Speaker recognition implemented as part of a video-editing system

Author: Sebastian Rojas Tveita

Supervisors:
Gisle Sælensminde (external)
Harald Soleim
Atle Birger Geitung
Daniel Patel

Master's thesis in Software Engineering at

Department of Graphics,
Bergen University College

Department of Informatics,
University of Bergen

June 2019

Western Norway
University of
Applied Sciences

Abstract

With machine learning rising to prominence over the last decades, a lot of companies are doing research on how it can be applied in their products or production. Some of these companies have used machine learning with a good deal of success. This thesis proposes a solution for integrating speaker recognition in a video-editing system. The proposed solution is a proof-of-concept pipeline that is hooked into a web-based video-editing system made by a company called Vizrt[1]. This pipeline takes a video and performs speaker diarization and classification on the audio from the video. To achieve this, two types of models are applied; Gaussian Mixture Models to create a Universal Background Model, and models applying the i-vector approach for use in the clustering. From the results of the machine learning algorithms, the pipeline will produce timecodes that are sent to the video-editing system. These timecodes show where different speakers are talking. This information will be presented to the user in the system UI, where the user will have the option of correcting the results from the diarization and classification.

The pipeline also adopts the results from the algorithms to provide further functionality. By using the generated timecodes, the pipeline is able to extract training data that is partitioned according to the speakers. This training data will be saved and can later be used to generate new models for the different speakers. These new models can be used in later runs through the pipeline to recognize known speakers and be improved by gathering more training data for the known speakers.

The thesis shows how machine learning can be applied in a pipeline to partition an audio track without any prior trained model. Using this information could be time-saving in a video-editing process, or in the process of creating training data. The pipeline also has the potential to be expanded with further functionality. This would require the pipeline to be further integrated into the video-editing system.

Acknowledgements

I would like to thank my supervisors at The Western Norway University of Applied Sciences, Harald Soleim, Daniel Patel, and Atle Geitung, who has throughout the duration of the thesis provided me with excellent feedback, constructive criticism and support.

I would also like to thank my external supervisor Gisle Sælensminde at Vizrt, who went above and beyond in guiding me in the work of this thesis. Without his invaluable help and input this thesis would not have been possible.

Contents

List of Figures

List of Abbreviations

UBM	**U**niversal **B**ackground **M**odel
GMM	**G**aussian **M**ixture **M**odel
ML	**M**achine **L**earning
DCT	**D**iscrete **C**osine **T**ransform
UI	**U**ser **I**nterface
NLE	**N**on **L**inear **E**ditor
EDL	**E**diting **D**ecision **L**ist
XML	e**X**tensible **M**arkup **L**anguage
UI	**U**ser **I**nterface
WAV	**W**aveform **A**udio **F**ile
PoC	**P**roof of **C**oncept
PCA	**P**rinciple **C**omponent **A**nalysis
BIC	**B**ayecian **I**informaton **C**riterion
JFA	**J**oint **F**actor **A**nalysis
MAP	**M**aximum **a** **P**osteriori

Glossary

Bayesian information criterion In statistics, the Bayesian information criterion (BIC) is a criterion for model selection among a finite set of models.

Classification In machine learning classification is the problem of deciding which category a new observation belongs to.

Discrete Cosine Transform A discrete cosine transform (DCT) expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies..

Edit Decision List An edit decision list (EDL) is used in the post-production process of film editing and video editing. The list contains an ordered list of reel and timecode data representing where each video clip can be obtained in order to conform the final cut..

Feature A distinctive attribute or aspect of something.

Gaussian Mixture Model A Gaussian Mixture Model (GMM) is a parametric probability density function represented as a weighted sum of Gaussian component densities. It speaker recognition it can be used a model for a speaker or utterance.

I-vector An i-vector is a modeling approach used in speaker recognition to model a speaker or utterance. It exist in a low dimensional space called the total-variability space.

Lower third In the television industry, a lower third is a graphic overlay placed in the lower area of the screen.

Machine Learning Machine Learning is the science of getting computers to learn and act like humans do, and improve their learning over time in autonomous fashion, by feeding them data and information in the form of observations and real-world interactions.

Non Linear Editor A non-linear editing system (NLE) is a video or audio editing digital audio workstation system that performs non-destructive editing on source material. Non-destructive editing is a form of audio, video, or image editing in which the original content is not modified in the course of editing; instead the edits are specified and modified by specialized software..

Principal Component Analysis Principal component analysis (PCA) is a mathematical procedure that transforms a number of (possibly) correlated variables into a (smaller) number of uncorrelated variables called principal components.

Rendering In video-editing, rendering is the process of applying the editing steps found in the edit decision list to the video.

Speaker diarization In the field of speaker recognition, speaker diarization is the problem of finding out at what times the different speakers in a audio-track speaks..

Universal Background Model A Universal Background Model (UBM) is a model used in a biometric verification system to represent general, person-independent feature characteristics to be compared against a model of person-specific feature characteristics when making an accept or reject decision.

User Interface The means by which the user and a computer system interact, in particular the use of input devices and software..

Viz Story In statistics, the Bayesian information criterion (BIC) is a criterion for model selection among a finite set of models.

Vizrt Vizrt is a worldwide market leader in the areas of real-time 3D graphics, studio automation, sports analysis and asset management tools.

WAV Waveform Audio File Format is a Microsoft and IBM audio file format standard for storing an audio bitstream on PCs..

Chapter 1

Introduction

1.1 Motivation

Over the years, the use of machine learning has become increasingly prominent. Both bigger companies like Google and Amazon and smaller companies are devoting more time and resources to developing and implementing products that use machine learning. This has also lead to the development of different frameworks to make machine learning more accessible to the average developer. An example of such a framework would be TensorFlow, which simplifies the process of making, for example, neural networks. Likewise if you do not want to build a network for yourself there is a good chance you can find an API that will do the work for you. Because of this, you see the average developer being able to utilize machine learning without needing a lot of prior knowledge or spending a lot of time building their models.

This rise in popularity has also lead to companies looking into automating or simplifying time-consuming tasks normally has to be done manually, Vizrt being one of them. Vizrt is a worldwide market leader in the areas of real-time 3D graphics, studio automation, sports analysis, and asset management tools for the media and entertainment industry[1].

This thesis is a cooperative project between Vizrt and HVL/UiB. The motivation for Vizrt was to look into a way to use machine learning in their editing software in general, but more precisely they wanted to use machine learning to extract some kind of metadata from the content in their editing software. Over time the problem was adjusted and got narrowed down to looking into extracting metadata related to the audio of videos. From this, it was decided that the thesis should focus on getting metadata directly related to tagging the video so that it can be used as training data in speaker recognition.

The problem that often arises when using machine learning is that gathering enough training data can be both hard and time-consuming. Often you will be able to find enough data to train your models online, but most of the time you are forced to spend a lot of time tagging that data.

When discussing the thesis, the fact that machine learning algorithms rarely will give consistently perfect results another fact was brought up, namely the fact t. Some of the users of machine learning have a tendency to be too trusting of the results they get from their algorithm. Even if it is understood in the ML field that the result will be an approximation of reality. In most cases, this is not a problem, as people do not tend to use ML to do tasks that have to be 100% correct. Even if this was also the case with this thesis, the advantage of being able to edit the results given from an algorithm could not be overlooked.

1.2 Goal of this thesis

The goal of this thesis is to look at the possibility of creating a pipeline that automates the process of extracting training data, which can then be used in the training of models through a pipeline, while also having the possibility of editing the data before using them. After extracting information from a video, the results will be presented to the user. If the algorithm classifies a segment as containing a certain speaker, the user should be able to change the classification to another speaker before the data is saved or used to edit a video. Also, to answer the research questions related to this.

Specifically, the thesis will focus on extracting data to use in speaker recognition and looking into some other uses for the extracted data. The result from this will hopefully be either complete training data that you can give to a training algorithm or some sort of metadata that will make the extraction and tagging of the data simpler and less time-consuming. The model approaches used will be a combination of Gaussian Mixture Models and identity-vectors. These models will be used in combination with Vizrt's systems and a clustering algorithm, to hopefully be able to extract data that can be used as training data. When uploading a video to Vizrt's systems the goal is to be able to extract the data before publishing it or during the publishing process, so the pipeline will likely end up looking like figure 1.1. The figure shows a simplified overview of how the program could be used in a media company's system. When a video is uploaded to the system it would at some point be sent to the program

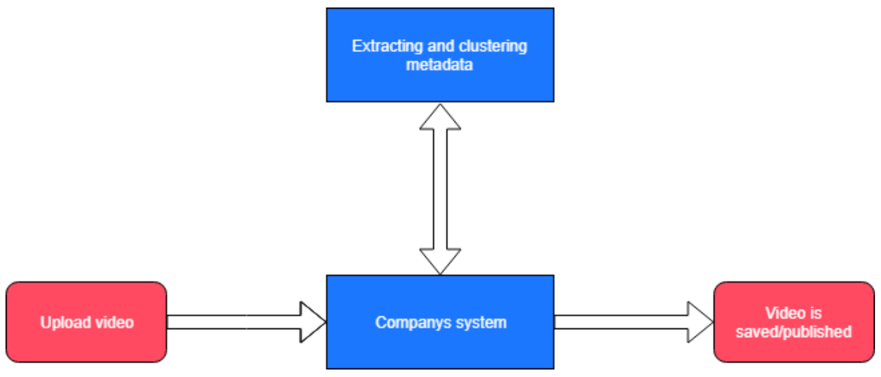


FIGURE 1.1: A diagram showing a very general idea of how the pipeline could work

where the desired information is extracted and then sent back to the system. The information can then be used in different ways like annotation and tagging, depending on the need of the company.

1.3 Related works

There has been done a lot of work within the field of both speaker recognition and speaker diarization. On top of that the field of machine learning, which include speaker recognition and diarization, has also made regular advancements. The work done in these fields goes back decades, and a lot of different techniques have been used through these decades with varying results [2]. Different models like Gaussian Mixture Models and modeling approaches like identity-vector has been used and advanced. At the same time, the field of machine learning has made significant advancements, like deep learning and frameworks for machine learning. These advancements let researchers more intuitively test different models with different techniques, all while improving how these models work. Today, big companies like Google have done a lot of advancements in these fields. They have also made a lot of APIs like TensorFlow that has made machine learning more readily available for people. Because of this thesis will focus on the case of using the existing techniques in the fields to add functionality to Vizrt's systems[2][3].

There have been made systems with similar functionality to the one that is being built in this thesis. Browsing and editing video by using the audio has been tried and patented earlier[4]. The system that has been looked at does not provide the exact functionality that this system hopefully will provide. Further, the disclosed patent does not provide information on the technologies used (model, algorithm, etc). Because of this, it is possible that the system built in this thesis could differ greatly from earlier systems.

1.4 Research questions

Question 1
In this work we aim to try to make a pipeline that will separate the voices of speakers in an audio-track by applying machine learning, and pipes this information into a system. By this, answer the question of; can machine learning be applied to partition an audio-track according to speakers without any prior trained models, and the result be piped into a system to train new speaker models?

Question 2
This thesis also aims to look into the possibilities of integrating machine learning into a video-editing system. Then answer the question of how a machine learning system be integrated into a video-editing system in a way that will help the users with time-consuming tasks?

1.5 Research method

In order to answer the research questions, I am going to do exploratory research where I integrate a speaker recognition system into an existing video-editing system. Using the UI of this editing system I will evaluate if integrating the machine learning system will help the users with simple yet time-consuming tasks, like tagging, or finding the correct time in a video. The evaluation will, for the most part, be qualitative and subjective in nature. Since concrete data about the time users spend on those tasks will not be available, but will also contain some concrete quantitative results since using the UI one can see if the result of the machine learning corresponds to reality. By looking at the results in the UI, we can see if the information gets piped correctly into Vizrt's systems. Also by checking if those results correspond with reality we can draw a conclusion about if the results are good enough to use as training data for new models. The information gathered will be used to conclude if the pipeline has the potential to be used in a production system, as well as what potential benefits one can expect from it.

1.6 Report outline

This thesis is divided into the following chapters

Chapter 2 - Background
This chapter is divided into two parts; signal processing and machine learning, each providing the necessary background information for the different fields. The signal processing section introduces the features used, namely MFCC, and the process of extracting them. The machine learning gives a presentation of the speaker recognition field and the different models used in this thesis.

Chapter 3 - Technologies and problem description
This chapter is divided into two parts. The first part presents a more detailed overview of the problem this thesis is aiming to solve. The second part presents the different techniques and technologies used and the motivation behind using them.

Chapter 4 - Design and implementation
This chapter presents the design and use of the proposed solution. As well as an overview of the work that went that was done in the implementation process.

Chapter 5 - Result
This chapter presents the result of this thesis. These will be divided into qualitative and quantitative results, and be presented separately.

Chapter 6 - Evaluation and conclusion
This chapter will discuss and evaluate the results presented in chapter 5. Additionally, the research question will be answered and a conclusion will be drawn. Finally, some future work for the proposed solution is given

Chapter 2

Background

This chapter will give a description of the underlying methods and theories used in this thesis. The reader will first get an introduction to signal processing and machine learning. Signal processing is in this context used to transform the data into a domain that the machine learning algorithm can use. The chapter will begin with giving an explanation of signal processing and the process used to transform the data. Then an explanation of machine learning and speaker recognition and how it uses the data acquired from the signal processing.

2.1 Signal processing

"A signal is a varying quantity whose value can be measured and which conveys information." This definition is pretty wide and can be used to describe a multitude of things. The kind of signals that are of interest to us in this thesis are digital signals, which are signals represented by numbers on a computer or digital hardware. More specifically, audio saved as a digital signal, which is called audio signals. These digital audio signals are usually saved as samples. These samples are taken at regular intervals. Each of these samples will contain numbers, where each number represents the relative voltage at that time. In other words, a sample is a measure of the signal at a certain time. We want to look at frequencies from different audio files and use that data in our algorithm. To do this we have to use some form of digital signal processing. The basic steps of digital signal processing can be seen in figure 2.1, even if this is not exactly the kind of processing done in this thesis. The transformation in this thesis involves using simple operations to transform the numbers representing the signals in one domain, into numbers representing the signal in another domain. In our case, we want to transform

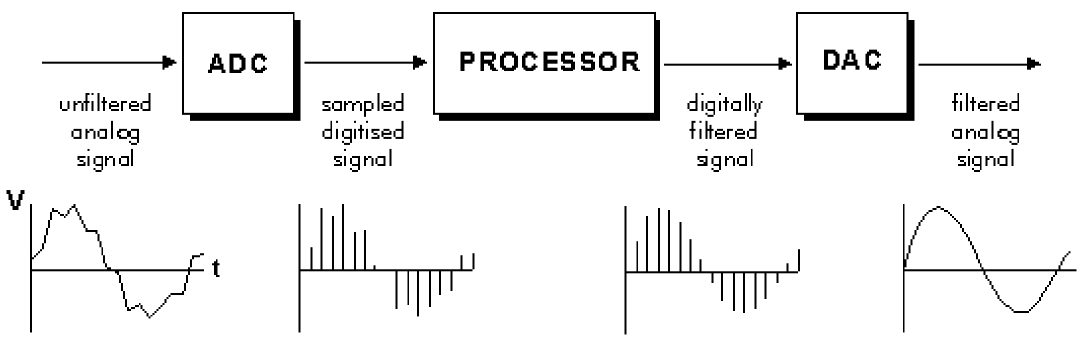


FIGURE 2.1: Block diagram showing the basic steps of digital signal processing [5]

the samples we got from our signal from the time domain into the frequency domain. In other words, we start by having an analysis of the data with respect to time, which we want to transform into an analysis with respect to the frequencies. This transformation is motivated by the human auditory system, this lets the computer model the way we humans interpret sound. How this transformation works and the motivation behind it will be explained later in this chapter.

2.1.1 Feature extraction

The reason you have to transform the data is that the raw samples of the audio will just look like noise to most machine learning algorithms. This is because, among other things, that data in the frequency domain is varies more slowly than data in the time domain. Since frequencies will change a lot over time, it will be observed as noise by the algorithm. Because of this, having the data in the frequency domain is necessary when working with audio in ML.

When doing speaker recognition (or any kind of machine learning tasks) you have to do feature extraction. This means identifying the components of the audio signal which are good for identifying the linguistic content and discard the part of the signal which contains information like background noise, emotions, etc. The first thing you have to do is decide on what feature you want your algorithm to base its decision on. This will naturally wary not only based on your choice of algorithm, but different kinds of machine learning tasks usually use different kinds of features. In for example machine learning tasks involving pictures, the features will often be the pixels of the pictures. This is a straight forward approach as the pixel data is pretty simple to use without changing it (sometimes you have to use gray-scaling etc, but that is also pretty straight forward). In speaker recognition, the best kinds of features are frequency-based features (the data is in the frequency domain). This will as mentioned keep the algorithm from observing the data as just noise. A common way to do this is to use a mel-frequency analysis to extract mel-frequency cepstrum coefficients(MFFC). The reason these features are common is that they are frequency based. Also, they mimic the human hearing, which is built to detect frequencies, not just the pressure changes[6].

2.1.2 Mel-frequency cepstrum

Mel-frequency analysis is based on human perception experiments Through different experiments, it has been observed that the human ear will concentrate more on certain parts of a frequency spectrum. So when doing a mel-frequency analysis you will end up with a representation of the spectrum that more closely mimics the way the human auditory system would perceive the frequency signal[7].

The representation you will end up with is called Mel Frequency Cepstral Coefficient, and as mentioned earlier is a feature that is widely used in speech recognition and speaker recognition. They were introduced in the 1980s and have been state-of-the-art ever since[8].

Mel-frequency analysis usually includes several steps as shown in figure 2.2. In a nutshell, the signal will get sliced into frames and each frame will have a window function applied to it. Afterward, we do a discrete fourier transform on each frame and then calculate the power spectrum. We then use the power spectrum to compute the mel-filter banks and take the logarithm of each filter bank. Finally, you use DCT to obtain the MFCC's[6]. Each of these steps and their motivations will be explained below.

Pre-emphasis

The first step in obtaining the MFCC's is to apply a pre-emphasis filter that amplifies the high frequencies. It is possible to obtain the MFCC's without using a pre-emphasis filter, but using it will have several benefits. It will balance the frequency spectrum since higher frequencies

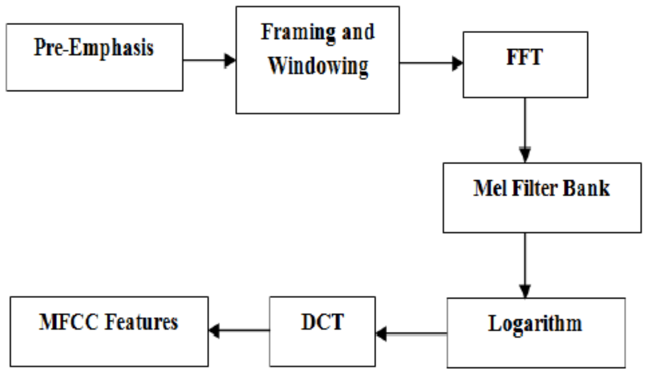


FIGURE 2.2: Block diagram of the mel-frequency analysis [9]

have a lower magnitude than lower frequencies, it will help you avoid numerical problems during the Fourier transform, and it will improve the signal-to-noise ratio.

The pre-emphasis filter can be applied to a signal x using this equation.

$$y(t) = x(t) - \alpha x(t-1) \tag{2.1}$$

Where α = filter coefficients, which are usually 0.95 or 0,95 and t = Time.

As mentioned you are able to obtain the MFCC's without using a pre-emphasis filter. The reason for this is that the filter only has a modest effect in a modern system. The signal-to-noise ratio can be improved with mean normalization, and the numerical issues with the Fourier transform should not be an issue in modern implementations of the Fourier transform[10]. It is still explained in this thesis as the package that is used for obtaining the MFCC's does use a pre-emphasis filter.

Framing and Windowing

When analyzing a frequency signal you will see that the signal is constantly changing over time. Therefore analyzing the whole signal at once does not make any sense, since we would lose the frequency contours over time. Instead, we assume the frequency is stationary over a shorter interval. Each of those intervals will be represented by a frame. A frame will consist of a number of continuous samples with the samples having a length given in time, often in milliseconds. These samples will often have a size of about 20-40 milliseconds. The frames will also have a set sampling rate. One can decide the sampling rate as long as it is high enough to pick up the relevant frequencies. For speech, this is 8000 Hz, but it is commonly set to 48000 Hz. This will give you 48 samples per millisecond. That will result in each frame having less change than the whole of the signal. Obviously, since the signal is constantly changing it is impossible to eliminate all the change even on shorter time-frames, but the change in each frame will be significantly reduced when compared to the entire signal. Finally, you use a window function on each of the frames by multiplying the frames with the window function. The function will multiply the frames with a factor close to 0 at the ends, and 1 one the middle of the frame. This will result in each frame being turned into a window where the signal evens out to close to 0 at the ends. In this routine the Hanning window function is used, where you get the hanning window w of length n by using equation 2.2 and N is the collection of samples[11].

$$w[n] = 0.54 - 0.46 * \cos(2\pi n/N - 1) \tag{2.2}$$

There are several reasons for doing this, but among them being to counteract the fact that the Fourier transform assumes the data infinite. Also when you even out the signals at the ends, you eliminate sharp transients between the windows [6]. A visual representation of applying a window function can be seen in figure 2.3, the effects this will have on the transitions can be seen in figure 2.4. The next step are then performed on each of the windows.

Fourier transform and power spectrum

In this step, the signal is transformed into the desired domain. At this stage, the signal will be in the time-domain, but we want to have our signal represented in the frequency domain. An example of this can be seen in figure 2.5. To achieve this we use a Fourier transformation. In this case, a variation of the Fourier transformation called a Discrete Fourier transformation is used. The Fourier transform uses a continuous signal while the Discrete Fourier transformation uses samples. In general, when doing a Fourier transformation on a signal you will decompose this signal into the frequency domain (in other words, take the signal from the time domain to the frequency domain). The reason we want to this is again to mimic the way the human auditory system works. The human ear has an organ called the cochlea that will vibrate depending on the frequencies of a signal. Depending on which part of the cochlea that vibrates it will send certain signals to the brain. The Fourier transform will do something similar to the signal and give us a representation

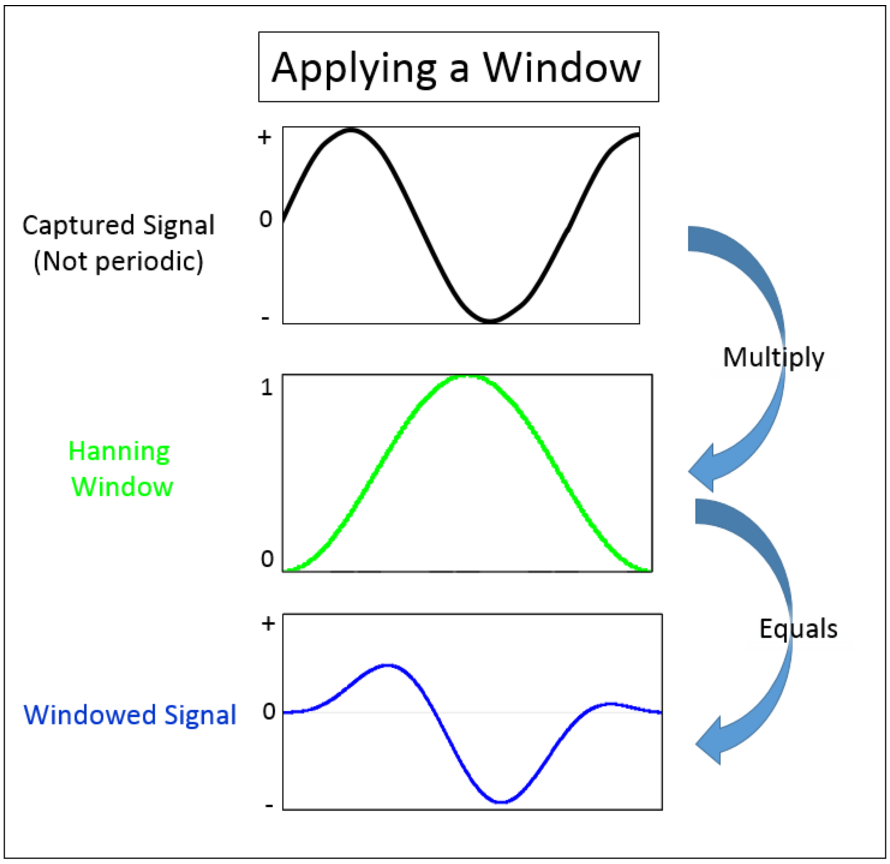


FIGURE 2.3: A signal (top) is multiplied by a window (middle) resulting in windowed signal (bottom) [11]

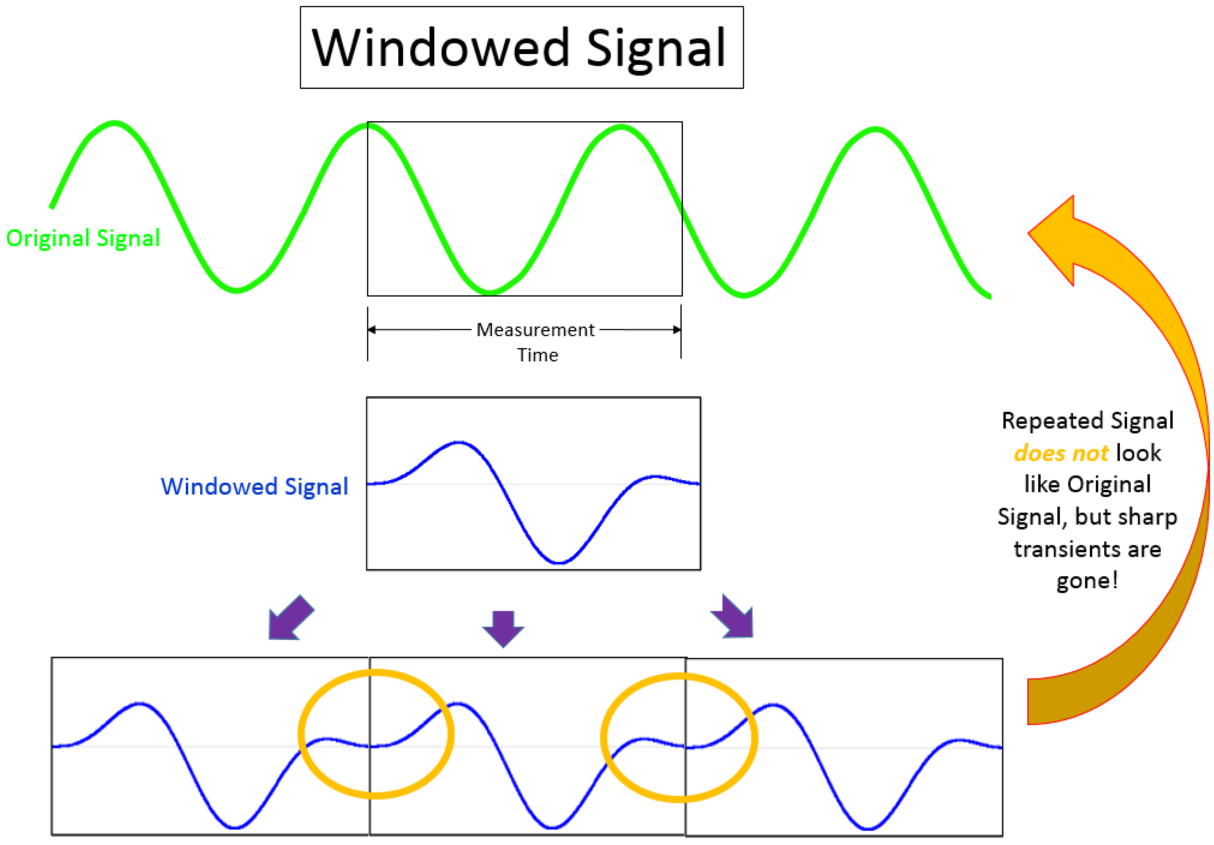

Windowed Signal
Original Signal
Measurement Time
Windowed Signal
Repeated Signal *does not* look like Original Signal, but sharp transients are gone!

FIGURE 2.4: After applying a window function to the captured signal, the sharp transients are eliminated [11]

of the signal that more closely resembles the way the brain interprets an auditory signal. The resulting data is called a frequency spectrum [12].

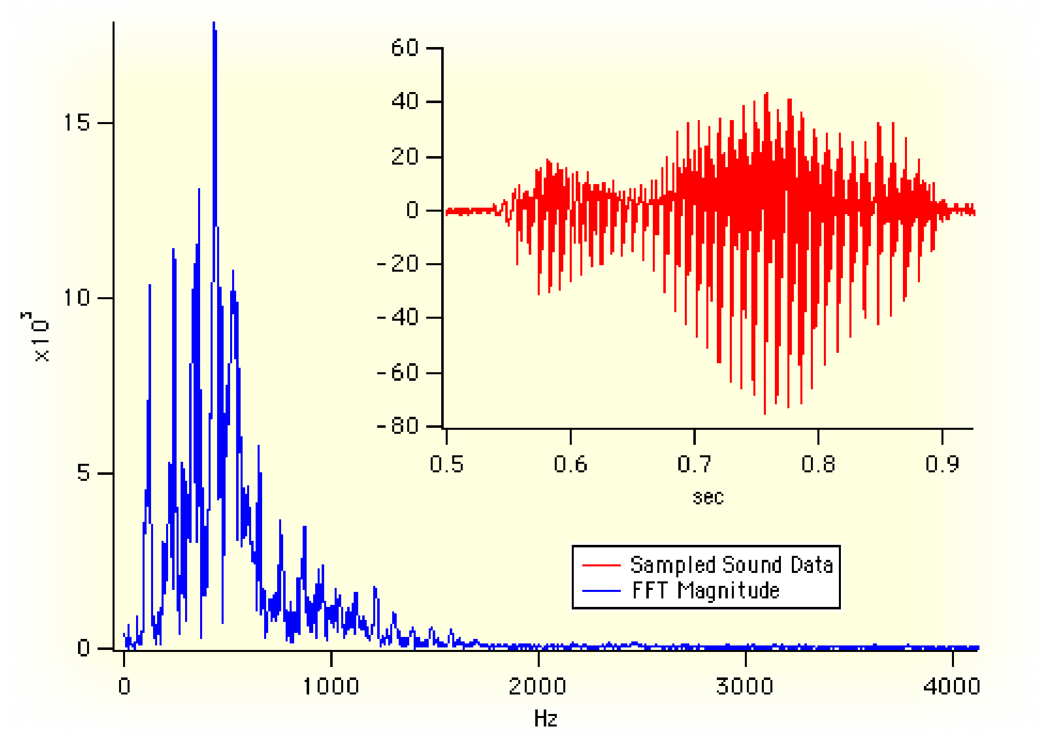


FIGURE 2.5: Example of a signal in the time-domain and the frequency-domain [9]

The Fourier transform will also give you some information that is not necessary for doing speech analysis. The values you get will be complex and contain information on both the magnitude and the phase of the frequencies. For speech analysis, we are only interested in the magnitude of the frequencies. This can be obtained by taking the absolute values of the complex values and squaring the result. Finally, you discard the second half of the values. The reason for this is that they are mirror values of the first half of the values. This is done on each of the frequency spectrum's and will result in a new spectrum called a power spectrum[12].

Mel-filter banks

Our power spectrum still has a lot of information that is not needed in speech analysis. Again this is because of how the human auditory system works, namely the ability or lack thereof to distinguish different frequencies. The cochlea lacks the ability to discern between two tones which have closely spaced frequencies. This effect will only become more pronounced as the frequencies increases. This means that

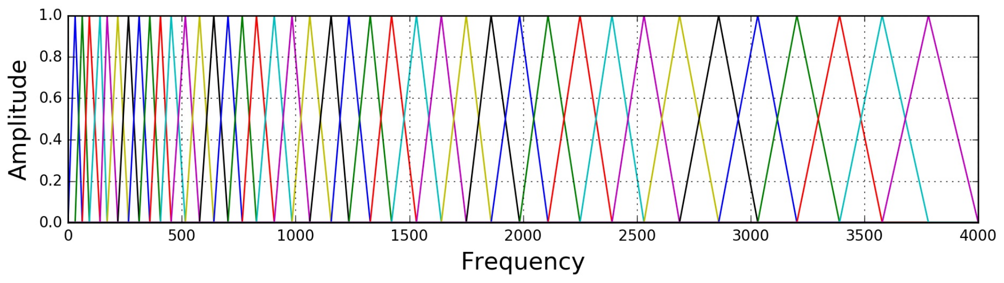


FIGURE 2.6: Filter bank on a Mel-Scale [6]

the human ear will not perceive much of the information given by the higher frequencies, and we can therefore disregard the corresponding data.

In an attempt to relate the way humans perceive frequencies to the actual frequency of a tone, the mel-scale was created. The mel-scale is linear up to about 1000Hz, and will be logarithmic after that[6]. The most common formula used for finding the mel-scale value for frequency f is:

$$M(f) = 1125.0 * ln(1.0 + f/700.0) \tag{2.3}$$

This formula is of course only used after 1000Hz since the mel scale is linear until that point. To get rid of the unnecessary information we want to calculate filter-banks, where the size of each bin is decided based on the mel-scale. These filter-banks will be in the form of vectors and are calculated based on the frequencies and the corresponding mel-values.

These are filter-banks applied to the power spectrum and summing up the result to see how much energy there is in each frequency region, resulting in a spectogram where you can see how energy is contained in each region. The spectrogram will be represented as a number of coefficients for each frame. As you can see in figure 2.6 the filter-banks will be narrower at lower frequencies and wider at higher frequencies. This is a result of using the mel-scale to decide the size of the filters and will result in the elimination of information at higher frequencies(in other words it is a dimensionality reduction of the power spectrum).

Logarithm of filter banks

Next, you take the logarithm of each of the filter banks. This is again motivated by the human auditory system. If you want to double the volume humans hear, you have to increase the energy of the signal 8 times [12]. Because of this, large variations in energy will not sound that different to humans if the sound already is loud. By taking the logarithm of the filter banks it will make our features match more closely to what humans actually hear[12].

Discrete Cosine Transformation

Finally, you want to convert your mel-filterbank into Mel-Frequency Cepstral Coefficients(MFCC). This is accomplished by taking the Discrete Cosine Transform(DCT) [13] on your log mel-filterbanks. Taking the DCT of a power spectrum will turn the spectrum into a cepstrum, which is an anagram for spectrum. There are two main reasons this is done. The first is that the filterbanks are overlapping so the energy in each filter will be heavily correlated. By doing the DCT we will decorrelate each filter. Secondly, you are able to only store the lower DCT coefficients. This is beneficial because the lower coefficients will capture a gradual change in the energies between filterbanks, and the higher coefficients will capture a faster change. In turn, this will ensure that the MFCC's will vary more slowly from frame to frame. It turns out that faster change in energies will degrade the performance of speaker recognition. The process of discarding the higher coefficients is called liftering, which is an anagram for filtering[12].

2.1.3 Delta and delta-delta

The MFCC's only describes the power spectral envelope of a frame, but the speech also contains dynamic information i.e in what directions the MFCC's are moving over time. Calculating these trajectories and appending them to your coefficients will improve the performance of the speaker recognition system[12]. So if you have 13 coefficients, after calculating the delta you will have 26, and after calculating the delta-delta you will have 39.

2.2 Machine Learning

"Machine Learning is the science of getting computers to learn and act like humans do, and improve their learning over time in autonomous

fashion, by feeding them data and information in the form of observations and real-world interactions."[14]

As you probably can understand from this definition ML is a very wide field that has a lot of applications. This thesis does not focus on general machine learning, but instead, one part of the field called speaker recognition and the models used in this field.

2.2.1 Speaker recognition

"In automatic speaker recognition, computer programs designed to operate independently with minimum human intervention identify a speaker's voice. The system user may adjust the design parameters, but to make the comparison between speech segments, all the user needs to do is provide the system with the audio recordings."[2]

This is a general definition of speaker recognition. As with many areas of ML, there are many different approaches to how one can do speaker recognition [15]. This project uses a fairly simple approach; train a model based on some features (in this case it is MFCC's) and match the trained models using some way of comparing them and giving a score based on that. In the context of machine learning, models are used to represent the subject that are evaluated in the algorithms. For this thesis models are for the most part used to represent speakers. The modeling approach used is called the i-vector approach [16]. These models are as the name suggests, represented by a vector. Because of this simple representation, it is easy to compare two i-vectors and give a score. In this work, a measure called cosine distance is used to compare and score the i-vectors. The following formula is used to calculate the cosine distance for vector A and B, where A and B are models for different utterances.

$$Similarity = \cos(\theta) = \frac{A \cdot B}{\|A\| \cdot \|B\|} \tag{2.4}$$

The i-vector approach is built upon different models used in speaker recognition through the years. These will be explored in the following section of the thesis.

What is being used in this thesis is not speaker recognition strictly speaking, but rather a subfield of speaker recognition called speaker diarization. The aim of speaker diarization is to answer the question of "who spoke when" [3]. In other words, figuring out at what time each different speaker has an utterance. By using speaker diarization the hope is that we can figure out when the different speakers in an audio track are speaking and use this information to cluster the data so it can be used as training data for further speaker recognition.

2.2.2 Speaker models

In this section, a detailed explanation of the different approaches to modeling and the motivation for using them will be given. These include GMM, UBM and i-vectors. A model called supervectors will also briefly be explained in the context of how it relates to the other models.

Gaussian Mixture Models

"***A Gaussian mixture model (GMM) is a parametric probability density function represented as a weighted sum of Gaussian component densities.***"[17]

GMMs are often used as a parametric feature of the probability distribution of continuous measurement or features in a biometric system. An example of such a biometric system can be vocal tract related spectral features in a speaker recognition system, which is the way they are used in this thesis. When using a GMM as a model you first have to train it. During training, the different data points will get assigned to the different Gaussian distributions. From this the GMM will be can be completely represented by the mean, covariance and mixture weights. One can then use them to do classification by comparing different GMMs and scoring them according to similarity. Using them in this way has the drawback that it will be classified as one of the trained voices, even it does not correspond to any of them or is just noise. In other words, you will be unable to tell if a speaker is unknown or separate speech from other kind of audio. To solve this problem you can use a Universal Background Model(UBM) (see section 2.2.2). To train a GMM you use training data to estimate the parameters of the GMM, often done by using an iterative expectation-maximization algorithm. The GMM is defined by a mixture or weighted sum of M component Gaussian densities given by this equation(where g(x | μi,σi) is a normal distribution with mean(μi) and covariance matrix(σi), w is weights and x is the vector you want to model)[17][18].

$$p\left(\mathbf{x}|\,\lambda\right)=\sum_{i=1}^{M}w_{i}g\left(\mathbf{x}|\,\mu_{i},\sigma_{i}\right) \tag{2.5}$$

The motivation behind using GMM's in this thesis was that a UBM was needed and [19] indicates that GMM's was a viable option for a UBM. In general, it does not seem that GMM's are used much as a speaker-dependent model anymore. As better options have become state-of-the-art, like i-vectors and more recently d-vectors[20].

Universal Background Models

"A universal background model (UBM) is a model used in a biometric verification system to represent general, person-independent feature characteristics to be compared against a model of person-specific feature characteristics when making an accept or reject decision."[21]

In other words, a UBM is a model of a general speaker. Instead of modeling a specific speaker, a UBM will be trained with recordings of many different speakers. In this way, the UBM will recognize speech but is unable to determine the identity of the speaker. The UBM in our system is a GMM that is trained with a large set of speakers to represent a general speaker. You can then use the UBM as a model for a general speaker and thereby eliminating the drawback of GMMs where they were unable to determine if a speaker was unknown. You can do this by comparing a GMM to a set of other GMM's which include the UBM. If the UBM scores the highest in the comparisons you can classify the GMM as an unknown speaker, or more generally, an unknown sound. UBM's can also be used in the training of a speaker-dependent model (see section 2.2.2), by acting as the prior model in a maximum a posteriori estimation[19]. An overview of how a UBM can be used is shown in figure 2.7. It shows how some extracted features are used to create a GMM from features extracted from a speech sample by adapting the UBM using maximum a posteriori. At the end of the figure, you can see that the created GMM is used in some form of scoring to determine if it should be accepted or rejected.

The motivation for using a UBM started with finding a way of covering the 0-hypothesis of an unknown speaker when using GMMs. Later on in this project, after more information was gathered, newer and better models were discovered. The motivation then changed from using it to identify a general speaker, to using it as a trained model for maximum a posteriori estimation when training the matrix for extracting i-vectors.

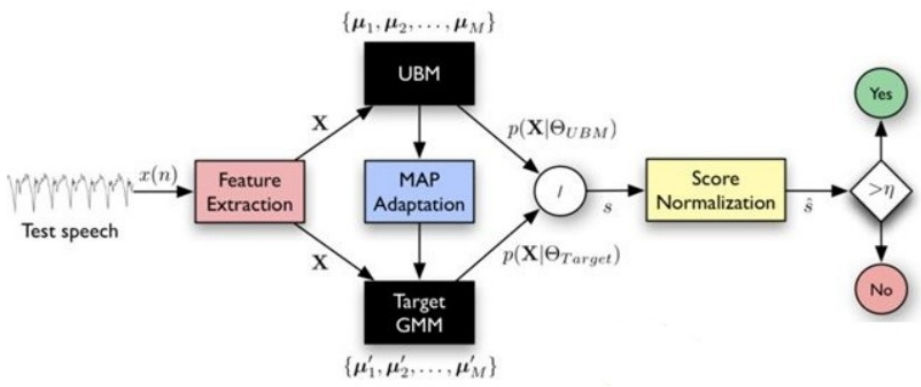

$\{\mu_1, \mu_2, \ldots, \mu_M\}$
UBM
X
$p(\mathbf{X}|\Theta_{UBM})$
Yes
$x(n)$
Feature Extraction
MAP Adaptation
Score Normalization
s
$\hat{s}$
$>\eta$
Test speech
X
$p(\mathbf{X}|\Theta_{Target})$
No
Target GMM
$\{\mu'_1, \mu'_2, \ldots, \mu'_M\}$

FIGURE 2.7: Overview of UBM usage [22]

Identity-vector

Identity-vectors (i-vectors) is an approach to model speakers where the speakers are represented in a low-dimensional space which is defined using joint factor analysis (JFA). This is in contrast to GMM's, which are in a higher-dimensional space. The JFA is used to decompose a speaker dependent model called supervector[2] into different types of components which is represented by a set of low dimensional factors. Supervectors is the precursor model to the i-vectors. The supervector model is derived from a GMM by stacking the mean vectors of the GMM components into a single vector. This can be seen in figure 2.8 where the μ's(means) are stacked together. By doing a dimensional reduction on the derived supervector(reducing the supervector into the lower dimensional space), you can derive the i-vector. The i-vector modeling approach has the advantage over supervectors that it is smaller, and discards noise etc from doing the dimension reduction. A set of the low dimensional factors is used to represent a speaker in the form of a vector. The low-dimensional space is both speaker and channel dependent and is named the total variability space, since it models both speaker and channel variability. In the total variability space, a i-vector will represent a single utterance. By using the posteriori adaptation of a UBM you are able to extract the necessary statistics to define the total variability space. Each of these vectors will model a speaker (single utterance) and contain information derived from both the channel variability and the variability from the utterance itself. The total variability space is defined by a total variability matrix T, UBM m and the i-vectors w. It is defined by the equation:

$$M = m + T \cdot w \tag{2.6}$$

Where M is a supervector derived from the UBM, and w is a random vector having a standard normal distribution N (0, I).

Classical JFA will define two spaces, one speaker dependent and one channel dependent. Through a experiment, it was discovered that the channel factors also contained information of the speakers. This was the motivation behind creating the new total variability space[16]. An undetailed overview of the steps involved can be seen in figure 2.8. Here you can see that the i-vectors follows the same steps as the UBM to make a MAP adapted GMM. This GMM is then turned into a supervector [2]. This is done by stacking the means for the different GMM components into a vector. As mentioned earlier, in order to turn this supervector into an i-vector a form of dimensionality reduction is used to represent the supervector in the total variability space.

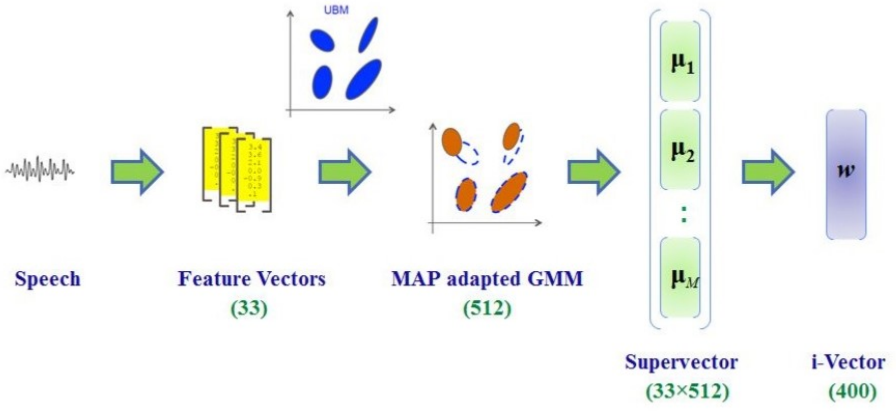


FIGURE 2.8: Overview of the steps involved in extracting an i-vector [22]

The motivation behind using the i-vector approach was that is was very close to being state-of-the-art[16] while making simple scoring using cosine distance possible. It is also an improvement over the UBM-GMM approach because the classification will be faster, as comparing two i-vectors is faster than comparing two GMMs. The resulting WAV file can then be used as training data in the pipeline or be sent through the pipeline. On top of that, each i-vector only requires a small amount of data (1-2 seconds) to be able to model a speaker [23]. (Note that this amount is only to extract a single i-vector.) This is offset by the fact that the UBM used to train the total variability matrix requires a large amount of data to train, in this case around 7-8 of data was used. But since there was a large amount of data available for notation this was not a problem.

2.2.3 Bayesian information criterion

The Bayesian Information Criterion (BIC) is a form of model selection criterion in statistical literature[24]. In the work of this thesis, the BIC is used to try to sample the audio stream according to speakers. When using the BIC to do segmentation, one wants to select between two models in a given frame. The choice is between a model where a change in speaker occurs between the given frame and the next, and one where it does not. This kind of selection is referred to as a model identification problem.

When selecting models, one wants to select the model that maximizes the BIC. The BIC is defined by equation 2.7. Where $X = \{x_i : i = 1, \ldots, N\}$ be the data set we are modeling; $M = \{M_i : i = 1, \ldots, K\}$ be the candidates of desired parametric models; λ is a penalty weight and $\#(M)$ is the number of parameters in model M. The likelihood function is maximized separately for each model M and we obtain $L(X, M)$[24].

$$BIC(M) = \log L(X, M) - \lambda \frac{1}{2} \#(M) \cdot \log(N) \tag{2.7}$$

T^2-Statistics

A problem with the BIC is that it is very computationally expensive, with quadratic complexity. This problem stems from the fact that BIC has to calculate two full covariance matrices and their determinant for each frame. Instead of doing this, you can use the T^2-statistics to suggest a change in the speaker. The T^2 assumes the covariance matrices of the two samples are the same, but unknown. Hotelling's T^2-Statistics is a multivariate analog of the t-distribution[25], and can be used for multivariate hypothesis testing. If you have two samples [1,b] and [b+1,N], the T^2-Statistics is defined as equation 2.8; where μ_1 and μ_2 are the means of the two samples and $\sum$ is the covariance matrix.

$$T^2 = \frac{b(N-b)}{N} (\mu_1 - \mu_2)' \sum^{-1} (\mu_1 - \mu_2) \tag{2.8}$$

When one has found a suggested change in speaker using T^2, one can then use the BIC in that frame to confirm or deny the change in speaker. By doing this you don't have to do the heavy computation on all the frames, but only where the T^2 suggests it.

Chapter 3

Technologies and problem description

In this chapter a wider description of the problem this thesis is aiming to solve will be given. The different techniques and technologies used and the motivation behind using them will be discussed as well in this chapter.

3.1 Problem description

When starting the work on this thesis, the general problem that was being looked at was to be able to separate two voices in the same audio track if they were not talking at the same time as shown in figure 3.1. This was of course only the base problem of the thesis. The problem later evolved into how could a machine learning algorithm be used in an existing system to add new functionality, and improve workflow. To achieve this, a pipeline that had the desired functionality had to be built and integrated with the existing system.

The first part of solving the problem was to figure out how to extract the necessary information, and what exactly this information should consist of. It was decided that the information should consist of time-stamps of when the different speakers in the track talked. This would allow Vizrt to, for example, use the time-stamps to either extract the audio to use as training data for a new model, extract the audio and compare them to existing models for classification or to use it to label the video in their editing software. Extracting the information would be done with BIC sampling, i-vectors and clustering, which is explained in more detail in a later chapter.

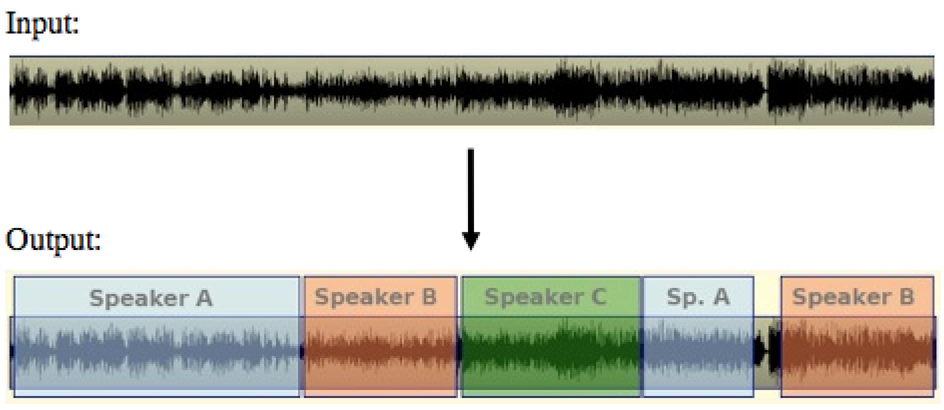


FIGURE 3.1: Visual representation of speaker diarization [26]

The Next part of the problem was piping the information into Vizrt's systems. To do this the information had to be in a format that was compatible with the pipeline used in their systems. Because of this, the information either had to be in the correct format after extracting it, or an extra step had to be added to make sure it was converted into a usable format. When looking into this part of the problem, the realization that hooking into their systems would provide some benefits was made. If this could be done, the UI could be used not only to test the algorithm and being able to see the results in a clear format. But using the UI would also give users the option of editing the results from the algorithm. Had the thesis work stretched over a longer period of time, a UI that is tailored to this pipeline would be the best option. With the lack of time, and the fact that the Viz Story UI does provide a lot of the necessary functionality that would have been in the tailored UI. Using the Viz Story UI to demonstrate the usage of the program seemed like a good option.

The problem of hooking the algorithm to the system and facilitating the sending of information back and forward would be solved by using their REST API. Subsequently, the problem in this part became how exactly to convert the timestamps from the algorithm into a usable format, and how to best use the API to send this information. Since the system was built on using XML-documents, the timestamps had to be converted into their existing XML-format.

In all, the problem that this thesis aims to solve is to build a pipeline that has certain functionality and can be used in by a video-editing system. Having integrated the pipeline, the users should be able to use the functionality provided in their video-editing. This pipeline's core functionality will be speaker diarization. The aim is to use the information gathered by the diarization to provide more functionality to the system using the pipeline. By using this information the system can train up new models to recognize people. These models should also be able to improve over time by running more videos through the pipeline containing the same people and gathering training data for them using the functionality in the pipeline. Improving the models will hopefully lead to the pipeline delivering better results over time to the existing video-editing system.

3.2 Potential functionality

The pipeline that is being built in the work of this thesis is only a proof of concept, and there will be potential to expand the functionality which the program provides. Much of the potential functionality will go untested in this thesis, but in the section below some speculation on how such a pipeline could be used together with the video-editing system.

Use the information from the program to search through all the files

After a file has been sent through the pipeline, the information gathered could be saved as metadata about the corresponding file. By doing this, the video-editing system could use this information to search through the files in the system. This could for example, let the users easily get all the video where Barack Obama speaks, without having to go through the work of tagging the video beforehand. How this functionality could look can be seen in figure 3.2.

Use the information from the program to search through the file in the editor

The metadata than could allow users to search through the files in the file library could also be used to search through the content of each individual files when used in the editor. If the editor knows when each of the speakers are speaking, the editor could use this information to let the users jump to the part of the file where a certain speaker is speaking. This could speed up the editing process by letting users quickly find the part of the video that they are interested in. It could also let the users quickly cut out all the speakers that they are not interested in. This

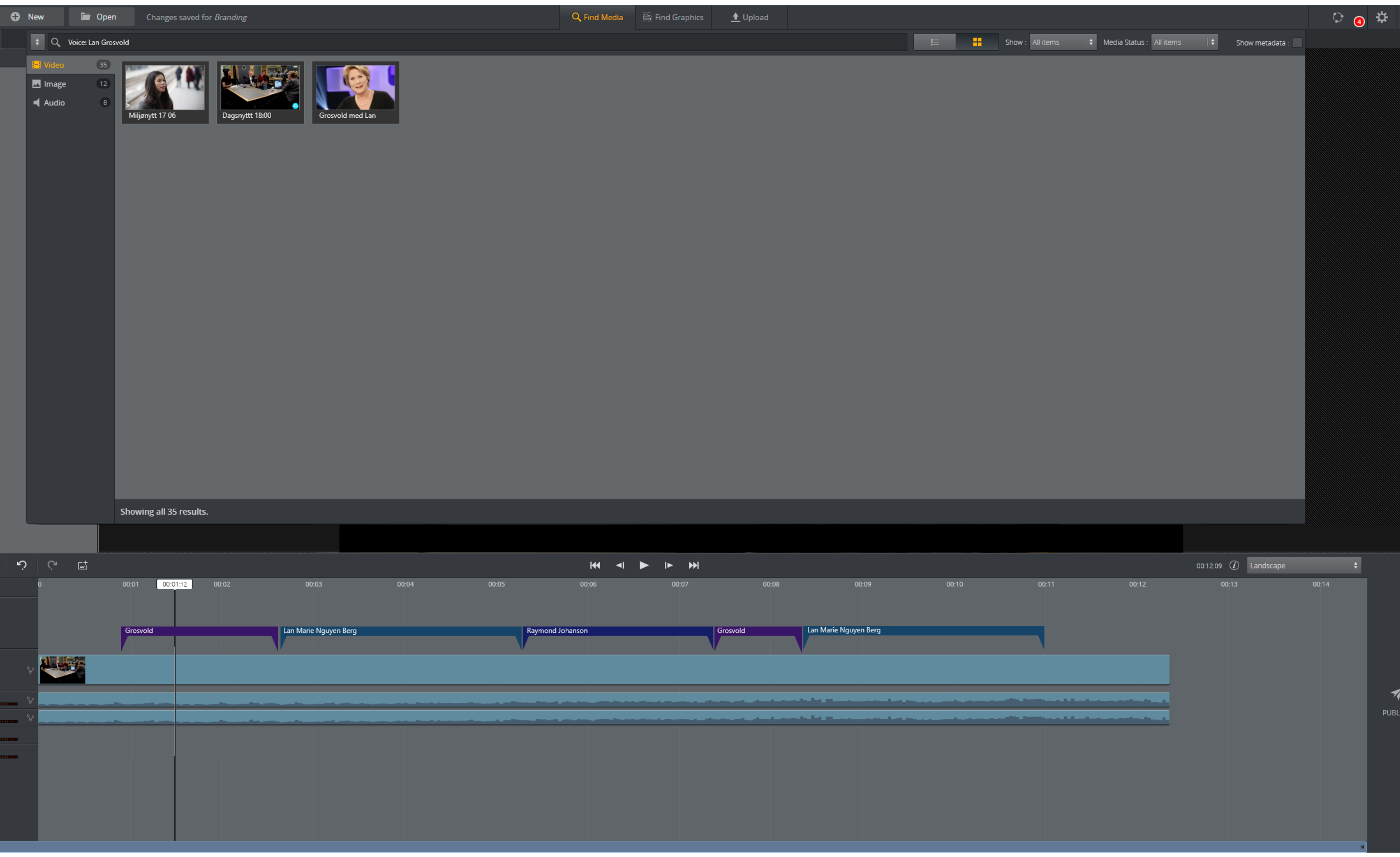


FIGURE 3.2: Mockup of how the UI could look if it has the functionality to search through the files based on known speakers

would require the system to have a certain confidence in the classification. Since if the classification was wrong, a user could unintentionally remove parts of the video that could be of interest.

Display information about the different speakers

How this functionality could look in the UI is shown in figure 3.3. This could be of assistance for certain types of users, especially journalists. By having the information displayed in the editor it could save the journalist time that they otherwise would have to spend looking it up. Of course this would require that someone manually enters the information into the system as metadata for the model for each person in the database.

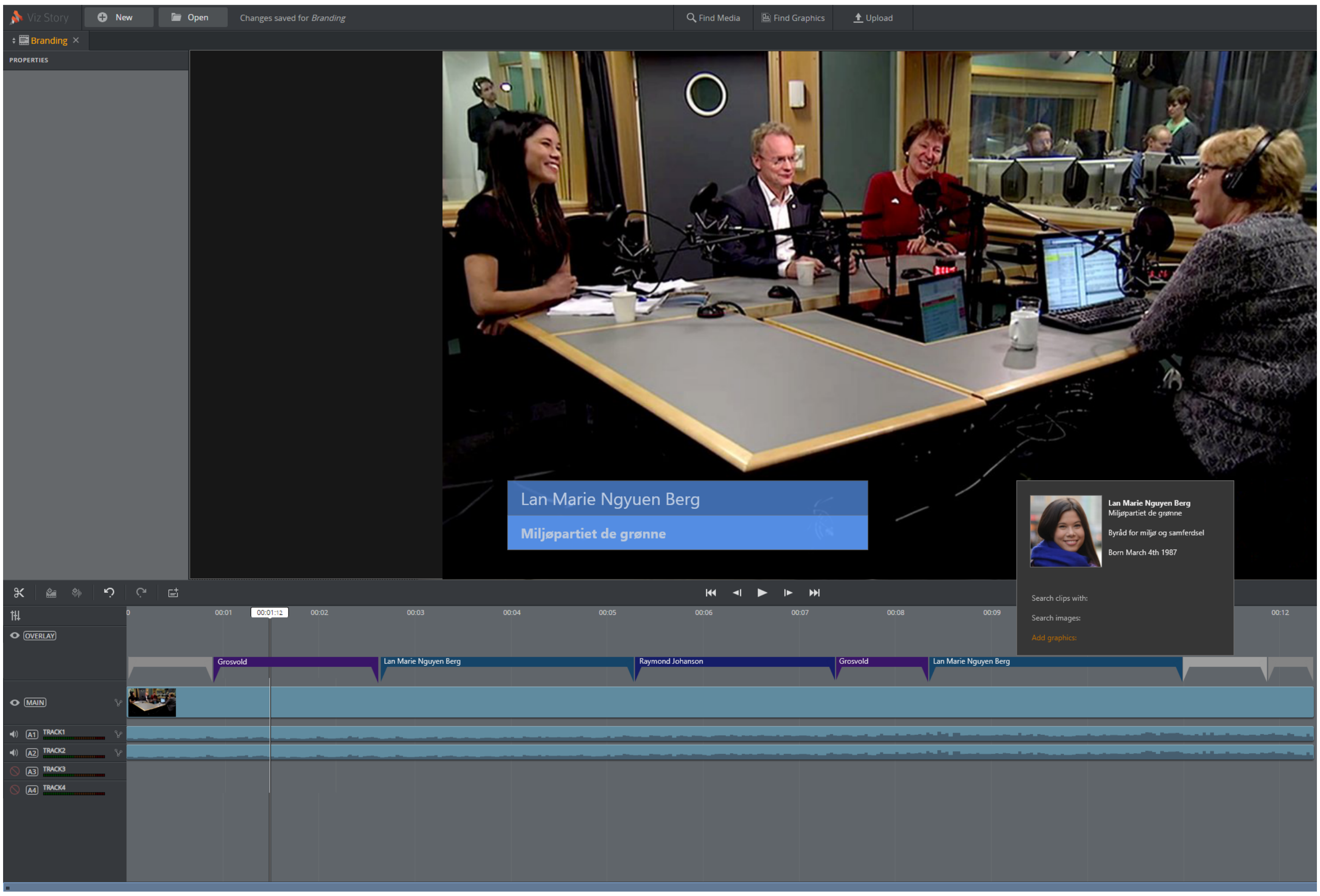


FIGURE 3.3: Mockup of how the UI could look if it has the functionality to display information about known speakers

Give a feedback about how certain the program is of its classification

Displaying this information would be more of a quality-of-life functionality. By displaying the certainty of the classification of known people as some sort of score, it allows the users to evaluate if they should trust it or not. So instead of being uncertain of the classification or going through and checking them, this would allow the users to only go through the classification that the program is uncertain of. At the same time the editor could display a list of alternative classification and the confidence it has in those different classifications.

Change what kinds of elements uses the information

The result of the classification will be displayed as a lower-third graphics element in the proof-of-concept version of the pipeline. Lower-third is a term used in the TV-business. It means the graphics displayed at the bottom of the screens, often containing the name of the person talking and some information about him/her. Having the alternative to chose what kind of element should be used in concurrence with the classification could be advantageous. While displaying them as graphics elements will be acceptable in certain types of videos (news broadcasts etc), this will not always be the case. By allowing users to chose this lets the classification be used more widely in the editor. A option to not display the classification at all should also be an option, since this would probably be the default option. From there one would let the user chose the types of elements they want to use, if any.

3.3 Technologies and techniques

When starting work on this thesis, different techniques for machine learning were discussed. The choices pertaining to technologies was more limited, as the only choice to be made here was what programming language to use. The different options and the motivation behind the choices made will be discussed in this section.

3.3.1 Technologies

When looking at the choices for programming languages, the first step was to do a little research on what is mostly used in these kinds of works. What was found was that for the most part one of two programming languages are used; C++ for the low-level parts of algorithms, which will often include parallelized linear algebra. Also Python or similar dynamic languages for high-level descriptions of the algorithm and API.

When evaluating what to use, what each of the languages would provide was looked at. Broadly speaking the two languages was opposites when it came two what they provided the thesis work. Python provides simplicity in its syntax as well as a lot of packages, but in exchange, python can be very slow when working with a lot of data. C++, on the other hand, is very effective, but its syntax can be difficult to work with. Looking at the advantages and disadvantages of the two options. the

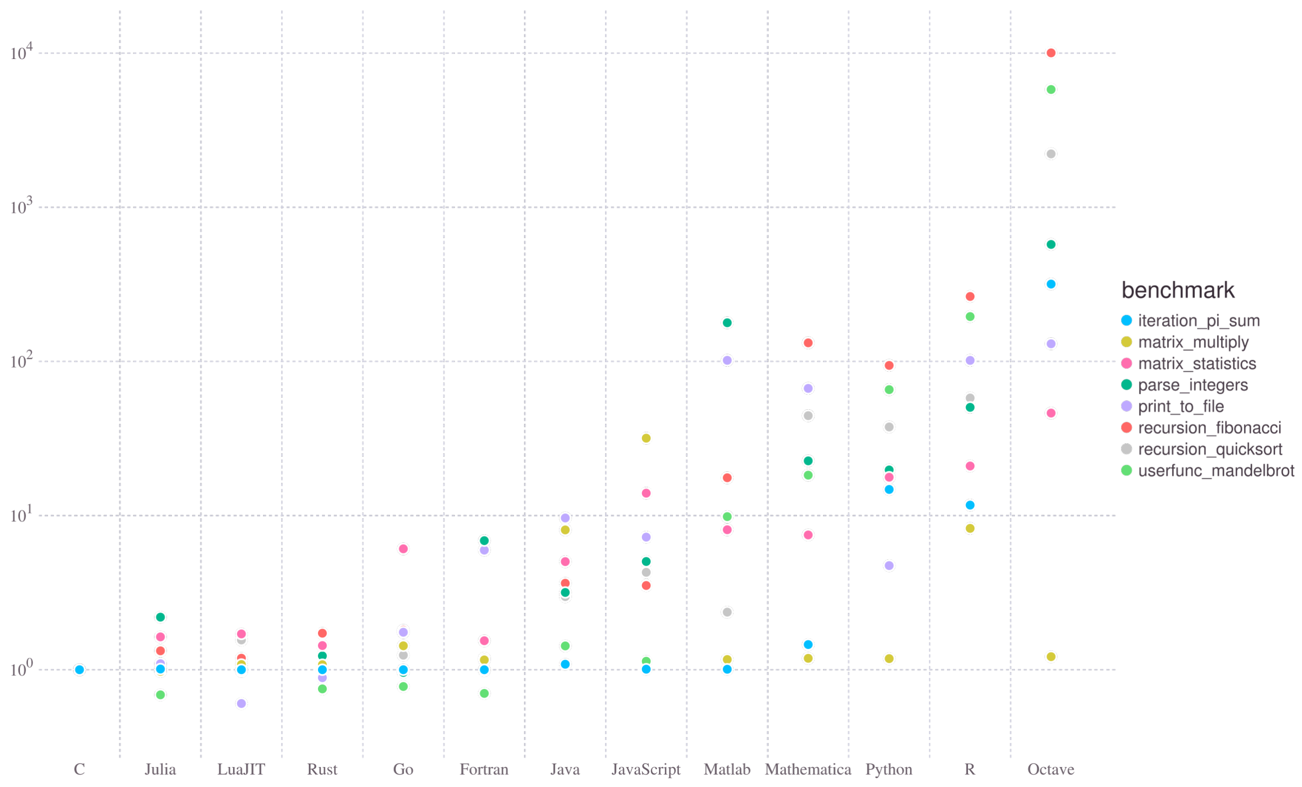


FIGURE 3.4: Benchmarks of different languages for a range of common code patterns [28]

decision to use Python was made, and convert the code to C++ at a later date if speed became a necessity.

After this decision was discussed with my external advisor at Vizrt he recommended that some research should be made into a programming language called Julia. As it turns out this language seemed to fit this work very well. It is advertised as being fast, as well as having a relatively simple syntax which seemed to be somewhat similar to python[27]. The way Julia does this is by having strong typing, unlike Python that has to spend time checking what types your variables and data is in. A comparison of the run time of some computations in different languages can be seen in figure 3.4. Julia seemed to cover the area of use for both Python and C++. For the need of this thesis, it seemed like Julia was the best of both worlds, providing both speed and simplicity. As well as having good packages available to do speaker recognition. Based on this, the recommendation from my external advisor and the fact that there were packages available in these languages to support the work done in this thesis. The final decision to use Julia and Python as the programming languages for this thesis was made.

Video editing system

This section will give an overview of the type of system that the algorithm will be hooked into. It will also talk about the specific system that is used in this thesis, namely the Viz Story system[29]. Vizrt's systems are relatively large, so this section will only focus on the parts that are relevant for this work.

Viz Story is a web-based video editing system and is what is called a Non-Linear Editor (NLE). An NLE is editing software that performs Non-destructive editing. Which means that the original media content is not edited during the editing process. Instead, you set up an Edit Decision List (EDL) which specifies the editing steps that should be taken each time the video is produced. Applying the different steps in the EDL to the video is a process called rendering. Many of today's software programs are NLE's. In figure 3.5 you can see the UI of an NLE. The lower part of the UI shows the editing steps that will be done when rendering the video. This is how the EDL for this video is represented in the UI.

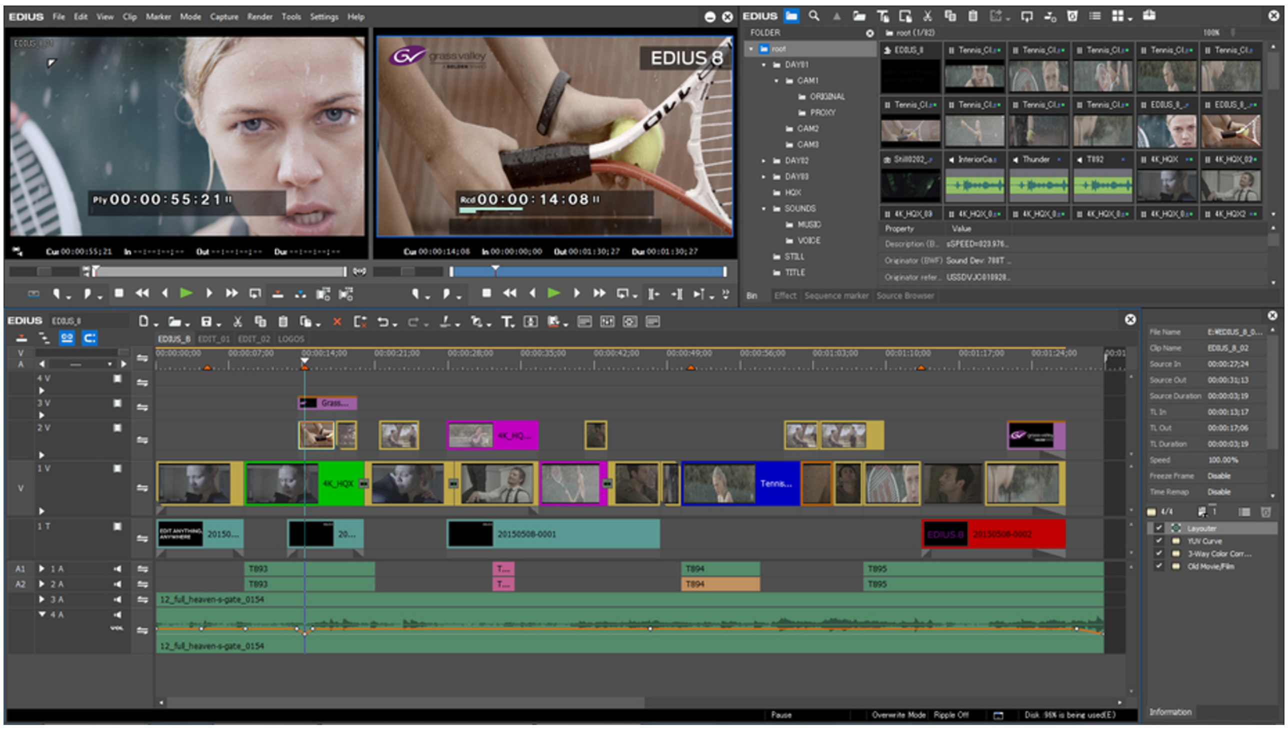
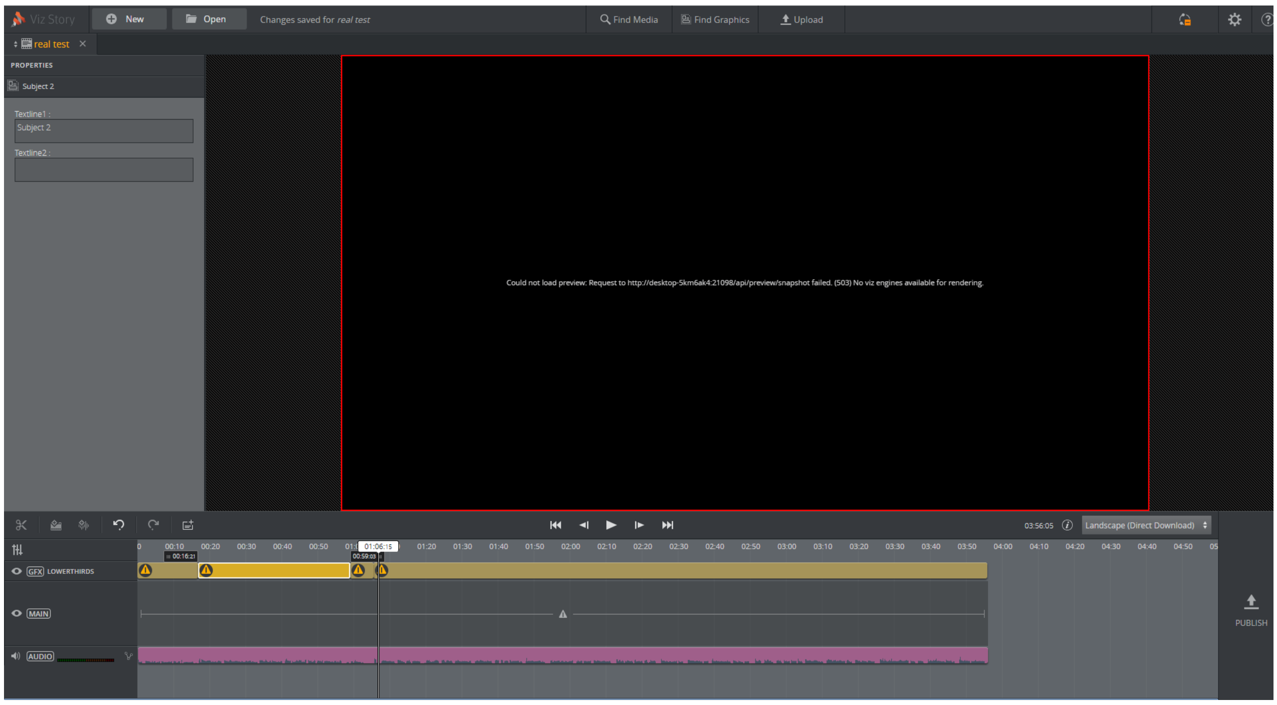


FIGURE 3.5: An UI of a Non-Linear Editor [30]

Viz Story allows users to access its functionality through its User-Interface. By taking advantage of this UI, we can give a demonstration of how custom UI for the program might look. It can graphically show where the different persons talks by showing the results from ML graphically, and it lets you edit the output from the algorithm. How this UI looks can



```
Could not load preview: Request to http://desktop-5km6ak4:21098/api/preview/snapshot failed. (503) No viz engines available for rendering.
```

FIGURE 3.6: The Viz Story UI after completing a run through the pipeline

be seen in 3.6. This figure shows where the different persons talk in the audio track by the yellow elements in the bottom part of the editor.

When a video is edited using the Viz Story an entity called a story is created. Part of this story is the EDL that is created during the editing process. This list contains all the editing that is done to the media-content during rendering (cuts, graphic elements, etc.) without containing the actual media content. The whole Viz Story system is based on the XML format. So when editing, the EDL that is generated will also be represented in XML format.

XML documents used in the system makes use of, among other things, the Atom format. The Atom format is an XML-based Web content and metadata format. It is also an application-level protocol for publishing and editing Web resources belonging to periodically updated websites[31]. The XML document that is generated will contain an XML-element called payload, which again will containing different elements. Most of these elements are not relevant to us, the element that we are interested in is called a feed. A feed is a series of related items that a content provider publishes on the Internet[32]. The feed element contains a link to the media-content used (again not the actual content) and all the editing decisions. These editing decisions come in the form of entry elements in the feed representing cuts, graphics, transitions, etc. An example of an entry element used in a story can be seen below(3.1). Only parts of the entry element are shown here since it would be too large to include in the thesis text. In the UI the content and the editing will be displayed in a timeline. This timeline is represented in the document by the entry elements in the feed. If you want to add say a graphic element to a video, you can just change the feed by adding a new entry element that tells Viz Story to add the graphics on the video in the UI.

```
<entry>
    <id>C5621FA4-6C52-404B-99F9-B7DBBB4434B3</id>
    <title type='text'>Subject 0/8</title>
    <updated>2017-08-02T10:35:27+02:00</updated>
    ⋮
        <payload xmlns='http://www.vizrt.com/types'
                model='http://localhost:8177/templates/10031/mode'>
            <field name='02-text-e'>
                <value>Subject 0</value>
            </field>
        ⋮
        </payload>
    </content>
    <vaext:start>60.9</vaext:start>
    <vaext:duration>5.140000000000008</vaext:duration>
    ⋮
    <vaext:track edittrack='overlay' togglelayer='Lowerthirds'/>
    ⋮
</entry>
```

LISTING 3.1: Parts of a entry element from a XML document representing an EDL

The whole Viz Story product is a web application with a front-end and back-end that is built upon a REST API. With the REST API, one is able to send requests to the back-end to change the data that is there. As mentioned earlier the system is based on the XML format. So when one wants to change the data in the back-end, you have to do it by changing the XML documents. The REST API lets one send the XML documents representing stories back forth from the back-end, and by doing this one can change the stories such that it contains the information that was retrieved using the machine learning algorithm.

Viz Story gives different options for what to do with the video once it is rendered. These options come in the form of different publishing targets. When publishing the video, the video will first be rendered before sending it to the publishing target. This differs from normal NLEs which usually just does the rendering. Each publishing target will have its own code that determines what is done with the rendered video. This can be uploading it to a website like YouTube using the REST API. Or it can be simply downloading the finished rendered video locally to the computer. By having this functionality, one has the option to write new publishing targets that runs some code each time a video is published using that target.

3.3.2 Techniques

Unlike with the choice of technologies, there were a lot of decisions to be made when deciding what techniques to use. A decision about what kind of machine learning method was best suited for the work had to be made. Then how to model the data that these methods used. These decisions had to be made in parallel to some degree, as what method you chose would limit the modeling options that were available, and vice-versa. In this section, the different options for both methods and models that were considered will be discussed and the motivation behind the choices that were made.

Sampling

When first starting to write the code for generating samples, the idea was to simply cut the audio track into 2 seconds samples. This was motivated by how much data was needed to extract an i-vector[23], and the simplicity of the implementation. Sampling the audio in this way worked adequately, while it did leave a lot of room for improvements. Among them being; even if 2 seconds was enough data to extract the i-vector it still is a small amount of data that could be larger. Subsequently, cutting the audio with no regards to the content of the audio, gave a good chance of a sample containing speech from different speakers.

Later during the work on the thesis, it was decided to go back and improve the way sampling was done. To do this more literature study around sampling performed. Based on this it was decided to use the Bayesian Information Criterion to do the sampling[24]. The motivation behind using the BIC was that it would both produce samples that were longer than two seconds, and it would sample according to speakers. This would make it so that each sample would contain only one speaker, which it seemed to be able to with decent results. The drawback was that it took a long time to sample an audio-track this way. To counteract this, the T^2-statistics was implemented to find suggestions for decision boundaries. Then let the BIC check those suggestions to accept or reject those boundaries.

Machine Learning Methods

The first part of deciding what ML method to use was looking at what fits our problem The two main areas that were considered were regression and classification. A long time ago these two were the primary areas of ML. With the advancements made over the years, ML has become much more versatile and has opened the possibility to solve problems by using other methods than just classification and regression. Regression is mostly used when estimations are involved. One example could be if one wanted to estimate how fast a car would drive given certain conditions one could use regression to create a model/formula for this. Classification is used when one wants to classify something. Say if one wants to decide what colour a car is one could use classification.

It was pretty obvious from the start that classification was the best option for this work. However, since this thesis would work mostly with unknown models classifying them directly by comparison and scoring would not work. Consequently it was decided to use clustering to do the classification. The goal of a clustering algorithm is to divide your data into different groups(clusters) according to some measure. From here one can classify the different clusters as different speakers.

A visual representation of a number different machine learning methods can be seen in figure 3.7. None of these methods are used in this thesis, but the figure should give you an idea of how many different methods are available in ML.

This decision did limit the available options a little, but not by a lot as many machine learning methods can be used for both classification and regression. The next decision that was made was to not to use neural-network, and by extension deep learning. Over the years, especially recent years a lot of progress has been made in neural networks and especially deep learning. Because of this, among other things, deep learning has become more and more popular, and the resources available for using deep learning are more widely available (Tensorflow, etc). Still, the decision not to use deep learning neural networks was made based on two reasons. First and most importantly, deep learning network usually requires a lot of training data. As it was unknown how much training

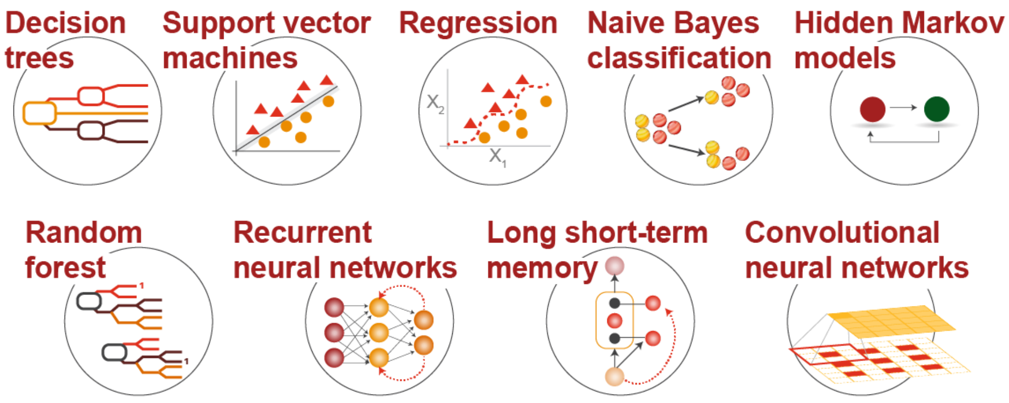


FIGURE 3.7: Visual representation of different machine learning methods [28]

data would be available for this work it did not seem like a good decision to risk using deep learning. The graph 3.8 shows a visual representation of why deep learning requires so much data. Traditional machine learning will spike in accuracy with relatively little data but will stop increasing in accuracy after reaching a certain amount of data. In comparison, deep learning needs more data before it becomes very accurate, but will continue to become more accurate the more data you feed it. Secondly, from what research that had been done at the time it did not seem like neural networks and deep learning was widely used within the field of speaker recognition.

At this point, the rest of the decisions revolved more around what models to use. It was decided to use the approach explained in section 2.2.1 of doing simple classification based on cosine distance(2.4). This simple approach led to most of the complexity of the machine learning revolved more around the models rather than the approach to classification. This decision was motivated both by its simplicity, and the fact that from the literature study done, it seemed like a common way to do it. It was not always the case that the classification was based on cosine distance, but in many cases classification was done by comparing models and scoring the accordingly[3][15].

Models

Details of the theory behind the different models that were tried out during this work were covered in section 2.2.2, so this section will not cover that again. Instead, this section will talk briefly about how the different models were tested and the motivation behind the final choice of model. The first model that was tested was gaussian mixture models.

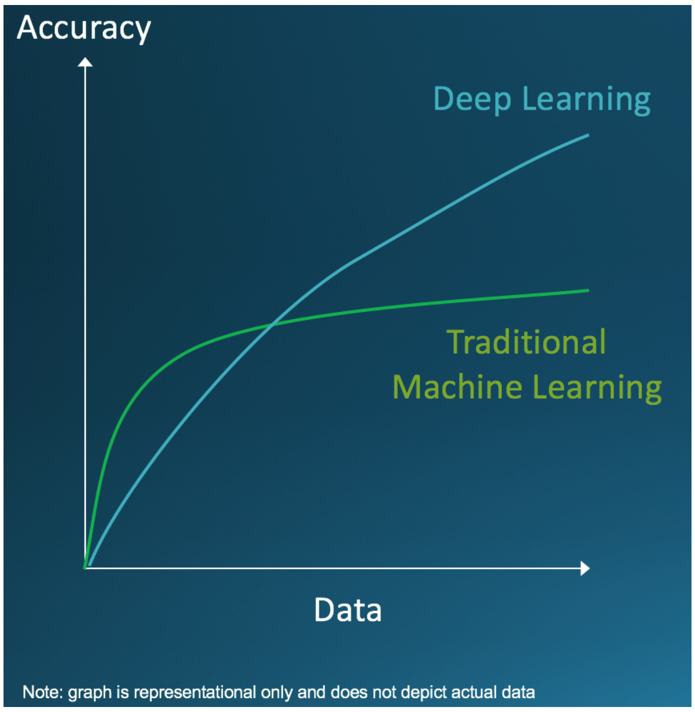


FIGURE 3.8: Graph showing how the amount of training data affects the accuracy of deep learning and traditional machine learning algorithms.[33].

This was motivated by [17] and that it seems like this is the model most of the modern research of models for speaker recognition tasks are built upon. To implement this model, a package for Julia was used [34]. This package was made by a man named David van Leeuwen. He is also responsible for creating the package used for the i-vectors. It was still decided to move on from this model, even if using it gave good results when used for speaker recognition. The reason for this was simply that it was an old model and the literature study discovered newer and better models had been presented in later years. When testing this model, it was trained using MFCC and had a density of 512. The density is a measure of how many components are in the GMM. To classify the models, a scoring system using the average log likelihood was used.

Next, the work moved towards using i-vectors. During this time a model called supervectors was tested [2]. Since this model was not used in the final work the details of it will not be explained, but the motivation for testing it was that it is the precursor to i-vectors and because of this it could be interesting to see how it worked compared to i-vectors. It seemed to deliver good results, but as i-vectors was a newer and therefore presumably better model it was not used going forward in the thesis work. The implementation of supervectors was done with a package for Julia [34].

Finally the i-vectors was was implemented. To reiterate, the motivation behind using i-vectors was mostly that at the research done at the time indicated that i-vectors was state-of-the-art and superior to older models. Also even though i-vectors work better with a larger amount of training data, there were cases where i-vectors had been used successfully with a relatively small amount of training data (1-2 second utterances)[23]. Which was a good fit for this work as it was unknown how much training data would be available during actual usage. It was tested by using MFCC as features for extracting the i-vectors, and cosine scoring to compare the i-vector against each other.

Chapter 4

Design and implementation

In this chapter, the design and use of the pipeline will be explained. The details on how the different technologies are used will be laid out and the process through which they were implemented will be talked about. Also how they are used in cooperation with Vizrt's system and API. Much of the details concerning the technology and system has been covered in earlier chapters so this will not be covered again here.

4.1 Implementation work

When starting the work on the implementations, there was still some discussion on what the exact end-goal was going to be. So the work started with implementing a program with GMM's that was able to do speaker recognition. When that was done the work moved on to turning those GMM's into supervectors. Even though at this point it was decided that i-vector would be used; It was decided to still implement the supervectors because I wanted to be able to gradually implement the different models to see how they work compared to each other. There also were some parts that were going to be used regardless of which models were implemented at the time, like the MFCC.

Before going through the different steps of the implementation in detail, a high-level description of the resulting pipeline will be given.

4.1.1 High-level description of the pipeline

The result of the implementation was a proof-of-concept pipeline that provides new functionality to a video-editing system. The main functionalities that are made available are speaker recognition and diarization. Using the main functionalities, the pipeline provides other functionalities that can be used in the editing process. These include; presenting the result of diarization in the system UI, extracting training data, and use the data to create new models for people. By using these functionalities the pipeline can take advantage of the new training data and create new and better models each time new training data is extracted.

The pipeline is hooked into the editing system via a REST API as a prototype for fully integrating it into a system. Doing this we can get an idea of how the pipeline can work together with an editing system and create a better editing experience for the users.

4.1.2 Creating training data

Before the work on the pipeline started, a method for getting training data had to be created. After looking at different sources it was decided to use the Swedish Riksdagen web-tv as a source for data[35]. There were several reasons for this decision. Since the data came from political debates there were few instances where people spoke at the same time. This was important since the pipeline is unable to process audio where people talk at the same time. The audio from the source was also high quality and contained little noise, and there was a great amount of data available. Taking this into account the videos from Riksdagen seemed like a very good source to use to collect training data.

To extract the data, videos were downloaded from the website and FFmpeg[36] was used to extract the audio from the videos. An example of the FFmpeg extraction command is shown below(4.1).

```
ffmpeg -y -i downloaded_video.mp4 -ss 0.0 -t 822.0 -vn -acodec pcm_s16le -f
    wav ./extracted_audio.wav
```

LISTING 4.1: FFmpeg exmaple command

The resulting WAV file can then be used as training data in the pipeline or be sent through the pipeline.

MFCC package

The MFCC'c were also implemented using a package[37]. The way this package work is that it takes either a WAV file or an array that contains the data from an audio file. There are different methods that the package provides for doing the MFCC routine. In this program, the highest level function is used. This interface provides an easy way of getting the MFCC with the most standard parameters. On top of that, it also lets you adjust some of the parameters, as well as set some options. For example, if you want the results normalized or if you want the MFCC to contain the derivatives. Both of these options were used in this implementation. If one wants to have some more fine-grained control over the routine, the package also gives you access to the methods that the interface calls. In this case it was not necessary as only a standard MFCC routine was needed.

Sampling with BIC

As mentioned in section 3.3.1 the first implementation of sampling only produced 2 seconds samples. This was later improved to use the BIC. The BIC was coded by referring to the formulas in [24]. When first implementing the BIC, it was quickly discovered that the algorithm was much to slow to be of any real use. As a result, a fair amount of time was used to try to improve the algorithm that was used.

The first step that was taken was to stop doing the BIC on the edge of the track. This was done because it was very unlikely that any changes of speakers would occur in the first and last couple of frames of the audio. Since this would mean that the audio track is cut off before the end or begins in the middle of the original audio track. It also removed some strange results that was produced when doing the BIC on the the edges of the track. The results of such a BIC can be seen in figure 4.1. Why these strange results were produced was never discovered, but since they did not affect the rest of the results they were ignored. Doing this did not speed up the algorithm much since the number of frames that were skipped was relatively small. Next, the algorithm was changed to incorporate a window of a small number of frames and increase the number of frames analyzed each run it did not find a boundary to cut at until a maximum number of frames were reached. When the maximum number of frames were reached, the window becomes a sliding window of fixed size, where it will look at a new set of frames each run until it

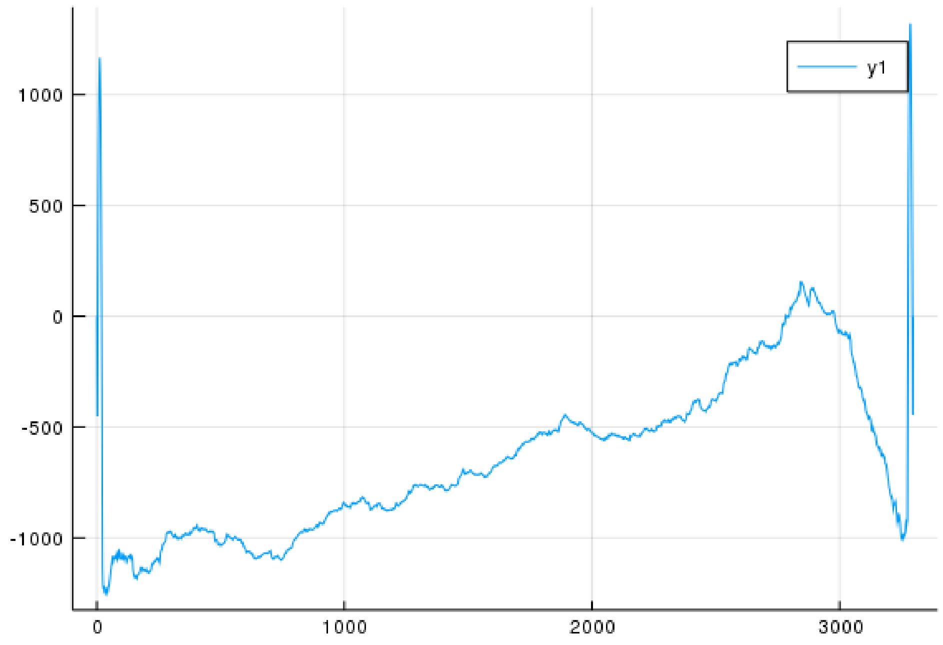


FIGURE 4.1: The result of doing BIC on an audiotrack. Where the x-axis shows the frame number and the y-axis shows the score from the BIC in that frame

finds a boundary. A visualization of how the sliding window works can be seen in figure 4.2. If a boundary is found the window will start over at the minimum size and repeat the process. These changes did noticeable speed up the algorithm since it did not require you to do the computations on each frame for each run through the algorithm. Even with these improvements, the algorithm was to slow to be of much use.

When doing some more research the T^2-statistics were discovered. This was implemented such that the T^2-statistic was used the same way that the BIC was used before (by checking a certain number of frames with a sliding-window etc). The highest score of the T^2-statistic was compared against a set threshold value to determine if there should be a cut in a certain frame. This threshold value was determined by testing different values and picking one that was not too sensitive and not too discriminatory. If the score was higher than the threshold a suggestion to cut in that frame was made. The BIC was then used in centralized on that frame. Meaning that only that frame was scored using the BIC and a certain number of frames from each side of the suggested frame. If the BIC also determined that there should be a cut in the suggested frame the time-code for that frame is saved as a cut. On the other hand, if the BIC rejects the suggested frame it is simply ignored and another run through the algorithm is started. Implementing the T^2-statistic finally sped up

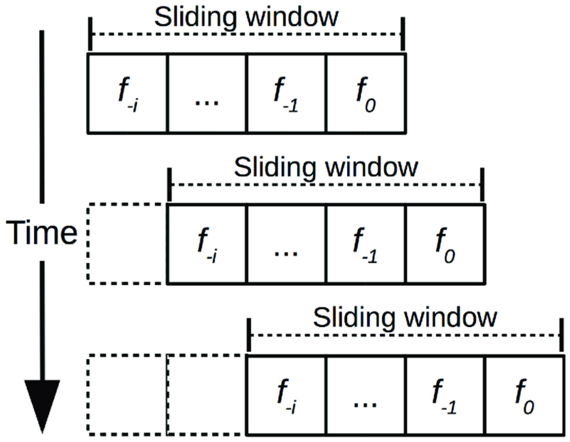


FIGURE 4.2: Visualization of the sliding-window technique, where the window slides with 1 frame increments[38]

the algorithm enough that is could be usable in the pipeline. The problem was that in none of the research papers that were read gave a good explanation of how to precisely chose the frame to cut in. As it seems that always picking the highest scoring one not always gives the best result.

Implementation of the i-vector package

After implementing the GMM and supervector, some time had passed and the understanding of the problem had become clearer. Not only did the program have to be able to separate speakers, but there were also the problems of being able to recognize speakers, being able to edit the results and being able to deliver the information about when someone is speaking to Vizrt's systems. To solve these problems the first step that was worked on was implementing the i-vectors since this was the most central part of the program. From implementing the GMM's and supervectors earlier, some of the parts that needed to be implemented were already in place, such as the functions for reading a WAV file and doing the MFCC. So what remained to implement was functions for training the i-vector extractor and extracting the i-vectors. To train the i-vector extractor a UBM is needed, but again this was already taken care of when implementing the GMM. The UBM was trained with around 7-8 hours of audio from [35], which is where most of the training and test data is taken from. As mentioned in section 3.3.2 I used packages to implement the GMM and supervectors, luckily there was a similar package available for i-vectors [39]. This made the implementation considerably easier since there was no need to implement the heavy mathematics that is behind the i-vector approach.As mentioned the package lets you train an i-vector extractor by using a previously trained UBM. It takes the statistics from this UBM and scales and centralizes them. These statistics are used in the function for training an i-vector extractor. This function takes the statistics, the number of iterations you want the training algorithm to run, and the number of dimensions the extractor should have. Finally one can use the extractor in the function for extracting i-vectors. This function takes the i-vector extractor, and the scaled and centralized data from the UBM and the MFCC data.

Work done to improve the results of the machine learning algorithm

When both the function to extract i-vectors and train i-vector extractor was implemented another problem arose, the i-vector did not perform as well as the GMM's or the supervectors when doing speaker recognition. They did not really perform well enough to do any kind of speaker recognition. How well a machine learning algorithm needs to perform varies depending on the use case. Normally you would, of course, want to have the best results possible, preferably as close to perfect as possible. In some cases, close to perfection could be a necessity, say if you want to use ML to do diagnose medical patients.

For this thesis getting a perfect result was never the goal, as part of the problem was to be able to edit the results that were classified wrong. Even so, some goals had to be set for what was an acceptable performance from the classification. After some discussion, it was decided that the goal should be a classification approach that can at least recognize around 80-90% of the utterances one is testing (even this could normally be considered sub-par). This goal was arbitrarily decided on from discussion with the external supervisor. At this point, the classification approach of directly comparing cosine-distance scores only correctly classified around 40-50% of the utterances.

The first step that was tried to improve this was to change the different parameters of the models. There were several parameters to regulate since one can change the parameters in the UBM and the i-vector extractor. First, the parameters of the UBM were changed. Here one can adjust the density of the GMM used and the number of iterations the training algorithm runs. Not much time was spent on this, as David van Leeuwen, the man responsible for making the packages was contacted. He gave the most common parameters used for both the extractor and the UBM. These parameters were a dimension of 400 and 3 iterations

Iterations	i-vector dimension	Result in %
10	400	15.8
9	400	7
8	400	0.3
7	400	55.5
6	400	59.2
5	400	55.5
4	400	44.4
3	400	62.9
2	400	44.4

TABLE 4.1: Test result with different number of iterations for training the extractor

for the extractor, and a density of 1024 or 512 for the UBM. The density was set to 1024 as this seemed to be the most common value for this parameter according to the different research papers that were looked at[23]. After setting these parameters the performance went a little up, but it was still not on a satisfying level. During the implementation, a test was done to see how the number of training iterations for the extractor affected the i-vectors. The result of this test can be seen in table 4.1, where the result shows many percents of the i-vectors were classified correctly. In this test the cosine-distance was used was used to classify the i-vectors.

An important part of machine learning is to pre-process the data. Until this point, it was assumed that the MFCC routine took care of normalizing and standardizing the data used to extract the i-vectors. Because of the lack of experience with working with machine learning an important step in the pre-processing was missed, namely the fact that the data used to make the MFCC's should also be pre-processed. When adding standardization to the data extracted from the WAV file, the classification results had an immediate improvement and went up to an acceptable level.

Implementation of the clustering algorithm

Implementing a clustering algorithm to classify the different segment was the last step of the machine learning part of the system that needed to be implemented. There was some debate on how the clustering should be implemented, mainly about how the number of clusters should be decided. The optimal solution for the system would have been if the algorithm could decide the number of clusters itself, but this would require the use of a more complex algorithm. It was decided to implement a simple algorithm, namely an algorithm based on k-means [40] and improve the algorithm at a later time if the amount of time left allowed it. The decision on what cluster each sample should be part of, and thereby what speaker they get classified as, was based on the cosine distance of the i-vectors of each sample and the i-vectors corresponding to the clusters. At the start of the algorithm a specified number of clusters are created each with a corresponding vector. Each of the sample i-vectors is then compared to the i-vectors corresponding to each cluster and is assigned to the clusters that gave the best cosine score(2.4). The i-vectors corresponding to the clusters is then recalculated by taking the mean of all the sample i-vectors in that cluster. This loop will continue until either no change is made between iterations, or a certain number of iterations has been done.

Unfortunately, this algorithm did not perform the clustering well enough to use the result. At this point, the time left for the thesis was running out, so as a solution a package running the k-means algorithm was implemented. This, of course, was not the solution this thesis wanted to work towards since the goal was to have a clustering algorithm that was based on cosine distance.By changing the approach to clustering, the scoring behind the classification of the segments with unknown also changed. Since the speaker classification for these segments was based on what cluster they were put in. This change did not affect the classification of known speakers since this was still based on directly comparing cosine-distance scores. What the reason behind the low performance of the first algorithm is uncertain. But a guess is the lack of optimization and the lack of weights in the implementation. Or it could, of course, be the result of some fault in the implementation.

Implementation of the connection between the ML and video-editing system

Until this point, all of the code had been written in Julia, but when the time came to integrate the machine learning system with Viz Story this changed to write part of the codebase in Python. The reason behind this was the packages available for handling XML documents and working with REST APIs in python. The first step that had to be done was to write code to transfer the result of the clustering to the python code. This was done simply by saving the clustered timecodes in a text file. Then the code for piping this information into Viz Story was written. To make the data more manageable, a function that concatenated the data was written. For example, if a cluster contained the timecodes "1-3" and "3-5" this would be concatenated into "1-5". Before this information could be sent to Viz Story it had to be in the correct format, namely an XML document. A template document was made by copying an existing EDL from Viz Story. To prepare the EDL which is sent to Viz Story, the template document is read into python using the lxml package[41]. Using this package a function was written for putting each of the timecodes and the corresponding classifications into the EDL. As mentioned earlier Viz Story uses REST, so to pipe the information into Viz Story requests are made using the Requests package in python. The finished EDL would then be sent to Viz Story overwriting the existing EDL or creating a new one depending on if a PUT or POST call was made.

4.1.3 Implementation of a publishing target

The final thing that was implemented was a new publishing target. This new publishing target will first download the video locally to the computer. It will then run the downloaded video through the pipeline. To do this the program first needs to find the correct URL for downloading the EDL to the corresponding story. It does this by first downloading an XML document containing a list of all the stories published by Viz Story, as well as their names and corresponding URL's. The names of the stories are then compared to the name of the story that was published to find the correct one. When the correct URL has been found, it is used together with the file that was downloaded to run through the pipeline.

4.2 Solution design

Julia workflow

The whole pipeline will start with Julia reading an MP4. This file will be downloaded by the publishing target. Which will also start the pipeline after the file is downloaded. The audio from the MP4 file will be extracted into a WAV file using a similar command as 4.1, then the WAV file will be read by a package and sampled by running the BIC routine. These samples will each then go to the clustering method where the MFCC-routine will be run and the resulting features are used to extract the i-vectors. The resulting i-vectors will get time-codes attached to them and be clustered according to the algorithm. For each resulting

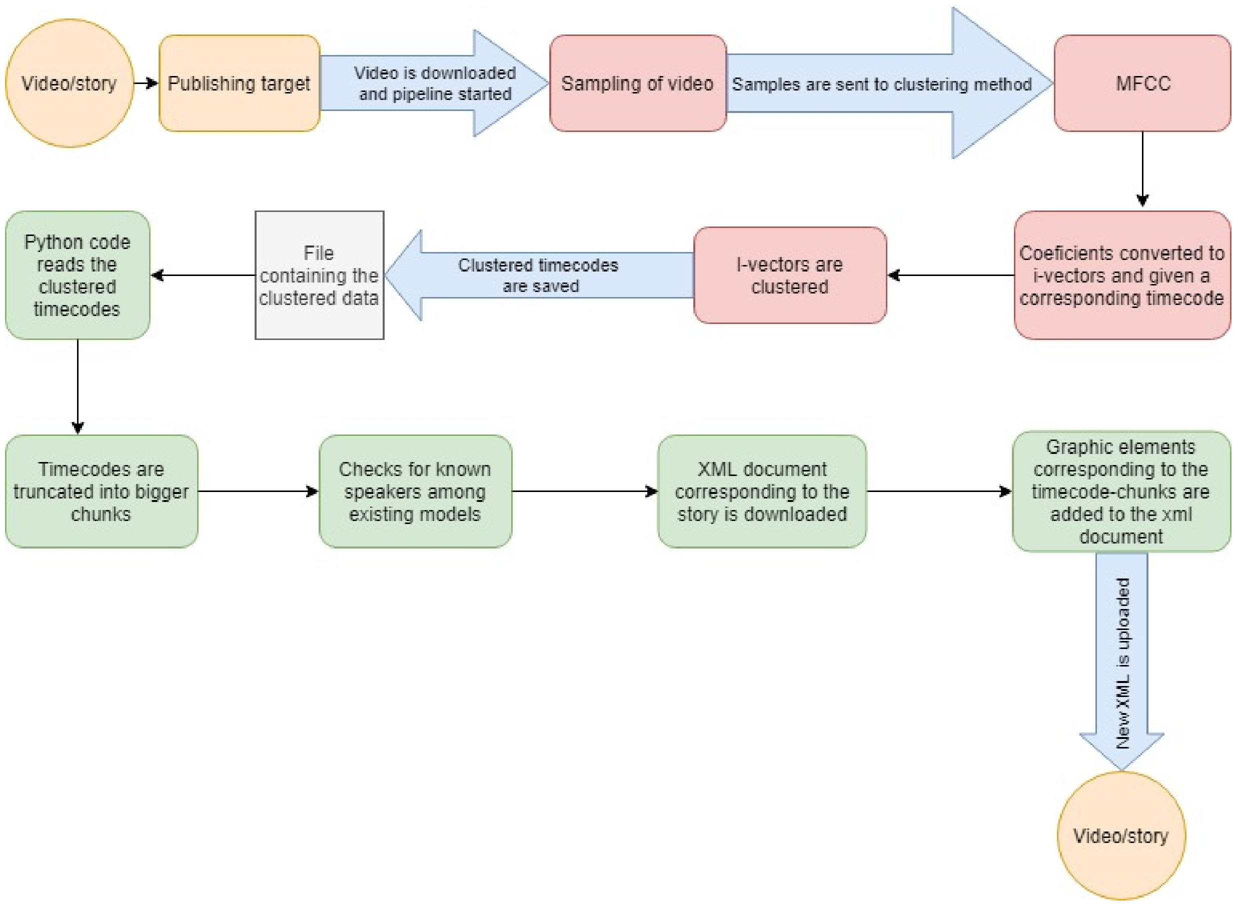

Video/story
Publishing target
Video is downloaded and pipeline started
Sampling of video
Samples are sent to clustering method
MFCC
Coeficients converted to i-vectors and given a corresponding timecode
I-vectors are clustered
Clustered timecodes are saved
File containing the clustered data
Python code reads the clustered timecodes
Timecodes are truncated into bigger chunks
Checks for known speakers among existing models
XML document corresponding to the story is downloaded
Graphic elements corresponding to the timecode-chunks are added to the xml document
New XML is uploaded
Video/story

FIGURE 4.3: Overview of the flow of the julia and python code

cluster, the time-codes will be saved to a text file so the python program can use them in the making of the EDL. The i-vectors will also be saved as a data frame so that the python part of the program can compare them to the known people that are saved in the database.

The program also lets you extract a single i-vector and save it in the database to be used as a model for a known person. In a production version of the pipeline, this would be used to train models for new people, but in the current pipeline, there is no functionality to do this at run time of the main routine.

Python workflow

At the end of the Julia part of the pipeline, either a command to run the Python code will be run, or the Python code will be run from the publishing target. The Python code will load the saved time-codes into a list which will be sent to a function for checking for known persons. The i-vectors will be loaded in a data frame and will be compared to the i-vectors in the database for know people. This comparison is done using cosine-distance. If a i-vector score over a threshold it and its associated time-code will be removed from the list, and added to a new list containing the time codes for people with known models. Both these lists will then be sent to the function that will concatenate the time-codes and returns lists of the new time codes for both lists.

When the time codes are ready its time to start building the EDL. The program will have access to an XML file that will contain a template EDL, this will be loaded in and copies of the elements that will be inserted into the EDL will be made. In the current pipeline, this will be lower-third graphic elements. An EDL associated with the video that was used will also be downloaded. The EDL will be downloaded from a URL that is either given by the user or found through the code in the publishing target. This EDL will then be expanded with elements containing the time codes, and also elements containing the name and time-code for the known person(s) that were found. After the EDL is completed it will be uploaded to Viz Story where it will replace the old EDL or add the new elements to the existing EDL.

4.2.1 Explanation of the user interface

As was pointed out earlier in the report, a big part of this thesis is to be able to edit the results we get from the machine learning before saving it in a system. This is where using the UI is very helpful. After analyzing a video with machine learning, the results will be put in the EDL and sent to the UI. From here one can edit the results. For this version of the system, the results are shown as graphics elements in the UI, so by editing these graphics tags you also edit the results from the machine learning. In a production version of this system, the results would not be shown as graphics elements, but instead as their own type of elements.

How the user interface could be used in combination with the pipeline

When a user is allowed to edit the results it will allow subsequent users to have more confidence in the result and the labels created from those results. Most ML systems will never produce results that are 100 percent correct, likewise, this is the case with this program. Instead of trusting the results completely, this system introduces a form of quality control originating from the ability to edit the results. This again is what will give other users more confidence in the resulting labels. The current Viz Story UI has a lot of functionality that likely would have been in a production version of the system.

Chapter 5

Results

5.1 Presentation of the results

In this chapter the results from the clustering and segmentation algorithms are presented. We will also present the integrated functionality provided by proposed solution.

5.1.1 Qualitative results

When using the proposed pipeline in a video-editing system (namely Viz Story) results in the following being provided to the user:

- Information about when each speaker talks gets presented to the user.
- Possibility to recognize known speakers, and to train models for new speakers one wants the system to recognize in the future.
- Extracting training data based on the segmentation and the clustering.

The code behind these results works well enough as a proof-of-concept. By using Viz Story with the pipeline integrated, one can see that the pipeline delivers adequate results and provides a good start-point for building a production version of the pipeline. As this was only a proof-of-concept more work would be required before one could comfortably use the pipeline in a production system. The clustering and segmentation does not work as well as it could, and the training of new model needs to be fully integrated into the pipeline etc. There are also some specific problems that should be looked at. For example, when comparing the highest score from the T^2-statistics against a threshold, if one puts the T^2 threshold to low and the BIC accepts the cut, one will end up with an unnecessary cut. This can lead two segments that should have been classified as the same speaker, being classified as different speakers. Problems like these could probably be fixed with time through trial end error. These kinds of problems, even if they are not prominent, are things that would have to be fixed before using the pipeline in a production system. There is also more functionality that potentially could have been added. Examples of what these functionalities could include is presented in section 3.2.

5.1.2 Quantitative results

It was never expected that the machine learning algorithm would produce perfect results. When ML is applied to a problem one has to keep in mind that the results will be an approximation of the truth. The proposed solution was therefore expected to make mistakes and gives to users the option to correct these mistakes.

To be able to give a more concrete picture of how well the ML part of the pipeline works, some qualitative results were extracted. To get the result a number of different audio-tracks with increasing length was used. They range from about 1 minute to about 14 minutes in length It was decided to look at the results from the sampling and clustering separately. This was because we were unaware of any measure that could cover both the sampling and the classification done by the clustering algorithm. Even so, these results are supposed to reflect a normal run through the pipeline. Therefore the results from the clustering and sampling will come from the same run, and the results will be extracted from looking at the video-editing UI. The results displayed in the UI is compared with the ground truth(reality) of when the speaker changes to determine if the sampling and classification is correct. One has to keep in mind that the result of the sampling will affect the clustering when looking at the results.

The results from running different audio tracks through the pipeline will be presented in the tables below. Table 5.2 presents the results of the classification done by the clustering, and table 5.1 presents the results of the sampling.

For the sampling the following measures was used

- Total number of cuts.

- Number of excess cuts. This means that if there is a cut without a change in speaker or moment of silence etc, it will count as a excess cut.
- Number of missing cut. This means that if there was a change in speaker without a cut, it will count as a missing cut.
- Number of inaccurate cut. This means that the segmentation did perform a cut, but the time at which the cut was made is wrong.
- Total inaccuracy of cut in seconds. This means that if there is a cut in 56s that should have been in 55s, one second will be added to the total. Keep in mind that this measure can be a little inaccurate, since it is hard to determine the exact time the cut should be in.

For the classification done by the clustering algorithm the following measures were used:

- Total number of segments
- Number of incorrectly classified segments.
- Number of correctly classified segments.

For both classification and segmentation the following parameters were used:

- UBM dimension: 1024
- I-vector dimension: 400
- T^2 threshold: 800.
- BIC is centralized around a frame, and uses +-150 frames around that one.

Both the tests uses the same data, which was gathered in the same way explained in section 4.1.2

5.1.3 Comparing the current pipeline with a production version

As one can understand from reading section 3.2, there are differences between what an imagined production version of the pipeline could possibly achieve, and what the proposed proof-of-concept pipeline can do

Track	#Cuts	#Excess cuts	#Missing cuts	#Total inaccuracy
1	3	0	0	2s
2	1	0	0	0,5s
3	3	0	0	1s
4	5	2	0	0
5	14	6	1	9s

TABLE 5.1: Result of testing the sampling with different audiotracks

Track	#speakers	#incorrect classifications	#correct classifications
1	3	1	4
2	2	0	2
3	3	2	2
4	3	2	3
5	5	8	7

TABLE 5.2: Result of testing the classifications provided by the clustering process with different audiotracks

today. Much of the speculated functionality that could be in a production version would require more time and intimate knowledge of the video-editing system to make and integrate. The production version's functionality could build upon the functionality that is already in the pipeline. It could use the existing functionality to provide the user with more functionality that a user can use directly. Such a system would also likely have a UI that is adapted to using the pipeline. Instead of using the existing elements in the UI in new ways, as the proof-of-concept pipeline does. This would hopefully lead to a better user experience.

Chapter 6

Evaluation and conclusion

In this chapter, the research questions will be answered. The results will be evaluated and discussed how they affect the answers to the research questions. At the end, a conclusion to the thesis answering the research question will be drawn.

6.1 Evaluation and discussion of the results

The evaluation of this thesis will be a self-evaluation of the results from the work done in the thesis.

The results presented in section 5.1.1 and section 5.1.2 are as a whole considered acceptable by the expectations set for this thesis, but are not perfect. As was mentioned when presenting the result, there is still more work that should be put into the pipeline before one can comfortably use it in a production system. All the functionality that was implemented works on a basic level. A user is able to publish their video from the UI and have the the audio-track partitioned according to speakers. From section 5.1.2 one can see that this partitioning does not deliver perfect results. The sampling seems to work better than the clustering. From table 5.1 one can see that the sampling works very well on shorter audio-tracks, but the results does degenerate a little on longer audio-tracks. The number of missing cuts are kept to a minimum, while the number of excess cuts is a little higher than it should be. This was done on purpose, the idea being that having the sampling be a little sensitive would be better than having it be to discriminatory. The idea behind this was that the clustering would concatenate the samples that should not have had cuts in them. Unfortunately the clustering did not perform as well as was expected, with an average accuracy of 0.673% (numbers taken from table 5.2) in the resulting classification. From the result of running track 5 through the pipeline, which had a generally high number of excess cuts, one can see that the clustering had trouble correctly classifying the high number of excess cuts. Which means the segments with excess cuts will not get concatenated into the correct segments. This led to having high number of both excess cuts and incorrect classification. To remedy this, the clustering part of the pipeline should be better. Compared to other solutions that uses the i-vector approach to do speaker recognition, for example[16], the results presented in table 5.2 are subpar. The presented solution does differ in from [16] in its approach to doing speaker recognition(verification), but comparing them we can see that the i-vector approach has the potential provide better results when applied to these kinds of problems. A reason it did not perform so well could be that the measure used in the k-means algorithm was not the best way to compare the i-vectors. Based on what was used in other scientific articles, the original approach of using the cosine distance to compare the i-vector would have delivered better results. To sum it up, the results from section 5.1.2 was not as good as I would have expected, especially the classification. Having a better algorithm for doing the clustering would have greatly benefited the result.

Of course it was expected that the machine learning algorithm would make mistakes. From the start, the idea behind the pipeline was to give the user the ability to edit the mistakes made by the machine learning algorithm. This is done through the UI, and is the backbone for the reason I can say that the results presented in 5.1.2 was good. Without the human quality-control of the partitioning and classification the end results might not be satisfactory. This would lead to much of the functionality being less usable. Extracting training data and building new models based on them would probably not be possible, as the training data and by extension the new models, would be far less reliable if the users just trusted the machine learning algorithm. By having the option to edit the result, the users can use this functionality with the knowledge that they will likely have been corrected by a human.
The idea behind using the functionality provided by the pipeline is to help users with time-consuming tasks. Something that using the functionality would definitely can do. All the end results from using the pipeline could also be achieved by manually labeling the audio-track for the use of extracting training data, or by manually creating metadata for the different stories. Anyone who has done this kind of work will probably tell you that it can take quite a bit of time. So even if the information the user gets from the machine learning is not perfect, it will still save the user time by giving them partially correct information and thereby

doing some of the labeling work for them. Or at the very least, it will give a a general idea of when there is a change in speaker, something that will help if you want to do manual labeling. One can also see how the pipeline will save time if one thinks of how it would be used in a production system by a reporter. Not only will the information be presented to the reporter, but it can help him/her with the editing of the video. If the reporter sees who speaks in certain parts of the video it will be easier to see what parts of the video would be interesting to them. Instead of having to sit through the entire video to find interesting parts.
To sum it up, one can definitely see that the pipeline would does deliver good enough result to help users save time. Even if the result presented could be better, it will still save time by giving partial information on the speakers as well as give the option to easily extract training data for future use.

6.1.1 Utility value of using the pipeline in a video editing system

In the current Viz Story UI, the only functionality that is used in concurrence with this pipeline is the ability to present and edit the results we get from the machine learning and the graphics tags. This functionality is important for the system but leaves a lot of potential functionality unused. The work in this thesis around using the results from the machine learning has focused on displaying the results and making them available to save as labels or time data in the Vizrt system. This is because building a UI for this system would have taken up to much time, but as a result, a lot of potential functionality was left unexplored. When the results from the machine learning can be edited in the UI, you also introduce the potential to use these results in the editing process. This could be in the form of skipping to certain speakers or removing instantly unwanted speakers from the video.

6.1.2 How the pipeline could affect the workflow of an editor

Much of the functionality the proposed pipeline provides is to benefit the end user of the editing system. In this thesis, the end user has often been imagined to be a journalist or someone editing news or debate programs. When these kinds of users takes advantage of a system that has a pipeline like the one proposed integrated, it will affect the way they use that editing system. Firstly it would make it easier for them to find out who is in the different videos they are editing. This will make it easier for them to find relevant information or clips to use. All the training data that the pipeline has extracted could also be made available to the user. By doing this the editor would have access to a big library with clips they could use when making reports on specific people, instead of the editor having to search for relevant clips to use of that person. The editor would also get the information about where the different speakers are in the video from the diarization, and who these speakers are from the classification. This could give the editor a quick overview of what the video contains even before looking through it. As well as providing them with graphic elements for the different speakers if this is desired. The combined functionality mentioned would make the workflow for the editor simpler and quicker, even if the editor has to spend a little time correcting eventual mistakes made by the machine learning part of the pipeline.

6.2 Evaluation of own work

Since this was my first time working on this type of project and my first time working with this kind of technology, there are very likely things that could have been done to improve the proposed solution. The first decision that I would have done differently if I could do it over is the decision not to use d-vectors. In the early phases of the thesis, it was decided that this thesis would not use neural-networks, or by extension, deep-learning. As a result, this excluded the use of d-vectors. Looking back I am unsure if this was a good decision. The decision was based on an uncertainty of how much data would be available, but with the amount of data that was available a deep-learning network could very likely have been trained. Of course it is not certain this would have delivered better results, but I assume that the d-vectors would give better results than the i-vector since it is a newer type of model, and deep-learning tends to deliver good results.

When it comes to programming languages, when doing this kind of project again I would take care of keeping most of the code in one language. The current program is split between Julia and Python. This works, but makes the code base really messy and hard to both read and maintain. The choice to use different languages was mostly done because of the different packages available, and the fact that maintainability was mostly ignored when writing the code. Still, when working with the code it would have made the work easier if the code was better structured and only in one language. Overall taking care to write better code should have been a higher priority during the work of this thesis.

During the early to middle stages of the thesis, some new information that was useful to the thesis came to light. Among them was the term diarization and the field surrounding it. While it was good that this was not discovered too late, it would have been better if it was discovered at the start of the work on the thesis. So looking back, doing more research before starting work on the implementation would have been a good idea. The project was not greatly impacted by the lack of research, but could have benefited from a better literature study in the early stages.

6.3 Conclusion

This thesis presents a pipeline that:

- Separates different speakers
- Classifies the speakers
- Presents the information to the user and lets the user edit it

It also has the functionality to extract training data based on the information, as well as the functionality to use this training data to train new models. Additionally the thesis proposes ways to use this functionality in a production system. The proposals are built around the idea that the users are TV-reporters or journalists, and how the pipeline would benefit those kinds of users. From this one can see how the pipeline can be beneficial not only to those kinds of users, but will also help a general user with navigating the video during editing. Based on this, I have reached the following conclusion to the research questions.

Question 1
In this work we aim to try to make a pipeline that will separate the voices of speakers in an audio-track by applying machine learning, and pipes this information into a system. By this, answer the question of; can machine learning be applied to partition an audio-track according to speakers without any prior trained models, and the result be piped into a system to train new speaker models?

Yes, one can clearly build a pipeline that covers the functionality in the research question. Even if the the results from the proposed solution were not perfect, it has been demonstrated that it is very much possible. The proposed pipeline is only one of a number of different possible solutions. There are different solutions that can achieve similar results. I can also conclude that using the proposed pipeline as a proof-of-concept could be a good starting point for future developers that wish to build a similar solutions.

Question 2
This thesis also aims to look into the possibilities of integrating machine learning into a video-editing system. Then answer the question of how a machine learning system be integrated into a video-editing system in a way that will help the users with time-consuming tasks?

The pipeline presented is one possible solution to how to integrate machine learning into a system. From the functionality provided by the pipeline I can also conclude that the proposed pipeline would help with time-consuming tasks. There are still functionalities and improvements than can be made to the pipeline that would improve the help provided with time-consuming tasks. From what is present in the pipeline I can conclude that it will at least reduce the time spent on the following tasks:

- Navigating through the video, and thereby help with editing
- Generating training data
- Gather information about who is present in the video

So in conclusion, the thesis proposes a pipeline, subsisting of a machine learning program, which can help users with a number of time-consuming tasks. The pipeline has the potential to be build upon to make it even more effective.

6.4 Future Work

The proposed solution in this thesis is only meant to be used a proof-of-concept, and there is a lot of work that can be done on it in the future.

6.4.1 Improving the algorithms

The algorithms used in the proposed solution would very likely have to be improved before being used in a production version. First and foremost steps would have to be taken to improve the results as much as possible, but there are also other improvements that could be done. As it stands, you have to tell the clustering algorithm how many clusters you want it to produce. There are clustering algorithms that can decide this without input, and it would benefit the pipeline to take advantage of such an algorithm.

6.4.2 Expanding functionality

The functionality implemented in the solution could potentially be expanded. When comparing what was speculated in section 3.2 and what the solution offers one can see that there is much that is not yet implemented in the pipeline. The functionality talked about in section 3.2 were of course only speculations and there has not been any research done on how to implement them, but in my opinion, the potential to implement them is there. This would most likely require that the video-editing system be adapted to facilitate the new functionality.

6.4.3 Build or adapt a UI

This would also be important if one wants to expand the functionality. Building a custom UI or adapting the UI of the current editing system would let you take better advantage of what the pipeline provides. What such a UI would have to contain would depend on what functionality one would want the pipeline to provide. This could, for example, be buttons to skip to the next speaker or a search bar for searching for speakers in the media library.

Bibliography

[1] Vizrt. *About us*. URL: http://www.vizrt.com/company/.

[2] John HL Hansen and Taufiq Hasan. “Speaker recognition by machines and humans: A tutorial review”. In: *IEEE Signal processing magazine* 32.6 (2015), pp. 74–99.

[3] Yi Liu et al. “Investigating Various Diarization Algorithms for Speaker in the Wild (SITW) Speaker Recognition Challenge.” In: *Interspeech*. 2016, pp. 853–857.

[4] Candemir Toklu and Shih-Ping Liou. *Method and system for video browsing and editing by employing audio*. US Patent 6,697,564. 2004.

[5] R. Port. *Digital Signal Processing Overview*. URL: https://www.cs.indiana.edu/~port/teach/541/sig.proc.html.

[6] Haytham Fayek. *Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between*. URL: https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html.

[7] Thair Khdour et al. “Arabic Audio News Retrieval System Using Dependent Speaker Mode, Mel Frequency Cepstral Coefficient and Dynamic Time Warping Techniques”. In: 7 (June 2014), pp. 5082–5097.

[8] Steven B Davis and Paul Mermelstein. “Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences”. In: *Readings in speech recognition*. Elsevier, 1990, pp. 65–74.

[9] Suman Saksamudre and Ratnadeep Deshmukh. “Comparative Study of Isolated Word Recognition System for Hindi Language”. In: *International Journal of Engineering Research Technology* 4 (July 2015), pp. 536–540. DOI: 10.17577/IJERTV4IS070443.

[10] Nickolay Shmyrev. *Why is pre-emphasis (i.e. passing the speech signal through a first order high pass filter) required in speech processing and how does it work?* URL: https://www.quora.com/Why-is-pre-emphasis-i-e-passing-the-speech-signal-through-a-first-order-high-pass-filter-required-in-speech-processing-

and-how-does-it-work/answer/Nickolay-Shmyrev?srid=e4nz&share=71ca3e28.

[11] Siemens Phenom. *Windows and Spectral Leakage*. URL: https://community.plm.automation.siemens.com/t5/Testing-Knowledge-Base/Windows-and-Spectral-Leakage/ta-p/432760.

[12] URL: http://www.practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/.

[13] T. NATARAJAN N. AHMED and K. R. RAO. *Discrete Cosine Transfonn*. URL: http://dasan.sejong.ac.kr/~dihan/dip/p5_DCT.pdf.

[14] Daniel Faggella. *What is Machine Learning?* URL: https://www.techemergence.com/what-is-machine-learning/.

[15] Santosh K Gaikwad, Bharti W Gawali, and Pravin Yannawar. "A review on speech recognition technique". In: *International Journal of Computer Applications* 10.3 (2010), pp. 16–24.

[16] Najim Dehak et al. "Front-end factor analysis for speaker verification". In: *IEEE Transactions on Audio, Speech, and Language Processing* 19.4 (2011), pp. 788–798.

[17] Douglas Reynolds. "Gaussian Mixture Models". In: *Encyclopedia of Biometrics*. Ed. by Stan Z. Li and Anil K. Jain. Boston, MA: Springer US, 2015, pp. 827–832. ISBN: 978-1-4899-7488-4. DOI: 10.1007/978-1-4899-7488-4_196. URL: https://doi.org/10.1007/978-1-4899-7488-4_196.

[18] Douglas A Reynolds, Thomas F Quatieri, and Robert B Dunn. "Speaker verification using adapted Gaussian mixture models". In: *Digital signal processing* 10.1-3 (2000), pp. 19–41.

[19] Douglas A Reynolds. "Comparison of background normalization methods for text-independent speaker verification". In: *Fifth European Conference on Speech Communication and Technology*. 1997.

[20] Ehsan Variani et al. "Deep neural networks for small footprint text-dependent speaker verification". In: May 2014, pp. 4052–4056. ISBN: 978-1-4799-2893-4. DOI: 10.1109/ICASSP.2014.6854363.

[21] Douglas Reynolds. "Universal Background Models". In: *Encyclopedia of Biometrics*. Ed. by Stan Z. Li and Anil K. Jain. Boston, MA: Springer US, 2015, pp. 1547–1550. ISBN: 978-1-4899-7488-4. DOI: 10.1007/978-1-4899-7488-4_197. URL: https://doi.org/10.1007/978-1-4899-7488-4_197.

[22] Universitat Politècnica de Catalunya. *Speaker ID I (D3L3 Deep Learning for Speech and Language UPC 2017)*. URL: https://www.slideshare.net/xavigiro/speaker-id-d3l3-deep-learning-for-speech-and-language-upc-2017.

[23] Gregory Sell and Daniel Garcia-Romero. "Speaker diarization with plda i-vector scoring and unsupervised calibration". In: *2014 IEEE Spoken Language Technology Workshop (SLT)* (2014), pp. 413–417.

[24] Scott Chen, Ponani Gopalakrishnan, et al. "Speaker, environment and channel change detection and clustering via the bayesian information criterion". In: *Proc. darpa broadcast news transcription and understanding workshop*. Vol. 8. Virginia, USA. 1998, pp. 127–132.

[25] Theodore Wilbur Anderson. *An introduction to multivariate statistical analysis*. Tech. rep. Wiley New York, 1962.

[26] Jitendra Keer. *Developed a system for identifying which person speaks at what interval of time in an Audio file, Speaker Diarization*. URL: https://angel.co/projects/302267-developed-a-system-for-identifying-which-person-speaks-at-what-interval-of-time-in-an-audio-file-speaker-diarization.

[27] *Julia language homepage*. URL: https://julialang.org/.

[28] *Julia Micro-Benchmarks*. URL: https://julialang.org/benchmarks/.

[29] Vizrt. *Viz Story*. URL: https://www.vizrt.com/products/viz-story.

[30] *EDIUS Pro 8 Non-Linear Video Editing Software (Windows)*. URL: https://www.fullcompass.com/prod/288294-grass-valley-edius-pro-8-edius-pro-8-non-linear-video-editing-software-windows.

[31] W3C. *INTRODUCTION TO ATOM*. URL: https://validator.w3.org/feed/docs/atom.html.

[32] IBM. *Overview of Atom feeds*. URL: https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.5.0/fundamentals/web/dfhtl_atom_whatis.html.

[33] Sumit Gupta. *Deep learning performance breakthrough*. URL: https://www.ibm.com/blogs/systems/deep-learning-performance-breakthrough/.

[34] David van Leeuwen. *A Julia package for Gaussian Mixture Models (GMMs)*. URL: https://github.com/davidavdav/GaussianMixtures.jl.

[35] Sveriges Riksdag. *Sveriges Riksdag webb-tv*. URL: http://www.riksdagen.se/sv/webb-tv/.

[36] *FFmpeg homesite*. URL: https://ffmpeg.org/.

[37] Julia DSP. *Mel Frequency Cepstral Coefficients calculation for Julia*. URL: https://github.com/JuliaDSP/MFCC.jl.

[38] Jair Ferreira et al. "Driver behavior profiling: An investigation with different smartphone sensors and machine learning". In: *PLOS ONE* 12 (Apr. 2017), pp. 1–16. DOI: 10.1371/journal.pone.0174959.

[39] David van Leeuwen. *i-vector training, extraction and scoring routines*. URL: https://github.com/davidavdav/IVectors.jl.

[40] Andrea Trevino. *Introduction To K-Means Clustering*. URL: https://www.datascience.com/blog/k-means-clustering.

[41] Andrea Trevino. *lxml - XML and HTML with Python*. URL: https://lxml.de/.