Gonçalo Filipe Torcato Mordido

Master of Science

Automated Organisation and Quality Analysis of User-Generated Audio Content

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Engineering

Adviser: Prof. Dr. Sofia Cavaco, Assistant Professor, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Co-adviser: Prof. Dr. João Magalhães, Assistant Professor, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Examination Committee

Chairperson: Prof. Dr. Fernando Birra
Raporteurs: Prof. Dr. Fernando Perdigão
Member: Prof. Dr. Sofia Cavaco

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

September, 2017

Automated Organisation and Quality Analysis of User-Generated Audio Content

This document was created using the (pdf)LaTeX processor, based in the "unlthesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/unlthesis [2] http://www.di.fct.unl.pt

Acknowledgements

This work was partially funded by the H2020 ICT project COGNITUS with the grant agreement No 687605 and by the project NOVA LINCS Ref. UID/CEC/04516/2013 in the form of a research grant which I am very grateful for. I would like to firstly thank Prof. Dr. Sofia Cavaco for all the tireless help and knowledge given during this last year, together with an exceptional work ethic. I could have not asked for a better supervisor. I would also like to thank Prof. Dr. João Magalhães for pointing me towards the right directions work-wise, and for all the life advice and insights that will certainly be remembered throughout my life.

Finally, I would of course like to thank my family and friends, more specifically to my brilliant sister for always being so accessible and supportive throughout my academic years.

Abstract

The abundance and ubiquity of user-generated content has opened horizons when it comes to the organization and analysis of vast and heterogeneous data, especially with the increase of quality of the recording devices witnessed nowadays. Most of the activity experienced in social networks today contains audio excerpts, either by belonging to a certain video file or an actual audio clip, therefore the analysis of the audio features present in such content is of extreme importance in order to better understand it. Such understanding would lead to a better handling of ubiquity data and would ultimately provide a better experience to the end-user.

The work discussed in this thesis revolves around using audio features to organize and retrieve meaningful insights from user-generated content crawled from social media websites, more particularly data related to concert clips. From its redundancy and abundance (*i.e.*, for the existence of several recordings of a given event), recordings from musical shows represent a very good use case to derive useful and practical conclusions around the scope of this thesis.

Mechanisms that provide a better understanding of such content are presented and already partly implemented, such as audio clustering based on the existence of overlapping audio segments between different audio clips, audio segmentation that synchronizes and relates the different cluster's clips in time, and techniques to infer audio quality of such clips. All the proposed methods use information retrieved from an audio fingerprinting algorithm, used for the synchronization of the different audio files, with methods for filtering possible false positives of the algorithm being also presented.

For the evaluation and validation of the proposed methods, we used one dataset made of several audio recordings regarding different concert clips manually crawled from YouTube.

Keywords: User-generated content, Audio fingerprinting, Audio clustering, Audio segmentation, Audio quality, Supervised learning

Resumo

A abundância e ubiquidade de conteúdo gerado por utilizadores tem aberto novos horizontes na organização e análise de grandes quantidades de dados heterogéneos, especialmente com o aumento da qualidade dos dispositivos de gravação observada hoje em dia. A maior parte da atividade experienciada nas redes sociais hoje contém excertos de áudio, quer provenientes de ficheiros de vídeo, quer de clips áudio. Assim, a análise das características do áudio presente nesse conteúdo é de extrema importância para o seu melhor entendimento. Tal compreensão levaria a uma melhor manipulação de dados ubíquos e providenciaria uma melhor experiência de utilização para o consumidor final.

O trabalho discutido nesta tese consiste em usar características de áudio para organizar e derivar perceções significativas de conteúdo gerado por utilizadores tirado de redes sociais, mais especificamente dados relativos a concertos musicais. Tendo em conta a sua redundância e abundância (*i.e.*, por haver várias gravações relativas a um dado evento), gravações de concertos representam um bom caso de uso para derivar conclusões úteis, num contexto prático, no âmbito desta tese.

Mecanismos que providenciam um melhor conhecimento de tal conteúdo são apresentados, como, por exemplo, o agrupamento baseado em segmentos sobrepostos entre clips diferentes de áudio, segmentação do áudio baseado em como esses clips se relacionam em termos de tempo, e técnicas para inferir a qualidade das amostras. Todos os métodos propostos usam informação retornada por um algoritmo de *audio fingerprinting*, usado para a sincronização dos diferentes ficheiros, com métodos de filtragem de possíveis falsos positivos por parte do algoritmo sendo também apresentados.

Os diferentes métodos propostos pela nossa solução foram testados e validados usando um conjunto de dados construído a partir de gravações de áudio de diferentes músicas de concertos, retirados manualmente do YouTube.

Palavras-chave: Conteúdo gerado pelo utilizador, *Audio fingerprinting*, Agrupamento baseado em áudio, Sincronização de áudio, Qualidade de áudio, Aprendizagem supervisionada

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

In this dissertation we propose mechanisms to achieve a better understanding of user-generated content by the analysis of their correspondent audio signals. The different methods permit the grouping of the different audio files by events, with supplementary information about how the different files synchronise with each other in terms of the event's timeline. Since these organisation methods use solely the information retrieved by an audio fingerprinting algorithm, we also propose two novel methods for filtering possible false positives from the algorithm. Furthermore, we propose one novel method to infer the audio quality of the different files relative to the rest of the files in their cluster, achieving more promising results in our test setup than the current state-of-the-art.

In this first chapter, we briefly described the main objectives of this work (section 1.1), introduce the project that motivated this thesis' goals (section 1.2), give an overview of the proposed methods (section 1.3), and, finally, we summarise the overall document structure (section 1.4)

1.1 Objectives

The abundance and ubiquity of user-generated content experienced in social networks nowadays (figures 1.1 and 1.2), generated the need to organise and analyse such vast and heterogeneous content. Such properties are fuelled by the increasingly need of sharing events in social media, together with the technology advancements that made all of this high-quality sharing possible in the first place. In this work, we focus on the audio content of several user and professional recordings of different concert songs from different festivals. Concert recordings represent a good use case for testing and validating the different proposed methods in this thesis since there are several recordings reporting the same event at a given time. Such redundancy enables the audio synchronisation between

the different samples, which consecutively allows their grouping and quality inference by the execution of the proposed methods in this thesis.

User-generated content often contains audio cues, either by belonging to a certain video file or audio clip, and therefore the analysis of the audio features might promote a better understanding of that content. Such understanding would then lead to a better handling of ubiquity data and would ultimately provide a better experience to the end-user, which in our use case could be seen in the availability of a full length recording of a given concert song composed by joining different smaller length existing recordings together, or by only presenting the higher quality recording to the user at a given time.

Figure 1.1: Smartphone video capture in a music act. Source: [8]

Figure 1.2: User-generated Content creation. Source: [8]

The objective of this research work is to analyse audio data derived from content uploaded by end-users on social media platforms and derive insights of the investigated data, such as for example performing audio clustering based on the existence of overlapped audio segments between different audio files, with each cluster representing an event that is reported by several audio recordings.The work to be developed is meant to consider only features obtained directly from the audio signal itself to organise and derive important information of the analysed data, such as audio quality. Such information would then be used for an application that enables the broadcasting of user-generated

content that will be further implemented in COGNITUS [8], a European project that provided the main background and motivation for this thesis' work, which will be described in the next section.

1.2 COGNITUS project

The research of this thesis started from a research grant under the European Union's Horizon 2020 funded project COGNITUS. This project involves eight European partners, both from the industry and academics, namely BBC R&D, Queen Mary University of London, FORTH, Forthnet, Arris, VITEC, and Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, with each of the partners having both individual and collaborative tasks.

The main aim of the project is to provide ultra-high definition (UHD) broadcasting generated from user-generated content, given its abundance in the social networks activities. This quantity and ubiquity of information allied with the high-quality of recording now possible with the current state of the broadcasting devices (*e.g.*, smart phones, tablets) served as basis for the project's objectives. Therefore, the end goal of COGNITUS is to deliver the proof of concept of a solution that supports high-quality and enables the upload of user-generated content, while promoting the enrichment of the end user's broadcasting experience in form of an application.

In this vast and collaborative process, FCT/UNL efforts are spread across several intimately related topics such as video annotation and analysis, event summarization, data quality inference, and collaborative audio composition. In order to integrate all these processes, an abstract view over the different assignments shall be made.

Given the need to handle the vast and heterogeneous social media data, stream indexing and searching is an essential starting point of the whole procedure. Next, some filtering is necessary to exclude the data that is not a good representative of a certain event. To achieve this, some quality control has to be performed, which, in the scope of this thesis, can be made through audio inference of a given clip. The end point of this process is to perform data summarization, where high-quality data related to a certain event must be gathered. For this, temporal alignment is essential since it is the starting point to perform the clustering based on audio cues (*i.e.*, audio clustering). These different steps are illustrated in figure 1.3.

Throughout this thesis we will propose methods that will enabled the quality control step, by inferring the quality of the audio files in a given dataset, and also the semantic and temporal alignment set, by clustering the different audio files in the different existing events, and by additionally representing how they related with each other in terms of their time offset. A more detailed explanation of this integration steps is further given in Chapter 4.

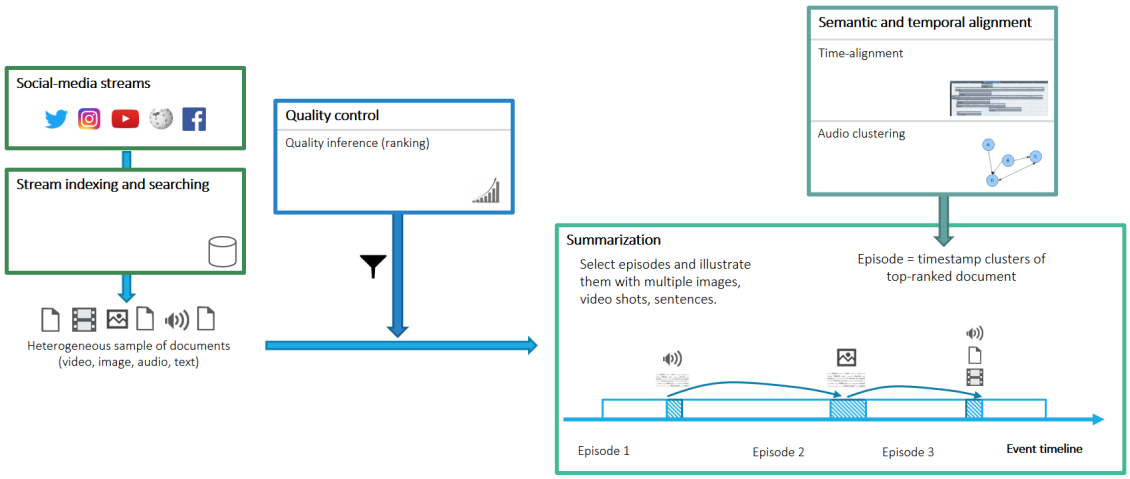

Figure 1.3: Overview of the main project steps, concerning the FCT/UNL work plan.

1.3 Proposed solution

As mentioned above our goal is to analyse and organise user-generated content containing audio cues. Hence, this dissertation focuses on the organisation of different audio recordings that are relative (*i.e.*, report) a given event. Considering the use case of this thesis being concert recordings, an event is considered to be a certain concert song. Since it is very likely to exist different recordings of the same concert song, the audio files are clustered by having common audio segments, with all recordings relative to a given concert song belonging to the same cluster in the end. Moreover, the distribution in terms of time of the different recordings throughout the overall event's timeline is also retrieved. We also focus on audio quality inference by attributing to each recording a quality score relative to the other recordings in its cluster.

Since the treated audio files are generated from user recordings, some challenges need to be tackled such as the different recording devices possibly used, and the different qualities inherent to each device. Moreover, it is very unlikely that two recordings are time synchronised and have the same duration. Thus, the development of this work will enable a better comprehension and management of a possibly large dataset of audio files by performing audio clustering and how the different audio files inside each cluster are distributed over time, as well as a better retrieval of information based on each file's quality inference.

In practical terms, the quality and clustering information retrieved by our method can be used to organise a list of audio files into the different events represented by the different clusters (figure 1.4), and further aid in the choice of which overlapped recordings to play to the end-user at a given time based on their quality scores (figure 1.5).

Several steps must be followed in order to perform such grouping and quality analysis over a database of audio files. First, the user generated files must be crawled from social media websites, then, to promote a better comparison between the different files, their

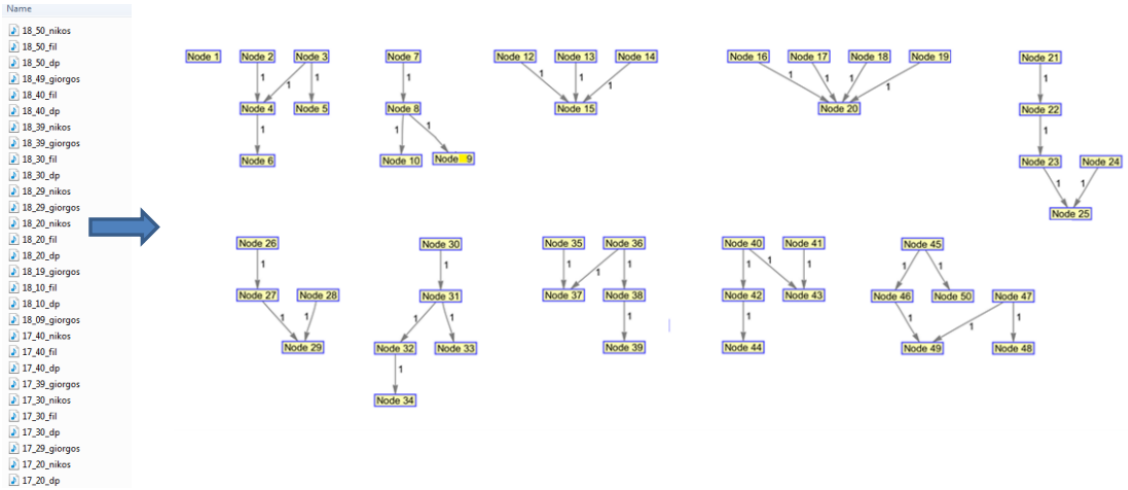

Figure 1.4: Grouping of different audio files. Source: [44].

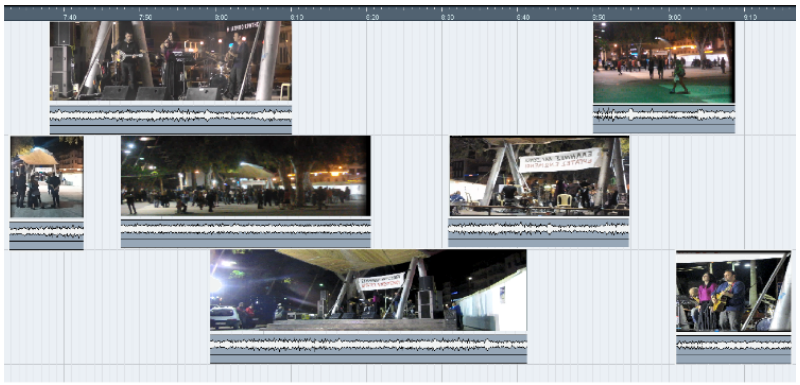

Figure 1.5: Time-aligned audio files. Information about the quality of each file is very useful to choose which one of the audio clips should be played. Source: [9].

synchronisation is required (*i.e.*, if they are relative to the same event). In our approach, the synchronisation results will serve as the basis to perform the grouping of the files in the different clusters, and enable to know how the different files are distributed over time in the respective clusters. The audio features used to cluster the data will ultimately help in deriving the quality of each file inside a given cluster. The detailed description of each one of these proposed methods is further presented in sub-sections 4.3, 4.5, and 4.4, respectively. However, since the synchronisation results that all these methods rely upon might contain false positives, two different filtering approaches are proposed to filter such false positive matches in sub-section 4.6.

Figure 1.6 describes these methods in a diagram-like view. Starting with a dataset of user-generated audio files, we perform their synchronisation whilst filtering the false positives matches, and we proceed on using the information on the true positive matches to perform the clustering and segmentation of the different files inside each cluster. Finally, we infer the quality of the different audio files both relative to the rest of the files in their clusters, and as well of the different audio files with the same overlapped common audio

segments (sub-section 4.5.2). The validation of the different proposed methods is given in Chapter 5.

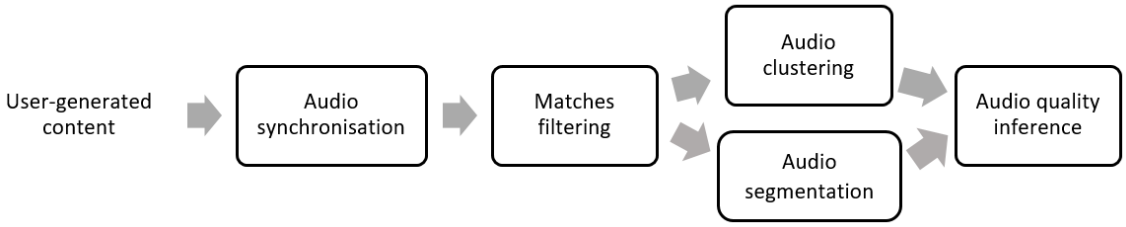

Figure 1.6: Diagram of the different methods needed to achieve the organisation, segmentation, and quality analysis of a large dataset of audio files.

The proposed work will be integrated with other COGNITUS' working components, more particularly for the ranking of the audio files based on quality score and the grouping of different audio files representing different events. Important to notice that such integration mechanisms must be incorporated and developed in the future but are not part of the work presented in this thesis.

Summary of contributions

The main novel contributions of the presented work are considered to be the following:

- Proposal and implementation of two new approaches to detect and filter false positives matches from the audio synchronisation phase. The first proposed approach analyses the derivatives of features derived from the synchronised matches to try to detect sudden drops that might indicate a false positives match, whilst the second uses traditional machine learning classification techniques to classify a match as a false positive or true positive.

- A novel way of inferring audio quality inference of the different audio files relative to the rest of the files in their cluster and overlapped segments that outperforms the current state-of-the-art.

- The creation of a considerably large dataset of different concert recordings manually crawled from YouTube containing about 200 recordings across 23 concert songs from different festivals. This dataset was used to further validate and evaluate the different methods proposed throughout this thesis.

The proposed methods were presented in two different (accepted) scientific papers, with their content and further information found in Appendix A.

1.4 Document outline

This thesis is composed of four chapters: **Introduction** (chapter 1) provides an overview of the objectives and the motivation for the work presented in the upcoming chapters; **Fundamental Concepts** (chapter 2) introduces some of the concepts that are important to be aware of in order to fully understand the presented work, namely the basics of sound and an introduction to hierarchical clustering; **State-of-the-art** (chapter 3) presents some of the current research state of concerning audio synchronisation and audio clustering; **Proposed solution** (chapter 4) gives an outlook of the research proposal as well as a detailed description of the algorithms proposed for the different methods; **Evaluation and results** (chapter 5) presents the validation tests made for each one of the proposed methods; and, finally, **Conclusion** (chapter 6) overviews the developed methods and elaborates on possible future work.

Chapter 2

Fundamental concepts

In order to promote a better understanding of the work and research of this thesis, some concepts are reviewed in this chapter. All the presented information in this chapter can be viewed as a survey of both Yost's book *Fundamental of Hearing* [56], serving as an introduction to sound, and the lecture notes of the Machine Learning course lectured at FCT/UNL [23], promoting a better comprehension of the machine learning concepts used throughout this thesis. Hence, readers who are familiar with the concepts described bellow can safely skip to chapter 3.

2.1 The basics of sound

In order to produce sound, an object must simply have the ability to vibrate. This ability depends on the properties of inertia and elasticity of each object, that are defined by the force that must be exerted in a given object for it to move and by the ability of that given object to return to its initial state, respectively.

These are the only prerequisites for possible sound production by the direct analysis of the physical definition of sound [56]. In order to achieve actually hearing, the sound wave needs a medium to propagate itself to ultimately be received as input to our auditory system. The pressure changes in the sound wave makes us witness and recognise different sounds.

2.1.1 Sinusoids

The most simple type of vibration is a sinusoid or sine wave, that is used to describe the continuous and regular displacement (*i.e.*, the distance an object moves) of a vibrating object. In this particular case, a sinusoid is symmetric in terms of the resting point and repeats perfectly over time. Figure 2.1 displays a complete transition (*i.e.*, a cycle) of

a sinusoid. This particular type of vibration is extremely important since all complex vibrations can be represented as a sum of sinusoids (and therefore also defined as a composition of simple vibrations).

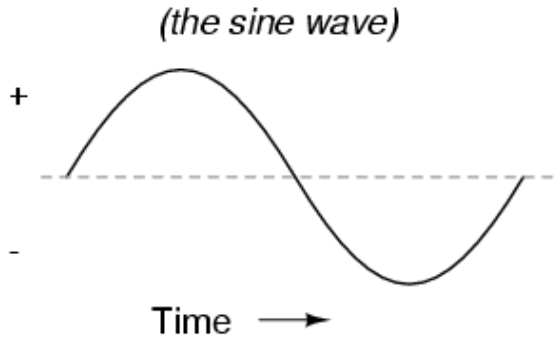

Figure 2.1: Sinusoid relation between displacement and time. Source: [3]

A particular sinusoid is specified by three parameters - amplitude, frequency and starting phase, and, given all vibrations can be defined as a sum of sinusoids, the same parameters apply. Unless all parameters are specified, there is ambiguity since the representation can correspond to more than one wave.

2.1.2 Sound wave properties

Now, we briefly describe the different parameters that characterise a sinusoid: amplitude, frequency, and starting phase.

2.1.2.1 Amplitude

Amplitude can be simply described as a measure of displacement (*i.e.*, how far a certain object moves). Figure 2.2 shows this measure as simply being the height of the wave.

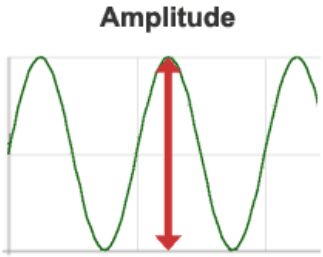

Figure 2.2: Amplitude representation on a waveform. Source: [28]

Amplitude can be specified in several forms. It can be displayed in an equation that varies over time, which enables the calculation of instantaneous amplitude. When temporal information is not relevant, as in periodic and continuous waves, some other types of amplitude metrics that do not rely on time are enough to describe the amplitude. Namely, **peak amplitude** describes the maximum positive displacement distance of a

waveform, whilst **peak-to-peak amplitude** refers to the distance between the maximum positive and maximum negative displacement. These values may be enough in many situations when analysing the amplitude of a given waveform, particularly in the case of periodic waves.

Even though these metrics are useful in some cases, they may give insufficient information when analysing the behaviour of complex waves, as the instantaneous amplitude variations cannot be represented in such manner. **Root-mean-square amplitude** enables the retrieval of more information about the wave behaviour, by squaring all amplitude values (to turn the negative amplitude values to positives), taking the mean of the squared values, and then taking the square root of the mean in order to go back to the original amplitude scale [54]. Thus, **root-mean-square amplitude** is widely used as an amplitude measure since it enables the representation of amplitude in both simple and complex wave forms, like noise. One drawback of this approach is that the calculation of the amplitude in a given moment of time (*i.e.*, instantaneous amplitude) is not possible.

2.1.2.2 Frequency

Frequency can generally be referred to the measure of how often an object oscillates. This is practically observed as the number of cycles a waveform completes per second, and it is represented by *hertz* (Hz). Thus, a vibration with frequency equal to 1 Hz means that it takes 1 sec for a wave to complete 1 cycle.

Theoretically, a complete cycle is defined when a vibratory patterns begins and ends at the same point after taking all possible values. The amount of time (in seconds) a vibration takes to achieve a complete cycle is called **period**. Therefore, frequency and period are intrinsically related as seen in the following equation:

$$frequency = \frac{1}{period}$$

These concepts can also be used to describe complex vibrations, under the condition that they contain some sort of periodic pattern that is repeated. Figure 2.3 shows an example of frequency variation in a sound wave.

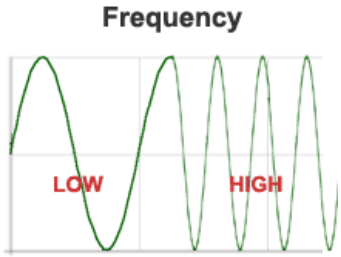

Figure 2.3: Frequency variations on a waveform. Source: [28]

2.1.2.3 Starting phase

Starting phase refers to the initial position of the object relative to the rest position. This is the initial phase of the sinusoid that describes the periodic movement of the object, that is the angle of the sinusoid at time 0 s. The measure is made in terms of degrees of angle and they are all relative to the state observed in the zero degree phase.

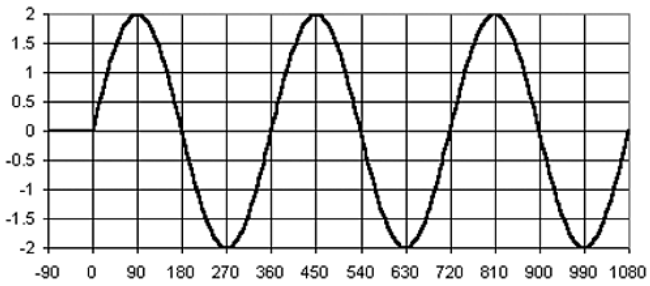

Figure 2.4: Sinusoid split in sections of phase in degrees. Source: [41]

When considering a sinusoid, by combining it with trigonometry as explained in [56], a complete cycle can be also represented as 360º. Figure 2.4 shows how a sinusoid is divided in different phases. Following this principle, a sinusoid with a phase angle of 270º, starts in the three-quarters before completion of the complete cycle of a zero degree phase sinusoid. This situation is illustrated in figure 2.5.

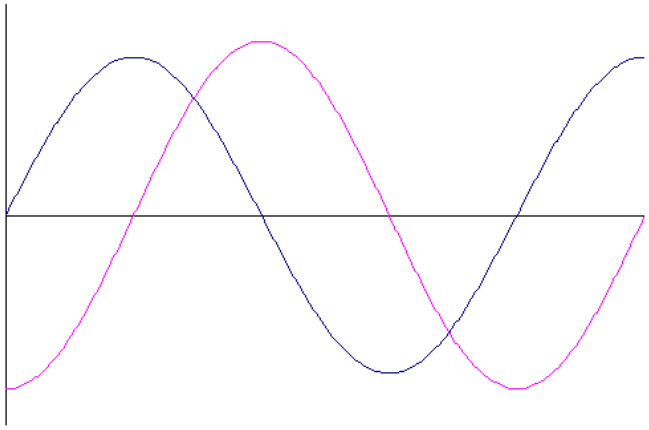

Figure 2.5: Representation of a zero-phase sinusoid (in blue) and a 270º angle phased sinusoid (in pink). Source: [1]

The notion of what the phase would be in any moment of time is called **instantaneous phase**, being the starting phase a particular value of it when t = 0.

2.1.3 Fast Fourier Transform

The **Fast Fourier Transform** (FFT) algorithm serves as a basis to compute the Fourier Transform of a (digital) audio signal with a faster and computationally less expensive

algorithm than the Discrete Fourier Transform (DFT). Its main practicality is the ability to convert a signal from time-domain to frequency-domain. The inverse can also be done by using the Inverse Fast Fourier Transform (IFFT) algorithm.

FFT serves as the basis for sound processing, since a lot of our perception of sound comes from its frequency-domain representation [56]. This can be easily observed by the ability of humans to identify and be extremely sensitive to changes of the pitch of musical songs, and, consecutively, to changes in the frequency of the audio signal. Pitch is a perceptual measure related to the fundamental frequency, which is the inverse of the period of repetition of complex waves.

2.1.4 Spectrogram

A spectrogram is built from a series of **spectra** that represent the frequency and amplitude of a signal in a moment of time. The stacking of several spectra over a period of time, provide the reasoning behind the creation of a spectrogram. A **spectrogram**, which is obtained with the Short Time Fourier Transform (STFT), is then a representation of how the spectrum of a certain signal changes over time. Even though STFT retrieves how the phase and magnitude of the signal varies over time and frequency, we will give more focus in the magnitude in this thesis since it consists of one of the features used in relation to frequency in the audio fingerprinting algorithm that will be later described in sub-section 3.1.6.

Additionally to the frequency being represented along the vertical axis and time along the horizontal axis, a spectrogram also represents energy in terms of magnitude at a given time and frequency. This is accomplished by a contour map drawn behind the axis normally varying within a colour or grey scale, that is generated from the magnitude valued matrix that represents the magnitude spectrogram. Thus, variations in the energy will produce variations in the background colour, creating a third dimension in the graphic representing the magnitude. Figure 2.6 shows an example of a spectrogram.

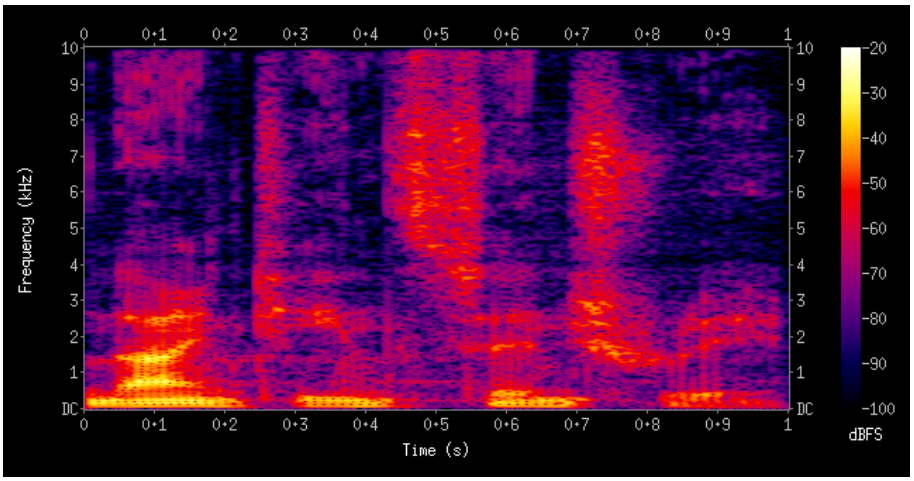

Figure 2.6: Spectrogram captured of a recording of some words. Source: [53].

2.2 Classification techniques

The process of predicting the class or category of an example from a finite set of samples is defined as a classification problem in Machine Learning, as illustrate in figure 2.7. This process can either be done by learning from a given set of samples to which the corresponding class is known (*i.e.*, the training data is labelled), or by predicting the classes of new samples without any ground-truth. The first approach is called **supervised learning** whilst the second is described as **unsupervised learning**. A merge of the two approaches can also occur, where there is both labelled and unlabelled samples in the training set; this approach is called **semi-supervised learning**.

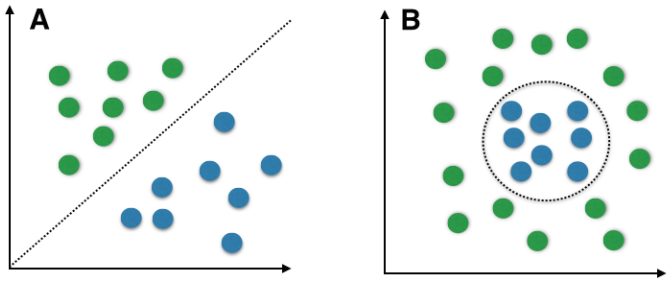

Figure 2.7: Example of two classification problems. The samples of each class are represented with different colours. On the left, the problem is linearly separable since all samples of each class can be separated by a line. On the right, the problem is non-linearly separable and another approach had to be used to separate both classes (a circle with radius r). Source: [42]

These different types of learning are closely dependant on the type of problem we are trying to solve and the type of data we are dealing with. For example, if we have access to a large amount of labelled data and the set of possible labels is known *a priori* and is static, following a supervised learning approach might suffice to achieve good prediction in the newly classified samples. On the other hand, if we want to derive new conclusions about a large set of unlabelled data, following an unsupervised learning approach suits best our needs.

In the scope of this thesis' problem we will focus on supervised learning. The features of our data are given as input to the classifier , which will use such features to predict the classes of new samples. Moreover, since we are dealing with labelled data, we can optimise our learning process by making the classifier predict some of the labels of our training samples and compare the its predictions with the corresponding ground-truth label of each sample. However, optimising our classifier to fit and adjust to our training samples might cause overfitting, which is undesirable since it prevents our classifier to generalise to new samples that were not learned in the training phase.

One simple way of preventing overfitting is to split the dataset in 3 separate sets: a training set, used solely to fit our model, a validation set, to select the best hypothesis that minimises the error (*i.e.*, a measure of wrongly predicting the labels of samples of

this set), and a test set, to estimate the true error of our model. Since none of the samples in the testing set is used to train our model, we can assume that this estimate is unbiased and therefore a valid measure of evaluation of our predictions. The next subsection (2.2.1) presents a more elegant and better approach to prevent overfitting, and it is followed up by 3 subsections that explain three of the most used methods in the context of solving classification problems with supervised learning.

2.2.1 Cross-Validation

Using the approach of splitting the dataset in training, validation, and testing sets to prevent overfitting is not ideal since it only takes into account one iteration over the randomly chosen samples in the different sets, affecting the modelling capability compared if done iteratively and with a larger amount of samples per set. Moreover, using this approach would also only take into consideration one hypothesis over the several possible hypothesises of our model.

With cross-validation, the samples in the dataset are randomly split into k disjoint folds (being k a number from 2 to the number of samples) and the model is trained with all folds except one, being the left out fold used for validation. This process is repeated for all folds and in the end the average of the validation error of all hypothesis is considered to be the true error of our model. The separation of the samples over the different folds should preserve the percentage of samples of each class in the original dataset, meaning the folds are stratified. This is important since it maintains the information of the original dataset, that therefore should reflect the real context of our problem.

Using cross-validation is a better approach than simply separating the dataset in 3 (*i.e.*, training, validation, and test set) since it tests different hypothesis of the model, by performing several iterations, and therefore by using more data to both train and validate the model since no samples were put aside for the test set. Thus, cross-validation is more likely to produce better predictions and to better generalise our model in the end since different training and validation sets were taken into consideration over the several iterations.

2.2.2 Logistic Regression

Despite being a regression model, logistic regression can also be used to solve classification problems. This is easily achieved by finding an hyperplane that separates the two classes where finding an example x of any of the classes lying in that hyperplane has equal probability in both cases.

$$P(C_1|x) = P(C_0|x). \tag{2.1}$$

This goes against the nature of a regression model, where the decision boundary is set to approximate the probability of every point to be in a certain class C [23]. Note that for

handling multiple classes one would need to generalise this solution and use Multinomial Logistic Regression but since the problem we tackle in this thesis is a binary classification problem, only two classes will be used and therefore the notions presented will suffice.

This modification is essential when dealing with classification problems since it makes the isolated points located further from the decision boundary to not have a big effect in its overall position. Thus, the ultimate goal when trying to classify a sample is to simply put it in the correct side of the decision frontier, that will consequently correspond to a correct prediction of its class.

The function used to differentiate the classes is a logistic function, and it can be directly obtained by the rearrangement of the conditional probabilities based equation (2.1), with its final form being given by:

$$g(\overrightarrow{x}, \widetilde{w}) = \frac{1}{1 + e^{-(\overrightarrow{w^T x} + w_0)}} \tag{2.2}$$

By being able to receive any real number as input, and outputting a value between 0 and 1, logistic functions are a good way of obtaining a probability of an example previously served as input. Furthermore, logistic functions have the property of varying a lot around a threshold but being nearly constant away from that threshold, as shown in figure 2.8. This is a good feature when dealing with the most further away points and how they affect the output and consecutively the decision boundary position.

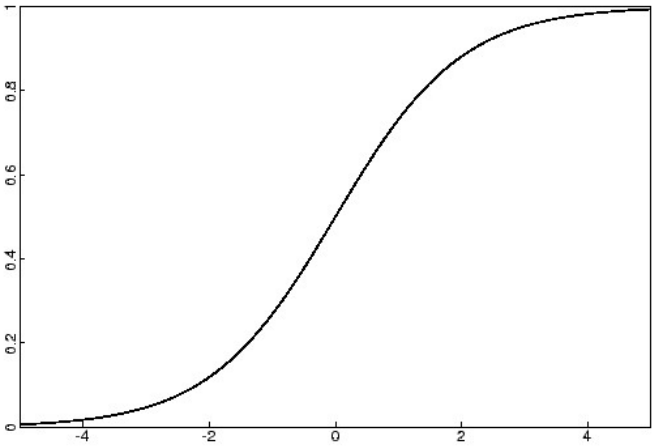

Figure 2.8: Example of a Logistic function. Source: [18]

Estimation using a Logistic Regression classifier is made by choosing the function parameters that maximise the likelihood of observing the expected sample values in our hypothesis [27]. This way we ensure that the chosen hypothesis fits well our training data, while concerns about possible overfitting are handled by fine tuning the regularisation parameter c. This parameter directly influences the slope of the logistic function, with a low value of c meaning less regularisation, and an increase of probability of overfitting the

training data. In practical terms, the parameter c influences the weights of the coefficients of the chosen hyperplane that separates both classes, as follows:

$$\frac{1}{c}\sum_{j=i}^{m} w_j^2 \tag{2.3}$$

meaning that bigger vector weights will cause a higher slope, while decreasing the weights will cause a smoother logistic function and therefore less overfitting.

One can use cross-validation to estimate the best value of this parameter c by seeing which model achieves best predictions (*i.e.*, lowest validation error) over the several iterations.

It is important to notice that this is a linear classifier, since it simply tries to find an hyperplane that divides the two classes. Therefore, if the classes are not linearly separable by default this classifier will suffice in correctly setting its decision boundary. To handle this problem we can increase the dimensions of our features (*i.e.*, we can either try to find more independent features of our samples or combine the existing features into new, dependant, ones). By increasing the feature space and consecutively increasing the computational power as a drawback, it is then possible for our linear classifier to correctly separate our training data.

2.2.3 K-Nearest Neighbours

Instead of fitting a model based on the samples in the training set, the K-Nearest Neighbours classifier simply compares never seen samples to the samples of the training set. The basic idea is that a new sample is given the same class of the majority of the k nearest samples in the training set (being k an odd number between 1 and the number of samples in the training dataset). A small k is more likely to promote overfitting, whereas a large k promotes underfitting (see figure 2.9). Thus, it is important to find the best number of neighbours by applying cross-validation, similarly to what was already described when choosing the c parameter of the Logistic Regression classifier.

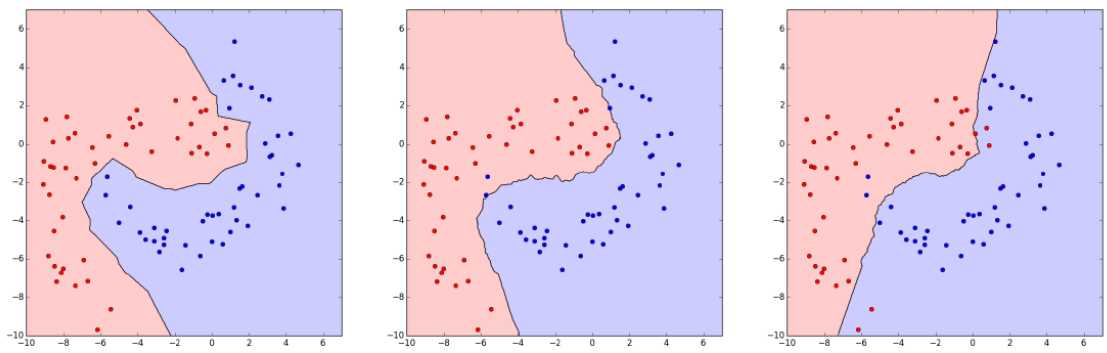

Figure 2.9: Comparison of using a different number of neighbours for the classifier (1, 13 and 25, respectively). Source: [23]

Since this is a distance-based method, a distance function must be chosen to calculate the closest training sample points to the new sample point in the feature space. Euclidean distance is the most used but another distance metrics might produce better results in specific problems, such as the Minkowsky distance for continuous numerical features, and the Hamming distance for categorical features [23].

Unlike the logistic regression classifier, increasing the dimensional space of our features will not necessarily produce better predictions when using the K-Nearest Neighbours classifier. This is inherent to any distance-based classification method since more dimensions normally mean points are more distant of each other and so are their neighbours, meaning that the class of the majority might be incorrect and wrongly predicted for a new sample. Thus, when dealing with high dimensional feature spaces and a high value of k, it might be better to analyse the distance of the points, by introducing a weight based on the distance metric [43]. This way closer neighbours would influence more the class prediction than the furthest neighbours.

2.2.4 Support Vector Machines

Similarly to the Logistic Regression classifier, Support Vector Machines (SVM) finds a hyperplane that separates two classes, assuming that the classes are indeed linearly separable. The main difference, however, is that with SVM the chosen hyperplane is the one that maximises the distance between the two classes. This solves an important problem of Logistic Regression, that consists of allowing the frontier to be closer to some points of a given class than the others. This is originated by the logistic function being almost flat away from the frontier, making small differences in the position of the decision boundary to not make any difference in the overall result. This results in the frontier being placed in a different position when we run the Logistic Regression Classifier several times, as shown in figure 2.10.

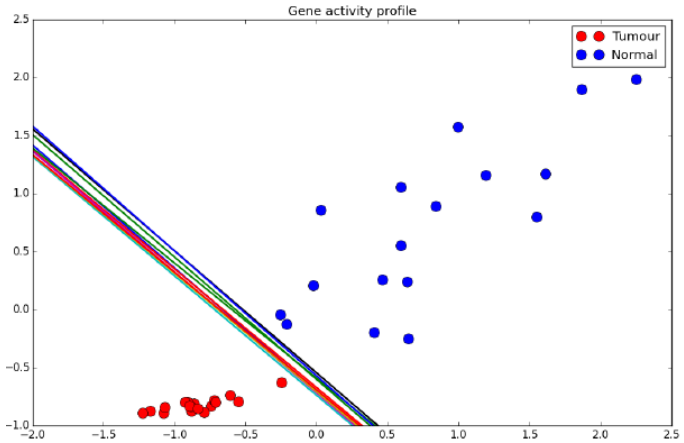

Figure 2.10: Different runs of the Logistic Regression classifier result in different positions of the frontier that separates both classes. Source: [23]

To maximise the margin between the frontier and classes, we look in the distance between the closest points of each class and the frontier (such points are represented in a vector form and described as support vectors). The goal is therefore to maximise the distance between the support vectors and the decision hyperplane, as demonstrated in figure 2.11. The chosen hyperplane can be formally defined as:

$$g(\overrightarrow{x}) = \overrightarrow{w}^T \overrightarrow{x} + w_0 \tag{2.4}$$

where $\overrightarrow{w}$ represents a vector of weights and w_0 the bias. This function, when served with an input vector (*i.e.*, one sample), returns values greater or equal than 1 for one class and values smaller or equal to -1 for the samples of the other class.

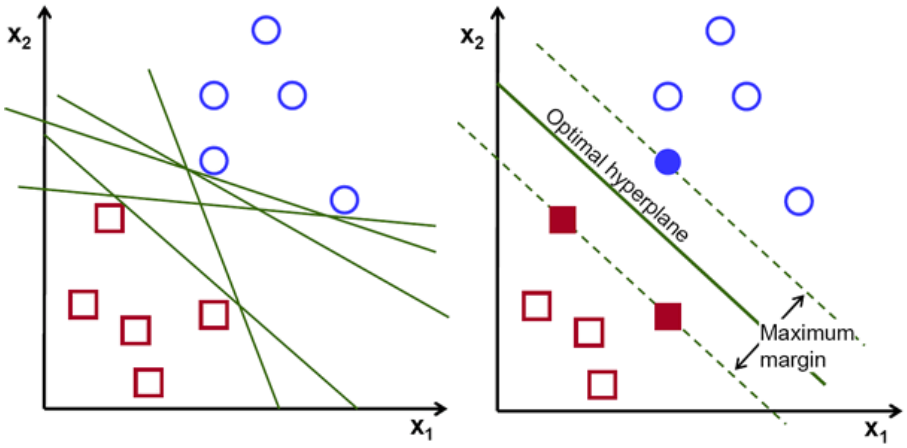

Figure 2.11: On the left, different possible hyper planes that successfully separate both classes are shown. On the right, only the hyperplane that had the maximum margin from the closest points of each class (denoted as support vectors). Source: [37]

The absolute value of the output of the function represents the distance from the input vector to the hyperplane, being this distance equal to 1 if the input vector is a support vector. Since the shortest distance between a point and a hyperplane is defined by:

$$d = \frac{|g(\overrightarrow{x})|}{\|\overrightarrow{w}\|} \tag{2.5}$$

and for support vectors $|g(\overrightarrow{x})|$ is equal to 1, one can maximise the margin between the frontier and the support vectors by minimising $\|\overrightarrow{w}\|$. This is a non-linear optimisation task that can be solved by using Lagrange Multipliers [45].

This process only gives a solution (*i.e.*, finding the hyperplane that maximises the distance) if the two classes are linearly separable, which might not always be the case. To fix this problem, a slack variable must be added to each weight vector, representing weather the point is on the right or wrong side of the margin, enabling this way a solution to our problem by loosen up the initial constraints. If the corresponding vector of the point is inside the margin, a value between 0 and 1 (exclusive) is given, whereas if the point lies in the wrong side of the hyperplane, a value greater than 1 is given. This

slack variable then represents the distance between a vector and the inside margin of the hyperplane.

By introducing the slack variables to each vector, we allow some points to fall on the wrong side of the margin by introducing a certain penalty to them. These penalties are upper bounded by the regularisation parameter c, meaning a high value of c a high penalty for margin violation, with no support vectors being placed inside the margins, whereas a low value of c will allow several points to be placed in the wrong side of the hyperplane. This idea is illustrated in figure 2.12.

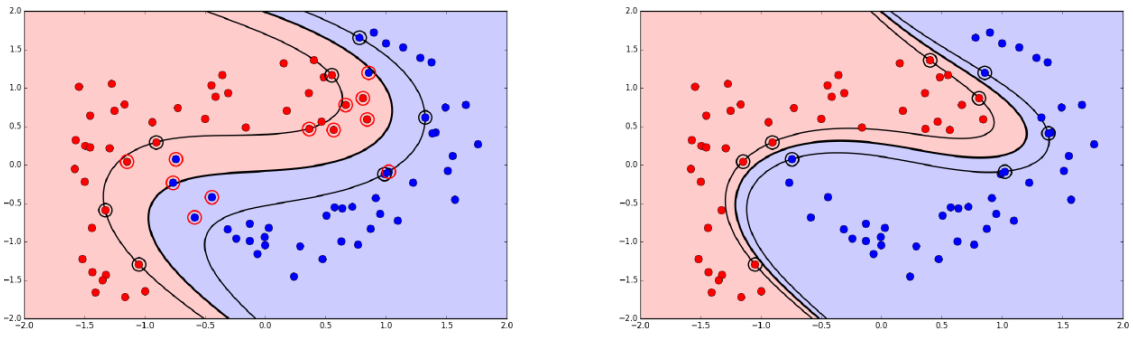

Figure 2.12: Different values of c were used for different runs of the SVM. For the left image a c value of 1 was used, where it is visible several support vectors being placed inside the margins (outlined with red circles), whilst in the right image c was equal to 1000 and no margin violations are observed. Source: [23]

Allowing some errors in the decision frontier might not work if there is too much overlap between both classes. In this case it is preferred to increase the dimensions of the feature space, as previously discussed. Increasing the number of features in SVM is considerably easy since we are only concerned about performing the inner product operations between the extended featured vectors. This operations can be done by using kernel functions, that return the inner product of the transformed vectors as a function of the original dimensional vectors. This is done by using an auxiliary function that adds dimensions to the original vectors (*e.g.*, receives a 2-dimensional vector and returns a 4-dimensional vector).

Different kernel functions (*e.g.*, linear, polynomial, Gaussian) can be used and tested, with the Gaussian Radial Basis Function (RBF) being normally set as the default kernel to use in non-linear solutions, since it maps the training examples into a higher dimensional space. Important to notice that this kernel has the same performance as a linear kernel using the right combination of parameters, and it is less complex than the polynomial kernel, since it uses less hyper parameters to select the best model [20].

Using the RBF kernel then gives the chance of tuning one more parameter, γ, that defines how far the influence of each training example reaches [24], and, consecutively, of each support vector. Thus, a high value of γ will make the kernel weight nearby points more heavily, influencing on the decision frontier to be placed closer to the example points and possibly promoting overfitting. A low value of γ will smooth and expand the

decision frontier [23]. A good way of finding the right combination of this two parameters (*i.e.*, c and γ) is to execute an exhaustive search over all possible combinations of a subset of possible values for each parameter, also described as grid-search. This process can be optimised by using cross-validation, choosing the pair of parameters' values that had the highest mean accuracy in the different folds.

2.3 Hierarchical Clustering

If we wish to organise and group samples together based on a certain similarity, we are dealing with a Clustering problem. One approach to decide the right number of clusters in a big dataset, is to do it in an iterative manner by joining first the samples that are more similar and then gradually merging similar clusters. This approach is called hierarchical clustering and is illustrated in figure 2.13, where the letters "a" to "f" are isolated and then are iteratively combined to form the combination "abcdef".

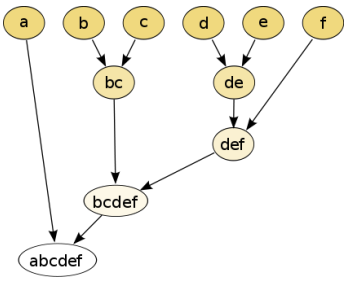

Figure 2.13: Example of hierarchical clustering. Source: [52]

This type of clustering suggests that is not only needed to find measures of similarity between samples, but also between clusters, so they are rightfully merged. These cluster measures can be represented by a simple Boolean value, where 1 indicates that two clusters are similar and 0 the opposite, or by more complex measurements like the distance between clusters. There are a lot of ways to perform the calculation of this distance (for more details see [23]). In this work the measure of similarity is set to be a Boolean, since that is what suits the best to our problem and presented solution.

Hierarchical clustering can be done by either starting with singleton clusters and proceed on merging similar clusters (*i.e.*, agglomerative clustering) or by following a top-down approach, where a single cluster with all examples is formed and then iteratively split into smaller clusters (*i.e.*, divisive clustering). Figure 2.14 shows an example of agglomerative clustering, where initially all samples are separated (*i.e.*, forming singleton clusters), and then are grouped together until they are all gathered in the same cluster.

Our proposed solution uses an adaptation of agglomerative clustering to group audio samples from music concerts into different clusters (section 4.3). It is an adaptation since the measure of similarity used is not by evaluation of cluster distance but from metrics retrieved by the audio synchronisation phase, more concretely from the audio

fingerprinting algorithm described in subsection 3.1.6. This idea of the audio clustering process will be firstly introduced in section 3.2 and practically described with detail in chapter 4.

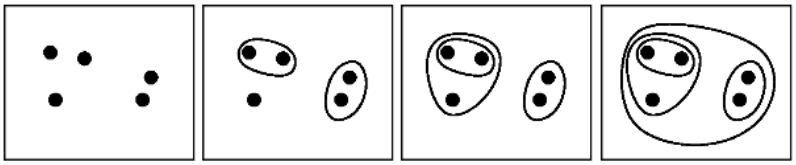

Figure 2.14: Example of agglomerative clustering. Source: [51]

Concluding remarks

In this chapter we introduced some basic concepts of sound that will enable a better understanding of the features used by the audio fingerprinting techniques that will be further explained in the next chapter. Moreover, machine learning methods together with some important supervised learning notions were also introduced. All the introduced concepts in this chapter create an important basis for the formulation of the different methods proposed throughout this thesis.

Chapter 3

State-of-the-art

In this chapter we present the state-of-the-art work by the time this research work began. We survey the different processes inherent to an audio fingerprinting algorithm, (section 3.1) and give more detailed information about the algorithm that was used in the actual implementation phase of this work. Furthermore, we introduce previous work that served as the basis of the different proposed methods in this thesis, focusing mainly on audio clustering (section 3.2).

3.1 Audio fingerprinting

The synchronisation of different audio files is an essential step to perform the audio clustering, since it is the mechanism used to group different files with common overlapping segments. Moreover, the information about the different audio files are distributed overtime inside a given cluster is made through the offset between the different samples. To derive the quality of the different files inside a given cluster, information about how the synchronisation process happened is also essential for the proposed method. The detailed explanation of these processes, and the different approaches that can be followed, can be found in sections 4.3, 4.5, and 4.4, respectively.

Audio Fingerprinting techniques are often used to perform the synchronisation between audio files by the usage of audio features that are relatively resistant to noise. This represents an ideal background to perform the synchronisation of our data since it enables to synchronise noisy samples against possibly less noisy audio files in the database, while only needing possibly short length common segments to synchronise both clips.

While audio fingerprinting has traditionally been used for song recognition, as made famous by *Shazam* but also other existing applications, such as *Midomi* [30], *Chromaprint* [29], and *ACRCloud* [2], its usage to perform grouping of a large set of audio files is still

relatively new ([22] is further analysed in section 3.2). In the latter case, instead of using the offset retrieved from the audio fingerprinting to align the audio files, that offset will serve to conclude that the files have a common segment and therefore should be simply put in the same cluster.

Fingerprint generation over any kind of data is an efficient mechanism to characterise possibly large data with a small representation. More explicitly, as an alternative to representations that involve large amounts of data, fingerprints are compact representations of the data that can be used for purposes that do not require dealing with all the intrinsic details of the data. This technique promotes a fast way of comparing the quality of two entities by trying to diminish the need to compare irrelevant features.

Any audio fingerprinting technique involves an extraction mechanism to generate the fingerprints of data and then a searching mechanism that looks for matching fingerprints in a pre-generated database. Different audio fingerprinting algorithms (such as [13, 16, 50]) can be compared based on their **robustness** (*i.e.*, how well the algorithm performs with additive noise), **reliability** (*i.e.*, the probability of having false positives), **search speed** (*i.e.*, how fast the algorithm can identify a match) and **scalability** (*i.e.*, how well the algorithm performs with a large database).

The variety of these benchmarks differ on how the algorithm is implemented and how the fingerprints are generated. Therefore, they are intrinsically associated with the chosen size of the fingerprints used, since it will directly influence the storage needed and search speed of the algorithm, and how many seconds are needed to identify a certain song (*i.e.*, granularity).

There are several steps inherent that have to be complete to achieve the match between two audio files. First, an extraction of some of the features present in the audio clip has to be done; second, those features, or a combination of them, are then used to generate a fingerprint that will characterise the audio clip in question; then, searching processes must be applied to access the fingerprints of the other files in the database; and finally matching mechanisms have to be performed to compare pairs of fingerprints and generate a match. Each one of these steps is explained with more detail over the next subsections.

3.1.1 Features extraction

How the fingerprints are generated from the features extracted from the audio signal is one of the main points of the choice of the audio fingerprinting algorithm to use. Some principals are inherent to almost every existing fingerprinting algorithm, such as splitting the audio signal in smaller pieces, and then proceed to extract features of such smaller sized portions. Regarding the first step, the splitting decision is on how tiny such frames will be, since excessive divisions can put an unnecessary computing power on the creation of the fingerprints, and a low number of divisions can suffice to create an efficient fingerprint representation.

Regarding the features to be extracted, the goal is to use features that are somehow

invariant to signal degradation such as: Fourier coefficients, with different and less computational expensive ways of calculating the Fast Fourier Transform (FFT) over the same samples [15]; Mel-Frequency Cepstral Coefficients (MFFC) [25], that have into the consideration the energy representation of the audio signal; statistics of features retrieved from Discrete Fourier Transform at each instant of time [4]; and several others amplitude driven methods as referred in [12].

Although each one of the extraction methods could be more beneficial to use depending on the context, it is known that most important features used by the auditory system to perceive sounds live in the frequency domain, as stated in [16] and already mentioned in subsection 2.1.3. Thus, algorithms that use Fourier Transform are preferable in the majority of the cases.

In order to illustrate an example of the features extraction process and how it simplifies the data representation, figure 3.1 represents the output of the features extraction used in *Shazam* [50]. In this case, the extraction is made through the selection of frequency peaks in the spectrogram of a given audio signal. This procedure is better explained in subsection 3.1.6, since the fingerprinting algorithm that ended up being used in the implementation phase is also inspired in this peak extraction concept.

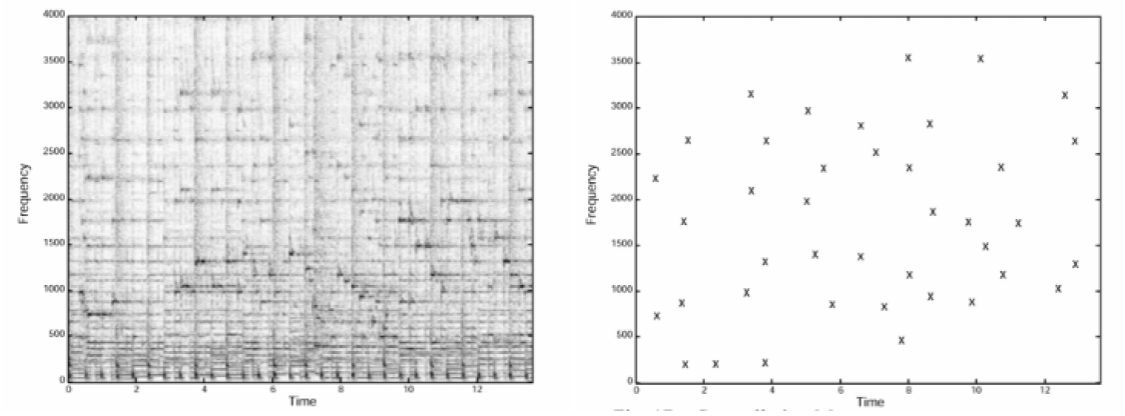

Figure 3.1: A spectrogram (in the left) with the correspondent extracted features on the right. Source: [50]

3.1.2 Fingerprint generation

After extracting the features from the data, the next step in any fingerprinting algorithm is to use such features to produce the hopefully correct and singular representation of the respective audio file. This generation process varies from different algorithms but in order to increase its singularity, it is important to combine multiple features previously extracted. This happens because single features from a certain moment of time normally do not have enough information to uniquely characterise an audio file. The set of all the information derived directly from the features that is needed to form the final fingerprint

of an audio clip is called sub-fingerprint, as introduced in [16]. Thus, one fingerprint can be seen as an aggregation of several sub-fingerprints.

These sub-fingerprints can be calculated not only using the raw values of the extracted features but also by performing combinations between such features, in order to promote a more unique representation of a given file. The way to relate features is therefore extremely important, since it might reduce the need of consider all extracted features but simply focus on their relation during certain periods of time.

One effective way of performing these combinations between single features was proposed by *Shazam* in [50]. Considering the peaks extracted in figure 3.1, the algorithm makes combinations between them to produce more detailed information about that audio file. This process is accomplished by choosing anchor points, that consists of a subset of points uniformly distributed in the set of the extracted peaks, and then combining each anchor point with other peaks inside a target zone, in a sequential way. This first part of the overall fingerprint generation process is illustrated in the left part of figure 3.2.

Since these peaks were retrieved from spectrogram, information about their frequency in a given moment of time is available. Thus, this combination is passed as input to a hash function, by giving the frequencies of both peaks and the time difference between them. This will result in a hash representation of that information (32-bit), ultimately consisting the sub-fingerprints of an audio file. This is demonstrated in the right part of figure 3.2. This whole process is then repeated for each anchor point, and then the list of all hashes will be used to characterise an audio clip, forming its fingerprint.

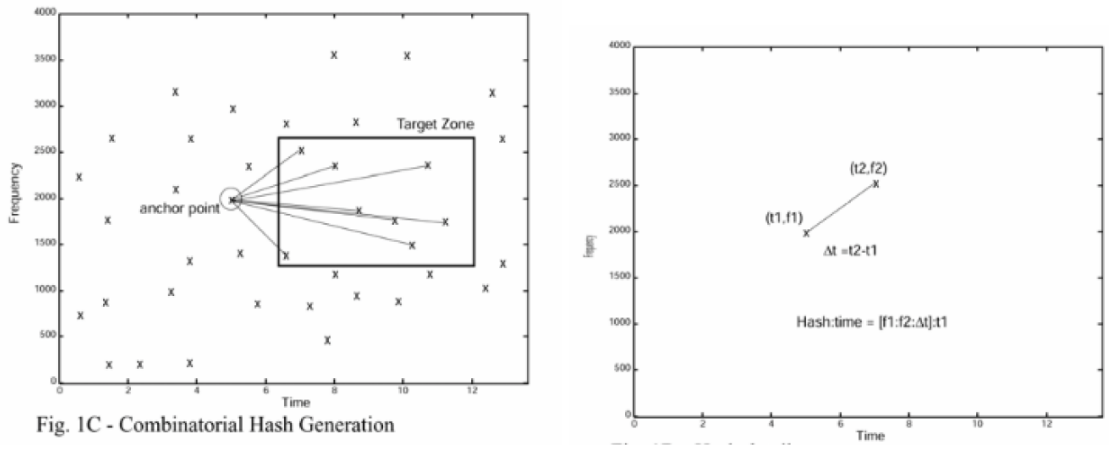

Figure 3.2: Description of the hash generation process in *Shazam*. Source: [50]

3.1.3 Robust audio hashing

The hash functions used in summarising audio data are based on a different notion than the ones used in cryptography. In cryptography, the hash value associated with certain

data is extremely fragile in the meanings of a very subtle difference on the content will result in a completely different hash value.

In the case of audio, there is a need to use a threshold since the audio files can be subject to degradation factors possibly originated by compression or conversion mechanisms. Thus, hashes of audio content imply that different quality versions of a given audio clip are given the same hash, as stated in [38] and [48]. This type of hashes are called **robust hashes**.

This is practically observed in the output of the fingerprinting algorithm described in subsection 3.1.6, where different versions of audio files are matched depending on the query file to match. This is shown by the display of how that version ranks in all hashes of that song.

This method of hashing audio content is widely used in audio fingerprinting, as opposed to audio watermarking [11] where some kind of signature is embedded in the original content before its release. Even though this technique is very useful in detecting copyright problems [17], it suffices to work in the context of our problem.

3.1.4 Searching process

Giving the likely abundance of audio files in the database, both when dealing with music recognition applications and data retrieved from social media websites, an efficient way of searching must be implemented. The size of the database is even more problematic considering that each song's fingerprint is composed by several bits of information representing the sub-fingerprints.

An extensive search of the entire database is not practical [16], especially when the response time matters which is a common requirement in most real-world applications. Thus, approaches for reducing the number of records to be accessed and/or strategies to optimise the way the information is saved and accessed must take place.

Such approaches vary in whether the information is indexed based on sub-fingerprints, which usually infers an exact match from the database (*e.g.*, searching of hash matches), or based on fingerprint-blocks, that consist of the whole information that characterises a given clip. This last approach normally infers that a similar match is needed and not an exact match, since two different clips with exactly the same fingerprint are unlikely to happen even if they represent different recordings of the same song.

When considering sub-fingerprint indexing, data structures used to store the information normally consist of hash tables ([16]), inverted index hash tables ([10]) or tree-structured based representations ([55]). These ways of representing the sub-fingerprints enable for a sparse representation of the information, therefore promoting practicability and an efficient way of searching [16].

When handling indexing of fingerprint-blocks, a reduction of the matching candidates is important to be implemented since the interest relies on finding similarities instead of exact matches. Thus, techniques to select the best candidates can take place, such as

building classes to group similar fingerprints ([7]) or inferring the similarity probability of a candidate ([16]).

3.1.5 Matching process

After obtaining the fingerprints, it is important to understand whether a match between two different clips is found or not. This decision process can be easily performed by following the methodology that if two clips have common sub-fingerprints (*e.g.*, hashes), then there exists a match between them. A minimum amount of matching hashes is normally taking into account to prevent false positives.

The matching process can be generalised as simply analysing the offset between the matching hashes of the clips that are being compared. This is done for every clip in the database, and clips with a sufficient number of matching hashes (usually a small number since the unlikeliness of false matches) over a given offset are considered to be a match relative to the query song.

This process can be practically achieved by the creation of bins for each audio file matched, with the added information of the offset between the two files. Thus, a bin is a representation of all the different matching hashes of a certain song in the database over several offsets. To identify the most frequent offsets of the different hash matches, one can generate a scatter plot of the different matches over time and check for diagonal lines as they indicate the same offset over different matches. This is illustrated in figure 3.3. Another way of identifying that frequency is simply by using a histogram and proceed to find relevant peaks (figure 3.4).

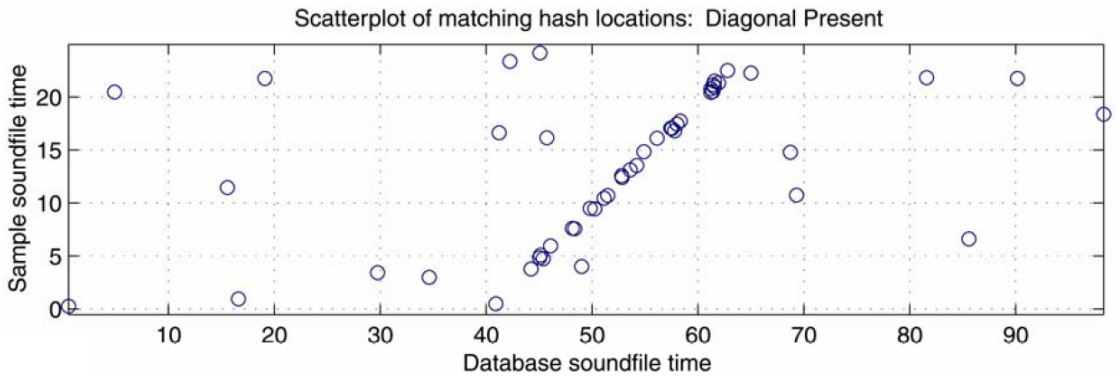

Figure 3.3: Scatter plot of the timestamps of the different matches. Source: [49]

This is the basis for the matching process used in the fingerprinting algorithm explained in subsection 3.1.6 and also commonly used in query-by-example (QBE) applications (term introduced by *Shazam* in [49]), common in all music recognition applications.

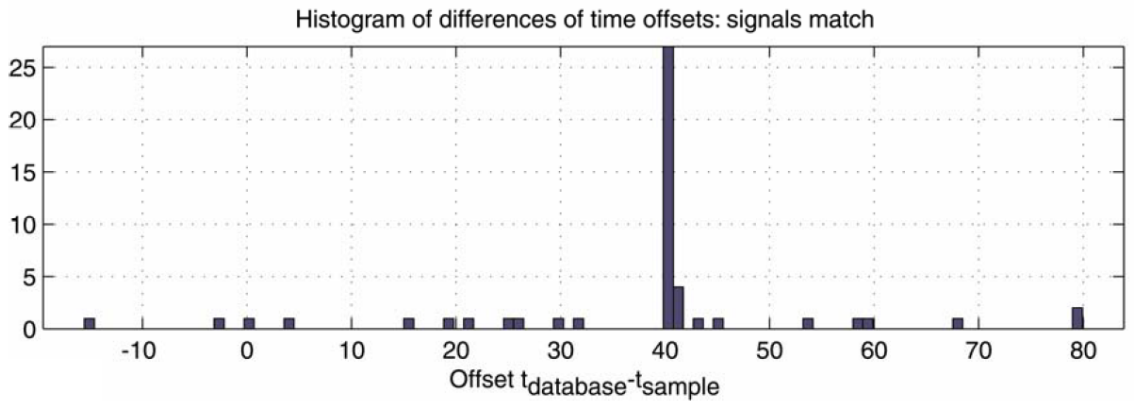

Figure 3.4: Histogram of the offsets of all matches. The real offset between the two clips is around 40 seconds, indicated by the peak in the histogram. Source: [49]

3.1.6 Landmark-based audio fingerprinting

Considering the accuracy of the *Shazam* algorithm, an approach based on the theoretical overview of their methodology used to perform audio fingerprinting (presented in [50]) will now be detailed explained. This algorithm is accessible in [13]. The system is set to be resistant against noise because the landmarks that characterise each track are created using frequencies, which are likely to be preserved under noisy situations. This is a very important property since the samples that will be fed into the algorithm in the later phases will be recorded from smartphones on overly crowded environments (*e.g.*, musical acts).

To better understand the system used, firstly we shall discuss the audio fingerprinting technique employed and what it involves. In this system, a fingerprint consists of a multiple number of **landmarks**, and its length is dependent on how long a song is and how many onsets (*i.e.*, the beginning of a musical note) are found. Each landmark consists of a pair of two peaks, similarly to the notion of anchor points and target zone already described in subsection 3.1.2.

The landmarks of a certain query song are presented in figure 3.5 as the red connect pairs of circles. Practically, each landmark contains the time stamp of when the first peak occurs, the frequency of the first and second peak, and the time difference between the two peaks.

To better understand the extraction of one or several peaks, figure 3.6 represents a frequency over magnitude graphic generated from a specific timestamp of the track used, more specifically frame 8817. A frame is represented as a column in the spectrogram showed in figure 3.5. Note that both of these figures were manually generated by us.

A song is then characterised by all the landmarks it contains, forming its fingerprint. To find if a query song matches a certain song in the database, the searching process has to take place, and only a small number of landmarks must match between the fingerprints of both songs in order to retrieve a match, due the unlikeliness of false landmarks matches.

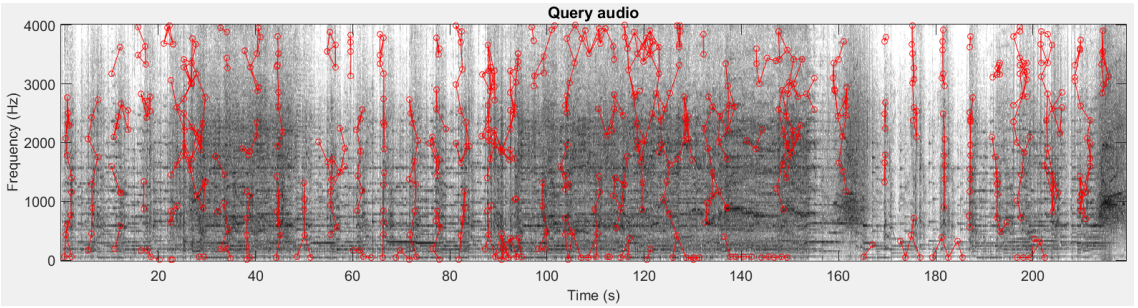

Figure 3.5: Spectrogram containing the landmarks of a query song.

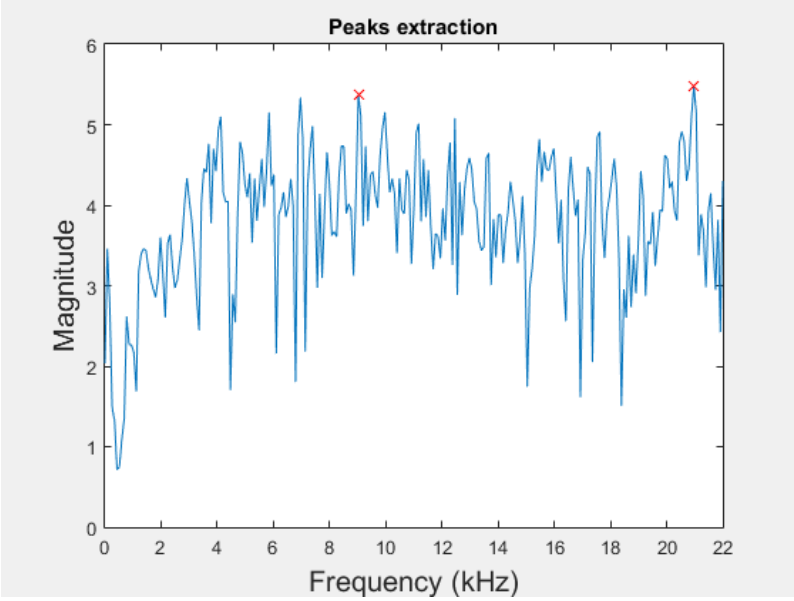

Figure 3.6: Spectrograms of a query song an a reference song with the matching landmarks highlighted.

The offset between the two clips is then the most frequent time difference between the landmarks, similarly of what already explained in subsection 3.1.5. Figure 3.7 shows the spectrograms of both a query song served as input to the system and a database song. Landmarks of both clips are represented as the red connections, with the matching landmarks being highlighted in green.

3.1.7 Possible applications

The use of audio fingerprint algorithms can lead to several benefits in various topics [6]. Identifying broadcast music is probably the most direct one, since it enables song recognition in a song database. The most known example of this is the mobile application *Shazam*, that practically incorporates the fingerprinting algorithm described in [50]. Audio fingerprinting can also be extremely useful on finding duplicates over a large database, promoting a better organisation and filtering over a possibly large set of files.

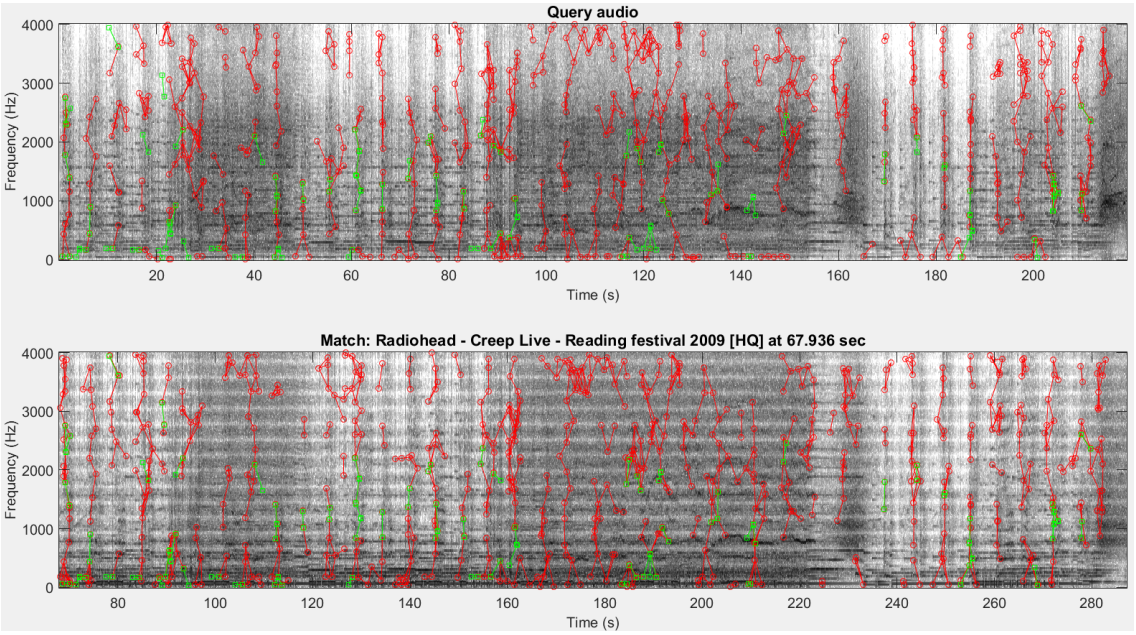

Figure 3.7: Spectrograms of both a query song and a song present in the database. The matching landmarks are highlighted in green.

The rich information obtained from each audio signal might also enable to retrieve musical features over a sound file (*e.g.*, estimated tempo, average spectrum, rhythm and tone inference), which can serve as input to enhance that file's metadata.

As previously mentioned, fingerprinting is also a good candidate for copyrighting detection since it does not implicate adding extra information to the original content prior release. Following the same concept it also enables content monitoring and tracking both in the end-user as in the distribution channel, which contributes greatly to understand users' behaviours. Audio fingerprinting can be additionally used for detecting repetitive sounds inherent of broadcasting music systems, such as jingles and advertisements detection [26, 35].

Another important application, and the one that we focus in this thesis, is the ability of performing clustering of audio data that contains the same audio segments. Concerning the enormous and heterogeneous amount of data experienced nowadays, organising such data is of extremely importance and it is one of the major things we plan to tackle with this work.

3.2 Audio Clustering

The main work presented in this thesis is related to the clustering of information based on audio cues. Thus, some research was made regarding the techniques used to group information retrieved from social media. The grouping of user-generated content lying upon audio features and audio fingerprinting in particular, can be considered a recent topic since it is a direct cause of the increase of the heterogeneous data now experienced

in almost every online platform. Moreover, to come across with this thesis objective, emphasis was given in data originated from musical concerts, which narrowed even more the current state-of-the-art.

3.2.1 Graph structure using fingerprinting

Kennedy and Naamnan proposed an approach to perform automated organisation of concert videos described in [22]. The linking and clustering described is made through connecting overlapping clips, and the cluster is formed in a graph-based structure where nodes are concert clips and edges represent that two clips have a common segment. In order to detect overlapping, the fingerprinting algorithm presented in [16] was used.

3.2.2 Interest inference

Performing this grouping over different recordings of the same concert also enabled to identify possible moments of interests from the audience that happened during the show. This was inferred simply by analysing over-populated clusters under the assumption that the most interesting moments cause a higher number of recordings, being that easily observed in the graph by a big number of clips overlapping in that section of time.

3.2.3 Higher quality audio selection

Audio fingerprinting techniques seem to work less efficiently when used to perform a match between two less-quality and more noisy clips [22]. Thus, fingerprinting algorithms tend to correctly identify matches when at least one of the clips is clean or with much less noise. This means a matching between a clean clip and a very noisy version of that clip is more likely to happen than when only two noisy versions are compared. This specific insight lead to the idea of selecting the nodes with more edges (*i.e.*, the clips that were matched more times against others) as the nodes with more quality in the graph, as stated in [22].

This work served as a basis for the work further developed and described in the section 4 of this thesis with the difference of not performing the grouping of clips of a given concert but from clips originated from different concerts. Moreover, it also served as a starting point to follow another approach that could eventually perform better when inferring audio quality inference presented in section 4.4.

Concluding remarks

In this chapter some state-of-the-art literature was presented and further explained concerning audio fingerprinting and audio clustering. Since all the proposed methods in the next chapter rely solely on the information retrieved by the audio fingerprinting algorithm used (*audfprint*), it is important to understand how such information is generated

internally. Furthermore, the work proposed for audio clustering and quality inference in [22] motivated the work presented in this thesis.

Chapter 4

Proposed solution

In this chapter we present the different methods proposed to achieve the organisation, segmentation, and quality inference of a dataset of audio files all relative to certain events. We also present two different approaches to filter the false positives retrieved by the audio fingerprinting algorithm used as the basis to perform all of the proposed methods bellow.

This chapter is divided in several sections: section 4.1 describes with more detail the nature of the proposed methods already introduced in section 1.3, the following sections describe the specific algorithms used to accomplish this thesis' goals , with the exception of section 4.7 that suggests how the overall output of the different methods can be practically used by the final COGNITUS' application. All the different proposed methods were further presented in two different accepted scientific papers that can be consulted in appendix A.

4.1 Proposal

In this dissertation we propose to explore techniques to organise and determine the quality of user-generated audio content retrieved from social media platforms. These mechanisms shall act as a way to better visualise and deal with extensive datasets of audio files. Moreover, the inference of the quality of each file within a given group of files has the ultimate goal of promoting a better user experience, since only the high-quality clips should be shown to the end-user.

As already briefly stated in Section 1.3, the proposed work can be seen as an integration of several modules or steps: the **crawling** of user-generated content from social media websites (*e.g.*, YouTube [57]), proceeding with the **audio synchronisation** of the files, grouping them in the different events and their respective timeline (*i.e.*, perform **audio clustering** and **audio segmentation**), whilst filtering possible false positives (*i.e.*,

matches filtering), and finally inferring **audio quality** of the different files. Figure 4.1 revisits how these different steps relate with one another. The focus of this chapter will be on explaining the different proposed methods (represented as boxes in the figure).

Important to notice that a dataset with multiple user and professional recordings of different concerts song was manually crawled from YouTube was further used to validate and evaluate the proposed methods, with the respective metrics about the dataset being presented in section 5.1.

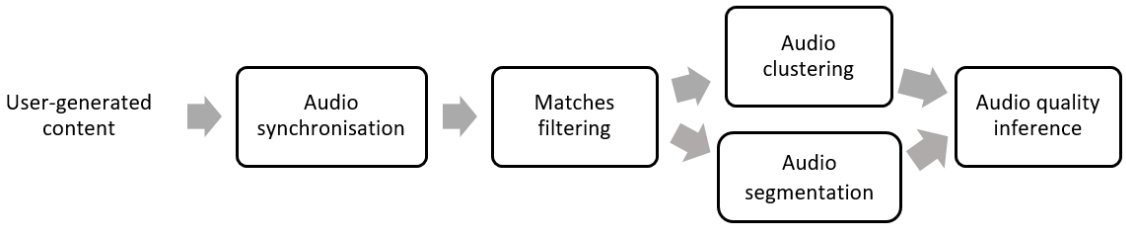

Figure 4.1: Diagram representing the relation between the proposed methods.

The algorithm presented by Haitsma and Kalker for audio synchronisation is widely used and would possibly be a good approach to synchronise our data, but after careful analysis of its extraction method we concluded its computational weight is too high (more detail in section 4.2) [16]. The extraction process consists of creating 32-bit sub-fingerprints every 11.6 milliseconds, which could be problematic in the creation of fingerprints for an extensive database of files. Therefore, we use **audfprint** [14], an implementation of the algorithm described in subsection 3.1.6, to get the time-alignment offset of the different audio files.

The information retrieved by the matches of the audio fingerprinting algorithm when synchronising the different songs consist of the only information needed to perform all of the proposed methods in this thesis. Since the implementation used is landmark-based, the information of how many matching landmarks a given song has, or how many landmarks were matched at a given offset between two songs, represent the basis to perform the filtering of false positive matches, audio clustering, and the segmentation and audio quality inference of the different songs inside each cluster.

The audio clustering phase is made according to whether it exists at least a common audio segment between the different files (*i.e.*, two files will belong to the same cluster if they have a common excerpt, even if they are not time-aligned). Thus, the grouping of files is made with the information directly retrieved from the audio fingerprinting algorithm used for audio synchronisation. Different approaches on how to perform the clustering with the matching information are described in section 4.3, and an approach to filtering the false positives possibly returned from the audio fingerprinting algorithm is presented in section 4.6, for better clustering results (*i.e.*, avoiding clusters from different events to be merged).

The output matches retrieved by the audio fingerprinting algorithm is further used

to derive how the different samples are distributed over time (*i.e.*, Audio Segmentation). This enables segmenting according to the time intervals where there is overlap (*i.e.*, where various samples are relative to the same moment of time of the event). The inference of the audio files' quality can be done either by comparing the quality of the files within a cluster or the different segments where a given audio file belongs. The proposed quality scoring method is presented in section 4.4 and it is similar for both situations.

4.2 Audio synchronisation

The first phase of our proposed solution is to perform the synchronisation of the different files in the dataset. We used the fingerprinting algorithm *audfprint* to perform the audio synchronisation and therefore retrieve information about the time-alignment between audio clips. The algorithm is explained in detail in section 3.1.6. This is a crucial phase since it is from the synchronisation information, more concretely to the output of the audio fingerprinting algorithm, that the rest of the proposed methods will be based on.

For each query sample, the audio fingerprinting algorithm retrieved the different matches that that sample had when comparing its fingerprint to the fingerprints of all samples present in a pre-existing database. For each match, there is information about how many matching landmarks there is between the two samples, and more specifically the number of matched landmarks of the most likely offset (*i.e.*, the offset that has the highest number of matched landmarks). Intuitively, the value of that offset is also retrieved. There is further information about the number of landmarks of both the query and matched samples. This different information obtained by the each match will be further used by each one of the proposed methods in different ways, as it will be discussed in the next sections.

As an example, table 4.1 shows the offset of a set of 4 chosen query songs that got matched with different offsets to a given song in the database (the same for all 4 queries in this case). The offset of each query represents the time displacement between both samples, *i.e.*, the amount of time that one should forward the reference track to achieve total audio synchronisation, given that the query song is played from the start. It is important to note that the chosen offsets for each query were the ones that had the highest number of matching landmarks, with the rest of the offsets not being considered.

4.3 Audio clustering

Given the data used in COGNITUS being heterogeneous and so vast, the clustering of information based on audio seemed a necessary and very useful approach, since it enables the grouping of recordings relative to a given event (*e.g.*, a certain song being played in a concert). Thus, in this system the ideal situation would be that all recordings of a given song would be put in the same cluster. This grouping approach is even more useful

Query songs			
Query Name	Query Length (mm:ss)	# Matching Landmarks	Offset (s)
Radiohead Creep live at Reading festival 2009	01:54	24	171.0720
Radiohead - Creep (whole) - Reading Festival 2009	04:36	54	16.8320
RADIOHEAD - CREEP @ READING FESTIVAL (30 AUGUST 2009)	04:15	13	37.1520
Radiohead Creep, Reading 2009	03:39	119	67.9360

Table 4.1: Query songs' offsets.

when also considering the quality inference of the different recordings of a given event, enabling only the best quality recordings inside a certain cluster to be chosen.

To group the different clips, we used the Audio Fingerprinting algorithm to find out which samples (*i.e.*, audio clips) have the same excerpts (*i.e.*, are matched by the algorithm) and, if so, they should belong to the same cluster. Thus, for this audio clustering phase we only check if there is a match between two different samples, meaning that those samples have at least 5 matching landmarks at a certain offset. This process is made through agglomerative clustering, where we start with singleton clusters (*i.e.*, each sample represents a cluster), and then clusters are merged if two samples belonging to different clusters were matched by the algorithm. Clusters that remained with only one element after this process are considered unmatched since no match was found in the database.

Graph-Based representation

In order to represent the matching between different samples, a graph-based approach was used. Thus, the different samples represent the nodes and an edge represents a matching between two samples, with the edge's weight being the offset (in seconds) between both files. Important to notice that the offset information between two samples is given by the audio fingerprinting algorithm itself.If there is a path between two nodes, then the samples representing those nodes are in the same cluster. Isolated nodes represent unmatched samples, since the algorithm could not find any match in the database for that specific sample. Figure 4.2 illustrates an example of this process.

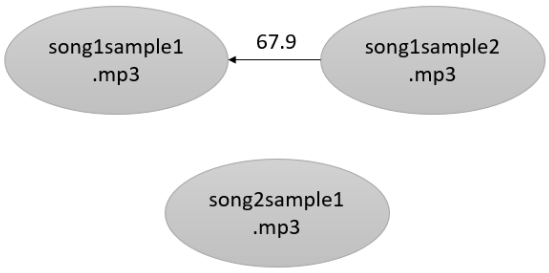

Figure 4.2: Graph representation of a database with 3 different samples, with song1sample1.mp3 and song1sample2.mp3 representing recordings of the same song (with a time offset of 67.9 seconds between them) and song2sample1.mp3 being an unmatched sample by the audio fingerprinting algorithm.

This allowed to easily represent all the matches retrieved from the database when matching a song, and also to easily perform the grouping of the different samples simply by seeing that if two nodes have a path (*i.e.*, are linked by a sequence of edges), they shall belong to the same cluster.

The clustering program receives a list of sound file paths as input, that will be firstly used to form the database created by the Audio Fingerprinting algorithm. Then, each path in the text file is removed from the database, served as input to the fingerprinting algorithm (query sample) to see if it has a match with any of the files in the database, and then added again to the database. The program then outputs the different clusters, represented by incremental id's, with the audio files of corresponded to each cluster. Both the input and output are read/written to text files. An example of both an input and output file is presented in listings 4.1 and 4.2, representing the list of paths of the audio files, and the clusters id's followed by the respective file paths in each cluster, respectively.

```
1 sounds/file1.mp3
2 sounds/file2.mp3
3 sounds/file3.mp3
4 sounds/file4.mp3
5 sounds/file5.mp3
6 sounds/file6.mp3
7 sounds/file7.mp3
```

Listing 4.1: Input file example with the paths of the sound files.

```
1 Cluster 1
2 sounds/file1.mp3
3 sounds/file4.mp3
4 Cluster 2
5 sounds/file3.mp3
6 sounds/file6.mp3
7 sounds/file7.mp3
8 unmatched
```

```
9  sounds/file2.mp3
10 sounds/file5.mp3
```

Listing 4.2: Output file example. Two clusters were formed, and 2 samples were considered unmatched. Cluster are represented by incremental id's (starting at 1), and the different files of each cluster are represented by their file paths.

One simple approach to perform the clustering is to consider only the file that is matched more often (*i.e.*, the sample in the database with the biggest number of matching landmarks to the query sample). Even though this technique would promote the correct grouping of the samples, it would also have some drawbacks since only one of several sample matches are considered. This would generate a large number of small clusters, which would not suffice to complete our goal of grouping all samples of a given concert song. Another approach would be to consider all matching samples of a given query sample. This way, we would use all of the information retrieved by the audio fingerprinting algorithm meaning that more samples would be grouped together, and larger clusters would be formed, resulting in a better clustering result in the end considering our goal.

4.4 Quality inference

As seen in the previous section, the output of the clustering phase consists of a set of clusters, represented by graph G, that organises the data according to events (samples with a common segment of audio belong to the same cluster). The vertices within a component of G may have a different number of neighbours as not all pairs of vertices within a component are adjacent. For instance, let s_1, s_2 and s_3 be vertices in G. If s_1 and s_2 are adjacent, and s_2 and s_3 are adjacent, all three vertices belong to the same component of G (same cluster) but this does not mean that there is an edge between s_1 and s_3: s_1 and s_2 can have a common excerpt, s_2 and s_3 can have another common excerpt, and s_1 and s_3 may lack common excerpts.

Kennedy and Naaman [22] propose to derive the quality of samples by the direct analysis of G, where a sample's quality is proportional to the number of neighbours of that sample's vertex, that is, the number of adjacent vertices. In the example above, s_1 and s_2 are neighbours, s_2 and s_3 are neighbours but s_1 and s_3 are not neighbours. Samples with more neighbours, that is, samples that got matched the most by the fingerprinting algorithm, are considered as those with better quality. This is supported by the idea that any two low-quality samples are less likely to match each other than when at least one of the samples has good quality.

Even though this is a suitable approach, it only considers the number of matches of a given sample, ignoring how the sample ranks in terms of matching landmarks in the overall list of matched samples. A sample with many matching landmarks is likely to have better quality than a sample with less matching landmarks. Thus, our proposed

solution considers that the score of a sample, s_i, depends on the total number of matching landmarks of that given sample against all the other samples in the database. Let $s_1, s_2, \ldots, s_N$ be the samples in the database. Assume s_i (for $1 \leqslant i \leqslant N$) has l_{1i} matching landmarks to sample s_1, l_{2i} matching landmarks to sample s_2, etc. Then the score of s_i is calculated as follows:

$$score(s_i) = \sum_{j=1}^{N} l_{ji}\,. \tag{4.1}$$

In chapter 5 we compare both of these approaches using our manually crawled recordings dataset. It is visible that our novel approach outperforms the latter, given the assumption that professional recorded samples should have higher quality than user-recorded samples in each cluster. Thus, our method always classifies the professional recordings equally, or in some cases even better than the method proposed in [22]. More details of this discussion will be given further ahead.

4.5 Audio segmentation

Analysing how the different samples of each event are scattered over each event's timeline is of extreme importance to better manage the different audio files. Therefore this section focus on finding in which time intervals (*i.e.*, segments) the different samples are distributed inside each cluster regarding their content. An important aspect of this synchronisation task is that it only requires information already obtained from the audio fingerprinting algorithm that was used to perform the clustering described in section 4.3.

The synchronisation in terms of offset returned by the audio fingerprinting algorithm (introduced in section 4.2) was further used to perform the alignment of the audio samples. Considering the graph-based representation of our samples described in section 4.3, by including the offset between two samples as the weight of the edge of their respective nodes, we can simply derive the offset between any two samples in the same cluster by adding the weights of all edges that are in their path (*i.e.*, by calculating the cost of the path). Important to notice that this only works because if there is a positive edge in the graph connecting two nodes, a negative edge on the opposite direction was further added as well. This two-way edges are not shown in figure 4.2 but it is how the graph was built in practice. We can then represent the offset o between two nodes i and j of any event's graph G as $o_{ij} = cost(G, i, j)$.

The actual way the synchronisation of all samples inside a cluster is made is by electing a **representative sample** and by getting the offset of all the other samples relative to this one. Note that the representative sample can be any of the cluster samples, since all samples of a given event are connected in the graph. After all offsets are obtained, if the representative sample is not the recording that has the earliest starting timestamp, the offset values are updated according to the sample with the earliest timestamp (*i.e.*, sample that starts first in the event's timeline).

We can then define the earliest starting sample e as the sample i that satisfies:

$$o_{er} = min(o_{ir}),$$

with r being the representative sample and $i \in V$, with $G = (V, E)$, being V the set of vertices and E the set of edges. This minimum value can either be 0, if the representative sample is indeed the earliest starting sample (since $o_{rr} = cost(G, r, r) = 0$), or a negative number, that would mean there is at least a sample i that starts before sample r. We finally proceed in updating all offsets o_{ir} to o_{ie}, by adding the value of o_{er} to the offsets of all samples i in the graph G:

$$for\ each\ i \in G:\ o_{ir} := o_{ir} + o_{er}$$

Using this approach, all offsets are greater or equal than 0 and correctly aligned in terms of their starting point along the event's timeline, since all offsets are now relative to the earliest starting sample.

4.5.1 Time-based segmentation

By having the overall offsets of all samples of a given cluster, together with having the respective duration of each one of those samples, one has the knowledge of which samples exist in a given moment of time. We encapsulate this information in a segment, that contains an initial and final timestamp, and all samples that overlap between that period of time. Each cluster or event is then composed by several segments, that give information on which samples are available in the different time intervals and therefore at any moment of time in the event's timeline.

Considering the offsets of all samples of a given event relative to the earliest starting sample e ($\{o_{ie} : \forall i \in V\}$) and their respective duration ($\{d(i) : \forall i \in V\}$), a new segment starting at t_s is created when one of the following situations occur:

- At the moment of time t_s in the overall event's timeline, there is a sample i with $o_{ie} = t_s$ (*i.e.*, a new sample starts).
- When there is a sample i with $o_{ie} + d(i) = t_s$ (*i.e.*, a sample ends).

As a consequence, whenever a new segment starts at t_s, there is a segment ending at $t_s - 1$, except when $t_s = 0$ meaning that it is the first segment of that event.

More than simply synchronising all samples in a cluster, this step provides the knowledge of which samples exist in a given moment of time. This is achieved by calculating the time segments where samples overlap, taking into consideration their starting timestamp relative to the first encountered sample, and their duration. All samples are then cut to generate various audio files, one for each segment that the given segment is part of. This is really important regarding the COGNITUS' perspective, since it allows the system to know which samples can be shown to the final user in a given moment of time. However

this process would be even more enriching if such samples were ordered by quality inside each segment; this way only the highest quality sample would be played in each segment.

4.5.2 Segmented quality inference

To perform such quality inference, we propose to use the same process as presented in section 4.4. However, given the possible small duration of the time segments together with the small number of segment's cut samples, make matches in such cases unlikely to happen. Thus, there is the need of increasing the number of *hashes/sec* performed by the Audio Fingerprinting algorithm. This increase can generate a higher number of matching landmarks between samples (as shown in figure 4.3 and 4.4) and in some cases retrieve matches on segments that previously had no matches (as observed in figure 4.5).

The lower number of matches also led to update the minimum number of landmarks matches to 1 to be considered an actual match by the algorithm, instead of 5 (as previously mentioned). This does not consist of a problem since the prior run of the algorithm and the consequence filtering process (explained in the next section) already assured the correctness of the matches by the filtering and removal of the false positive matches. Thus, since all samples were cut accordingly to the synchronisation results (offsets), only the matches with an offset of 0 seconds are considered and all the rest discarded. Important to notice that for actually inferring the quality of each file in the segment one has to call the audio fingerprinting algorithm again using each of the segment's files as a query sample against the rest of the files in the segment.

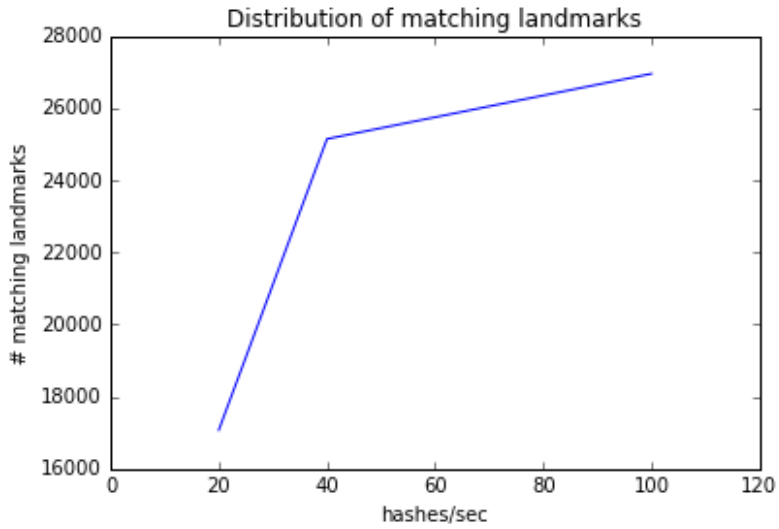

Figure 4.3: The number of matching landmarks obtained increases with the *hashes/sec* used in the Audio Fingerprinting algorithm.

The output of the audio segmentation process is shown in listing 4.3, where the different segments found for cluster 1 are presented, together with the list of all audio samples in each segment ordered descending by quality. It is observable that in some

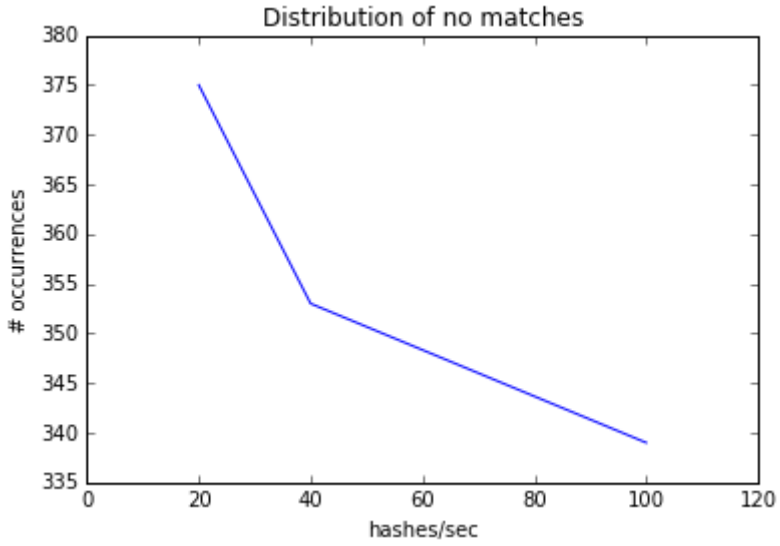

Figure 4.4: As we increase the number of *hashes/sec*, the number of samples with no matches returned by the Audio Fingerprinting algorithm decreases.

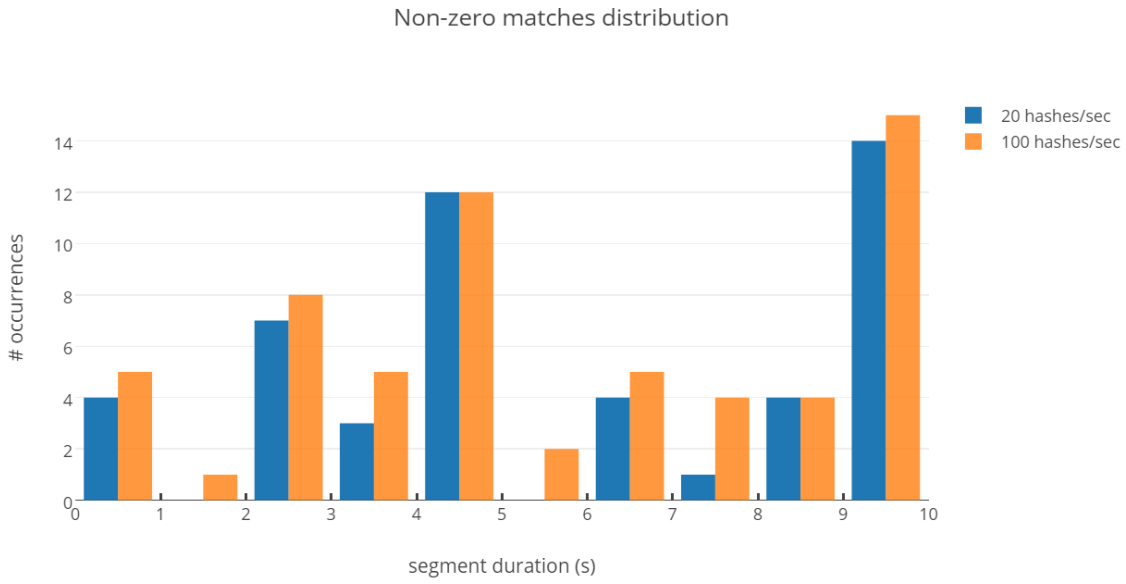

Figure 4.5: Distribution of the number of non-zero matches according to the segment duration. With 20 *hashes/sec* all segments between 1-2 seconds and 5-6 seconds had no match, but some matches occurred when using 100 *hashes/sec*.

segments the quality could not be inferred, more concretely in the earlier and latest segments. This can be explained by the already mentioned small length of the segments and the small number of samples they contain. One possible way of handling these cases is to consider the overall quality inference of the tied samples, and proceed to choose the ones with higher quality score.

```
1 Cluster 1
2 00:00:0-00:04:3
```

```
sounds_cut/6/0.0/song1sample1.mp3 0
00:04:3-00:11:3
sounds_cut/6/4.3/song1sample1.mp3 0
sounds_cut/6/4.3/song1sample4.mp3 0
00:11:3-00:18:3
sounds_cut/6/11.3/song1sample5.mp3 7.0
sounds_cut/6/11.3/song1sample4.mp3 6.0
sounds_cut/6/11.3/song1sample1.mp3 0
00:18:3-00:48:8
sounds_cut/6/18.3/song1sample5.mp3 83.0
sounds_cut/6/18.3/song1sample3.mp3 66.0
sounds_cut/6/18.3/song1sample4.mp3 44.0
sounds_cut/6/18.3/song1sample1.mp3 19.0
00:48:8-01:40:769
sounds_cut/6/48.8/song1sample3.mp3 108.0
sounds_cut/6/48.8/song1sample5.mp3 85.0
sounds_cut/6/48.8/song1sample4.mp3 82.0
sounds_cut/6/48.8/song1sample2.mp3 55.0
sounds_cut/6/48.8/song1sample1.mp3 42.0
01:40:769-04:40:832
sounds_cut/6/100.769/song1sample5.mp3 250.0
sounds_cut/6/100.769/song1sample4.mp3 165.0
sounds_cut/6/100.769/song1sample2.mp3 161.0
sounds_cut/6/100.769/song1sample1.mp3 54.0
04:40:832-04:40:924
sounds_cut/6/280.832/song1sample2.mp3 0
sounds_cut/6/280.832/song1sample5.mp3 0
sounds_cut/6/280.832/song1sample1.mp3 0
04:40:924-04:45:283
sounds_cut/6/280.924/song1sample1.mp3 0
sounds_cut/6/280.924/song1sample5.mp3 0
04:45:283-04:47:362
sounds_cut/6/285.283/song1sample5.mp3 0
```

Listing 4.3: Example of output file with the different segments of cluster 1.

4.6 Matches filtering

Considering all the matches retrieved from the audio fingerprinting algorithm would be ideal if false positives did not occur. These false positives can happen in **landmark-level** (*i.e.*, when several landmarks are matched with different offsets over just two samples), or in **sample-level** (*i.e.*, samples not referring to recordings of the same song were matched). In the first case, it is important to notice that only one of the retrieve offsets is correct, since two clips can only have one offset. Thus, all the other offsets can be considered false positives.

The first problem, where different landmarks are matched in more than one offset, can be easily observed by the repetition of a sample in another's matching list (*i.e.*, the

song5sample2.mp3's matching list			
Sample name	Offset (s)	# Matching Landmarks	# Total Landmarks
song5sample5.mp3	4.7	167	1202
song5sample3.mp3	21.2	88	429
song5sample6.mp3	-8.8	50	132
song5sample1.mp3	22.2	41	188
song5sample4.mp3	71.1	17	557
song5sample5.mp3	2.0	12	1202
song5sample5.mp3	15.6	7	1202

Table 4.2: Example of sample repetition in the matching list (song5sample5.mp3 appears 3 times).

list of samples that share some of their landmarks with the query sample). Such situation can be observed in table 4.2, where the output information of the number of matching samples and their offset of a given sample of the database was put in table format for easier comprehension. In this example, song5sample5.mp3 appears three times in the list because it was matched to song5sample2 with three different offsets: 4.7 s, 2.0 s and 15.6 s. By direct analysis of the number of matching landmarks of the repeated sample (167, 12, and 7, respectively), we can conclude that only the first match should be considered as the right match due to having the highest number of matched landmarks, whilst the rest should be discarded. Thus, to tackle this problem, any repetitions included in the matching list of a given sample are eliminated and only the match with higher number of matching landmarks is considered.

In order to handle the situation where samples from different events are matched together (*i.e.*, sample-level), it is important to observe in which context they appear. Even though unlikely, the probability of their appearance is greater than zero and this was observed in some of the tests made being one of them illustrated in table 4.3. In this work we present two different approaches to tackle the filtering of the false positives matches: a **derivatives approach** (section 4.6.1) that consist of the analysis of significant drops in the percentage of matching landmarks of the matching list's samples and a **learning approach** (subsection 4.6.2) that uses classification techniques to assign a match as a true or false positive.

4.6.1 Derivatives approach

From the analysis of figure 4.3, it is clear that song2sample5.mp3 belongs to a different song than song8sample7.mp3 and therefore represents a false positive. The analysis of this and more situations, showed that false positives of samples that are not meant to match do not happen often and when they do it is with a low number of matching landmarks. As can be observed in table 4.3, the number of matching landmarks for false positive sample is 7. While this number does not seem much lower than the number of

song8sample7.mp3's matching list				
Sample name	Offset (s)	# Matching Landmarks	# Total Landmarks	% Matching Landmarks
song8sample2.mp3	-309.8	77	1283	6.0
song8sample3.mp3	-116.0	71	1433	4.95
song8sample8.mp3	-72.3	55	3035	1.81
song8sample6.mp3	-328.4	50	1305	3.83
song8sample1.mp3	-292.6	42	1886	2.23
song8sample5.mp3	-326.7	30	1934	1.55
song8sample4.mp3	-421.5	12	308	3.90
song2sample5.mp3	-398.1	7	1392	0.5

Table 4.3: Example of a false positive sample (song2sample5.mp3) in the matching list. The number of matching landmarks of that sample (7) and percentage (0.5 %) are the lowest in all matches.

matching landmarks for some of the other samples (like 12 for sample song8sample4), we must also take into account the number of total landmarks. Thus, by observing the column with the percentage of matching landmarks, we observe a slightly bigger decrease when analysing the difference between the matched landmarks in a false and a true positive.

The ideal situation would be to discard all the wrong matches in the matching list and to utilise all the correct matches. An approach that would possibly achieve this is by stop considering matches as soon as the percentage matching landmarks significantly drops. In order to achieve this, one could analyse the derivative on all consecutive pairs of points in the graphic (considering the percentage of matching landmarks of each file/point) and stop when the derivative is lower than a certain value. This approach could possibly be done by performing the following steps:

1. Sort the percentage of landmarks of each all samples in the matching list (in descending order);

2. Calculating the derivative between all consecutive pair of points;

3. Find the first point where derivative is lower than a certain value (-0.07 by default);

4. Discard all samples positioned after that point.

This would be a reasonable approach to follow if the difference between the number of landmarks in correct matches would never decrease significantly, similarly to what is observed in the wrong matches. If that happens, the proposed algorithm will discard a high number of correct matches. Thus, this approach has to take into consideration more parameters to better choose when to filter matches.

One strong candidate is to only filter matches when they are bellow the average of the percentage of matching landmarks in the matching sample's list. By adding this simple parameter, the number of correct matches discarded will decrease, whilst hopefully maintaining the right discarding of the wrong matches since they will almost certainly always have a percentage of matching landmarks lower than the average. Thus, the step 4 of the previously filtering process would be changed to:

4. Find the first point where derivative is lower than a certain value (-0.07 by default) and the percentage of matching landmarks is lower than the average percentage of matching landmarks in the list.

For an easier analysis of the retrieved results presented in table 4.3, the information is displayed in a graphic that shows the distribution of the percentage of matching landmarks over the total number of landmarks of each sample (figure 4.6). The ideal situation would be to only exclude the last file since it is the only one wrongly matched, and, for this situation, following this refined approach correctly achieves that.

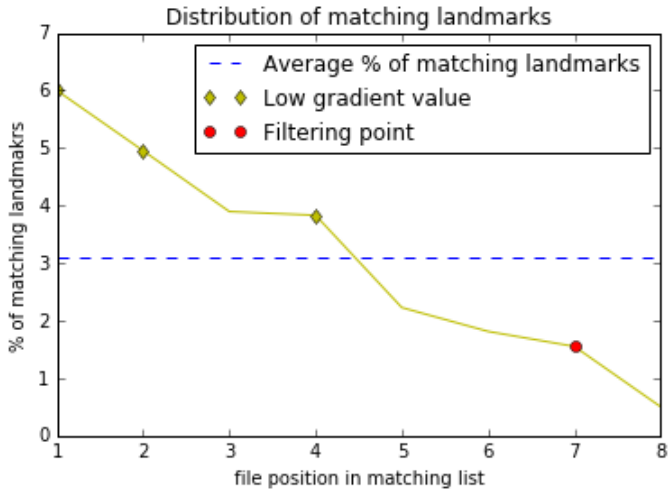

Figure 4.6: Distribution of matching landmarks between the matching samples. The breaking point indicates the last sample to be accepted as a true match, being all the files with lower percentage of matching landmarks than that sample discarded (in this case the 8th sample).

Important to notice that this filtering process will increase the number of false negatives (by filter some true positives as well) but would certainly decrease the number of false positives, which is extremely important to prevent clusters without matching segments to merge. Furthermore, the derivative threshold used (-0.07) was obtained by the analysis of the derivatives of the false matches present in the dataset used to validate and test our methods further described in chapter 5, with its value being fine-tuned accordingly in order to filter all of them. This discussion is further presented in our paper in [31].

4.6.2 Learning approach

Even though the approach presented above achieves good results (as will be described further in chapter 5), it might not be good enough on generalising to different datasets. Therefore, an approach that considers machine learning may be more advantageous and more suitable once the dataset is extended. The main goal using this learning filtering approach is then to correctly predict if a never seen match (*i.e.*, a match not present in the training set) is a false positive or a true positive.

We used three different methods to solve this classification problem: Logistic Regression, k-Nearest Neighbours (KNN), and Support Vector Machines (SVM). The purpose of using different classifiers is to have a broader way of comparison on how the different features used influence the outcome of the overall predictions of the different methods. All the used classifiers were already introduced and overviewed in chapter 2.

We used double cross-validation to retrieve the model with lowest validation error for each classifier's parameters: we start by performing **leave-one-song-out** cross-validation, in which all songs in the training set except one are used to train the model with k-fold cross-validation, whilst the left-out song is used to test the model; this process is then repeated until all songs have been left-out and repeated in every combination of possible parameters assigned for each classifier. Note that leaving one song out of the training set means leaving all samples from that song out of this set. The training and validation error of each model is the average of the error occurred in all the leave-one-song-out iterations, with the accuracy of the model being tested on the overall predictions of all left-out songs' samples.

Following these steps for all designated ranges of possible values for the different classifiers' parameters (further described in sub-subsection 4.6.2.3), we assign the model with lowest validation error in the k-fold validation for each classifier as the most suitable model, with its accuracy being the accuracy of its prediction when performing the leave-one-song-out cross-validation. This discussion is further presented in our second paper [32].

4.6.2.1 Training data

Since the goal of our models is to predict whether a sample is a false match or a true match, there is only two classes possible - 0 and 1, respectively. Matches are considered false positives if the two matched songs do not have any common audio segment (retrieved as no match by the audio fingerprinting algorithm), or if they have indeed a common part but the assigned offset is not correct. The latter case can be easily detected by some songs appearing in the matching list of a query several times with different offsets, described as repetitions. Therefore, only the matched song's sample with highest number of matching landmarks of a given offset is considered a true match (*i.e.*, assigned to class 1), whilst all the other samples relative to that song are considered false matches (*i.e.*, assigned to class 0).

4.6.2.2 Feature selection

Choosing the right features for our samples is essential to construct any good model, since a bad choice of features can influence in whether our classes are linearly separable or intensively overlapped, increase training times, and overly complicating the problem. Using the most relevant and essential properties, and excluding the rest, helps the model to achieve better accuracy in the end, even with less training data [5]. Using only the necessary features also avoids the *Curse of Dimensionality*, that it is a consequence of using too many features and therefore increasing the feature space dimensions, resulting in the data becoming more sparse than needed in the feature space, increasing the risk of wrong classifications.

In the context of our problem, a (training/test) sample contains information about the similarity between two audio clips. The feature vector representing a training/test sample is derived from the output matches retrieved by the Audio Fingerprinting algorithm. If we ask the algorithm to match a query song against the other songs in the database, the algorithm returns: the number of landmarks of the query song (#QSL), the offset (in seconds) between the two songs (O), the number of matching landmarks with that offset (#ML), and the number of total matching landmarks in all offsets (#TML). Since the offset does not directly influence a false or positive match, it is not considered to enter the feature space, however, all the other features might be a good indicator of a false and true positive. Important to notice that when songs are added in the database, the number of matching landmarks computed for that song is also retrieved from the algorithm, hence the number of landmarks of all songs are known, including the number of landmarks of the matched song (#MSL).

The set of available and possibly relevant features is then the following: F = {#ML, DML, #TML, #QSL, #MSL}. We tested our models with several sub-sets of F, more specifically {#ML, #TML}, {#ML, #TML, #QSL}, {#ML, #QSL, #MSL}, {#ML, #TML, #QSL, #MSL}. Each classifier was trained with all these features sub-sets to assign which combination generated the best model.

4.6.2.3 Parameters setting

For Logistic Regression, we doubled the value of the regularisation parameter c during 20 iterations (with its initial value being set to 1.0), whereas for k-Nearest Neighbours, the number of neighbours k ranged between all odd numbers between 1 and 39. For each iteration of both classifiers, we used leave-one-song-out and k-fold cross-validation, which led to the following error graphics 4.7 and 4.8, that represent the validation and training error of the different range of parameters regarding a randomly chosen iteration of Logistic Regression and KNN, respectively, when using the 2-feature combination (*i.e.*, the number of matching landmarks with the highest number of matched landmarks offset, and the number of total matching landmarks in all offsets) with the dataset further described in section 5.1 as an example.

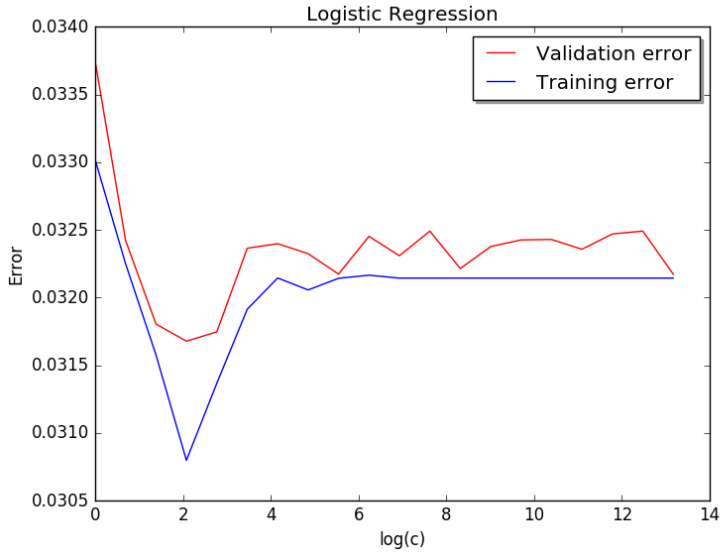

Figure 4.7: The variation of the regularisation parameter c is represented in a logarithmic scale in the x-axis to promote an easier visualisation. From the different c values, $c = 8$ (with $log(c) = 2.079$) was the one who achieved the lowest validation error (≈ 0.0316), representing the best parameter value for our model when using ({#ML, #TML}) as features.

Regarding SVM, we used the RBF kernel and the optimal values for c and γ were obtained by executing an exhaustive search over all possible combinations of a subset of possible values for each parameter following the methodology of using exponentially growing sequences suggested in [19]. More specifically varying c to the following values 2^{-5}, 2^{-3}, ... , 2^{15}, 2^{17} and γ to 2^{-15}, 2^{-13}, ... , 2^{3}, 2^{5}. This searching process is often described as Grid-search, and it returns the best value of each parameter of a given model (*i.e.*, the hypothesis that achieves the highest accuracy).

Grid-search was performed for each possible training set when performing leave-one-song-out cross-validation, that corresponds to the number of different songs in the dataset. Since repetitions of the same combinations of parameter values are eliminated, a set of possibly good pairs of values concerning our dataset (with length between 1 and the number of songs), were used to train the different SVMs.

Continuing with the plotting of our different models' error when using the 2-feature combination {#ML, #TML}, the set of best parameter values for SVM retrieved after the grid-search process were the following: $\{(c = 0.125, \gamma = 32), (c = 32, \gamma = 0.5), (c = 0.5, \gamma = 2), (c = 8, \gamma = 0.125), (c = 2048, \gamma = 0.0078125), (c = 32, \gamma = 2), (c = 2, \gamma = 32), (c = 8192, \gamma = 2), (c = 0.125, \gamma = 8), (c = 0.5, \gamma = 8)\}$, being this a sub-set of all possible combinations of all parameter values presented above. Figure 4.9 plots both training and validation errors of each pair of values of this set. Note that the plot of the errors concerning the all classifiers and feature combinations are further presented in chapter 5.

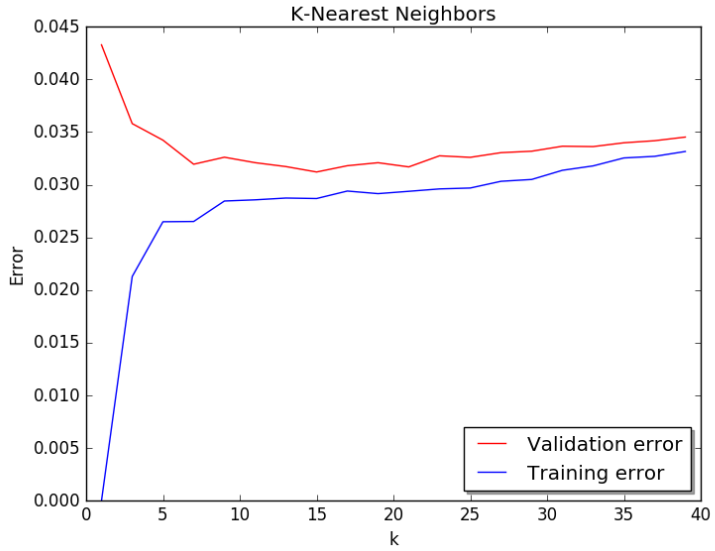

Figure 4.8: As the number of neighbours increases after a certain threshold, we also notice a slightly increase on both validation and training error, meaning that the model is less restrained by local conditions but is more susceptible to producing wrong predictions. The k value that achieved the lowest validation error (≈ 0.0315) is when $k = 15$, being this the number of neighbours considered when dealing with the 2-feature combination.

4.6.2.4 Extended learning

The training set can be further expanded by the analysis of the information retrieved from the audio fingerprint algorithm combined with our model predictions. Such extension would augment the training data of our models and could ultimately lead to better predictions. This extension can occur in two stages: during the audio clustering phase (subsection 4.3), and by the analysis of the matches between the cut samples when performing the audio quality inference inside each segment (subsection 4.5.2).

Concerning the first presented stage, one can consider all repetitions of a given matched song in another's matching list as false positives and update the training set by retrieving the features of the matches repetitions and assign it to class 0. This is supported by the assumption that since only one offset is possible between two songs, the correct offset is the one that generated more matching landmarks, whilst the others are discarded.

The latter stage can serve as a confirmation for some of the samples that were predicted true positive matches as result of the filtering method. Since all false positive matches are filtered in the audio clustering phase (section 4.3), either by the discarding of the repetitions or by false matches classifications, all the samples of the songs present in the audio segmentation phase (section 4.5) were therefore predicted as positive matches by our model. Hence, after cutting each song according to the different segments they appear, and by matching all cut songs with one another inside a segment to infer the quality, all matches should be assigned to offset 0.0 seconds since all the cut songs are

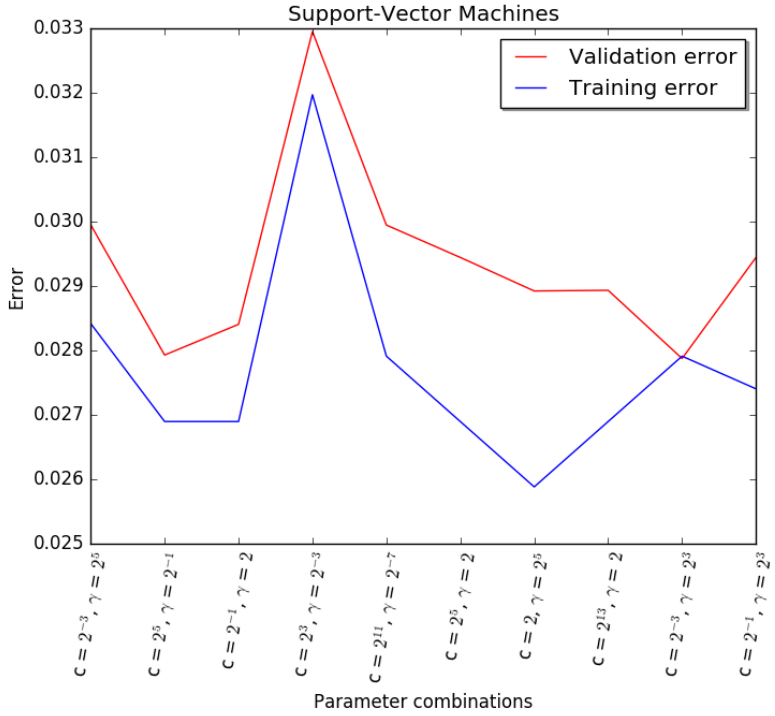

Figure 4.9: Each parameter combination presented in the x-axis was returned at least once as the best parameters from performing grid-search using leave-one-song-out cross-validation. All retrieved pairs are then compared in terms of their model's validation error, with the combination ($c = 0.125, \gamma = 8$) presenting the lowest validation error (≈ 0.0279).

meant to be synchronised in time.

Let us define the function $offset(M, s_1, S)$ as returning the list of offsets values of the different matches between s_1 and each song in list S in the overall list of matches M, function $count(L, v)$ retrieves the number of occurrences of the actual number v in list L, $len(L)$ retrieves the length of list L, and $TP(m)$ assigns the features of sample match m to class 1 in the training set. Moreover, representing all filtered matches (*i.e.*, samples) of all songs of a given cluster c as M_c, S_c as the set of segments of cluster c, S_s as the set of songs in segment s, and M_s as the set of matches of segment s, we can define the following expression for a given cluster c:

$$(\forall s \in S_c \ \forall s_1 \in S_s \ count(offset(M_s, s_1, S_s \setminus \{s_1\}), 0.0) = len(S_s)) \Rightarrow \forall m \in M_c : TP(m)$$

To sum up, one can then assume that, if for each cut song inside each cluster's segments there is a match with offset of 0.0 seconds against all the other cut songs, then all samples that previously contributed in the formation of that given cluster are considered true positives.

4.7 Component integration

Considering the several and variety of the components of COGNITUS, some work has to be followed to surpass integration issues that may arise when dealing with the joining of the different project components. For instance, we had to change the programming language of the clustering algorithm from MATLAB to Python, as well as the fingerprinting algorithm's implementation (from MATLAB's version [13] to Python's version [14]) for an easier server integration.

Several libraries were used in the programming of the different methods, namely *mutagen.mp3* [33], for the retrieval of MPEG metadata from the audio files, *pydub's Audio Segment* [21], to cut the different audio files accordingly to the segments they are part of inside the cluster, *NumPy* [36] for array and matrix operations, *NetworkX* [34] for plotting the overall samples graph derived from the audio clustering phase, and *Plotly* [39] for the making of some of the result graphics presented in this thesis. From all the libraries, the latter two can be dispensed since they are only used for informational purposes and do not compromise the overall execution of the implemented methods. These libraries were chosen not only based on their utility but also on their licensing properties to avoid further copyright problems down the line.

Considering the overall COGNITUS goal of converging user-generated content in a UHD (Ultra-High Definition) broadcast, the methods proposed in this thesis can be seen as facilitators for this end goal (revisit figure 1.3 for a reminder of the different project's steps). Assuming a already existing database of user-generated recordings, our algorithm has the ability of clustering such files based on a given event by the analysis of overlapped audio sections, whilst additionally giving information and synchronising the different files inside a given event in time.

The quality control step can be seen as a simple analysis of the quality score of each file made by the proposed Audio Quality Inference method, where each audio file's quality score being relative to the other audio files inside its cluster or segments. Ideally, all low quality samples would be filtered and only the high-quality files would be played for the end-user of the final application. Furthermore, since our method also infer the relative quality of the audio files inside each segment, one can then only play the top-ranked audio file inside each cluster. However, some segments are of very short duration and it might not be efficient to perform this approach on every single segment, being only suitable switching the currently played song if the new segment duration is greater than a given time threshold or if the quality score of the audio files differ by a given amount. Nevertheless, our methods provide all the detailed information that will aid the final application in choosing the right audio files to be played at any given time.

Concluding remarks

The overall proposal of this thesis is revisited and various methods to achieve the correct organisation and quality inference of user-generated content are proposed in this chapter. Moreover, an overview of how such methods could be practically integrated in the overall COGNITUS architecture is also presented. The validation and evaluation of the proposed methods are further described in the next chapter.

Chapter 5

Evaluation and results

This chapter represents the validation and evaluation of the different methods proposed in the previous chapter. The results of our solution were obtained by using recordings manually crawled from YouTube and are further presented and discussed.

5.1 Test setup

In order to test the proposed methods, a realistic testbed was designed by manually collecting a dataset made of several concert clips of various songs from YouTube, consisting of one professional recording of a certain song in a specific concert and several user recordings of the same song captured with different devices. This dataset was made keeping in mind the project's goals, so it could possibly be used in the different project components, therefore each song has a sample of a professional recording from BBC.

Originally, the dataset consisted of 90 recordings of 10 concert songs all retrieved from different acts and editions of the Reading Festival. Apart from the 10 professional recorded samples, all samples were recorded by users, which means different recording devices with different qualities apply. Table 5.1 shows the diversity when it comes to number of clips retrieved from each concert song and average length of the subset of clips. The distribution of all clip lengths in the original dataset is shown in figure 5.1. Efforts were made to have recordings with lengths across a larger scope in order to promote a more diverse dataset but most of the recordings crawled were within the 5 minutes range.

Although this dataset sufficed to test and validate the previously proposed methods - Audio Clustering (section 4.3), Audio Segmentation (section 4.5), Audio Quality Inference (section 4.4), and Matches Filtering using derivatives (section 4.6) - with the further results discussed in the sections bellow, an extension of the crawled songs could be used to better train our models when using the machine learning approach when performing

Original Dataset			
Band/Song Name	Festival	# clips	Average clip length (mm:ss)
Arctic Monkeys - Do I Wanna Know	Reading 2014	8	3:04
blink-182 - All The Small Things	Reading 2010	8	2:31
Foo Fighters - Times Like These	Reading 2012	8	5:50
Green Day - Boulevard of Broken Dreams	Reading 2013	11	2:41
Metallica - Enter Sandman	Reading 2015	11	3:55
My Chemical Romance - Na Na Na	Reading 2011	6	2:54
Panic! at the Disco - Bohemian Rhapsody (cover)	Reading 2015	11	3:58
Queens of the Stone Age - No One Knows	Reading 2014	5	3:50
Radiohead - Creep	Reading 2009	15	2:40
Red Hot Chili Peppers - Under the Bridge	Reading 2016	7	3:49

Table 5.1: Extended dataset information. A different number of samples and recording lengths was crawled for each song.

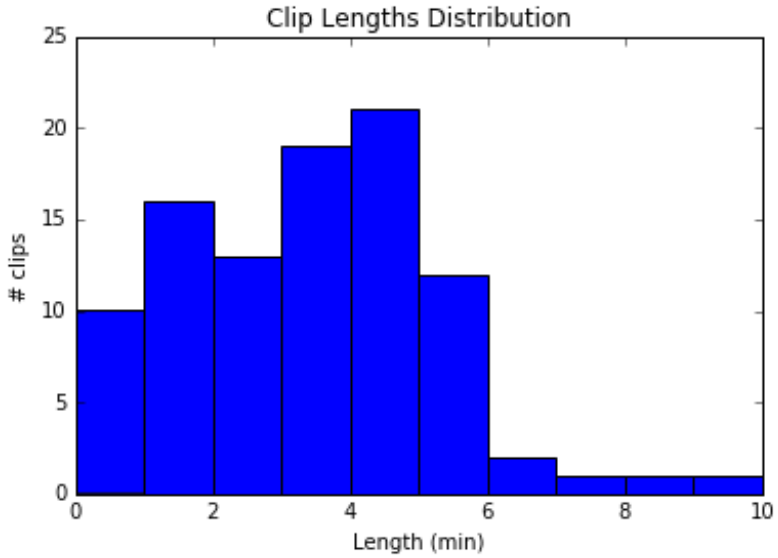

Figure 5.1: Distribution of the various clip lengths of our dataset.

the matches filtering.

Therefore, 13 more songs were crawled, which led to 107 new recordings being added to the original dataset, with their metadata information being presented in table 5.2 and the overall length distribution (*i.e.*, concerning the total of the 23 songs now presented in this extended dataset) being illustrated in figure 5.2. Important to notice that this extended dataset was only used to train and validate our models in the matches filtering using machine learning approach, since it enabled our system to train with more data and therefore increasing its overall prediction accuracy.

Extended Dataset			
Band/Song Name	Festival	# clips	Average clip length (mm:ss)
Alicia Keys - Empire State of Mind	Big Weekend 2010	7	3:54
Bruno Mars - Just The Way You Are	Big Weekend 2013	6	2:42
Coldplay - Yellow	Big Weekend 2014	10	3:51
Coldplay - Clocks	Glastonbury 2011	11	4:01
Coldplay - The Scientist	Glastonbury 2011	12	4:26
Coldplay - Viva La Vida	Glastonbury 2011	13	3:14
Ed Sheeran - Sing	Glastonbury 2014	6	3:39
Franz Ferdinand - Take Me Out	Glastonbury 2009	7	2:24
Hozier - Take Me To Church	Glastonbury 2015	6	1:16
Lady Gaga - Born This Way	Big Weekend 2011	6	5:39
Of Monsters & Men - Little Talks	Glastonbury 2013	6	2:49
Rihanna - Love The Way You Lie	Big Weekend 2012	8	3:43
Vampire Weekend - A-Punk	Glastonbury 2010	9	1:40

Table 5.2: Extended dataset information.

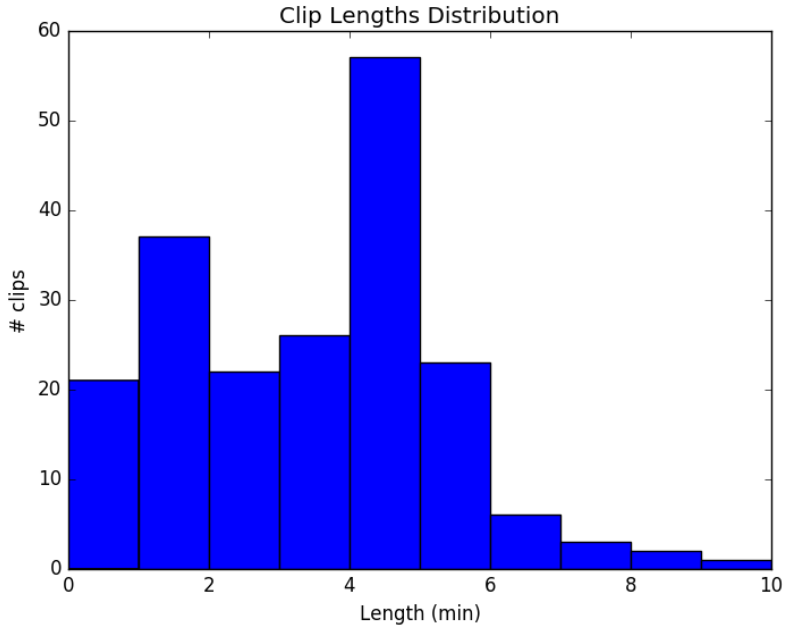

Figure 5.2: Distribution of the various clip lengths on the extended dataset.

5.2 Audio clustering

The original dataset with 10 songs was used to evaluate the method proposed to perform the grouping (clustering) of the different audio files described in section 4.3. Since our goal is to cluster the different samples in the different events (*i.e.*, songs), the number of

clusters retrieved should equal to the number of existing songs (10). Listing 5.1 represents the output of our algorithm, without performing any filtering of the false positives matches retrieved by the audio fingerprinting algorithm.

```
Cluster 1
sounds_my_dataset/song6sample5.mp3
sounds_my_dataset/song10sample1.mp3
sounds_my_dataset/song10sample9.mp3
sounds_my_dataset/song6sample1.mp3
sounds_my_dataset/song6sample10.mp3
sounds_my_dataset/song6sample7.mp3
sounds_my_dataset/song6sample6.mp3
sounds_my_dataset/song10sample3.mp3
sounds_my_dataset/song10sample10.mp3
sounds_my_dataset/song10sample6.mp3
sounds_my_dataset/song8sample2.mp3
sounds_my_dataset/song10sample8.mp3
sounds_my_dataset/song2sample3.mp3
sounds_my_dataset/song8sample6.mp3
sounds_my_dataset/song8sample3.mp3
sounds_my_dataset/song8sample1.mp3
sounds_my_dataset/song8sample8.mp3
sounds_my_dataset/song6sample11.mp3
sounds_my_dataset/song8sample7.mp3
sounds_my_dataset/song8sample5.mp3
sounds_my_dataset/song6sample4.mp3
sounds_my_dataset/song2sample8.mp3
sounds_my_dataset/song10sample11.mp3
sounds_my_dataset/song2sample5.mp3
sounds_my_dataset/song2sample7.mp3
sounds_my_dataset/song2sample4.mp3
sounds_my_dataset/song10sample2.mp3
sounds_my_dataset/song2sample1.mp3
sounds_my_dataset/song2sample2.mp3
sounds_my_dataset/song6sample8.mp3
sounds_my_dataset/song10sample7.mp3
sounds_my_dataset/song6sample2.mp3
sounds_my_dataset/song8sample4.mp3
sounds_my_dataset/song10sample5.mp3
sounds_my_dataset/song10sample4.mp3
sounds_my_dataset/song2sample6.mp
sounds_my_dataset/song6sample3.mp3
sounds_my_dataset/song6sample9.mp3
Cluster 2
sounds_my_dataset/song7sample3.mp3
sounds_my_dataset/song7sample4.mp3
sounds_my_dataset/song7sample5.mp3
sounds_my_dataset/song7sample1.mp3
sounds_my_dataset/song7sample2.mp3
sounds_my_dataset/song9sample5.mp3
```

```
sounds_my_dataset/song9sample4.mp3
sounds_my_dataset/song7sample6.mp3
sounds_my_dataset/song9sample2.mp3
sounds_my_dataset/song9sample1.mp3
sounds_my_dataset/song7sample7.mp3
sounds_my_dataset/song9sample3.mp3
sounds_my_dataset/song7sample8.mp3
Cluster 3
sounds_my_dataset/song5sample5.mp3
sounds_my_dataset/song5sample2.mp3
sounds_my_dataset/song5sample3.mp3
sounds_my_dataset/song5sample6.mp3
sounds_my_dataset/song5sample1.mp3
sounds_my_dataset/song5sample4.mp3
Cluster 4
sounds_my_dataset/song1sample9.mp3
sounds_my_dataset/song1sample4.mp3
sounds_my_dataset/song1sample1.mp3
sounds_my_dataset/song1sample6.mp3
sounds_my_dataset/song1sample3.mp3
sounds_my_dataset/song1sample7.mp3
sounds_my_dataset/song1sample13.mp3
sounds_my_dataset/song1sample14.mp3
sounds_my_dataset/song1sample15.mp3
sounds_my_dataset/song1sample2.mp3
sounds_my_dataset/song1sample11.mp3
sounds_my_dataset/song1sample8.mp3
sounds_my_dataset/song1sample12.mp3
sounds_my_dataset/song1sample5.mp3
Cluster 5
sounds_my_dataset/song4sample1.mp3
sounds_my_dataset/song4sample9.mp3
sounds_my_dataset/song4sample8.mp3
sounds_my_dataset/song4sample5.mp3
sounds_my_dataset/song4sample6.mp3
sounds_my_dataset/song4sample2.mp3
sounds_my_dataset/song4sample11.mp3
sounds_my_dataset/song4sample3.mp3
sounds_my_dataset/song4sample4.mp3
sounds_my_dataset/song4sample7.mp3
Cluster 6
sounds_my_dataset/song3sample1.mp3
sounds_my_dataset/song3sample2.mp3
sounds_my_dataset/song3sample6.mp3
sounds_my_dataset/song3sample7.mp3
sounds_my_dataset/song3sample4.mp3
sounds_my_dataset/song3sample3.mp3
sounds_my_dataset/song3sample5.mp3
unmatched
sounds_my_dataset/song1sample10.mp3
```

```
97 sounds_my_dataset/song4sample10.mp3
```

Listing 5.1: Output clusters file without filtering. Clusters which correctly grouped only recordings from the same song/event are underlined.

Directly analysing listing 5.1, it is visible that 6 clusters were retrieved instead of the 10 expected from having 10 different songs in our dataset. However, this only occurred because there was no filtering of the wrong matches, meaning that clusters were wrongly merged at some point. Moreover, the audio fingerprinting algorithm failed to find matches in the database for 3 recordings, more concretely in:

- sounds_my_dataset/song1sample10.mp3
- sounds_my_dataset/song4sample10.mp3
- sounds_my_dataset/song1sample12.mp3

but since the latter was present in another sample's matching list, more concretely in sounds_my_dataset/song1sample1.mp3, it was put in that song's cluster (by the existence of a path between the two nodes) and therefore was not assigned as unmatched by our algorithm.

Regarding the incorrect merging of clusters generated from false positives matches, one can then try to filter them with the derivatives approach from the matches filtering method described in subsection 4.6. The clustering result after the matches filtering is shown in listing 5.2. From its analysis, we can see that now the algorithm returns the right number of clusters (10) with all samples from the same songs being put in the same cluster. More detailed information about the filtering process results is further discussed in section 5.3.

```
1  Cluster 1
2  sounds_my_dataset/song2sample3.mp3
3  sounds_my_dataset/song2sample8.mp3
4  sounds_my_dataset/song2sample5.mp3
5  sounds_my_dataset/song2sample4.mp3
6  sounds_my_dataset/song2sample1.mp3
7  sounds_my_dataset/song2sample6.mp3
8  sounds_my_dataset/song2sample2.mp3
9  sounds_my_dataset/song2sample7.mp3
10 Cluster 2
11 sounds_my_dataset/song7sample3.mp3
12 sounds_my_dataset/song7sample4.mp3
13 sounds_my_dataset/song7sample5.mp3
14 sounds_my_dataset/song7sample1.mp3
15 sounds_my_dataset/song7sample2.mp3
16 sounds_my_dataset/song7sample6.mp3
17 sounds_my_dataset/song7sample7.mp3
18 sounds_my_dataset/song7sample8.mp3
19 Cluster 3
```

```
sounds_my_dataset/song5sample2.mp3
sounds_my_dataset/song5sample5.mp3
sounds_my_dataset/song5sample3.mp3
sounds_my_dataset/song5sample6.mp3
sounds_my_dataset/song5sample1.mp3
sounds_my_dataset/song5sample4.mp3
Cluster 4
sounds_my_dataset/song1sample9.mp3
sounds_my_dataset/song1sample1.mp3
sounds_my_dataset/song1sample4.mp3
sounds_my_dataset/song1sample3.mp3
sounds_my_dataset/song1sample7.mp3
sounds_my_dataset/song1sample6.mp3
sounds_my_dataset/song1sample14.mp3
sounds_my_dataset/song1sample15.mp3
sounds_my_dataset/song1sample13.mp3
sounds_my_dataset/song1sample2.mp3
sounds_my_dataset/song1sample11.mp3
sounds_my_dataset/song1sample8.mp34
sounds_my_dataset/song1sample12.mp3
sounds_my_dataset/song1sample5.mp3
Cluster 5
sounds_my_dataset/song10sample1.mp3
sounds_my_dataset/song10sample9.mp3
sounds_my_dataset/song10sample3.mp3
sounds_my_dataset/song10sample10.mp3
sounds_my_dataset/song10sample8.mp3
sounds_my_dataset/song10sample6.mp3
sounds_my_dataset/song10sample11.mp3
sounds_my_dataset/song10sample2.mp3
sounds_my_dataset/song10sample5.mp3
sounds_my_dataset/song10sample4.mp3
sounds_my_dataset/song10sample7.mp3
Cluster 6
sounds_my_dataset/song9sample5.mp3
sounds_my_dataset/song9sample4.mp3
sounds_my_dataset/song9sample2.mp3
sounds_my_dataset/song9sample1.mp3
sounds_my_dataset/song9sample3.mp3
Cluster 7
sounds_my_dataset/song8sample2.mp3
sounds_my_dataset/song8sample6.mp3
sounds_my_dataset/song8sample3.mp3
sounds_my_dataset/song8sample1.mp3
sounds_my_dataset/song8sample5.mp3
sounds_my_dataset/song8sample8.mp3
sounds_my_dataset/song8sample7.mp3
sounds_my_dataset/song8sample4.mp3
Cluster 8
sounds_my_dataset/song4sample1.mp3
```

```
sounds_my_dataset/song4sample9.mp3
sounds_my_dataset/song4sample8.mp3
sounds_my_dataset/song4sample6.mp3
sounds_my_dataset/song4sample5.mp3
sounds_my_dataset/song4sample2.mp3
sounds_my_dataset/song4sample11.mp3
sounds_my_dataset/song4sample3.mp3
sounds_my_dataset/song4sample4.mp3
sounds_my_dataset/song4sample7.mp3
Cluster 9
sounds_my_dataset/song6sample5.mp3
sounds_my_dataset/song6sample10.mp3
sounds_my_dataset/song6sample7.mp3
sounds_my_dataset/song6sample6.mp3
sounds_my_dataset/song6sample1.mp3
sounds_my_dataset/song6sample11.mp3
sounds_my_dataset/song6sample4.mp3
sounds_my_dataset/song6sample2.mp3
sounds_my_dataset/song6sample3.mp3
sounds_my_dataset/song6sample9.mp3
sounds_my_dataset/song6sample8.mp3
Cluster 10
sounds_my_dataset/song3sample1.mp3
sounds_my_dataset/song3sample2.mp3
sounds_my_dataset/song3sample6.mp3
sounds_my_dataset/song3sample4.mp3
sounds_my_dataset/song3sample7.mp3
sounds_my_dataset/song3sample3.mp3
sounds_my_dataset/song3sample5.mp3
unmatched
sounds_my_dataset/song1sample10.mp3
sounds_my_dataset/song4sample10.mp3
```

Listing 5.2: Output clusters file.

The audio clustering method proposed can then be evaluated by the number of clusters formed and by the number of samples unable to be assigned to any cluster (unmatched samples). Table 5.3 provides the benchmarks of our clustering approach when performing filtering or not regarding the ground-truth of our dataset. The number of retrieved clusters when considering all matches and performing filtering is correct (that is, it is the same as for the ground-truth), whereas without the filtering process clusters were wrongly merged, making the system only retrieve 6 clusters instead of 10.

5.3 Matches filtering - Derivatives approach

Following the successful clustering results enabled by the correct filtering of the wrong matches, we will now discuss the metrics details of the overall filtering process. By analysis of table 5.4, we can see that the analysis of derivative drops succeeded in detecting all

	# Clusters	# Unmatched samples
Ground-truth	10	0
Clustering (no filtering)	6	2
Clustering (filtering)	10	2

Table 5.3: Clustering benchmarks.

	# Instances	% Instances	# Filtered	% Filtered
Total matches	620	100.00%	70	11.29%
False positives (landmark-level)	5	0.81%	5	100.00%
False positives (sample-level)	4	4.40%	4	100.00%
False negatives (landmark-level)	-	-	65	10.48%

Table 5.4: Filtering benchmarks using the derivatives analysis approach.

false positives retrieved by the audio fingerprinting algorithm – 1 out of the 5 landmark-level false positives was discarded by eliminating repetitions whilst the rest were detected by the analysis of the derivative values. It is important to notice that even though there were occurrences of false negatives (10.48 % of the overall retrieved landmarks), this does not affect the clustering results, as presented in table 5.3 by the retrieval of the right number of clusters.

The derivative analysis of the 4 sample-level wrong matches, and the consecutive discarded samples, are illustrated in figure 5.3. Note that all wrong matches are the last positioned sample in all matching lists, and, thence, the last x value of the graphics. Considering the top graphics, we can see that some true positives matches were also discarded whereas in the bottom graphs only the false positive match was discarded.

Each graphic in figure 5.3 is related to the information about each sample in the matching list of the following samples:

- sounds_my_dataset/song6sample1.mp3
- sounds_my_dataset/song8sample3.mp3
- sounds_my_dataset/song9sample2.mp3
- sounds_my_dataset/song10sample8.mp3

and that were matched with samples belonging to different events (*i.e.*, songs) as their own, more specifically to:

- sounds_my_dataset/song8sample2.mp3
- sounds_my_dataset/song10sample8.mp3

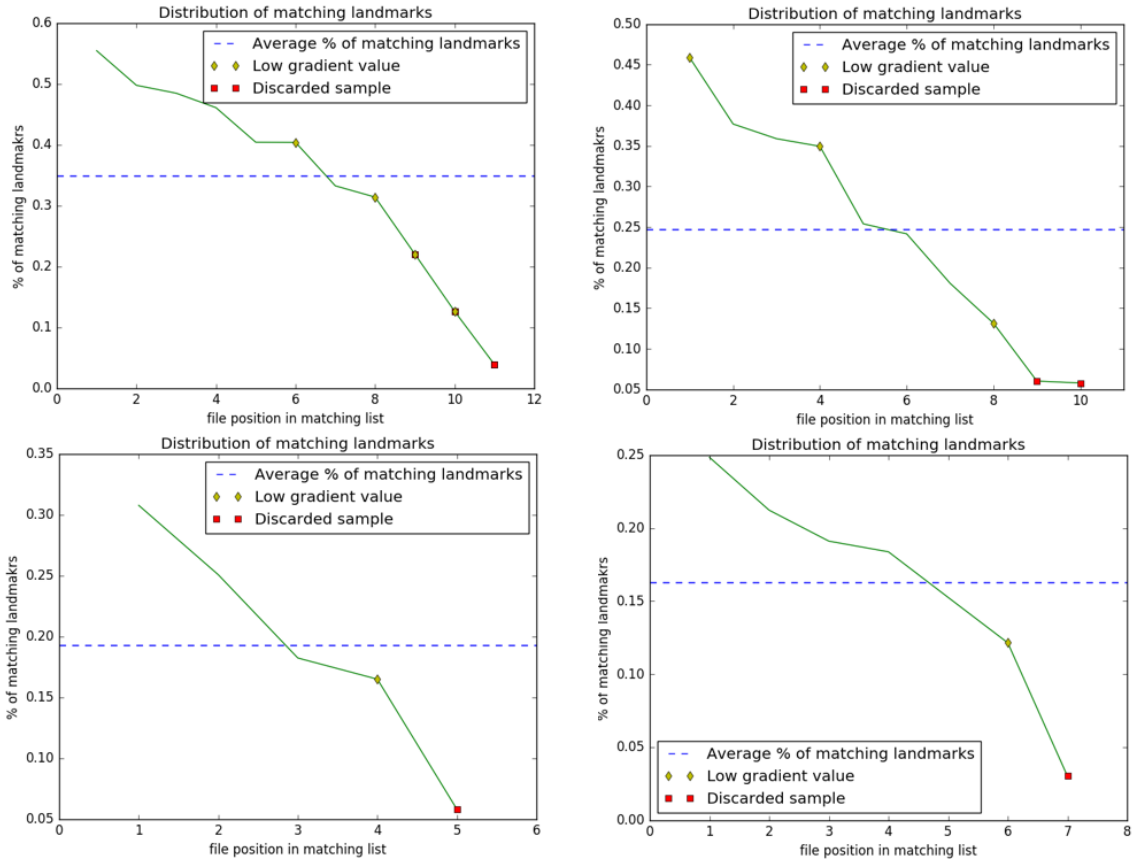

Figure 5.3: Analysis of the percentage of matching landmarks of the different matches information in the matching lists where a false positive match was present. The derivatives that are lower than the derivative threshold (set to -0.07) are represented as yellow diamonds, whereas the effectively discarded samples are represented with red squares. Important to notice that samples can only be discarded if their percentage of matching landmarks is lower than the average of all samples' matching landmarks in the matching list (represented as the blue dotted line in the graphs).

- sounds_my_dataset/song7sample4.mp3
- sounds_my_dataset/song2sample2.mp3

5.4 Audio segmentation

Following the Audio Segmentation approach proposed in section 4.5, it is possible to know how the different samples of a given cluster are spread across that cluster's timeline. Our algorithm gives this information by the form of segments, that contain two timestamps, concerning a given time range, and a list of samples that are available in that time. The different segments found for each of the 10 clusters are presented in listing B.1 in appendix B.

The samples inside each segment are represented with their respective file path, since new samples were generated by cutting the original sample audio file for all segments

that audio file is part of. This is one of the key aspects to perform the audio quality inference inside a given segment, since it allow us to match the correct intervals of the different samples. Furthermore it allows the extension of the training set described in subsection 4.6.2.4 by the analysis of the different matched offsets - 0.0 sec is the expected offset for all matches since they are all time synchronised.

Additionally, the audio quality score of each sample (presented in section 4.4) is also represented, being the samples listed in descending order of their quality score. Important to notice, however, that in the cases where the time interval of the segments or the number of samples present in a given segment is considerably small, such quality inference cannot be assigned, being the score set to 0, meaning that such audio sample did not have matched landmarks with any of the other samples inside its segment.

5.5 Audio quality inference

To evaluate the quality inference of our solution, we analysed the position of the professional recording (consisting of all `sample1` of each song in our dataset) relative to the position of all the other samples in the cluster using our method and Kennedy and Naaman's method in [22] (K.M. method). This approach considers the nodes with more edges in the samples graph as the ones with higher quality. Listing 5.2 already presented information about each sample's quality, since all samples inside each cluster are in descending order based on their score. Nevertheless, for this evaluation step, we are only concerned how the `sample1` of each cluster ranks in the overall sample list, by checking its position.

This evaluation assumes that the professional recordings represents one of the highest quality recordings in any cluster, and, therefore, should be placed at the top of the ranking list. Table 5.5 shows the comparison between the two methods. The second column indicates the number of samples in the cluster, the third and fourth columns show the position of the professional recording in the ranking list with our method and the K.M. method, respectively. The first position in the ranking list corresponds to the highest score.

It is important to note that there may be ambiguity on the quality scores of the K.M. method, because several samples can have the same score. This is observed in table 5.5 by the range of positions of the professional sample. Such ambiguity does not appear with our approach, which is essential for aiding the end-users on the decision of which samples to use. The table also shows that the score of the professional track with our method is always the same or better than with the K.M method, as it is either included within the range given by the K.M method or it is even ranked better (clusters 5 and 8).

Cluster	# Samples	Proposed method	K.M. method
1	8	5th	3th-5th
2	8	4th	1st-4th
3	6	5th	3th-5th
4	14	2nd	3th-4th
5	11	1st	2nd-4th
6	5	4th	1st-5th
7	8	4th	1st-4th
8	10	1st	2nd-4th
9	11	5th	3th-5th
10	7	1st	1st-4th

Table 5.5: Quality inference. Position of the professional recording in the ranking list with our method (third column) and Kennedy and Naaman's method (fourth column).

5.6 Matches filtering - Learning approach

Even though the derivative approach for filtering the false positive matches succeeded in filtering all false positive of the original dataset, it only sufficed to filter 4 out of the 6 wrong matches present in the extended dataset (as shown in figure 5.4). This came to prove the possible lack of generalisation inherent to the derivatives approach, since the threshold used was fine-tuned after the analysis of the derivatives of all the false matches returned by the audio fingerprinting algorithm in order to filter all the wrong matches whilst minimising the number of true positives also filtered.

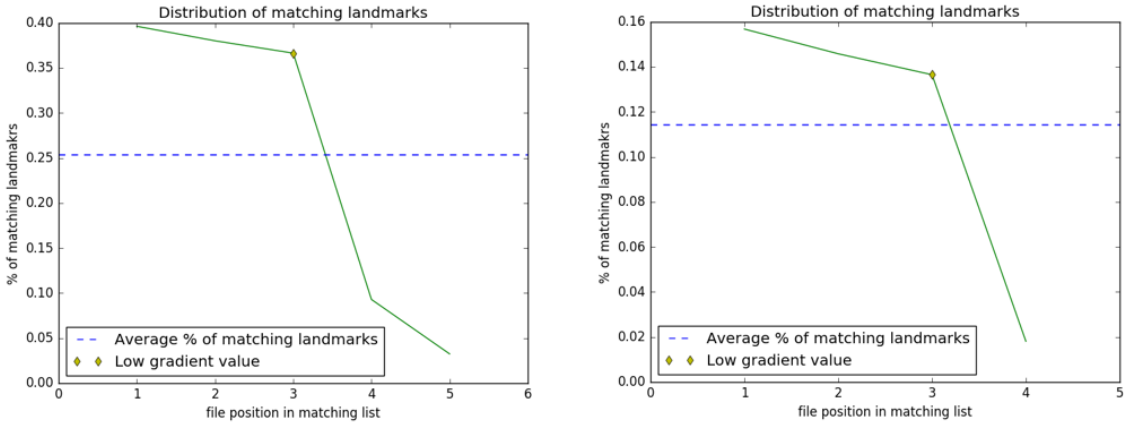

Figure 5.4: In these two examples of the matching lists of two different songs, since no derivative lower than the threshold was found bellow the average percentage of matching landmarks of the matching list (represented as the blue dotted line) no sample was discarded even though the last samples of each graphs were false positive matches (*i.e.*, 5'th and 4'th sample, for the left-side and the right-side graphic respectively).

One possible modification to the derivatives approach would be to exclude the constraint of only discarding samples when derivatives are from samples that have the percentage of matching landmarks lower than the average of the matching list. This would

suffice to filter the wrong matched samples present in both graphs of figure 5.4. However, this would implicate that a lot more true positive samples would be considered false positives and therefore discarded, which could affect the overall creation of the clusters. Thence, the next subsections will then focus on performing the matches filtering using machine learning techniques in order to allow more generalisation in terms of the datasets used.

5.6.1 Training set

The extended 23-song dataset previously referred in section 5.1, generated 3098 matches (referred as training/test samples) by the audio fingerprinting algorithm, and that consisted of the actual dataset used to both train, validate, and test our models. The number of true matched samples were 1071 whereas the number of false matched samples (*e.g.*, class 1) were 2027, from which 2021 were repetition samples and 6 were false matches (*i.e.*, recordings from different events were retrieved as a match by the audio fingerprinting algorithm). Important to notice that every time a new model was trained, the training set was balanced, meaning that the number of samples of class 0 was equal to the number of samples of class 1.

5.6.2 Parameters setting

In order to select the best models for the different classifiers (Logistic Regression, K-Nearest Neighbours, and Support-Vector Machines), the parameters used in the models with lowest validation error during the leave-one-song-out cross-validation were assigned as the most suitable parameters, as previously discussed in subsection 4.6.2. Since we wanted to also test which feature combinations would be more relevant, this process was made for each feature sub-set presented in subsection 4.6.2.2. The model errors, more concretely the training and validation error, for each classifier's parameter values tested are presented in figures 5.5, 5.6, 5.7, and 5.8, with each one of the figures representing the results obtained for each sub-set of feature spaces previously described in sub-subsection 4.6.2.2.

It is important to notice that the different parameter combinations used for each SVM is different for each sub-set of features used since these values were obtained by performing Grid-Search for all the different training sets used whilst doing leave-one-song-out cross-validation. Moreover, the leave-one-song-out cross-validation is what allowed us to test the different models considering the whole dataset, enabling to get the prediction accuracy of the different models. This analysis is further discussed in the next subsection 5.6.3.

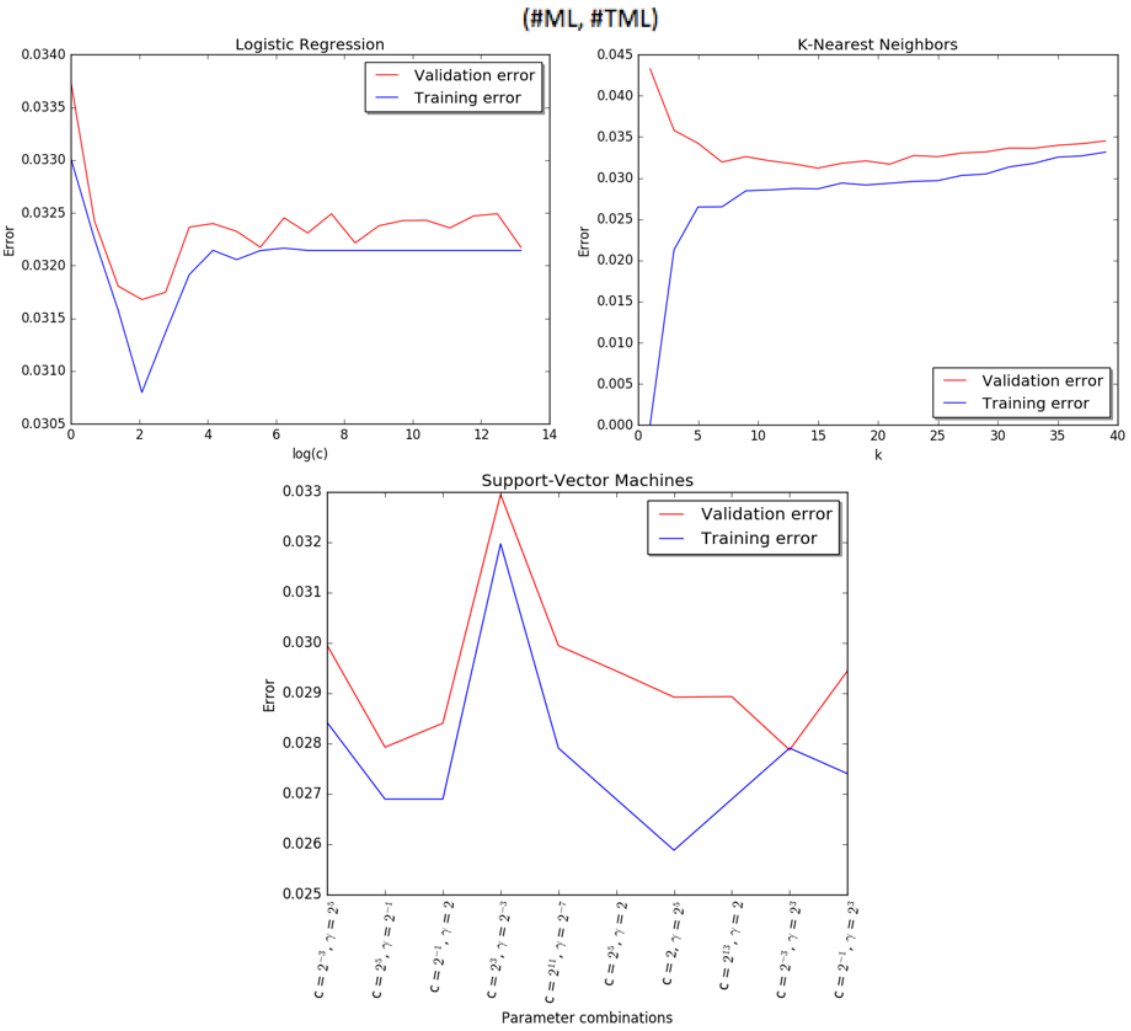

Figure 5.5: Model errors of the different classifiers using the number of matching landmarks of the right offset (#ML) and the total number of matching landmarks in all offsets found (total#ML) as features. The parameter values with lowest validation errors (v_a) for each classifier are: LR (c = 8, log(c) = 2.079, $v_a \approx 0.0316$), KNN (k = 15, $v_a \approx 0.0315$), SVM ($c = 0.125, \gamma = 8$, $v_a \approx 0.0279$).

5.6.3 Prediction results

The results in terms of the accuracy of the predictions of the different classifier's models for the respective sub-set of features (previously described in sub-subsection 4.6.2.2) is shown in Figure 5.9. The SVM showed better results across the different feature combinations (98.23%, 97.22%,, 96.12%, and 97.68% respectively) but was closely followed by the other classifiers with the exception of when using Logistic Regression with the number of matching landmarks, and the overall number of landmarks of the query and the match song, that achieved considerably lower accuracy (82.07%).

Despite the high accuracy, false positives in the scope of our problem implicates songs of different events to be assigned to the same cluster, leading ultimately the merge of clusters of different events. Therefore, instead of simply choosing the model with lowest

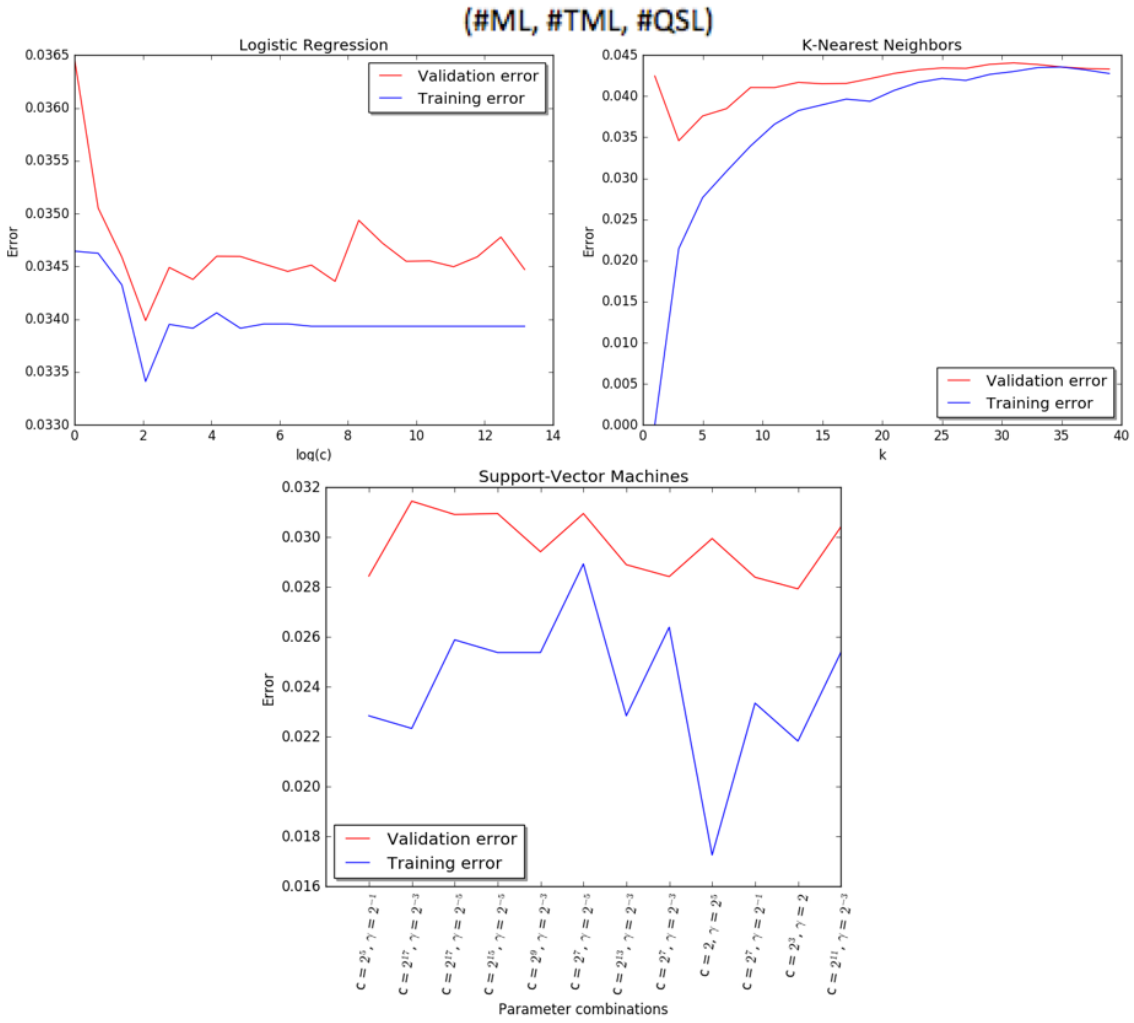

Figure 5.6: Model errors of the different classifiers using the number of matching landmarks of the right offset (#ML), the total number of matching landmarks in all offsets found (total#ML), and the number of landmarks of the query song (query#ML) as features. The parameter values with lowest validation errors (v_a) for each classifier are: LR (c = 8, log(c) = 2.079, $v_a \approx 0.0342$), KNN (k = 3, $v_a \approx 0.0347$), SVM ($c = 8, \gamma = 2$, $v_a \approx 0.0279$).

validation error for each classifier, we can discard all models that wrongly classified the false positives and choose the lower validation error model of the remaining. Figure 5.10 shows the updated classifiers results adding this constraint.

Even though the accuracy slightly decreased, we managed to find new models for KNN and SVM that satisfy our condition of not allowing false positives, whilst maintaining a high accuracy (97.12% and 97.49%, respectively). The Logistic Regression models already presented in Figure 5.9 remained intact since there was no wrong classification of false positives, except when using the feature combination (#ML, total#ML, match#ML), with their accuracy of 97.40% for the first presented feature combination, and 97.21% for the latter.

In sum, we managed to achieve high accuracy results in most of the situations, with the

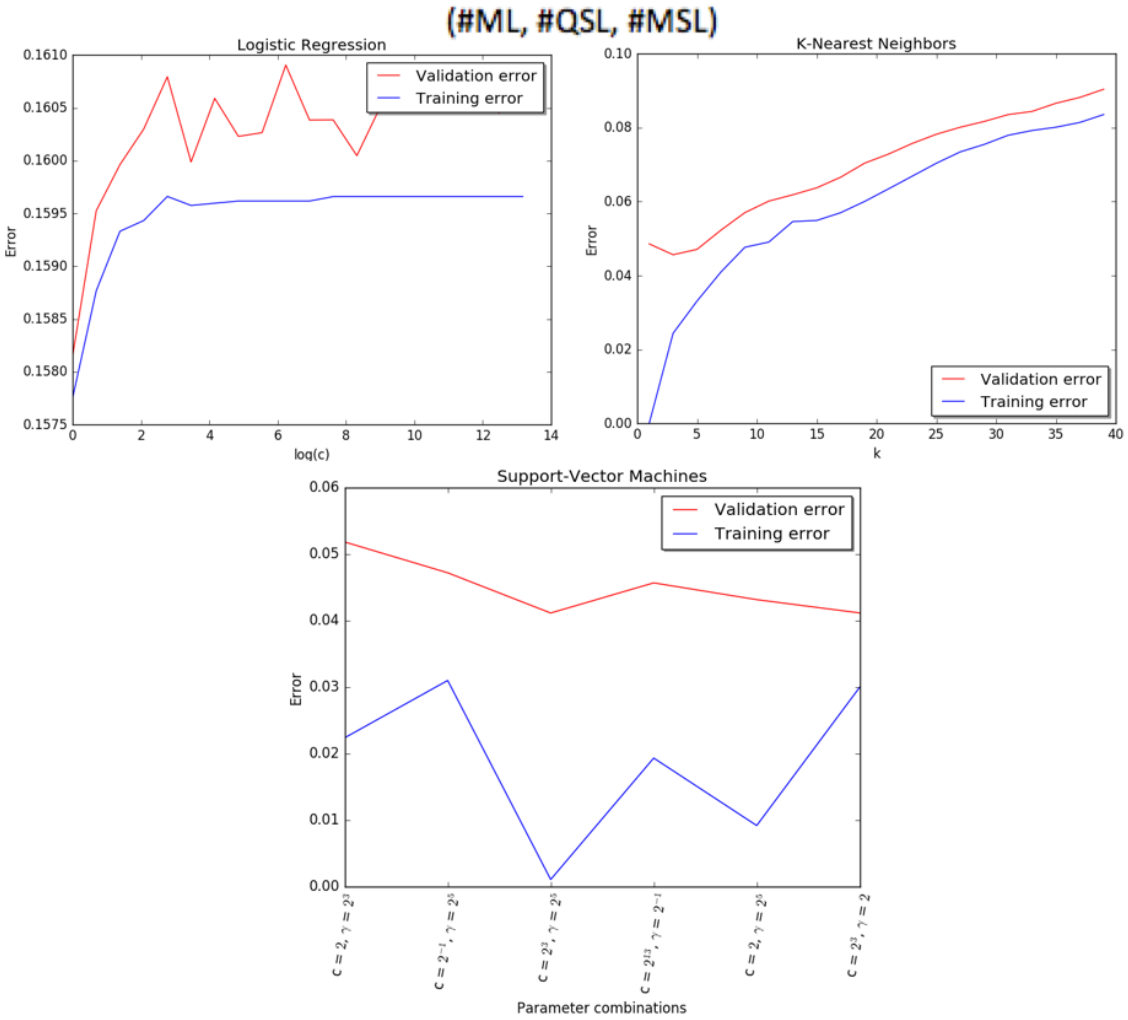

Figure 5.7: Model errors of the different classifiers using the number of matching landmarks of the right offset (#ML), the number of landmarks of the query song (query#ML) and matched song (match#ML) as features. The parameter values with lowest validation errors (v_a) for each classifier are: LR (c = 1, log(c) = 0, $v_a \approx 0.1582$), KNN (k = 3, $v_a \approx 0.0454$), SVM ($c = 8, \gamma = 32$, $v_a \approx 0.0410$).

context of using 4 features with SVM representing a slight advantage when considering only the models with no wrong classification of false positives since their filtering is crucial in the proposed solution. However, using Logistic Regression and KNN with the features set as the matching landmarks in the right offset and the total number of matching landmarks in all detected offsets, as well as using Logistic Regression with the 4 feature-combination, would also represent practically viable options for the presented filtering approach.

Concluding remarks

The results retrieved by our solution when using an implementation of the previously proposed methods are presented in this chapter. Such evaluation was made using two different datasets of concert recordings crawled from YouTube, with the extended version

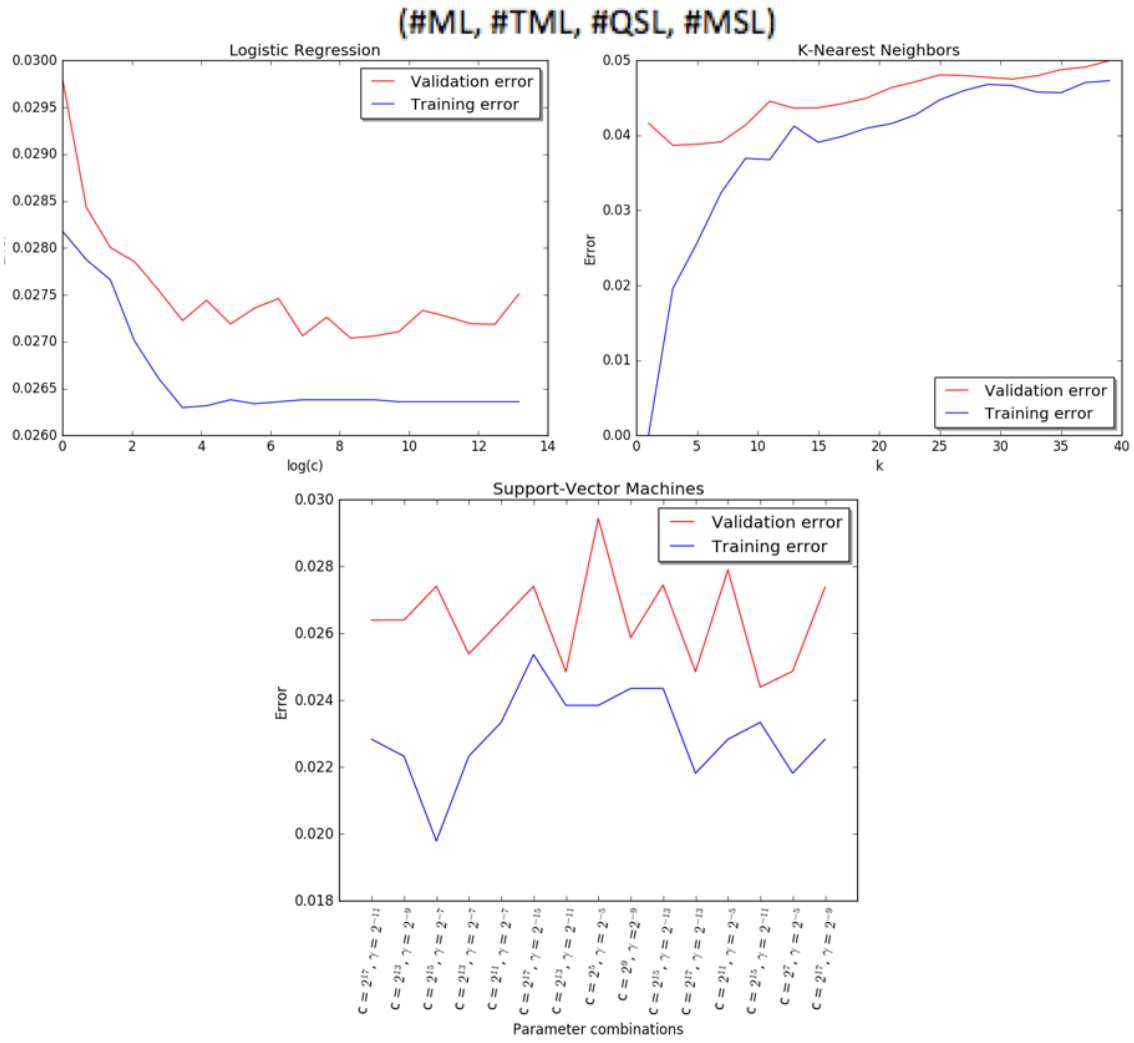

Figure 5.8: Model errors of the different classifiers using the number of matching landmarks of the right offset (#ML), the total number of matching landmarks in all offsets found (total#ML), the number of landmarks of the query song (query#ML) and matched song (match#ML) as features. The parameter values with lowest validation errors (v_a) for each classifier are: LR (c = 16384, log(c) = 9.704, $v_a \approx 0.0269$), KNN (k = 3, $v_a \approx 0.0.0380$), SVM ($c = 2^{15}, \gamma = 2^{-11}$, $v_a \approx 0.0243$).

being only used to validate the matches filtering learning approach. The results obtained proved that the different methods of our solution work even when dealing with real-world data, and that the proposed method for inferring the audio quality of the audio files outperforms the current state-of-the-art.

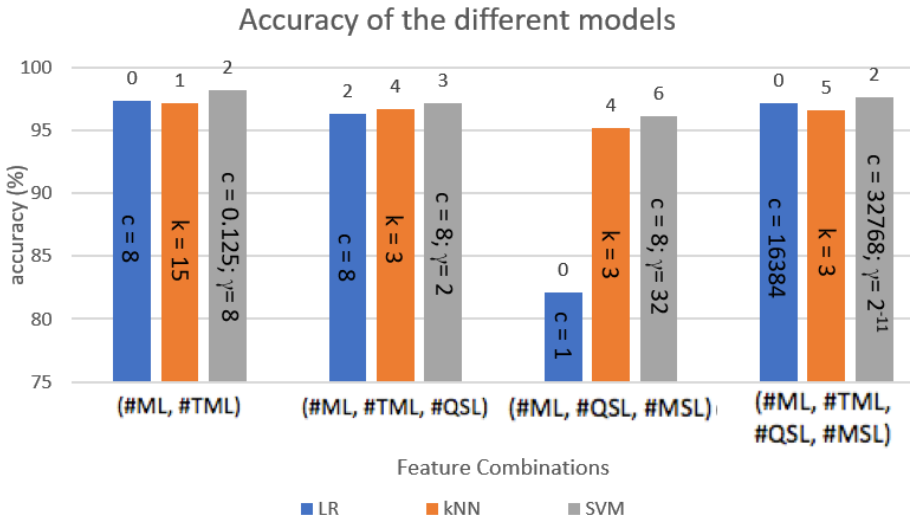

Figure 5.9: Accuracy of the best models (*i.e.*, with lowest validation error) of each classifier for the different combination of features, with the respective parameter values described inside each bar. The numbers placed on top of each bar represent the number of false positives retrieved by each model.

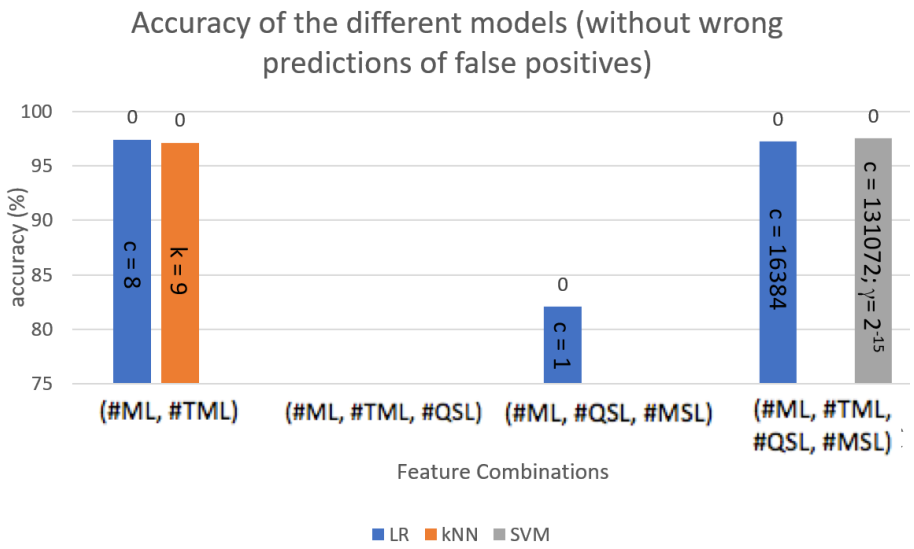

Figure 5.10: Accuracy of the models with lowest validation error that did not retrieved any false positives. The missing models represent that all retrieved models classified at least one false positive, with no classifier being able to surpass this constraint when using the features combination (#ML, total#ML, query#L). New models with different parameter values were found for both KNN and SVM whilst respecting this condition.

Chapter 6

Conclusion

In this thesis we propose methods that enable a better understanding of user-generated audio recordings from various concert songs. This chapter will give an overview of all the different proposed methodologies, including the insights derived from doing this thesis and possible future work.

For a better comprehension and management of large datasets of audio files, we propose a method that clusters the data relative to certain events by the analysis of overlapping audio intervals between the different files retrieved by the audio fingerprinting algorithm. While performing audio clustering using audio fingerprinting was previously proposed in [22], we propose relevant improvements to their methodology, more specifically the detection and filtering of possible false positives retrieved by the audio fingerprinting algorithm. We thereby present two different approaches to achieve this. The first is based on the analysis of relevant drops in the percentage of matching landmarks of all non-repetitive samples in the matching list (section 4.6.1). The second involves classification techniques commonly used in machine learning in order to classify matches as true positives or false positives (subsection 4.6.2), whilst ensuring that the trained classifiers learn from previous predictions by extending the training set with further runs of our algorithm (sub-subsection 4.6.2.4).

In addition, we also propose a concrete method that retrieves how the different audio files inside a cluster are distributed over time (section 4.5). Since the offset information between all matches made by the audio fingerprinting algorithm is represented as the edges weights in a graph where the nodes are all existing samples in the database, this method only needs to calculate the cost of the path between the different nodes to get the offset between them. This synchronisation of all samples in a cluster is effectively accomplished by retrieving all offsets of all audio files relative to the earliest starting cluster sample.

Moreover, we propose a novel quality inference method that attributes a quality score to the different samples, both cluster-wise (*i.e.*, relative to all the other samples in the cluster) and segment-wise (*i.e.*, relative to the rest of the samples in the respective segment). The proposed scoring method is relative to the number of matching landmarks a given song had, when matched against all the other songs in the original database. This is based not only in the notion that higher-quality audio files will produce more matches by the audio fingerprinting algorithm, since the lower-quality files will most likely match in their less noisy sections to those files, but also that those matches will be more accurate, meaning that they will produce more landmark matches. Thus, instead of only counting the matches a given audio file had, as mentioned by Kennedy and Naaman in [22], we further look at how those matches were obtained by looking at the number of matching landmarks of the higher matched offset.

The proposed methods were further validated and evaluated by the 10-song test setup described in chapter 5. In terms of the organisation of the different samples, the clustering method proposed in section 4.3 succeeded in correctly identifying the right number of clusters, whilst grouping correctly the samples in the respective clusters, with exception of 2 audio files that could not be matched by the audio fingerprinting algorithm itself. Important to notice that these clustering results were only obtained by accomplishing the filtering of all false positives matches by the derivatives approach presented in subsection 4.6.1. Furthermore, the synchronisation of the different files of each one of the clusters was also performed correctly. In terms of quality inference, the proposed method succeeded in either classifying equally, and even better the professional recordings in some cases, to when compared to the method proposed in [22], whilst also avoiding ambiguity in the quality scores of the audio files.

Furthermore, the test setup was extended in order to evaluate the learning approach presented in subsection 4.6.2 to classify and consecutively filter the false positives classified matches, promoting a better training process for the different classifiers due to the increase of the original dataset. Several feature combinations were evaluated, together with several parameter values for each classifier (*i.e.*, Logistic Regression, K-Nearest Neighbours and Support-Vector Machines). Performing leave-one-song-out cross-validation, that consists of leaving out of the training set all matches relative to all samples of a given song and use such samples as the test set and repeating this process until all songs were left-out, we managed to evaluate the accuracy of the predictions of each classifier's models.

By looking solely at the percentage of correct predictions of the different models, we managed to achieve high accuracy for the best models of each classifier (above 95 % in all cases except for Logistic Regression when using the number of matching landmarks at the right offset, and the total number of the query and matched sample as features that obtained an accuracy of 82.07%). By considering only the models with optimal filtering results (*i.e.*, that successfully filtered all false positives in the dataset) we maintained high accuracy values, achieving 97.49% of correct predictions when using SVM with 4

features. Thus, the learning approach might be more suitable in most cases than the derivatives approach since it will most likely classify less false negatives, enabling the correct formation of all clusters, and it empowers more generalisation of the dataset by not having the need to fine tune threshold values like the derivatives threshold inherent to the derivatives approach. Moreover, the extension of the training set described in subsection 4.6.2 enables our models to be continuously learning with the usage of our algorithm.

6.1 Future work

One possible way of improving the learning process would be to add features inherent of the audio signal itself, more particularly high-level features retrieved directly from the spectrogram. This way, by merging this information with the output of the audio fingerprinting algorithm (by maintaining the set of features used in this work), one could better represent the audio files that were part of each match which could be a good indicator for separating noisy files from good quality files in the feature space. Such separation would hopefully mean an easier separation of both classes by the classifiers and better prediction results. However, one would have to be cautious about the *Curse of Dimensionality*, since there would be an increase in the number of features used and therefore the dimension of the feature space.

Instead of evaluating the audio quality inference methods by the rank quality scores of the professional recordings, assuming therefore that they have always higher quality than the user-generated recordings (which might not necessarily be the case given the high-quality devices in the market), one could perform subjective listening tests, where a large amount of participants is asked questions regarding the sound quality of several songs. Even though these types of tests can produce good results when adapted to better match some human listening capabilities (as discussed in [40]), it would not only be impractical, since our dataset is considerably big, but also very time consuming.

On the other hand, one could use an objective testing approach. These could rely upon using audio features from the signal itself (*i.e.*, operating in the time and spectral domain [47]) to derive the audio quality of a given audio file, or even using techniques in the perceptual domain (*i.e.*, based on models of the human auditory system) to infer such audio quality such as PEAQ ([46]). Although objective score measurements might not always reflect real world subjective opinions, it still proves to be a reliable version given their ability to perform a large number of tests that would not be practical to perform otherwise and therefore could consist of a good approach to follow to further evaluate the proposed quality inference method.

Bibliography

[1] A.C.Circuits. *Unit 2: Introduction to Alternating Current and Voltage*. URL: http://nreeder.com/eet1155/mod02.htm.

[2] ACRCloud. *Automatic Content Recognition Cloud Services*. URL: https://www.acrcloud.com.

[3] AllAboutCircuits. *AC Waveforms*. URL: http://www.allaboutcircuits.com/textbook/alternating-current/chpt-1/ac-waveforms/.

[4] E. Allamanche, J. Herre, O. Hellmuth, B. Fröba, T. Kastner, and M. Cremer. "Content-based Identification of Audio Material Using MPEG-7 Low Level Description." In: *ISMIR*. 2001.

[5] J. Brownlee. *An Introduction to Feature Selection*. URL: http://machinelearningmastery.com/an-introduction-to-feature-selection/.

[6] P. Cano, E. Batlle, E. Gómez, L. Gomes, and M. Bonnet. "Audio Fingerprinting Concepts and Applications." In: Springer-Verlag, 2005, pp. 233–245. URL: files/publications/0e9cd9-Springer05-pcano.pdf.

[7] P. Cano. "Content-based audio search: from fingerprinting to semantic audio retrieval". PhD thesis. Citeseer, 2006.

[8] COGNITUS. *COGNITUS Website*. URL: http://cognitus-h2020.eu/.

[9] COGNITUS. *Two open access datasets with User Generated Audio recordings*. URL: http://cognitus-h2020.eu/index.php/2017/01/06/two-open-access-datasets-with-user-generated-audio-recordings/.

[10] C. V. Cotton and D. P. Ellis. "Audio fingerprinting to identify multiple videos of an event". In: *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*. IEEE. 2010, pp. 2386–2389.

[11] I. J. Cox, J. Kilian, F. T. Leighton, and T. Shamoon. "Secure Spread Spectrum Watermarking for Multimedia". In: *Trans. Img. Proc.* 6.12 (Dec. 1997), pp. 1673–1687. ISSN: 1057-7149. DOI: 10.1109/83.650120. URL: http://dx.doi.org/10.1109/83.650120.

[12] M. Cremer, B. Froba, O. Hellmuth, J. Herre, and E. Allamanche. "AudioID: Towards Content-Based Identification of Audio Material". In: *Audio Engineering Society Convention 110*. 2001. URL: http://www.aes.org/e-lib/browse.cfm?elib=10019.

[13] D. Ellis. *Robust Landmark-Based Audio Fingerprinting*. 2009. URL: http://labrosa.ee.columbia.edu/matlab/fingerprint/.

[14] D. Ellis. *Landmark-based audio fingerprinting*. URL: https://github.com/dpwe/audfprint.

[15] D. Fragoulis, G. Rousopoulos, T. Panagopoulos, C. Alexiou, and C. Papaodysseus. "On the Automated Recognition of Seriously Distorted Musical Recordings". In: *Trans. Sig. Proc.* 49.4 (Apr. 2001), pp. 898–908. ISSN: 1053-587X. DOI: 10.1109/78.912932. URL: http://dx.doi.org/10.1109/78.912932.

[16] J. Haitsma and T. Kalker. "A Highly Robust Audio Fingerprinting System." In: *ISMIR*.

[17] J. Haitsma, M. van der Veen, T. Kalker, and F. Bruekers. "Audio Watermarking for Monitoring and Copy Protection". In: *Proceedings of the 2000 ACM Workshops on Multimedia*. MULTIMEDIA '00. ACM, 2000, pp. 119–122. ISBN: 1-58113-311-1. DOI: 10.1145/357744.357895. URL: http://doi.acm.org/10.1145/357744.357895.

[18] W. S. Harlan. *Bounded geometric growth: motivation for the logistic function*. URL: http://www.billharlan.com/papers/logistic/logistic.html.

[19] C. Hsu, C. Chang, C. Lin, et al. "A practical guide to support vector classification". In: (2003).

[20] C. wei Hsu, C. chung Chang, and C. jen Lin. *A practical guide to support vector classification*. 2010.

[21] Jiaaro. *Pydub*. URL: https://github.com/jiaaro/pydubl.

[22] L. Kennedy and M. Naaman. "Less Talk, More Rock: Automated Organization of Community-contributed Collections of Concert Videos". In: *Proceedings of the 18th International Conference on World Wide Web*. WWW '09. ACM, 2009, pp. 311–320. ISBN: 978-1-60558-487-4. DOI: 10.1145/1526709.1526752. URL: http://doi.acm.org/10.1145/1526709.1526752.

[23] L. Krippahl. *Aprendizagem Automática (Machine Learning) - Lecture Notes*. 2016.

[24] scikit learn. *RBF SVM parameters*. URL: http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html.

[25] B. Logan. "Mel Frequency Cepstral Coefficients for Music Modeling". In: *In International Symposium on Music Information Retrieval*. 2000.

[26] C. Lopes, A. Veiga, and F. Perdigão. "Using Fingerprinting to Aid Audio Segmentation". In: *Proc. of the VI Jornadas en Tecnología del Habla and II Iberian SLTech Workshop, FALA*. 2010.

[27] MEDCALC. *Logistic regression*. URL: https://www.medcalc.org/manual/logistic_regression.php.

[28] MediaCollege. *Sound Wave Properties*. URL: http://www.mediacollege.com/audio/01/wave-properties.html.

[29] Midomi. *Chromaprint | AcoustID*. URL: https://acoustid.org/chromaprint.

[30] Midomi. *Search for Music Using Your Voice*. URL: http://www.midomi.com/index.php.

[31] G. Mordido, J. Magalhaes, and S. Cavaco. "Automatic Organisation and Quality Analysis of User-Generated Content with Audio Fingerprinting". In: *2017 25th European Signal Processing Conference (EUSIPCO) (EUSIPCO 2017)*. 2017, pp. 1864–1868.

[32] G. Mordido, J. Magalhaes, and S. Cavaco. "Automatic Organisation, Segmentation, and Filtering of User-Generated Audio Content". In: *2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP)*. 2017.

[33] Mutagen. *MP3 - Mutagen*. URL: http://mutagen.readthedocs.io/en/latest/api/mp3.html.

[34] NetworkX. *NetworkX*. URL: https://networkx.github.io/.

[35] C. Neves, A. Veiga, L. Sá, and F Perdigo. "Audio fingerprinting system for broadcast streams". In: *Proc Conf Telecommunications*. Vol. 1. 2009, pp. 481–484.

[36] NumPy. *NumPy*. URL: http://www.numpy.org/.

[37] OpenCV. *Introduction to Support Vector Machines*. URL: http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html.

[38] H. Ozer, B. Sankur, and N. Memon. "Robust audio hashing for audio identification". In: *Signal Processing Conference, 2004 12th European*. IEEE. 2004, pp. 2091–2094.

[39] Plotly. *Plotly*. URL: https://plot.ly/.

[40] K. Precoda and T. H. Meng. "Subjective Audio Testing Methodology and Human Performance Factors". In: *Audio Engineering Society Convention 103*. 1997. URL: http://www.aes.org/e-lib/browse.cfm?elib=7194.

[41] ProSoundWeb. *Phase & Polarity: Causes And Effects, Differences, Consequences*. URL: http://www.prosoundweb.com/article/polarity_and_phase_explained/.

[42] sebastianraschka. *Naive Bayes and Text Classification*. URL: http://sebastianraschka.com/Articles/2014_naive_bayes_1.html.

[43] Statistica. *k-Nearest Neighbors*. URL: http://www.statsoft.com/Textbook/k-Nearest-Neighbors#distancemetric.

[44] N. Stefanakis. *Organization of UGC based on audio*. URL: http://cognitus-h2020.eu/index.php/2017/01/06/two-open-access-datasets-with-user-generated-audio-recordings/.

[45] SVMTutorial. *SVM - Understanding the math - Duality and Lagrange multipliers*. URL: https://www.svm-tutorial.com/2016/09/duality-lagrange-multipliers/.

[46] T. Thiede, W. C. Treurniet, R. Bitto, C. Schmidmer, T. Sporer, J. G. Beerends, and C. Colomes. "PEAQ - The ITU Standard for Objective Measurement of Perceived Audio Quality". In: *J. Acoust. Soc. Am.* 48.1/2 (2000).

[47] L. Thorpe and W. Yang. "Performance of current perceptual objective speech quality measures". In: *1999 IEEE Workshop on Speech Coding Proceedings. Model, Coders, and Error Criteria (Cat. No.99EX351)*. 1999, pp. 144–146. DOI: 10.1109/SCFT.1999.781512.

[48] J. H. Ton and T. Kalker. "Robust Audio Hashing for Content Identification". In: *In Content-Based Multimedia Indexing (CBMI*. 2001.

[49] A. Wang. "The Shazam Music Recognition Service". In: *Commun. ACM* 49.8 (Aug. 2006), pp. 44–48. ISSN: 0001-0782. DOI: 10.1145/1145287.1145312. URL: http://doi.acm.org/10.1145/1145287.1145312.

[50] A. L.-C. Wang. *An Industrial-Strength Audio Search Algorithm*. http://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf. 2003.

[51] wikipedia. *Agglomerative Clustering*. URL: http://stackoverflow.com/questions/8016313/agglomerative-clustering-in-matlab.

[52] wikipedia. *Hierarchical clustering*. URL: https://en.wikipedia.org/wiki/Hierarchical_clustering.

[53] Wikiwand. *Spectrogram*. URL: http://www.wikiwand.com/en/Spectrogram.

[54] U. of Wisconsin-Madison. *RMS Amplitude*. URL: https://csd.wisc.edu/vcd202/rms.html.

[55] Q. Xiao, M. Suzuki, and K. Kita. "Fast Hamming Space Search for Audio Fingerprinting Systems". In: *Proceedings of the 12th International Society for Music Information Retrieval Conference*. http://ismir2011.ismir.net/papers/PS1-16.pdf. Miami (Florida), USA, 2011, pp. 133–138.

[56] W. A. Yost. *Fundamentals of Hearing: An Introduction, 5th edition*.

[57] YouTube. *YouTube*. URL: https://www.youtube.com/.

Appendix

Accepted scientific papers

In this appendix we present both of the accepted papers submissions to EUSIPCO 2017 (25th European Signal Processing Conference) held in Kos Island, Greece, and MMSP 2017 (IEEE 19th International Workshop on Multimedia Signal Processing) located in Luton, UK, respectively.

Automatic organisation and quality analysis of user-generated content with audio fingerprinting

Gonçalo Mordido, João Magalhães, Sofia Cavaco
NOVA LINCS, Departamento de Informática
Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa
2829-516 Caparica, Portugal
goncalomordido@gmail.com, {jmag, scavaco}@fct.unl.pt

***Abstract*—The increase of the quantity of user-generated content experienced in social media has boosted the importance of analysing and organising the content by its quality. Here, we propose a method that uses audio fingerprinting to organise and infer the quality of user-generated audio content. The proposed method detects the overlapping segments between different audio clips to organise and cluster the data according to events, and to infer the audio quality of the samples. A test setup with concert recordings manually crawled from YouTube is used to validate the presented method. The results show that the proposed method achieves better results than previous methods.**

I. Introduction

The abundance and ubiquity of user-generated content has increased the demand for tools for the organisation and analysis of vast and heterogeneous data. Most of the activity experienced in social networks today contains audio excerpts, either from video files or actual audio clips. Therefore, the analysis of the audio features present in such content can contribute with relevant information for managing the data and ultimately provide a better experience to the end-user.

We propose a method that uses audio features to organise and determine the quality of user-generated audio content crawled from social media websites. In particular, we focus on data related to concert clips. The existence of several recordings of a given event, creates an abundant and redundant pool of recordings. As such, musical shows represent a very good use case for the presented work. The proposed method shall act as a way to better understand and deal with extensive datasets of audio files. The inference of the quality of each file within a given group of files has the ultimate goal of promoting a better user experience, since only the high-quality clips should be shown to the end-user.

The proposed method detects the overlapping sections between different audio clips to organise and group (*i.e.* cluster) the data. It then infers the audio quality of the samples directly from the features used to perform the clustering. The method uses an audio fingerprinting algorithm to this end.

Audio fingerprinting algorithms have traditionally been used for music recognition as made famous by Shazam, where a query sample is matched against other samples in a database of audio files [1, 2, 7, 9]. Here we use this technique for a different purpose, more specifically to synchronise different samples and use the synchronisation information to perform their clustering and infer their quality. In fact, other authors have shown that audio fingerprinting can be used to perform the synchronisation between different samples from the same event that are not time aligned [3, 5, 6, 8].

While Kennedy and Naaman have also used audio fingerprinting to this end [6], we propose two important improvements: (1) our clustering phase includes a filtering approach to avoid false positives, and (2) the proposed technique to infer the samples quality uses information from the audio fingerprinting algorithm that was not used before. Consequently, the analysis of the ranking of the samples in terms of quality achieves better results with the proposed method than with previous methods, with the assumption that professional edited recordings should have higher quality scores than user-generated recordings.

II. The data organisation method

The proposed method can be used to organise multiple concert recordings, which here we call samples. There may be several samples from the same music. More specifically, the method focuses on the grouping of the audio samples, based on them having a common segment of audio, and on their relative quality inference. Since these samples are generated from user recordings, some challenges need to be tackled such as those related to the audio recording qualities inherent to each device. Moreover, it is very unlikely that any two recordings are time synchronised and have the same duration.

In practical terms, the information retrieved by our method can be used to organise and aid with the choice of which overlapped recordings to use at a given time based on their quality (figure 1). Several steps must be followed to perform the grouping and quality analysis of the different audio samples.

To promote a better comparison between the different samples, their (1) synchronisation is required (for recordings of the same song or event). The synchronisation results will serve as the basis to perform the (2) grouping of the samples: recordings of the same song will be clustered together. The clustering information will ultimately help in deriving the

[2]Image from: COGNITUS, http://cognitus-h2020.eu/index.php/2017/01/06/two-open-access-datasets-with-user-generated-audio-recordings/

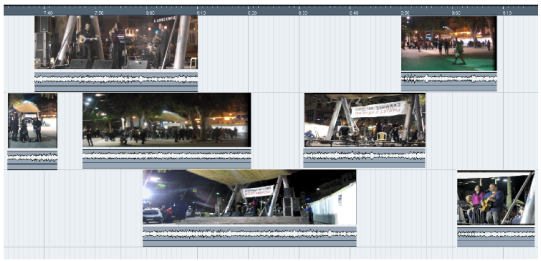

Fig. 1. Time-aligned audio clips[2]. Information about the quality of the clips is very useful to choose which clip should be played.

Fig. 2. Diagram of the different proposed steps.

(3) quality of each sample within a cluster. Evaluation and analysis of the attributed cluster and quality score of each sound clip must be also performed to validate the proposed system. Figure 2 illustrates how the different steps are executed in chronological order. These steps are described in detail in sections II-A, II-B, and II-C, with further validation being presented in section III.

A. Audio Synchronisation

The synchronisation of the samples is an essential step to perform the audio clustering, since its information is used to group different samples with common overlapping audio segments. Moreover, such information is also used to derive the quality of the different samples inside a given cluster.

Fingerprinting techniques' resistance to noise is particularly relevant when dealing with low quality music recordings. This characteristic is ideal for our method since it enables to synchronise noisy samples against possibly less noisy samples in the database, while only needing possibly short length common segments to synchronise different clips.

Fingerprint generation over any kind of data is an efficient mechanism to characterise possibly large data with a small representation. More explicitly, as an alternative to representations that involve large amounts of data, fingerprints are compact representations of the data that can be used for purposes that do not require dealing with all the intrinsic details of the data. This technique promotes a fast way of comparing the quality of two entities by trying to diminish the need to compare irrelevant features.

There are several steps commonly inherent to any audio fingerprint technique: first, an extraction of some of the features present in the audio clip is done; second, those features, or a combination of them, are used to generate a fingerprint that will characterise the audio clip; then, searching processes are applied to access the fingerprints of the other clips in the database; and finally matching mechanisms compare pairs of fingerprints and generate matches.

While other fingerprinting algorithms are compatible with our method, we used Cotton and Ellis fingerprinting algorithm Python implementation[3] based on the methodologies proposed by Wang [4, 9]. The fingerprint generation proposed in this approach is landmark-based (*i.e.* fingerprints are composed by several landmarks). A landmark is created by the analysis of frequency peaks with high energy, since these high spectral energy characteristics of the songs are likely to be resistant to noise and distortion [4]. A landmark is a pair of two peaks, and contains information about each peak frequency, the time at which the first peak occurred, and the time offset between them.

Two samples are considered to match if they have more than t_l common landmarks. Since false matches are unlikely (i.e. have a low probability of occurrence but still greater than 0), this threshold can have a small value (such as $t_l = 5$, being this the default value in the audio fingerprinting algorithm used). For each sample s in a database of previously added samples, the audio fingerprinting phase finds the **matching list** for sample s, that is, all samples that match s. This is done by taking into account the number of common landmarks between query sample s and each of the other samples in the database. The common landmarks are called **matching landmarks**.

B. Audio Clustering

Since we are dealing with several concert recordings as the context of our problem, events can be characterised as the different songs played in the different concerts. Thus, the goal of the clustering phase is to group all recordings of a given concert's song in the same cluster by the analysis of common audio segments.

1) Audio samples grouping: As proposed by Kennedy and Naaman, the results from the audio fingerprinting algorithm can be used to cluster the samples in events [6]. The matched samples have some matching landmarks, which is an indication that, potentially, the samples have a common excerpt and thus are recordings of the same song. The clustering phase uses this information to cluster together the samples that are matched in the audio fingerprinting phase. Therefore, we consider all database matches retrieved by the audio fingerprinting algorithm (*i.e.* all samples $s_i, \ldots, s_j$ matching a given query sample s).

To find the clusters, we represent the matches between different samples with a graph, G [6]. Each sample in the database is represented by a vertex in G. The edges represent the matches between samples. In other words, if sample s_1 matches sample s_2, then there is an edge between vertex s_1 and vertex s_2[4]. The edge weight is the offset (in seconds) between the two samples. The whole graph can have several components. Each component of G corresponds to a different cluster. If there is a path between two vertices, then the corresponding samples are in the same cluster. Isolated vertices

[3]https://github.com/dpwe/audfprint

[4]While samples and vertices are different entities, here there is a one-to-one relationship between them. Thus, to simplify the notation, we will use the same name to represent samples and vertices. For instance, s_i can represent sample s_i and vertex s_i. Also, to simplify the explanations in the paper, we may refer to the vertices in G as samples.

represent unmatched samples, for which the algorithm could not find any match in the database.

Even though unlikely, the probability of false positives samples retrieved by the algorithm is greater than zero, leading to the merging clusters that should not be merged. For example, if sample s_1 from song 1 and sample s_2 from song 2 are incorrectly matched, their clusters will be wrongly merged. In order to overcome this drawback, we introduced a filtering stage to the clustering algorithm (section II-B2). This filtering approach aims to optimise the clustering results.

2) Matches filtering: Considering all the matches retrieved by the audio fingerprinting algorithm would be ideal if false positives did not occur. These false positives can be **sample-level** or **landmark-level**. The first case happens when samples not referring to recordings of the same song are matched. The latter case happens when several landmarks are matched with different offsets over just two samples. For example, samples s_1 and s_2 from the same song have l_1 matching landmarks with offset o_1, l_2 matching landmarks with offset o_2, etc. It is important to notice that only one of the retrieve offsets may be correct, since two samples can only have one offset. Thus, all the other offsets shall be considered false positives.

Landmark-level false positives are easily detected by the repetition of a sample in a query sample's matching list (output from the audio fingerprinting phase). To tackle this problem, only the match with higher number of matching landmarks is considered while any other sample repetitions in the list are eliminated. In the example above, the match considered is the match with offset o_i such that $l_i = max(l_1, ..., l_k)$.

To handle sample-level false positives, it is important to understand in which context they appear because, even though unlikely, their probability is greater than zero. The analysis of such cases, showed that false positives have a lower number of matching landmarks than the true positives in the same matching list. This also applies when comparing the percentage, p, of matched landmarks in the overall number of landmarks of false and true positives. That is, $\forall_{s_t,s_f}\ p_t > p_f$, where s_t is a true positive and s_f is a false positive, p_t and p_f are the percentages of matching landmarks of samples s_t and s_f, respectively, and p is defined as follows

$$p_i = \frac{l_i}{t_i}\,, \tag{1}$$

where l_i is the number of matching landmarks between sample s_i and query sample s, and t_i is the total number of landmarks of sample s_i. It was also observed that when we consider the percentage of matching landmarks in decreasing order, the slope that leads to a false positive (*i.e.* to p_f) is steeper. Following this analysis, an appropriate filtering approach would be:

> For each sample s in the database, consider all the samples s_i that match s as retrieved from the audio fingerprinting phase. Let us assume we have n such samples.
>
> 1. For all those matching samples, consider the percentage of matching landmarks in decreasing order: $(p_1, p_2, \ldots, p_n)$, where $p_1 \geqslant p_2 \geqslant \ldots \geqslant p_n$.
> 2. Analyse the derivative on all consecutive pairs of points $(p_1, p_2, \ldots, p_n)$ in the graph of the percentage of matching landmarks.
> 3. Finally, consider as matches all the samples s_1 to s_j up to a point where the derivative of this graph is higher than a certain value. In other words, stop considering matches as soon as the percentage of matching landmarks significantly drops.

In order to achieve this, one can use a threshold, t_d, and the graph's slope: $\Delta_i = p_{i+1} - p_i$. If $\Delta_j \leqslant t_d$ and $\Delta_i > t_d$ for $i \in \{1, \ldots, j-1\}$, then the algorithm considers that only the samples up to sample s_j are matches to s. That is, only samples $s_1, s_2, \ldots, s_j$ are considered as matches. All remaining samples $s_{j+1}, \ldots, s_n$ are considered as false positives. Our algorithm is using $t_d = -0.07$, as a result of fine-tuning this parameter to exclude all false positives in our dataset described in sub-section III-A.

This would be a reasonable approach to follow if the difference between the percentage of matching landmarks between consecutive true positives would never decrease significantly. Yet, due to the variety in quality and duration of user-generated content, such situation can occur and would cause a high number of true positives to be discarded. Thus, the filtering approach has to take into consideration more parameters to better choose when to filter the returned matches. Since the probability of finding false matches is low and so is their percentage of matched landmarks, a sample should only be considered a false positive if its percentage of matched landmarks is lower than the average for all the retrieved matching samples. Thus, the algorithm's step 3 can be changed to:

> 3. Consider as matches (1) all the samples s_1 to s_k such that $\forall_{1 \leqslant i \leqslant k}\ p_i \geqslant avg(p1, p_2, \ldots, p_n)$ and (2) all the samples s_{k+1} to s_j such that $\Delta_j \leqslant t_d$ and $\forall_{k+1 \leqslant i \leqslant j-1}\ \Delta_i > t_d$.

For an easier analysis of the filtering process, figure 3 displays an example of a matching list with 8 samples in form of a graph with the distribution of the percentage of matching landmarks over the total number of landmarks of each matched sample. In this example, there is only one false positive, which is sample 8. Therefore, ideally this should be the only discarded sample. As observed, the proposed approach achieves this by considering the low slope point (sample 7) under the average of the percentage of landmarks (marked with the dashed line) as the last accepted match, and discarding all samples after it (*i.e.* sample 8). While there are other low gradient points (marked with a diamond) and some have slopes lower than t_d, the samples at these positions are not discarded because they are above the average line.

C. Audio Quality Inference

The output of the clustering phase consists of a set of clusters, represented by graph G, that organises the data according to events (samples with a common segment of audio

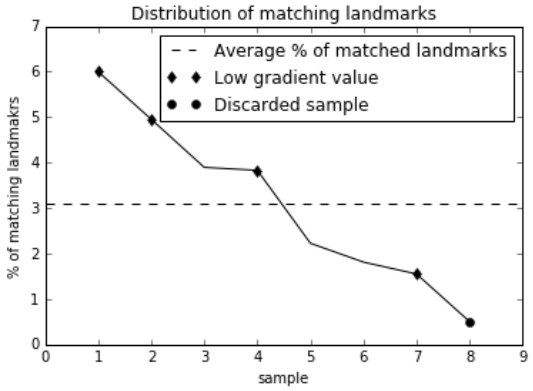

Fig. 3. Distribution of matching landmarks from the matching samples. The only false positive in the matching list (sample 8) is successfully discarded by our approach.

	#Instances	%Instances	#Filtered	%Filtered
Total matches	620	100.00%	70	11.29%
False positives (landmark-level)	5	0.81%	5	100.00%
False positives (sample-level)	4	4.40%	4	100.00%
False negatives (landmark-level)	-	-	65	10.48%

TABLE I
FILTERING BENCHMARKS.

	# Clusters	# Unmatched samples
Ground-truth	10	0
All matches (no filtering)	6	2
All matches (filtering)	10	2

TABLE II
CLUSTERING BENCHMARKS.

belong to the same cluster). The vertices within a component of G may have a different number of neighbours as not all pairs of vertices within a component are adjacent. For instance, let s_1, s_2 and s_3 be vertices in G. If s_1 and s_2 are adjacent, and s_2 and s_3 are adjacent, all three vertices belong to the same component of G (same cluster) but this does not mean that there is an edge between s_1 and s_3: s_1 and s_2 can have a common excerpt, s_2 and s_3 can have another common excerpt, and s_1 and s_3 may lack common excerpts.

Kennedy and Naaman propose to derive the quality of samples by the direct analysis of G, where a sample's quality is proportional to the number of neighbours of that sample's vertex, that is, the number of adjacent vertices [6]. In the example above, s_1 and s_2 are neighbours, s_2 and s_3 are neighbours but s_1 and s_3 are not neighbours. Samples with more neighbours, that is, samples that got matched the most by the fingerprinting algorithm, are considered as those with better quality. This is supported by the idea that any two low-quality samples are less likely to match each other than when at least one of the samples has good quality.

Even though this is a suitable approach, it only considers the number of matches of a given sample, ignoring how the sample ranks in terms of matching landmarks in the overall list of matched samples. A sample with many matching landmarks is likely to have better quality than a sample with less matching landmarks. Thus, our proposed solution considers that the score of a sample, s_i, depends on the total number of matching landmarks of that given sample against all the other samples in the database. Let $s_1, s_2, \ldots, s_N$ be the samples in the database. Assume s_i (for $1 \leqslant i \leqslant N$) has l_{1i} matching landmarks to sample s_1, l_{2i} matching landmarks to sample s_2, etc. Then the score of s_i is calculated as follows:

$$score(s_i) = \sum_{j=1}^{N} l_{ji}\,. \tag{2}$$

III. EVALUATION AND ANALYSIS

A. Test setup

To test the proposed algorithm, a realistic testbed was designed by manually collecting from YouTube several concert clips of various songs captured with different devices. For each song we collected one professional recording and several user recordings of the same song in the same concert.

The dataset[5] consists of 91 samples of 10 different songs, all part of different editions of the Reading Festival. Apart from the 10 professional recorded samples, all samples were recorded by users, which means different recording devices with different qualities apply. Moreover, the number of recordings retrieved for each song, as well as their respective lengths, differ between each cluster. Efforts were made to have recordings with lengths across a large scope in order to promote a more diverse dataset. Nonetheless most of the recordings crawled were within the 5 minutes range.

B. Results

Using the dataset described in III-A, we validated the proposed method, namely the filtering and clustering phases. By analysis of table I, we can see that our filtering method succeeded in filtering all false positives retrieved by the audio fingerprinting algorithm – 1 out of the 5 landmark-level false positives were discarded by eliminating repetitions whilst the rest were detected by the analysis of the derivative values.

It is important to notice that even though there were occurrences of false negatives (*i.e.* 10.48 % of the overall retrieved landmarks), this does not affect the clustering results, as presented in table II. The number of retrieved clusters when considering all matches and performing filtering is correct (that is, it is the same as for the ground-truth), whereas without the filtering process clusters were wrongly merged, making the system only retrieve 6 clusters instead of 10.

To evaluate the quality inference of our solution, we analysed the score of the professional recording relative to the scores of the other samples in the cluster using our method

[5]The dataset is available at http://novasearch.org/datasets/

Cluster	# Samples	Proposed method	K.M. method
1	8	5th	3th-5th
2	8	4th	1st-4th
3	6	5th	3th-5th
4	15	3th	3th-4th
5	11	1st	2nd-4th
6	5	4th	1st-5th
7	8	4th	1st-4th
8	10	1st	2nd-4th
9	11	5th	3th-5th
10	7	1st	1st-4th

TABLE III
QUALITY INFERENCE. POSITION OF THE PROFESSIONAL RECORDING IN THE RANKING LIST WITH OUR METHOD (THIRD COLUMN) AND KENNEDY AND NAAMAN'S METHOD (FOURTH COLUMN).

and Kennedy and Naaman's method (K.M. method). This evaluation assumes that the professional recordings represents one of the highest quality recordings in any cluster, and, therefore, should be placed at the top of the ranking list. Table III shows the comparison between the two methods. The second column indicates the number of samples in the cluster, the third and fourth columns show the position of the professional recording in the ranking list with our method and the K.M. method, respectively. The first position in the ranking list corresponds to the highest score.

It is important to note that there may be ambiguity on the quality scores of the K.M. method, because several samples can have the same score. This is observed in table III by the range of positions of the professional sample. Such ambiguity does not appear with our approach, which is essential for aiding end users on the decision of which samples to use. The table also shows that the score of the professional track with our method is always the same or better than with the K.M method, as it is either included within the range given by the K.M method or it is even ranked better (clusters 5 and 8).

IV. CONCLUSION

For a better comprehension and management of large datasets of audio files, we propose a method that clusters the data according to events and infers the relative quality of audio files. The method uses audio fingerprints to determine the clusters and quality of the samples. While our method uses some of the methodology proposed by Kennedy and Naaman [6], we propose relevant improvements to their methodology.

A major improvement offered by our method consists of avoiding false positives. On top of eliminating repetitions, our method uses a filtering approach that looks at the distribution of the percentage of matching landmarks (sets of fingerprints common to two samples) and uses the derivative of this distribution to detect false positives.

In terms of quality inference, by looking at the number of matching landmarks, instead of only checking if there was a match between two samples, the proposed method succeeds in classifying better the professional recordings in some cases to when compared to the method proposed in [6]. Furthermore, it also avoids ambiguity quality scores using more detailed information than the previous method.

The results show that the proposed filtering technique successfully avoids false negatives. Also, the quality inference results from our method show improvements over previous methods.

REFERENCES

[1] AcoustID. *Chromaprint — AcoustID*. URL: https://acoustid.org/chromaprint.

[2] ACRCloud. *Automatic Content Recognition Cloud Services*. URL: https://www.acrcloud.com.

[3] R. Bardeli, J. Schwenninger, and D. Stein. "Audio fingerprinting for media synchronisation and duplicate detection". In: *Proc. MediaSync* (2012), pp. 1–4.

[4] V. Cotton and D. Ellis. "Audio fingerprinting to identify multiple videos of an event." In: *ICASSP*. IEEE, 2010, pp. 2386–2389.

[5] N. Duong and F. Thudor. "Movie synchronization by audio landmark matching". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2013.

[6] L. Kennedy and M. Naaman. "Less Talk, More Rock: Automated Organization of Community-contributed Collections of Concert Videos". In: *Proceedings of the 18th International Conference on World Wide Web*. WWW '09. ACM, 2009, pp. 311–320.

[7] Midomi. *Search for Music Using Your Voice*. URL: http://www.midomi.com/index.php.

[8] P. Shrestha, M. Barbieri, and H. Weda. "Synchronization of Multi-camera Video Recordings Based on Audio". In: *Proceedings of the 15th ACM International Conference on Multimedia*. MM '07. ACM, 2007, pp. 545–548.

[9] A. L. Wang. "An industrial-strength audio search algorithm". In: *Proceedings of the 4 th International Conference on Music Information Retrieval*. 2003.

AUTOMATIC ORGANISATION, SEGMENTATION, AND FILTERING OF USER-GENERATED AUDIO CONTENT

Gonçalo Mordido, João Magalhães, Sofia Cavaco

NOVA LINCS, Departamento de Informática
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
2829-516 Caparica, Portugal

goncalomordido@gmail.com, {jm.magalhaes, scavaco}@fct.unl.pt

ABSTRACT

Using solely the information retrieved by audio fingerprinting techniques, we propose methods to treat a possibly large dataset of user-generated audio content, that (1) enable the grouping of several audio files that contain a common audio excerpt (*i.e.* are relative to the same event), and (2) give information about how those files are correlated in terms of time and quality inside each event. Furthermore, we use supervised learning to detect incorrect matches that may arise from the audio fingerprinting algorithm itself, whilst ensuring our model learns with previous predictions. All the presented methods were further validated by user-generated recordings of several different concerts manually crawled from YouTube.

Index Terms— audio fingerprinting, user-generated content, audio synchronisation, supervised learning

1. INTRODUCTION

Given the abundance and ubiquity of video-oriented content (and, consequently, audio content) experienced in most social networks nowadays, it is important to understand such large amount of information in a meaningful way. One important step to achieve such understanding is to group the content in several clusters based on similarity, which in the context of this work is based on events. When we consider several user-generated recordings of different lengths reporting the same event, which is very likely to happen due to the nature of user-generated content, the existence of overlapping sections between two of such recordings means that they should belong to the same cluster/event.

Audio fingerprinting has been primarily used to detect if a given query song matches other songs in a preexisting database [1, 2, 6, 8]. Nonetheless, this algorithm retrieves very valuable information, that can be used for several other purposes. Here, we propose to use it to perform the organisation (clustering), segmentation and alignment, of audio recordings of music events.

978-1-5090-3649-3/17/$31.00

The main contributions of this paper are then the organisation of a large dataset of audio recordings into the different events they portrait (section 2), with additional information regarding how each event's recordings are distributed over time (section 3), and the detection and filtering of incorrect matches possibly retrieved from the audio fingerprinting algorithm using a supervised learning approach (section 4), whilst ensuring our model learns with previous predictions (section 5).

Moreover, finding correlations between the content inside each cluster can also be very beneficial to achieve a better comprehension of the data. In this work we propose to align all event's song clips over time and we further use a quality inference technique already presented in previous work to order them in terms of their relative quality [7].

2. DATA ORGANISATION

Considering the abundant and ubiquitous nature of user-generated content, it is very likely to deal with a database of several different events (*e.g.* audio recordings of several concerts), in which each event has several recordings reporting it (*e.g.* relative to a certain concert song). Our goal is then to gather all song clips of a given event into the same cluster. Our technique to group clips of a given event is based on them having common excerpts of audio. Given the likely noisy and time sparse nature of user-generated content (*i.e.* the different recordings capture different parts of the event, with possible overlaps), we need to use a technique that is resistant to noise and at the same time can identify overlapping excerpts in music recordings from the same event.

Audio fingerprinting enables synchronising a query song s_q against several other audio clips present in a formerly created database, whilst being relatively resistant to noise. Note that when we refer to a query song, we do not mean that we are dealing with the whole song, instead, we are referring to a portion of the whole song that has been recorded in an audio clip. Section 2.1 explains in more detail why using audio fingerprinting to characterise and compare the different audio

files (similarly to what already proposed in our previous work [7]) is appropriate when dealing with possibly very noisy audio files, which is likely to be experienced in the context of our problem.

Once the audio fingerprints of the different audio files are compared and possibly matched, we use this information to identify overlapping excerpts and cluster our data into the different events. Section 2.2 takes a deeper look on how the grouping of the different recorded clips is indeed achieved and internally represented.

2.1. Audio Fingerprinting

The first step of our algorithm is to characterise the data with audio fingerprints. Using a fingerprint to characterise each recorded clip enables to efficiently represent and compare different clips, which is essential considering the vast occurrence of user-generated data experienced nowadays. Since the generation of this fingerprint involves the direct usage, or a combination of features from the audio signal, it is important to pick the features that are the most representative and, to some extent, invariant to distortion. Similarities between fingerprints of different song clips lead to a match of the clips.

Our algorithm uses Cotton and Ellis' landmark-based audio fingerprinting algorithm[1], which is based in the well-known approach formerly proposed by Wang [3, 8]. A fingerprint is composed of several landmarks, which in turn are generated through the analysis of two frequency peaks with high energy in a small period of time. More specifically, a landmark is a pair of two peaks, and contains information about each peak frequency, the time at which the first peak occurred, and the time offset between them.

Given a query song s_q, our algorithm uses the audio fingerprints information to match it against the song clips s in the database [7]. Since each audio clip's fingerprints are a list of landmarks, our algorithm considers that two song clips, s_q and s, contain the same audio excerpt if more than a certain number of landmarks are equal in both of their fingerprints; the threshold used is normally a small value (*e.g.* 5) since wrong matches are unlikely. To discover the time offset between the two clips we simply need to analyse the time difference between the timestamp of the equal landmarks in s_q and s.

2.2. Audio Clustering

The second step of our algorithm is to organise the data into clusters, such that the clips from the same event (*i.e.* from the same whole music) are in the same cluster. This is achieved using the information retrieved from the fingerprinting stage. Moreover, since the different recordings will very likely be in different ranges of quality, from extremely noisy to clean recordings, audio fingerprinting permits the synchronisation of the low-quality recordings against better quality recordings in the database, conceivably in common audio portions that might not be too affected by noise in the low-quality recording.

In order to organise the audio clips in the database, we match each clip to all the other ones present in the database, ensuring all clips are tested against one another. In other words, for each clip in the database we consider it as a query song and use the fingerprints information to match it against all other clips in the database. Since there may be multiple clips for the same event (*i.e.*, whole song) each query song will likely have several song matches, that together compose the **matching list** of the query song.

The fingerprints information is used to build a graph, $G = (V, E)$, with the nodes, V, representing all song clips in our database and each edge in E representing a match between two clips. Since each clip is represented by a node, we will use the same name (s) to refer to song clip s and the node that represents that clip. Moreover, each edge is assigned a weight that consists of the offset (in seconds) between the two connected clips. In other words, if edge $(s_1, s_2, o_{12}) \in E$, then there is a match between clips s_1 and s_2 with offset o_{12}. Isolated nodes in the graph represent clips that have an empty matching list and that are not present in any of the other clips' matching lists. Even though the analysis of the weights of the paths is not necessary to performing the clustering, it will be essential to perform the audio segmentation presented in section 3.

The basis to detect and distribute the clips to the different clusters resides in the notion that if there is a path between two clips, then they should belong to the same cluster. This is an adaptation of Kennedy and Naaman's algorithm, that uses this graph-based representation to detect different episodes inside a given event [5].

3. AUDIO SEGMENTATION

Analysing how the different clips of each event are scattered over each event's timeline is of extreme importance to better manage the different audio files. Therefore this section focuses on finding the time intervals (*i.e.* segments) in which the different clips are distributed inside each cluster. An important aspect of this synchronisation task is that it only requires information already obtained from the audio fingerprinting algorithm that was used to perform the clustering described in section 2.

3.1. Audio Synchronisation

The offsets returned by the audio fingerprinting algorithm were further used to perform the alignment of the audio clips. This task uses the graph-based representation of our clips, G described in section 2.2. As mentioned above, the weights

[1]This landmark-based audio fingerprinting algorithm is available in https://github.com/dpwe/audfprint.

of the edges are the offset (returned by the audio fingerprinting) between two clips. Following the paths in G, we can derive the offsets between any two clips in the same cluster by adding the weights in the path (*i.e.,* by calculating the cost of the path). It is important to notice that this only works because if there is a positive edge in the graph connecting two nodes, there is also a negative edge in the opposite direction. We can then represent the offset o_{ij} between any two nodes s_i and s_j that are in the same cluster, as $o_{ij} = cost(G, s_i, s_j)$.

The actual way the synchronisation of all clips inside a cluster is made is by electing a **representative song clip** and by getting the offset of all the other clips relative to this one (that is, o_{ir} for every song clip s_i in the cluster). Note that the representative clip can be any of the cluster's clips, since all clips of a given event (cluster) are connected in the graph. After all offsets are obtained, if the representative clip is not the recording that has the earliest starting timestamp, the offset values are updated according to the clip with the earliest timestamp (*i.e.* the clip that starts first in the event's timeline).

We can define the earliest starting song clip s_e as the clip with the minimum distance to the reference clip s_r:

$$\forall_{s_i \in V} \; o_{er} \leq o_{ir} \, .$$

This minimum distance can either be 0, if the representative song is indeed the earliest starting clip (since $o_{rr} = 0$), or a negative number, if s_e starts before song s_r. Afterwards, we calculate all offsets o_{ie}.These can be obtained by adding the value of o_{er} to the previously calculated offsets o_{ir}:

$$\forall_{s_i \in V} \; o_{ie} = o_{ir} + o_{er}$$

Using this approach, all offsets are greater or equal than 0 and correctly aligned in terms of their starting point along the event's timeline, since all offsets are now relative to the earliest starting clip.

3.2. Time-based Segmentation

By having the overall offsets of all clips of a given cluster, together with the duration of each clip, one has the knowledge of which clips exist in a given moment of time. Thus, we can organise an event with **segments**, such that segments coincide with the time interval of overlapping clips.

The overall event's timeline will be segmented into several non-overlapping segments. Given all offsets o_{ie} (for all s_i in the cluster), a new segment from time t_{start} to time t_{end} is created when one of the following situations occurs: (1) A new song clip s_i starts at time t_{start} ($o_{ie} = t_{start}$). (2) A clip s_i with duration $d(s_i)$ ends at time t_{end} (that is, $o_{ie} + d(s_i) = t_{end}$). As a consequence, whenever a new segment starts at t_{start}, there is a segment ending at $t_{start-1}$, except when $t_{start} = 0$ meaning that it is the first segment of that event.

The song clips can then be cut according to the timestamps of each of the segments they are part of. For instance, if song s_1 belongs to segment A and B, then the song is cut into song s_{1A} and s_{1B} (s_1 is equal to the concatenation of s_{1A} and s_{1B}).

This information is encapsulated in a tuple that represents a segment. The tuple contains an initial and final timestamp, and all clips that overlap between that period of time, $(t_{start}, t_{end}, s_{1A}, s_{2A}, \ldots)$. Each cluster, or event, is then composed by several segments, that give information on which clips are available in the different time intervals and therefore at any moment of time in the event's timeline.

3.3. Quality Inference

In previous work we proposed a method to infer the quality of each song clip relative to all the other clips inside a given cluster by analysing the sum of each clip's number of matching landmarks against the rest of the clips in the database [7]. This method can be further used to infer the quality of the clips inside each segment by matching them using the audio fingerprinting algorithm (that is, the algorithm is called once more but with the clips within the segment and not all clips in the database).

However, given the possible small time length of the segments, together with the possible small number of clips within each segment, matches are less likely to happen. Thus, increasing the number of landmarks by increasing the number of landmarks/sec performed by the algorithm for each clip, generates a higher number of matching landmarks between different song clips and therefore increases the likeliness of matches to occur.

Since the clusters were formed based on song clips with common excerpts, and after the filtering of false matches that will be presented in section 4, we can eliminate the matching landmarks threshold leading a match to be declared even with only 1 matching landmark between two clips. Since all clips inside a segment are time-aligned, the expected offset returned by the algorithm should be 0 seconds, meaning all the other matching landmarks with different offsets can be discarded and not considered for the clip's quality score.

This quality inference step enables ultimately for song clips to be ordered based on their relative quality inside each segment. Thus, on top of having information to which clips are available at a given time in the overall event's timeline, we now know how the different song clips inside the segment relate in terms of their relative quality.

4. FILTERING METHOD

Even though unlikely, the probability of a false match between two clips from the audio fingerprinting algorithm is still greater than 0. We propose a method to filter out such false matches from the clusters.

In previous work, we proposed a filtering approach based on the analysis of significant drops on the derivatives of the

percentage of matching landmarks between the query and matched song relative to the overall number of the matched clip's landmarks [7]. Here, we present an alternative method that uses machine learning to detect such false matches.

4.1. Feature Selection

Our samples, or feature vectors, are derived from the fingerprinting algorithm's output, and every song is represented by several samples. Each sample corresponds to a match returned by the fingerprinting algorithm.

Given a query song s_q, the fingerprinting algorithm returns the following for every song s_i in the database: (1) the number of landmarks, $\#L_{s_q}$, of the query song s_q, (2) the offset between s_q and s_i, that is o_{qi}, (3) the number of matching landmarks with offset o_{qi}, which we call $\#ML_{o_{qi}}$, and (4) the number of total matching landmarks in all offsets, $\#TML$. Note that when a song is added to the database, the number of landmarks computed for that song is also retrieved from the algorithm, hence the number of landmarks of all songs are known. Thus, (5) the number of landmarks , $\#L_{s_i}$, of song s_i is also known. Since the actual value of the offset does not directly influence if a match is correct or incorrect, it is not considered to enter the feature space. However, all the other referred features might be a good indicator of a false match.

The set of available and possibly relevant features, for each pair (s_q, s_i), is then the following:
$F = \{\#ML_{o_{qi}}, \#TML, \#L_{s_q}, \#L_{s_i}\}$. We tested our models with several subsets of F, more specifically:

- $\{\#ML_{o_{qi}}, \#TML\}$
- $\{\#ML_{o_{qi}}, \#TML, \#L_{s_q}\}$
- $\{\#ML_{o_{qi}}, \#L_{s_q}, \#L_{s_i}\}$
- $\{\#ML_{o_{qi}}, \#TML, \#L_{s_q}, \#L_{s_i}\}$

Each one of our classifiers was trained with these features subsets to access which combination generates the best model.

4.2. Training Data

Since the goal of our models is to predict whether a sample is a false match or a true match, there are only two classes: 0 and 1, respectively. **False matches** are incorrect matches. These can be **wrong matches**, if the two matched songs do not have any common audio excerpt, or **repetition matches**, if they have indeed a common excerpt but the assigned offset is not correct. The latter case can be easily detected as it happens when a song s_i appears in the matching list of a query s_q several times with different offsets, described as repetitions. In this case, the match offset (o_{qi}) with the highest number of matching landmarks is considered a true match (*i.e.,* assigned to class 1), whilst all other match offsets (o'_{qi}, o''_{qi}, o'''_{qi}, ...) are considered false matches (*i.e.,* assigned to class 0).

A dataset of 198 audio recording files, retrieved from 23 different concert songs from YouTube, was used as the database of the audio fingerprinting algorithm, which corresponded to an average of 8.6 different recordings per concert song (*i.e.* event). This database generated 3098 matches, which were used to train, validate, and test our models. From these, there were 1071 true matches (class 1) and 2027 false matches (class 0) from which 2021 were repetition matches and 6 were wrong matches. Note that we balanced the training set every time a new model was trained (*i.e.* the number of samples of class 0 was equal to the number of samples of class 1).

4.3. Model Estimation

We used three different methods to solve this classification problem: logistic regression, k-nearest neighbours (kNN), and support vector machines (SVM). The purpose of using different classifiers is to have a broader way of comparison on how the different features used influence the outcome of the overall predictions of the different methods.

Apart from trying different feature vectors, we also varied the classifiers parameters. For logistic regression, we doubled the value of the regularisation parameter c during 20 iterations (with its initial value being set to 1.0). We tried all odd numbers between 1 and 39 for the number of neighbours k in the kNN classifier. Regarding the SVM classifier, we used the RBF kernel and the optimal values for c and γ were obtained by executing an exhaustive search over all possible combinations of a subset of possible values for each parameter. For this we followed the methodology of using exponentially growing sequences [4]. More specifically varying c to the following values 2^{-5}, 2^{-3}, ... , 2^{15}, 2^{17} and γ to 2^{-15}, 2^{-13}, ... , 2^{3}, 2^{5}. This searching process is often described as Grid-search, and it returns the best value of each parameter of a given model (*i.e.* the hypothesis that achieves the highest accuracy).

We used double cross-validation to retrieve the model with lowest validation error for each classifier (varying the parameters as explained above): we start by performing **leave-one-song-out** cross-validation, in which every song in the training set except one are used to train the model with a k-fold cross-validation, with $k = 10$, whilst the left-out song is used to test the model; this process is then repeated until all songs have been left-out and repeated in every combination of possible parameters assigned for each classifier. The training and validation error of each model is the average of the error occurred in all the leave-one-song-out iterations, with the accuracy of the model being tested on the overall predictions of all left-out songs' samples. Following these steps for all designated ranges of possible values for the different classifiers' parameters, we assign the model with lowest validation error in the 10-fold validation for each classifier as the most suitable model.

4.4. Prediction Results

The accuracy results for each classifier is shown in figure 1. The SVM showed better results across the different feature combinations (98.23%, 97.22%, 96.12%, and 97.68%, respectively) but was closely followed by the other classifiers with the exception of logistic regression with $(\#ML_{o_{qi}}, \#L_{s_q}, \#L_{s_i})$, that achieved a considerably lower accuracy (82.07%).

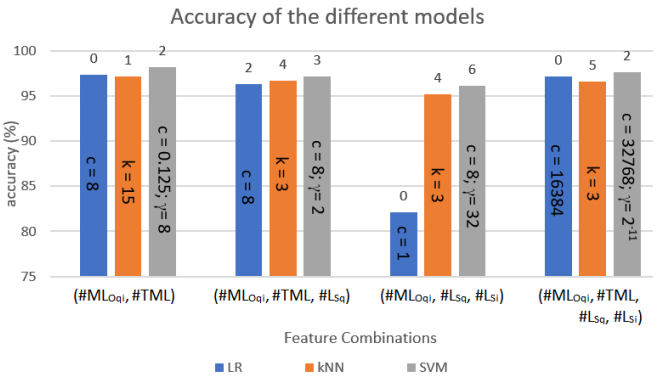

Fig. 1. Accuracy of the best models (*i.e.* with lowest validation error) of each classifier for the different combination of features. The parameter values are described inside each bar. The numbers placed on top of each bar represent the number of false positives for each model.

Despite their high accuracy, models that incorrectly classify wrong matches (false positives) have songs of different events assigned to the same cluster, leading ultimately to the merge of clusters of different events. Therefore, instead of simply choosing the model with lowest validation error for each classifier, we can discard all models that wrongly classified the wrong matches and choose the lower validation error model of the remaining. Figure 2 shows the updated classifiers results adding this constraint.

Even though the models' accuracy slightly decreased, we managed to find new models for kNN and SVM that satisfy our condition of classifying wrong matches correctly (that is, to class 0), whilst maintaining a high accuracy (97.12% and 97.49%, respectively). The logistic regression models already presented in figure 1 remained intact since they had no incorrect classifications of wrong matches, except when using the feature combination $(\#ML_{o_{qi}}, \#TML, \#L_{s_q})$, with their accuracy of 97.40% for the first presented feature combination, and 97.21% for the latter.

In sum, there is a slight advantage of considering only the models with no incorrect classification of wrong matches since their filtering is crucial in the proposed solution. We managed to achieve high accuracy results with each of the three classifiers. Using logistic regression and kNN with $(\#ML_{o_{qi}}, \#TML)$, that is, the matching landmarks in the right offset and the total number of matching landmarks in all

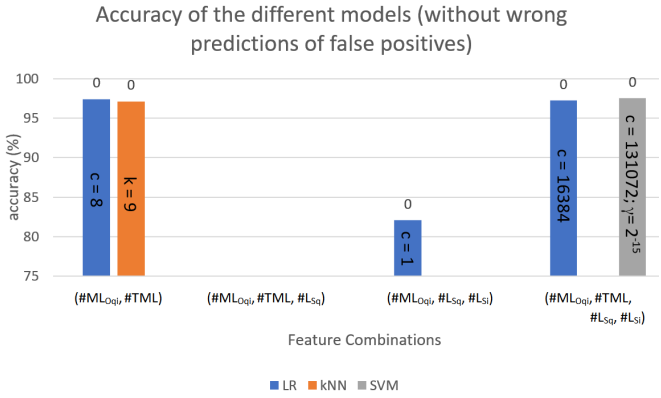

Fig. 2. The models that are missing in the grah incorrectly classified at least one wrong match. New models with different parameter values were found for both kNN and SVM whilst respecting this condition.

detected offsets, as well as using logistic regression and SVM with the 4 feature-combination, would represent practically viable options for the presented filtering approach.

5. LEARNING EXTENSION

The training set can be further expanded by the analysis of the information retrieved from the audio fingerprint algorithm combined with our model predictions. This extension can occur in two stages: during the audio clustering phase (section 2.2), and by the analysis of the matches between the cut samples when performing the audio quality inference inside each segment (section 3.3).

During the audio clustering phase, all repetition matches (repetitions of a given matched song in another's matching list) can be added to the training set: the feature vectors of the repetition matches are assigned to class 0. This is supported by the assumption that since only one offset is possible between two songs, the correct offset is the one that generated more matching landmarks, whilst the others are discarded.

The quality inference stage can serve as a confirmation for some of the samples that were predicted as true matches after the filtering method. Since all matches classified as wrong matches are filtered in the Audio Clustering phase (section 2.2), either by the discarding of the repetitions or by false matches classifications, all the samples of the songs present in the Audio Segmentation phase (section 3) were therefore predicted as true matches by our model (*i.e.* assigned to class 1). Hence, after cutting each song according to the different segments in which it appears, and by matching all cut songs with one another inside a segment to infer the quality, all matches should be assigned to offset 0.0 seconds since all the cut songs are meant to be synchronised in time.

Let us define the function $offset(s, S)$ as returning the set of offset values of the different matches between song s and

each song in set S. Function *count*(A, v) retrieves the number of occurrences of the real number v in set A, and $TM(m)$ assigns sample match m to class 1 in the training set. Then, we can define the following expression for every cluster c:

$$\forall t \in T_c \;\; \forall s \in S_t \;\; \textit{count}(\textit{offset}(s, S_t \backslash \{s\}), 0) = \|S_t \backslash \{s\}\| \Rightarrow \forall m \in M_c : TM(m)$$

where t is a segment in T_c, which in turn is the set of (time) segments of cluster c, and S_t is the set of songs in segment t. M_c is the set of all matches (*i.e.* samples) in that cluster.

To sum up, one can then assume that, if for each cut song inside each cluster's segments there is a match to each of the other cut songs with offset of 0 seconds, then all samples that previously contributed to the formation of that given cluster are considered true matches and added to the training set with their class assigned to 1.

6. CONCLUSION

In this work we propose different methods that manipulate and correlate different user-generated recordings in a possibly large dataset of audio files, contributing ultimately for a better comprehension of the data. The basis of all presented work relied upon the direct analysis of the information retrieved by the matches of the different audio files from the audio fingerprinting algorithm.

Although using audio fingerprinting to organise different audio files with common audio excerpts was initially proposed by Kennedy and Naaman [5] and further extended in our previous work [7], here we introduced a novel filtering approach by using machine learning techniques and achieving optimal filtering results (*i.e.* successfully filtering all wrong matches) whilst also achieving high prediction accuracy in our considerable large test setup (*e.g.* 97.49% using SVM and 4 features). Moreover, we introduce the possibility of extending our learning by increasing the training set in different possible stages, more concretely in the Audio Organisation and Audio Segmentation phases, by the detection of repetitions and by the analysis of the previous predictions.

We additionally proposed Audio Segmentation inside each cluster/event which provides valuable insight on how the different event's audio files are correlated in terms of time. This can be extremely useful since it provides the knowledge of which audio files are available at a given moment of time. Moreover, using a previously proposed audio inference approach [7] with parameter adaptations in the audio fingerprinting algorithm, we also represent how the different audio files relate in terms of their relative audio quality inside each segment of a given cluster.

Acknowledgments

This work was partially funded by the H2020 ICT project COGNITUS with grant agreement No 687605 and by the Portuguese Foundation for Science and Technology under project NOVA-LINCS PEest/UID/CEC/04516/2013.

References

[1] AcoustID. *Chromaprint — AcoustID*. URL: `https://acoustid.org/chromaprint`.

[2] ACRCloud. *Automatic Content Recognition Cloud Services*. URL: `https://www.acrcloud.com`.

[3] V. Cotton and D. Ellis. "Audio fingerprinting to identify multiple videos of an event." In: *ICASSP*. IEEE, 2010, pp. 2386–2389.

[4] C. Hsu, C. Chang, C. Lin, et al. "A practical guide to support vector classification". In: (2003).

[5] L. Kennedy and M. Naaman. "Less Talk, More Rock: Automated Organization of Community-contributed Collections of Concert Videos". In: *Proceedings of the 18th International Conference on World Wide Web*. WWW '09. ACM, 2009, pp. 311–320.

[6] Midomi. *Search for Music Using Your Voice*. URL: `http://www.midomi.com/index.php`.

[7] G. Mordido, J. Magalhães, and S. Cavaco. "Automatic Organisation and Quality Analysis of User-Generated Content with Audio Fingerprinting". In: *2017 25th European Signal Processing Conference (EUSIPCO) (EUSIPCO 2017)*. 2017, pp. 1864–1868.

[8] A. L. Wang. "An industrial-strength audio search algorithm". In: *Proceedings of the 4 th International Conference on Music Information Retrieval*. 2003.

Appendix B

Audio synchronisation output

This appendix exhibits the output file returned by the audio segmentation phase, containing the different segments found for each cluster using the original 10-song dataset.

```
Cluster 1
00:00:0-00:20:0
sounds_cut/1/0.0/song2sample1.mp3 0
00:20:0-00:22:1
sounds_cut/1/20.0/song2sample5.mp3 0
sounds_cut/1/20.0/song2sample1.mp3 0
00:22:1-00:29:3
sounds_cut/1/22.1/song2sample1.mp3 0
sounds_cut/1/22.1/song2sample5.mp3 0
sounds_cut/1/22.1/song2sample7.mp3 0
00:29:3-00:32:0
sounds_cut/1/29.3/song2sample3.mp3 3.0
sounds_cut/1/29.3/song2sample1.mp3 2.0
sounds_cut/1/29.3/song2sample7.mp3 0
sounds_cut/1/29.3/song2sample5.mp3 0
00:32:0-00:38:0
sounds_cut/1/32.0/song2sample5.mp3 3.0
sounds_cut/1/32.0/song2sample1.mp3 2.0
sounds_cut/1/32.0/song2sample7.mp3 0
sounds_cut/1/32.0/song2sample8.mp3 0
sounds_cut/1/32.0/song2sample3.mp3 0
00:38:0-00:48:6
sounds_cut/1/38.0/song2sample8.mp3 16.0
sounds_cut/1/38.0/song2sample3.mp3 13.0
sounds_cut/1/38.0/song2sample5.mp3 9.0
sounds_cut/1/38.0/song2sample7.mp3 9.0
sounds_cut/1/38.0/song2sample1.mp3 8.0
sounds_cut/1/38.0/song2sample4.mp3 6.0
```

```
00:48:6-00:58:0
sounds_cut/1/48.6/song2sample3.mp3 14.0
sounds_cut/1/48.6/song2sample4.mp3 6.0
sounds_cut/1/48.6/song2sample7.mp3 5.0
sounds_cut/1/48.6/song2sample2.mp3 4.0
sounds_cut/1/48.6/song2sample8.mp3 4.0
sounds_cut/1/48.6/song2sample5.mp3 2.0
sounds_cut/1/48.6/song2sample1.mp3 0
00:58:0-02:01:033
sounds_cut/1/58.0/song2sample3.mp3 132.0
sounds_cut/1/58.0/song2sample8.mp3 88.0
sounds_cut/1/58.0/song2sample6.mp3 83.0
sounds_cut/1/58.0/song2sample4.mp3 80.0
sounds_cut/1/58.0/song2sample5.mp3 73.0
sounds_cut/1/58.0/song2sample7.mp3 71.0
sounds_cut/1/58.0/song2sample2.mp3 56.0
sounds_cut/1/58.0/song2sample1.mp3 34.0
02:01:033-02:57:492
sounds_cut/1/121.033/song2sample3.mp3 107.0
sounds_cut/1/121.033/song2sample5.mp3 85.0
sounds_cut/1/121.033/song2sample7.mp3 83.0
sounds_cut/1/121.033/song2sample8.mp3 67.0
sounds_cut/1/121.033/song2sample4.mp3 50.0
sounds_cut/1/121.033/song2sample1.mp3 46.0
sounds_cut/1/121.033/song2sample2.mp3 31.0
02:57:492-03:06:567
sounds_cut/1/177.492/song2sample3.mp3 15.0
sounds_cut/1/177.492/song2sample8.mp3 8.0
sounds_cut/1/177.492/song2sample1.mp3 3.0
sounds_cut/1/177.492/song2sample7.mp3 2.0
sounds_cut/1/177.492/song2sample2.mp3 0
sounds_cut/1/177.492/song2sample4.mp3 0
03:06:567-03:07:376
sounds_cut/1/186.567/song2sample3.mp3 12.0
sounds_cut/1/186.567/song2sample4.mp3 10.0
sounds_cut/1/186.567/song2sample2.mp3 2.0
sounds_cut/1/186.567/song2sample8.mp3 0
sounds_cut/1/186.567/song2sample1.mp3 0
03:07:376-03:08:83
sounds_cut/1/187.376/song2sample3.mp3 9.0
sounds_cut/1/187.376/song2sample2.mp3 0
sounds_cut/1/187.376/song2sample4.mp3 0
sounds_cut/1/187.376/song2sample1.mp3 0
03:08:83-03:12:384
sounds_cut/1/188.83/song2sample1.mp3 0
sounds_cut/1/188.83/song2sample4.mp3 0
sounds_cut/1/188.83/song2sample2.mp3 0
03:12:384-03:15:095
sounds_cut/1/192.384/song2sample2.mp3 0
sounds_cut/1/192.384/song2sample1.mp3 0
```

```
03:15:095-03:30:991
sounds_cut/1/195.095/song2sample1.mp3 0

Cluster 2
00:00:0-00:00:5
sounds_cut/2/0.0/song7sample2.mp3 0
00:00:5-00:13:0
sounds_cut/2/0.5/song7sample3.mp3 0
sounds_cut/2/0.5/song7sample2.mp3 0
00:13:0-00:18:9
sounds_cut/2/13.0/song7sample1.mp3 4.0
sounds_cut/2/13.0/song7sample3.mp3 0
sounds_cut/2/13.0/song7sample2.mp3 0
00:18:9-00:19:7
sounds_cut/2/18.9/song7sample5.mp3 0
sounds_cut/2/18.9/song7sample2.mp3 0
sounds_cut/2/18.9/song7sample1.mp3 0
sounds_cut/2/18.9/song7sample3.mp3 0
00:19:7-00:21:4
sounds_cut/2/19.7/song7sample4.mp3 0
sounds_cut/2/19.7/song7sample1.mp3 0
sounds_cut/2/19.7/song7sample3.mp3 0
sounds_cut/2/19.7/song7sample5.mp3 0
sounds_cut/2/19.7/song7sample2.mp3 0
00:21:4-00:39:5
sounds_cut/2/21.4/song7sample3.mp3 11.0
sounds_cut/2/21.4/song7sample4.mp3 4.0
sounds_cut/2/21.4/song7sample5.mp3 3.0
sounds_cut/2/21.4/song7sample1.mp3 2.0
sounds_cut/2/21.4/song7sample8.mp3 0
sounds_cut/2/21.4/song7sample2.mp3 0
00:39:5-00:41:5
sounds_cut/2/39.5/song7sample5.mp3 0
sounds_cut/2/39.5/song7sample2.mp3 0
sounds_cut/2/39.5/song7sample4.mp3 0
sounds_cut/2/39.5/song7sample6.mp3 0
sounds_cut/2/39.5/song7sample8.mp3 0
sounds_cut/2/39.5/song7sample1.mp3 0
sounds_cut/2/39.5/song7sample3.mp3 0
00:41:5-01:07:741
sounds_cut/2/41.5/song7sample1.mp3 88.0
sounds_cut/2/41.5/song7sample6.mp3 86.0
sounds_cut/2/41.5/song7sample4.mp3 74.0
sounds_cut/2/41.5/song7sample5.mp3 71.0
sounds_cut/2/41.5/song7sample7.mp3 55.0
sounds_cut/2/41.5/song7sample3.mp3 54.0
sounds_cut/2/41.5/song7sample8.mp3 43.0
sounds_cut/2/41.5/song7sample2.mp3 11.0
01:07:741-01:54:08
sounds_cut/2/67.741/song7sample3.mp3 126.0
```

```
sounds_cut/2/67.741/song7sample6.mp3 123.0
sounds_cut/2/67.741/song7sample4.mp3 114.0
sounds_cut/2/67.741/song7sample5.mp3 104.0
sounds_cut/2/67.741/song7sample1.mp3 90.0
sounds_cut/2/67.741/song7sample7.mp3 69.0
sounds_cut/2/67.741/song7sample2.mp3 15.0
01:54:08-02:09:298
sounds_cut/2/114.08/song7sample3.mp3 38.0
sounds_cut/2/114.08/song7sample4.mp3 29.0
sounds_cut/2/114.08/song7sample7.mp3 26.0
sounds_cut/2/114.08/song7sample5.mp3 18.0
sounds_cut/2/114.08/song7sample1.mp3 9.0
sounds_cut/2/114.08/song7sample2.mp3 6.0
02:09:298-03:35:85
sounds_cut/2/129.298/song7sample3.mp3 185.0
sounds_cut/2/129.298/song7sample4.mp3 183.0
sounds_cut/2/129.298/song7sample5.mp3 160.0
sounds_cut/2/129.298/song7sample1.mp3 140.0
sounds_cut/2/129.298/song7sample2.mp3 33.0
03:35:85-04:32:706
sounds_cut/2/215.85/song7sample3.mp3 78.0
sounds_cut/2/215.85/song7sample4.mp3 69.0
sounds_cut/2/215.85/song7sample1.mp3 52.0
sounds_cut/2/215.85/song7sample5.mp3 44.0
04:32:706-04:34:812
sounds_cut/2/272.706/song7sample3.mp3 0
sounds_cut/2/272.706/song7sample4.mp3 0
sounds_cut/2/272.706/song7sample1.mp3 0
04:34:812-04:37:516
sounds_cut/2/274.812/song7sample4.mp3 0
sounds_cut/2/274.812/song7sample1.mp3 0
04:37:516-04:40:977
sounds_cut/2/277.516/song7sample4.mp3 0

Cluster 3
00:00:0-00:48:8
sounds_cut/3/0.0/song5sample4.mp3 0
00:48:8-00:49:8
sounds_cut/3/48.8/song5sample4.mp3 0
sounds_cut/3/48.8/song5sample1.mp3 0
00:49:8-01:06:3
sounds_cut/3/49.8/song5sample3.mp3 0
sounds_cut/3/49.8/song5sample1.mp3 0
sounds_cut/3/49.8/song5sample4.mp3 0
01:06:3-01:11:0
sounds_cut/3/66.3/song5sample4.mp3 4.0
sounds_cut/3/66.3/song5sample5.mp3 0
sounds_cut/3/66.3/song5sample1.mp3 0
sounds_cut/3/66.3/song5sample3.mp3 0
01:11:0-01:19:9
```

```
sounds_cut/3/71.0/song5sample4.mp3 15.0
sounds_cut/3/71.0/song5sample2.mp3 13.0
sounds_cut/3/71.0/song5sample5.mp3 9.0
sounds_cut/3/71.0/song5sample1.mp3 2.0
sounds_cut/3/71.0/song5sample3.mp3 0
01:19:9-01:28:216
sounds_cut/3/79.9/song5sample2.mp3 20.0
sounds_cut/3/79.9/song5sample4.mp3 14.0
sounds_cut/3/79.9/song5sample5.mp3 6.0
sounds_cut/3/79.9/song5sample3.mp3 2.0
sounds_cut/3/79.9/song5sample6.mp3 0
sounds_cut/3/79.9/song5sample1.mp3 0
01:28:216-04:09:547
sounds_cut/3/88.216/song5sample5.mp3 176.0
sounds_cut/3/88.216/song5sample2.mp3 175.0
sounds_cut/3/88.216/song5sample1.mp3 97.0
sounds_cut/3/88.216/song5sample6.mp3 88.0
sounds_cut/3/88.216/song5sample3.mp3 0
04:09:547-04:13:79
sounds_cut/3/249.547/song5sample5.mp3 0
sounds_cut/3/249.547/song5sample1.mp3 0
sounds_cut/3/249.547/song5sample6.mp3 0
sounds_cut/3/249.547/song5sample3.mp3 0
04:13:79-04:15:871
sounds_cut/3/253.79/song5sample1.mp3 0
sounds_cut/3/253.79/song5sample6.mp3 0
sounds_cut/3/253.79/song5sample5.mp3 0
04:15:871-04:16:604
sounds_cut/3/255.871/song5sample6.mp3 0
sounds_cut/3/255.871/song5sample1.mp3 0
04:16:604-04:17:376
sounds_cut/3/256.604/song5sample6.mp3 0

Cluster 4
00:00:0-00:33:5
sounds_cut/4/0.0/song1sample6.mp3 0
00:33:5-00:50:2
sounds_cut/4/33.5/song1sample6.mp3 0
sounds_cut/4/33.5/song1sample1.mp3 0
00:50:2-01:10:6
sounds_cut/4/50.2/song1sample4.mp3 0
sounds_cut/4/50.2/song1sample1.mp3 0
sounds_cut/4/50.2/song1sample6.mp3 0
01:10:6-01:14:2
sounds_cut/4/70.6/song1sample5.mp3 0
sounds_cut/4/70.6/song1sample6.mp3 0
sounds_cut/4/70.6/song1sample4.mp3 0
sounds_cut/4/70.6/song1sample1.mp3 0
01:14:2-01:26:4
sounds_cut/4/74.2/song1sample12.mp3 0
```

```
sounds_cut/4/74.2/song1sample5.mp3 0
sounds_cut/4/74.2/song1sample6.mp3 0
sounds_cut/4/74.2/song1sample4.mp3 0
sounds_cut/4/74.2/song1sample1.mp3 0
01:26:4-01:27:9
sounds_cut/4/86.4/song1sample4.mp3 0
sounds_cut/4/86.4/song1sample6.mp3 0
sounds_cut/4/86.4/song1sample5.mp3 0
sounds_cut/4/86.4/song1sample14.mp3 0
sounds_cut/4/86.4/song1sample1.mp3 0
sounds_cut/4/86.4/song1sample12.mp3 0
01:27:9-01:44:9
sounds_cut/4/87.9/song1sample1.mp3 16.0
sounds_cut/4/87.9/song1sample14.mp3 10.0
sounds_cut/4/87.9/song1sample4.mp3 10.0
sounds_cut/4/87.9/song1sample6.mp3 9.0
sounds_cut/4/87.9/song1sample5.mp3 7.0
sounds_cut/4/87.9/song1sample12.mp3 3.0
sounds_cut/4/87.9/song1sample7.mp3 0
01:44:9-01:46:9
sounds_cut/4/104.9/song1sample4.mp3 0
sounds_cut/4/104.9/song1sample6.mp3 0
sounds_cut/4/104.9/song1sample5.mp3 0
sounds_cut/4/104.9/song1sample12.mp3 0
sounds_cut/4/104.9/song1sample7.mp3 0
sounds_cut/4/104.9/song1sample14.mp3 0
sounds_cut/4/104.9/song1sample1.mp3 0
sounds_cut/4/104.9/song1sample11.mp3 0
01:46:9-01:47:7
sounds_cut/4/106.9/song1sample6.mp3 0
sounds_cut/4/106.9/song1sample4.mp3 0
sounds_cut/4/106.9/song1sample9.mp3 0
sounds_cut/4/106.9/song1sample7.mp3 0
sounds_cut/4/106.9/song1sample5.mp3 0
sounds_cut/4/106.9/song1sample1.mp3 0
sounds_cut/4/106.9/song1sample12.mp3 0
sounds_cut/4/106.9/song1sample14.mp3 0
sounds_cut/4/106.9/song1sample11.mp3 0
01:47:7-02:09:867
sounds_cut/4/107.7/song1sample4.mp3 39.0
sounds_cut/4/107.7/song1sample7.mp3 30.0
sounds_cut/4/107.7/song1sample9.mp3 29.0
sounds_cut/4/107.7/song1sample14.mp3 29.0
sounds_cut/4/107.7/song1sample1.mp3 22.0
sounds_cut/4/107.7/song1sample13.mp3 10.0
sounds_cut/4/107.7/song1sample6.mp3 7.0
sounds_cut/4/107.7/song1sample12.mp3 3.0
sounds_cut/4/107.7/song1sample11.mp3 0
sounds_cut/4/107.7/song1sample5.mp3 0
02:09:867-02:33:77
```

```
sounds_cut/4/129.867/song1sample9.mp3 24.0
sounds_cut/4/129.867/song1sample4.mp3 16.0
sounds_cut/4/129.867/song1sample13.mp3 11.0
sounds_cut/4/129.867/song1sample1.mp3 9.0
sounds_cut/4/129.867/song1sample6.mp3 9.0
sounds_cut/4/129.867/song1sample7.mp3 8.0
sounds_cut/4/129.867/song1sample14.mp3 6.0
sounds_cut/4/129.867/song1sample11.mp3 5.0
sounds_cut/4/129.867/song1sample5.mp3 0
02:33:77-02:34:2
sounds_cut/4/153.77/song1sample13.mp3 0
sounds_cut/4/153.77/song1sample11.mp3 0
sounds_cut/4/153.77/song1sample7.mp3 0
sounds_cut/4/153.77/song1sample5.mp3 0
sounds_cut/4/153.77/song1sample9.mp3 0
sounds_cut/4/153.77/song1sample6.mp3 0
sounds_cut/4/153.77/song1sample4.mp3 0
sounds_cut/4/153.77/song1sample1.mp3 0
02:34:2-02:37:7
sounds_cut/4/154.2/song1sample9.mp3 2.0
sounds_cut/4/154.2/song1sample4.mp3 0
sounds_cut/4/154.2/song1sample6.mp3 0
sounds_cut/4/154.2/song1sample5.mp3 0
sounds_cut/4/154.2/song1sample7.mp3 0
sounds_cut/4/154.2/song1sample1.mp3 0
sounds_cut/4/154.2/song1sample13.mp3 0
sounds_cut/4/154.2/song1sample8.mp3 0
sounds_cut/4/154.2/song1sample11.mp3 0
02:37:7-03:22:735
sounds_cut/4/157.7/song1sample9.mp3 78.0
sounds_cut/4/157.7/song1sample4.mp3 58.0
sounds_cut/4/157.7/song1sample6.mp3 57.0
sounds_cut/4/157.7/song1sample2.mp3 38.0
sounds_cut/4/157.7/song1sample8.mp3 31.0
sounds_cut/4/157.7/song1sample13.mp3 25.0
sounds_cut/4/157.7/song1sample7.mp3 22.0
sounds_cut/4/157.7/song1sample1.mp3 20.0
sounds_cut/4/157.7/song1sample11.mp3 15.0
sounds_cut/4/157.7/song1sample5.mp3 8.0
03:22:735-03:23:728
sounds_cut/4/202.735/song1sample8.mp3 0
sounds_cut/4/202.735/song1sample9.mp3 0
sounds_cut/4/202.735/song1sample7.mp3 0
sounds_cut/4/202.735/song1sample5.mp3 0
sounds_cut/4/202.735/song1sample11.mp3 0
sounds_cut/4/202.735/song1sample13.mp3 0
sounds_cut/4/202.735/song1sample6.mp3 0
sounds_cut/4/202.735/song1sample1.mp3 0
sounds_cut/4/202.735/song1sample4.mp3 0
03:23:728-03:24:5
```

```
sounds_cut/4/203.728/song1sample13.mp3 0
sounds_cut/4/203.728/song1sample9.mp3 0
sounds_cut/4/203.728/song1sample6.mp3 0
sounds_cut/4/203.728/song1sample4.mp3 0
sounds_cut/4/203.728/song1sample1.mp3 0
sounds_cut/4/203.728/song1sample7.mp3 0
sounds_cut/4/203.728/song1sample11.mp3 0
sounds_cut/4/203.728/song1sample5.mp3 0
03:24:5-03:45:8
sounds_cut/4/204.5/song1sample9.mp3 30.0
sounds_cut/4/204.5/song1sample3.mp3 23.0
sounds_cut/4/204.5/song1sample13.mp3 22.0
sounds_cut/4/204.5/song1sample6.mp3 18.0
sounds_cut/4/204.5/song1sample5.mp3 15.0
sounds_cut/4/204.5/song1sample7.mp3 14.0
sounds_cut/4/204.5/song1sample4.mp3 10.0
sounds_cut/4/204.5/song1sample11.mp3 0
sounds_cut/4/204.5/song1sample1.mp3 0
03:45:8-04:17:747
sounds_cut/4/225.8/song1sample9.mp3 56.0
sounds_cut/4/225.8/song1sample3.mp3 50.0
sounds_cut/4/225.8/song1sample13.mp3 34.0
sounds_cut/4/225.8/song1sample15.mp3 30.0
sounds_cut/4/225.8/song1sample7.mp3 17.0
sounds_cut/4/225.8/song1sample4.mp3 16.0
sounds_cut/4/225.8/song1sample1.mp3 13.0
sounds_cut/4/225.8/song1sample11.mp3 12.0
sounds_cut/4/225.8/song1sample6.mp3 10.0
sounds_cut/4/225.8/song1sample5.mp3 0
04:17:747-04:27:309
sounds_cut/4/257.747/song1sample5.mp3 0
sounds_cut/4/257.747/song1sample15.mp3 0
sounds_cut/4/257.747/song1sample7.mp3 0
sounds_cut/4/257.747/song1sample4.mp3 0
sounds_cut/4/257.747/song1sample1.mp3 0
sounds_cut/4/257.747/song1sample6.mp3 0
sounds_cut/4/257.747/song1sample3.mp3 0
sounds_cut/4/257.747/song1sample11.mp3 0
sounds_cut/4/257.747/song1sample9.mp3 0
04:27:309-05:18:916
sounds_cut/4/267.309/song1sample9.mp3 18.0
sounds_cut/4/267.309/song1sample6.mp3 16.0
sounds_cut/4/267.309/song1sample7.mp3 14.0
sounds_cut/4/267.309/song1sample1.mp3 13.0
sounds_cut/4/267.309/song1sample11.mp3 11.0
sounds_cut/4/267.309/song1sample5.mp3 8.0
sounds_cut/4/267.309/song1sample3.mp3 6.0
sounds_cut/4/267.309/song1sample4.mp3 0
05:18:916-05:19:458
sounds_cut/4/318.916/song1sample7.mp3 0
```

```
sounds_cut/4/318.916/song1sample5.mp3 0
sounds_cut/4/318.916/song1sample6.mp3 0
sounds_cut/4/318.916/song1sample4.mp3 0
sounds_cut/4/318.916/song1sample1.mp3 0
sounds_cut/4/318.916/song1sample9.mp3 0
sounds_cut/4/318.916/song1sample11.mp3 0
05:19:458-05:19:893
sounds_cut/4/319.458/song1sample11.mp3 0
sounds_cut/4/319.458/song1sample5.mp3 0
sounds_cut/4/319.458/song1sample7.mp3 0
sounds_cut/4/319.458/song1sample4.mp3 0
sounds_cut/4/319.458/song1sample1.mp3 0
sounds_cut/4/319.458/song1sample6.mp3 0
05:19:893-05:20:679
sounds_cut/4/319.893/song1sample5.mp3 0
sounds_cut/4/319.893/song1sample1.mp3 0
sounds_cut/4/319.893/song1sample4.mp3 0
sounds_cut/4/319.893/song1sample11.mp3 0
sounds_cut/4/319.893/song1sample6.mp3 0
05:20:679-05:21:134
sounds_cut/4/320.679/song1sample11.mp3 0
sounds_cut/4/320.679/song1sample5.mp3 0
sounds_cut/4/320.679/song1sample1.mp3 0
sounds_cut/4/320.679/song1sample4.mp3 0
05:21:134-05:26:182
sounds_cut/4/321.134/song1sample11.mp3 0
sounds_cut/4/321.134/song1sample5.mp3 0
sounds_cut/4/321.134/song1sample4.mp3 0
05:26:182-05:26:262
sounds_cut/4/326.182/song1sample11.mp3 0
sounds_cut/4/326.182/song1sample4.mp3 0
05:26:262-05:31:956
sounds_cut/4/326.262/song1sample11.mp3 0

Cluster 5
00:00:0-00:12:4
sounds_cut/5/0.0/song10sample1.mp3 0
00:12:4-00:19:7
sounds_cut/5/12.4/song10sample2.mp3 0
sounds_cut/5/12.4/song10sample1.mp3 0
00:19:7-00:27:5
sounds_cut/5/19.7/song10sample1.mp3 0
sounds_cut/5/19.7/song10sample11.mp3 0
sounds_cut/5/19.7/song10sample2.mp3 0
00:27:5-00:31:2
sounds_cut/5/27.5/song10sample1.mp3 0
sounds_cut/5/27.5/song10sample11.mp3 0
sounds_cut/5/27.5/song10sample8.mp3 0
sounds_cut/5/27.5/song10sample2.mp3 0
00:31:2-00:41:3
```

```
sounds_cut/5/31.2/song10sample2.mp3 8.0
sounds_cut/5/31.2/song10sample1.mp3 7.0
sounds_cut/5/31.2/song10sample8.mp3 3.0
sounds_cut/5/31.2/song10sample11.mp3 0
sounds_cut/5/31.2/song10sample10.mp3 0
00:41:3-01:16:9
sounds_cut/5/41.3/song10sample2.mp3 41.0
sounds_cut/5/41.3/song10sample1.mp3 24.0
sounds_cut/5/41.3/song10sample11.mp3 12.0
sounds_cut/5/41.3/song10sample10.mp3 11.0
sounds_cut/5/41.3/song10sample8.mp3 8.0
sounds_cut/5/41.3/song10sample7.mp3 2.0
01:16:9-01:26:9
sounds_cut/5/76.9/song10sample1.mp3 11.0
sounds_cut/5/76.9/song10sample2.mp3 6.0
sounds_cut/5/76.9/song10sample3.mp3 5.0
sounds_cut/5/76.9/song10sample8.mp3 3.0
sounds_cut/5/76.9/song10sample10.mp3 2.0
sounds_cut/5/76.9/song10sample11.mp3 0
sounds_cut/5/76.9/song10sample7.mp3 0
01:26:9-01:50:5
sounds_cut/5/86.9/song10sample1.mp3 50.0
sounds_cut/5/86.9/song10sample2.mp3 47.0
sounds_cut/5/86.9/song10sample10.mp3 36.0
sounds_cut/5/86.9/song10sample9.mp3 36.0
sounds_cut/5/86.9/song10sample8.mp3 24.0
sounds_cut/5/86.9/song10sample3.mp3 21.0
sounds_cut/5/86.9/song10sample7.mp3 9.0
sounds_cut/5/86.9/song10sample11.mp3 5.0
01:50:5-02:13:53
sounds_cut/5/110.5/song10sample1.mp3 96.0
sounds_cut/5/110.5/song10sample9.mp3 71.0
sounds_cut/5/110.5/song10sample3.mp3 68.0
sounds_cut/5/110.5/song10sample2.mp3 58.0
sounds_cut/5/110.5/song10sample10.mp3 55.0
sounds_cut/5/110.5/song10sample6.mp3 46.0
sounds_cut/5/110.5/song10sample8.mp3 37.0
sounds_cut/5/110.5/song10sample11.mp3 27.0
sounds_cut/5/110.5/song10sample7.mp3 15.0
02:13:53-03:10:3
sounds_cut/5/133.53/song10sample1.mp3 309.0
sounds_cut/5/133.53/song10sample8.mp3 212.0
sounds_cut/5/133.53/song10sample9.mp3 171.0
sounds_cut/5/133.53/song10sample3.mp3 127.0
sounds_cut/5/133.53/song10sample6.mp3 115.0
sounds_cut/5/133.53/song10sample10.mp3 71.0
sounds_cut/5/133.53/song10sample7.mp3 47.0
sounds_cut/5/133.53/song10sample11.mp3 43.0
03:10:3-03:26:0
sounds_cut/5/190.3/song10sample1.mp3 26.0
```

```
sounds_cut/5/190.3/song10sample9.mp3 20.0
sounds_cut/5/190.3/song10sample3.mp3 8.0
sounds_cut/5/190.3/song10sample8.mp3 6.0
sounds_cut/5/190.3/song10sample6.mp3 4.0
sounds_cut/5/190.3/song10sample10.mp3 2.0
sounds_cut/5/190.3/song10sample7.mp3 0
sounds_cut/5/190.3/song10sample5.mp3 0
sounds_cut/5/190.3/song10sample11.mp3 0
03:26:0-04:34:364
sounds_cut/5/206.0/song10sample9.mp3 108.0
sounds_cut/5/206.0/song10sample1.mp3 96.0
sounds_cut/5/206.0/song10sample3.mp3 68.0
sounds_cut/5/206.0/song10sample6.mp3 59.0
sounds_cut/5/206.0/song10sample4.mp3 54.0
sounds_cut/5/206.0/song10sample5.mp3 27.0
sounds_cut/5/206.0/song10sample10.mp3 26.0
sounds_cut/5/206.0/song10sample11.mp3 23.0
sounds_cut/5/206.0/song10sample7.mp3 18.0
sounds_cut/5/206.0/song10sample8.mp3 12.0
04:34:364-04:55:44
sounds_cut/5/274.364/song10sample1.mp3 46.0
sounds_cut/5/274.364/song10sample9.mp3 28.0
sounds_cut/5/274.364/song10sample4.mp3 24.0
sounds_cut/5/274.364/song10sample3.mp3 22.0
sounds_cut/5/274.364/song10sample10.mp3 11.0
sounds_cut/5/274.364/song10sample11.mp3 10.0
sounds_cut/5/274.364/song10sample8.mp3 2.0
sounds_cut/5/274.364/song10sample5.mp3 0
sounds_cut/5/274.364/song10sample6.mp3 0
04:55:44-04:58:056
sounds_cut/5/295.44/song10sample6.mp3 0
sounds_cut/5/295.44/song10sample4.mp3 0
sounds_cut/5/295.44/song10sample5.mp3 0
sounds_cut/5/295.44/song10sample10.mp3 0
sounds_cut/5/295.44/song10sample11.mp3 0
sounds_cut/5/295.44/song10sample9.mp3 0
sounds_cut/5/295.44/song10sample1.mp3 0
sounds_cut/5/295.44/song10sample8.mp3 0
04:58:056-04:58:238
sounds_cut/5/298.056/song10sample8.mp3 0
sounds_cut/5/298.056/song10sample9.mp3 0
sounds_cut/5/298.056/song10sample5.mp3 0
sounds_cut/5/298.056/song10sample11.mp3 0
sounds_cut/5/298.056/song10sample10.mp3 0
sounds_cut/5/298.056/song10sample1.mp3 0
sounds_cut/5/298.056/song10sample6.mp3 0
04:58:238-05:21:192
sounds_cut/5/298.238/song10sample1.mp3 65.0
sounds_cut/5/298.238/song10sample6.mp3 37.0
sounds_cut/5/298.238/song10sample9.mp3 24.0
```

```
sounds_cut/5/298.238/song10sample10.mp3 14.0
sounds_cut/5/298.238/song10sample11.mp3 11.0
sounds_cut/5/298.238/song10sample8.mp3 0
05:21:192-05:44:261
sounds_cut/5/321.192/song10sample1.mp3 7.0
sounds_cut/5/321.192/song10sample6.mp3 4.0
sounds_cut/5/321.192/song10sample10.mp3 3.0
sounds_cut/5/321.192/song10sample8.mp3 0
sounds_cut/5/321.192/song10sample11.mp3 0
05:44:261-05:59:893
sounds_cut/5/344.261/song10sample10.mp3 0
sounds_cut/5/344.261/song10sample1.mp3 0
sounds_cut/5/344.261/song10sample6.mp3 0
sounds_cut/5/344.261/song10sample11.mp3 0
05:59:893-06:01:388
sounds_cut/5/359.893/song10sample1.mp3 0
sounds_cut/5/359.893/song10sample10.mp3 0
sounds_cut/5/359.893/song10sample6.mp3 0
06:01:388-06:04:985
sounds_cut/5/361.388/song10sample1.mp3 3.0
sounds_cut/5/361.388/song10sample6.mp3 0
06:04:985-06:11:592
sounds_cut/5/364.985/song10sample1.mp3 0

Cluster 6
00:00:0-00:04:3
sounds_cut/6/0.0/song9sample1.mp3 0
00:04:3-00:11:3
sounds_cut/6/4.3/song9sample1.mp3 0
sounds_cut/6/4.3/song9sample4.mp3 0
00:11:3-00:18:3
sounds_cut/6/11.3/song9sample4.mp3 5.0
sounds_cut/6/11.3/song9sample5.mp3 0
sounds_cut/6/11.3/song9sample1.mp3 0
00:18:3-00:48:8
sounds_cut/6/18.3/song9sample5.mp3 53.0
sounds_cut/6/18.3/song9sample3.mp3 40.0
sounds_cut/6/18.3/song9sample4.mp3 28.0
sounds_cut/6/18.3/song9sample1.mp3 11.0
00:48:8-01:40:769
sounds_cut/6/48.8/song9sample3.mp3 57.0
sounds_cut/6/48.8/song9sample5.mp3 48.0
sounds_cut/6/48.8/song9sample2.mp3 43.0
sounds_cut/6/48.8/song9sample4.mp3 41.0
sounds_cut/6/48.8/song9sample1.mp3 32.0
01:40:769-04:40:832
sounds_cut/6/100.769/song9sample5.mp3 172.0
sounds_cut/6/100.769/song9sample4.mp3 161.0
sounds_cut/6/100.769/song9sample2.mp3 107.0
sounds_cut/6/100.769/song9sample1.mp3 72.0
```

```
04:40:832-04:40:924
sounds_cut/6/280.832/song9sample2.mp3 0
sounds_cut/6/280.832/song9sample5.mp3 0
sounds_cut/6/280.832/song9sample1.mp3 0
04:40:924-04:45:283
sounds_cut/6/280.924/song9sample1.mp3 0
sounds_cut/6/280.924/song9sample5.mp3 0
04:45:283-04:47:362
sounds_cut/6/285.283/song9sample5.mp3 0

Cluster 7
00:00:0-01:12:4
sounds_cut/7/0.0/song8sample7.mp3 0
01:12:4-01:56:1
sounds_cut/7/72.4/song8sample7.mp3 0
sounds_cut/7/72.4/song8sample8.mp3 0
01:56:1-04:52:6
sounds_cut/7/116.1/song8sample7.mp3 44.0
sounds_cut/7/116.1/song8sample8.mp3 29.0
sounds_cut/7/116.1/song8sample3.mp3 23.0
04:52:6-05:09:8
sounds_cut/7/292.6/song8sample1.mp3 7.0
sounds_cut/7/292.6/song8sample3.mp3 6.0
sounds_cut/7/292.6/song8sample8.mp3 0
sounds_cut/7/292.6/song8sample7.mp3 0
05:09:8-05:28:4
sounds_cut/7/309.8/song8sample2.mp3 13.0
sounds_cut/7/309.8/song8sample7.mp3 5.0
sounds_cut/7/309.8/song8sample3.mp3 2.0
sounds_cut/7/309.8/song8sample8.mp3 0
sounds_cut/7/309.8/song8sample1.mp3 0
05:28:4-05:53:6
sounds_cut/7/328.4/song8sample3.mp3 33.0
sounds_cut/7/328.4/song8sample7.mp3 26.0
sounds_cut/7/328.4/song8sample6.mp3 22.0
sounds_cut/7/328.4/song8sample2.mp3 22.0
sounds_cut/7/328.4/song8sample8.mp3 21.0
sounds_cut/7/328.4/song8sample1.mp3 17.0
05:53:6-07:01:4
sounds_cut/7/353.6/song8sample2.mp3 117.0
sounds_cut/7/353.6/song8sample8.mp3 108.0
sounds_cut/7/353.6/song8sample1.mp3 102.0
sounds_cut/7/353.6/song8sample3.mp3 87.0
sounds_cut/7/353.6/song8sample7.mp3 67.0
sounds_cut/7/353.6/song8sample6.mp3 65.0
sounds_cut/7/353.6/song8sample5.mp3 0
07:01:4-07:47:63
sounds_cut/7/421.4/song8sample2.mp3 88.0
sounds_cut/7/421.4/song8sample6.mp3 70.0
sounds_cut/7/421.4/song8sample8.mp3 68.0
```

```
sounds_cut/7/421.4/song8sample3.mp3 54.0
sounds_cut/7/421.4/song8sample1.mp3 53.0
sounds_cut/7/421.4/song8sample7.mp3 45.0
sounds_cut/7/421.4/song8sample4.mp3 30.0
sounds_cut/7/421.4/song8sample5.mp3 0
07:47:63-08:23:137
sounds_cut/7/467.63/song8sample6.mp3 43.0
sounds_cut/7/467.63/song8sample2.mp3 33.0
sounds_cut/7/467.63/song8sample8.mp3 26.0
sounds_cut/7/467.63/song8sample4.mp3 17.0
sounds_cut/7/467.63/song8sample7.mp3 8.0
sounds_cut/7/467.63/song8sample1.mp3 7.0
sounds_cut/7/467.63/song8sample5.mp3 0
08:23:137-10:06:317
sounds_cut/7/503.137/song8sample8.mp3 150.0
sounds_cut/7/503.137/song8sample6.mp3 133.0
sounds_cut/7/503.137/song8sample5.mp3 133.0
sounds_cut/7/503.137/song8sample2.mp3 128.0
sounds_cut/7/503.137/song8sample1.mp3 80.0
sounds_cut/7/503.137/song8sample7.mp3 77.0
10:06:317-10:08:98
sounds_cut/7/606.317/song8sample5.mp3 0
sounds_cut/7/606.317/song8sample8.mp3 0
sounds_cut/7/606.317/song8sample7.mp3 0
sounds_cut/7/606.317/song8sample2.mp3 0
sounds_cut/7/606.317/song8sample1.mp3 0
10:08:98-10:10:653
sounds_cut/7/608.98/song8sample1.mp3 0
sounds_cut/7/608.98/song8sample7.mp3 0
sounds_cut/7/608.98/song8sample8.mp3 0
sounds_cut/7/608.98/song8sample5.mp3 0
10:10:653-10:11:869
sounds_cut/7/610.653/song8sample1.mp3 0
sounds_cut/7/610.653/song8sample5.mp3 0
sounds_cut/7/610.653/song8sample7.mp3 0
10:11:869-10:17:273
sounds_cut/7/611.869/song8sample5.mp3 0
sounds_cut/7/611.869/song8sample7.mp3 0
10:17:273-11:11:275
sounds_cut/7/617.273/song8sample5.mp3 0

Cluster 8
00:00:0-00:06:0
sounds_cut/8/0.0/song4sample1.mp3 0
00:06:0-00:06:9
sounds_cut/8/6.0/song4sample7.mp3 0
sounds_cut/8/6.0/song4sample1.mp3 0
00:06:9-00:12:5
sounds_cut/8/6.9/song4sample1.mp3 0
sounds_cut/8/6.9/song4sample6.mp3 0
```

```
679 sounds_cut/8/6.9/song4sample7.mp3 0
680 00:12:5-00:14:5
681 sounds_cut/8/12.5/song4sample1.mp3 3.0
682 sounds_cut/8/12.5/song4sample6.mp3 0
683 sounds_cut/8/12.5/song4sample8.mp3 0
684 sounds_cut/8/12.5/song4sample7.mp3 0
685 00:14:5-00:18:2
686 sounds_cut/8/14.5/song4sample6.mp3 0
687 sounds_cut/8/14.5/song4sample1.mp3 0
688 sounds_cut/8/14.5/song4sample7.mp3 0
689 sounds_cut/8/14.5/song4sample8.mp3 0
690 sounds_cut/8/14.5/song4sample9.mp3 0
691 00:18:2-00:23:1
692 sounds_cut/8/18.2/song4sample6.mp3 0
693 sounds_cut/8/18.2/song4sample1.mp3 0
694 sounds_cut/8/18.2/song4sample4.mp3 0
695 sounds_cut/8/18.2/song4sample7.mp3 0
696 sounds_cut/8/18.2/song4sample8.mp3 0
697 sounds_cut/8/18.2/song4sample9.mp3 0
698 00:23:1-00:40:3
699 sounds_cut/8/23.1/song4sample9.mp3 14.0
700 sounds_cut/8/23.1/song4sample4.mp3 10.0
701 sounds_cut/8/23.1/song4sample8.mp3 9.0
702 sounds_cut/8/23.1/song4sample1.mp3 8.0
703 sounds_cut/8/23.1/song4sample5.mp3 5.0
704 sounds_cut/8/23.1/song4sample7.mp3 2.0
705 sounds_cut/8/23.1/song4sample6.mp3 0
706 00:40:3-01:51:562
707 sounds_cut/8/40.3/song4sample1.mp3 113.0
708 sounds_cut/8/40.3/song4sample3.mp3 71.0
709 sounds_cut/8/40.3/song4sample9.mp3 67.0
710 sounds_cut/8/40.3/song4sample8.mp3 63.0
711 sounds_cut/8/40.3/song4sample4.mp3 60.0
712 sounds_cut/8/40.3/song4sample6.mp3 53.0
713 sounds_cut/8/40.3/song4sample7.mp3 44.0
714 sounds_cut/8/40.3/song4sample5.mp3 43.0
715 01:51:562-01:53:051
716 sounds_cut/8/111.562/song4sample6.mp3 0
717 sounds_cut/8/111.562/song4sample7.mp3 0
718 sounds_cut/8/111.562/song4sample5.mp3 0
719 sounds_cut/8/111.562/song4sample9.mp3 0
720 sounds_cut/8/111.562/song4sample3.mp3 0
721 sounds_cut/8/111.562/song4sample1.mp3 0
722 sounds_cut/8/111.562/song4sample8.mp3 0
723 01:53:051-01:56:498
724 sounds_cut/8/113.051/song4sample6.mp3 2.0
725 sounds_cut/8/113.051/song4sample9.mp3 0
726 sounds_cut/8/113.051/song4sample8.mp3 0
727 sounds_cut/8/113.051/song4sample5.mp3 0
728 sounds_cut/8/113.051/song4sample7.mp3 0
```

```
729 sounds_cut/8/113.051/song4sample1.mp3 0
730 01:56:498-02:43:4
731 sounds_cut/8/116.498/song4sample1.mp3 18.0
732 sounds_cut/8/116.498/song4sample9.mp3 18.0
733 sounds_cut/8/116.498/song4sample8.mp3 16.0
734 sounds_cut/8/116.498/song4sample5.mp3 14.0
735 sounds_cut/8/116.498/song4sample6.mp3 13.0
736 02:43:4-02:58:0
737 sounds_cut/8/163.4/song4sample8.mp3 22.0
738 sounds_cut/8/163.4/song4sample9.mp3 21.0
739 sounds_cut/8/163.4/song4sample1.mp3 18.0
740 sounds_cut/8/163.4/song4sample5.mp3 11.0
741 sounds_cut/8/163.4/song4sample6.mp3 8.0
742 sounds_cut/8/163.4/song4sample2.mp3 0
743 02:58:0-03:38:516
744 sounds_cut/8/178.0/song4sample9.mp3 99.0
745 sounds_cut/8/178.0/song4sample5.mp3 83.0
746 sounds_cut/8/178.0/song4sample11.mp3 79.0
747 sounds_cut/8/178.0/song4sample6.mp3 77.0
748 sounds_cut/8/178.0/song4sample1.mp3 74.0
749 sounds_cut/8/178.0/song4sample8.mp3 70.0
750 sounds_cut/8/178.0/song4sample2.mp3 34.0
751 03:38:516-03:48:971
752 sounds_cut/8/218.516/song4sample9.mp3 23.0
753 sounds_cut/8/218.516/song4sample8.mp3 11.0
754 sounds_cut/8/218.516/song4sample2.mp3 9.0
755 sounds_cut/8/218.516/song4sample1.mp3 4.0
756 sounds_cut/8/218.516/song4sample5.mp3 0
757 sounds_cut/8/218.516/song4sample6.mp3 0
758 03:48:971-03:49:238
759 sounds_cut/8/228.971/song4sample6.mp3 0
760 sounds_cut/8/228.971/song4sample1.mp3 0
761 sounds_cut/8/228.971/song4sample2.mp3 0
762 sounds_cut/8/228.971/song4sample8.mp3 0
763 sounds_cut/8/228.971/song4sample9.mp3 0
764 03:49:238-04:15:847
765 sounds_cut/8/229.238/song4sample9.mp3 15.0
766 sounds_cut/8/229.238/song4sample1.mp3 15.0
767 sounds_cut/8/229.238/song4sample6.mp3 14.0
768 sounds_cut/8/229.238/song4sample2.mp3 4.0
769 04:15:847-04:16:474
770 sounds_cut/8/255.847/song4sample1.mp3 0
771 sounds_cut/8/255.847/song4sample6.mp3 0
772 sounds_cut/8/255.847/song4sample9.mp3 0
773 04:16:474-05:37:87
774 sounds_cut/8/256.474/song4sample1.mp3 31.0
775 sounds_cut/8/256.474/song4sample9.mp3 21.0
776 05:37:87-05:45:469
777 sounds_cut/8/337.87/song4sample1.mp3 0
778
```

```
Cluster 9
00:00:0-00:01:3
sounds_cut/9/0.0/song6sample6.mp3 0
00:01:3-00:04:6
sounds_cut/9/1.3/song6sample6.mp3 0
sounds_cut/9/1.3/song6sample1.mp3 0
00:04:6-00:10:7
sounds_cut/9/4.6/song6sample5.mp3 3.0
sounds_cut/9/4.6/song6sample1.mp3 3.0
sounds_cut/9/4.6/song6sample6.mp3 0
00:10:7-00:27:7
sounds_cut/9/10.7/song6sample5.mp3 14.0
sounds_cut/9/10.7/song6sample1.mp3 14.0
sounds_cut/9/10.7/song6sample6.mp3 10.0
sounds_cut/9/10.7/song6sample4.mp3 3.0
00:27:7-00:42:7
sounds_cut/9/27.7/song6sample4.mp3 7.0
sounds_cut/9/27.7/song6sample6.mp3 7.0
sounds_cut/9/27.7/song6sample5.mp3 3.0
sounds_cut/9/27.7/song6sample10.mp3 3.0
sounds_cut/9/27.7/song6sample1.mp3 0
00:42:7-01:40:2
sounds_cut/9/42.7/song6sample11.mp3 63.0
sounds_cut/9/42.7/song6sample5.mp3 48.0
sounds_cut/9/42.7/song6sample1.mp3 46.0
sounds_cut/9/42.7/song6sample10.mp3 31.0
sounds_cut/9/42.7/song6sample4.mp3 18.0
sounds_cut/9/42.7/song6sample6.mp3 17.0
01:40:2-01:44:3
sounds_cut/9/100.2/song6sample11.mp3 12.0
sounds_cut/9/100.2/song6sample10.mp3 6.0
sounds_cut/9/100.2/song6sample5.mp3 4.0
sounds_cut/9/100.2/song6sample6.mp3 2.0
sounds_cut/9/100.2/song6sample1.mp3 2.0
sounds_cut/9/100.2/song6sample4.mp3 2.0
sounds_cut/9/100.2/song6sample3.mp3 0
01:44:3-01:48:9
sounds_cut/9/104.3/song6sample11.mp3 13.0
sounds_cut/9/104.3/song6sample1.mp3 4.0
sounds_cut/9/104.3/song6sample10.mp3 4.0
sounds_cut/9/104.3/song6sample3.mp3 3.0
sounds_cut/9/104.3/song6sample6.mp3 2.0
sounds_cut/9/104.3/song6sample5.mp3 0
sounds_cut/9/104.3/song6sample8.mp3 0
sounds_cut/9/104.3/song6sample4.mp3 0
01:48:9-01:51:2
sounds_cut/9/108.9/song6sample11.mp3 5.0
sounds_cut/9/108.9/song6sample7.mp3 4.0
sounds_cut/9/108.9/song6sample10.mp3 3.0
sounds_cut/9/108.9/song6sample3.mp3 2.0
```

```
sounds_cut/9/108.9/song6sample6.mp3 0
sounds_cut/9/108.9/song6sample4.mp3 0
sounds_cut/9/108.9/song6sample1.mp3 0
sounds_cut/9/108.9/song6sample8.mp3 0
sounds_cut/9/108.9/song6sample5.mp3 0
01:51:2-02:20:063
sounds_cut/9/111.2/song6sample7.mp3 83.0
sounds_cut/9/111.2/song6sample1.mp3 72.0
sounds_cut/9/111.2/song6sample3.mp3 71.0
sounds_cut/9/111.2/song6sample11.mp3 69.0
sounds_cut/9/111.2/song6sample5.mp3 67.0
sounds_cut/9/111.2/song6sample6.mp3 67.0
sounds_cut/9/111.2/song6sample2.mp3 60.0
sounds_cut/9/111.2/song6sample10.mp3 60.0
sounds_cut/9/111.2/song6sample4.mp3 23.0
sounds_cut/9/111.2/song6sample8.mp3 0
02:20:063-02:31:533
sounds_cut/9/140.063/song6sample7.mp3 59.0
sounds_cut/9/140.063/song6sample11.mp3 42.0
sounds_cut/9/140.063/song6sample1.mp3 40.0
sounds_cut/9/140.063/song6sample6.mp3 30.0
sounds_cut/9/140.063/song6sample10.mp3 28.0
sounds_cut/9/140.063/song6sample5.mp3 26.0
sounds_cut/9/140.063/song6sample2.mp3 22.0
sounds_cut/9/140.063/song6sample4.mp3 10.0
sounds_cut/9/140.063/song6sample8.mp3 8.0
02:31:533-02:58:772
sounds_cut/9/151.533/song6sample1.mp3 75.0
sounds_cut/9/151.533/song6sample7.mp3 70.0
sounds_cut/9/151.533/song6sample6.mp3 69.0
sounds_cut/9/151.533/song6sample5.mp3 64.0
sounds_cut/9/151.533/song6sample11.mp3 57.0
sounds_cut/9/151.533/song6sample10.mp3 42.0
sounds_cut/9/151.533/song6sample4.mp3 36.0
sounds_cut/9/151.533/song6sample8.mp3 28.0
02:58:772-04:07:3
sounds_cut/9/178.772/song6sample7.mp3 277.0
sounds_cut/9/178.772/song6sample1.mp3 241.0
sounds_cut/9/178.772/song6sample6.mp3 219.0
sounds_cut/9/178.772/song6sample5.mp3 174.0
sounds_cut/9/178.772/song6sample10.mp3 158.0
sounds_cut/9/178.772/song6sample8.mp3 24.0
sounds_cut/9/178.772/song6sample4.mp3 4.0
04:07:3-04:33:617
sounds_cut/9/247.3/song6sample5.mp3 47.0
sounds_cut/9/247.3/song6sample7.mp3 47.0
sounds_cut/9/247.3/song6sample1.mp3 39.0
sounds_cut/9/247.3/song6sample10.mp3 32.0
sounds_cut/9/247.3/song6sample6.mp3 24.0
sounds_cut/9/247.3/song6sample8.mp3 10.0
```

```
sounds_cut/9/247.3/song6sample9.mp3 10.0
sounds_cut/9/247.3/song6sample4.mp3 0
04:33:617-05:03:491
sounds_cut/9/273.617/song6sample5.mp3 43.0
sounds_cut/9/273.617/song6sample7.mp3 34.0
sounds_cut/9/273.617/song6sample9.mp3 22.0
sounds_cut/9/273.617/song6sample6.mp3 15.0
sounds_cut/9/273.617/song6sample1.mp3 14.0
sounds_cut/9/273.617/song6sample8.mp3 4.0
sounds_cut/9/273.617/song6sample4.mp3 0
05:03:491-05:14:409
sounds_cut/9/303.491/song6sample7.mp3 16.0
sounds_cut/9/303.491/song6sample9.mp3 11.0
sounds_cut/9/303.491/song6sample5.mp3 3.0
sounds_cut/9/303.491/song6sample1.mp3 2.0
sounds_cut/9/303.491/song6sample8.mp3 0
sounds_cut/9/303.491/song6sample4.mp3 0
05:14:409-06:03:902
sounds_cut/9/314.409/song6sample7.mp3 35.0
sounds_cut/9/314.409/song6sample5.mp3 35.0
sounds_cut/9/314.409/song6sample1.mp3 29.0
sounds_cut/9/314.409/song6sample8.mp3 0
sounds_cut/9/314.409/song6sample4.mp3 0
06:03:902-06:31:63
sounds_cut/9/363.902/song6sample7.mp3 14.0
sounds_cut/9/363.902/song6sample1.mp3 10.0
sounds_cut/9/363.902/song6sample5.mp3 7.0
sounds_cut/9/363.902/song6sample8.mp3 5.0
06:31:63-06:36:717
sounds_cut/9/391.63/song6sample1.mp3 0
sounds_cut/9/391.63/song6sample7.mp3 0
sounds_cut/9/391.63/song6sample8.mp3 0
06:36:717-06:48:313
sounds_cut/9/396.717/song6sample1.mp3 0
sounds_cut/9/396.717/song6sample8.mp3 0
06:48:313-07:11:955
sounds_cut/9/408.313/song6sample1.mp3 0

Cluster 10
00:00:0-00:10:6
sounds_cut/10/0.0/song3sample1.mp3 0
00:10:6-00:10:7
sounds_cut/10/10.6/song3sample1.mp3 0
sounds_cut/10/10.6/song3sample6.mp3 0
00:10:7-00:13:8
sounds_cut/10/10.7/song3sample2.mp3 0
sounds_cut/10/10.7/song3sample6.mp3 0
sounds_cut/10/10.7/song3sample1.mp3 0
00:13:8-00:16:3
sounds_cut/10/13.8/song3sample2.mp3 0
```

```
sounds_cut/10/13.8/song3sample3.mp3 0
sounds_cut/10/13.8/song3sample6.mp3 0
sounds_cut/10/13.8/song3sample1.mp3 0
00:16:3-00:18:1
sounds_cut/10/16.3/song3sample1.mp3 0
sounds_cut/10/16.3/song3sample3.mp3 0
sounds_cut/10/16.3/song3sample6.mp3 0
sounds_cut/10/16.3/song3sample7.mp3 0
sounds_cut/10/16.3/song3sample2.mp3 0
00:18:1-01:01:0
sounds_cut/10/18.1/song3sample6.mp3 29.0
sounds_cut/10/18.1/song3sample1.mp3 25.0
sounds_cut/10/18.1/song3sample7.mp3 16.0
sounds_cut/10/18.1/song3sample4.mp3 16.0
sounds_cut/10/18.1/song3sample2.mp3 15.0
sounds_cut/10/18.1/song3sample3.mp3 12.0
01:01:0-02:03:746
sounds_cut/10/61.0/song3sample1.mp3 141.0
sounds_cut/10/61.0/song3sample3.mp3 124.0
sounds_cut/10/61.0/song3sample6.mp3 118.0
sounds_cut/10/61.0/song3sample2.mp3 83.0
sounds_cut/10/61.0/song3sample4.mp3 78.0
sounds_cut/10/61.0/song3sample5.mp3 74.0
sounds_cut/10/61.0/song3sample7.mp3 62.0
02:03:746-04:00:909
sounds_cut/10/123.746/song3sample2.mp3 206.0
sounds_cut/10/123.746/song3sample6.mp3 199.0
sounds_cut/10/123.746/song3sample1.mp3 191.0
sounds_cut/10/123.746/song3sample3.mp3 157.0
sounds_cut/10/123.746/song3sample4.mp3 143.0
sounds_cut/10/123.746/song3sample7.mp3 133.0
04:00:909-04:30:775
sounds_cut/10/240.909/song3sample1.mp3 49.0
sounds_cut/10/240.909/song3sample6.mp3 40.0
sounds_cut/10/240.909/song3sample7.mp3 24.0
sounds_cut/10/240.909/song3sample2.mp3 18.0
sounds_cut/10/240.909/song3sample4.mp3 13.0
04:30:775-04:32:507
sounds_cut/10/270.775/song3sample7.mp3 0
sounds_cut/10/270.775/song3sample6.mp3 0
sounds_cut/10/270.775/song3sample4.mp3 0
sounds_cut/10/270.775/song3sample1.mp3 0
04:32:507-04:35:173
sounds_cut/10/272.507/song3sample6.mp3 0
sounds_cut/10/272.507/song3sample1.mp3 0
sounds_cut/10/272.507/song3sample7.mp3 0
04:35:173-04:35:357
sounds_cut/10/275.173/song3sample1.mp3 0
sounds_cut/10/275.173/song3sample6.mp3 0
04:35:357-04:37:519
```

```
979 sounds_cut/10/275.357/song3sample6.mp3 0
```

Listing B.1: Output clusters' segments file representing the different segments of each cluster. Each segment contains the file paths of the different files it contains, ordered by their quality scores (presented on the right end of each path).