Pattern Annotation and Classification in Broadcast Content of Radio Productions

Tom Valk

SID: 3308180027

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Data Science

2nd December 2019
THESSALONIKI – GREECE

Pattern Annotation and Classification in Broadcast Content of Radio Productions

Tom Valk

SID: 3308180027

Supervisor: Lecturer Rigas Kotsakis

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Data Science

2nd December 2019
THESSALONIKI – GREECE

Abstract

In the current digital century, there are plenty of radio stations to choose from. However, the choice usually is only based on the music genre, and the listener has to recognize if the program, schedule, and amount of talking suits their demands. In order to compare the amount of music/talking on a radio station, it could either be compared manually by listening, although, in modern times, this could also be automated by the usage of machine learning. This study concentrates on the recognition of speech and non-speech on their patterns by using radio productions as input and optimizing the extraction of numerical values, algorithms, and methods to combine and precise the accuracy over distinguishing the different categories and labels. The distinguishing is achieved by using knowledge from earlier research and combining modern newly introduced technologies and ideas, the paper experiments with a multi-layer classical machine learning setup. The numerical extraction from the audio input is executed with the usage of existing research and technologies from the digital signal processing and audio processing fields in combination with optimized parameters.

Based on the literature review, the experimental setup extracts a set of features from the audio tracks, which are manually labeled to create ground truth label data. The experiments are covering three algorithms and will compare not only the algorithms but also the methods of extracting by tuning the hop and window sizes. Furthermore, two algorithms in the multi-layer setup are being parameter tuned using grid-search methods to result in an optimal setup specialized on the numerical data.

The results indicate that the numerical extraction and the decision between the hop and window size is one of the most critical parameters. Furthermore, the results indicate that both MLP and XGBoost are very good in performance and show both similar results with negligible differences. Further research and experiments are demanded to optimize and increase the performance of the models by, for example, focusing on silence periods and reducing the impact of background noise on the performance.

Acknowledgements

I would like to thank my supervisor, Prof. R. Kotsakis, for his great and patient guidance and motivation through the early stage of the research, experiments, and writing of the dissertation. It has been a pleasure working with my supervisor and having this very interesting and challenging subject for my dissertation. I would also thank the school and university staff that helped me through my master program and especially the research. Also, in my process of the research, I would like to thank Ms. G. Roidouli for providing the academic resources through the library of the university. I would also like to thank my classmates and co-students for helping me out with ideas and support when necessary.

Furthermore, I would like to thank all my friends, here and abroad and of course, my beloved family, for all the support during the period of my research and writing.

Contents

List of Figures

List of Tables

Listings

1 Introduction

In nearly every populated areas, the FM-bands are well filled with radio stations of every kind. This makes it challenging for users to make a preferred choice. It could be possible that one individual does not prefer a specific station because of the music genre or, more particularly, the shows or number of advertisements that are present at the station.

For the audience, it is tough to find the correct and perfect station that fits well at specific times of the day. Perhaps the listener desires to listen to calm down and does not fancy to be listening to the talk-shows one particular station puts on at a specific time. Making decisions based on these reasons requires to know the programs, shows, and events that are occurring on the stations.

The dissertation presents the work of distinguishing given radio streams between speech and non-speech based on recognizing patterns within the obtained numerical values.

1.1 Problem definition

In modern times, we have plenty of choices concerning radio stations and our music tastes. Nevertheless, the aforementioned extensive number of options causes making a single selection more challenging, and for one music taste, there might be several radio stations that suit the wants of a particular person. Although it might be simple just to put on a radio station, there are still differences among some stations concerning advertisements and how much talking is going on besides playing music. There is not a way to differentiate radio stations with a simple metric of how much the ratio of music/speech is.

In order to compare with the metric mentioned earlier, one could attempt to identify and denote the speech and non-speech on a radio station by human work, but, in modern times, this could also be achieved automatically.

Earlier research provided some insights regarding radio production pattern recognition based on older techniques. However, in modern times, the techniques used for this have been improved, and the field of machine learning is more complete and more accessible to apply comparing to the earlier days.

Because it is an audio problem, and audio does not directly have numerical values that could be applied in a machine learning algorithm, it is not merely a problem with given numerical data. The problem of identifying, deriving numerical data, and coping with the fact that the initial audio is compressed audio is a substantial challenge. Mainly the challenge is to extract the appropriate numerical data, with the suitable intervals, and therefor use the appropriate machine learning algorithm with the best parameters according to the given numerical values and extraction methods.

The global problem is to discriminate speech and non-speech based on patterns in radio productions with various experiments in a manner for further research and building applications to produce and apply a metric for comparing radio stations based on the speech and non-speech ratio.

2 Related work

Earlier research was done by the Department of Computer Science and Engineering of the National Institute of Technology Karnataka (NITK) in India. The research from 2015 was done with the goal of identifying advertisements in real-time radio productions. The paper describes the usage of different kinds of models like the Hidden Markov Models, Artificial Neural Networks, and Ensemble Method for the classification of the audio on advertisement patterns. In the end, the paper concludes, for the specific situation, the usage of an ensemble model is the best according to the performance metric. Furthermore, the paper shows a visualization of the distribution of advertisements per hour of the day generated with the data obtained from the output of the model. One of the primary observations in the paper is that recognizing patterns to distinguish advertisements on radio productions can be performed on a real-time basis in a particularly efficient and effective way by using MFCCs for extracting numerical values. The paper also analyses the behavior of advertisement speech and concludes that the pitch and speaking rate of a speaker is higher in advertisements due to the fact the air-time is expensive, and the advertisers attempt to bring as much information as possible with talking faster. Furthermore, the paper describes the usage of prosodic features in order to distinguish between the two types of speech. The research also shows the importance of an algorithm that can handle a high noise tolerance due to the nature of the input audio data [1].

Another study is executed at the Computing Department of the University of Colombo in Sri Lanka with the purpose of audio monitoring of radio broadcasts across the country of Sri Lanka. The aforementioned research intended to produce a list with songs per radio station and to recognize the played songs from the audio streams to make sure the rights of the intellectual property and their proceedings are being paid to the artists of the music. In the preprocessing steps described in the paper, the researchers first strip-off the speech and advertisements in order to only focus on the music. The rest of the experiment is a content-based audio identification algorithm that recognizes songs from parts of audio given. The paper describes a way to store fingerprints of songs in a database with an efficient searching algorithm. This algorithm will provide an approximate matching, which is enough for the experiment. Songs that are not yet known can be registered in the database, and existing songs will get matched from the input audio. The extraction process is executed with creating and processing similarity hashes from specific window ranges on the audio [2].

Another research executed by Rao from the Department of Electrical Engineering at the Indian Institute of Technology in India has shown some insights in extracting numerical values from audio signals and processing the signals. The chapter of the book describes the characteristics of audible sound and human hearing. Furthermore, the chapter describes the representation of audio signals and the difference between time-domain and frequency-domain representations. Part of this includes the description of the spectrogram and how the Fourier transform is part of the transformation from time to frequency representation. Furthermore, the chapter describes methods to extract numerical values from the audio signal and from the different representations described earlier. The paragraphs about the numerical extraction have detailed descriptions about the two categories as well as the short and long term, also called static and dynamic methods of extracting. This is executed with the feature temporal analysis, where the features are summarized based on a more extensive length. The chapter of the book describes features like the Zero-Crossing Rate, Short-Time Energy, Band-Level Energy, Spectral Centroid, Spectral Roll-Off, Spectral Flux, Fundamental Frequency, and the Mel-Frequency Cepstral Coefficients which are well-known in the field. Furthermore, the chapter also researches speech-music discrimination, where the literature review concludes that the usage of some features is essential. For example, speech itself will result in a high energy and have mostly low-frequency contents. This behavior leads to a higher variation in the Zero-Crossing Rate output, as well as in some spectral features due to the shape of the spectral representation [3].

Research done by Scherer, Schwenker and Palm from the Institute for Neural Information Processing at the University of Ulm in Germany about emotion recognition from speech shows some insights into the usage of specific features for only speech. The chapter in the book describes the usage of Linear Predictive Coding and Mel-Frequency Ceptstal Coefficients in the progress of recognizing emotions in speech audio fragments. Besides those two features, the chapter also describes the usage of energy and pitch features. It is not straightforward to just obtain the pitch period from an audio fragment because a lot of different methods and definitions have been proposed over the years. The proposed multi-classifier system works, as expected, with multiple classifiers to increase the total accuracy of the system. The proposed system consists of K-Means, Learning Vector Quantization (LVQ), and Radial Basis Function (RBF), all assembled in the multi-classifier system with fusion. The fusion progress is done by the Decision Templates (DT) method. In the end, the chapter describes how the results are showing that the later added MFCC features yielded the best results for the emotion discrimination [4].

Another research focused on voice recognition has been done by Muda, Begam and Elamvazuthi from the Department of Electrical and Electronic Engineering at the Universiti Teknologi PETRONAS in Malaysia. The research paper shows insights into feature extraction and the optimization for voice recognition usage. The paper explains the usage of MFCCs in the extraction process with the steps and detailed text about the framing, windowing, and mel-filtering. Furthermore, the Dynamic Time Warping (DTW) algorithm has been used and is well described in the paper. The algorithm is for measuring the similarity between two given time series, which may differ in time or speed. The technology is used to determinate the optimal alignment between two time-series when one of the two might be "warped". The algorithm uses this and measures the warping in order to compute the similarity. Furthermore, the paper describes differences in male and female voices with displayed figures with visualized extracted MFCC features [5].

The paper published by McFee et al. describes the open-source library librosa version 0.4.0. The author of the paper is the main contributor of the open-sourced library and works for the New York University in New York, United States of America. The paper describes the goal of the library, the background, and the technical design goals when developing the library. The goal of the library is to provide an alternative for the Music Information Retrieval (MIR) field beside the usage of MATLAB. The language, and in specific, some packages for MATLAB, are popular in the relatively young field of MIR. However, the downside is that the performance of the implementations to process and retrieve information from audio is quite bad in the MATLAB and MIRToolbox implementation. Recently there is more interest in moving the MIR field towards Python, and because of a lack of similar libraries or tools, the author of the paper started a project himself which can be called librosa now. The goal is to make a flexible, high-level, and fast library in which basic and advanced audio processing and information retrieval could be done. The implementation is mainly done with the background of the libraries scipy and numpy, which efficiently provide mathematical and statistical implementations by using C implementations, which are very low-level and without the overhead. The paper describes the initial architecture of the library in which the team decided to use a flat package layout in order to allow the low-barrier entry from former MATLAB users. The paper describes the implementation of loading audio files in Python with librosa and the resampling that is done with this task, but optional. Furthermore, the paper describes the idea behind the data structures and the implementations of some core functions such as *load*, *resample*, and *to_mono*. Also, the paper describes the libraries' implementation for the conversion of the time-domain towards the frequency-domain by using the Short-Time Fourier Transform with the *stft* function. Later in the paper, the methods of extracting numerical features are described by mentioning some features such as the MFCCs and some other spectral features. Furthermore, the paper describes methods to visualize audio fragments in different representations such as the melspectrogram with the usage of MFCCs as well as some other visualizations with, for example, the waveform [6, 7].

Another paper on an implementation has been published by from the University of Jyväskylä in Finland describing the library MIRtoolbox for MATLAB. The toolbox is being used by a broad community consisting mainly of users from the MIR field. The paper describes in detail the musical features to be able to extract with provided functions. Furthermore, the paper also covers some specific methods on loading, but especially on how to retrieve features from the loaded audio file. Some features described include the well-known MFCCs and estimating the tempo. The paper describes in detail how reusable components are used in order to make almost every feature extraction possible with doing intermediate tasks only a single time [8].

The paper published by Eyben et al. from the Machine Intelligence & Signal Processing Group of the Technische Universität München in Germany describes the multimedia feature extraction toolkit called openSMILE. The paper describes some ideas behind the toolkit and some of its notable facts, such as the programming language in which openSMILE is implemented, C++. Furthermore, the paper describes the availability of popular audio feature extraction methods for MFCCs, chroma, CENS, loudness, voice quality, local binary pattern, color, and optical flow histogram. The implementation is available for Unix and Windows platforms and has a modular architecture [9].

2.1 Earlier research

This subsection describes earlier research on the topic of distinguishing speech and non-speech and is specifically related to the current dissertation.

The chapter written by Alexandre et al. from the Department of Signal Theory and Communications of the University of Alcalá in Spain in the book Speech, Audio, Image and Biomedical Signal Processing using Neural Networks describes the usage of speech and non-speech classification in hearing aids by using tailored neural networks. The chapter explores the ability to improve hearing aids by using tailored neural networks in order to distinguish and classify audio signals into either speech or non-speech categories. In the specific situation of searching for a possible solution for hearing aids, the chapter makes use of the tailored neural network in order to meet the requirements of low battery consumption by lowering the computational complexity but keeping the benefits of using neural networks. In order to achieve this, the chapter describes the usage of a tailored neural network in which the model is optimized for both performance and a lower computational complexity. According to the chapter, only 20% of the hearing impaired people who could benefit from buying a hearing aid actually purchased it. Furthermore, 25% of them do not wear them because of irritating and unpleasant problems related to the background noise appearing elsewhere in their everyday life. The problem is not only related to economic reasons. Unfortunately, modern hearing aids still lack an automatic adaptation to change the acoustic environment. The chapter describes in detail the hardware limitations of hearing aids such as the complexity which is required when adding machine learning algorithms in them and how to handle the extra battery requirements. Furthermore, the chapter describes the usage of the two-class system instead of a multi-class system. The reason behind this is that it is more troublesome when different classes apply at the same time, such as speech in a noisy background. The usage of a two-class system results theoretically in less wrongly classifications, and the chapter contains a visualization of this problem in figure 4 on page 151. Additionally, the chapter of the book describes the selection and extraction of features where it explains beside the usage of the spectral centroid also the usage of voice2white, a feature which measures the energy within the human speaking band of 300-4000 Hz respect to the total energy of the signal in the frame. Besides those also the spectral flux and the short time energy is used. The short time energy is defined as the mean energy of the signal within each analysis frame consisting of a number of samples. Furthermore, the chapter describes the used machine learning implementation by using the neural network classifier Multi-Layer Perceptron (MLP). In conclusion, the chapter describes the result of an error rate of 9,5% when using a small neural network with only two hidden neurons, which is within the maximum sustainable computational resources of the hardware capabilities on the hearing aid [10].

Another research has been executed in the same field of the current dissertation by Kotsakis, Kalliris and Dimoulas from the Laboratory of Electronic Media at the Aristotle University of Thessaloniki in Greece. The paper shows a similar interest in distinguishing between speech and non-speech with a more specific interest in the actual person behind a specific voice and the actual music genres. The paper describes the challenge to distinguish audio in radio productions because of the nature of the audio, the existence of background noise, and the non-stop and continuous flow of audio without easily distinguished pauses. The paper describes an experiment with the usage of a radio production podcast from a Greek radio station recorded in only a specific time range of a day to only cover a specific program. Furthermore, the experiment is described to consist of the following implementation steps: audio signal preprocessing, pattern definition rules and ground truth acquisition, feature extraction, feature selection, artificial neural system training, and finally, the performance evaluation. The preprocessing step consisted of uncompressing the earlier compressed audio and converting from stereo to mono based on the original source online. The window used to extract features has a constant length of 1 second. Furthermore, the ground truth consisted of different voices, telephone voices, and different music genres, and the annotation has been done manually. The step of feature extraction consists of the following features: Mean Signal Envelope, RMS Energy, Audio Low Energy, Average Attack Time & Slope, Number of Signal Peaks, Entropy, Zero-Crossing Rate, Audio Signal Autocorrelation, Roll-off Frequency, Average Event Spectral Density, Fundamental Frequency, Inharmonicity, Spectral Centroid & Spread & Skewness & Kurtosis & Flatness & Roughness & Irregularity & Brightness, Audio Spectrum Fluctuation, 13 Mel-frequency Cepstral Coefficients. From all the features, only a selection of 23 features was made after applying dimension reduction techniques. The experiment consisted out of different model designs with different kind of multi-classifier systems [11].

3 Literature Review

The literature review contains information about the background of the problem and explains specific technologies used in the experiments.

3.1 Digital audio

When we talk about audio and our human hearing capabilities, we can conclude that the aspects of audio are analog at the moment we hear things by the human body. However, to store and process audio signals digitally, there is a need to process the analog signal into a digital signal and eventually having methods to store and contain the signals so that it can be processed further and reproduced in an analog way.

The analog audio signals the human beings can hear are situated in the range from 20 Hz (0,02 kHz) to 20.000 Hz (20 kHz) [12, 13]. However, most microphones and digitization methods capture and process a more extensive range of audio frequencies. When processed by a microphone, the signal is converted into a digital signal.

3.1.1 Pulse-Code Modulation

The Pulse-Code Modulation technology was introduced by Alec Harley Reeves in 1937, and the technology was granted a patent in various countries in the years following the introduction [14]. Nowadays, PCM has several variations. However, in general, the standard will use the considerably same steps and methodology.

Sampling

Sampling of the audio signal is one of the steps in the process of digitalization of audio in the PCM standard. During this step, the captured signal is divided into a given number of samples per second [15]. The optimal value for the parameter of the number of samples per second can be obtained using the Nyquist rate theorem. The Nyquist rate is named after the Nyquist criteria by the engineer Harry Nyquist and is sometimes also described as the Nyquist sample rate [16]. The theorem of the Nyquist rate states that the sampling rate should be at least double the highest frequency in the original audio signal to be able to reproduce the original signal [17].

$$sr \geq 2 \cdot f_{max} \tag{1}$$

Where sr represents the minimum sample rate

The most commonly used sampling rate is 44,1 kHz, which will capture and store the most common frequency ranges. Based on the given sample rate and the method of sampling, let's say 41,1 kHz, the signal is being split up by 41.100 data points per second.

Table 1: Computer Audio Sampling Rates [15]

Sampling rate	**Description**
8 kHz	The G.711 telephony standard
16 kHz	Used by the G.722 compression standard
18,9 kHz	CD-ROM standard
22,05 kHz	Half the CD sampling rate
24 kHz	One-half 48 kHz rate
32 kHz	Used in digital radio
44,1 kHz	The CD sampling rate
48 kHz	The standard professional audio rate. Used for DVD and DAT and some digital video streaming.

Quantization

Sampling the audio is just one step in the process. Following this step, the process of quantization starts. In the quantization step, the sound waves, and therefore the sample data points, will get a specific range based on the bit-depth given as a parameter. For less demanding and typical applications, this would be 16-bits, while professional users and applications would demand a higher resolution of 20 or 24-bits [15]. When using 16-bits, there will be 65.536 possible levels to divide frequencies per sample and to represent the level of one single data point.

Encoding

The encoding process of PCM is different for the flavor and implementation of PCM used. However, the basics are nearly the same and consist of saving the processed samples with the resolution given in sequence to a file containing metadata on how to process and decode the given sample stream.

3.1.2 Short-Time Fourier Transform

The Short-Time Fourier Transform uses the Discrete Fourier Transform (DFT) over short periods with overlapping windows to represent a time-frequency domain signal [18]. The mathematical definition of the STFT is as follow [19]:

$$X_m(\omega) = \sum_{n=-\infty}^{\infty} x(n)w(n - mR)e^{-j\omega n}$$

$$=$$

where

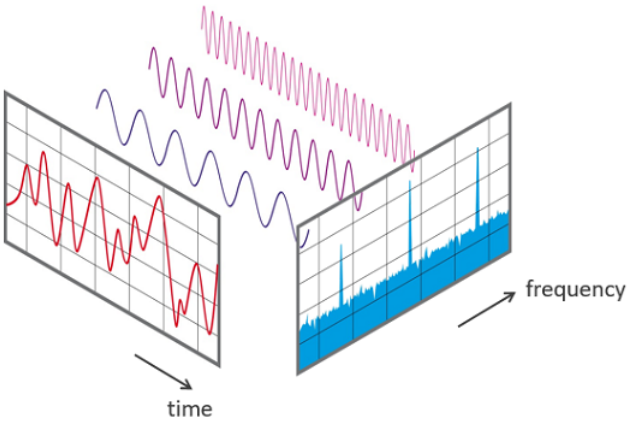
$x(n) =$ input signal at time n

$w(n) =$ length M window function (e.g., Hamming)

$X_m(\omega) =$ DTFT of windowed data centered about time mR

$R =$ hop size, in samples, between succcessive DTFTs.

Figure 1: Overview of the audio signal in time and frequency domain [20]



3.2 Audio compression

Uncompressed audio is large in size, and therefore, compression is used to transfer and store audio in an efficient way to reduce disk and bandwidth usage.

There are two major categories of audio compression, first of all, there is the category of lossless compressions where there will be no loss in the signal after decompression an compressed signal. The signal which will be produced by the algorithm when decoding will be exactly the same as before compressing. This method is excellent for efficiently storing original signals without losing any information.

Secondly, there is the category of lossy compressions which will not produce the same signal after decompression. However, most compression algorithms try to prevent any hearable difference in the signal after decompressing. Some algorithms do this by reducing the frequencies stored or apply a specific technology such as the masking effect.

3.2.1 Lossless compression

Lossless compression is used in situations where the loss of data is not desired. For example, in storing raw audio before processing or when storing recorded music from a live performance before further distribution. The decompressed audio will be the audio that is originally input to compress the data without any change or loss of signals.

Linear predictors

One of the popular and open formats used for lossless compression is FLAC, which stands for Free Lossless Audio Codec, and it might be the first entirely free audio codec [21]. The FLAC format uses linear prediction for compressing. In fact, FLAC uses four methods for predictions and can produce a significant decrease in file size of an audio file [22].

Combinations of predictors

The introduction of MPEG-4 Audio Lossless Coding did introduce the combination of short-term predictor, just like in FLAC, and the long-term predictor [23]. The long-term predictor improves the compression for sounds with rich amounts of harmonics present in musical instruments and in the human voice.

3.2.2 Lossy compression

On the other hand, there is lossy compression, which will result in a loss of data when compressing. This does not necessarily mean that the audio will be in poor quality or to lose audio data that is important for the hearing. For instance, MP3 (formally MPEG-1 Audio Layer III or MPEG-2 Audio Layer III) format uses the masking effect to eliminate frequencies that a human will not be able to hear because of a previous louder signal [24]. This results in the same result when listening to the audio but reduces the data size significantly.

Masking effect

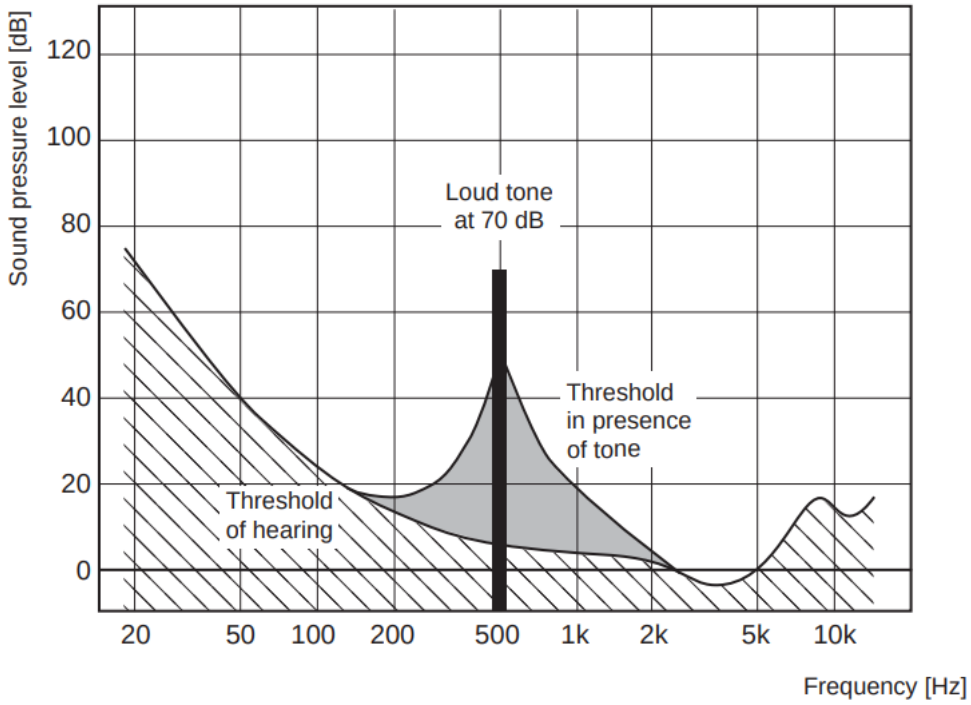
The American physicist Alfred M. Mayer introduced new facts about humans not being able to hear a tone that is significantly lower than the previous tone [25]. Later in 1959, Richard Ehmer introduced a complete set of auditory curves of the theory presented by Alfred M. Mayer [26]. The two discoveries were the foundation for the technology used nowadays, called the masking effect, which ended up in the MP3 compression.

Figure 2 presents the masking effect when a loud tone of 500 Hz is produced. The area in gray is masked by the tone, and humans will not be able to hear any tones inside of the area due to the masking effect.

Unheard frequencies

Another trendy technology used in lossy compression algorithms for audio data is the removal of unheard frequencies. As written earlier, a human being is only capable of hearing between the range from 20 Hz (0,02 kHz) to 20.000 Hz (20 kHz) [12, 13]. By removing all frequencies above and below this threshold, the algorithm can, therefore, use a lower sample rate according to the Nyquist theorem [16]. However, this is only applicable when the original audio contains maximum frequencies above the human threshold of 20 kHz.

Figure 2: Audio masking effect on a loud tone of 500 Hz, the gray area is masked by the tone [15].



3.3 Audio features

Audio on itself could not just be used in a machine learning algorithm because the data is not numerical or not in any form to be able to be processed and used in an algorithm. Therefore there is a necessity to extract numerical values based on the audio data using different statistical and mathematical methods and equations.

It is possible to retrieve numerical values from two perspectives of the audio content. The first perspective is the time-domain and is how the audio is represented when captured and shows the signal waves. On the other hand, there is the time-frequency representation that contains information about the frequencies used.

Table 2: Audio features grouped by their category [27]

Group	**Features**
Temporal features	RMS Energy, Entropy
Spectral features	Zero-crossing rate, Centroid, Polynomial, Roll-off, Bandwidth, Contrast, Flatness measure, MFCC

3.3.1 Time-domain features

Time-domain features are directly extracted from the audio samples of the digital audio signal.

RMS energy

The Root Mean Square Energy is easily retrieved by processing and applying statistical methods on the waveform of the audio given [4]. The output energy is a simplistic but yet effective numerical feature thinking about the distinguishment between speech and non-speech audio due to the energy used in wider frequencies giving different values for the two classes.

$$x_{rms} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2 \tag{2}$$

3.3.2 Time-frequency features

More advanced features can be retrieved from the spectral perspective of the audio signal, also called the time-frequency representation. The time-domain signal is transformed into a spectrogram with the Short-Time Fourier Transform described earlier in the subsection 3.1.2.

Zero-crossing rate

The Zero-Crossing Rate provides important spectral information and is performance-wise very low-cost when computing for a given audio stream. The aforementioned feature is strongly related to the average frequency since the Zero-Crossing Rate will provide a numeric value, which is nearly directly related to the time between waves [3].

$$ZCR_r = \frac{1}{2}\sum_{n=1}^{N} |sign(x_r(n)) - sign(x_{r-1}(n))| \tag{3}$$

Where

$$sign(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases} \tag{4}$$

Spectral centroid

The Spectral Centroid provides the center of gravity of the spectrum over the audio fragment given [3].

$$C_r = \frac{\sum_{k=1}^{\frac{N}{2}} f[k]|X_r[k]|}{\sum_{k=1}^{\frac{N}{2}} |X_r[k]|} \tag{5}$$

Polynomial features

The polynomial feature is based on a principle called polynomial curve fitting, and in specific, the curve fitting with the least-squares method. The outputs of the polynomial features are the coefficients found of fitting an nth-order polynomial to the columns of a spectrogram [28].

Spectral roll-off

The Spectral Roll-Off feature computes the maximum frequency situated within the percentage of power given out of the total power over the given spectral data. This feature uses RMS Energy with filters to calculate the percentage of energy out of the total energy in the spectral representation [29].

$$R_F = f[K] = k \cdot df : \sum_{k=1}^{K} |X(k)| \leq \sum_{k=1}^{N/1} |X(k)| \tag{6}$$

Spectral bandwidth

The spectral bandwidth feature is the calculated bandwidth on the spectrogram frame given. The calculation is done with the following equation [30]:

$$BW = \sqrt{\frac{\sum_{k=1}^{N/2}(f - f_c)^2 \cdot |X(k)|^2}{\sum_{k=1}^{N/2}|X(k)|^2}} = \sqrt{\frac{\sum_{k=1}^{N/2}(k \cdot df - SP_c)^2 \cdot |X(k)|^2}{\sum_{k=1}^{N/2}|X(k)|^2}} \tag{7}$$

Spectral contrast

This feature computes the contrast of the spectral data provided. The numerical value retrieved from this feature represents how narrow-band or wide-band the signals are. A higher contrast value may be a result of a broader and more clear signal, while a low contrast might be coming from a signal with more noise and more extensive usage of frequencies [31].

Spectral flatness measurement

The spectral flatness measures the constancy, width, and noisiness of a given spectrogram. The lower the flatness measurement is, the less uniform the power spectrum is in frequency structure. When the measurement is of a higher value, it states that the power over the frequency bands is less uniform, and this could, for example, be occurring when the audio is speech [32]. In computing the spectral flatness, the measurement is defined as the geometrical mean of the values divided by the arithmetic mean.

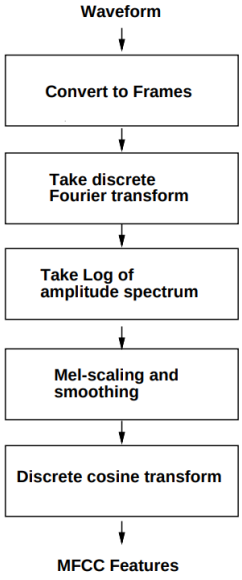
$$SFM = \frac{\sqrt[N]{\prod_{n=0}^{N-1} x(n)}}{\left(\frac{\sum_{n=0}^{N-1} x(n)}{N}\right)} \tag{8}$$

Mel-frequency cepstral coefficients

MFCC is a feature consisting of multiple numerical outputs depending on the input parameter of the number of envelopes to return. The feature is a compact representation of the short-time spectrum envelope and has long been applied in speech recognition but, more recently, also in recognizing music and their genres [33, 34, 35]. The process of computing the MFCC feature begins with transforming the windowed audio data frame by DFT (Discrete Fourier Transform), followed by taking the logarithm of the amplitude spectrum. The logarithm transformation serves to deconvolve multiplicative components of the spectrum, such as the source and filter transfer function [3]. At the end of the process of computing the feature, Mel-scaling and smoothing are applied and followed by the final process, which consists of applying a reverse process of the Fourier Transform, the inverse Discrete Fourier transform [33]. The complete process of computing the MFCC feature is also visualized in figure 3.

The input parameter is the number of filters used and will represent the number of output values per window. According to research about the effectiveness and the different implementations of MFCC, it concludes that for every problem and data, a different number of filters could result in a completely different performance in prediction problems [36, 37].

Figure 3: The pipeline of MFCC [33]



3.4 Machine learning algorithms

3.4.1 XGBoost classifier

The XGBoost Classifier is a gradient boosting classifier build by the open-source community for various platforms such as C++, Java, Python, R, and Julia and works on different operating systems. The classifier is also integrated within several different distributed processing frameworks such as Apache Hadoop and Apache Spark. The classifier uses the technology of tree boosting, which is used in large-scale applications and produces state-of-the-art results [38].

3.4.2 Multilayer perceptron

Multilayer Perceptron (MLP) is a neural network classifier that consists of at least three layers of nodes, where the first layer is the input layer, the middle layer(s) consist of the hidden layer(s), and the final layer is the output layer. The neural network utilizes a supervised learning method called backpropagation for training [39].

3.4.3 Random forest

The random forest classifier consists of multiple layers of decision trees that are constructed during the training process of the classifier by the algorithm [40]. One of the best features of this classifier, and also the XGBoost classifier, is that the risk of overfitting, which is very high in standard decision trees, is not or nearly corrected by the fact that it uses multiple trees [41].

4 Data & Methodology

The executed experiments did require a specific methodology to retrieve, record, and process the radio signal into audio fragments, and eventually into numerical data. This section describes in detail how the data is retrieved, stored, and processed.

4.1 Data

Having enough data is often a challenge, especially when working with data that requires manual work by a human person to process and create ground truth. In this case of the described problem and the recordings of the audio, the data is not yet numerical data when captured. Hence, we have to process the audio signals from the radio production and store it on disk to be able to process and execute computations. The progress of extracting and computing numerical values is achieved with several feature extraction methods described in the literature review.
In total, the data consist of 241 hours of radio recordings, which result in 15,1 GB in total size of the compressed audio.

4.2 Methodology

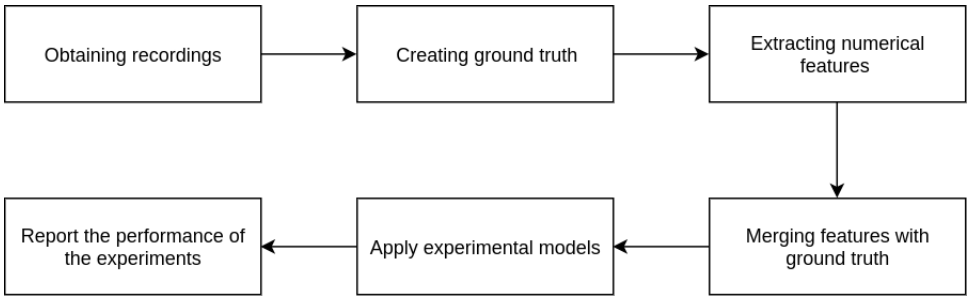


Figure 4: Overview of the experiment methodology

4.2.1 Recording radio productions

In the experimental setup, FFmpeg is used to record internet radio. Unfortunately, the live-stream consists of compressed audio with a compression encoding, which results in some loss of sound.

Audio of two radio stations was captured for both a minimum of four days of stream non-stop. The two radio stations consist of:

- **WNYC**: Local news radio station situated in New York, United States;
- **Heart London**: Local radio station with contemporary music situated in London, United Kingdom.

Both radio stations have different specifications of programs and content. The choice of two stations that are entirely contrary is made to experiment with the effectiveness of the experiment and the used technologies on totally different kind of radio stations available. While both radio stations have a high-quality stream we still have to cope with the fact that we lose specific frequencies due to the compression of the codec used, for example as a result of the masking effect in the MP3 codec used in the streaming as described in the paragraph situated in the section 3.2.2 of the literature review.

Storage Encoding

Since the audio streams are already compressed, there is no need to store the stream with a lossless codec, so for the experiments, MP3 is used. However, before using it in any processing, it needs to be converted back to PCM (WAV), this can be done in memory or as a preparation step on the disk. Unfortunately, we already lost specific frequencies due to the codec used in streaming, and converting back to WAV is only to correctly load the file into the used tools and scripts used in the experiment.

Data Stored

In total, the stored and retrieved data consist of:

- WNYC: 97 hours, 97 files and 4,2 GB of compressed audio;
- Heart London: 144 hours, 144 files and 10,9 GB of compressed audio.

4.2.2 Labeling recordings/ground truth

To use any metrics or to learn any supervised machine learning algorithm, there is a demand in having ground-truth data. When handling audio problems, most of the time, this ground-truth does not exist and needs to be manually created by a human being, which is very time consuming for this specific problem.

For both radio stations, there have been six hours of radio production annotated, resulting in a total of 12 hours of annotated ground-truth data.

Method of labeling

In this problem, especially when handling raw audio data, it is more favorable to precisely annotate every part of speech and non-speech (and potentially other labels) directly from an editor. The ground truth is created by using the software called Audacity by adding a label-track to the file loaded. Figure 5 shows a screenshot of the Audacity editor with labeled audio, and table 3 contains the audio categories and labels used in annotating the audio tracks.

Figure 5: Labeling audio in Audacity

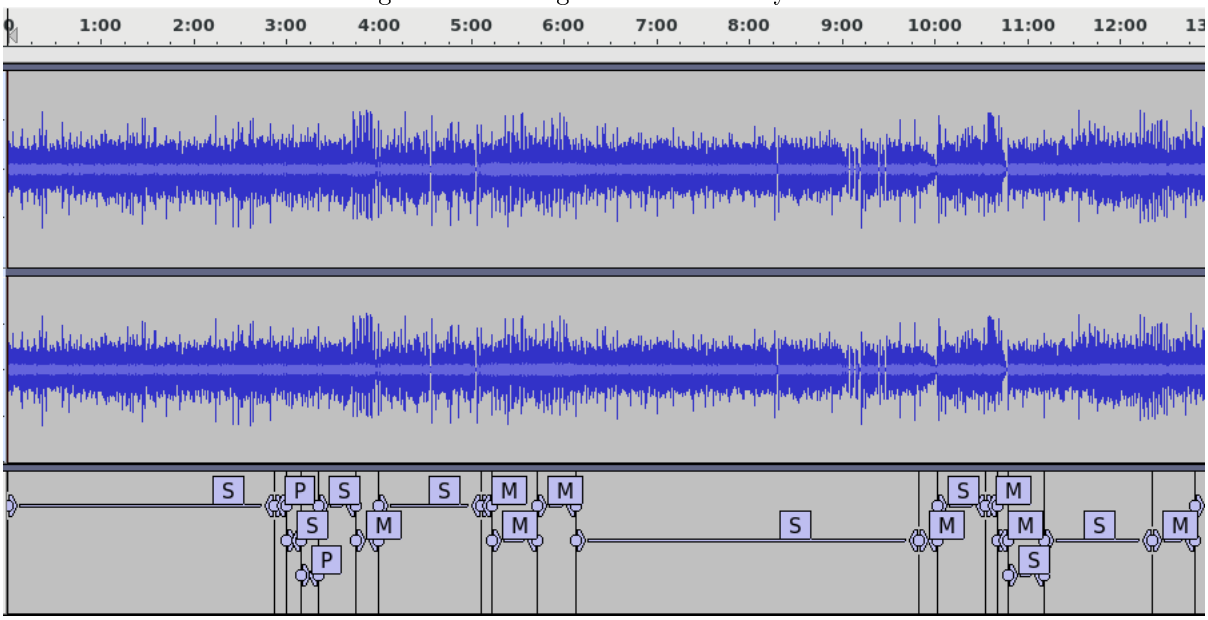


Table 3: Audio categories and labels used in annotating

Category	Label	Description
S - Speech	S	(Normal) Speech
	P	Telephone Speech
	AS	Advertisement with mainly speech
N - Non-Speech	M	Music
	A	Advertisement
	J	Jingle

Exporting labels

From the software used, the label data is exported to a text file containing three columns: the start time, the end time, and the label text. All the times are in seconds in decimal format to guarantee a high precision.

The file can later be read and used by any data manipulation tools to create and combine the ground truth with the obtained features of the audio depending on the individual configurations. An example output can be found in the listing 1.

```
start           end             label
0.000000        172.880447      S
172.880447      180.984218      P
180.984218      190.492643      S
190.492643      201.081570      P
201.081570      225.933135      S
225.933135      240.087721      M
240.087721      306.862794      S
306.862794      313.778012      M
313.778012      343.275738      M
343.275738      368.235353      M
368.235353      590.170627      S
590.170627      602.812510      M
```

Listing 1: Exported Labels File

4.2.3 Extracting features

One of the most critical steps in the experiment implies finding the best split and window time and extracting the correct features from the radio recordings. The feature extraction is executed in Python with the audio library *librosa*. The library is focused on users migrating from MATLAB or with MATLAB experience and was introduced by Brian McFee [6]. Furthermore, the library contains all significant features for signal processing and extracting several numerical values from audio streams.

Library details

In general, the library covers a broad spectrum of signal processing. However, the library is mainly focused on audio problems and processing. Furthermore, the library contains methods to load and resample audio from a file on the disk. While the resampling is optional, it will be turned on by default. The library mainly uses the libraries numpy and scipy for efficiently calculating, computing, and processing the data according to the functions.

Loading an audio file from disk will result in two outputs, the audio signal represented in a one-dimensional numpy array mainly denoted as *y*. Secondly, the sample rate in hertz mainly denoted as *sr*. Together you could calculate the total duration of the audio signal with [6]:

```
duration = float(len(y)) / sr
```

The library furthermore contains methods to convert the signal to a spectrogram with the Short-Time Fourier Transform with the method (`stft`) and also contains the inverse method for the STFT with `istft`.

Besides the methods for extracting features from the audio signal with several options and parameters, it also contains methods to represent spectral and temporal plots with the use of matplotlib. Later on, some examples will show the results based on the different categories of annotated data.

The library is currently maintained mainly by McFee himself and the community on GitHub, where the project's source code is hosted under the ISC (Internet Software Consortium) open source license.

Splitting audio

When extracting features, it is possible to adjust the walking frame and hop for each feature when supported. Every problem and every data has its own best hop and window size. In the experiment, there will be a few combinations tried, and the best outcome will be used to optimize further. All possible combinations can be found in the table 4

Table 4: Combinations of hop and window sizes to use in the experiment

Hop size	**Window size**
250 ms	None
	500 ms
500 ms	None
	750 ms
	1000 ms
750 ms	None
	1000 ms
1000 ms	None
	1250 ms

Used features

In the experiment, the numerical represented data will consist of a total of 43 numerical columns extracted from the following features. The used features are also summed in table 5

Roll-off percentages A total of 11 roll-off features with the following percentages: 10%, 25%, 50%, 60%, 70%, 75%, 80%, 85%, 90%, 93% and 96% to represent a wide variety of the frequency spread of the signal.

MFCCs The computed MFCCs with an input parameter of 24 filters resulting in 24 extracted numerical values.

Zero-crossing The computed zero-crossing rate, a single numerical value.

RMS Energy The single numerical value based on the time-domain feature extracting the RMS energy.

Spectral centroid, bandwidth and contrast The numerical values of the three spectral features: centroid, bandwidth and contrast.

Spectral flatness measurement The single numerical value of the computed flatness measurement on the spectral representation of the audio segment.

Polynomial orders The extracted numerical values of the polynomial fitting in the two orders resulting in two numerical values.

Table 5: Extracted audio features in the experiment

Category	**Feature**
Temporal features	1 × RMS Energy
Spectral features	11 × Roll-off percentage
	24 × MFCCs
	1 × Zero-crossing
	1 × Spectral centroid
	1 × Spectral bandwidth
	1 × Spectral contrast
	1 × Spectral flatness measurement
	2 × Polynomial orders

Process of extracting

The process of extracting is quite an extended process because, for a single audio file of one hour, the matrix of the possible window and hop sizes will require to perform the extraction process nine times, according to the table 4. Below are some code snippets of the process with an explanation about the function of the code. The exact and full source-code for the extraction is separately available.

Loading audio file In the code listing 2, the file will be loaded and resampled to mono audio, of which the data will be stored in the two variables y and sr representing the audio signals. After this, the hop and window sizes will be converted from milliseconds to the number of samples according to the formula $samples = seconds \cdot sr$. This is done with a helper method provided with the librosa library. The variable `split_times` is very important for the exact positional reference of the splits in seconds. The times are generated by using numpy and an inline for-loop. This variable will be used later when combining the features into a data frame.

```
y, sr = librosa.core.load(file, mono=True, sr=44100)

hop_split_seconds = 500
hop_length = librosa.core.time_to_samples(hop_split_seconds, sr=sr)
win_split_seconds = None
win_length = None
if win_split_seconds is not None:
win_length = librosa.core.time_to_samples(win_split_seconds, sr=sr)

total_splits = len(y) / hop_length

split_times = np.array([hop_split_seconds * i for i in range(0, int(total_splits)
    +1)])
```

Listing 2: Loading audio file in librosa and prepare variables

Extracting features The code listing 3 the process of extracting is done by using the librosa extraction methods. Furthermore, the extracted features are of different shapes. Some features contain a multi-dimensional value that is returned based on their implementation and parameters, for example, the MFCC features, which will return several dimensions based on the number of filters used.

```
mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=24, hop_length=hop_length)
zero_crossing = librosa.feature.zero_crossing_rate(y=y, hop_length=hop_length,
    frame_length=win_length or 2048)
rms_energy = librosa.feature.rms(y=y, hop_length=hop_length, frame_length=
    win_length or 2048)
centroid = librosa.feature.spectral_centroid(y=y, sr=sr, hop_length=hop_length)
bandwidth = librosa.feature.spectral_bandwidth(y=y, hop_length=hop_length)
contrast = librosa.feature.spectral_contrast(y=y, hop_length=hop_length)
flatness = librosa.feature.spectral_flatness(y=y, hop_length=hop_length)
rolloff = dict(
percentage_10=librosa.feature.spectral_rolloff(y=y, sr=sr, hop_length=hop_length,
    roll_percent=0.10),
percentage_25=librosa.feature.spectral_rolloff(y=y, sr=sr, hop_length=hop_length,
    roll_percent=0.25),
percentage_50=librosa.feature.spectral_rolloff(y=y, sr=sr, hop_length=hop_length,
    roll_percent=0.50),
percentage_60=librosa.feature.spectral_rolloff(y=y, sr=sr, hop_length=hop_length,
    roll_percent=0.60),
percentage_70=librosa.feature.spectral_rolloff(y=y, sr=sr, hop_length=hop_length,
    roll_percent=0.70),
percentage_75=librosa.feature.spectral_rolloff(y=y, sr=sr, hop_length=hop_length,
    roll_percent=0.75),
percentage_80=librosa.feature.spectral_rolloff(y=y, sr=sr, hop_length=hop_length,
    roll_percent=0.80),
percentage_85=librosa.feature.spectral_rolloff(y=y, sr=sr, hop_length=hop_length,
    roll_percent=0.85),
percentage_90=librosa.feature.spectral_rolloff(y=y, sr=sr, hop_length=hop_length,
    roll_percent=0.90),
percentage_93=librosa.feature.spectral_rolloff(y=y, sr=sr, hop_length=hop_length,
    roll_percent=0.93),
percentage_96=librosa.feature.spectral_rolloff(y=y, sr=sr, hop_length=hop_length,
    roll_percent=0.96),
)
poly_features = librosa.feature.poly_features(y=y, sr=sr, hop_length=hop_length)
```

Listing 3: Extracting features in librosa

Combining and storing numerical features The final part is done by the code in listing 4 where the extracted numerical values will be combined together in a numpy array with the numpy method `column_stack` and annotated based on their split timestamps with the earlier generated `split_times` variable. After the process of combining, the file will be saved to the file given by the variable `out_file` in CSV format. The code for the header is only executed on the first line, denoting the headers in the CSV file format.

```
if len(header) == 0:
header.append('split_times')
for perc in rolloff.keys():
header.append('rolloff_{}'.format(perc))
for idx, _ in enumerate(mfcc):
header.append('mfcc_{}'.format(idx))
header.append('zero_crossing')
header.append('rms_energy')
header.append('centroid')
header.append('bandwidth')
header.append('contrast')
header.append('flatness')
for idx, _ in enumerate(poly_features):
header.append('poly_features_{}'.format(idx))
print('HEADER: {}'.format(', '.join(header)))

f = np.column_stack((
split_times,
*[rv[0] for rv in rolloff.values()],
*[m for m in mfcc],
zero_crossing[0],
rms_energy[0],
centroid[0],
bandwidth[0],
contrast[0],
flatness[0],
*[m for m in poly_features],
))

# Write output
with open(out_file, 'wb') as out_handle:
out_handle.write('{}\n'.format(','.join(header)).encode())
np.savetxt(out_file, f, delimiter=',')
```

Listing 4: Manipulating and storing features in librosa with numpy

4.2.4 Combining ground truth

After the process of extracting numerical features based on the process described in section 4.2.3 the next step is to enrich the feature file with the ground truth, but only for the cases where we have the ground truth.

Reading input files

In order to combine the features file with the annotations, we have to define methods for reading both files. The code listing 5 contains two functions for reading both types of files.

```
structure = {
'S': [
'S',  # General Speech
'P',  # Phone Speech
'AS', # Advertisement, mainly speech
],
'N': [
'M',  # Music
'J',  # Jingle
'A',  # Advertisement
]
}

def read_feature_file(file, audio_file):
df = pd.read_csv(file, header=0)

# Detect feature configuration.
hop, win = re.findall(r'\.wav_hop_([0-9]+)__win_([0-9]+|none)', os.path.basename(
    file))[0]

# Drop the first row as it always contains invalid data.
df = df[df.split_times != 0.00]

# Insert the filename and configuration to the dataframe.
df['file'] = os.path.basename(audio_file)
df['station'] = os.path.basename(os.path.dirname(audio_file))
df['hop_size'] = hop
df['win_size'] = win
return df

def read_label_file(file):
df = pd.read_csv(file, sep='\t', names=['label_start', 'label_end', 'label'])

# Replace add new columns, cat and subcat, denoting the head category and
    subcategory of the label.
df['category'] = df['label'].replace(structure['S'], 'S').replace(structure['N'], '
    N')
df['subcategory'] = df['label']
return df
```

Listing 5: Reading input files with Pandas

Combining features with labels

After the loading has been finished, we have to enrich the feature data frame with the labels. Both datasets are not ready to be merged directly because of the format of the label files. In order to combine, we have to merge based on statements. Fortunately, pandas comes with a very advanced merging functionality, which is used in the code listing 6. This code is obtained from the original for-loop, which will execute this for all files. The input parameters are `feature_file` for the features CSV-file and `file` for the labels file. The output file will contain extra columns based on the merging properties.

```
# Read the labels file.
label_df = read_label_file(labels_file)
label_df = label_df.assign(key=1)

out_file = '{}__annotated.csv'.format(feature_file[:-4])

# Read the feature file.
feature_df = read_feature_file(feature_file, file)
feature_df = feature_df.assign(key=1)

# Merge process, first make a intermediate merge.
merge_df = pd.merge(feature_df, label_df, on='key').drop('key', axis=1)
merge_df = merge_df.query('split_times >= label_start and split_times < label_end')

# Convert to the output dataframe, prepare for the output format.
output_df = feature_df.merge(
merge_df[[
'split_times', 'label_start', 'label_end', 'label', 'category', 'subcategory'
]], on='split_times', how='left'
)
output_df = output_df[output_df.label != None]
output_df = output_df.drop('key', axis=1)

# Output to csv.
output_df.to_csv(out_file, index=False)
```

Listing 6: Merging feature dataframe and labels with Pandas

In the end, the output file will have the following columns extra besides the feature columns: `file`, `station`, `hop_size`, `win_size`, `label_start`, `label_end`, `label`, `category`, `subcategory`. The file columns represent the original WAV file, the station is the radio station code which could be either 'WNYC' or 'HeartLondon', the hop and window size contains the required hop size and optional window size in milliseconds, the label start and end columns contains the start and the end timestamp of the labeled entry, the label and subcategory are the original label while the category column consists of the speech or non-speech binary specification and could either be 'S' (Speech) or 'N' (Non-Speech).

4.2.5 Machine learning models

Algorithms

For both two experimental setups, there has been defaulted parameter tests at first with the following three machine learning algorithms:

- Multilayer Perceptron;
- Random Forest;
- XGBoost.

Splits and cross-validation

Initially, with the first observation tests, the split consisted out of four hours of train data and two hours of testing, which is a split percentage of 66,6% train and 33,3% test data. The splitting is done with loading only the four out of six available feature files for the train data and the other two for testing purposes. Everything is done per radio station, so the measurements and performance output is different per radio station.
Furthermore, to optimize and obtain a fair observation, cross-validation is applied over the complete six hours of extracted features, again per radio station. In the progress of cross-validation, the number of K-folds has been set to 10-folds.

Parameter tuning

Based on the results of the cross-validation performance of the multi-layer models, two algorithms will be tested with optimized parameters, tuned with grid searching, and the cross-validation methods with five folds.

Single-model experiment

The first of the two main experiments with machine learning has been executed with only a single layer model based on all the annotated labels of the audio file seen as one single problem. The model is also graphically explained in figure 6. In this setup, the following Machine Learning algorithms are being tested: Multilayer Perceptron (MLP), XGBoost, and Random Forest.

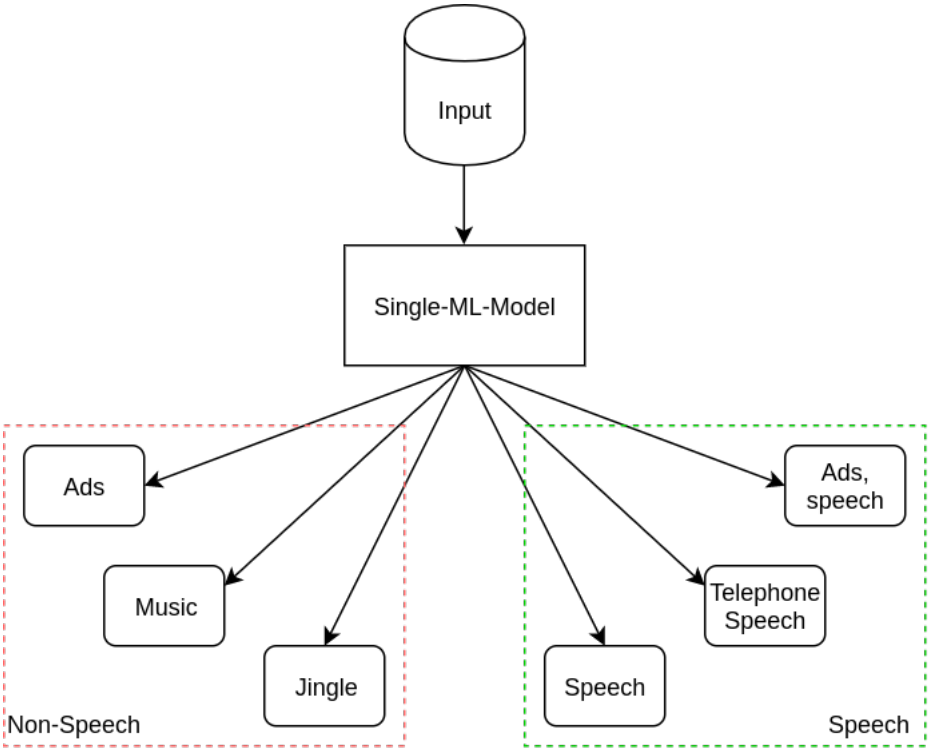


Figure 6: The single-model experiment setup

Multi-layer model experiment

Another experiment will be executed by having two layers of models with a total of three models where the first layer consists of one model that decides if the input given is either speech or non-speech and two underlying models that decide the exact label under the categories of speech and non-speech. The model is illustrated in figure 7. Because the first model, the categorical model, is using binary classification, it could potentially lead to an increase of performance on the performance of distinguishing speech and non-speech. In this setup, the following Machine Learning algorithms are being tested in both the main and sub models: Multilayer Perceptron (MLP), XGBoost, and Random Forest.

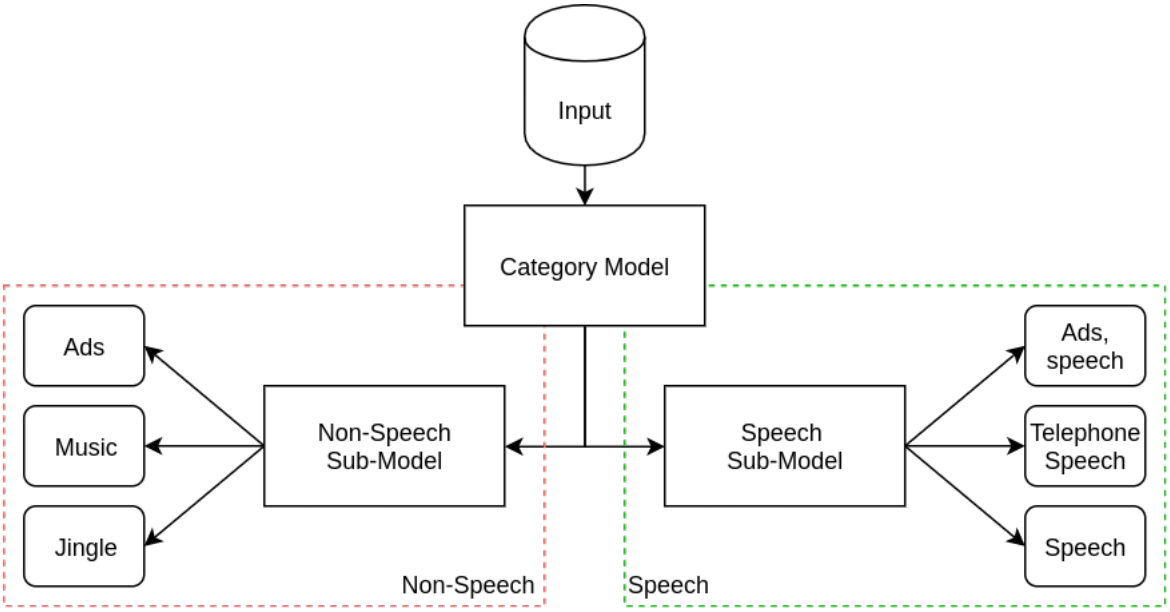


Figure 7: The multi-layer model experiment setup

5 Results

This section contains the results of the executed experiments described earlier.

5.1 Annotation Tool

Earlier in the process of the experiment, a tool has been created to annotate and display predictions on an audio player interactively. This project has been developed during the early phase of the dissertation and is still experimental. In the end, the tool has not completely been used but may be interesting for further development and sharing with other researchers in the field. The two figures 8 & 9 display screenshots of the tool in usage. The technologies behind the developed tool are Python with Tornado as a back-end server and Angular on the front-end.
The tool will be expanded and may be useful to expand to an online service that could be used to provide the service to a broader public in the web browser and could be expanded to meet the demands for overall content annotation and management for creating and managing ground-truth data.

Figure 8: Audio stamper tool, annotating the audio track.

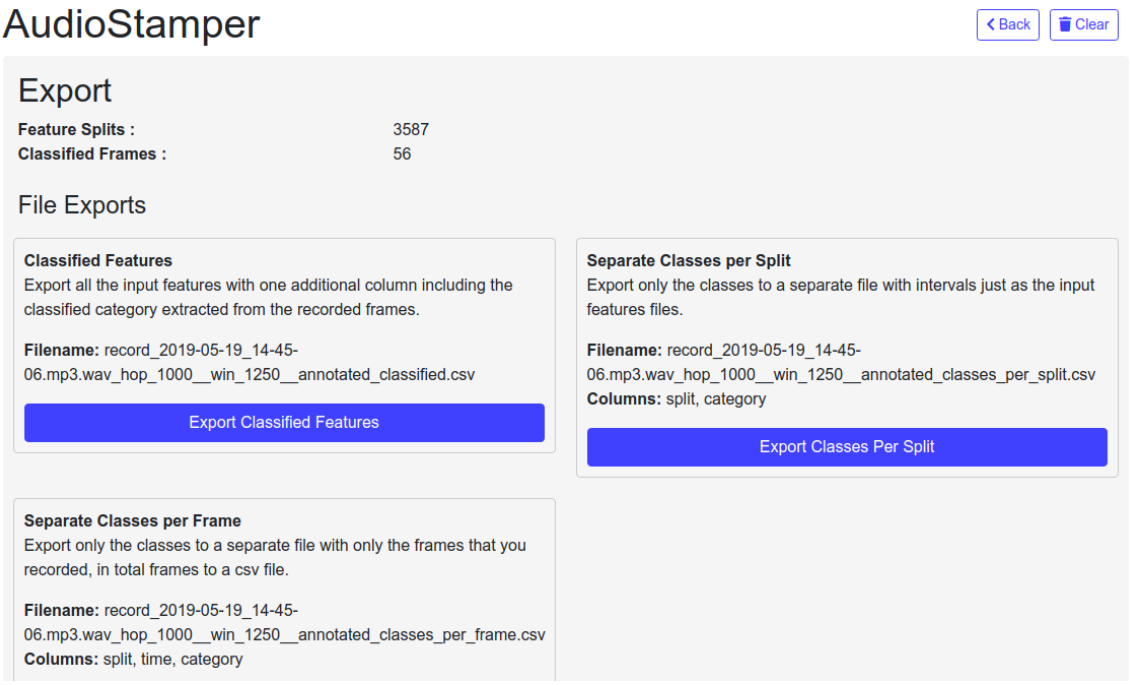
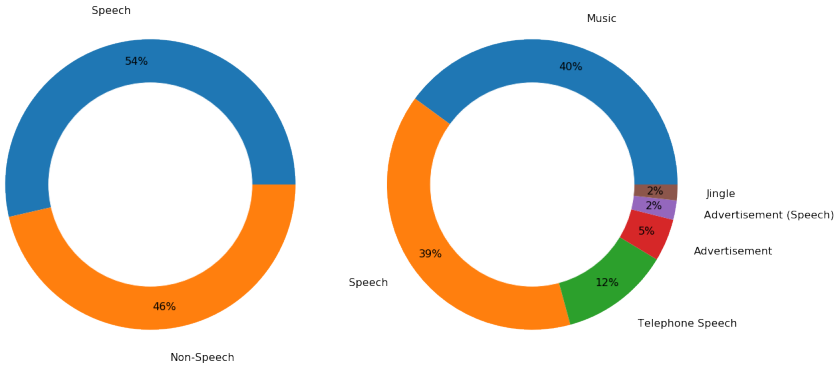


Figure 9: Audio stamper tool, the possible export methods

5.2 Annotated Audio

Based on the human-annotated audio, we gained a few insights into our data. In the pie-charts shown in figure 10, it is clear that both radio stations together make a quite good share in both categories of speech and non-speech.

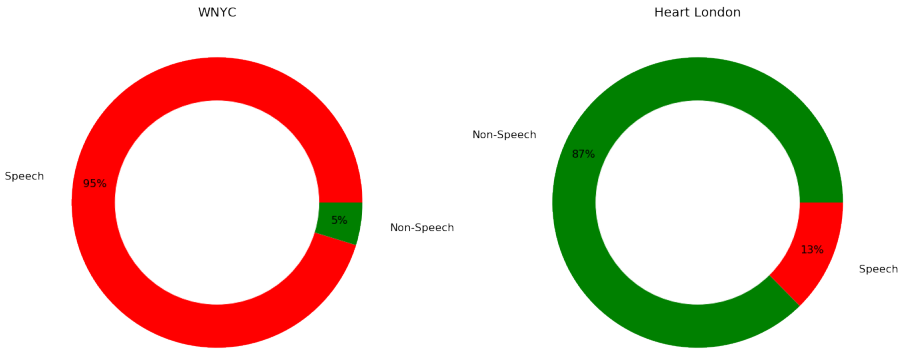


(a) Category balance (b) Label balance

Figure 10: Label and category balance over the complete annotated data set

Unbalanced data

However, if we look closely and split the two radio stations and redraw the plot, it shows us that for both radio stations, the data is quite unbalanced. This is not a strange phenomenon according to the chosen radio stations where one is a news station, and the other station is playing significantly more music. This difference is shown in figure 11. The fact that there is unbalanced data could lead to incorrect measurements of the performance. However, the final problem could still be solved by applying a model to the audio with the pre-trained data.



(a) WNYC (News) (b) Heart London (Contemporary)

Figure 11: Category balance per radio station

5.3 Feature Extraction

The feature extraction is done in the programming language Python with the audio library *librosa*. The library is focused on people migrating from or having a background in MATLAB and is introduced by Brian McFee [6]. Furthermore, the library contains all significant features for signal processing and extracting several numerical values from audio streams.

5.4 Machine Learning Algorithms Results

5.4.1 Single-model experiment

In the Single-Model experiment, we ran all the predefined combinations of possible configurations among the three algorithms and with two validation methods. The results are summarized in table 6, where the best results and configuration combinations are shown based on the accuracy score.

			Optimal Configuration		Results	
Algorithm	**Validation Method**	**Station**	**Hop Size**	**Window Size**	**Accuracy**	**F1**
MLP	Split, 33%	WNYC	500	750	0.945954657784463	0.944773985976810
MLP	Split, 33%	HeartLondon	250	500	0.817028698271762	0.805407091867788
MLP	CV, 10 folds	WNYC	250	-	0.945745595186936	0.943646119825491
MLP	CV, 10 folds	HeartLondon	250	-	0.815865969029121	0.796161956534341
RF	Split, 33%	WNYC	250	500	0.936882251826386	0.929231514995747
RF	Split, 33%	HeartLondon	250	500	0.799323502986100	0.751769117219394
RF	CV, 10 folds	WNYC	250	500	0.937043403523850	0.929279848894466
RF	CV, 10 folds	HeartLondon	250	500	0.800274826911897	0.752953352455008
XGBoost	Split, 33%	WNYC	500	750	0.933598366820673	0.926918875870289
XGBoost	Split, 33%	HeartLondon	500	1000	0.794102103371737	0.749654185602018
XGBoost	CV, 10 folds	WNYC	500	750	0.945954657784463	0.942316869777580
XGBoost	CV, 10 folds	HeartLondon	250	500	0.813487659214629	0.783358914663545

Table 6: Results of the best configuration for the Single-Layer Models

5.4.2 Multi-layer model experiment

The performance in the Multi-Layer Model experiment is more difficult to interpret. Because of the issue of the multiple performance indicators due to the multiple algorithms used, the complete model accuracy is utilized as a performance metric. This metric is computed with the formula $perf = \frac{(main \cdot speech + main \cdot nonspeech)}{2}$, where the parameters are the accuracy metrics of the individual and main model. The results are presented in the table 7.

			Optimal Configuration		
Algorithm	**Validation Method**	**Station**	**Hop Size**	**Window Size**	**Accuracy**
MLP	Split, 33%	WNYC	250	-	0.957464249382319
MLP	Split, 33%	HeartLondon	250	500	0.770827877563129
MLP	CV, 10 folds	WNYC	500	1000	0.961432071043146
MLP	CV, 10 folds	HeartLondon	500	1000	0.774364573502094
RF	Split, 33%	WNYC	750	1000	0.951539269479086
RF	Split, 33%	HeartLondon	500	1000	0.758797187730085
RF	CV, 10 folds	WNYC	750	1000	0.951140062305160
RF	CV, 10 folds	HeartLondon	500	750	0.757104084658736
XGBoost	Split, 33%	WNYC	750	1000	0.953073495882124
XGBoost	Split, 33%	HeartLondon	500	1000	0.761184711076996
XGBoost	CV, 10 folds	WNYC	250	500	0.957134219515392
XGBoost	CV, 10 folds	HeartLondon	500	1000	0.776487036919862

Table 7: Results of the best configuration for the Multi-Layer Models

Parameter Tuning

Based on the outcomes in the Multi-Layer Model experiment, the parameter tuning experiment is being executed on two algorithms with their configurations listed bellow in table 8.

Algorithm	**Station**	**Hop Size**	**Window Size**	**Parameters to tune**
MLP	WNYC	500	1000	*hidden_layer_sizes, activation, solver, alpha, learning_rate*
MLP	HeartLondon	500	1000	
XGBoost	WNYC	250	500	*max_depth, min_child_weight, subsample, colsample_bytree, gamma*
XGBoost	HeartLondon	500	1000	

Table 8: Parameter-tuning overview of parameters and algorithms

Table 9 below shows the used values per algorithm. Per layer, the best values are being determined with the usage of the full grid-search method with a total of five K-folds.

Algorithm	**Parameter**	**Values tested**
MLP	hidden_layer_sizes	*(200, 100), (150, 75), (100, 50)*
MLP	activation	*relu, tanh*
MLP	solver	*adam, sgd*
MLP	alpha	*0, 0.0001, 0.001, 0.01, 0.05, 0.1*
MLP	learning_rate	*adaptive, constant*
Algorithm	**Parameter**	**Values tested**
XGBoost	max_depth	*3, 4, 5, 6, 7, 8*
XGBoost	min_child_weight	*1, 5, 10*
XGBoost	gamma	*0.5, 1, 1.5, 2, 5*
XGBoost	subsample	*0.6, 0.8, 1.0*
XGBoost	colsample_bytree	*0.6, 0.8, 1.0*

Table 9: Parameter-tuning overview with parameter values tuned

Best parameter values The outcome of all the grid-search experiments based on the parameters and values described earlier can be observed in table 10. In the table, some grid-search entries have the same values in their tuned parameters, and this is shown as one cell.

Algo	Station	Model	Accuracy	Parameter Values
MLP	WNYC **0.959823229**	Main	~ 0.9739981	hidden_layer_sizes: (200, 100); activation: relu; solver: adam; alpha: 0.001; learning_rate: adaptive
		Speech	~ 0.9708935	
		Non-Speech	1.0	
	HeartLondon **0.778477557**	Main	~ 0.9021245	
		Speech	~ 0.8364094	
		Non-Speech	~ 0.8894666	
XGBoost	WNYC **0.958035027**	Main	~ 0.9712075	max_depth: 8; min_child_weight: 5; subsample: 0.6; colsample_bytree: 1.0; gamma: 1.0
		Speech	~ 0.9728739	max_depth: 8; min_child_weight: 1; subsample: 0.6; colsample_bytree: 1.0; gamma: 0.5
		Non-Speech	1.0	
	HeartLondon **0.778795077**	Main	~ 0.9036042	max_depth: 8; min_child_weight: 5; subsample: 0.6; colsample_bytree: 1.0; gamma: 1.0
		Speech	~ 0.8305369	
		Non-Speech	~ 0.8932156	

Table 10: Parameter-tuning results with best parameter values. The value below the station is the accuracy of the total model.

According to the results shown in the table, the accuracy of both algorithms is very close when comparing it per radio station. Actually, for one of the radio stations, MLP is preferred above the XGBoost algorithm and vice versa. However, the difference is so low that there is not a single algorithm that could be seen as best in this case.

6 Conclusion

6.1 Summary

Based on the literature review, the earlier research and the findings, it is a big challenge to distinguish speech from non-speech when handling generically. For each radio station, each numerical extraction and each algorithm need to be tuned individually to achieve the best performance.

The experiments show precisely this fact, and while the difference in the performance is not always extensive, it is significantly notable when optimizing the frame and hop sizes for a specific radio station. Furthermore, the tuning of the algorithms and the use of multiple layers of classifiers increased the complexity but gave a more stable result in the end, especially the first-layer classification of the speech and non-speech.

According to the executed research, it is shown that the usage of MLP or the XGBoost classifier algorithms are the best in the extracted numerical data. With stable accuracy, MLP shows an accuracy of up to 97,45% on the WNYC speech/non-speech model based on a hop size of 500ms and a window size of 750ms with tuned parameters and two hidden layers.

The usage of the Two-Layer Model is especially performing well on the top layer. This is also because of unbalanced data, which is a difficulty in the described problem of the audio as one always has more music or talking, based on the provided radio station. However, with the Two-Layer model, the accuracy of the category (speech and non-speech) is very high, while the exact label (music, regular talk, telephone talk, et cetera) can be low according to the radio station and the background noise. For example, the radio station Heart London uses a lot of background jingles and music when talking, which leads to inaccurate results and a lower accuracy, especially the non-speech part, which is entirely understandable according to the nature of the input audio.

Furthermore, the difference in accuracy between the parameter tuned algorithms MLP and XGBoost is negligible. Choosing between the two algorithms can instead be made based on their performance in terms of speed to train and predict for further development or implementation in an application.

6.2 Limitations

The current implementation of the Two-Layered Model is quite expensive and resource-intensive. It might not work well for real-time situations, although the first-layer could only be used if the goal is to distinguish speech and non-speech. Furthermore, the results and performance metrics are entirely depending on how much noise exists in the input audio. For example, the station Heart London has an enormous amount of background music and effects while talking to the listeners. This is, unfortunately, a challenging problem to solve.

Furthermore, the current implementation does not work in real-time and could not be directly implemented in an application. However, the currently trained models are saved and could be used for further research and implementation.

Another limitation is the silence periods existing in the input audio. Sometimes in the audio fragments, there are more extended silence periods in which the classifier can not determinate the correct category and label. By using another method when recognizing silence periods, this could potentially be solved.

6.3 Further Research

At the time of writing, there is a broad interest in the relatively new Convolutional Neural Networks (also known as CNN). CNN is a deep neural learning class that has images as input. The ability to recognize patterns based on the complete data and earlier seen data could be beneficial in the problem given and for example, to reduce the amount of misclassified labels and categories in the middle of a song.

Another interesting topic would be the implementation of a different model to recognize the class of the silence periods. Once a silence period is detected, a different algorithm model could be used, which uses lower amounts of hop and window sizes to recognize the background noise in the silence periods. This different model could be very straightforward with just a few features which only apply to the silence periods.

Furthermore, an implementation could be made based on the researched topic to make a comparison inside a specific application in order to provide users the metric to compare between radio stations based on the speech and non-speech ratio. An example could be seen in figure 12.

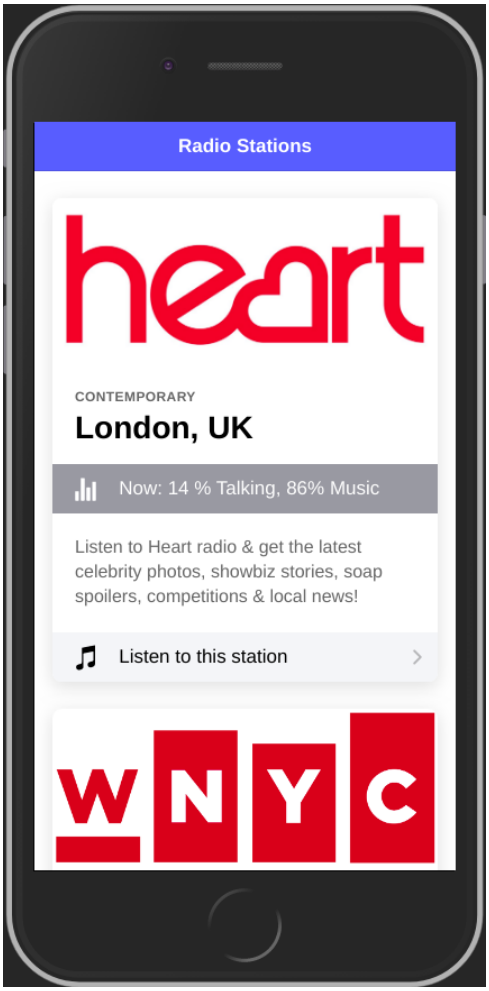


Figure 12: An experimental implementation of the technologies for end-users.

References

[1] S. G. Koolagudi et al. 'Advertisement detection in commercial radio channels'. In: *2015 IEEE 10th International Conference on Industrial and Information Systems (ICIIS)*. 2015, pp. 272–277. DOI: 10.1109/ICIINFS.2015.7399023.

[2] E. D. N. W. Senevirathna and K. L. Jayaratne. 'Automated audio monitoring approach for radio broadcasting channels in Sri Lanka'. In: *2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ICTer)*. 2017, pp. 1–7. DOI: 10.1109/ICTER.2017.8257811.

[3] Preeti Rao. 'Audio Signal Processing'. In: *Speech, Audio, Image and Biomedical Signal Processing using Neural Networks*. Vol. 83. Jan. 2008, pp. 169–189. DOI: 10.1007/978-3-540-75398-8.

[4] Stefan Scherer, Friedhelm Schwenker and Gunther Palm. 'Emotion Recognition from Speech Using MCS and RBF-Ensembles'. In: *Speech, Audio, Image and Biomedical Signal Processing using Neural Networks*. Vol. 83. Jan. 2008, pp. 49–70. DOI: 10.1007/978-3-540-75398-8.

[5] Lindasalwa Muda, Mumtaj Begam and Irraivan Elamvazuthi. 'Voice Recognition Algorithms using Mel Frequency Cepstral Coefficient (MFCC) and Dynamic Time Warping (DTW) Techniques'. In: *J Comput* 2 (Mar. 2010).

[6] Brian McFee et al. 'librosa: Audio and Music Signal Analysis in Python'. In: *Proceedings of the 14th Python in Science Conference*. Jan. 2015, pp. 18–24. DOI: 10.25080/Majora-7b98e3ed-003.

[7] Brian McFee et al. *librosa/librosa: 0.7.1*. Version 0.7.1. Oct. 2019. DOI: 10.5281/zenodo.3478579. URL: https://doi.org/10.5281/zenodo.3478579.

[8] 'A MATLAB toolbox for musical feature extraction from audio'. In: *Proc. of the 10th Int. Conference on Digital Audio Effects*.

[9] Florian Eyben et al. 'Recent Developments in openSMILE, the Munich Open-source Multimedia Feature Extractor'. In: *Proceedings of the 21st ACM International Conference on Multimedia*. MM '13. Barcelona, Spain: ACM, 2013, pp. 835–838. ISBN: 978-1-4503-2404-5. DOI: 10.1145/2502081.2502224. URL: http://doi.acm.org/10.1145/2502081.2502224.

[10] Enrique Alexandre et al. 'Speech/Non-Speech Classification in Hearing Aids by Tailored Neural Networks'. In: *Speech, Audio, Image and Biomedical Signal Processing using Neural Networks*. Vol. 83. Jan. 2008, pp. 145–167. DOI: 10.1007/978-3-540-75398-8.

[11] R. Kotsakis, G. Kalliris and C. Dimoulas. 'Investigation of broadcast-audio semantic analysis scenarios employing radio-programme-adaptive pattern classification'. In: *Speech Communication* 54.6 (2012), pp. 743 –762. ISSN: 0167-6393. DOI: https://doi.org/10.1016/j.specom.2012.01.004. URL: http://www.sciencedirect.com/science/article/pii/S0167639312000118.

[12] H.F. Olson. *Music, Physics and Engineering*. Dover Books. Dover Publications, 1967. ISBN: 9780486217697.

[13] Bhanu Prasad and S. Prasanna. *Speech, Audio, Image and Biomedical Signal Processing using Neural Networks*. Vol. 83. Jan. 2008. DOI: 10.1007/978-3-540-75398-8.

[14] J. Bray and Institution of Electrical Engineers. *Innovation and the Communications Revolution: From the Victorian Pioneers to Broadband Internet*. History and Management of Technology. Institution of Engineering and Technology, 2002. ISBN: 9780852962183.

[15] D. Austerberry. *The Technology of Video and Audio Streaming*. Focal Press, 2005. ISBN: 9780240805801.

[16] U. Grenander. *Probability and Statistics: The Harald Cramér Volume*. Wiley Publications in Statistics. Alqvist & Wiksell, 1959.

[17] H. J. Landau. 'Sampling, data transmission, and the Nyquist rate'. In: *Proceedings of the IEEE* 55.10 (1967), pp. 1701–1706. DOI: 10.1109/PROC.1967.5962.

[18] Julius O. Smith. *Spectral Audio Signal Processing*. online book, 2011 edition. http://-ccrma.stanford.edu/~jos/sasp/, accessed 04/11/2019.

[19] J. B. Allen and L. R. Rabiner. 'A unified approach to short-time Fourier analysis and synthesis'. In: *Proceedings of the IEEE* 65.11 (1977), pp. 1558–1564. DOI: 10.1109/PROC.1977.10770.

[20] *FFT - NTI Audio*. accessed 17/11/2019. URL: https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft.

[21] D. Salomon, D. Bryant and G. Motta. *Handbook of Data Compression*. Springer London, 2010. ISBN: 9781848829039.

[22] J. Coalson and A. Weaver. *Free Lossless Audio Codec*. Internet-Draft draft-ietf-cellar-flac-00.txt. The IETF Trust, 2019. URL: https://tools.ietf.org/pdf/draft-ietf-cellar-flac-00.pdf.

[23] T. Liebchen and Y. A. Reznik. 'MPEG-4 ALS: an emerging standard for lossless audio coding'. In: *Data Compression Conference, 2004. Proceedings. DCC 2004*. 2004, pp. 439–448. DOI: 10.1109/DCC.2004.1281489.

[24] R. Finlayson. *A More Loss-Tolerant RTP Payload Format for MP3 Audio*. RFC 5219. The IETF Trust, 2008, pp. 1–22. URL: https://tools.ietf.org/pdf/rfc5219.pdf.

[25] Alfred M. Mayer. *XXIII. Researches in Acoustics. —No. IX*. Mar. 1894. DOI: 10.1080/14786449408620544. URL: https://doi.org/10.1080/14786449408620544.

[26] Richard H. Ehmer. 'Masking by Tones vs Noise Bands'. In: *Acoustical Society of America Journal* 31.9 (1959), p. 1253. DOI: 10.1121/1.1907853.

[27] Mrinmoy Bhattacharjee, S. Prasanna and Prithwijit Guha. *Time-Frequency Audio Features for Speech-Music Classification*. Nov. 2018.

[28] *Librosa Documentation. librosa.feature.poly_features*. 2019. URL: https://librosa.github.io/librosa/generated/librosa.feature.poly_features.html#librosa.feature.poly_features (visited on 21st Oct. 2019).

[29] Marko Kos, Zdravko Kačič and Damjan Vlaj. 'Acoustic classification and segmentation using modified spectral roll-off and variance-based features'. In: *Digital Signal Processing* 23.2 (2013), pp. 659 –674. ISSN: 1051-2004. DOI: https://doi.org/10.1016/j.dsp.2012.10.008.

[30] Anssi Klapuri and Manuel Davy. *Signal Processing Methods for Music Transcription*. Jan. 2006. Chap. 5. DOI: 10.1007/0-387-32845-9.

[31] Dan-Ning Jiang et al. 'Music type classification by spectral contrast feature'. In: *Proceedings. IEEE International Conference on Multimedia and Expo*. Vol. 1. 2002, 113–116 vol.1. DOI: 10.1109/ICME.2002.1035731.

[32] Yanna Ma and Akinori Nishihara. 'Efficient voice activity detection algorithm using long-term spectral flatness measure'. In: *EURASIP Journal on Audio, Speech, and Music Processing* 2013.1 (2013), p. 87. ISSN: 1687-4722. DOI: 10.1186/1687-4722-2013-21.

[33] Beth Logan. 'Mel Frequency Cepstral Coefficients for Music Modeling'. In: *Proc. 1st Int. Symposium Music Information Retrieval* (Nov. 2000).

[34] M. Müller. *Information Retrieval for Music and Motion*. Springer Berlin Heidelberg, 2007. ISBN: 9783540740483.

[35] Todor Ganchev, Nikos Fakotakis and George Kokkinakis. 'Comparative Evaluation of Various MFCC Implementations on the Speaker Verification Task'. In: *10th International Conference on Speech and Computer (SPECOM 2005)*. Vol. 1. Wire Communications Laboratory, University of Patras, 2011, pp. 191–194.

[36] Vibha Tiwari. 'MFCC and its applications in speaker recognition'. In: *Int. J. Emerg. Technol.* 1 (Jan. 2010).

[37] Wei Han et al. 'An efficient MFCC extraction method in speech recognition'. In: *2006 IEEE International Symposium on Circuits and Systems*. 2006, 4 pp.–. DOI: 10.1109/ISCAS.2006.1692543.

[38] Tianqi Chen and Carlos Guestrin. 'XGBoost: A Scalable Tree Boosting System'. In: *CoRR* abs/1603.02754 (2016). arXiv: 1603.02754. URL: http://arxiv.org/abs/1603.02754.

[39] D.E. Rumelhart and J.L. McClelland. 'Learning Internal Representations by Error Propagation'. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. MITP, 1987. URL: https://ieeexplore.ieee.org/document/6302929.

[40] Tin Kam Ho. 'Random decision forests'. In: *Proceedings of 3rd International Conference on Document Analysis and Recognition*. Vol. 1. 1995, 278–282 vol.1. DOI: 10.1109/ICDAR.1995.598994.

[41] Trevor Hastie, Robert Tibshirani and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. 2nd ed. Springer, 2009. URL: http://www-stat.stanford.edu/~tibs/ElemStatLearn/.