models and clustering algorithms.

The next chapter will delve into data annotation, analysis and preprocessing steps, to create the training and testing datasets, to develop and compare our different implementations. Also, the evaluation metrics that will be used for this comparison are introduced and discussed in detail.

Chapter 3

Experimental Setup

In this chapter, we lay the foundations for evaluating and measuring the effectiveness of the different approaches to CAD that are implemented in the following sections. We begin by presenting the available data and introducing the data annotation protocol, specially designed for this work.

To assess the performance and generalization power of our supervised approaches, we carefully explain the data splitting process. The data is divided into training and test sets, but also creating five groups of audio from different lessons. This grouping allows us to measure the system's performance on known and unknown voices or lessons, and with increasing training data.

Another critical aspect of the evaluation process is the selection of appropriate visualizations and metrics. We introduce label density as a function of time, a novel approach to show the CAD results that provides a more concise representation of the activity dynamics in the virtual classroom environment. It is particularly well suited for very fragmented interventions that are typically observed in primary school lessons. Additionally, we justify our choice to evaluate density estimation using the correlation coefficient instead of regular regression metrics.

Throughout the chapter, we illustrate various examples to highlight the challenges encountered during the experimental setup and the solutions devised to overcome them. And finally, we calculate these metrics on a subset of the data that was redundantly annotated by three different human annotators, in order to calibrate expectations. By meticulously designing the experimental framework, we aim to provide robust and meaningful insights into the performance and capabilities of all our CAD models.

3.1 Data and labeling

The data for this work consist of 25 video recordings, composed of 20 in *Pensamiento Computacional* and 5 in *Ceibal en Inglés*. These videos are referenced throughout the text by their keys **pc1, ..., pc20** and **eng1, ..., eng5**, having a one-to-one correspondence with their actual file names.

The duration of each of these videos is approximately 45 minutes. Although some recordings span more than 1 hour -presumably accidentally- the actual class duration varies between 40-50 minutes, with irrelevant video/audio at the beginning and the end.

3.1.1 Manual annotation protocol

The audio from these classes was labeled for this work, using two separate tracks: *speaker* and *ambient*. The former is the most relevant since it contains the actual classification labels, and the latter is just used as an optional mask to indicate when the classroom environment is noisy and disorganized, which might lead to less accurate or ambiguous labels in the first track.

The speaker track, contains the following labels:

- **p:** Teacher's voice, with no gender distinction.
- **a:** Student's voice, gender is not apparent from the voice.
- **b:** Student's voice, apparently boy.
- **g:** Student's voice, apparently girl.
- **m:** Multiple voices speaking simultaneously (e.g: group work or answering questions)
- **c:** Multiple voices speaking synchronously (e.g: choir-like short responses)
- **l:** Local teacher intervention.
- **o:** Other audio sources (e.g: music, recordings)
- **n:** None of the known categories.

The ambient track, might optionally indicate:

- **d:** long disorganized classroom activity, which does not seem intentional regarding the proposed task.
- **r:** Sudden short noise masking some label from the speaker track (e.g., moving furniture, distortion).

In practice, the labeling process can sometimes be ambiguous when the classroom activity is unclear due to overlapping situations. So, for example, while the teacher is speaking (label *p*) it is quite frequent to hear short interventions from students (labels *a, b, g, m*), sometimes with very short duration. It is even more frequent to hear the voices of other children when there is participation from the students, which makes it difficult to select between the labels *a, b, g*, or the *m*. As a general criterion, it was decided that labels with a duration less than 1 second can be avoided during labeling, in order to make the process more straightforward. Since the labeling resources are quite limited for this work, it is important to optimize the process to get the highest value from it, and delimiting all short fragments can be a time-consuming task with low benefits for the purpose of this work.

In order to have an estimate of the effort that it takes to manually label a recorded class, the human time required in each case was logged in a spreadsheet. The observed values vary greatly depending on the organization of the class and the experience of the annotator. The median value is around 1.5 hs of effort to annotate an entire audio recording (approximately 45-50 minutes long, as mentioned before), with some instances taking from 1 hour to 2.5 hs to fully annotate.

The total audio duration of all these lessons (discarding any accidental recordings greater than 50 minutes) is **18.25 hs**, and the label coverage is about **81%** considering only the speaker track, which adds up to **14.7 hs** of annotated audio in total. This coverage is approximately constant on each audio, and the excluded segments correspond mostly to the beginning and end of the lesson (students entering or leaving the classroom, etc.) and silences.

3.1.2 Simplifying labels: remapping

The annotation protocol defined previously has a high granularity in the types of labels defined. This is because since the annotation process is manual and requires a lot of effort to listen to the audios many times at each interval, it is not a significant overhead to add some more specific labels that might be simple to distinguish by a human (as long as the list does not become too long or complex). Also, some of these labels might be useful beyond the needs of this particular work.

However, the main focus of this work is to distinguish two situations: whether the teacher or the students are speaking. As a secondary objective, it is also useful to distinguish when a single student is speaking, from the moments when there are many students speaking at the same time, or the kind of background voices and noises that are present when the students are working on some task while the teacher waits for it to complete.

For evaluation purposes, only the following 4 labels are considered, and all other labels are remapped to one of them as indicated:

- **p**: Includes **p, o**. This avoids detecting other sources of audio (e.g: music, recordings) as students' participation.
- **a**: Includes **a, b, g, l**. This basically groups any single speaker in the classroom, that is not the remote teacher. The local teacher participations are almost negligible in practice.
- **m**: Includes **m, c**. Any kind of multiple or background voices. The aim is to associate this label with students working in groups or individual activities.
- **-**: Null label, includes **n**, any non-labeled or silence intervals. To ease the manual annotation process, some silence pauses in speech are left as part of a single longer segment, but then these small differences are ignored in error metrics by forcing any segment below a 30 dB to this *null label*.

3.1.3 Data splitting

In order to evaluate supervised methods, we need to split the data at least into *train* and *test* sets, in the first place. But also, one of the objectives of this work is to understand how the amount and type of annotated data affects the performance of the models, and to gain a better idea of the generalization power of the solution. For example, it is important to know if a model is able to identify new teacher's voices, even when there are no samples of their particular voices in the training set.

To answer those questions, the 25 recordings (with different teacher's voices) are divided into five groups of five lessons. Since teachers are not repeated, this means that the groups contain disjoint teacher voices. Each of these groups is divided into train and test.

Since we have five test groups containing different teacher voices, we can calculate five independent values for any performance metric, not only to get their mean value but also to understand the variability of the predictions on different kinds of voices and lessons.

Furthermore, in order to understand how the annotated time per lesson impacts the performance of the supervised models, the training set is further divided into five splits, which can be added incrementally. In quantitative terms, 50% of each audio is assigned to the test set (named *Split 0*), and the remaining 50% is further divided into 5 train splits (*Split 1..5*). This means that the training data can be added in steps of 10% of the entire dataset.

The reason to reserve this unusually small fraction of the data for training purposes and so much for testing, will become clear when analyzing the results. In short, small increments in the amount of training data were required in order to find the point where the supervised approach is comparable to the unsupervised one.

In summary, there are 5 groups of audios, each one divided into 6 splits (*Splits 0..5*). Table 3.1 shows the amount of annotated data (in seconds) of all these subsets, and the total amounts after aggregating all available train and test data.

The first two columns show each of the five groups with the corresponding classroom recordings that are combined to create them. At the bottom of the table, it is shown that the total annotated time in the entire train set is 26698 s (7.4 hs) and in the entire test set is 26432 s (7.3 hs). The small

Group	Lessons	Test	Train				
		Split 0	Split 1	Split 2	Split 3	Split 4	Split 5
1	eng1,pc3,pc7,pc12,pc17	4949	1086	1074	1074	1109	1111
2	eng2,pc2,pc8,pc13,pc18	5162	1030	1014	998	1048	1061
3	eng3,pc4,pc10,pc19,pc20	6010	1195	1203	1125	1164	1208
4	eng4,pc5,pc9,pc14,pc15	5469	1144	1082	1128	1073	1022
5	eng5,pc1,pc6,pc11,pc16	4842	958	963	918	925	983
All groups		26432	5413	5337	5244	5319	5385
			26698				

Table 3.1: Duration -in **seconds**- of each group/split in which the data is divided. Split number is not related to the position of the subset in the original audio.

difference between the two groups is because the split times are calculated using both speaker and ambient track annotations, but the table times only consider the speaker track annotations.

The numbering of the splits is not related to its position in the original audios, so it is important to note that the test split (Split 0) is not necessarily at the beginning of the class recordings. The exact mapping of the splits and their position in the original lesson audio are shown on the data map in Figure 3.1.

One problem that may arise when a dataset is divided is that the resulting subsets might not have the same balance or proportion of each label as the whole dataset, in which case the data is said to be unbalanced. The grouping and order of the splits in the audios were chosen to make the train and test splits as balanced as possible, following the procedure described in the next section.

Creating balanced splits/groups

Since this is a sequence classification task, each sample needs a surrounding context to be classified and, in general, there is overlap between the sequences that are provided as context to classify each sample. Considering this overlapping between sequences, the standard *stratification* techniques to create balanced subsets, cannot be used (i.e., it is not possible to pick individual samples from the whole dataset to create train/test sets in such a way that the proportion of labels is maintained, because samples may overlap), because it would lead to leaks between those sets. Hence, splitting of the data in sequence prediction problems needs to partition the input sequences into two or more chunks, and map them to train or test as a whole, so that the resulting samples in each set are consecutive.

In this work, the procedure to create the splits and groups from the input lessons recordings is entirely manual and fixed, in order to obtain subsets that are approximately balanced and with similar amount of labels:

1. Each lesson recording (approx. 45 minutes length) is manually assigned a group number.
2. Each lesson is manually assigned a split ordering (e.g., $[1, 0, 5, 3, 4, 2]$), indicating the position of each split in the original audio.
3. The labeled duration of each recording (from the first label to the last label in the audio) is divided into 2, to obtain the train/test durations.
4. The train duration is further divided into 5 segments, with the same labeled duration.
5. The original audio and labels are divided into chunks from beginning to end, assigning each chunk to a split according to the order of step 2, with the corresponding duration.
6. To create a group split, filter the lessons that belong to the group, filter the desired split interval from each lesson, and then concatenate the audio and the annotations from all those chunks.

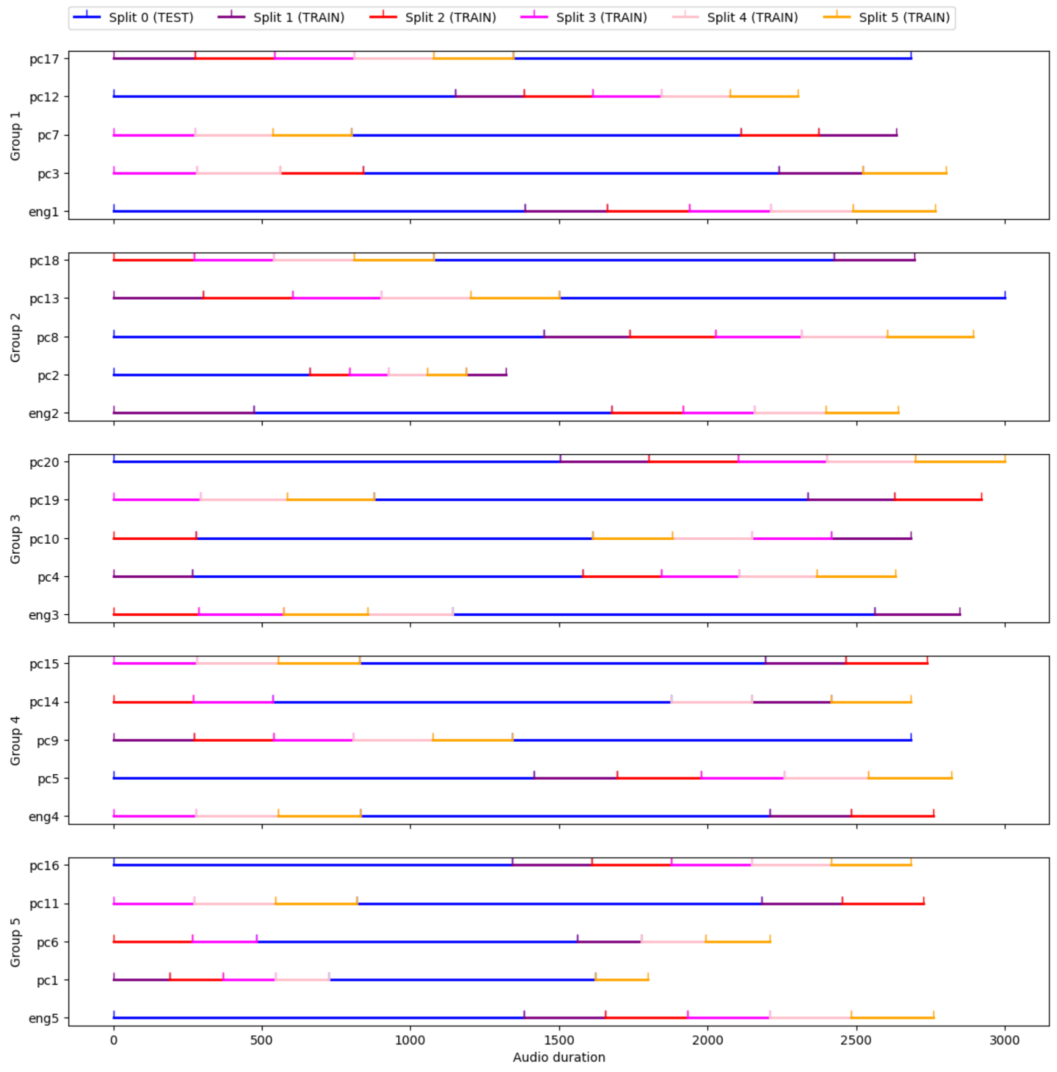
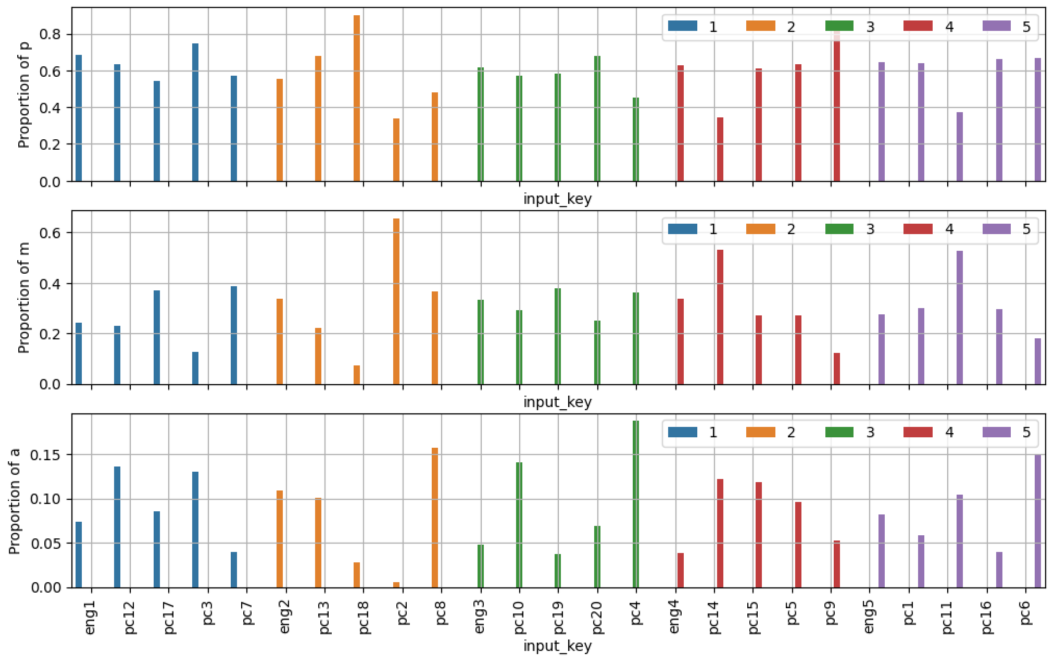


Figure 3.1: Position of each split in the original audios. The test (Split 0, in blue) and train segments can be allocated in different positions to avoid biases towards some particular part of the class, and to balance the label proportions throughout groups and splits.

The groups composition and the split numbers are configured manually by observing the resulting label balance. Label proportions are fixed in each lesson recording and split, but can be combined in groups and splits in order to mitigate this unbalance. For example, by observing Figure 3.2 (top), the groups' composition (*input_key* with the same color) can be changed to maintain the average fraction of each label approximately equal. The results are then verified by checking the bottom plot, and in case there is an unbalanced group in some label, the upper plot can be revisited to check what input keys could have their group assignments swapped. The images shown correspond to the final assignments, which led to appropriately balanced groups, as can be seen.

After balancing the groups, the splits can also be balanced using a similar strategy, by observing Figure 3.3. In this case, when there is an unbalanced split, each input_key needs to be examined to find one in which the split composition is similarly unbalanced. For example, if the resulting



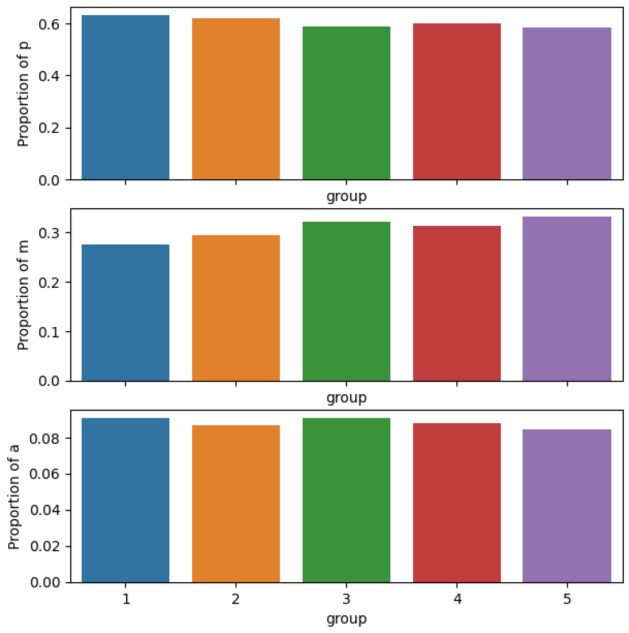
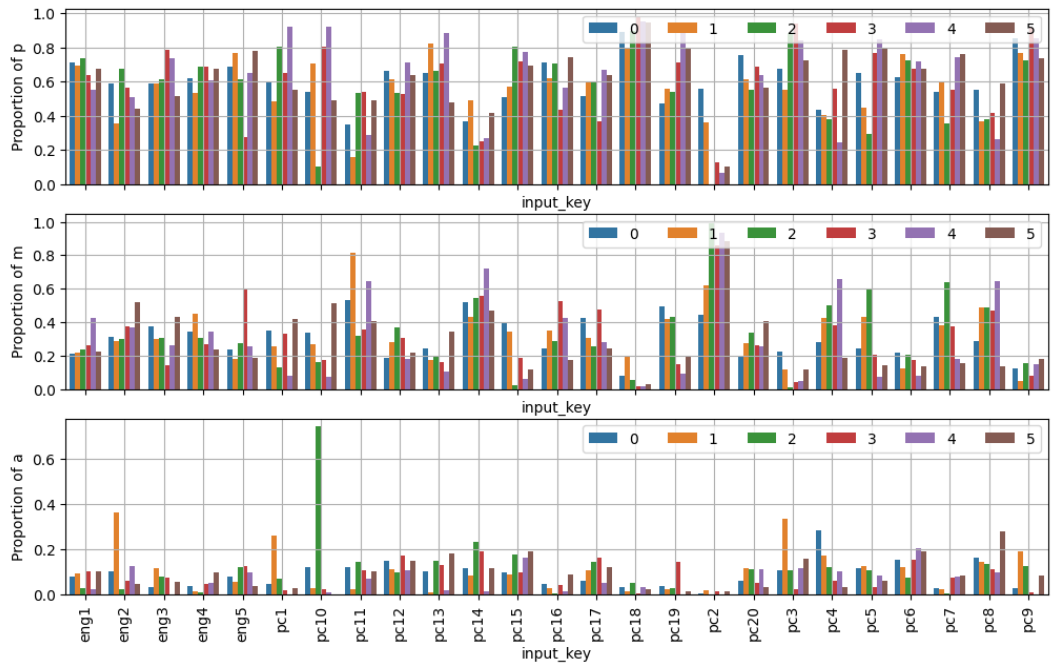


Figure 3.2: **Top:** Balance of labels per lesson recording file *input_key*, and their assigned group (colors 1-5). **Bottom:** Resulting balance per group and label (the colors correspond to the groups on the top image).



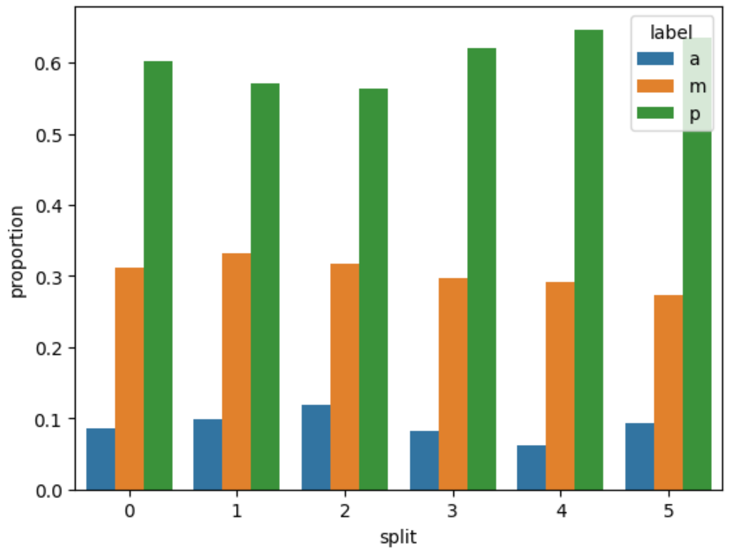


Figure 3.3: **Top:** Balance of labels per lesson recording file *input_key*, and split number (colors 0-5). Split 0 (blue) is always assigned to the test set. **Bottom:** Resulting balance of labels per split (colors do not correspond to the split colors on the top image).

split 5 contains more p time and less a time than the average, then we need to examine individual lessons and find which one is causing the problem, and then change the split ordering accordingly to reverse the balance.

After this procedure, the resulting subsets are at least approximately balanced at an aggregated level: groups are balanced considering all splits, and splits are balanced considering all groups. In particular, Split 0 (test) has a similar balance to the average of the train splits.

3.2 Evaluation metrics

In this section, we explain the evaluation criteria used to assess and compare the results of our models. In particular, we discuss the reasons for using a different visualization style based on plotting label densities as a function of time, instead of the usual CAD or diarization graphics (as shown in Figures 1.2 and 2.11). Also, we explain why the reliability of density estimations is measured in terms of the correlation coefficient instead of more standard regression metrics.

Despite the fact that a diarization system is being used as one of the approaches to CAD in this work, the standard error metrics used in those problems (e.g., *DER*, *JER* and *EER* mentioned in Section 2.4.2), were not deemed useful after a deeper consideration. In particular, because exact temporal precision is not a requirement in this case (as mentioned in the motivation section), and since these metrics are based on matching predicted and labeled segments in their exact position, they turned out to be poorly informative and unintuitive to understand the performance of each model.

Also, since the main objective is to distinguish the teacher's voice from student activity, it is of interest to know what kind of errors the models are more prone to. Errors in predicted labels are more important when the labels a and p (student and teacher) are swapped than when there is a mistake between a and m (single or multiple students) for example. To assess these errors, we also describe in detail the confusion matrix visualization that is used throughout the following sections.

In light of these requirements, there were several iterations testing different metrics and ways to visualize the results for this work.

3.2.1 Label density estimation

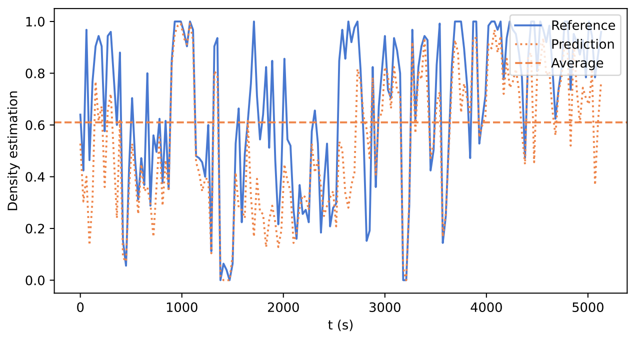
One of the challenges of evaluating the predictions for a 45-minute lesson is that the segments in which students participate are often very short (i.e., duration of a couple seconds), so it is really hard to visualize those intervals at the time scale of the entire lesson.

Verifying the predictions for a single lesson would require one to zoom in shorter intervals and slide through the whole session looking at how the predicted segments match the reference annotations. Aside from the fact that this requires interactivity (which is only feasible for an output video as shown in Figure 1.1), it quickly becomes unmanageable to process many lessons.

An alternative visualization which significantly improved the evaluation process was to create a moving window (with a duration of 30 seconds in this case) and calculate the *label density* in each window: add the duration of all segments of a given label inside the window and divide it by the window length (one density per label). The equation is as follows:

$$D_t(L) = \frac{1}{W} \sum_{k=t\cdot W}^{(t+1)W} \mathbb{I}(\text{label}_k = L). \tag{3.1}$$

Where label_k is the discrete label in a sampling step k, hence $\mathbb{I}(\text{label}_k = L)$ takes a value of one on all occurrences of label L inside each window of length W starting in sample $t \cdot W$.



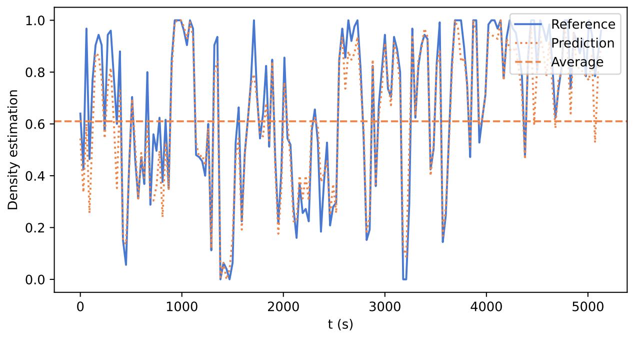


Figure 3.4: Examples of the density estimation for label *p* in the same audio. The bottom image shows a better estimation compared to the top image, where some of the peaks are not correctly detected. The horizontal line corresponds to the average value of the reference density.

Examples of this density estimation can be seen in figures 3.5 (for student's voice label $L = a$) and 3.4 (for teacher's voice label $L = p$). Aside of providing a concise and intuitive representation, estimating this density function also removes the need for precise segmentation (which is not required for our use case as mentioned in Section 1.1.1), as long as the average duration of each label is correctly estimated on each window $D_t(L)$.

This means that if a model makes predictions that do not exactly match the boundaries of reference annotated segments, but statistically produces more segments of type a in zones where there is high student participation, it will be evaluated as a good model by looking only at the density estimation. This is actually a more robust way to evaluate a model since the labels for student participation are usually ambiguous due to overlapping with the teacher's voice, and so detecting zones of high student participation is more important than detecting exactly the same boundaries that were labeled by humans.

For evaluators who need a general idea of the lesson development and to quickly traverse the recording to listen to specific sections, this visualization is much more useful than a fine-grained classification per segment. Therefore, instead of measuring the overlap between segments, the main metric to use will be based on how accurate this density estimation is.

Metrics to evaluate the density estimation

Looking at the problem as an approximation of a real-valued function, the initial idea that usually comes to mind is to judge the estimation using regression metrics. As such, several of these metrics were studied and compared, including scale-dependent (i.e. absolute error or distance between

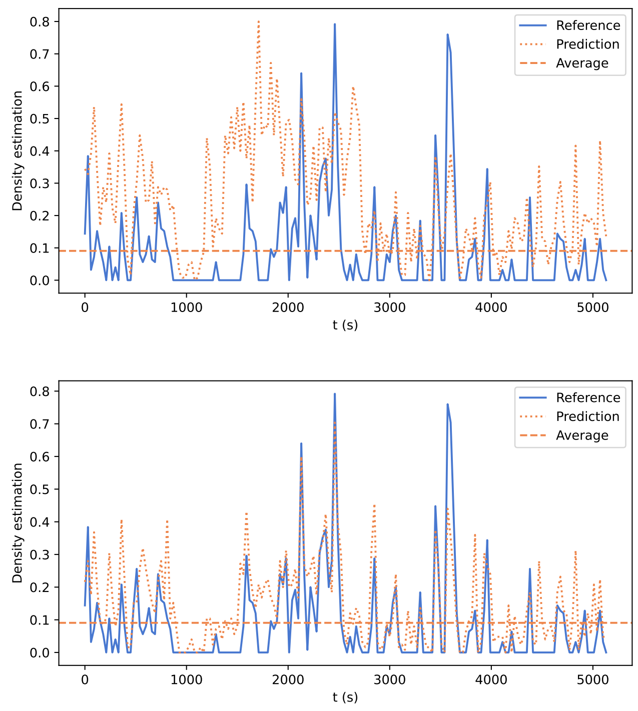


Figure 3.5: Examples of the density estimation for label *a* in the same audio. At the top image there's a bad estimation compared to the bottom one. In the latter example, the peaks of the predicted density match pretty accurately the peaks of the reference labels. The horizontal line corresponds to the average value of the reference density.

curves) and scale-independent (i.e. relative errors, like percentage or normalized to range [0-1]) [22].

In the first category, there are some well-known metrics like *RMSE: Root Mean Square Error* and *MAE: Mean Absolute Error*. Other metrics like *MSE: Mean Square Error* are not ideal because its value is not comparable to the difference in the signals, since the magnitude corresponds to the squared error (i.e: the error unit is not the same as the unit of the signals).

From these alternatives, the preferred metric is **MAE**, since it is intuitive and straightforward to interpret (and does not have a bias to over-penalize larger errors like the RMSE does due to squared terms in the summation). For label L, we can calculate MAE as:

$$\text{MAE(L)} = \frac{1}{T}\sum_{t=1}^{T}|P_t(L) - R_t(L)|. \tag{3.2}$$

Where $P_t(L)$ and $R_t(L)$ are the predicted and reference densities for label L, respectively, calculated according to 3.1. In our use case, MAE will be restricted to the interval $[0, 1]$, since the densities P_t, R_t only take values in this range.

The problem with regression metrics

Despite being a straightforward metric to interpret, it is important to keep in mind that MAE is a scale-dependent metric, since multiplying the reference and predicted densities by a scalar will also scale the MAE value. Therefore, signals with larger average values tend to show larger errors if the distance between curves is similar in relative terms.

In particular, the scale of the signal must be considered when the densities of the label p are compared to m or a, the former showing larger absolute errors in general because the function takes larger values, even though the approximation might appear to be much better in relative terms when looking at the plots.

The first approach to try and solve this problem is to use relative error metrics such as *MAPE: Mean Absolute Percentage Error* and *SMAPE: Symmetric MAPE* [5], which are basically the MAE divided by the reference value at each point or the sum of the reference and predicted values, respectively.

However, both of these metrics have problems when dealing with signals that may often take zero values. In the case of MAPE, the metric is not even defined when the reference value is zero, and the SMAPE solves this problem by adding the predicted value in the denominator, but still the error is 100% at these points, leading to an overestimation of errors for these kinds of signal.

This problem is relevant in this use case because the predicted densities for label a are usually very low or null. As an example, the SMAPE (using the standard definition in [5]) for the signal at the bottom of Figure 3.5 has a value of 55.1%, which does not seem intuitive from the differences observed in the signal. For reference, the image at the top of Figure 3.4 has a SMAPE of 16.7%, even though the errors are at least comparable. The high value in the first case is due to the reference signal taking null values at several points, which adds an error of 100% even if the predicted value is very small but not null.

Other variants of SMAPE and MAPE have been implemented and tested to try to overcome this problem, some of which use an average value of the whole signal in the denominator (such as the metric known as Weighted-MAPE or WMAPE), raising another set of issues. More complex relative error metrics like *MAAPE: Mean-Arctangent Percent Error* [22] were also tested, showing results that were not convincing or not easily interpretable.

After a thorough case-by-case analysis, a deeper problem was identified due to the very nature of regression metrics, including the already discussed scale-dependent MAE. To contextualize this problem, let us compare the MAE values in Figure 3.5, all of which belong to the same label a so there is no issue of scale dependence. The values are as follows:

- MAE for *bottom* image prediction: 0.07916.
- MAE for *top* image prediction: 0.1802.
- MAE predicting *Average* value (horizontal dashed line): 0.09871.

As expected, the lowest error is for the *bottom* image, which shows a good prediction where most peaks would be actually useful for a person trying to find moments of high student activity. However, this value is very close to the MAE measured by comparing the *Reference* signal against the constant function with the *Average* value.

In fact, the constant average signal has a much lower error than the signal in the top image. This may actually make sense for a regression metric, but when considering that the aim of this system is to provide the observer with useful time markers to find particular moments of the lesson with different degrees of participation, it becomes evident that a constant signal is completely useless for this purpose.

Even though the predictions of the top image are bad compared to the bottom example (the discussion is still about Figure 3.5), the curves are in both cases informative about the general development of the lesson. The point is that both predictions in the mentioned image are informative, while the constant value is not. Then a metric should be chosen accordingly to indicate how

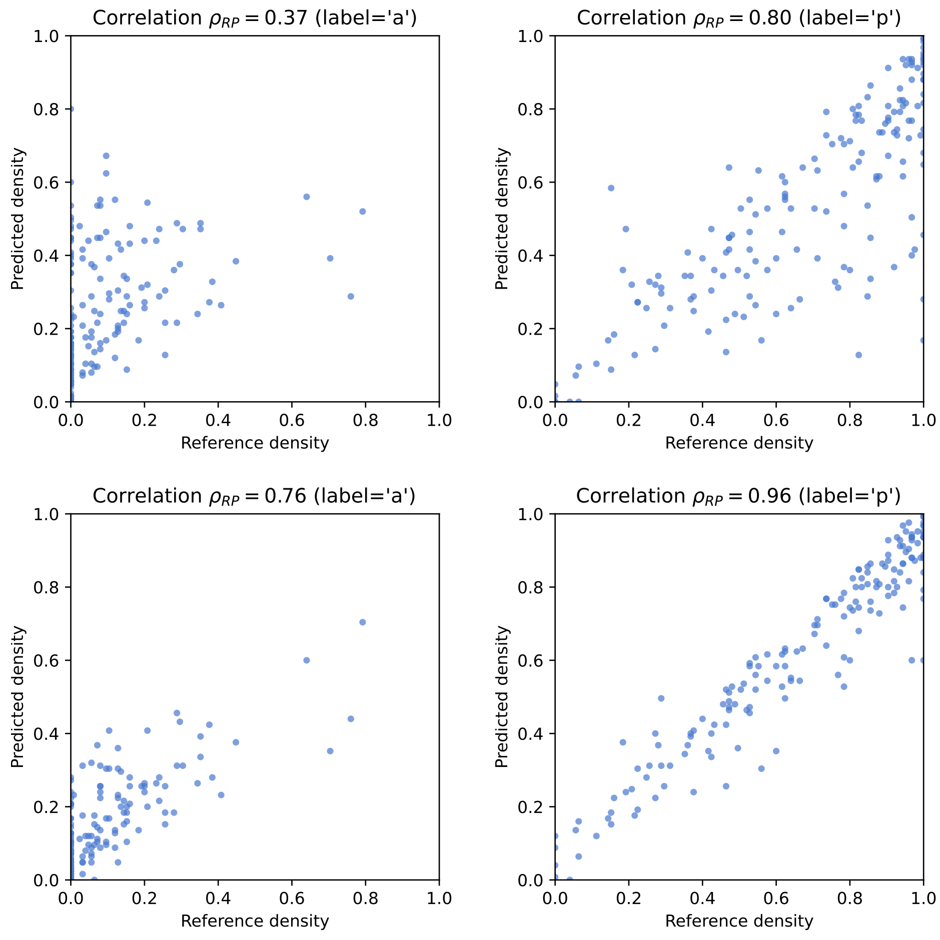


Figure 3.6: Alternative visualization to show the correlation between the predicted P_t and reference R_t values of the density estimation. The **Pearson correlation coefficients** are included in the title for each case. Note that the temporal dimension is lost.
Top images: correlations for the top examples of figures 3.4 and 3.5 (labels *a* and *p*), which corresponded to bad density estimations.
Bottom images: correlations for the bottom images of the same figures, which had better density estimations for labels *a* and *p*.

informative the predicted curve is, despite the possible over/under-estimations. This is not the aim of regression metrics, either absolute or relative.

The metric chosen to indicate how informative is the predicted density, is the widely known **Pearson correlation coefficient**, defined as [39]:

$$\text{corr(R,P)} = \frac{\text{cov(R, P)}}{\sigma_R \sigma_P} = \frac{\sum_t^T (R_t - \bar{R})(P_t - \bar{P})}{\sqrt{\sum_t^T (R_t - \bar{R})^2}\sqrt{\sum_t^T (P_t - \bar{P})^2}}. \tag{3.3}$$

Where $\bar{R}, \bar{P}$ are the average values of the respective reference and predicted densities, and T is the total number of samples or steps t. The numerator in the first equality is the covariance between both signals P_t and R_t, and the denominator is the product of their standard deviations.

Recall that covariance measures the extent to which the signals vary together. A positive covariance indicates that the signals tend to increase or decrease together, while a negative covariance indicates that they tend to increase or decrease in opposite directions. Values near 0 indicate that the two signals vary independently and are not related.

The magnitude of the covariance indicates the strength of the relationship between the signals, but it also depends on the variance of each signal. By dividing between the standard deviations, we obtain this normalized correlation coefficient in range $[-1, 1]$, that can be used to compare signals regardless of the scale.

Figure 3.6 shows examples of higher and lower correlations, corresponding to the same signals in Figures 3.4 and 3.5. Note that the highest value corresponds to the best prediction, which is visualized as a line near the identity function. The low values correspond to point clouds where a difference in the horizontal axis does not lead to a difference in the vertical axis with high probability. For example, a horizontal line (i.e: predicting a constant value) leads to null correlation, which is coherent with the fact that it does not provide any useful information about the development of the lesson.

3.2.2 Confusion, Precision and Recall Metrics

Raw Confusion Matrix

Another visualization that is very useful for multi-label classification problems like this is the confusion matrix plot, which aims to clearly show which labels are being confused by the model.

From the previous density estimations, when there is an error in the prediction for a label, it is not clear why the error happens, what is exactly the other label that the model is choosing instead of the right one. It is possible to estimate it by looking at the density functions for all labels, and compare what label is being underestimated vs. overestimated. But in order to see the more general patterns of the model, it is better to have a more compact visualization for this.

Figure 3.7 shows the most basic form of the confusion matrix, which just counts the number of samples that fall in each cell. For example, we see that there are 12902 samples correctly classified as p, 3446 correct predictions for label m, and only 1139 for a. In turn, there are 1093 samples misclassified as a, since they were actually annotated as m.

Normalizing the Confusion Matrix

The main problem with the unnormalized -i.e., raw- confusion visualization, is that it depends on the total number of samples. These numbers will change for another audio with different length. One way to overcome this, is to normalize by the total sample count in the whole audio, which will result in a matrix whose entries should add up to 1.

Reading such a matrix would answer the following question: what's the probability of a sample being annotated with some ground-truth label and at the same time, being predicted with the given predicted label? The result, it turns out, is highly dependant on the label balance, and is very likely to lead to false conclusions. Also, the information it provides is not very relevant.

The problem is that label p, which has 60% of the total sample count, will always have a greater probability to be chosen in the first place, and also any random model is more likely to classify it correctly just by chance. As a result, even if the classifier is actually far better distinguishing the label a (only 10% of the samples), the metrics will still show a much greater probability to select and classify correctly a sample with label p.

As an illustrative example, a perfect classifier would show a probability $P(pred = p, ref = p) = 0.6$ in the teacher's cell, and $P(pred = a, ref = a) = 0.1$, even though the probability to hit the correct label is 1 in both cases.

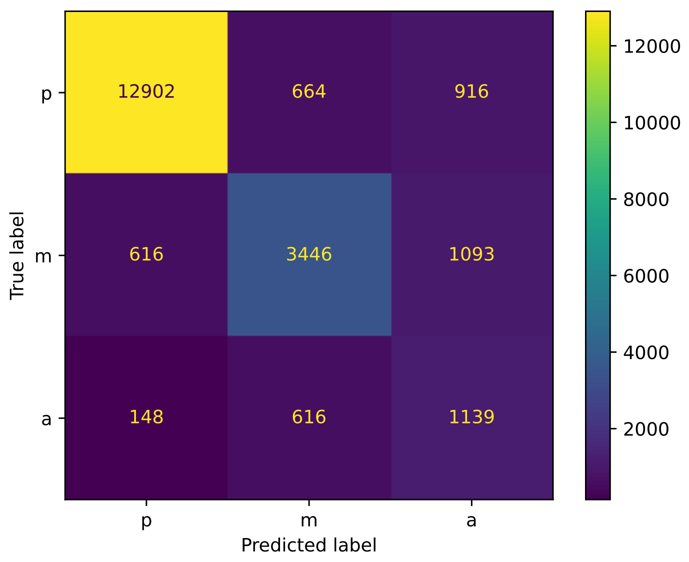


Figure 3.7: Unnormalized Confusion Matrix between labels, showing the total sample count per cell. Each row corresponds to a ground-truth label, while columns indicate the predicted labels. For example, the cell in the first row (top) and second column, shows how many samples with label *p* where actually classified as *m* by the model.

A better metric to evaluate the classification performance is: given a sample with some ground-truth label A, measure the probability that it will be classified as B. For example: if we are looking at a sample labeled as a, show the probability that the classifier will predict it correctly, which can be written as $P(pred = a|ref = a)$ (the pipe symbol means "given that", which is the standard notation in probability theory).

Note that the probability conditioned to an event, multiplied by the probability of the event itself, is the combined probability used before:

$$\begin{aligned} P(pred = a|ref = a) \times P(ref = a) = P(pred = a, ref = a) \\ \implies P(pred = A|ref = B) = \frac{P(pred = A, ref = B)}{P(ref = B)}. \end{aligned} \tag{3.4}$$

The last line is the well-known Bayes' equation. This division can be readily calculated from the unnormalized confusion matrix:

$$P(pred = A|ref = B) = \frac{count(pred = A, ref = B)/total}{count(ref = B)/total} = \frac{count(pred = A, ref = B)}{count(ref = B)}. \tag{3.5}$$

Calculating this is equivalent to **normalizing by row**, since $count(ref = B)$ is the sum of all cells with the reference label -True label- B.

Figure 3.8 shows the confusion matrix for the same predictions that were in the previous raw matrix, but normalized by row in this case.

These results are better to interpret. For example, 89% of the teacher's voice is correctly covered by the classifier. 60% of the student samples are correctly classified and only 7.8% are

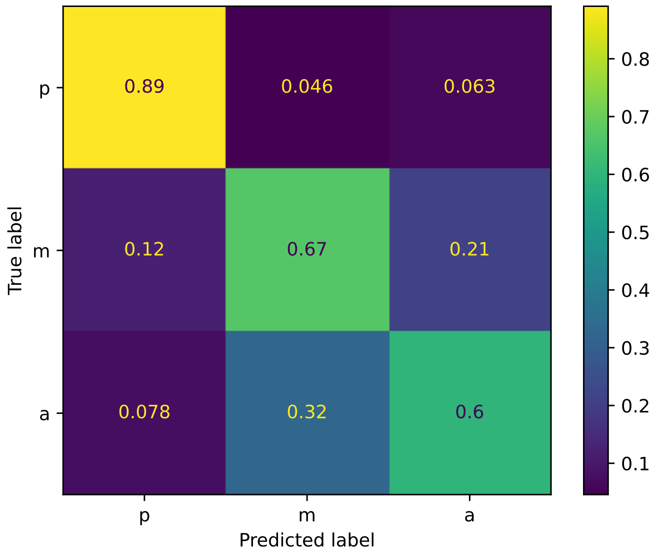


Figure 3.8: Confusion matrix normalized by row, i.e. the sample count in each cell is divided by the number of samples with this ground-truth label. The diagonal elements are equivalent to **recall** per label: what fraction of all samples annotated with this label are classified correctly)

incorrectly classified as p. The remaining 32% is due to the difficult challenge of distinguishing between a and m. A similar analysis can be performed for the row with label m.

Reference Confusion Matrices

With the normalization described previously, a perfect classifier would result in a confusion matrix equal to the identity matrix (see the right side of Figure 3.9). The probability of hitting the right labels (diagonal) is always 1, and for wrong labels (off-diagonal) it is 0. This is convenient for interpretation.

However, there is still one issue to consider when using this matrix. It is still a harder challenge for any model to "find" correctly the labels that are less frequent. This is a real challenge though, it is not a consequence of the metric used. But it is important to consider this fact in order to evaluate the performance of the classifier fairly.

To illustrate this issue, let us consider a blind model which selects labels randomly, but with probabilities that match the actual balance of the data, so that in average it is not biased towards any particular label. To avoid systematic overestimation or underestimation of any label, this model should be calibrated with the same balance of the training set:

$$\begin{aligned} P_{blind}(pred = p) &= 0.61 \\ P_{blind}(pred = m) &= 0.30 \\ P_{blind}(pred = a) &= 0.09. \end{aligned}$$

Since this random model is "blind", it does not look into the audio features, so the probabilities of predicting any given label are completely independent of the actual label of the sample. Mathematically speaking, this means that:

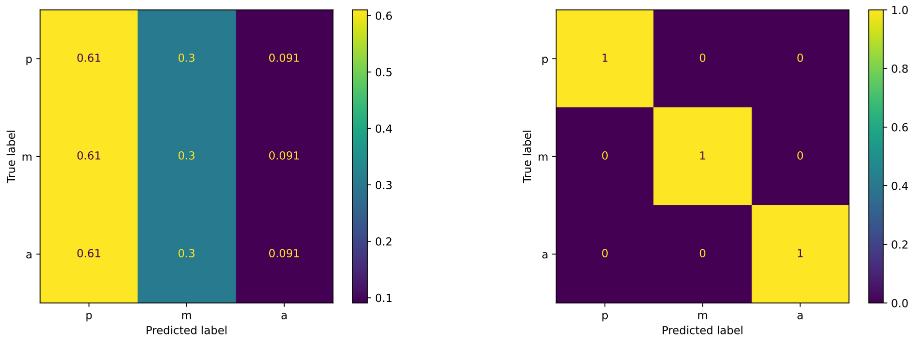


Figure 3.9: Reference confusion matrices. **Left:** confusion (normalized by row) for a blind model that randomly predicts the label for each sample, with probability according to the label balance. i.e., the lower boundary to compare any classifier model's confusion. **Right:** confusion for a perfect ideal model, that catches all labels (diagonal elements are 1) and never makes mistakes (off-diagonal elements are 0).

$$P_{blind}(pred = A) = P_{blind}(pred = A|ref = ...)$$

which results in the confusion matrix shown on the left side of Figure 3.9. Since this is the performance achieved by a random model, the diagonal values are the lowest boundaries that any machine learning model should aspire to.

Looking back at the matrix from Figure 3.8, and considering this random/blind model matrix as a lower-limit reference, the performance of each label can be better contextualized. So, for example, the label p has the highest performance of 89%. Remembering the probabilistic interpretation, it means that if we look at 10 samples with that label, 9/10 of them would be correctly predicted using the model, but using the random model we could have predicted 6/10.

Reasoning in the same way for the label a, only 1/10 would have been guessed using the random choice, but the machine learning model increases this chance to 6/10. In this case there are 6X more chances to get it right using the model, which shows that it is actually working pretty well to detect that label. And that is the reason why this reference matrix is useful.

Relation to Precision and Recall

Precision and *Recall* metrics are ubiquitous in classification problems, and they will also be explicitly included in some sections of this work. So it is worth mentioning their relation to the confusion matrix described previously.

In the first place, *Recall* is defined as the number of true positives (correctly classified samples) divided by the total number of elements that belong to that class. This is exactly equivalent to the elements on the diagonal of the confusion matrix, which are normalized by row (total number of samples for that *True label*). It can also be deduced by looking at Equation 3.5 and setting $A = B$ for the predicted and reference labels:

$$Recall(A) = P(pred = A|ref = A) = \frac{count(pred = A, ref = A)}{count(ref = A)}. \tag{3.6}$$

Secondly, *Precision* is defined with the same numerator (number of true positives for the label) but divided by the total number of elements predicted with this label. So, the equation now becomes:

$$Precision(A) = P(ref = A|pred = A) = \frac{count(pred = A, ref = A)}{count(pred = A)}. \tag{3.7}$$

This, in turn, is equivalent to normalizing the confusion matrix by columns instead of rows and extracting the diagonal values. But this normalization of the confusion matrix is not going to be used in this work. Instead, only the precision values per label will be included when relevant.

In summary, the *Precision* (per label) is the probability that the predicted label is actually correct, when looking at a sample that was classified with that label by the model. And *Recall* (per label) is the probability that a sample with the given reference label will be correctly predicted by the model.

F1 score

Finally, another metric that can be used to combine both aforementioned values for a given label is the F1 score. This is the harmonic mean of *Precision* and *Recall*, balancing both metrics to give a single performance value:

$$F_1 = 2 \times ((Precision \times Recall)/(Precision + Recall)). \tag{3.8}$$

The F1 score is also widely used in classification problems, especially when dealing with unbalanced data sets. It is a good substitute for the *Accuracy* value, which will not be used in this work, as it can be misleading (e.g., the model might achieve high accuracy by simply predicting p most of the time).

By combining both metrics, the F1 score provides a comprehensive evaluation of a classifier's performance, reflecting both its ability to correctly identify positive instances and its ability to avoid false positives. However, it does not have a probabilistic interpretation, which makes it conceptually less clear, and that is the reason to prefer the previous metrics individually, when summarization is not required.

3.3 Evaluation metrics over different human annotators

3.3.1 Redundant human annotations

In order to verify the designed annotation protocol (explained in Section 3.1) and also to better understand the selected evaluation metrics, some audio segments were selected from the available recorded lessons, and redundantly labeled by 3 different human annotators over the same time intervals, following the same annotation protocol.

This process served as a warm-up before annotating all available data, and was particularly useful to refine the procedure, unify criteria, and detect ambiguities or possible misinterpretations in the instructions. The hypothesis was that after some iterations comparing and improving the instructions, more uniform results should be obtained in the subsequent stage of annotating the entire data set.

After this redundant labeling process was completed, these annotations were used to test the evaluation metrics on them. The metrics are designed to compare the output *predictions* against a ground truth *reference*. But in this case, there are 3 possible *references*, since all human annotators are considered equally trained for the task. In order to measure the errors ensuring that there are no asymmetries or biases towards one particular annotator, all metrics were calculated over all the pair permutations of annotators, reversing which one is considered reference. For each segment annotated by the 3 annotators, namely `ann_1`, `ann_2` and `ann_3`, the reference and prediction labels are created by concatenating them as follows:

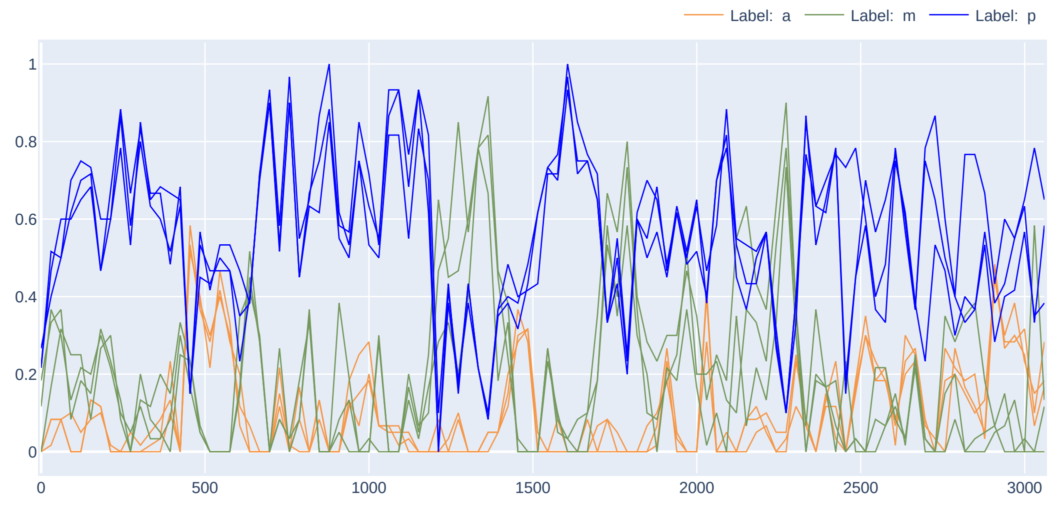


Figure 3.10: Comparison of the temporal density calculated over manual labels *a,m,p*, annotated by 3 different annotators. In this visualization, all the annotated audios (5 segments of 10 minutes each, 3000 seconds approx.) were concatenated.

```
reference = [ann_1, ann_1, ann_2, ann_2, ann_3, ann_3]
prediction = [ann_2, ann_3, ann_1, ann_3, ann_1, ann_2]
```

In this way, all pairs of labels and their reverse (swapping prediction and reference) are compared when calculating the metrics. This redundantly labeled subset of the data consisted of five segments of 10 minutes extracted from 5 different lessons, for a total of **50 minutes of activity**, particularly selected to include student participation, in order to test diverse situations (otherwise, label p is usually dominant).

3.3.2 Human performance metrics

As mentioned in the previous sections, the most relevant metrics (correlation and MAE) are calculated over a temporal density function (per label) that is obtained using Equation 3.1, which is basically an indicator of the fraction of time that each label (a,m,p) appear on a moving window of a given size (30 seconds in this case).

The resulting density function for all 3 different human annotators and labels is shown in Figure 3.10. The overall differences between human annotators can be observed in this figure, and some relevant parts will be analyzed deeper.

Note that the sum of the densities of the three possible labels (a,m,p) does not necessarily add up to 1, because there are frequent short intervals that are not labeled by the annotators (i.e: silences, non-identified sounds, or sounds that do not belong to any of the labels *a,m,p* evaluated here).

Once these densities are obtained, the correlation (**corr**) and error (**MAE**) can be calculated on top of them (see equations 3.3 and 3.2). These metrics are calculated individually for each of the five segments-of 10 minutes each- mentioned before, and also for all of them concatenated -as a single 50-minute segment, leading to a kind of average value-, as shown in Table 3.2.

It is interesting to note that in many cases there is an apparent discrepancy between both metrics, meaning that a lower MAE (error) does not necessarily mean a higher correlation.

Segment	**MAE**			**corr**		
	a	**m**	**p**	**a**	**m**	**p**
segment 1 (eng)	0.0264	0.0399	0.0341	0.9731	0.8919	0.9654
segment 2 (eng)	0.0204	0.0271	0.0256	0.8838	0.9641	0.9738
segment 3 (PC)	0.0265	0.0399	0.0261	0.9200	0.9814	0.9914
segment 4 (PC)	0.0371	0.0531	0.0287	0.8078	0.9266	0.9739
segment 5 (PC)	0.0414	0.0804	0.0614	0.9061	0.1805	0.7325
all segments	0.0316	0.0462	0.0344	0.9344	0.9288	0.9526

Table 3.2: MAE (absolute error in Eq. 3.2) and correlation (Pearson's coefficient in Eq. 3.3) of density functions for all labels, between all pairs of human annotators. In the first 5 rows, they are calculated over individual segments (2 in english -*eng*- and 3 in computational thinking -*PC*- lessons), and in the last row, all segments are concatenated.

As a first example, the MAE value for the *m* label in *segment 4* seems quite high, but the correlation for that same label is not particularly low compared to other entries (e.g., label *a* in *segment 4* has a lower MAE and lower correlation as well, and the same happens with label *m* in *segment 1* and label *a* in *segment 2*).

This example with high error but also relatively high correlation is shown in Figure 3.11, and the reason for this phenomenon becomes apparent from the top image: the density for *annotator 3* (in green) has a constant overestimation bias with respect to the other 2 annotators. This kind of shift affects the MAE (i.e., the mean distance between curves; see Equation 3.2) but not the correlation, which only accounts for deviations from the mean value (see Equation 3.3).

In fact, comparing both plots at the bottom of Figure 3.11, it is shown that the correlation coefficient (see the title of each plot) is similar in both cases, even though the points on the right side (i.e: comparing against *annotator 3*) have a constant vertical deviation from the identity line $x = y$ (not shown but can be inferred as the exact diagonal). As mentioned before in Section 3.2, this is a desired feature of the correlation metric, aimed at capturing how informative the curve is to find peaks of activity of the different labels during the lesson.

Another example worth examining in detail from Table 3.2, is in the *segment 5* (last row), where the correlation value for label *m* is surprisingly low. Although the MAE value is the highest for this entry as well, the correlation is disproportionately low considering the full range [0-1] that both metrics have. This example is shown in Figure 3.12.

In the top image of the figure, it is shown that the curves have important differences but in some segments they are actually not so far from each other, since all the signals have a low average value. Looking more carefully, however, it becomes clear that the 3 largest peaks from the green curve (annotator 3) are out of phase with the peaks in the other annotators. Also, there are other non-matching peaks between the other annotators. This causes many points in the x-y axes which push the correlation towards zero in the left-bottom plot (remember that a completely horizontal or a vertical line means zero correlation). Even worse, in the right-bottom plot there is a negative correlation, caused by the fact that the peaks are out of phase and hence an increase in one signal seems to produce a decrease in the other one.

Looking more carefully at this last *segment 5* in Table 3.2, it is not clear what the exact problem is with this particular label *m*, since there is no other label with a comparable low correlation, to establish the cause of confusion. Although the main candidate is the label *p* because it also has the lowest correlation of all examples, it does not seem to fully explain the cause.

To answer those questions, the confusion plots shown in Figure 3.13 are quite useful. By observing the right plot (corresponding to the mentioned example) it is clear that the bottom row is much worse than the compared segment to the left. To interpret these results, recall that each

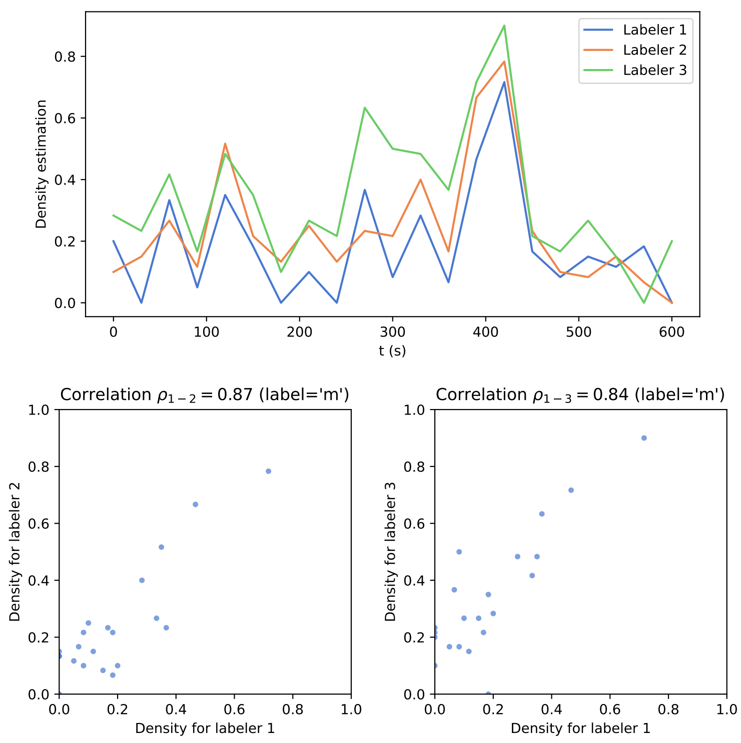


Figure 3.11: Comparison of densities for label *m* for the 3 human annotators. This example corresponds *segment 4* in Table 3.2, which has a higher correlation than many entries with lower errors (MAE). **Top:** density estimation for the 3 annotators as a function of time. **Bottom:** correlation of density values compared between pairs of annotators (comparison between annotators 2 and 3 is excluded).

cell represents the count of samples in it, divided by the total number of samples with the same reference label (*True label*). Hence, each row should add up to 1, except for the fact that there are non-labeled segments of the audio.

Taking into account the above recap, the results mean that from all samples labeled as m only 33% match between pairs of annotators, and then 21% and 16% predicted as p or a, respectively. The remaining 30% is not labeled by any other annotator.

Trying to better understand the cause for these differences in annotations requires listening to the particular audio while looking at the corresponding annotations. Indeed, this case is unique because it is a lesson where there are 3 remote teachers connected to the same lesson, and it is a special day (one of the first days back in school after the COVID-19 pandemic), so the ambient is very noisy and the amount of overlapping even between teachers is exceptionally high, leading to an ambiguity about when to label p or m.

The main purpose of reviewing these examples in this section, is to calibrate the expectations on what can be achieved with a machine learning system evaluated on manually labeled data, considering the challenges that are shown in some of these examples. Also, it is very useful to catch some relevant subtleties about the metrics used, by understanding some cases that may seem discrepant at first sight when comparing the obtained metric values. Both the absolute error (MAE) and the correlation metrics may be useful to understand different aspects of the signals

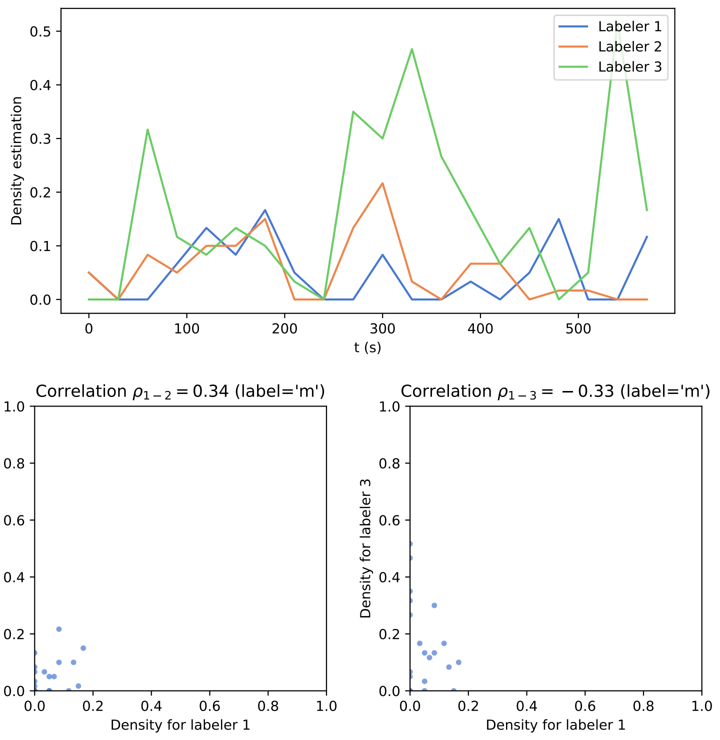


Figure 3.12: Comparison of densities for label *m* for the 3 human annotators. This example corresponds *segment 5* in Table 3.2, which has a very low correlation value for this label. In fact this is the worse example in terms of discrepancy between annotators. **Top:** density estimation for the 3 annotators as a function of time. **Bottom:** correlation of density values compared between pairs of annotators (comparison between annotators 2 and 3 is excluded).

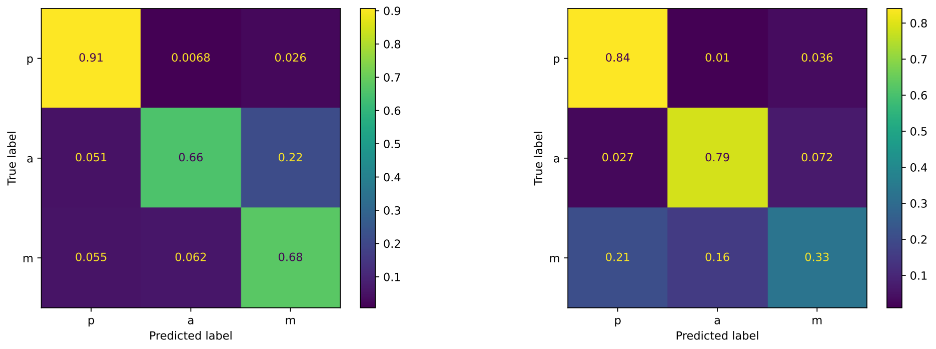


Figure 3.13: Confusion matrices between human annotators for *segment 4* (**left**) and *segment 5* (**right**) from Table 3.2. The comparison is done considering all pairs of annotators (swapping reference/prediction values as described at the beginning of Section 3.3).

being evaluated, and examining them on a per-label basis provides much more information than aggregating them in one single value.

In this chapter, we presented the available data and the manual annotation protocol. Additionally, we defined the metrics to be used and tested them on redundantly annotated data, providing a reference for the best performance achievable by a system trained and evaluated on it.

These fundamental analyses and definitions serve as the building blocks for the subsequent chapters. In the upcoming sections, we will implement, evaluate, and compare various models against each other.

Chapter 4

Implementation

In this chapter, we delve into the practical implementation of all our Classroom Activity Detection (CAD) variants. Our focus lies on two different approaches to CAD: supervised classification models based on XGBoost and LSTM, both utilizing the same set of audio features, and an unsupervised diarization pipeline, specifically adapted for this task.

We individually evaluate the models, comparing different parameter settings to identify the best-performing candidate within each model family. The selected best candidates will be further compared in the subsequent chapter, where we analyze and contrast the results obtained by each approach.

At this point, we assume the reader's familiarity with the methodological foundations presented earlier in Chapter 2, and now we delve into additional practical details and reviews to reinforce the implementation process.

4.1 Implementation of Audio Classifiers

The data and the audio features used for both implemented classifiers (LSTM and XGBoost) are exactly the same, so that their performance can be fairly compared.

As detailed in Section 3.1.3, the data are first divided into many groups and splits, from which we get a total of 7.4 hs of annotated audio for **training** purposes, and 7.3 hs for **testing**. The most important rule here is that the test data cannot be used in any way by the models during the training process.

In order to avoid *overfitting* or *fitting to the noise*, these models also need a **validation set**. So, they can update their parameters using some other data, and then validate if the performance actually improved on this set, which was left out during the parameter updating process.

This is done periodically during training and is intended to make sure that the model actually learns generalizable patterns that extend to unseen data, instead of memorizing the training set. In practice, this can be verified by comparing training versus validation losses, as can be seen in figure 4.1. These are two real examples while training the LSTM models, which clearly show that if the training process was not stopped, the training loss would continue to improve but the validation loss actually shows that the performance of the model is getting worse.

This technique is called *Early Stopping*, and was used in both XGBoost and LSTM classifiers. The size of the validation set in both classifiers was 20% of the training data, but it is not reserved beforehand, it is randomly chosen for each training job (although the random seed is fixed for

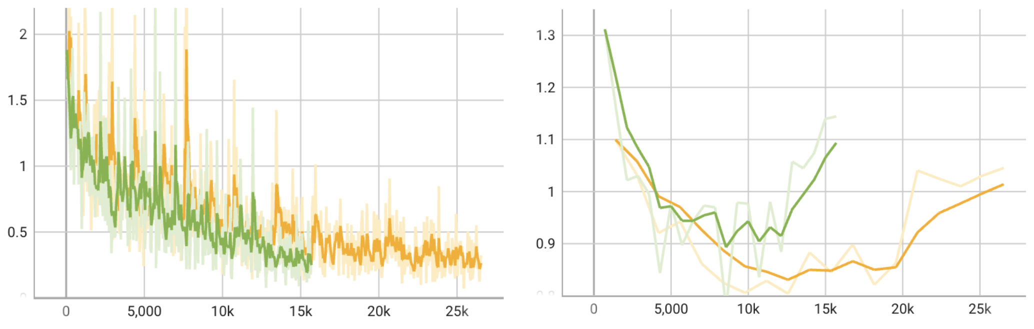


Figure 4.1: Loss function values for two different models during the training process, comparing evaluation over Train (left) vs. Validation (right) sets. The horizontal axis is the training step. Each color represents a different model being trained. The stop criteria is a certain number of steps without improvements in the validation loss.

reproducibility). Since the convention is to use the names *Training / Validation / Test* for these sets (where *Train* is only the data that are used to update the actual parameters), we may also refer to the set of union *Training + Validation* as the **Development set** (containing all 7.4 hs data, from which 80% is *Training* and 20% is *Validation*).

4.1.1 Audio Features

As mentioned above, both classifier families use the same audio features. These are extracted over sliding windows (frames) of 30 ms length and a step (hop) size of 15 ms, hence overlapping at 50%.

The entire feature vector is composed of the following components (see Section 2.1) for a description of each of them:

- **MFCC**: 12 coefficients, from a Mel-filterbank of 60 bands, ranging from 60 Hz to 8 kHz.
- **Spectral Flatness**: scalar value.
- **Spectral Centroid**: scalar value.
- **Spectral Bandwidth**: scalar value.
- **Spectral Contrast**: 7 values/bands.
- **Signal Power (dB)**: scalar value.

In total, the feature vector has **23 components** for each audio frame.

MFCC values are calculated using the *speechbrain* library [40], since it has better interoperability with the rest of the code which uses *torchaudio* [46] for the most part. The rest of the features are extracted using *librosa* [48], on top of the same STFT that was already extracted with *speechbrain*.

The functionality is encapsulated in a specialized *Featurizer* object, which is used by both types of classifiers. This object also implements a *cache* to temporarily save the features of the input audio, speeding up computations by a large margin when multiple models are being evaluated over the same test audios.

4.1.2 LSTM based classifier

The LSTM classifier for this work is implemented using *PyTorch* [31].

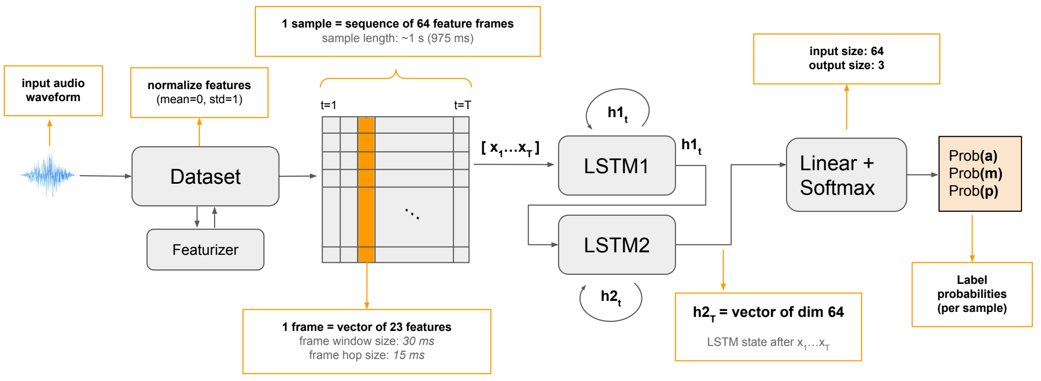


Figure 4.2: Diagram of the LSTM-based pipeline for Classroom Activity Detection. Each input audio is converted into a dataset of consecutive samples, each one ($\simeq 1\,\mathrm{s}$) being a sequence of feature vectors representing an audio segment. The output is a vector with the label probabilities for each sample.

Any recurrent neural network requires that its inputs be a sequence of vectors. In this case, the vectors are the audio features extracted over the sliding windows (frames) described in the previous section.

Unless otherwise specified, each sample provided to the LSTM consists of a sequence of 64 frames. Given that the hop between frames is 15 ms (see previous section), this means that each sample covers a span of approximately 1 second ($\simeq 15ms \times 64$), which seems a reasonable amount of time to distinguish class moments, according to the experience collected while analyzing and labeling the data.

Figure 4.2 shows the whole pipeline for this classification system, receiving an audio waveform signal, and predicting class probabilities for each sample, which represents a small segment of the input audio. The label with the highest class probability is set as the label for the segment.

The figure shows the default configuration and parameters, where there are two layers of LSTM blocks stacked. This means that the state of the first layer serves as the input sequence to the second layer (as discussed in Section 2.2). Only the final state $h2_T$ from the second layer (dimension `hidden_dim=64` unless specified otherwise) is used to feed a linear layer (single fully connected). The latter reduces the dimensionality to the number of labels in this classification problem, which is set to 3 (a, p, m) in general. This layer is actually an affine transformation of the form:

$$\mathbf{y} = A \cdot h2_T + b$$

where $A \in M^{3\times 64}, b \in \mathbb{R}^3$ (assuming 3 labels and `hidden_dim=64`) are learnable parameters. Other versions were also tested to output the finer-grained labels and then postprocess them for remapping (see Section 3.1.2), but the results favored the current approach.

The final layer is just a softmax transformation:

$$\mathrm{softmax}(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{3} e^{y_j}}$$

which maps the 3 raw outputs from the linear layer into probability-like values in range $[0-1]$.

Feature normalization

In the first stage of the pipeline, the audio signal is wrapped in a *data set* object. Different dataset objects need to be created for training, validation, and testing purposes. All share the same audio features, and so they are extracted using an instance of the same *Featurizer* object introduced in the previous section.

A standard practice for neural networks is to normalize the input features, so that they have zero mean value and unitary standard deviation. For the system to work properly, the same transformation should be applied to all inputs, either training data or any other audio. However, the model should not have access to the test data during the training process, so these parameters are calculated using only the training data, assuming that it represents the data distribution.

When the *Training Dataset* is created, the *mean ($\bar{x}_{train}$)* and *standard deviation (std or σ_{train})* of each feature is calculated as follows:

$$\bar{x}_{\text{train}} = \frac{1}{N_{\text{train}}} \sum_{i \in \text{train}} \tilde{x}_i$$

$$\sigma_{\text{train}} = \sqrt{\frac{1}{N_{\text{train}}} \sum_{i \in \text{train}} (\tilde{x}_i - \bar{x}_{\text{train}})^2},$$

where $\tilde{x}_i$ is the non-normalized feature vector for the sample i, N_{train} is the number of samples in the training set, and the abuse of notation $i \in$ train is adopted to indicate only the training sample indices.

These two vectors are stored in the same folder as the model weights, since they are needed to normalize the features from any input audio (train, test, or any other inference) as follows:

$$x_i = \frac{\tilde{x}_i - \bar{x}_{\text{train}}}{\sigma_{\text{train}}} \quad \forall i \in \text{train,val,test}.$$

The *Dataset* object encapsulates all this functionality, allowing normalization of the training data features and retrieval of the mean and std values used, and also to provide external mean and std values (i.e., the saved values from the training data) to normalize the test data or any audio features where the model is used for prediction.

Sample overlapping

Another thing that is slightly different in the train vs. validation / test data is the overlap between samples. The *Training Dataset* is configured with a separation between samples that allows overlapping in order to provide the model with time-shifted versions of samples, which is not exactly data augmentation, but it serves the same purpose of increasing the generalization power. Although this technique increases the correlation between samples weakening the *iid* assumption [21] of neural network training procedures, it has shown better results in practice (note that the samples in each training batch are randomized). In particular, this parameter is set to an overlap of 75% between consecutive samples during training.

However, this overlap is not desired during evaluation stages (i.e: validation or test), since it would slow down the process and even cast a pall of doubts on the results unnecessarily. Therefore, for these datasets, the evaluated samples do not overlap.

Note that the validation data is a random subset of training data, representing 20% of the available audio time. But since this subset does not allow overlapping between samples, the number of samples is less than 20% of training samples -the proportion is defined in audio span duration-.

Training procedure

The first step of the process is to load all available audio segments for training. This is configurable to some degree to allow only a subset of the data to be used, but the test splits are not available for loading at this point. The *Train Dataset* is created by concatenating all these segments and their labels, extracting their features and then defining the sequence of frames that belong to each sample.

The training loop is implemented using the *PyTorch Lightning* library [30], and the total training time is not deterministic, but it is rarely longer than 10 minutes on a single GPU (running on a *nvidia 3080*, although the model would fit on a much smaller device) when using all the data.

The most important components and parameters used in the implemented training loop are the following:

- **Adam optimizer** [25] from *PyTorch* [31] with an initial learning rate of 1×10^{-3} that goes down to 1×10^{-5} using the *ReduceLROnPlateau* strategy from *PyTorch* [31], which decreases by a factor of 0.1 when the validation loss does not improve for 5 epochs.
- **Loss function:** is a weighted *NLLLoss* from *PyTorch* [31] on top of the log-softmax values from the classifier. The weights configuration is described in the next section.
- **Early stopping** criteria, which stops the training when the validation loss does not improve for 10 epochs. This implies a minimum number of epochs of 10 and, in practice, it means that the loop stops well below the established limit of 200 max. epochs.
- **Checkpointing:** only the model weights that correspond to the minimum validation loss are saved and used for prediction afterwards.
- **Batch size:** 64 samples. Also tried 32, 128 and 256, but the results were similar or worse.

Loss function: dealing with unbalanced and unlabeled data

The *NLLLoss* function mentioned above allows the use of different weights for each label. This is very useful to tackle the problem of imbalanced data by using weights that counteract the different fractions of samples with each label. The weight for each label is then calculated as:

$$\text{weight}(\text{label}) = \frac{N_{\text{labeled}}}{\sum_{i \in \text{train}} I(\text{label}_i = \text{label})}$$

which is simply the inverse of the label fraction, compared to the whole labeled duration. It is worth noting that N_{labeled} only counts labeled samples in the training set.

For unlabeled samples, the loss function also has a parameter to ignore a given label ("-" in this case), such that it does not contribute to the gradient in the backpropagation parameter updating process. This also implies that the model will never predict the null label "-", which instead is set in a post-processing step when the signal power is below a given threshold.

Hyperparameter selection

Summing up the different parameters for the LSTM mentioned before, the default values chosen for each of them are the following:

- `hidden_dim`: 64
- `num_layers`: 2
- `frames_per_sample`: 64

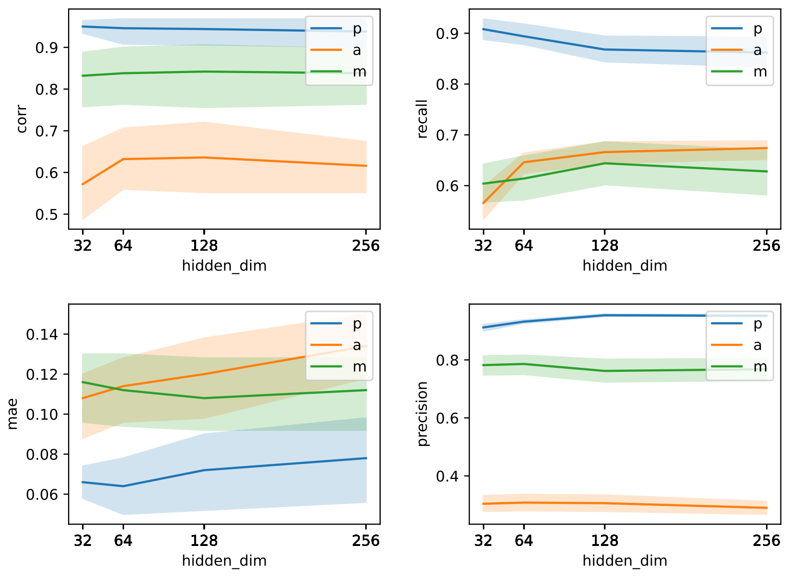


Figure 4.3: Evaluation of the LSTM classifier varying hidden layer dimension, separated by label. The colored spans around each curve represents the standard deviation of each metric, as measured on all 5 test groups.

Where **`hidden_dim`** is the size of the state vector in each layer of the LSTM, **`num_layers`** is the number of stacked LSTM blocks (in Figure 4.2 it was fixed in 2, but it could be easily changed using this parameter), and **`frames_per_sample`** is the sequence length or number of frames that are included in each sample (represented by the letter T in the same figure).

Other important aspects of the system like the features to use or the number of fully connected layers at the final classifier head could also be tuned, and in fact some variations were tested without showing better results.

To understand whether the capacity of the model was adequate for the amount of data available, the **`hidden_dim`** parameter was tested over a range of possible values. This requires training different variants of the model and evaluating each version on all 5 test groups (see Section 3.1.3), allowing one to estimate the mean value and standard deviation of different metrics.

Figure 4.3 shows this comparison, making it clear that a value of **`hidden_dim=64`** is the best candidate considering all factors. The main improvement at that point can be observed in the recall curve for the label a, meaning that more student audio segments are correctly detected. This is very desirable because these are not frequent labels, and so this improvement should pays off for the decrease in other metrics such as the recall for the label p.

The MAE (error) also increases for the label a at that point, suggesting that there are more false positives detected as well, but this should not be a serious issue according to the precision plot.

Another important parameter that was tested over a range of values is the sequence length or **`frames_per_sample`**. From a sequence of 32 frames up to 256, which corresponds to time spans of $\simeq 0.5\,\text{s}$ up to $\simeq 4\,\text{s}$.

From Figure 4.4 it can be deduced that a value of 64 frames (i.e., segments of $\simeq 1\,\text{s}$) is the one that shows the greatest benefit. There is some increase in the recall for label a at 128 frames, but

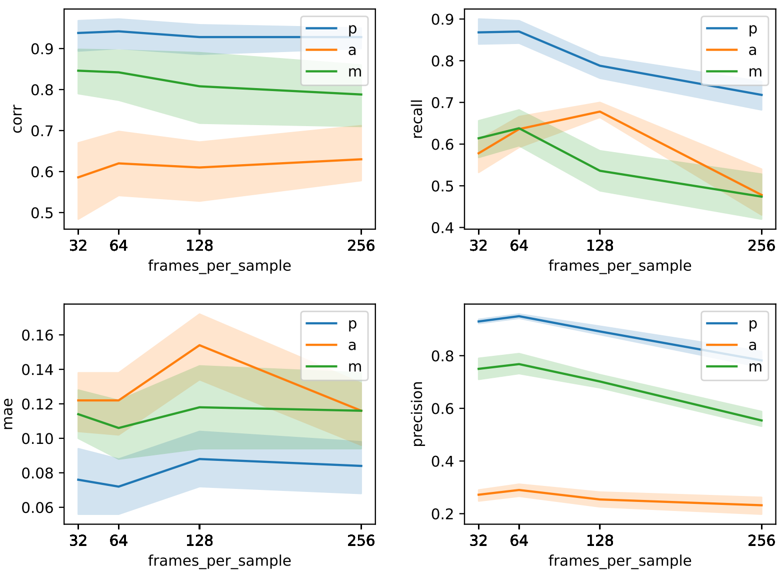


Figure 4.4: Evaluation of the LSTM classifier with different number of frames for each sample sequence, separated by label. The colored spans around each curve represents the standard deviation of each metric, as measured on all 5 test groups.

this comes with a drop in the precision and an important increase in the relative and absolute errors for this label, aside from a big cost in the recall of all other labels. Basically, with the exception of that metric, all other performance metrics show a peak at 64 frames and errors are minimum as well. This might be due to the fact that we only use one label for the whole sequence, and some annotated segments have around 1 second of duration. If this was the case, it could be solved by using a seq-to-seq model instead.

4.1.3 XGBoost based classifier

The audio features used for this classifier are the same vectors of 23 components used as input in the LSTM pipeline (see Section 4.1.1 above).

The main difference in the preprocessing of these features, is that now these vectors are not normalized in mean and variance because that is not needed for decision trees.

Also, since the decision trees do not have a state to process the inputs sequentially, the context for each frame window (duration of 30 ms) has to be added manually. This is implemented by using an extended vector of concatenated features for a certain number of previous and following frames, as can be seen in Figure 4.5. The number of frames to include is controlled using the `context_window` parameter, which causes the input to the XGBoost model to multiply accordingly.

For example, a value `context_window=10` means that 10 frames before and after the current one are concatenated together with it, resulting in an input of dimension $(10 + 10 + 1) \times 23 = 483$ for the model. This allows the classifier to consider an extended time span of approximately $21 \times 15\ ms \simeq 300$ ms.

This parameter turns out to be the one with the greatest impact on all classification metrics, as can be seen in Figure 4.6. Three models were trained with different values for this parameter,

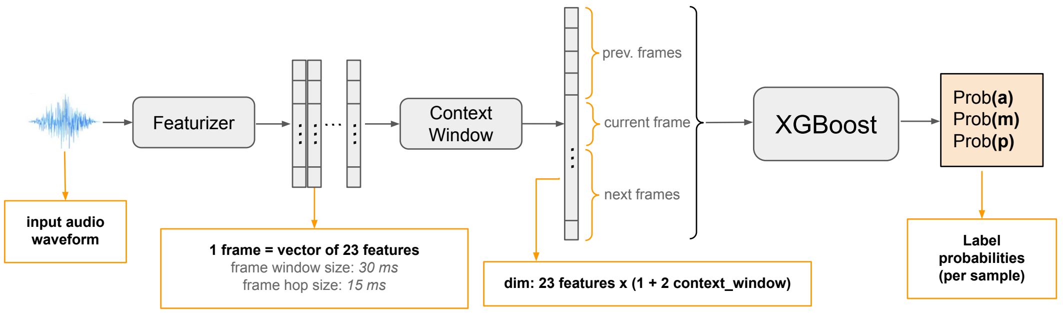


Figure 4.5: XGBoost classifier pipeline for Classroom Activity Detection. The featurizer generates a sequence of vectors using a sliding window with the size and hop indicated. To create a sample, previous and following frames are concatenated together with the current one, creating an input vector with higher dimension for the XGBoost model, which also covers a larger time span to perform the prediction. Finally, the label from the current frame is predicted in the XGBoost model.

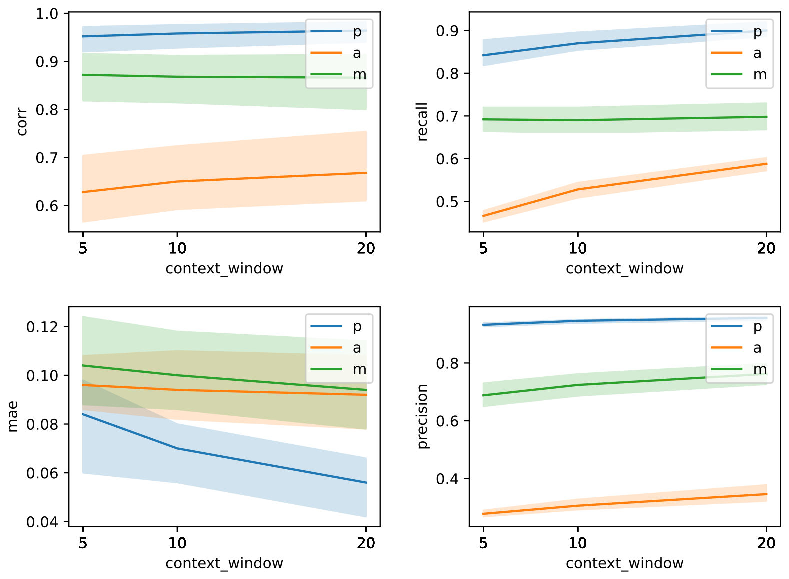


Figure 4.6: Evaluation of the XGBoost classifier with context windows size, separated by label.

and the largest context window shows the best results.

It is not clear whether or not using a larger context would continue improving the results, but training and testing those variants was not possible with the available hardware resources, since the concatenated feature vectors ends up with a very large dimensionality, consuming all the system memory and computing resources, to the point of breaking the training process.

Also, although it is clear that with a context window of 20 the results are better, the pipeline is also twice as fast when using a context of 10, making the overall solution a lot better with only a marginal performance drop in that step. The highest improvement slope is obtained by using `context_window=10` instead of `context_window=5`. The difference in execution times is not so significant in those cases.

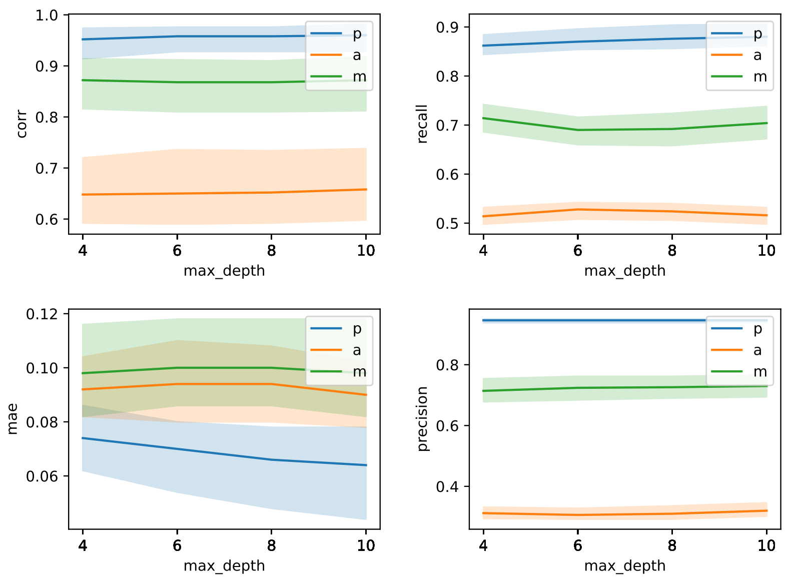


Figure 4.7: Evaluation of the XGBoost model with different maximum tree depths, separated by label.

Another parameter that also makes a difference in execution time but not so much in the classification metrics is the `max_depth` parameter, which controls for the complexity of the trees by limiting their allowed depth, which in practice also impacts in the maximum number of leaves the tree can have (i.e. it limits the T value in Section 2.3).

The impact of this parameter is shown in Figure 4.7, and the tendency is not monotone for most metrics and labels, although the point at `max_depth=10` seems to be the best. However, the impact of this parameter on execution time is very important, and in fact, inference is twice as fast for the model at `max_depth=6`, without a huge performance drop. Both of these models are included in the benchmark of execution times when comparing them to the LSTM system (see Table 5.2 in Section 5.2).

The regularization parameter referred to as γ in Section 2.3 had no impact whatsoever on the range of values tested, which started at $\gamma = 0$ and went up to $\gamma = 2 \times 10^{-3}$

Training XGBoost: Undersampling, Nulls Removal, Early Stopping

To deal with the problem of unbalanced data, and mainly with the issue of the over-representation of label p, the technique of *undersampling* is used, and the criteria are to randomly discard samples until reaching the same amount as the second majority class. In this case, the second majority class is the label m, which has approximately 30% of samples (see Section 3.1.3), so the original proportion of 60% for p is halved.

Another consideration is that only 80% of the audio is annotated. There are 20% samples without any label (i.e., null labels). In the LSTM this was achieved by ignoring them in the loss function calculation, but the XGBoost model does not have such functionality. Therefore, these samples are removed from the training samples after the context window operation. They are still used as context for contiguous segments with valid labels.

The maximum number of trees for this model is 3000 (“weak learners” in Section 2.3), which is a complexity that would easily overfit to the training data. This is avoided by using *early stopping* (described in the same section). A validation set is needed for this purpose, which is randomly selected as 20% of the total training samples. The loss function used is a multi-class logistic loss (aka cross-entropy loss) on top of softmax outputs with the prediction scores for each class.

4.2 Diarization based pipeline

The diarization pipeline used in this work is the state-of-the-art system available with the open source library *pyannote.audio*, being still under active development at the time of writing (version 2.1 is used) [33].

4.2.1 Voice, Segmentation and Overlap Detection

One of the key aspects of the pipeline used is the use of a novel overlap-aware segmentation model developed by Hervé Bredin (2021) [37], which performs Voice Activity Detection (VAD), Speaker Segmentation and Speech Overlap Detection all in one single model. This is achieved by using a permutation-invariant speaker classification neural network, which is trained to select between $K_{max} = 3$ possible speakers in short segments of 5 s, and the output provides information that can be aggregated and post-processed to estimate voice probability, speaker changes, and overlapping, all at once.

Figure 4.8 shows two example outputs from this segmentation model, each corresponding to a segment of 5 seconds of audio. The main assumption of this model is that it is very rare to find such segments with more than $K_{max} = 3$ different speakers. So the model is aimed to locally solve a classification problem with 3 possible labels (speakers), which may have various types of voices. The model is trained to distinguish between three potentially different voices in a segment of 5 seconds, regardless of their particular sounding.

Each speaker color in Figure 4.8 corresponds to one particular reference speaker, but only during the local segment that is predicted. Thanks to the permutation-invariant training, note how the orange speaker corresponds to the blue reference speaker in the left side segment and then in the right side it corresponds to the red speaker. Both segments correspond to different moments of conversation between the same speakers.

This trick actually consists of using a permutation-invariant loss function that does not penalize swapping the reference and predicted speaker numbers, as long as each of them corresponds to one particular speaker during the local 5 s segment [37]:

$$\mathcal{L}(y, \hat{y}) = \min_{perm(y)} \mathcal{L}_{BCE}(perm(y), \hat{y})$$

where $\hat{y}$ is the predicted speaker for a given set of segments and y is the ground-truth label. The *BCE: Binary Cross Entropy* loss function is actually calculated over a matrix of $K_{max} \times K_{max} = 3 \times 3$ between all pairs of labels, and then the correspondence between labels with the minimum total cost is selected, solving an optimal assignment problem (the Hungarian algorithm is used for this purpose [37]). For example, assume that the reference labels are r_1, r_2, r_3 and the predicted labels are p_1, p_2, p_3. The correspondence between r_i and p_j is not known a priori, so all combinations are tried (e.g. $r_1 \longleftrightarrow p_1, r_2 \longleftrightarrow p_2, r_3 \longleftrightarrow p_3$, then also $r_2 \longleftrightarrow p_1, r_1 \longleftrightarrow p_2, r_3 \longleftrightarrow p_3$, and so on with all possible combinations), and the one with the minimum total loss is chosen.

After having the output with the speaker probabilities as shown in Figure 4.8, the voice activity can be detected by just using a threshold on any speaker's probability, and the speaker changes and overlapping can also be decided using similar heuristics (see [37] for more details).

Since the label m in this work actually means "multiple speakers", which refers to moments of group work or students answering all at the same time (see Section 3.1 for exact definition), one of

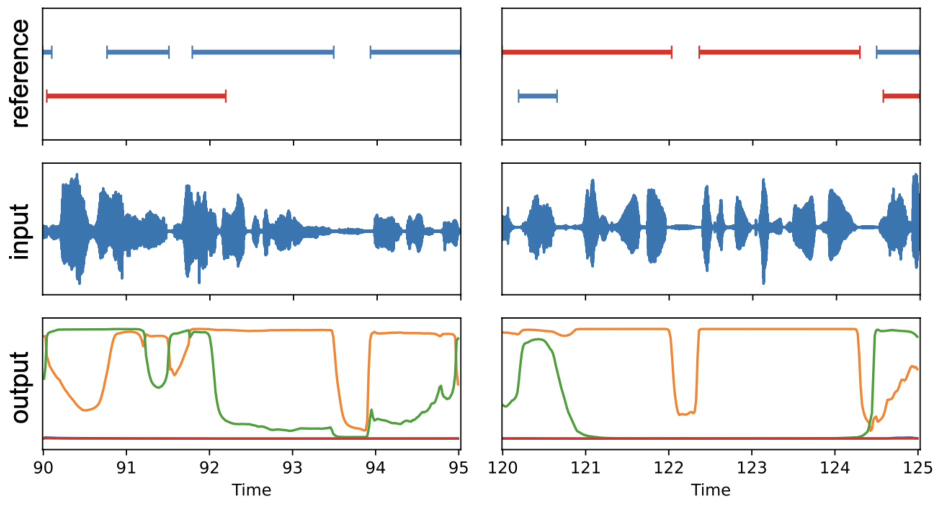


Figure 4.8: Outputs from the permutation-invariant speaker classification model, showing two example segments of 5 s at left and right -which actually correspond to different moments of a conversation between the same speakers-. **Top:** Reference ground-truth labels for two different speakers. **Center:** Input audio segments waveform. **Bottom:** Predicted probabilities for 3 speakers, each one has a different color. One of the speakers (red) has zero probability the whole time, since there are only 2 speakers (potentially simultaneously) in these segments. The orange/green probabilities correspond to blue/red speakers on the left side respectively, but they swap to the red/blue speakers on the right, thanks to permutation-invariant training. Source: [37]

the ideas was that maybe this label could be detected when the output of the segmentation model -discussed previously- indicates that there are overlapping speakers. However, the attempt to do it in practice was short-lived, because the label m is very often used when there are indistinguishable voices in the background, typical when students are working in group activities. This situation was not detected by the model, which is designed to work with foreground voices that are more clearly defined.

4.2.2 Speaker Embeddings, Clustering and Resegmentation

The output from the speaker segmentation stage is a series of well-defined time segments that should contain one particular intervention from one single speaker. It may overlap with other segments from other speakers, but the particular speaker should be present during the whole segment.

The output segments are provided in 3 separate channels, one for each speaker. One particular speaker corresponds to one particular channel only during one single intervention, subsequent interventions for the same speaker may show up as a segment in a different channel due to the permutation-invariant nature of the segmentation model (as was previously explained).

These well-defined segments are fed into a speaker embedding model, which extracts vectors that should characterize one particular speaker (see Section 2.4 for details on speaker embeddings). The model used for this pipeline is the *ECAPA-TDNN: Emphasized Channel Attention, Propagation and Aggregation over a Time-Delay Neural Network* [35], which is also a state-of-the-art model that was successfully used in speaker recognition and diarization challenges [44], and is available on the *SpeechBrain* open-source library [40] -used also in many of the building blocks of this work.

The result of this step is that each segment that corresponds to one intervention from one particular speaker, is associated to one speaker embedding.

The next step is to run the clustering algorithm over these embeddings, in order to create

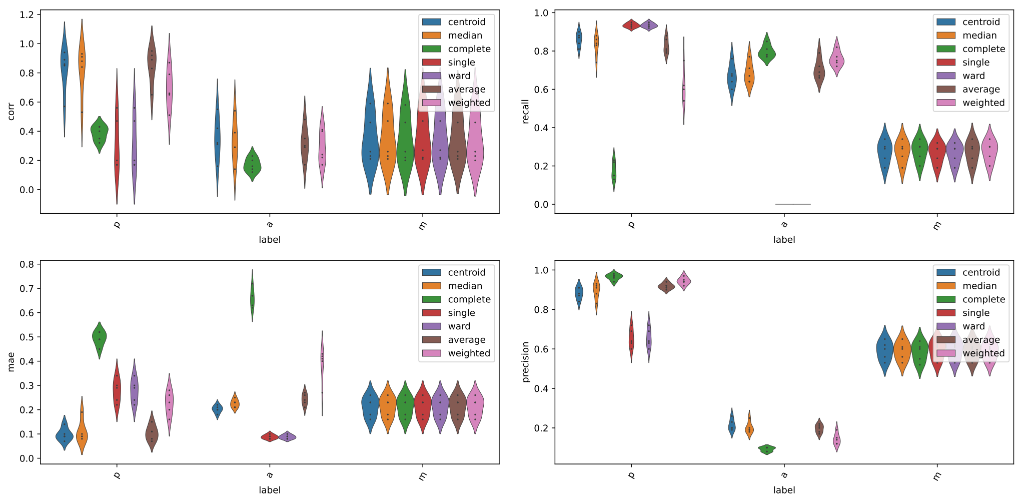


Figure 4.9: Evaluation of the diarization pipeline with different clustering linkage criterions, separated by label.

groups of similar embeddings and associate one speaker to each group.

The clustering algorithm used in this particular pipeline is Agglomerative Clustering, which was described in Section 2.4. This is the method that produced the best results in this particular pipeline, despite it does not leverage the fact that embeddings near in time are more likely to belong to the same speaker. The Spectral Clustering algorithm described in Section 2.4 (which is very popular in diarization pipelines) is also available to use with the library, but the results were not comparably as good.

In order to select the appropriate level of clustering between the bottom and top extremes of the dendogram, a threshold on the similarity between clusters is used in this case. When the similarity between the next clusters to merge is below this threshold, the process is stopped and the resulting clusters up to that point are used. This threshold needs to be tuned for the specific embeddings and similarity measure used.

As mentioned in Section 2.4, there are many strategies to measure similarity between clusters to merge, in an operation that is called *linkage*. The linkage criterion determines the distance between clusters as a function of the pairwise distances between their observations.

For example, the *single* criterion means that the distance between two clusters is determined by the minimum distance between pairs of observations in those clusters. On the other hand, the *complete* criterion uses the maximum distance between pairs. The *median* and *average* criterions use the median and average distances between pairs, respectively. And the *centroid* criterion computes the cluster centroids as a first step, and then measures the distance between to determine their similarity [23].

In this work, these linkage criterions were put to test and compared, as shown in Figure 4.9. The performance of the entire diarization pipeline is measured in the 5 test groups available for each linkage method. It can be observed that the *centroid*, *median* and *average* criterions are the ones with better and more consistent performances across metrics and labels. From those, the *centroid* method was selected because it is the default method used in this pipeline and there is no other method with a notorious advantage to justify a change.

Another important parameter to define in this pipeline is the minimum size that a cluster can have, to be considered as a speaker. Clusters with fewer embeddings than the minimum are

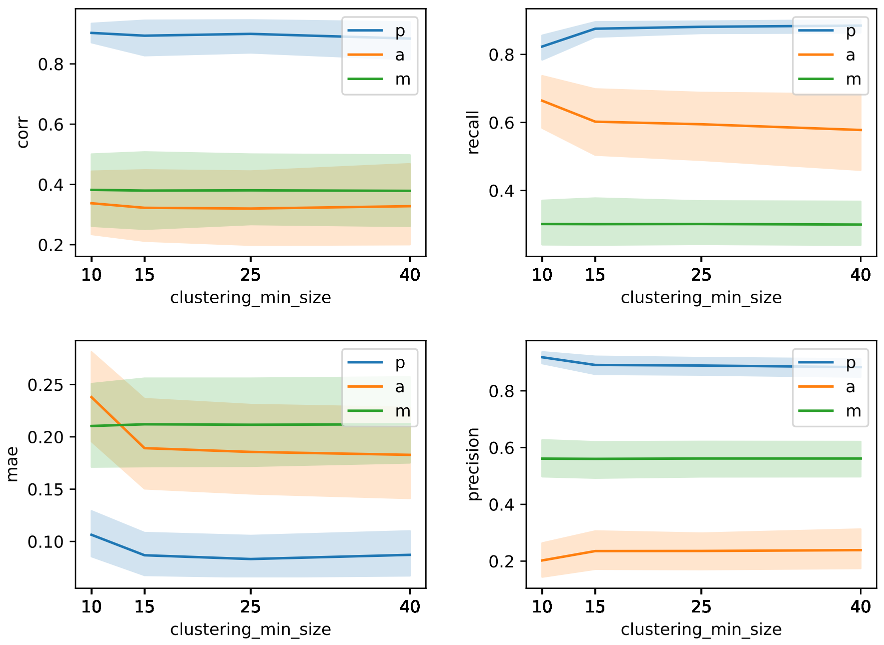


Figure 4.10: Evaluation of the diarization pipeline with different cluster minimum size, separated by label.

simply discarded because the segments belonging to that speaker would be too short and considered insignificant.

The minimum cluster size was also evaluated at many values, and the results can be analyzed in Figure 4.10. There is a clear drop in the MAE value (better, since this is an error metric) for labels a and p when using `clustering_min_size=15`. The recall drops for label a but the precision increases, and the opposite occurs for label p, which indicates that there are fewer false positives in the detection of student participation. This is the value used for the parameter.

The reason why this happens will become clear in the next section, where the heuristic to convert the diarization outputs into labels for Classroom Activity Detection is described.

4.2.3 Using Diarization for Classroom Activity Detection

The output of the diarization pipeline is actually a list of segments with their respective speaker numbers.

Diarization does not solve the classification problem (or CAD, for this matter) directly, since it does not predict if any given speaker is the teacher or some of the students. But assuming that the teacher's voice is always the most frequent speaker (which is the case in all the lesson recordings that were analyzed for this work, as shown in Section 3.1.3), the speaker detected with the largest time across the lesson can be assigned to it.

Note that this heuristic fails badly if the diarization predictions are not above a certain quality threshold. From the results seen so far, where the correlation values for the teacher's predicted density are above 0.8, the conclusion is that the rule works successfully for that label.

The heuristics for assigning labels m and a are not as clearly effective. The rule is that any speaker that is not the most frequent speaker but gets assigned its own cluster, is considered a student, i.e. it is assigned label a. This assumes that single students participating in class have a clear voice and those interventions are large enough as to be distinguished as a speaker.

The remaining segments, which are not assigned to any particular speaker, are classified with label m, since there is no other possible label to use at this point (as was also the case for the classifiers). Removing or including in this label the segments that are classified as overlapping speech did not have any visible effect on the performance of the model.

Observing the results so far, the heuristic to detect the teacher's voice seems to be appropriate, but the conclusion is not so clear for the other labels a and m.

In this chapter, we have explored the practical details of all our CAD approaches, evaluating each model individually to adjust their parameters. We described the implementation and evaluation of two supervised classification models (LSTM and XGBoost), and one unsupervised diarization pipeline, for activity detection, specifically with the metrics defined in the previous chapter. These efforts have laid the groundwork for subsequent analysis and comparison of results, where we will work with the best versions of each model and compare them with each other.

Chapter 5

Analysis and comparison of results

In this chapter, we provide a comprehensive analysis and comparison of the results obtained from our different Classroom Activity Detection (CAD) systems. We start by evaluating the LSTM model, considered as the main candidate because of its speed and suitability for sequence classification tasks. Then, we aim to understand its potential advantages compared to the more generic XGBoost model, which also operates on the same audio features.

Next, we compare the supervised LSTM and the unsupervised diarization pipeline. Despite being vastly different approaches, with the latter requiring no training data, we seek to ascertain the worth of the effort invested in annotating data for training a supervised system. We aim to gain insights into the trade-offs and effectiveness of each approach.

5.1 Analysis of the LSTM Classifier

As a first analysis, let us explore some results after performing CAD using the LSTM neural network model. This is one of the supervised classifiers as explained in the previous section, and it is trained with the same features as the XGBoost model.

The performance of both classifier models is similar -as will be shown in the following section-, but the LSTM was mainly preferred to be explored in more detail because the inference process runs considerably faster. Since this is a recurrent neural network, it is particularly well suited for classifying sequences (see Section 2.2) and it can leverage the context information better (e.g., the XGBoost has a major practical limitation on the context window length).

It is also important to mention that although the current implementation of this model overestimates the duration of label *a* more aggressively than XGBoost, this is only due to the particular settings used. The label weights in the loss function of this model can be calibrated very precisely, so this model is actually more flexible in this sense and could be readily adjusted upon users' request.

The LSTM parameters in use, are those shown to produce the best results in Section 4.1.2:

- `hidden_dim`: 64.
- `num_layers`: 2.
- `frames_per_sample`: 64.

As an overview of the results, Figure 5.1 shows both of the main metrics that can be calculated on top of the density estimations: correlation and Mean Absolute Error (MAE). This visualization

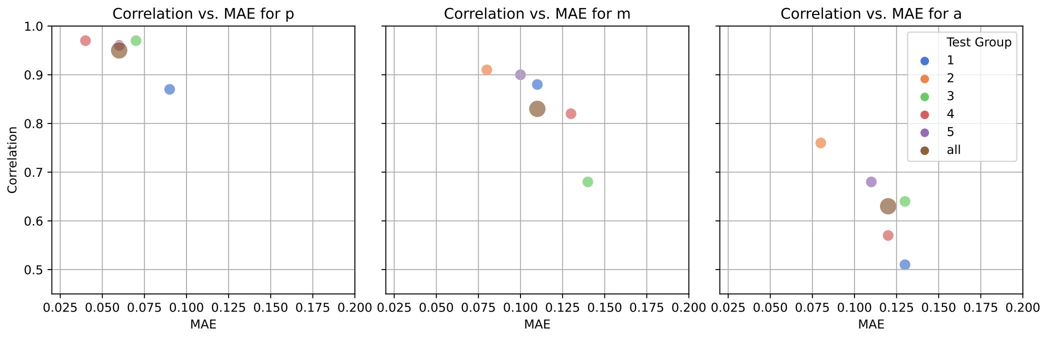


Figure 5.1: LSTM values of correlation vs. MAE for all labels *p*, *m* and *a* (left to right) and all test groups (groups 2 and 4 (red and orange) are overlapping on the label *p* plot). Note that top-left directions are better (higher correlation and lower MAE).

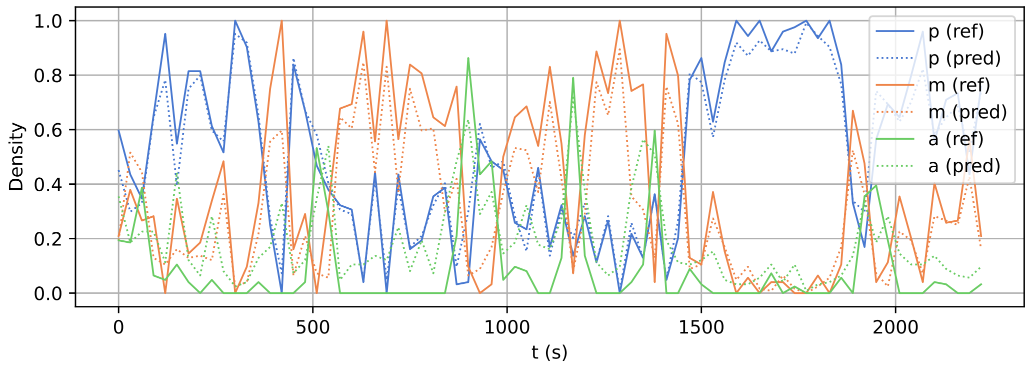


Figure 5.2: Ground truth (ref) and LSTM (pred) densities for all labels over test group 5, which has an average performance in terms of correlation and MAE (see Figure 5.1). This is an example of the result that a user may see at the time of analyzing a 45-minutes lesson (slightly cleaner since the user will not have ground-truth densities).

is useful to distinguish precisely which test groups are performing best or worst. As a reminder, each of these 5 test groups corresponds to the test splits of different groups of audios, hence they include different lessons. More precisely, each group contains five lessons.

The plots also show the metrics calculated on top of all test groups concatenated as one single audio signal, so it is a kind of average value (see the brown dot labeled as group *all*). As expected, there is a tendency in which higher correlation also mean lower MAE values, both of them being associated to better estimations. In particular, group 2 is the one in which the model is working better for all labels (it's overlapping for label p with group 4), and group 1 is the worst for labels p and a.

But let us first take a look at the performance of the model on an average group. Looking back at Figure 5.1, it can be seen that the dot corresponding to test group 5 is close to the average performance (the brown dot representing all groups) in all cases. The predictions of the model on this average group can be seen in Figure 5.2. This plot is intended to show the output in a format similar to that seen by a user of this CAD pipeline, showing the reference ground-truth densities as solid lines and the predicted classroom activity as dotted lines, with the same color for each label.

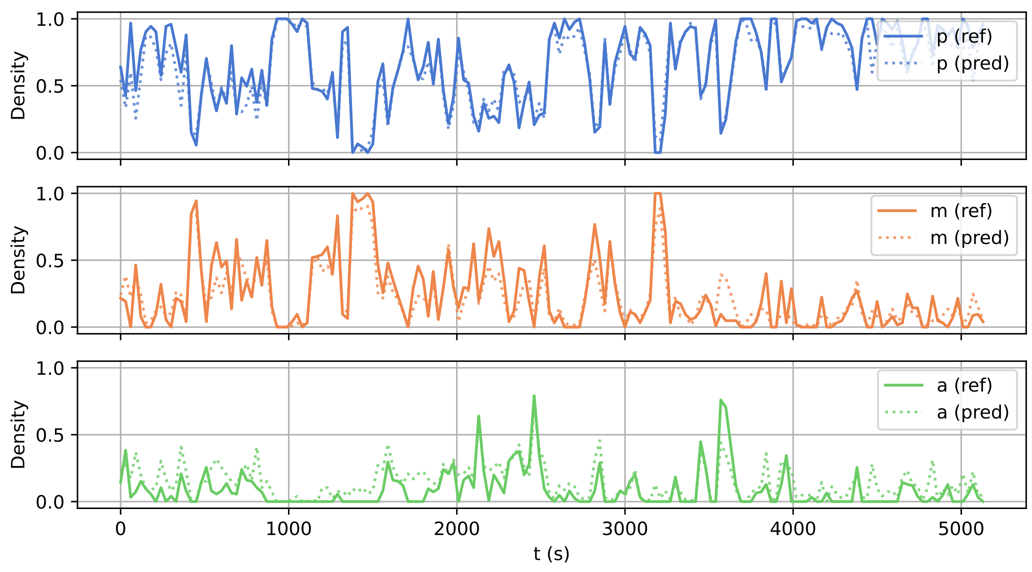


Figure 5.3: LSTM predictions for test group 2, which is the best example in terms of correlation and MAE for all labels (see Figure 5.1)

The figure shows that the general shape of the curves is very closely related, allowing the observer to estimate the main moments of the lesson precisely. At the beginning and end there is more activity in the p (teacher introducing to the class, then explaining tasks, and finally closing the lesson). In the middle of the lesson, there seems to be some group work activity that involves the whole class (higher values on the label m), and there are particular peaks of activity in label a where single students are intervening. All these key moments of the lesson are correctly detected, despite some false detections on individual student participations, and some underestimation of multiple voices.

All the peaks in label a are basically captured, except for the one around $t \simeq 400\,\text{s}$, which seems to be also mixed with label m in the reference. The false positives in label a generate small density peaks in zones that should be null, for example around $t \simeq 750$, among other places. Those in general correspond to short fragments where some student's voice stands out during group work instances. As was already discussed in previous sections, these cases can be very challenging to disambiguate, even for a human annotator.

Now, let us take a closer look at comparing the predictions with the reference densities. The best and worst performing groups (discussed above according to Figure 5.1) were chosen as a reference, to better understand and calibrate what these metric values mean in terms of predicted densities.

Figure 5.3 shows the predictions for test group 2, which corresponds to the best group in all labels. Each label is shown on a separate plot in order to visualize it together with the reference value and to note any subtle differences.

In this example, there are still overestimations in the label a, but the great majority of predicted peaks are actually present in the reference. For label p the prediction is basically matching the reference except for some minor shifts.

On the other hand, Figure 5.4 shows the predictions for test group 1, corresponding to the worst estimations in labels p and a. In fact, the student activity is largely overestimated across the

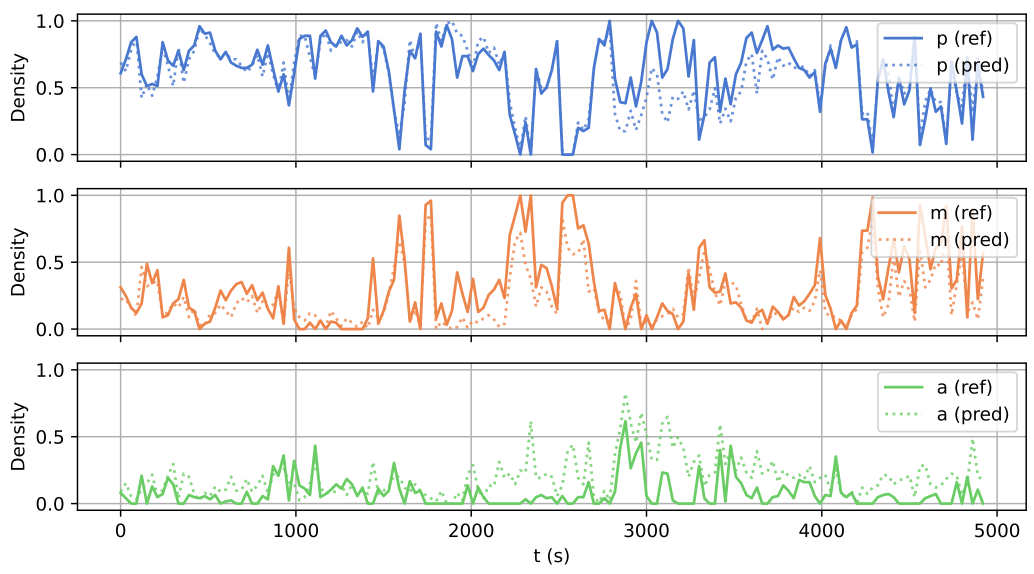


Figure 5.4: LSTM predictions for test group 1, which is the worst group for labels *p* and *a* in terms of correlation and MAE (see Figure 5.1)

lesson, being confused by both the teacher's voice and group work. Aside of a couple false positive peaks, it is also worth noting that the largest predicted peaks are still real and that no peak is missed in this label.

In summary, in this introductory section, we have seen the performance of the LSTM model in terms of correlation and MAE on all test groups and related some selected metric values to actual estimated densities. Examples of the best -and worst- case predictions were shown, and also one example showing all labels together on a 45-minutes audio with average performance, which is similar to the results that would be seen by a user of the CAD module.

5.2 Comparing LSTM vs. XGBoost

This section aims to compare both supervised classification models against each other. To recall from the implementation Section 4.1, these models are trained on top of the exact same audio features to make sure that the comparison refers to the capabilities of each model.

Both of these models are viable contenders for this particular application and may be more appropriate to use depending on the requirements and resources available. In particular, the LSTM benefits by a huge margin in terms of inference time, which is an important aspect to consider for this application. However, it requires GPU hardware in order to be trained, so this may be a limiting factor for some future use cases where the system needs to be retrained frequently, for example in the context of an active-learning platform.

As a first step, these models will be compared in terms of prediction performance in order to understand whether or not they could be used interchangeably. If one of them is clearly a better option, then the needed resources should be prioritized accordingly.

The comparison in this section is done using the best parameters for each model. The only exception to this criteria is the `context_window` parameter in the XGBoost model, which makes the

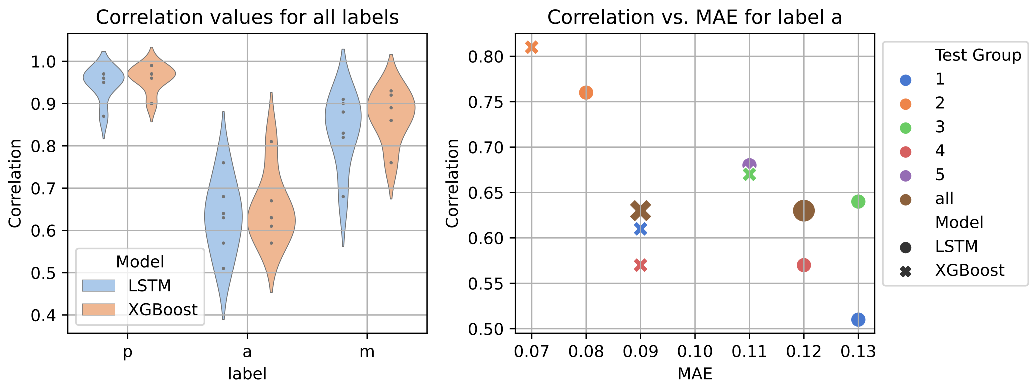


Figure 5.5: Comparison between LSTM and XGBoost models, both supervised. **Left:** correlation values (vertical axis) for all labels (horizontal axis) are shown for all 5 test groups and their concatenation (dots inside each violin plot, some values may overlap), showing a similar performance in all cases. **Right:** correlation vs. MAE values, only for label *a*, with different colors for each test group, allowing to see which group has each correlation value. Note that the bigger points show the values calculated over *all* test groups concatenated, as an average value. The point for test group 5 of XGBoost is not visible since it overlaps with the group *all* at (MAE=0.09, Correlation=0.63).

inference process very slow to run and does not justify the marginal gain in prediction performance. So the parameters chosen for each model are the following:

LSTM parameters

- `hidden_dim`: 64.
- `num_layers`: 2.
- `frames_per_sample`: 64.

XGBoost parameters

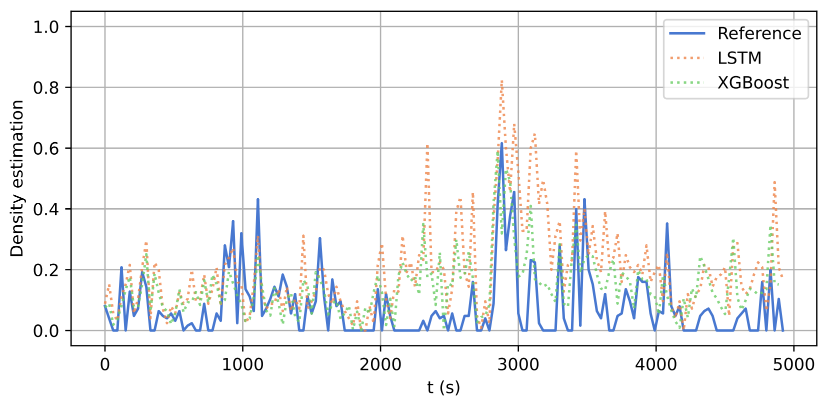
- `context_window`: 10.
- `max_depth`: 10.
- `gamma`: 0.

For more information on how these parameters were compared and chosen, see the Implementation Section 4.1).

The models with these parameters were run over all testing groups 1-5 (see Section 3.1.3), and there is also another group *all* which consists of the concatenation of all these testing groups. Since it is not obvious how to average each metric over data with slightly different label proportion and duration, the metric values in this group can be considered as the average performance values.

On the left side of Figure 5.5 the correlation values are shown per label, for both models, in all the available test groups. It can be seen that the distribution of correlation values across groups in both models is very similar. This is a first indication that it might be possible to swap between these models without a huge decrease in performance.

Looking inside of the distribution curves, there seems to be also a direct correspondence of points in both models, so an observer might assume that these points correspond to the same test groups (e.g.: it might be assumed that the points with the highest and lowest correlation in both models for label *a* belong to the same group).



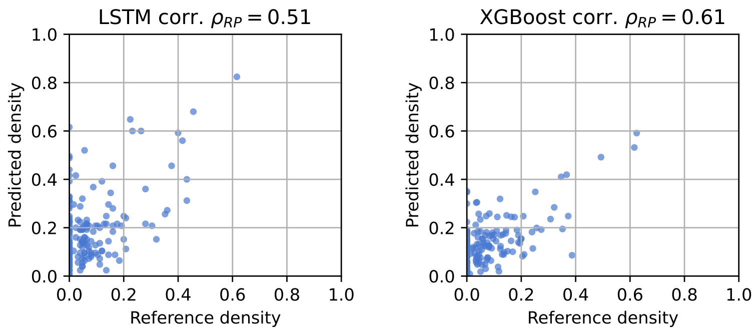
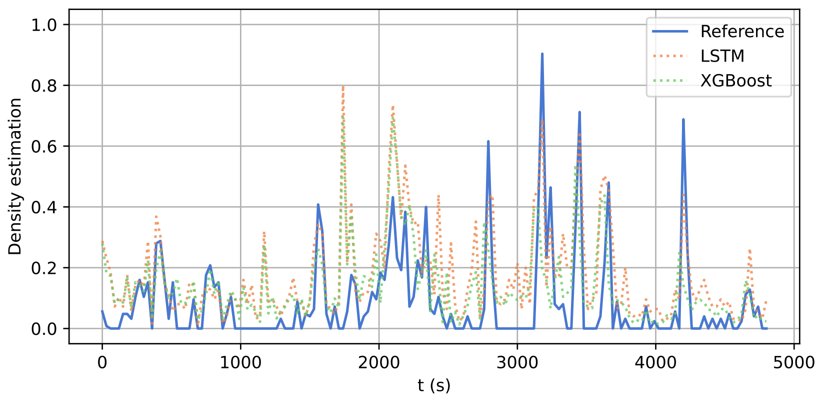


Figure 5.6: Comparison of predictions for label *a* over **test group 1**. This is the group with the largest difference in favor of XGBoost, both in terms of MAE and correlation. **Top:** density for label *a* as a function of time, removing non-labeled segments (there are still zero values in the reference which correspond to windows where only other labels were used). **Bottom:** correlation plots and coefficient values (see plot titles) for the same predictions shown on the top image. The temporal dimension is lost in this case.

To verify if this is the case, the plot on the right side of the same Figure 5.5 shows each test group in a different color, and each model with a different shape, but only covers the label *a*. From that plot, it is seen that the average correlation (group *all*) is basically equivalent, but the MAE is 0.03 higher (worse) for the LSTM.

To better visualize some of these differences with a concrete example, let us compare the predictions for label *a* over group 1, which is the one with the highest difference in favor of XGBoost according to the figure, both in terms of correlation and MAE. Figure 5.6 shows this comparison, where its clear that both models tend to overestimate this label, but the LSTM does it to a larger extent. Aside of that fact, both models are actually quite good at detecting peaks. From the correlation plots (at the bottom of the same figure) it can be confirmed that there are not many points where the reference is high but not the predictions, which is an indicator that both these models are useful to find moments in which the students speak, even though they produce some false positives.

A curious fact about the plot on the right in Figure 5.5, is that despite the correlation being higher in groups 1,2,3 for XGBoost, and only better for LSTM in group 5 (which overlaps with group *all* at (MAE=0.09, Correlation=0.63)), the average correlation for *all* groups is basically the



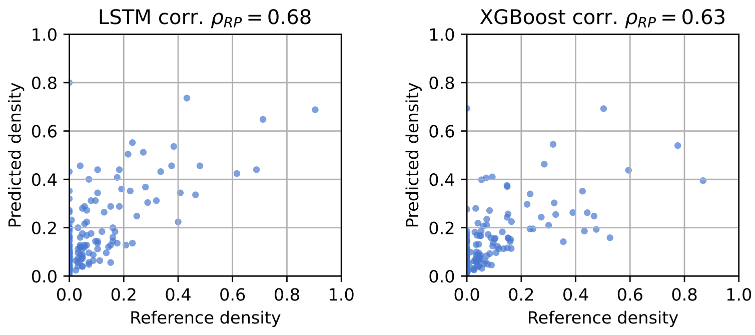


Figure 5.7: Comparison of predictions for label *a* over **test group 5**. This is the only group where there is a relevant difference in favor of the LSTM, only looking at the correlation value. However, it is enough to push the average correlation values together (group *all* in Figure 5.5). **Top:** density for label *a* as a function of time, removing non-labeled segments. **Bottom:** correlation plots and coefficient values (see plot titles) for the same predictions shown on the top image. The temporal dimension is lost in this case.

same for both models.

To try and understand why the correlation of group 5 has such a big impact in the average calculation, the predictions for it are also included in Figure 5.7. The particularity of that test set is that it contains some large peaks in which the LSTM seems to work really well. Compared to the previously analyzed test group 1 (in Figure 5.6), this group has sharper and higher peaks in the reference density. Looking at the correlation plots at the bottom of the figure, it seems that the LSTM follows those large values better (closer to the diagonal $x = y$).

In turn, large values tend to have a greater impact on the Pearson correlation coefficient being used (introduced in Equation 3.3). In fact, a well-known problem of this particular coefficient is that it can be affected by outliers, like these high density peaks appear to be. If this turned out to be a serious problem, there are alternative formulations like *Spearman's rank correlation coefficient* [4] which are more robust to outliers, although they may also introduce undesirable effects.

To verify if these peak points are really outliers in the whole test data and to see if it is actually a serious issue, all the predictions from both models for all test groups are included in the correlation plots shown in Figure 5.8.

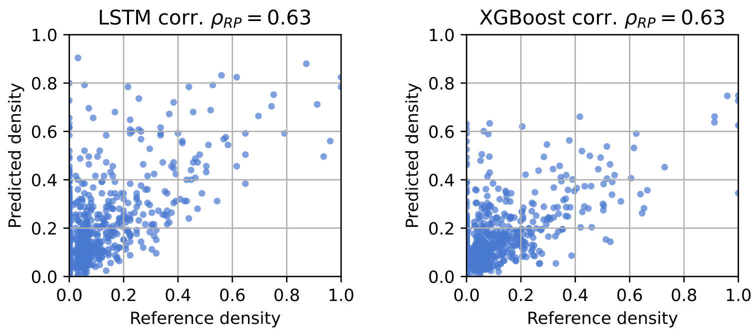


Figure 5.8: Correlation of density for label *a* predicting with both models over all the test groups concatenated (group *all*). There is a tie in the values of the correlation coefficients but actually each model has a distinctive pattern of predictions: the LSTM seems to follow better the larger density values at the price of overestimating more often. These predictions are not shown as a function of time because they cover a large time span and the visualization is not clear.

From that figure, it seems more natural that both correlation coefficients are basically equal. On one hand, although the LSTM model (left) tends to overestimate the density more often, it is also better at predicting when the value is actually high. On the other hand, the XGBoost model (right) shows more points below the diagonal line $x = y$, which means that there are some peaks in student activity that may be underestimated or even go undetected in this model.

However, this is not necessarily a deficiency of either approach. This calibration can be configured differently by modifying the label weights in the loss function for the LSTM model, or the undersampling ratio for the p label that is used in the XGBoost model (see Section 4.1). In both cases, these techniques aim to mitigate the severe label imbalance problem in which the p is way more frequent, but there is room for refinements in their configuration (see Section 3.1.3).

In hope of increasing the probability of detecting less frequent labels such as a, some overestimation might be acceptable even if it leads to false positive moments of student participation. The good news is that this trade-off can be adjusted by iterating over these parameters, retraining, and evaluating with the potential users.

This tendency to overestimate some labels and how it affects the detection of other labels can be better understood by looking at the confusion matrix in Figure 5.9. Recalling the meaning of this matrix from Section 3.2.2, each cell indicates the probability of hitting the *Predicted label*, conditioned on the sample having the *True label* for that row (i.e., probabilities add up to 1 when added per row). For example, for the LSTM (left), if we pick a segment with a reference label m, there is a 10% chance that the prediction will be p, but only 4.7% to predict m if the true label is p.

By comparing the confusion matrices in Figure 5.9, it can be seen that the probability to correctly detect student samples a (that is, the recall of the label) is better for the LSTM than for XGBoost (64% vs. 51%, respectively), but it also missclassifies more m segments as a (29% of the m segments are wrongly classified as a in the LSTM vs. 23% in XGBoost).

5.2.1 Teacher Talking Time with Classification Models

A widely used metric in Classroom Activity Detection systems is the *TTT: Teacher Talking Time* estimation (see Section 3.2), the total time the teacher talks in a lesson.

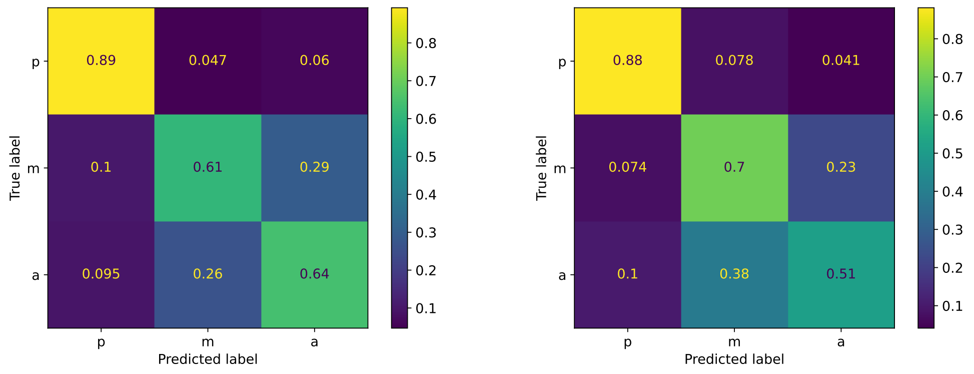


Figure 5.9: Confusion matrices for the LSTM (left) and the XGBoost (right) models, considering all test groups. The matrix is normalized to show hit probability conditioned on *True label* (see Section 3.2.2).

model	label	Min. Rel. Error (%)	Max. Rel. Error (%)
XGBoost	p	-9.2	-1.5
XGBoost	ma (a or m)	2.1	16.4
XGBoost	a	29.1	77.4
XGBoost	m	-14.0	11.7
LSTM	p	-6.1	0.8
LSTM	ma (a or m)	-1.2	11.6
LSTM	a	65.4	146.9
LSTM	m	-33.4	-8.3

Table 5.1: Comparison of relative errors in total label times estimated. The relative error is calculated as indicated in Eq. 5.1, for each label. It was calculated separately on each of the 5 test groups (see Section 3.1.3), and only minimum and maximum values are shown. The distribution of the errors is represented in Figure 5.10.

In the setup used in this work, the TTT is just the total duration of the label p. For completeness, the duration of the other labels a and m was also estimated, and as a final step they were also measured together as one single label: ma, which is complementary to the label p.

The error for these estimations was measured separately in each of the 5 test groups available, whose durations are between 5000-6000 s as seen in Table 3.1. These errors were then calculated in relative terms so that they can easily be extended to any other audio duration, like a 45-minute lesson. The relative error for the duration of each label is calculated simply using:

$$\text{Rel. Error}(\%) = 100 \times \frac{\text{Predicted duration} - \text{Annotated duration}}{\text{Annotated duration}}. \tag{5.1}$$

The *Annotated duration* is the reference duration of ground-truth labels (human annotations). The minimum and maximum relative errors for each label, after measuring in the 5 test groups, are shown in Table 5.1. This corresponds to the worst-case underestimations (min error) and overestimations (max error) of each model and label.

This table shows only the extreme values of the errors, but the distribution of those in the different test groups can be appreciated in Figure 5.10. In that figure, the relative error is rescaled using the average label duration in a lesson of 45 minutes. The proportion used for each label is approximately equal as calculated in the data analysis section (3.1.3) and shown in fig. 3.3:

- Label p: 60%, or 27 minutes.

- Label m: 30%, or 13.5 minutes.
- Label a: 10%, or 4.5 minutes.
- Label ma (a or m): 18 minutes.

Then it is straightforward to translate the relative error to error in terms of minutes, by multiplying with these factors.

In Figure 5.10, it becomes even more clear that the LSTM model overestimates the label a and underestimates m to a greater extent than XGBoost (as was already shown with the confusion matrix in the previous section). This is the consequence of the calibration of the loss weights discussed in the previous section. These weights could be tuned to avoid this overestimation problem, but it would be at the cost of potentially missing some density peaks in label a, which is challenging to detect and is usually valuable information for the density plots, since they correspond to some particular student's participation.

With current calibration, the TTT is close to the ground-truth value in both models (that is, the estimated duration for the label p only). Although it is being underestimated in most cases, the error shift seems to be quite systematic in the LSTM, to the point that it could be corrected by multiplying it by some fixed coefficient, which should be adjusted together with the loss weights. This may be a workaround to better estimate the TTT without compromising the student peaks detection, as discussed above.

It is interesting to note that although the LSTM is worse at estimating labels a and m separately because of the calibration discussed, when both labels are considered together, the estimations are slightly better than the XGBoost for both labels p and ma (bottom image of Figure 5.10).

5.2.2 Benchmark of execution times

Given that the performance of both models is similar in terms of classification accuracy, the inference speed of each model becomes more relevant in order to use these models in a real-world application in the future.

In this section, the execution time of both models was measured on the same hardware, which has the following characteristics:

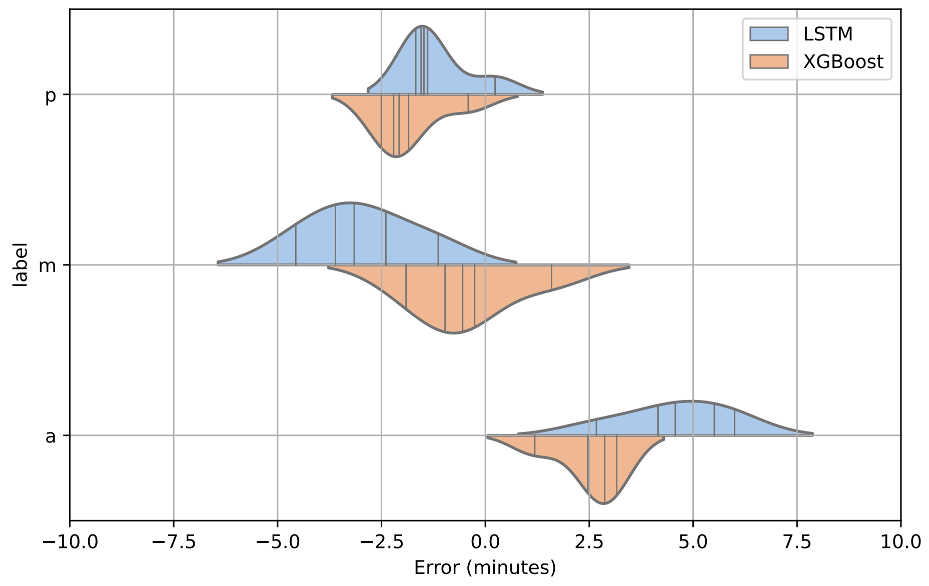
- **CPU:** Intel(R) Core(TM) i7-10700F CPU @ 2.90GHz (64 bits).
- **GPU:** NVIDIA GeForce RTX 3090 @ 24 GiB RAM.
- **RAM:** 64 GiB DDR4.

It is worth noting that actually both models could run on a machine with much more limited hardware, the only important requirement for the LSTM model is that it needs 770MiB of GPU RAM. The XGBoost implementation cannot leverage GPU to perform faster inference (it does so for faster training only).

The models were run on all five test groups two times. Since the test groups have slightly different durations, the total inference time was divided by the input audio duration to normalize the results to a per second basis. This was done after verifying that the total inference time is directly proportional to the input duration, in both models and also in the feature extraction stage.

Since the feature extraction stage is exactly the same in both models (see Section 4.1), this step was measured separately in both cases. The feature extraction time should be added to the model inference in all cases to get the total processing time for the entire classification system. The results are shown in Table 5.2.

In the table, two different versions of the XGBoost model are included because the inference time is strongly dependent on this parameter. The reason to include those particular versions



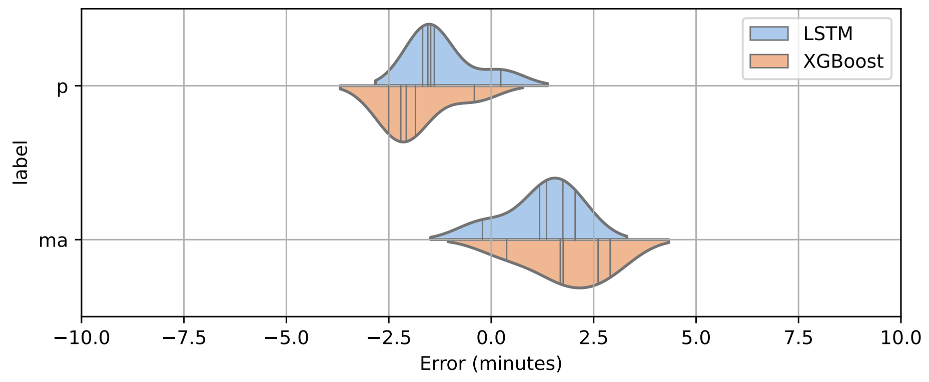


Figure 5.10: Errors in the estimation of total times per label. The slashed lines indicate the average duration of each label, as a reference value. The error was calculated on the usual 5 test groups, and then rescaled to the duration of a regular lesson of 45 minutes. **Top:** Total durations are estimated separately for labels *a*, *m* and *p*. Both models overestimate *a* and underestimate *m*. **Bottom:** Labels *m* and *a* are grouped together as one label *ma*, so their errors tend to cancel out. As a result, *ma* is slightly overestimated, while *p* is slightly underestimated.

Model	**1 second of audio (ms)**	**45 minutes of audio (s)**
Feature extractor	1.05 ± 0.10	2.83 ± 0.27
LSTM	0.92 ± 0.02	2.50 ± 0.05
XGBoost (max_depth=6)	10.51 ± 0.11	28.38 ± 0.31
XGBoost (max_depth=10)	20.30 ± 0.25	54.82 ± 0.67

Table 5.2: Benchmark of inference execution times for the classification models. Inference time is directly proportional to the input audio duration, so the first column corresponds to processing each segment of 1 s of audio (note that inference time is in milliseconds in this case) and the last column is the time required to process a whole lesson recording (assumed as 45 minutes of audio). The feature extraction stage is measured separately in both models and should be added to either of them to calculate the total classification time.

is that the best model that was used in the comparison in the previous section is the one with `max_depth=10`, but actually the alternative with `max_depth=6` is twice as fast and has a very similar performance in terms of accuracy (as was seen in Section 4.1).

In any case, the table shows that the LSTM is one order of magnitude faster than the best XGBoost competitor. In fact, this model is even faster than the feature extraction stage because it takes advantage of the parallel processing that the GPU hardware enables.

5.2.3 Summary: LSTM vs. XGBoost

In terms of prediction accuracy, there is no clear winner between the LSTM and XGBoost classification models. Any of them could be useful, depending on the available hardware and the requirements of the particular application to develop. With the current settings, the LSTM seems to overestimate the predictions for the student activity more often than the XGBoost model, but it is also better at detecting peaks in this kind of activity. However, this could be adjusted in both models (by tuning the loss weights in the LSTM or the under-sampling strategy in XGBoost) and so it does not seem like a strong reason to prefer any of them.

The main decision factor would be the desired inference speed and available hardware resources. The LSTM runs much faster, but it is only feasible if a GPU is available, so the XGBoost should be considered as a backup option in the cases where the hardware budget is tight or when the execution time is not considered a critical aspect of the system as a whole.

5.3 LSTM vs. Diarization

In this section, the supervised (LSTM) and unsupervised (Diarization) approaches to Classroom Activity Detection are compared. In the previous section, it was observed that the differences between supervised classifiers (LSTM and XGBoost) were not significant enough to discard any of them. Instead, the decision should be based on the available hardware resources.

For this stage of the work, the GPU hardware (already mentioned in previous section) is available and so the LSTM is chosen for the comparison in this section, since its much faster to run inference on it. It is important to recall that the diarization pipeline being used also requires a GPU to run the segmentation and embedding extraction steps.

As a first approach to the comparison, Figure 5.11 (left) summarizes all correlation values for all labels a, m and p, in each of the five test groups.

On the right side of the same Figure 5.11, it is possible to see which group has the highest or lowest correlation and MAE values. Test group 2 (orange) has the lowest performance for the diarization system both in terms of correlation and MAE (bottom right circle dot). On the other hand, group 5 (violet circle dot) is the one with the best performance for diarization.

Both test groups 2 and 5 overlap with other points for the LSTM values (they should be orange and violet crosses), but they will be shown individually in the following figures.

Figure 5.12 shows the comparison of both models, particularly for group 2 (the one with the worst performance for diarization). It is interesting to note that the LSTM model has a very high correlation value of $\rho_{RP} = 0.96$ for that case. On the other hand, the diarization system is missing some high peaks (see around $t = 1000$) and in turn it is predicting peaks that should not be there (near $t = 1500$).

Unfortunately, it is not possible to include audio in this report, but listening to the audio of those intervals manually, it was observed that there is nothing particularly challenging in those sections. On the contrary, the teacher's voice is very clear. For some reason, however, the diarization system is assigning those segments to another speaker different from the teacher, causing the teacher's density values to draw away from the reference.

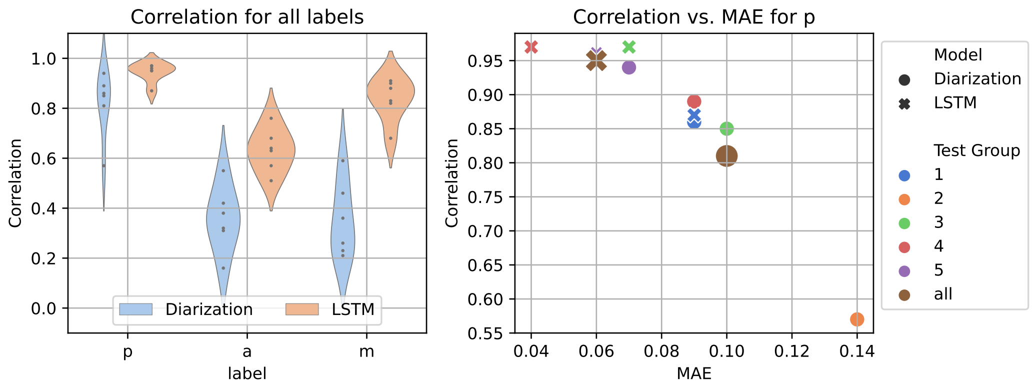
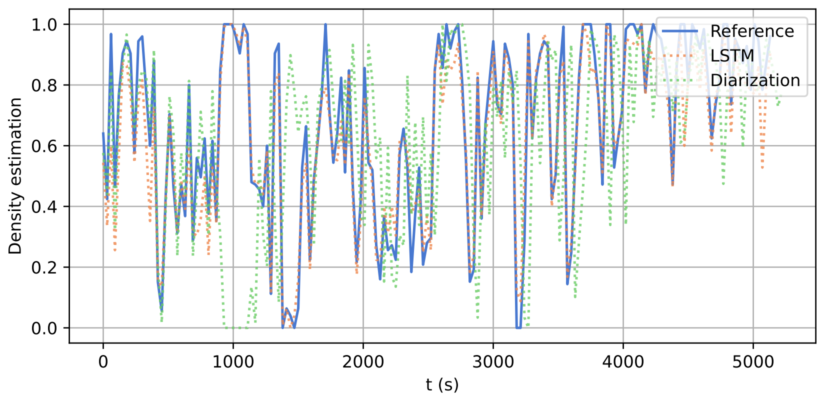


Figure 5.11: Comparison between LSTM (supervised) and Diarization (unsupervised) models. **Left:** Correlation values for all labels in all 5 test groups and their concatenation (dots inside each violin plot, some values may overlap). **Right:** Correlation vs. MAE values (horizontal axis) only for label *p*, with different colors for each test group, allowing to see which group has each correlation value. Note that the bigger points show the values calculated over *all* test groups concatenated, as an average value.



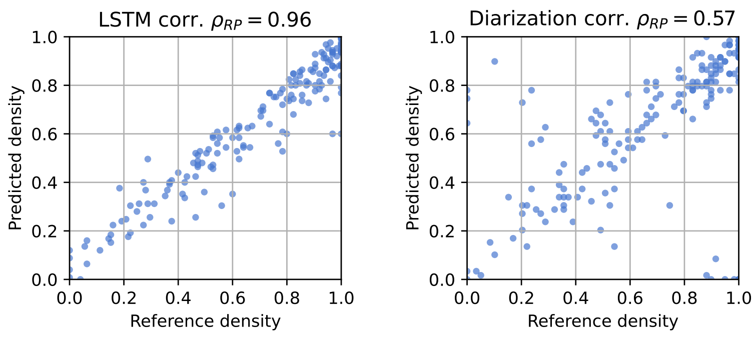
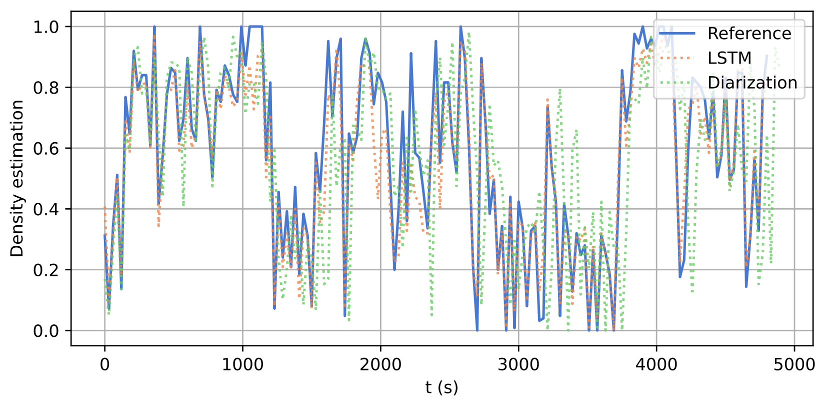


Figure 5.12: Comparison of predictions for label *p* over **test group 2**. This is the group with the worst performance for the diarization system. **Top:** Predicted and References densities for label *p* as a function of time, removing non-labeled segments. **Bottom:** correlation plots and coefficient values (see plot titles) for the same predictions shown on the top image.



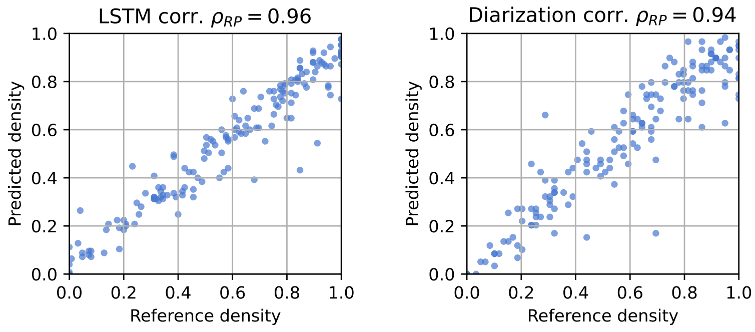


Figure 5.13: Comparison of predictions for label p over **test group 5**. This is the group with the best performance for the diarization system. **Top:** Predicted and References densities for label p as a function of time, removing non-labeled segments. **Bottom:** correlation plots and coefficient values (see plot titles) for the same predictions shown on the top image.

Having analyzed the case where the diarization system performs worse, let us now take a look at Figure 5.13 which shows the test group with the best performance (group 5 as mentioned previously).

In this case, the LSTM correlation is still better than the diarization value, but the difference is very small. In fact, considering again Figure 5.11 and adding the correlation values for the LSTM in groups 2 and 5 in figures 5.12 and 5.13, it can be confirmed that the LSTM works better than the diarization system in all test groups for the teacher's voice detection. However, except for test group 2, the difference is not as great for this particular label.

However, for the other labels a and m, there appears to be a significant gain in performance when comparing these models in Figure 5.11.

The main performance issue in the diarization model seems to be because it is not correctly distinguishing the labels m and a -considering that the performance of p is much better than the others-. To verify that hypothesis, the confusion matrices for these models can be compared in Figure 5.14.

Looking at the confusions, it becomes clear that the main issue with the diarization model is that the strategy used to select label m is not a good one. Indeed, picking a sample that was

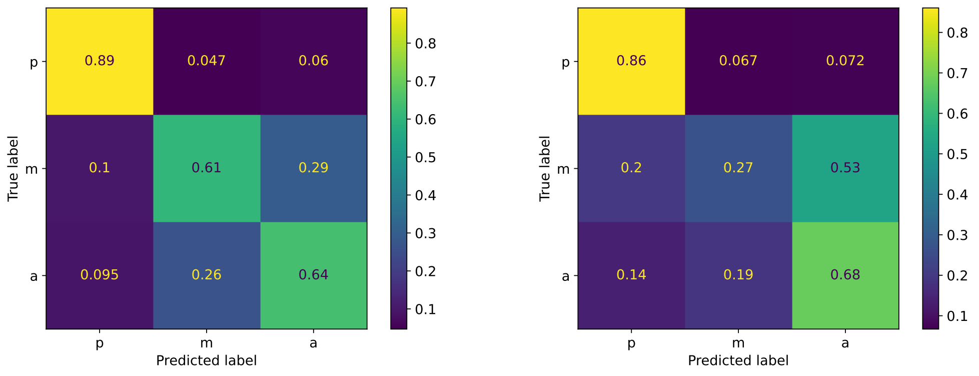


Figure 5.14: Confusion matrices for the LSTM model (left) and the diarization pipeline (right). The matrix is normalized to show hit probability conditioned on *True label* (see Section 3.2.2).

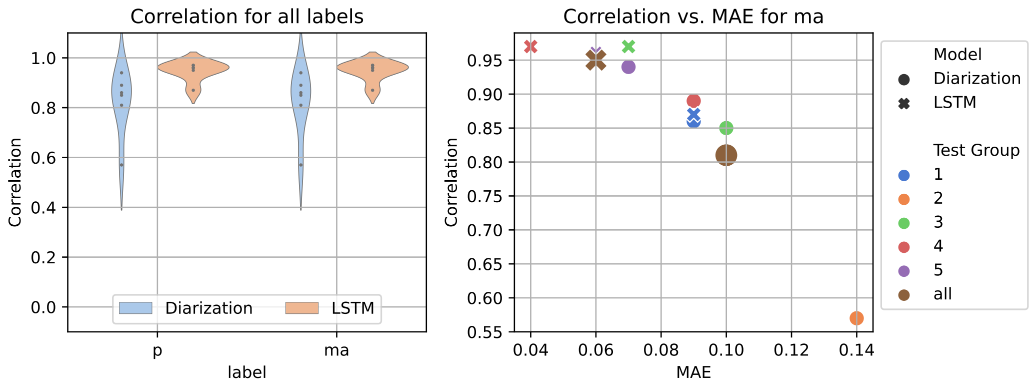


Figure 5.15: Comparison of correlations for both models on all 5 test groups, only trying to distinguish the teacher's voice *p*, or any of *a*,*m* (label *ma*). As expected, the distribution of each model in both labels are equal, since the non-labeled segments are removed and so the proportion of *p* is complimentary to the proportion of *ma*.

manually labeled as m (True label), there is a 53% probability that the diarization system will make a mistake and choose label a instead, while the probability to predict it correctly is only 27%.

The probabilities of mistaking the labels a and m with p are also higher, as can be seen by comparing the first column of both matrices. But since the problem is unbalanced and the label p is way more frequent than the others, these mistakes are not so significant as to degrade the performance in that label -as can be verified by looking at the top row of the matrices-.

Up to this point, it is clear that the diarization system is not good at distinguishing labels a and m. However, the most important feature that these models need to have is to distinguish the teacher's voice (label p) from any student activity (labels m or a). So, to get an idea about how good an unsupervised system can be for this task, this system was also evaluated using only two labels: p vs. ma ($a + m$).

Figure 5.15 shows this evaluation, where both systems compared only need to distinguish the teacher's voice, versus any of the m or a labels (labeled ma). As a sanity check, the first thing to observe is that actually the distribution of each model in both labels is exactly equivalent, since

model	label	Min. Rel. Error (%)	Max. Rel. Error (%)
Diarization	p	-7.1	8.1
Diarization	ma (a or m)	-10.6	12.8
Diarization	a	167.8	235.4
Diarization	m	-67.8	-43.8
LSTM	p	-6.1	0.8
LSTM	ma (a or m)	-1.2	11.6
LSTM	a	65.4	146.9
LSTM	m	-33.4	-8.3

Table 5.3: Comparison of relative errors in total label times estimated. The relative error is calculated as indicated in Eq. 5.1, for each label. It was calculated separately on each of the 5 test groups (see Section 3.1.3), and only minimum and maximum values are shown. The distribution of the errors is represented in Figure 5.16.

their densities are complimentary. In other words, we already had this information by looking only at the *p* label in Figure 5.11 of this section.

Aside of that fact, this figure shows more clearly that the diarization system is comparable to the LSTM when trying to distinguish only the teacher's voice, considering that it is an unsupervised method that does not require any training to run. Except for the test group 2 (the lowest point for the diarization correlation), the differences between those models are not so large as to discard the unsupervised option, as long as distinguishing *m* from *a* is not a priority issue for the desired application.

5.3.1 Teacher Talking Time with Diarization vs. Classification

As was already estimated in the comparison between supervised models (see Section 5.2.1, another important metric to compare between models is the Teacher Talking Time (TTT).

Fortunately, this information only depends on the estimation of the *p* label, and hence it should be possible to have a relatively good approximation by using the unsupervised approach. As shown in the previous section, for this particular label the performance of the unsupervised diarization pipeline is worse in general but at least comparable to the supervised system (except in 1 of 5 test groups where it is considerably worse).

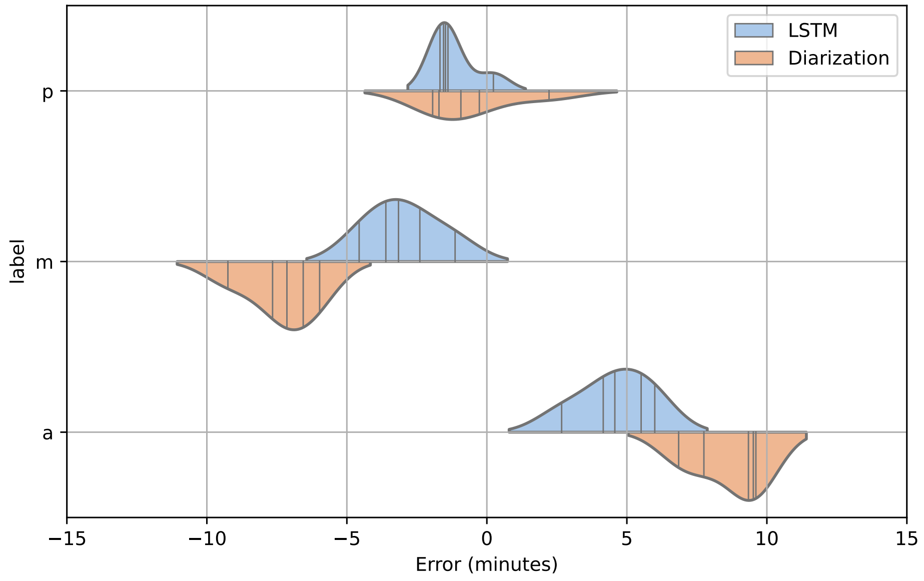
The error was again measured separately on the five available test groups and then calculated in relative terms using the elementary Eq. 5.1.

The results of measuring the relative error with that equation are shown in Table 5.3, where it is seen again that it is only in an acceptable range for labels *p* or the combined label *ma*.

These relative errors are rescaled to the duration of a 45 minute lesson, using the same procedure as in Section 5.2.1, so that the errors can be represented in minutes in Figure 5.16.

It is important to note that the diarization estimations are way off the reference value for labels *m* and *a* separately, to the point that the durations seem even reversed (swapping the estimated durations for these labels, would produce better results for this particular metric). This indicates once more that the current heuristic for separating these labels in the diarization pipeline should be revised or not used at all.

For the supervised model, it might be possible to decrease this error by tuning the label weights in the loss function (see Section 2.2) to avoid such a large overestimation, but the problem is still very challenging due to the very nature of these labels, which are very hard to disambiguate even for a human in many situations. Furthermore, lowering the weight for *a* would mean that some peaks of student activity may be lost in density estimates, which would probably be a negative effect in terms of usefulness for evaluators.



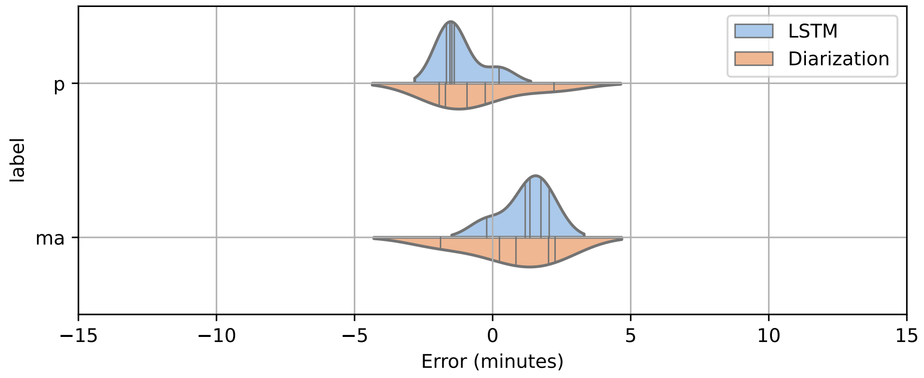


Figure 5.16: Distribution of errors in the estimation of total times per label. The slashed lines indicate the average duration of each label, as a reference value. The error was calculated on the usual 5 test groups, and then rescaled to the duration of a regular lesson of 45 minutes. **Top:** Total durations are estimated separately for labels *a*, *m* and *p*. It is important to note that the diarization distribution corresponding to label *a* is actually around the average time of *m*, so the error is a lot higher than it may appear at first sight. Both models overestimate *a* and underestimate *m*, but the diarization is way off the correct value. **Bottom:** Labels *m* and *a* are grouped together as one label *ma*, so their errors tend to cancel out. As a result, *ma* is slightly overestimated, while *p* is slightly underestimated.

5.4 Impact of Data on a Supervised Model

Aside of the different technical foundations, the two approaches being compared in this section are very different in the requirements that they need to work.

With the amount of manually annotated data available, it was possible to use a part of it for training the supervised classification models, and the remaining data were used to compare the performance between them and also against the unsupervised diarization pipeline.

Up to this point, it was shown that the supervised approach was superior using the best models that could be trained on these data. However, the real usage of these models will be to detect classroom activity in new recorded lessons, not necessarily annotated by humans. The new lessons may be from different teachers than those in which the system was trained.

So, one of the key questions that remains unanswered is whether or not the supervised approach is able to generalize on new data. And how much human-annotated data is needed to train it such that it is equal or better than the unsupervised approach?

More precisely, it is important to answer these questions about the supervised approach:

- How many lessons (i.e: different voices) do we need to add to the training set, in order to generalize to new (unknown) lessons?
- How much time from each lesson do we need to annotate in order to make those voices "known" to the model?

To answer both of these questions, many versions of the LSTM classification model were created and compared, each of them trained on different variants of the available dataset. Only the training data was changed, and each version was evaluated in the same 5 test groups as always.

The different variants of the training dataset were created using different combinations of the data groups and divisions defined in Section 3.1.3. In fact, all the complexity involved in creating such data groups was intended to answer these questions.

5.4.1 Adding more lessons

The aim of this section is to answer the first question posed above, which can also be interpreted as: can we find the point at which the supervised model no longer improves after adding new lessons and teacher voices? What is the minimum amount of training data required to surpass the performance of the unsupervised system? Or how does the model improve when we add new lessons?

It is important to recall from Section 3.1.3, that each of the 5 different data groups contained audio from different lessons. In other words, the 25 available lessons are divided into these 5 groups.

The strategy to answer how the supervised model generalizes to new (unknown) lessons is to train it on a dataset that only contains some groups, and then evaluate it on the groups that were left out from the training set.

For example, as a first step, the model is trained using only group 1, and then the prediction performance is measured on the test splits from groups [2, 3, 4, 5].

Only one example of this model being trained in group 1 and evaluated in test group 2 is shown in Figure 5.17. The performance of this LSTM trained on reduced data is comparable to the Diarization approach, although the sections where each model fails to detect peaks are different. This test group was also used for comparison but using the full LSTM model, in Figure 5.12. But in this new image, the predictions are significantly worse because not only the model is predicting over unseen teacher voices, but it is only trained on 5 lessons (1 train group).

The same estimation is performed on all other test groups [3, 4, 5] that were not present in the training set. On each of them, the correlation value is calculated.

Since there is nothing special about training with group 1, the model is also trained on group 4 and tested on [1, 2, 3, 5], and also trained on group 5 and tested on [1, 2, 3, 4].

So, only for the first step (*Number of training groups: 1*), there are $3 \times 4 = 12$ test densities to calculate metrics and build a confidence interval and average value. This is the first point for the LSTM value in Figure 5.18, where it is seen that the performance of the supervised and

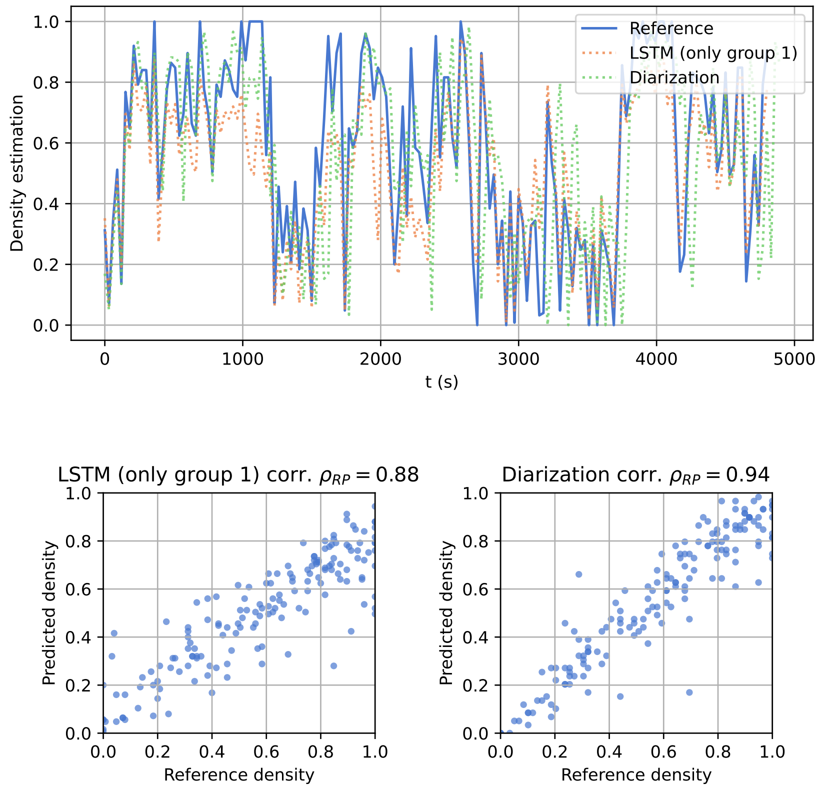


Figure 5.17: Comparison of predictions for label p over **test group 5**. The LSTM model was only trained with train group 1, so its performance is still worse than the Diarization pipeline. **Top:** Predicted and References densities for label p. **Bottom:** correlation plots and coefficient values (see plot titles) for the same predictions shown on the top image.

unsupervised approaches is quite similar in terms of correlation, when we only add one training group.

The second step for the same figure (*Number of training groups: 2*), evaluates the same model (with exact same hyperparameters) only trained on 2 groups, and evaluated on different remaining 3 groups. In this case, 3 different model variants are evaluated: trained in groups [1, 2] and tested in [3, 4, 5], trained in [4, 5] and tested in [1, 2, 3], and trained in [1, 4] and tested in [2, 3, 5]. Hence, there are 9 test points to build the confidence interval and average value.

The same basic procedure is repeated for all the other points, with the caveat that the more training groups are added, the less test groups are available. In Step 3, each version of the trained model can be evaluated in 2 groups (3 versions are trained, so there are $3 \times 2 = 6$ test points to build the confidence interval and average value). In Step 4, since there is only 1 test group left, 5 different model versions are trained.

In the final step (*Number of training groups: 5*), the model is evaluated with previously seen voices (since there are no groups left out). So, this step is added in order to show what the gain is of evaluating the model only with known teachers. Surprisingly, this step is not particularly large for label p after all previous audios were added.

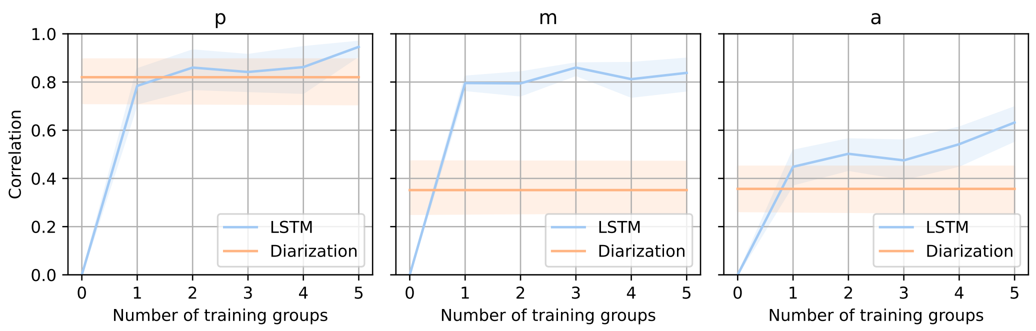


Figure 5.18: Comparison between the unsupervised (Diarization) system vs. the supervised (LSTM) model, while adding new groups of audios (i.e: adding new lessons and teachers) to the supervised model. In all cases the model is tested only on non-training groups (unseen voices), except for the last one (since all 5 groups are used for training). The colored spans around each curve represent the confidence interval of 95% of each metric, as measured on all 5 test groups.

From this observation, it can be concluded that after adding 20 different teacher voices ($4\ groups \times 5\ voices/group$), there is no significant gain in adding more voices to the training set. Even if the model is to be evaluated on unseen voices, there is no great benefit in adding some annotated samples with these voices to the training set. At least, this seems to be the case for label p, which is actually the one that should be more prone to a lack of generalization over the voices of new teachers.

It is interesting to note that for label m there is no apparent benefit in adding new audios to the training set. This may not be very surprising, considering that these samples are similar across different lessons, consisting of some noises and multiple background -most frequently indistinguishable- voices with overlapping happening when students are working in pairs or answering the same questions together.

Perhaps more surprising is the fact that the label a is the one that improves the most when more training data is added. Even extrapolating beyond 5 training groups, it seems to be the case that this label could continue improving. The explanation for this fact might be that this label is actually the one with the lowest proportion of samples (only around 10%, as shown in the previous section and in Section 3.1.3). And, actually, these samples correspond to students whose voices are clear and distinguishable. So, these samples should not be imagined as a generic children's voice but instead as other particular voices, only with very few samples each one.

For this label, it could be said that even though the performance of the supervised model is better than the unsupervised one, it is not yet at a level of maximum generalization. In other words, the supervised model would benefit from a larger training set, with more samples with label a and more diverse student voices, particularly.

To put these metrics in more context, we can visualize the evolution of the LSTM model as more training data are added. Figure 5.19 shows the particular example of the predictions for test group 5 and label p, while Figure 5.20 shows the same evolution for the same test group, but for label a.

In both cases, it is worth noting that the estimations and the correlation values (shown in the titles of the plots at the bottom as ρ_{RP}) are not monotonically improving as more data are added. There are some groups that actually make the model worse at predicting this particular test group (e.g., groups 2 and 4 in this case, have lower correlations than groups 1 and 3 respectively).

This is not so surprising, considering that the data being added belong to completely different

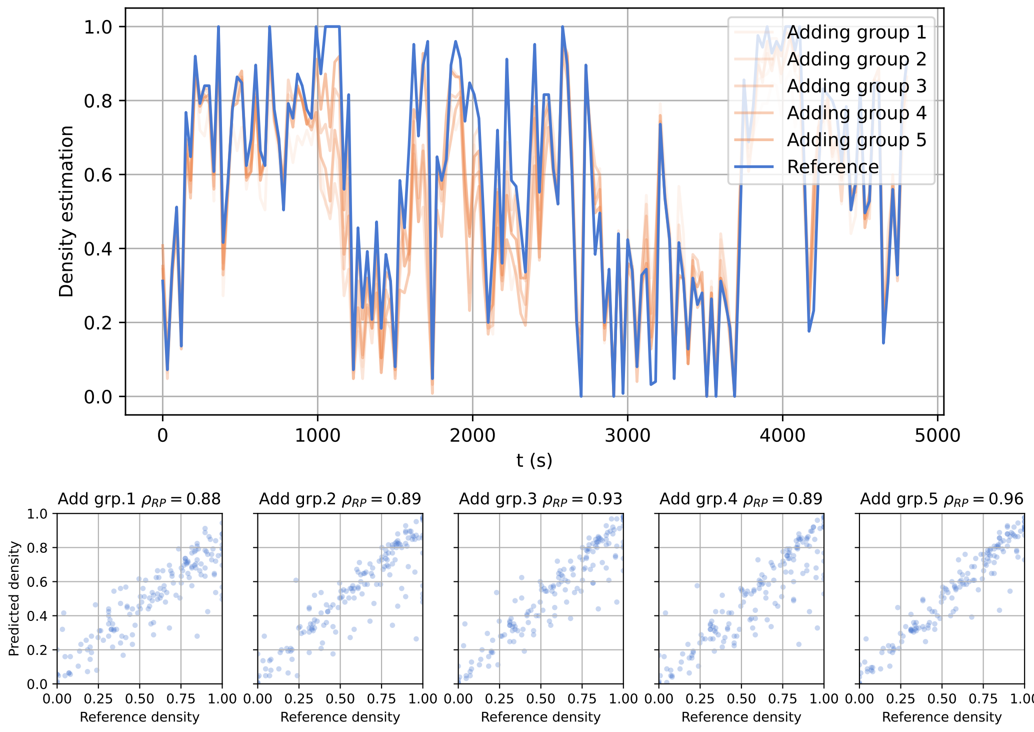


Figure 5.19: Improvement in predictions for label p over **test group 5**, while adding training groups 1 to 5. **Top:** Predicted and References densities. **Bottom:** correlation plots and coefficient values (see plot titles) for the same predictions shown on the top image.

lessons with different voices and even environments (noise levels, classroom organization, etc.). But looking at the first and last correlation values, it is clear that the improvement in the correlation value is significant.

5.4.2 Adding more time per lesson

In this part, the aim is to answer the second question posed at the beginning of this section: How much time from each lesson do we need to annotate in order to make those voices "known" to the model?

This question can also be stated as: How does the model improve when we add more annotated time, always using the same groups of lessons?

Again, the strategy to answer these questions involves training many versions of the same model on different training datasets. This time, making use of the training splits that were also defined in Section 3.1.3.

From that section, let us also recall that the position of the splits and the audios included on each group, were carefully chosen to create balanced splits and groups, in terms of the resulting label proportions. This balance is guaranteed when any given split is selected, as long as all the groups are included. For example, we should not create a training dataset using only split 1 from group 1, because that particular subset might have a balance between labels that is completely

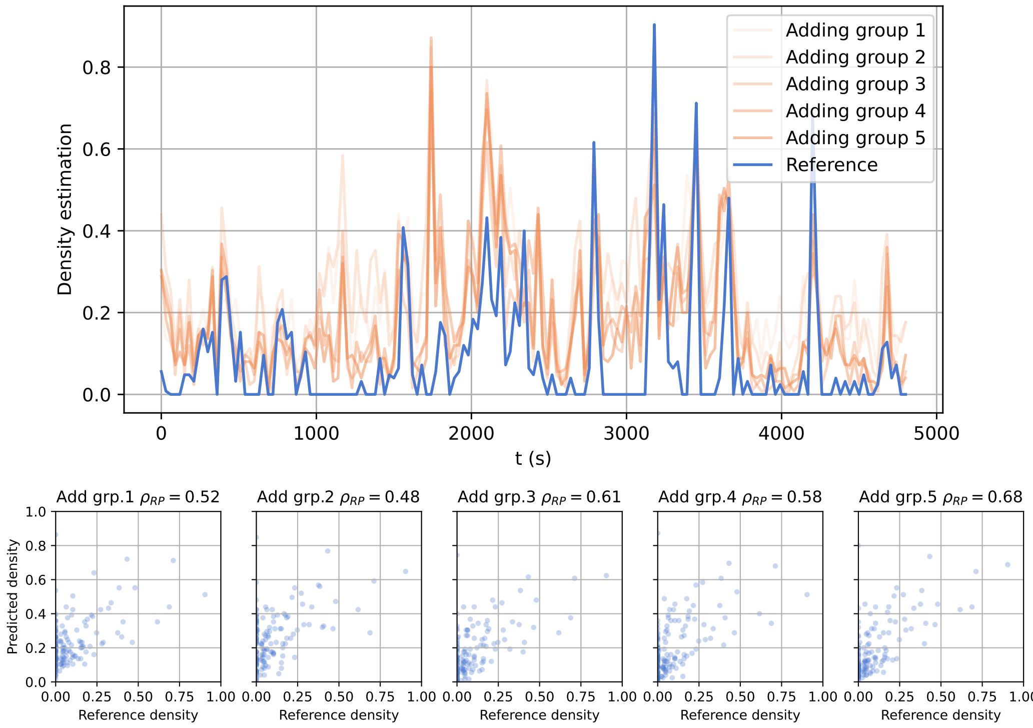


Figure 5.20: Improvement in predictions for label *a* over **test group 5**, while adding training groups 1 to 5. **Top:** Predicted and References densities. **Bottom:** correlation plots and coefficient values (see plot titles) for the same predictions shown on the top image.

different from the test splits where it is going to be evaluated.

So, the models are trained using always all the groups, but only some splits. As a first step, the model is trained using only split 1 from all groups. Then it is evaluated on the test split 0 from all groups separately. Another version of the model is trained using only split 2, and yet another using only split 3. All are tested on the same five test splits, so we get $3 \times 5 = 15$ data points to measure the prediction performance. These measurements are averaged and used to build a confidence interval at the first data point (*Number of training splits: 1*) in Figure 5.21.

The same procedure is repeated using only two splits (2 versions are trained: using only splits [1, 2] and using [4, 5]), and so on up to including all the 5 splits. The result is shown in all the other values of Figure 5.21.

Unfortunately, by observing that figure, it is seen that except for the label *a*, the other labels seem to reach plateau performance even after adding only one training split. There must be a point at which the supervised model is worse than the unsupervised one, but finding it would require even less training data than the minimum size of splits used in Section 3.1.3.

Note that it is a challenging problem to create smaller splits keeping the label balance in the data. Currently, each training split is created using only 10% of each lesson's audio (test split 0 takes 50% of each lesson, and the remaining is divided into five training splits), and the proportion between labels in those segments varies widely (as shown in Figure 3.3 of that section).

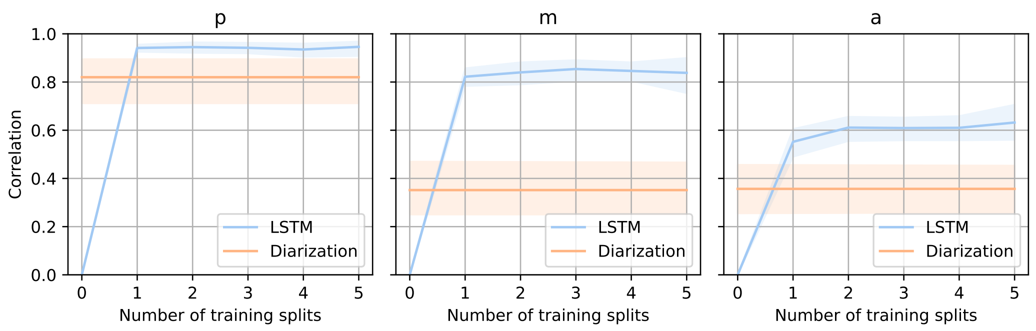


Figure 5.21: Comparison -in terms of correlation between predicted vs. reference densities- between the unsupervised (Diarization) system vs. the supervised (LSTM) model, while adding more labeled time in the same groups of audios (i.e: more time from the same lessons and teachers) to the supervised model. The colored spans around each curve represent the confidence interval of 95% of each metric, as measured on all 5 test groups.

In summary, we are unable to find the point at which the supervised model is worse than the unsupervised one, but these results suggest that labeling 10% of each lesson is enough to train the best possible version of the model, at least with the current features and architecture.

5.5 Chapter Summary

In this chapter, we have conducted a thorough comparison of all implemented models, focusing on their ability to accurately localize classroom activities in time, as evidenced by the label density estimation plots. Additionally, we extracted global metrics, such as Teacher Talking Time, to understand the overall performance of each model.

To establish a baseline for our comparisons, the LSTM model was used as a reference point and the results of the other approaches were evaluated in comparison to it. This analysis enabled us to grasp the strengths and limitations of each model, offering valuable insights for further refinement. Additionally, since both supervised classification models turned out to be similar in terms of detection performance, they were also benchmarked to highlight important differences in their inference speeds.

Finally, since supervised classification demonstrated considerably better detection performance compared to the unsupervised diarization system, we also conducted comparisons by incrementally adding more data to the supervised approach. By considering two distinct scenarios, one involving different lessons with diverse teacher and student voices, and the other focusing on adding more annotated audio time from the same lessons, we gained valuable understanding about the generalization power and the impact of labeled data on the supervised model's performance.

Chapter 6

Conclusions

6.1 Main Findings

The first thing that this study suggests is that performing Classroom Activity Detection (CAD) for primary school lessons poses greater challenges compared to college-level courses. Fundamental differences in class organization, fragmented student participation, short speaker turns, and high levels of noise and overlap from other students make it more complex. It is worth noting that the existing literature on CAD primarily focuses on college-level courses, making direct comparisons difficult and hardly relevant.

From the supervised models proposed, both the XGBoost and LSTM models demonstrate similar classification performance, despite the inherent lack of a built-in mechanism for sequence classification in XGBoost. To overcome this limitation, a context window was used, which extends the input vector by adding features from neighboring frames. But it is important to consider that the context cannot be excessively large in this approach due to the associated high computational requirements. In this sense, the LSTM is more flexible for sequence classification.

In terms of unsupervised audio diarization, the current heuristic which assigns the most frequent speaker as the teacher achieves a satisfactory performance in distinguishing his voice from the students'. Notably, this method does not require specific training data, but relies on the more general application for which it has been pre-trained. Nevertheless, even with a limited amount of training samples, supervised classifiers outperform the diarization method, even when faced with new voices from teachers and students that were not part of the training set.

By dividing the training data into subgroups, the performance of the LSTM could be studied as a function of the available data. This analysis was done in two ways: by adding new voices and by adding more time from the same voices. It was observed that the supervised model generalizes when faced with previously unseen voices, to the point that adding one single training group -5 lessons- is enough to attain the diarization system's performance. Furthermore, the study revealed that augmenting the annotated time from the same lessons proves effective only when focusing on segments with significant student participation. In contrast, the classification performance for the teacher's voice plateaus after incorporating the first split of data.

The complexity of diarization extends beyond what can be gleaned from research papers alone, in general. Various subtleties, such as the aggregation of overlapping embeddings and output clusters into segments, are often omitted in the literature. By dismantling the pipeline implemented in the library *pyannote.audio*, many of these details came to light. This is an important reason to prefer open source solutions, which can also be further adapted to the application needs.

One limitation of the diarization approach is the incapability to distinguish labels a from m (single students vs. group work and background noise). Using a speaker segmentation model to identify multiple voices or voice overlap does not work properly, since the voices of students engaged in group work often constitute background noise, lacking clear definition and not necessarily overlapping continuously.

With the current implementation, the LSTM model exhibits nearly the same speed as the feature extraction step, thanks to GPU utilization. On the other hand, XGBoost is slower but still reasonably fast, taking less than a minute for inference execution on our hardware. The advantage of XGBoost lies in its independence from GPU requirements, so it can be considered as an alternative solution if the available hardware is limited in that way.

The utilization of density estimation for predicted labels proves to be an effective means of summarizing results concisely. This is one important difference with respect to other CAD systems, in the way that results are shown. The usual plot that shows a plain timeline with student or teacher segments, is too congested with short segments in this case. Visualizing the density estimation proved useful to understand the results easier for this use case.

On top of the density estimation, the correlation coefficient allows to evaluate the prediction in terms of the information provided to an observer, that needs to quickly understand the key moments of the lesson. This metric evaluates that the shapes of the curves (peaks and valleys) are similar, but it is invariant to systematic biases (i.e., vertical shifts) in detecting any particular label.

To complement the correlation coefficient, the absolute error (MAE) and total label times are good complimentary metrics to consider. In that sense, the total times for labels a and m were not correctly estimated at an individual level -they are also hard to disambiguate manually-, but combining them or measuring the total time for label p led to very precise estimations of the Teacher Talking Time (TTT), which is widely employed in classroom activity analysis.

6.2 Review of Research Questions

Aside of the implementation and evaluation of different models, this study also posed some research questions (see Section 1.2) to answer during the process. Let us review them and verify if they were properly answered.

- *What useful information can be extracted by using an out-of-the-box diarization system over the classroom audio?*

  As was shown in Section 2.4 and mentioned above, the diarization system allows to distinguish the teacher's voice from other kind of participation, but it is less precise than the supervised classifiers and it is not good at refining those labels any further.

- *What's the human effort required and the best annotation criteria to create training and evaluation datasets for our task?*

  This question was mainly answered in Section 3.1 combined with the analysis on how the LSTM works with different amounts of data, in Section 5.4. About the annotation criteria, it was hard to disambiguate in practice between labels a and m, but the classifiers were able to distinguish them to some degree, although it is the main source of confusion in the classified samples. The usefulness of providing those labels separately could be further discussed with the final users.

- *How to compare diarization and audio classification systems, in a way that is significant for the actual problem that we are trying to solve?*

  There were several iterations to find a proper visualization and a performance metric that reflects the value of the information provided by the system. As shown in Section 3.2, using the label density estimations and correlation coefficient was considered the most effective way to evaluate the results.

- *Is the supervised approach able to generalize to new recorded lessons? How much human-annotated data is needed to make it comparable to the unsupervised approach?*

  In Section 5.4 it was clear that the supervised LSTM generalizes to new voices, working better than the diarization system even with reduced data (10% of the dataset), as was already mentioned above. The amount of data needed to reach or surpass diarization performance is less than the smallest subgroup of data that could be evaluated -it is hard to create even smaller subgroups maintaining label balance-.

6.3 Contributions

This research makes some contributions to the field of CAD using audio-based methods. These contributions are summarized as follows:

- An audio annotation protocol that enabled the creation of a data set for the training and evaluation of CAD models. Although the dataset cannot be released to the general public, the labeling criteria and effort estimation were described. The challenges and ambiguities involved in the process were also discussed by comparing redundant human annotations in some audio fragments.
- A comprehensive comparison between unsupervised diarization and supervised audio classification methods was conducted over the annotated dataset. This comparison is particularly relevant as unsupervised methods do not require annotated training data.
- A simple heuristic was proposed to use diarization outputs as a basic CAD pipeline. Notably, the experiments demonstrated that this approach performed well in identifying the teacher's voice (label p) but was very limited in accurately distinguishing between labels m (multiple background voices, group work) and a (student intervention).
- A data partitioning approach was proposed to create balanced subgroups for incremental training of the supervised model. This approach enabled a systematic exploration of how model performance improves with increased data and facilitated the estimation of its generalization capabilities to new voices. The experiments revealed that the supervised classifier demonstrates an effective generalization to new lessons after incorporating only five different lessons. Moreover, the experiments provided insights into where to prioritize data annotation efforts, as label a consistently improved while labels p and m exhibited limited improvement with increasing data from the same lessons.
- A visualization and evaluation method based on label density estimation and correlation was introduced to address the challenge of handling fragmented interventions and detections. This approach differs from conventional CAD systems intended for college-level courses and offers improved insight into the analysis of primary school lessons.
- An efficient classifier based on Long Short-Term Memory (LSTM) networks and classical audio features was implemented, which proved to be effective for the task at hand. Additionally, an alternative classifier that uses decision trees (XGBoost) was developed, which does not require specialized hardware such as a Graphics Processing Unit (GPU). Despite its slower inference speed and limitation on the maximum sequence length to process, the XGBoost classifier achieved comparable results.

These contributions collectively enhance our understanding of CAD in educational settings and provide insights into the performance and limitations of different methods. The findings of this research may be helpful for developing more robust and accurate CAD systems, especially in primary school environments characterized by unique challenges.

6.4 Future Work

While this research provides some insights into Classroom Activity Detection using audio-based features, its scope was limited, so there are inherent limitations that should be acknowledged and several avenues for future exploration.

To begin with, while the dataset used in this study underwent manual annotations, the resources allocated for reviewing and refining these annotations were limited. Therefore, a critical area for future work lies in conducting thorough reviews of the annotations to address any ambiguities or inconsistencies. By refining the dataset and ensuring its accuracy, the overall performance and generalizability of the CAD system can be significantly enhanced.

Additionally, exploring more advanced audio feature extraction techniques should be a priority. The current study relies on basic audio features, which may limit the richness of information captured. By incorporating more sophisticated approaches, we can potentially improve classification performance and enhance the system's ability to accurately detect and classify classroom activities.

Further exploration of the architecture and hyperparameters of the models in use, should also be conducted. In particular for the LSTM, there are other approaches like *seqtoseq* classification and bidirectional evaluation, that were only considered during the analysis of results, and could lead to better results with relatively little effort.

Another aspect to consider is that the predictive models employed in this research are relatively basic, lacking for example the utilization of state-of-the-art transformer architectures, Time-Delay or Convolutional Neural Networks, and end-to-end models that operate on the waveform domain. It is important to consider that these complex models often require substantial amounts of data for effective training. However, techiques like transfer-learning, upsampling or data augmentation, and semi-supervised learning may allow fine-tuning bigger pre-trained models for this particular data.

In order to increase the amount of annotated data, an active learning loop could be implemented to foster continuous improvement of the CAD system. By actively engaging users and incorporating their feedback, the system can adapt to diverse classroom environments and variations in speech patterns. This iterative feedback loop will contribute to the system's performance enhancement and overall user satisfaction. Although the current models are not showing a great improvement of the performance as new data is added, this situation would probably change if much larger models with learnable features are used. In this sense, the current models would be a starting point to enable further iterations with more data and larger models.

Self-supervised learning techniques offer promising avenues for further investigation. These approaches do not require manual annotations and could be trained on all available recordings, in order to learn audio representations particularly fine-tuned for speaker recognition in the domain of primary school lessons. This in turn could be used as input to larger models with more discrimination power.

Efforts should also be made to explore combined approaches that tackle both diarization and transcription simultaneously. Although challenging -due to the different embeddings required for each task, as mentioned in the introduction chapter-, the current research trends are focused towards developing generalistic foundation models trained on vast amounts of data. These pre-trained models can be then prompt engineered or fine-tuned to solve specific tasks, as has been the case with natural language and image generation models [43], and even object detection [47]. If such a model reaches human-perception level of audio signals, it should be able to distinguish and transcribe speaker voices with great accuracy, for different speaker ages, languages and even in noisy or overlapping conditions. At the moment, models like Whisper by OpenAI [45] -trained on vast amounts of data and is intended to reach human-level understanding- are intended for transcription and do not provide speaker diarization or prompting capabilities.

In conclusion, future work in CAD should focus on incorporating advanced audio feature extraction techniques, leveraging more complex predictive models, thoroughly reviewing and refining annotations, exploring transformer-based and self-supervised learning approaches, investigating generalistic models for audio, and implementing an active learning loop to ensure continuous system improvement.

6.5 Personal Reflections

This thesis was part of a larger project that aimed to develop useful tools for Ceibal evaluators. Thanks to the ANII scholarship, I had the opportunity to dedicate full-time periods to research and development of one of the components of this toolkit, the Classroom Activity Detection module. Although it may have taken longer than expected, the journey itself was incredibly enriching. Aside from all the technical challenges and learning-by-doing, envisioning this work as a real-world product that could potentially be implemented one day was truly exciting.

Coming from an industry experience background, at certain point of the development process I had to change gears to "academy mode". This transition involved stepping away for a bit from software implementation and model fine-tuning and instead focusing on crafting a coherent narrative with my findings and results. This shift required a different skill set, especially in terms of academic writing, in which I had very limited experience. Once again, the support and guidance of my mentors were instrumental during this phase. They provided valuable advice and helped me navigate the process of transforming my work into an academic article. Additionally, they helped to find opportunities to present my research, which shifted my focus towards engaging with the academic community.

Throughout this experience, I had the freedom and time to explore academic research and experiment with different approaches, taking the time to better understand the underlying principles, which differs from the fast-paced nature of some industry projects. This self-guided journey also differed from traditional university courses, as it required me to navigate uncertain territories where the amount of time and effort required for each task was not always clear and depended to a great extent on my own interests and judgement.

In general, this thesis provided a valuable opportunity for personal growth and development. It allowed me to delve into academic research, refine my writing skills, and gain a deeper understanding of various engineering topics. I am deeply thankful to my mentors and project teammates for their invaluable help and support, and for making this journey possible.

Bibliography

[1] H. W. Kuhn. “The Hungarian method for the assignment problem”. In: *Naval Research Logistics Quarterly* 2.1 (Mar. 1955), pp. 83–97. ISSN: 00281441, 19319193. DOI: 10.1002/nav.3800020109. URL: https://onlinelibrary.wiley.com/doi/10.1002/nav.3800020109 (visited on 04/25/2023).

[2] S. Davis and P. Mermelstein. “Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28.4 (1980), pp. 357–366. DOI: 10.1109/TASSP.1980.1163420.

[3] Leo Breiman. *Classification and Regression Trees.* (The Wadsworth statistics / probability series). Wadsworth International Group, 1984. ISBN: 9780534980535. URL: https://books.google.com.uy/books?id=uxPvAAAAMAAJ.

[4] C. Spearman. “The Proof and Measurement of Association between Two Things”. In: *The American Journal of Psychology* 100.3/4 (1987), pp. 441–471. ISSN: 00029556. URL: http://www.jstor.org/stable/1422689 (visited on 05/24/2023).

[5] Spyros Makridakis. “Accuracy measures: theoretical and practical concerns”. In: *International Journal of Forecasting* 9.4 (Dec. 1993), pp. 527–529. ISSN: 01692070. DOI: 10.1016/0169-2070(93)90079-3. URL: https://linkinghub.elsevier.com/retrieve/pii/0169207093900793 (visited on 05/14/2023).

[6] Leon Cohen. *Time-frequency analysis.* Prentice Hall signal processing series. Englewood Cliffs, N.J: Prentice Hall PTR, 1995. ISBN: 978-0-13-594532-2.

[7] Alexandra Canavan, David Graff, and George Zipperlen. *CALLHOME American English Speech.* Type: dataset. 1997. DOI: 10.35111/EXQ3-X930. URL: https://catalog.ldc.upenn.edu/LDC97S42 (visited on 04/20/2023).

[8] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (Nov. 1, 1997), pp. 1735–1780. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco.1997.9.8.1735. URL: https://direct.mit.edu/neco/article/9/8/1735-1780/6109 (visited on 04/26/2023).

[9] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. “Learning to Forget: Continual Prediction with LSTM”. In: *Neural Computation* 12.10 (Oct. 1, 2000), pp. 2451–2471. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/089976600300015015. URL: https://direct.mit.edu/neco/article/12/10/2451-2471/6415 (visited on 04/26/2023).

[10] Jerome H. Friedman. “Greedy function approximation: A gradient boosting machine.” In: *The Annals of Statistics* 29.5 (Oct. 2001), pp. 1189–1232. ISSN: 0090-5364, 2168-8966. DOI: 10.1214/aos/1013203451. URL: https://projecteuclid.org/journals/annals-of-statistics/volume-29/issue-5/Greedy-function-approximation-A-gradient-boosting-machine/10.1214/aos/1013203451.full (visited on 04/27/2023).

[11] S. Dubnov. "Generalization of Spectral Flatness Measure for Non-Gaussian Linear Processes". In: *IEEE Signal Processing Letters* 11.8 (Aug. 2004), pp. 698–701. ISSN: 1070-9908. DOI: 10.1109/LSP.2004.831663. URL: http://ieeexplore.ieee.org/document/1316889/ (visited on 04/26/2023).

[12] Ulrike von Luxburg. *A Tutorial on Spectral Clustering*. arXiv.org. Nov. 1, 2007. URL: https://arxiv.org/abs/0711.0189v1 (visited on 04/25/2023).

[13] Lawrence R. Rabiner and Ronald W. Schafer. *Theory and applications of digital speech processing*. 1st ed. Pearson, 2011. ISBN: 9780136034285.

[14] Hao Tang et al. "Partially Supervised Speaker Clustering". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.5 (May 2012), pp. 959–971. ISSN: 0162-8828, 2160-9292. DOI: 10.1109/TPAMI.2011.174. URL: http://ieeexplore.ieee.org/document/5989833/ (visited on 07/15/2022).

[15] Haşim Sak, Andrew Senior, and Françoise Beaufays. *Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition*. Feb. 5, 2014. DOI: 10.48550/arXiv.1402.1128. arXiv: 1402.1128[cs,stat]. URL: http://arxiv.org/abs/1402.1128 (visited on 04/27/2023).

[16] Zuowei Wang et al. "Automatic classification of activities in classroom discourse". en. In: *Computers & Education* 78 (Sept. 2014), pp. 115–123. ISSN: 0360-1315. DOI: 10.1016/j.compedu.2014.05.010. URL: https://www.sciencedirect.com/science/article/pii/S0360131514001328 (visited on 06/16/2023).

[17] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. "An empirical exploration of recurrent network architectures". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France: JMLR.org, July 6, 2015, pp. 2342–2350. (Visited on 04/26/2023).

[18] Christopher Olah. *Understanding LSTM Networks – colah's blog*. Aug. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 04/27/2023).

[19] Etto Salomons and Paul Havinga. "A Survey on the Feasibility of Sound Classification on Wireless Sensor Nodes". In: *Sensors* 15 (Apr. 2015), pp. 7462–7498. DOI: 10.3390/s150407462.

[20] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: http://doi.acm.org/10.1145/2939672.2939785.

[21] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016. ISBN: 9780262035613. URL: https://books.google.com.uy/books?id=Np9SDQAAQBAJ.

[22] Sungil Kim and Heeyoung Kim. "A new metric of absolute percentage error for intermittent demand forecasts". In: *International Journal of Forecasting* 32.3 (2016), pp. 669–679. ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2015.12.003. URL: https://www.sciencedirect.com/science/article/pii/S0169207016000121.

[23] Frank Nielsen. "Hierarchical Clustering". In: *Introduction to HPC with MPI for Data Science*. Cham: Springer International Publishing, 2016, pp. 195–211. ISBN: 9783319219028 9783319219035. DOI: 10.1007/978-3-319-21903-5_8. URL: http://link.springer.com/10.1007/978-3-319-21903-5_8 (visited on 06/19/2023).

[24] Hervé Bredin. "pyannote.metrics: a toolkit for reproducible evaluation, diagnostic, and error analysis of speaker diarization systems". In: *Interspeech 2017, 18th Annual Con-*

ference of the International Speech Communication Association. Stockholm, Sweden, Aug. 2017. URL: `http://pyannote.github.io/pyannote-metrics`.

[25] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. arXiv:1412.6980 [cs]. Jan. 2017. DOI: `10.48550/arXiv.1412.6980`. URL: `http://arxiv.org/abs/1412.6980` (visited on 05/09/2023).

[26] Melinda T. Owens et al. "Classroom sound can be used to classify teaching practices in college science courses". In: *Proceedings of the National Academy of Sciences* 114.12 (Mar. 21, 2017), pp. 3085–3090. ISSN: 0027-8424, 1091-6490. DOI: `10.1073/pnas.1618693114`. URL: `https://pnas.org/doi/full/10.1073/pnas.1618693114` (visited on 06/15/2023).

[27] Church et al. *DIHARD Challenge*. 2018. URL: `https://dihardchallenge.github.io/dihard1/overview.html` (visited on 07/19/2022).

[28] Quan Wang. *Google's Diarization System: Speaker Diarization with LSTM*. ICASSP 2018, 2018. URL: `https://www.youtube.com/watch?v=pjxGPZQeeO4`.

[29] Robin Cosbey, Allison Wusterbarth, and Brian Hutchinson. "Deep Learning for Classroom Activity Detection from Audio". In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Brighton, United Kingdom: IEEE, May 2019, pp. 3727–3731. ISBN: 9781479981311. DOI: `10.1109/ICASSP.2019.8683365`. URL: `https://ieeexplore.ieee.org/document/8683365/` (visited on 06/16/2023).

[30] William Falcon and The PyTorch Lightning team. *PyTorch Lightning*. Version 1.4. Mar. 2019. DOI: `10.5281/zenodo.3828935`. URL: `https://github.com/Lightning-AI/lightning`.

[31] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[32] Neville Ryant et al. *The Second DIHARD Diarization Challenge: Dataset, task, and baselines*. arXiv.org. June 18, 2019. URL: `https://arxiv.org/abs/1906.07839v1` (visited on 04/20/2023).

[33] Hervé Bredin et al. "pyannote.audio: neural building blocks for speaker diarization". In: *ICASSP 2020, IEEE International Conference on Acoustics, Speech, and Signal Processing*. 2020.

[34] Joon Son Chung et al. *Spot the conversation: speaker diarisation in the wild*. arXiv.org. July 2, 2020. DOI: `10.21437/Interspeech.2020-2337`. URL: `https://arxiv.org/abs/2007.01216v3` (visited on 04/20/2023).

[35] Brecht Desplanques, Jenthe Thienpondt, and Kris Demuynck. "ECAPA-TDNN: Emphasized Channel Attention, Propagation and Aggregation in TDNN Based Speaker Verification". In: *Interspeech 2020*. Oct. 25, 2020, pp. 3830–3834. DOI: `10.21437/Interspeech.2020-2650`. arXiv: `2005.07143[cs,eess]`. URL: `http://arxiv.org/abs/2005.07143` (visited on 04/25/2023).

[36] Hang Li et al. *Siamese Neural Networks for Class Activity Detection*. May 15, 2020. DOI: `10.48550/arXiv.2005.07549`. arXiv: `2005.07549[cs,eess]`. URL: `http://arxiv.org/abs/2005.07549` (visited on 06/15/2023).

[37] Hervé Bredin and Antoine Laurent. "End-to-end speaker segmentation for overlap-aware resegmentation". In: *Proc. Interspeech 2021*. 2021.

[38] Brown et al. *VoxCeleb Speaker Recognition Challenge*. 2021. URL: `https://www.robots.ox.ac.uk/~vgg/data/voxceleb/competition2021.html` (visited on 07/19/2022).

[39] James Gareth et al. *An Introduction to Statistical Learning*. 2nd ed. Springer, 2021. URL: `https://www.statlearning.com` (visited on 04/27/2023).

[40] Mirco Ravanelli et al. *SpeechBrain: A General-Purpose Speech Toolkit*. Tech. rep. arXiv:2106.04624. arXiv:2106.04624 [cs, eess] type: article. arXiv, June 2021. URL: `http://arxiv.org/abs/2106.04624` (visited on 07/15/2022).

[41] Eric Slyman et al. *Fine-Grained Classroom Activity Detection from Audio with Neural Networks*. Nov. 9, 2021. DOI: `10.48550/arXiv.2107.14369`. arXiv: `2107.14369[cs, eess]`. URL: `http://arxiv.org/abs/2107.14369` (visited on 06/15/2023).

[42] Neil Zeghidour et al. *LEAF: A Learnable Frontend for Audio Classification*. Jan. 21, 2021. DOI: `10.48550/arXiv.2101.08596`. arXiv: `2101.08596[cs,eess]`. URL: `http://arxiv.org/abs/2101.08596` (visited on 04/26/2023).

[43] Rishi Bommasani et al. *On the Opportunities and Risks of Foundation Models*. July 12, 2022. DOI: `10.48550/arXiv.2108.07258`. arXiv: `2108.07258[cs]`. URL: `http://arxiv.org/abs/2108.07258` (visited on 06/23/2023).

[44] Andrew Brown et al. *VoxSRC 2021: The Third VoxCeleb Speaker Recognition Challenge*. arXiv:2201.04583 [cs, eess]. Nov. 2022. DOI: `10.48550/arXiv.2201.04583`. URL: `http://arxiv.org/abs/2201.04583` (visited on 06/19/2023).

[45] Alec Radford et al. *Robust Speech Recognition via Large-Scale Weak Supervision*. Dec. 6, 2022. DOI: `10.48550/arXiv.2212.04356`. arXiv: `2212.04356[cs,eess]`. URL: `http://arxiv.org/abs/2212.04356` (visited on 04/20/2023).

[46] Yao-Yuan Yang et al. *TorchAudio: Building Blocks for Audio and Speech Processing*. Tech. rep. arXiv:2110.15018. arXiv:2110.15018 [cs, eess] type: article. arXiv, Feb. 2022. URL: `http://arxiv.org/abs/2110.15018` (visited on 07/15/2022).

[47] Alexander Kirillov et al. *Segment Anything*. Apr. 5, 2023. DOI: `10.48550/arXiv.2304.02643`. arXiv: `2304.02643[cs]`. URL: `http://arxiv.org/abs/2304.02643` (visited on 06/23/2023).

[48] McFee, Brian et al. *librosa/librosa: 0.10.0.post2*. Version 0.10.0.post2. Mar. 17, 2023. DOI: `10.5281/ZENODO.7746972`. URL: `https://zenodo.org/record/7746972` (visited on 04/26/2023).