Faculdade de Engenharia da Universidade do Porto

FEUP

Singing Voice Detection in Polyphonic Music Signals

Pedro Luis Cameira Sollari Allegro

VERSÃO PROVISÓRIA

Dissertação realizada no âmbito do
Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Ramo de Telecomunicações

Orientadores:
Prof. Fabien Gouyon
Prof. Jaime S. Cardoso

Junho de 2008

Resumo

O canto é uma componente muito importante da música pois a sua flexibilidade permite-lhe acompanhar diversos instrumentos sendo por isso usada em quase todos os géneros musicais. Torna-se assim crítico criar técnicas robustas de análise de forma a extrair informação sobre esta importante componente da música. Estas técnicas podem auxiliar aplicações de gestão de músicas fornecendo um factor de discriminação como são por exemplo os instrumentos ou a batida da música. A detecção de canto pode ser utilizada em muitas outras aplicações como karaoke, reconhecimento do idioma ou trancrição de letras de músicas.

Neste trabalho um esquema geral de detecção de canto em músicas polifónicas será testado para melhor compreender quais sao as limitações e o que pode ser melhorado nas técnicas existentes. Algumas técnicas de preprocessamento e de diferentes atributos dos sinais de música serão utilizados de maneira a determinar a sua utilidade na detecção de canto. Nas referidas experiências foram usadas três diferentes conjuntos de músicas. Dois dos quais já foram usados por outros investigadores enquanto que o outro foi criado para este projecto.

Os parâmetros que demonstraram resultados mais interessantes foram o aumento da janela de processamento, o uso dos coeficientes MFCC e a sua combinação com outros coeficientes utilizados neste projecto. Os melhores resultados foram obtidos quando os coeficientes MFCC, LSP, LPCC e panning foram usados com os classificadores SVM e J48 atingindo valores a rondar os 89%.

Abstract

The singing voice is a very important component of music because its flexibility allows it to accompany many different instruments thus enabling it to appear in almost all of the existing musical genres. It becomes critical that robust analysis techniques are created to withdraw information about this important component of music. These techniques can help music management applications by supplying a discriminator factor as are the instruments or the rhythmic aspects. Singing voice detection techniques can be applied in many other applications like karaoke, language recognition or lyric transcription.

In this work a general framework for voice detection in polyphonic music will be tested in order to better comprehend what are the limitations and what can be improved in the current existing techniques. Some preprocessing techniques and different features will be used in order to comprehend their usefulness on singing voice detection in music signals. For these experiments three different sets of data were used. One data set was created for this work while the other two were already used by other investigators.

Of all the tested parameters, the increase of the texture windows, the use of MFCC coefficients and their combination with other features presented very interesting results. The best results were achieved when combining the MFCC, LSP , LPCC and panning features with the SVM and J48 classifiers reaching values around 89% .

Acknowledgments

I would like to thank several people that contributed for the development of this thesis.

First and foremost I must thank Prof. Fabien Gouyon and Prof. Jaime S. Cardoso for their guidance, support and for the encouragement always demonstrated, helping me whenever possible. I have also been assisted by other members of the UTM unit at INESC Porto but I would particularly like to thank Luis Gustavo Martins for all the valuable advices given.

I would also like to thank my family for the comprehension and comfort given during the course of this work. A final word of appreciation should be delivered to all my closest friends for their patience and companionship, especially to João Oliveira.

Table of Contents

List of Figures

List of Tables

Acronym List

ARFF - Attribute-Relation File Format

FEUP - Faculdade de Engenharia da Universidade do Porto

LPCC - Linear Predictive Coding derived Cepstral Coefficients

LSP - Linear Spectral Pairs

MARSYAS - Music Analysis, Retrieval and Synthesis for Audio Signals

MFCC – Mel-Frequency Cepstral Coefficients

MIR - Music Information Retrieval

SMO - Sequential Minimal Optimization

SVM – Support Vector Machine

WEKA - Waikato Environment for Knowledge Analysis

Chapter 1

1. Introduction

This thesis is part of the final course of the 2º semester, of the 5th year of the Integrated Masters in Engenharia Electrotécnica e de Computadores at Faculdade de Engenharia da Universidade do Porto (FEUP).

With the development of technologies such as portable music players and the increase of internet transfer data rates, music has know a popularity increase because it becomes easier for people to listen to the songs they like the most. With the increasing interest of general population in music applications came the need for innovation in this field. These applications can for example help the users to search for specific songs or manage them according to size, genre, duration or other music characteristics. Other applications could enable the user to modify songs based on timbre, rhythm, or by adding or subtracting instruments. There are many characteristics of music that can be explored for categorization like the rhythm and instruments but for this project only the singing voice will be looked upon.

The wide range of sounds the human voice is capable of producing and the capacity it has to draw our attention turned it to the most prevalent instrument in music. Currently many singing voice detection techniques exist but none achieves satisfactory results. As an application example one could consider a music player software that enabled the user to advance to the sections of the song that contained singing voice or automatically play these sections with increased volume or other effects. Still considering this application, it could also by user request create several playlists of songs based on the percentages of singing voice in the total track time. Another application example could be a music editing software that would be able to mixture singing voice segments from one song with instrument segments from another song. Consider also an automatic karaoke software application were the user could input any music file and the singing voice sections would be automatically separated from the main mixture and the lyrics would be matched using lyric alignment software. For this application to work a singing voice alignment technique [1] would have to be utilized. This lyric alignment software could also be used for a key word managing system that would play songs based on their lyrics.

In addition to the mentioned applications, singing voice detection can aid in many other research areas because if the segments of the songs where voice is present can be correctly identified it would make some types of investigation more effective such as:

- Singer identification [2] and [3].
- Identification of multiple singers.
- Singer quality evaluation - used for example in games [4].
- Musical style detection [5]– because different types of music styles have different contribution from singers.
- Language identification [6].

The rest of this chapter is divided into four sections. The objectives for this thesis are revealed in the first section along with the motivations for working on this dissertation while on the second section some relevant software tools are discribed. On the third section the main structure of this document is explained and finally on the fourth section the contributions of this work are identified.

1.1 Context

The goal of this dissertation is to study the effects of preprocessing techniques and features extraction on singing voice detection in polyphonic music signals.

Polyphonic music is composed of sounds that can be originated from several different instruments. For the human ear it is relatively easy to differentiate sound sources or instruments but the same is not valid for computers. Of all the sounds present in music, voice is the most prominent one but generally computers cannot distinguish them perfectly from other sounds. This fact is more relevant when dealing with melodic sounds. Melodic sounds are sequences of notes that produce harmonized music like the guitar as opposed to the ones generated by drums. Singing voice detection tries to help on the resolution of this problem by identifying the presence of singing voice in a music signal. Rather than directly separate the singing voice, this operation tries to label the segments where singing voice is found. On this thesis some experiments will be executed in order to comprehend what are the effects of changing any of the parameters of the voice detection framework that will be explained in chapter 2.

This section, the context, is divided in three subsections. On the first one the scope of this dissertation is briefly explained, on the second a short explanation of the work environment is presented while on the last section the personal motivations are revealed.

1.1.1 MIR Research

The work developed during the course of this dissertation is inserted in the area of Music Information Retrieval (MIR). MIR is the interdisciplinary science of retrieving information from music. It’s still not very well known due to its recent creation. Despite this fact it has known an

incredible growth in the last few years mainly because of the potentialities in academic, industrial and entertainment applications. A great aid in this growth has been the collaboration and information exchange between the members of MIR. An example of a MIR system is the web search engine FindSounds.com [7] that enables the user to find any sound file based file formats, sample rate, file size and others. Another example, Foafing the Music is a web based [8] music recommender system that suggest songs to the user based on given information such as residence, likings, etc. Many other examples can be found in the MIR systems website [9].

1.1.2 Research at INESC Porto

This research was carried out at the Institute for Systems and Computer Engineering of Porto (INESC Porto). The main activities of INESC Porto are scientific research and technological development as well as consulting and advanced training. Activities are clustered as follows: Telecommunications and Multimedia, Power Systems, Manufacturing Systems Engineering, Information and Communication Systems and Optoelectronics

INESC Porto is an institution created to act as an interface between the academic world, the world of industry and services, as well as the public administration, in the framework of the Information Technologies, Telecommunications and Electronics (ITT&E). Its activities range from research and development, to technology transfer, consulting and advanced training.

More specifically this project was developed at the Telecommunications and Multimedia Unit (UTM). UTM carries out Research&Development activities in some key areas that lay the foundations of the modern communications networks and services namely network architectures, telecommunications services, signal and image processing, microelectronics, digital TV and multimedia. UTM has been working on the MIR area for several years participating in projects like VISNET II [10], Sonic Interaction Design [11] and Digitópia [12] that is a platform for the development of musical creation using computers.

1.1.3 Personal Trajectory

The work presented in this dissertation spans over 4,5 months and it gave me the chance to apply some knowledge I obtained during my private, professional and student life.

The inclination for music started in my teenage years while listening to bands such as Queen and Nirvana. As the years passed I naturally broadened my musical horizons by exploring other musical styles, by starting to learn classic guitar and generally discuss music related topics. Despite this I never considered working or studying in the music research area until the 5th year of my Integrated Masters Degree where I developed a project on digital signal processing techniques applied to voice analysis. Other courses were very useful for the conclusion of this project like Instrumentation and Measurements or Digital Signal Processing allowing me to understand some signal processing algorithms.

Parallel to student and personal life, I also gradually gained experience in acoustic propagation, sound devices operation among others by working on the Auditorium of FEUP while preparing concerts, conferences and plays.

1.2 Software Tools

On this section the software applications used for the development of this project will be presented. A general explanation will be given while its specific use will be addressed in future sections.

1.2.1 MARSYAS

Marsyas (Music Analysis, Retrieval and Synthesis for Audio Signals) is an open source software framework for audio processing with specific emphasis on Music Information Retrieval applications. The basic goal is to provide a general, extensible and flexible architecture that allows easy experimentation with algorithms and provides fast performance that is useful in developing real time audio analysis and synthesis tools. A variety of existing building blocks that form the basis of most published algorithms in Computer Audition are already available as part of the framework and extending the framework with new components/building blocks is straightforward. This means that the user can either create complex systems using several existing processing blocks or he/she can adapt a specific block to his/her desires. MARSYAS can be used in Linux, Mac Os and Microsoft operating systems. MARSYAS is not a standalone application and so it depends on other applications to work properly. In order to use this application on Windows OS one needs Microsoft Visual Studio which is an IDE (integrated development environment) application. MARSYAS has been designed and written by George Tzanetakis with help from students and researchers from around the world and can be found at [13].

As an example let's consider a basic amplifier .The objective is to insert an input audio file, pass it through a gain block and play it. It's easy to observe that 3 separate blocks will have to be used as depicted in Figure 1:

- Input block- imports the audio file (in this case a WAV file).
- Gain block- amplifies the received signal.
- Play block – connects with the computer´s audio card to play the received sound file.

All of the blocks mentioned already exist in MARSYAS, so instead of writing manually the code, one has only to "call" them. The code used for this example was:

```
1- MarSystem* playbacknet = mng.create("Series", "playbacknet");
2- playbacknet->addMarSystem(mng.create("SoundFileSource", "src"));
```

3- playbacknet->addMarSystem(mng.create("Gain", "gt"));
4- playbacknet->addMarSystem(dest);

To do this one simply has to create a chain (in this case it's called playbacknet) as can be seen in line 1 of the previous paragraph and insert the blocks into it (lines 2,3 and 4). Each of the called blocks corresponds to a cpp file that already exists on the MARSYAS library. So in the second line the soundfilesource.cpp is called, in the third line, the gain.cpp file is called. The cpp files contain the code that enable the blocks to work.

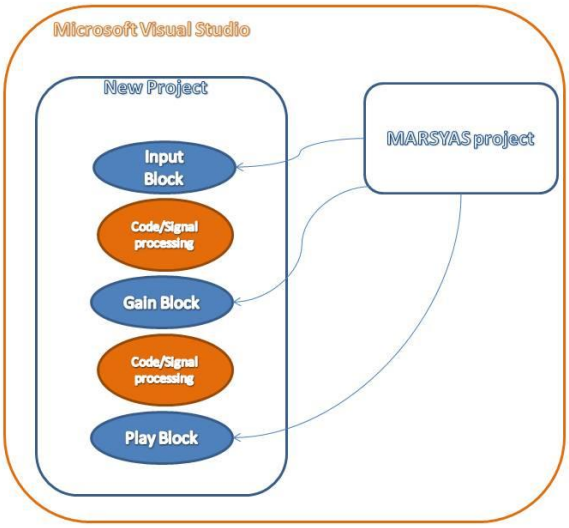

Figure 1:Example of a MARSYAS project using some of its components

On this project MARSYAS will be responsible for processing the music signals using several available functions. The specific use of MARSYAS will be explained in chapter 3.

1.2.2 WEKA

WEKA is open source data mining software used for machine learning written in Java and its name comes from "Waikato Environment for Knowledge Analysis". This application was developed at the University of Waikato in New Zealand and runs under Linux, Windows and Macintosh operating systems. The WEKA workbench is a collection of machine learning algorithms and data preprocessing tools and on this project will be responsible for interpreting and classifying the resulting data represented as coefficients of several extracted features. In WEKA, the user can choose several operation modes but for this project only two were used, the Explorer and the CLI (Command Line Interface). The Explorer provides a visual interface enabling the user to see visual representations of the analyzed data as illustrated in Figure 2. This enables the user to better understand the data he/she is working with. It also becomes easier to use because all of the options are available with the respective help window.

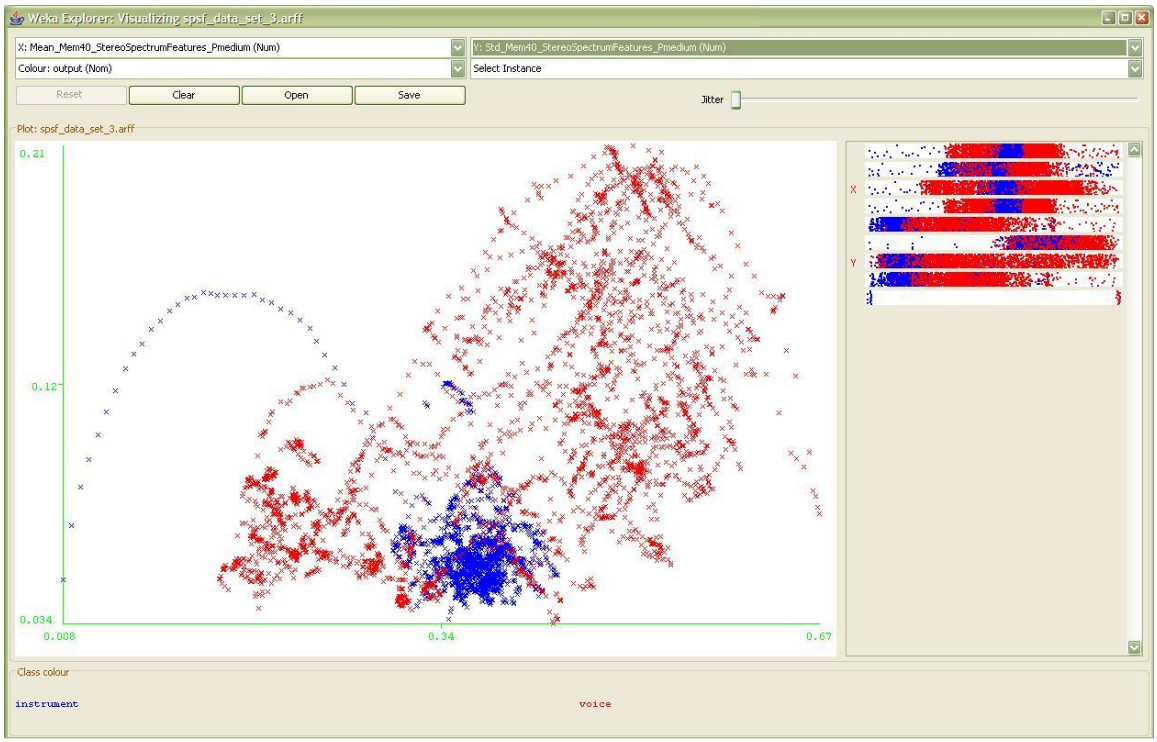

Figure 2: Visual representation of 2 attributes using WEKA Explorer

One downside of the Explorer is the limitation to use only one data file at a time so if the user intends to process several different data files the best option is to use the CLI. It can be accessed through WEKA or through an operating system´s command-line interface by running the desired commands using a java engine and the weka.jar file.

1.2.3 MATLAB

MATLAB, short for matrix laboratory is a numerical computing environment that was invented in the late 1970s by Cleve Moler, by then chairman of the computer science department at the University of New Mexico. MATLAB is a well known application and it´s frequently used in the industry and academic worlds. As such it has known a constant upgrading and updating by Math Works. MATLAB allows easy matrix manipulation, plotting of functions and data, implementation of algorithms, creation of user interfaces and interfacing with programs in other languages. In this project, MATLAB will be used for file editing and managing purposes that will be made clear further ahead.

1.2.4 Other Software Used

During the course of this project, other software programs were used. One of them is the very useful Audacity audio software. Audacity is a free, open source software for recording and editing sounds. It is available for Mac OS X, Microsoft Windows, GNU/Linux, and other operating

systems and can be found at [14]. In this project it was useful for the manipulation of the data sets audio files and its specific use will be explained in chapter 3.

Another very important application that needs to be mentioned is the Windows command line because it enabled through the use of BAT execution files the processing of several MARSYAS and WEKA functions without user interaction.

1.3 Structure of the Dissertation

Concerning the structure of this document, it´s divided in 6 chapters. In this chapter, the context of the research was introduced, the aims of the dissertation were described and the relevant contributions will be identified. Chapter 2 focuses on explaining the area of singing voice detection by considering work developed by other authors. Chapter 3 presents the general voice detection framework and deals with a specific explanation of the used data sets and the general classification approach. Chapter 4 describes the analysis techniques and the experiments to be executed while on Chapter 5 the results obtained are discussed. Chapter 6 contains a small revision of the developed work, identifies the relevant contributions of this dissertation and proposes future work to be developed.

1.4 Relevant Contributions

The most important contribution of this thesis is the understanding of which techniques are useful for singing voice detection in polyphonic music signals. For musical genres that have a low frequency of change, the increase of the texture window can be an interesting approach. For the independent use of features, the MFCC proved to be the most useful achieving classification results ranging between 79% and 85%. The feature combination revealed to be a good technique given the right conditions achieving for some cases results around 89% of correctly classified instances. Another important contribution of this work was the creation of the Allegro data set which can be used for future experiments by other investigators.

Chapter 2

2. Singing Voice Detection Survey

In this chapter, a description of the current state of voice detection will be given. In this work a general framework for voice detection in polyphonic music will be tested in order to better comprehend what are the limitations and what can be improved in the current existing techniques. The structure of this framework was based on the work of several authors and can be divided into two main fields which will be explained on the following sections:

- Voice enhancement by signal processing
- Feature extraction and classification

2.1 Voice enhancement by signal processing

One way to detect voice in music signals is to highlight certain characteristics of the voice components. An example is the use of a chebychev filter followed by a comb filter suggested in [3]. The author first tries to detect energy on the frequency bands where voice is usually present using a chebychev filter of 12^{th} order. The problem with this approach is that other instruments also fall on this frequency bands and are not attenuated. So, in order to effectively separate the voice signals the author used an inverse comb filter that is based on the fact that 90% of the voice is composed of voiced sounds that are highly harmonic. Considering the fact that some instruments that were not attenuated with the chebychev filter can also be harmonic, this filter also differentiates how these instruments spread on the spectrum in the same way and calculates a harmonicity value that will be used for determining the class of the considered segment. Results showed this is not a good technique to be implemented. One vulnerability of this technique consists on the simple threshold used since it's difficult to define the exact boundaries for the decision.

Although proven not to be successful in voice detection, voice enhancement algorithms are very useful as preprocessing techniques making feature extraction easier. This concept will be explained on the next section.

2.2 Feature extraction and classification

This framework is an alternative to the one mentioned in the previous section. This does not exclude, as already mentioned, the use of some voice enhancement techniques for the improvement of feature extraction. There are several methods of detecting voice segments in music that consist on the extraction of features and its respective classification. The basic idea is to extract from the music signals characteristics that enable a classifier to determine to which class, in this case voice or instrument, a determined segment corresponds to.

Two options for this framework are featured in Figure 3 and some examples will be given so that the process becomes clearer. One method extracts the features from the music files and classifies them using a classifier. The second method only differs from the first one because the music files are first preprocessed and only then the features are extracted.

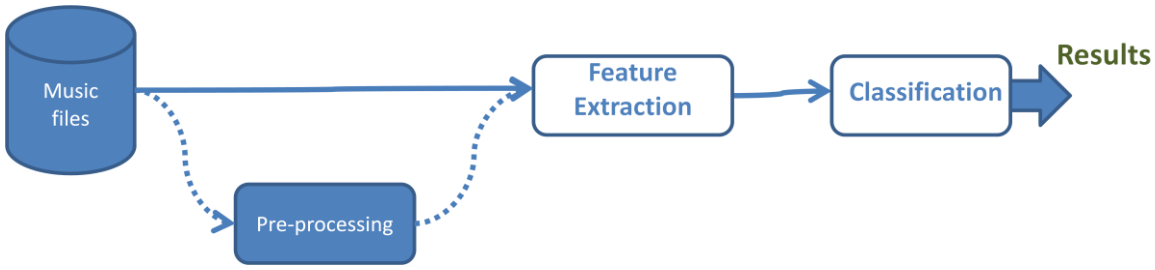

Figure 3: Block diagram of the voice detection model

In [15], the authors aim to calculate feature vectors that represent vocal characteristics and in order to achieve this they need to extract certain features from the music signals. In the attempt to help the extraction process, the authors first execute an accompaniment sound reduction that consists on the reduction of non vocal sounds based on their harmonicity. On the feature extraction phase, the authors used two features being one the LPC-derived mel cepstral coefficients. After the creation of the features vectors, comes the reliable frame selection that is basically a classification process that tries to securely eliminate frames that do not contain voice. In [15], three separate stages were used corresponding to the scheme presented in Figure 3 being the preprocessing block the voice enhancement process, the feature extraction block being the creation of the feature vectors and the classification block being the reliable frame selection.

This same structure is used by other authors although with different processes in each block as in [16] where a preprocessing technique is presented that consists on the resynthesis of the input signal based on its most prominent peak clusters. This technique will be explained in detail in chapter 4. After this preprocessing phase, the authors extract the MFCC coefficients from the resynthesized signal and lastly they use them for classification with ZeroR, Naïve Bayes and SVM classifiers. Another example of a voice detection framework that uses preprocessing is presented in [17] where the authors use an Autoregressive Moving Average (ARMA) filter as a voice

enhancement technique. This approach uses the MFCC coefficients as features and the GMM as a classifier achieving results for voice frame detection around 75% and for instrument frame detection around 90%.

In [18] the authors aim to detect the onsets of the syllables sung and the respective pitch of vocal signals so that a process of automatic lyric alignment of cantonese songs can be correctly executed. The authors propose three stages, an initial voice enhancement stage, followed by an onset detector and finally a classifier for determining the vocal onsets. The first stage is useful for suppressing the instrument signals and enhancing the vocal signals while the second stage detects the start times/onsets of the syllables sung. Finally on the third stage the non-vocal onsets are pruned by a voice detection classifier.

The four mentioned examples use preprocessing techniques to help the feature extraction but not all authors adopt this procedure. Other investigators prefer to extract features directly from the music signals and supply them to the classifiers like in [19], where the authors extract 13 types of features from the music signals without any prior processing technique. After the extraction of the features, they are supplied to several classifiers, the multidimensional Gaussian maximum a posteriori estimator, a Gaussian mixture model (GMM), a spatial partitioning scheme based on kd trees and the nearest neighbour.

Another example is presented in [20] where the authors explore stereo panning information in order to identify the music style. They extract the panning features along with the MFCC and test the results by using 4 different classifiers, ZeroR, Naïve Bayes, SMO and J48.

Another example of the use of this framework is presented in [2] where the authors aim to automatically identify singers using the MFCC coefficients as features and the GMM for classification. Before trying to identify the singer, the authors applied a vocal frame selection operation so that the vocal frames could be safely identified. The report presented by these authors is relevant because the data set used by them will also be used in this project with the name of Greek data set and will be presented in chapter 3. Unfortunately the result comparison cannot be achieved because the authors use only 21 songs of the data set and they also do not reveal which ones. Another problem that prevents the comparison is the fact that the authors only care about finding some vocal segments in each song and not all of the vocal segments. This happens because they only need some samples for extracting singer characteristics. Knowing this, they choose to safely identify the voice segments by minimizing the number of frames labelled as vocal that were in fact instrumental frames and not caring if vocal frames were identified as instrumental.

In [21] the authors use a data set for locating singing voice segments that is similar to the Ellis data set used on this project. On the mentioned report, the authors present a detector of speech like sounds based on an acoustic classifier in order to discriminate singing voice from vocal accompaniment achieving accuracy of 80%. The authors investigate the idea that a speech-trained acoustic model can respond differently to singing voice and instruments. They use several features like Posterior Probability Features (PPF), PLP cepstral coefficients among

others and a HMM (Hidden Markov Model) framework for frame classification. The results obtained in this report will be compared with the ones achieved on this project in chapter 5.

In [22], the authors compare audio descriptors for singing voice detection in music audio files. The authors extract several features like the LPC, MFCC, and Zero Crossing Rate (ZCR) among others and test them using several classifiers like the SVM, decision trees and the nearest neighbour.

In this section a framework consisting of feature extraction and classification blocks was presented based on the work of other authors. In addition to this simple framework, some authors chose to use a preprocessing block to help the feature extraction. Analyzing the results obtained by the mentioned authors, it was not possible to ascertain which approach was better.

Since none of these two approaches revealed to be better than the other, the framework that will be used in this dissertation will be based on both as depicted in Figure 3. Some experiments will employ preprocessing techniques while some will not. The specific processes for each block will be explained in the next chapter.

Chapter 3

3. Method

In this chapter a detailed explanation of the framework will be given while presenting the data sets and the classification methods. The framework that will be used was already explained on the previous chapter and is depicted in Figure 3. The framework presented earlier is a general view of how the voice detection sequence should work and for this project a more specific model was used. Considering that same framework, the used applications for the experiments are depicted in Figure 4. It´s possible to see that the framework structure does not change but each block was adapted. The feature extraction block will be executed in MARSYAS and the classification block will be executed in WEKA. The preprocess block is not present in Figure 4 because it will not be used in every experiment. Considering the first block, the feature extraction, it´s important to explain how MARSYAS extracts features from the music signals.

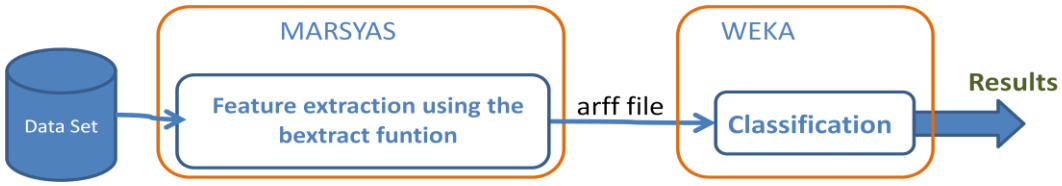

Figure 4: General Framework design for this project

The bextract function in MARSYAS has the ability to extract features from multiple music files and save them in a file with the arff format. In truth, the bextract function only accepts one file as input so instead of processing one music signal at a time, collections were used. Collections are files that contain the path and name of music files along with its respective class like for example “c:\song1 voice”. To create the voice and instrument collections the mkcollection function available in MARSYAS was used. These collections were used by the bextract function to identify the audio files and the respective label so that the features could be extracted. The bextract function saves the feature extraction results in arff (Attribute-Relation File Format) files according to the audio file class as illustrated in Annex A. An arff file is an ASCII text file that describes a list of instances sharing a set of attributes and corresponding to a certain output. The example in Annex A contains the attributes which are the mean and the standard

deviation of the LSP coefficients and the outputs are the voice and instrument classes. The first section is the header and contains the attributes and the output classes. After the header come the actual data values or the instances that begin after the line “@Data”. Each line corresponds to an instance containing the values of the attributes extracted from the analyzed texture window. In the end of each line, or instance, the respective output class is present. Some of these concepts will be made clear later on. This function allows some parameters to be altered like the extractor type, the analysis window size, the hop size and the texture window size. To better understand these concepts let´s consider the following example.

The first operation is to open the data files to be processed, then for each file, bextract computes for a user defined number of samples (analysis window) the desired coefficients depending on the user selected extractor. There are several extractors like the MFCC, the LPCC and the LSP. For the next window, bextract advances a defined number of samples (hop size) and executes the same operation. As is depicted in Figure 5, each analysis window (red lines) has 1024 samples and the overlap is consisted of 512 samples (hop size of 50% represented by the green line).

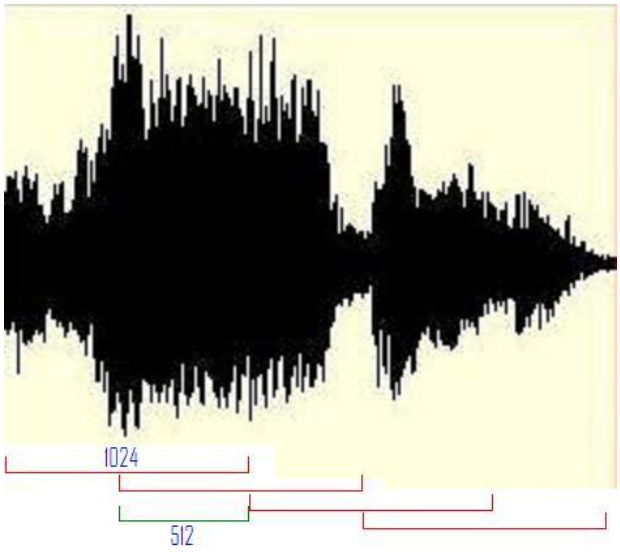

Figure 5: Window size and Hop size

Although each window has their respective feature values, they are not meaningful if the analysis window is small (1024 samples correspond to 23,2 milliseconds for 44100Hz signals) since music has a variation standard of hundreds of milliseconds. Having this in consideration, bextract calculates the average and the standard deviation of m analysis windows creating the texture windows (named memory size in bextract), as depicted in Figure 6.

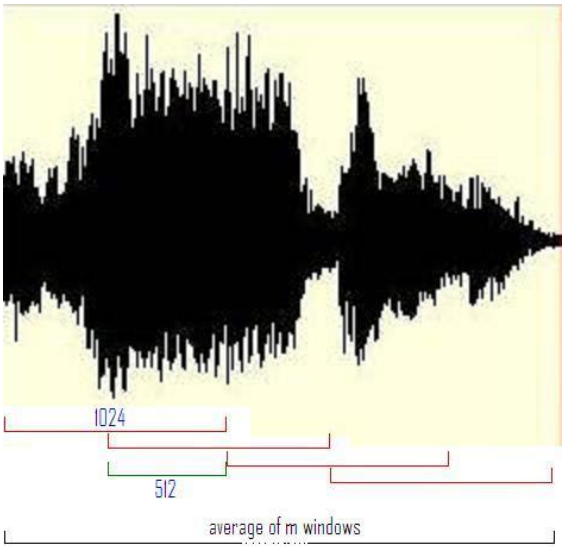

Figure 6: Texture Window

After the mean and the standard deviation are obtained, bextract saves the results in an arff file with the respective class (instrument or voice) of the analyzed segment. This operation is repeated until the coefficients for the last analysis window and respective texture window are calculated and saved in the arff file thus ending the feature extraction operation.

After the creation of the arff file containing the extracted features, the classification step occurs. On this final step, the arff file is analyzed and classified using the classifiers available on WEKA. This operation will generate a report with the classification results and also some other useful information about the classification process.

In this introduction of the third chapter, the basic structure for the executed experiments was described, however some experiments will have slightly different structures. On the following section a description of the data used for the experiments will be given.

3.1 Data Sets

The data sets on this work were used for the construction of the classification model with the supervised learning and also for the testing of the respective classifiers. The mentioned operations cannot be accomplished with any data set therefore the used ones had specific characteristics. Since the objective of this project is to distinguish voice from other sounds in music, it is necessary that the data sets respect that demand by having each of these types of segment correctly separated. For the voice class, the segments may contain any type of sound as long as there is singing voice present. For the other class, the instrument class, the segments may contain silence and any instruments as long as no singing voice is present. Another requirement for the data sets was the use of wav files because MARSYAS works better with these types of files.

In order to obtain more results, three different data sets were used. One advantage of using several data sets is the opportunity of safely comparing results with experiments where those data sets were used thus giving a better understanding of the value of the used parameters. At the same time, using these data sets will enable future investigators the comparison with the results that will be presented in this report. Another advantage of using several data sets is the chance to work on a more global scope and not only with the specific data of each data set. This way it´s possible to declare that a certain method works for several types of data and not only with a controlled and limited one.

3.1.1 Greek Data set

This data set is part of a data set used by Andre Holzapfel and Yannis Stylianou in [2], and can be obtained at [23]. For this work it will be called the Greek Data Set and is composed of 59 stereo wav files for each class (voice and instrument). The songs have varied length and were sampled at 44100 Hz with 16 bits. The songs belong to popular Rembetiko singers which causes the dataset to be somewhat homogeneous and it also contains a lot of noisy recordings because some of them were digitalized from old gramophone disks. This data set was originally formed by mp3 music files with no voice separation. All the mp3 files contained a respective lab file that identified the vocal and instrument segments. For use in the experiments an adaptation had to be made. This adaptation consisted on the conversion of the files from mp3 to wav using audacity and the separation of the vocal and instrument segments of each song using MATLAB functions. The conversion from mp3 to wav was executed because some MARSYAS functions work better with wav files and because a special plug-in would have to be installed for mp3 processing.

After having obtained the wav files, the next step was to separate the files in voice and instruments classes in accordance to the respective labels. The original labels were stored in a .lab file, as shown in Figure 7. The first column corresponds to the beginning of the segment while the second corresponds to the end of the segment being both represented in seconds. The third column represents the label of the segment.

```
0.0000000 1.0815493 SIL
1.0815493 27.0808711 INSTR
27.0808711 53.0380546 VOICE
53.0380546 77.1832919 INSTR
77.1832919 102.8314613 VOICE
102.8314613 105.1069287 OTHER
105.1069287 127.1031135 INSTR
127.1031135 151.9787477 VOICE
151.9787477 152.8355595 INSTR
152.8355595 154.6896440 OTHER
154.6896440 176.2082615 INSTR
176.2082615 202.1654450 VOICE
202.1654450 203.6514539 SIL
```

Figure 7: Example of a label for the original Greek data set

In total, 8 different types of labels existed as can be seen in Table 1.

Table 1: Greek Data Set original labels

Labels	Description
SIL	Consisted of silence
INSTR	Music sections were only instruments are present
VOICE	Sections that contain voice which can be accompanied by music or not
VOICE2	The same singer(voice) singing in a different manner
OTHER	Music sections that contain people talking, screaming or clapping
MIXED	Sections where two people are singing with or without instruments
SPEECH	Sections where people are talking without instruments
VOCONL	Sections where there is only the singer either talking or singing without instruments

These labels where used for experiments that had a different objective than this project and therefore should not be used in the same way. Since the objective is to differentiate vocal sections from non-vocal sections the use of two labels for future experiments was enough. The selected labels are listed in Table 2.

Table 2: Used labels for Greek Data Set

Labels	Description
INSTRUMENTS	Sections that don't contain any type of voice, usually consisted of instruments or silence
VOICE	Sections contain singing voice, with or without instruments

In order to create these two labels it was necessary to adapt the existing eight labels from the original data set as is depicted in Figure 8.

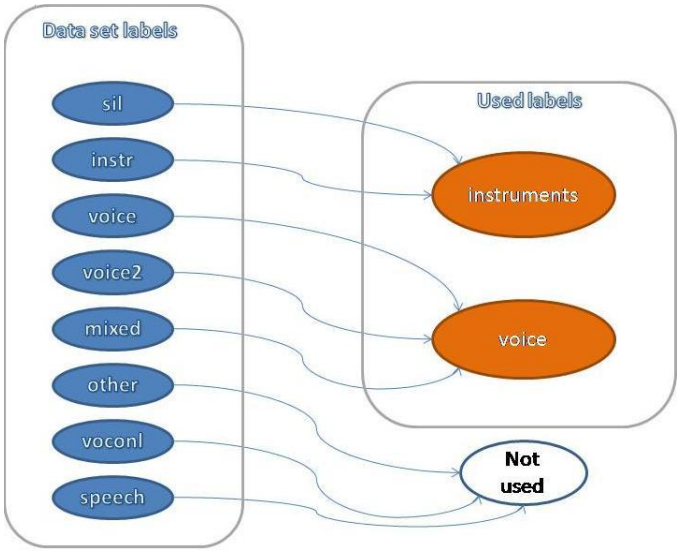

Figure 8: Label conversion for the new Greek data set

After the conversion to wav and the label adaptation, the next step was to separate the data set into two separate audio sets, the vocal audio set and the instrument audio set. One could separate de audio files into its different sections obtaining for each song several separate segments but instead the sections types in each song were added up. This way, each song would be divided into two parts, one consisting of the vocal sections and the second of the non-vocal sections. So in accordance to the respective labels, the sections could be added to one of the audio files of each class or it could be deleted. In Figure 9 is an example of a song that will belong to the voice class of the respective data set.

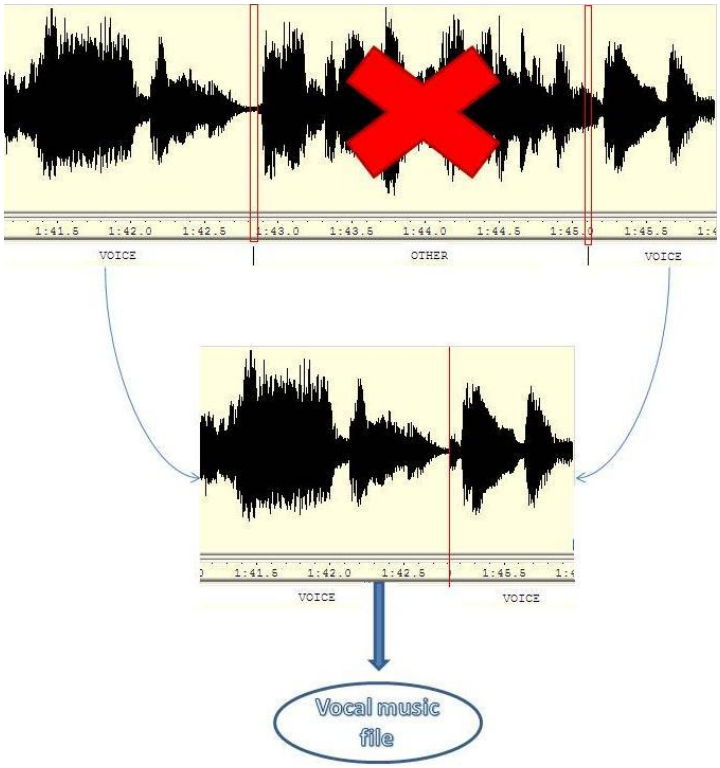

Figure 9: Example of the creation of a vocal wav file by adding two non consecutive segments

Since the labelling was done manually, it's wise to assume that the human error factor could be present. So, in order to minimize it, a window of 75ms before and after each section was ignored as can be seen in Figure 10.

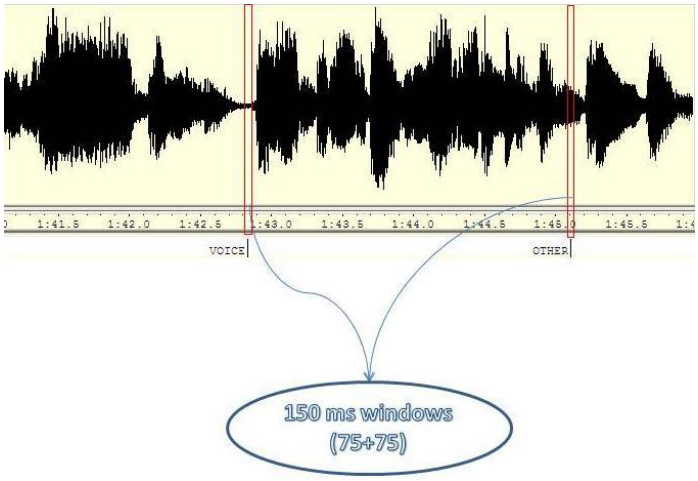

Figure 10: Ignored segments in music files due to presumable human error

In conclusion, the resulting data set consists of two classes of 59 wav files each and the detailed list can be obtained at [24]. The vocal class occupies 980MB of disk space while the in-

strument class occupies 900MB. The execution of the explained separation process was made possible by MATLAB and the used m file can obtained at [24].

3.1.2 Ellis Data Set

The second data set can be called Ellis Data Set. This data set was originally comprised of some 60 15-second extracts collected randomly from the radio by Eric Scheirer in 1996 and later divided into a set of music with vocals and a set of music without vocals by G. Williams & D. Ellis in 1999. This data set can be found at [25]. All the files are mono with 32 bits and sampled at 22050 Hz. The music files contain several different instruments since a lot of musical styles are represented such as jazz, classic, pop, bossanova, rock, electronic, among others. Some of the files in the music with vocals section are ill divided since some segments varying from 0,5 to 1 second do not contain vocal sounds. This fact may compromise future results.

3.1.3 Ellis Data Set (corrected)

This data set is almost equal to the previous one only diverging in the section of music with vocal. In truth this data set was constructed because of the flaws mentioned in the previous data set. In total, 6 files were altered in the attempt to correct the section of music with vocals. A more detailed description of the altered files can be found in Annex B.

3.1.4 Allegro Data Set

This data set was created by Pedro Allegro at INESC Porto during the development of this thesis. The wav files that comprise this data set were sampled at 44100 Hz with 32bits. Two types of sections were created as in the previously described data sets. One section, voice, possesses wav files containing singing voice accompanied or not by instruments. The other section, instruments, is comprised of wav files containing instruments with no voice and silences. The wav files were created from several different genres of songs such as rock, pop, reggae, fado, MPB (popular Brazilian music), among others. These files were divided with the use of audacity by hearing each song and extracting the desired segments. The reason for the creation of this data set came from the need to have more stereo data to compare with the Greek data set since the Ellis data set only possesses mono wav files. Also, and if the data is not corrupted it is better to have another data set to challenge the results thus giving them more credit. A detailed description of this data set can be found at Annex C and it will be made available at [24] for future comparison of results.

3.2 Machine Learning

Machine learning can be seen as a field of artificial intelligence, where the design and development of algorithms and techniques that allow computers to learn are the essential objec-

tives. For the development of this work, machine learning techniques were executed with the WEKA application and were responsible for the construction of the classification model and subsequent evaluation using features extracted from the music files. The construction of classification models under a labelled set of data is called supervised learning. “The learning scheme is presented with a set of classified examples from which it is expected to learn a way of classifying unseen examples” [26]. The data given to the classifier is divided into classes predefined by the user being them in this project the vocal and instruments classes. The resulting classification model will be used later for the actual classification of the data.

3.2.1 Cross Validation

Regarding the construction of the classification model and its respective evaluation, there are several ways it can be executed. One way is to use all the data for the construction of the classification model and for the actual classification process. Another approach is to divide the data set into two separate parts where one is used for the construction of the classification model and the other one for the classification process. Another approach is the one applied on this project, the cross validation. Cross validation consists on partitioning a sample of data into n subsets such that the analysis is initially performed on a set of (n-1) subsets constructing the classification model, while the remaining subset is retained for subsequent use in classification. This operation is repeated for each partition and the final result consists on the average of the classification obtained by all the partitions as depicted in Figure 11. On this example, the data is partitioned into 3 groups corresponding to a cross validation value of 3. After the partition three different classification processes take place corresponding to the red, green and blue lines. For each of these processes two groups represented by the dotted lines are used to train the classifier by building the classification model. The remaining group is used for classification using the created classification model. Finally the average of the three obtained results is calculated.

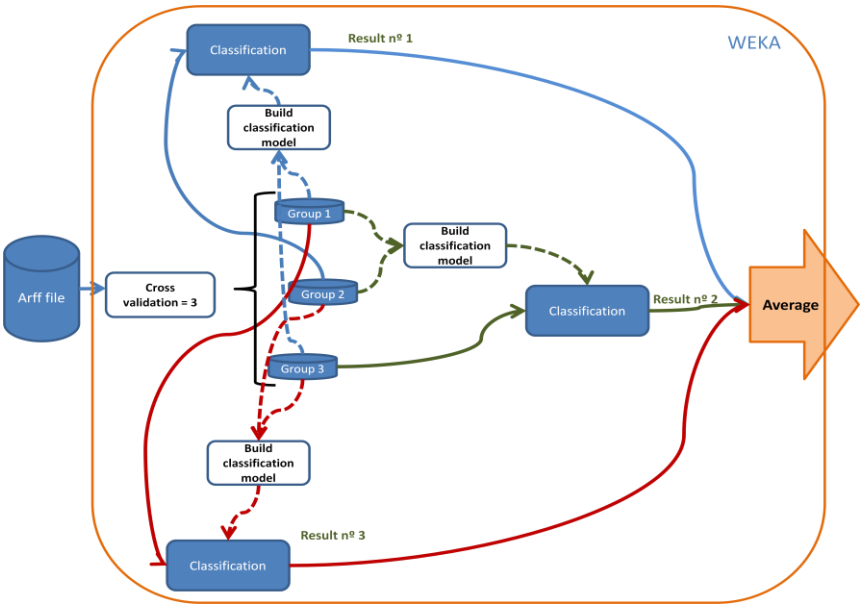

Figure 11: Block diagram for cross validation

The results represent the errors made on the instances used to build the classification models. This means that what is measured is the performance of the classifier on the supplied data because there is no way to do it for undocumented data, "…what we are interested in is the likely performance on new data, not the past performance of old data." [26].

These concepts will be explored in the rest of this section along with the description of the used classifiers.

3.2.2 Classifiers

In classification, a set of example records called a training set, where each record consists of several fields or attributes is considered. Attributes are either numerical (coming from an ordered domain), or categorical (coming from an unordered domain). One of the attributes, called the class label field (target field), indicates the class to which each example belongs. The objective of classification is to build a model of the class label based on the attributes of the example records. After a model is built, it can be used to determine the class label of unclassified records.

In the rest of this subsection, the SVM, Naïve Bayes and the J48 classifiers will be presented since they were the ones used for classification in the conducted experiments. On the first experiments the nearest neighbour was used for classification but was later discarded due to the excessive time required for computation. More specifically the K-nearest neighbour was used with the classification model being constructed considering the 3 nearest neighbours for each instance. This task would take an enormous amount of time to be completed especially for the Greek data set because of its size.

3.2.2.1 Naive Bayes

Naive Bayes is a simple probabilistic classifier that uses all of the attributes allowing them to make contributions to the decision that are equally important and independent from one another. For a given input, the classifier multiplies the fractions of each attribute obtaining a likelihood value for each output class. The fractions of each attribute are calculated by dividing the number of the occurrences correspondent to the input attribute and output class and dividing them for the total number of occurrences in that class. The higher class likelihood determines the output of the input. If one input values is not present in the training instances, then the classifier calculates its value through a Gaussian probability distribution. As an example let´s consider the weather case where there are 5 attributes and the classes are yes and no. In Figure 12, the frequencies and probabilities for the weather data training set are available.

outlook			temperature			humidity			windy			play	
	yes	no		yes	no		yes	no		yes	no	yes	no
sunny	2	3	hot	2	2	high	3	4	false	6	2	9	5
overcast	4	0	mild	4	2	normal	6	1	true	3	3		
rainy	3	2	cool	3	1								
	yes	no		yes	no		yes	no		yes	no	yes	no
sunny	2/9	3/5	hot	2/9	2/5	high	3/9	4/5	false	6/9	2/5	9/14	5/14
overcast	4/9	0/5	mild	4/9	2/5	normal	6/9	1/5	true	3/9	3/5		
rainy	3/9	2/5	cool	3/9	1/5								

Figure 12: Frequencies and probabilities for the weather data [http://homepages.inf.ed.ac.uk/keller/teaching/connectionism/lecture10_4up.pdf]

Now assume that we have to classify the following new instance:

- outlook : sunny
- Temp. : cool
- humidity : high
- windy : true
- Play:?

The main objective is to compute a probability for each class based on the probability distribution in the training data. First it's necessary to take into account the probability of each attribute, knowing they are equally important, by multiply the probabilities $P(yes) = \frac{2}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{3}{9} = 0{,}0082$ and $P(no) = \frac{3}{5} \times \frac{1}{5} \times \frac{4}{5} \times \frac{3}{5} = 0{,}0577$. Now taking into account the overall probability of a given class and multiplying it with the probabilities of the attributes $P(yes) = 0{,}0082 \times \frac{9}{14} =$

$0{,}0053$ and $P(no) = 0{,}0577 \times \frac{5}{14} = 0{,}0206$, the class that maximizes this probability is chosen. In this case it means that the new instance will be classified as no.

3.2.2.2 Support Vector Machine

The Support Vector Machine (SVM) is based on the concept of decision planes (hyperplanes) that separate the data according to their classes. The hyperplanes try to leave the largest possible fraction of instances of the same class on the same side. At the same time they try to maximize their distance to the classes so the risk of misclassifying is reduced. A simple example can be seen in Figure 13 where the squares represent one class and the circles represent another class.

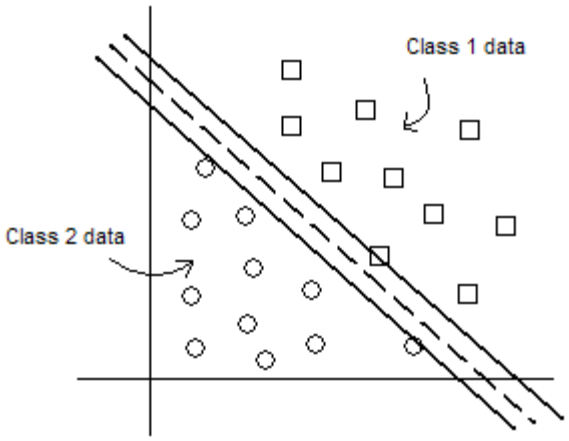

Figure 13: 2-D representation of the SVM algorithm[http://cnx.org/content/m13131/latest/]

The simplest way to divide two groups is with a straight line, flat plane or an N-dimensional hyperplane, but if the points are separated by a nonlinear region such as depicted in Figure 14 a non linear line for class division is needed.

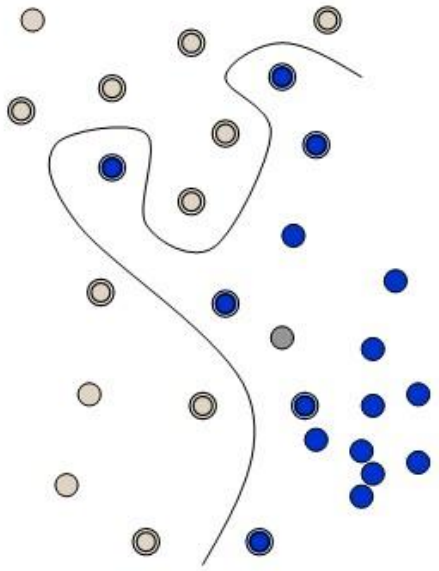

Figure 14: Second 2-D representation of the SVM algorithm[http://www.dtreg.com/svm.htm]

Rather than fitting nonlinear curves to the data, SVM handles this by using a kernel function to map the data into a different space where a hyperplane can be used for the separation.

3.2.2.3 Decision Tree

A decision tree is a simple structure where non-terminal nodes represent tests on one or more attributes and terminal nodes reflect decision outcomes. The tree is built by recursively partitioning the data. Partitioning continues until each partition is either pure (all members belong to the same class) or sufficiently small (a parameter set by the user). The initial lists created from the training set are associated with the root of the decision tree. As the tree grows and nodes are split to create new children, the attribute list for each node is partitioned and associated with the children.

A decision tree classifier is built in two phases:

- A growth phase
- A prune phase

After the initial tree has been built (the growth phase), a sub-tree is built with the least estimated error rate (the prune phase). The process of pruning the initial tree consists of removing small, deep nodes of the tree resulting from noise contained in the training data, thus reducing the risk of overfitting, and resulting in a more accurate classification of unknown data.

While the decision tree is being built, the goal at each node is to determine the split attribute and the split point that best divides the training records belonging to that leaf. The value of a split point depends on how well it separates the classes. As an example consider a tree that was constructed from the training set represented by the table in Figure 15.

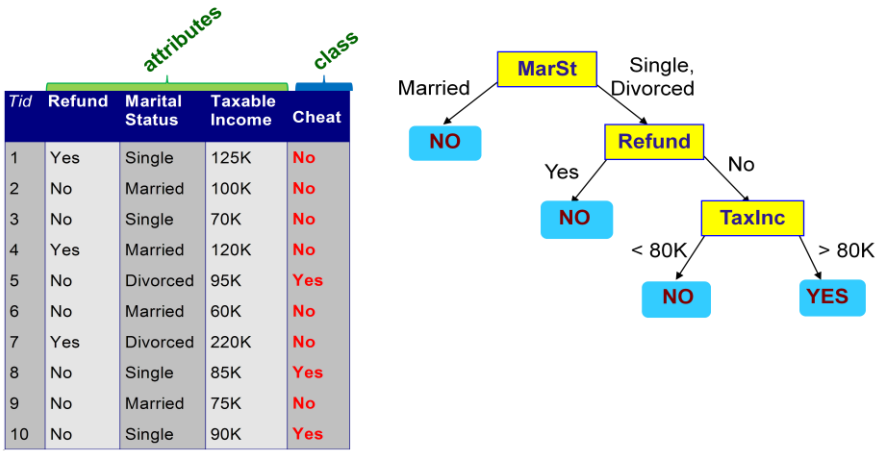

Figure 15: Example of training model for tree classifier. http://www-users.cs.umn.edu/~kumar/dmbook/dmslides/

After the tree is constructed, the test data is supplied to the tree classifier and flows through the nodes until it reaches the end or a decision that for this example is “NO”, as can be seen in Figure 16.

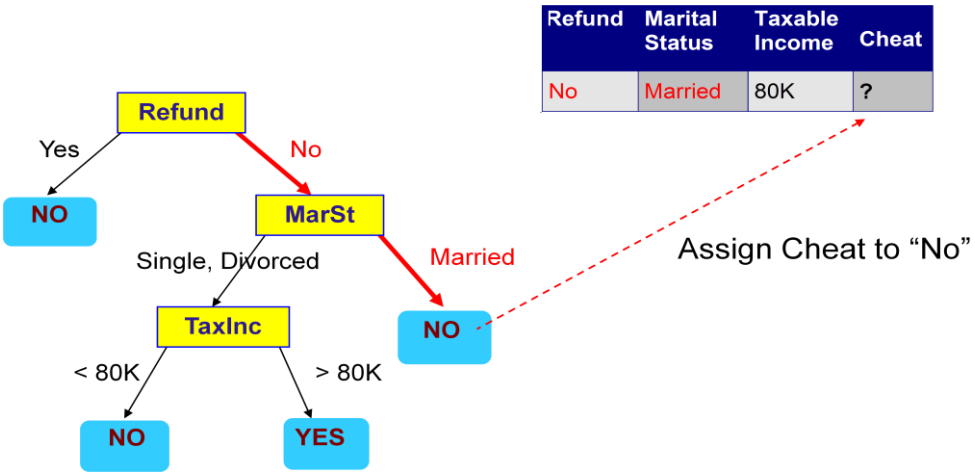

Figure 16 Example of test model for tree classifier. http://www-users.cs.umn.edu/~kumar/dmbook/dmslides/

This is a visual representation of a tree, but in WEKA the trees are presented in another manner. As an example, let´s consider the tree presented in Figure 15. The respective representation in WEKA would be the following:

```
Refund = no
| MarSt = Married: no
| MarSt = Single,Divorced
| | TaxInc <= 80k : no
| | TaxInc > 80k : yes
Refund = yes : no


Number of Leaves :  4

Size of the tree :  7
```

Each line represents a node in the tree. The second and third lines, those that start with a '|', are child nodes of the first line. In the general case, a node with one or more '|' characters before the rule is a child node of the node that the left-most line of '|' characters terminates at. The next part of the line declares the rule. If the expression is true for a given instance, it is either classified if the rule is followed by a double dot and a class designation (that designation becomes the classification of the rule), or if it is followed by a semicolon it continues to the next node in the tree (i.e. the first child node of the node that was just evaluated). If the expression is instead false, it continues to the “sister" node of the previously evaluated node (the node that has the same number of '|' characters before it and the same parent node).

Nodes that generate a classification, such as

```
| MarSt = Married: no (161.0),
```

are followed by a number (sometimes two) in parentheses. The first number tells how many instances in the training set are correctly classified by this node, in this case 161 are. The second number, if it exists (if not, it is taken to be 0.0), represents the number of instances incorrectly classified by the node.

The pruning phase occurs after the tree is built. The pruning of the decision tree can be done by replacing a whole subtree by a leaf node. The replacement takes place if a decision rule establishes that the expected error rate in the subtree is greater than in the single leaf. Another method for pruning is the subtree raising which is more complex. Basically it consists on “raising” a subtree by replacing the parent node from where that subtree origins from with the subtree. To ensure that this replacement does not cause many errors it is usually restricted to when the subtree has more examples on the training data than its brothers (the one to be eliminated) from the parent node. This operation is time consuming and not always worthwhile.

On the experiments, the C4.5 tree classifies, known as J48 in WEKA will be used. C4.5 relies on the fact that each attribute of the data can be used to make a decision that splits the data into smaller subsets. C4.5 examines the normalized information gain (difference in entropy) that results from choosing an attribute for splitting the data. The attribute with the highest normalized information gain is the one used to make the decision. The algorithm then recurs on the smaller sublists.

On this algorithm some important options should be considered:

- Confidence factor- determines the confidence value to be used when pruning the tree (removing branches that are deemed to provide little or no gain in statistical accuracy of the model). The default of 25% works reasonably well in most cases. However, if the actual error rate on real data (or the error rate on cross-validation) is significantly higher than the error rate on the training data, the confidence factor should be decreased in order to cause more drastic pruning, and a more general model of the data.
- minNumObj- determines the minimum number of instances that must be present in the training data for a new leaf to be created in the decision tree to handle those particular instances. This too can create a more generalized or specialized tree. A higher number will create a more generalized tree and a lower number will create a more specialized tree.

3.2.3 Filters

It´s also important to mentioned some of the filters used in WEKA since they were used in almost all of the experiments. The used filters are named prefilters because they deal with the data after it is inputted in WEKA and before it is classified. This means that these filters manipulate the data before the construction of the classification model on the supervised learning phase.

The specific filter used is the supervised instance resample filter since it allows two operations that were considered relevant for the experiments. One of the operations is the ability to balance the number of instances in each class thus providing a non biased data arrangement for the construction of the classification model. The second operation is the downsampling of the data. This downsampling operation discards random instances and its use will be explained in future sections.

Chapter 4

4. Experiments

On this chapter the conducted experiments will be described while giving the necessary theoretical background and presenting the specific parameters. The experiments can be divided into three separate types, features, preprocessing and feature combination.

4.1 Features

On this section some considerations about the features used on the extraction process will be discussed along with the explanation of some conducted experiments regarding those same features. From all of the features contained in MARSYAS four where selected based on the results of other authors or because they seemed interesting. On the following experiments each feature was tested with several classifiers and the results will be useful for comparison with experiments that will be presented in future sections. The following explanation of the experiments is valid for the extraction of the four types of features presented in this section and is depicted in Figure 17. The variants in each experiment reside on the extracted features.

The coefficients were extracted using the Greek, Ellis and the Allegro data sets. For each data set, a collection of the vocal and instrumental audio files was produced by the mkcollection and the bextract extracted the attributes saving them in an arff file. This function calculates the mean and the standard deviation of each coefficient for every texture window. The used window size was 1024 samples while the hop size was consisted of 512 samples having each texture window 40 analysis windows.The number of instances in each class was then balanced using the WEKA supervised filter. After the data was balanced, it was classified using three different classifiers:

- Naive Bayes – with the command “weka.classifiers.bayes.NaiveBayes” and with a cross validation value of 3.
- Support Vector Machine – the SMO version with the command” weka.classifiers.functions.SMO -L 0.0010 -P 1.0E-12” and with a cross validation value of 3. The “-L” option is the tolerance parameter while the “-P” option is the epsilon value for the round-off error.

- J48 – Using the command “weka.classifiers.trees.J48 -C 0.25 -M 200 -t mfcc.arff -x 3 -i” where the “C” option is the confidence factor and the “M” option is the minimum number of objects per leaf. For this classifier, six different values for the “M” option were used, 200, 500, 2500, 3500, 4500 and 6000. The range of these values is large because the data sets also have large range of samples and therefore a minimum number of objects of 2500 can be insignificant for a huge data set while being overwhelming for a small one.

Finally the results were saved on a txt file.

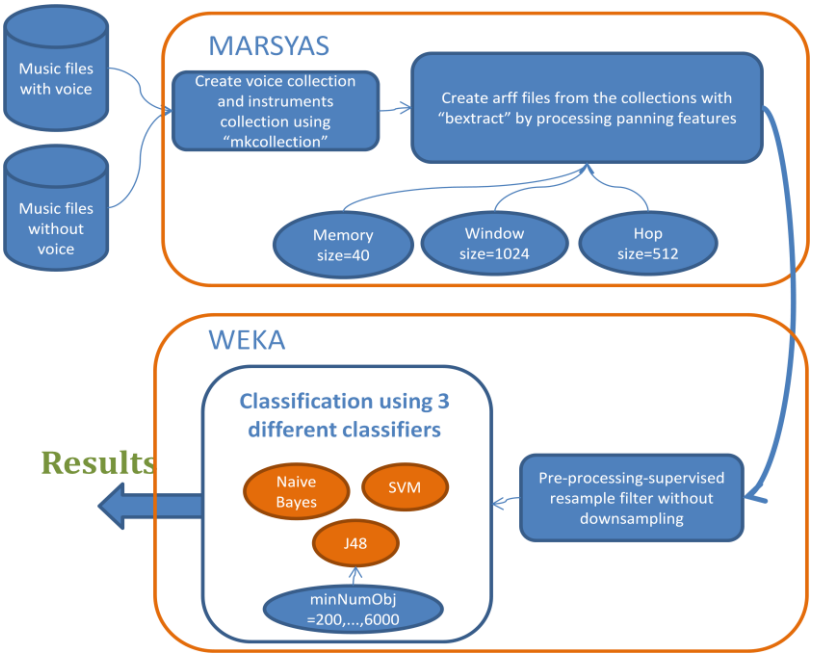

Figure 17: Block diagram of the MFCC experiment

4.1.1 MFCC Features

On this experiment, the usefulness of the MFCC will be tested. These features were selected based on the reports of several authors where the obtained classification results for voice/instrument segments using MFCC were high [22], [27], [2].

MFCC are coefficients derived from a cepstral representation of audio signals and are commonly used for speech processing and data compression. These coefficients are a perceptual representation of pitches based on the Mel scale where the frequency bands are positioned in the logarithmic scale to better simulate the human auditive system. The pitch is the perception of how high (higher frequency) or low (lower frequency) a sound is for the human ear in relation to the note (fundamental frequency) from which it derives from. Roughly, lower coefficients represent spectral envelope while higher ones represent finer details of the spectrum.

These coefficients were extracted using the procedure present on the beginning of this section corresponding to the scheme of Figure 17. For the extraction of these coefficients, the command used was “bextract -mfcc instrument_class.mf voice_class.mf -ws 1024 -hp 512 -w

mfcc.arff". This function considers 13 MFCC coefficients but since it calculates the mean and the standard deviation for every texture window, the resulting number of attributes is equal to 26.

4.1.2 Panning Features

This experiment was motivated by the results obtained by George Tzanetakis, Randy Jones, and Kirk McNally on [20] . On the mentioned report, a new feature for audio analysis is proposed. This feature is called stereo panning spectrum (SPS) and is based on the difference between the 2 channels of a stereo music file. The main idea is to capture the amount of panning in different frequency bands as well as how it changes over time. On this experiment the usefulness of these features for voice/instrument classification will be tested.

The basic idea behind the SPS is to compare the left and right signals in the time-frequency plane to derive a two dimensional map that identifies the different panning gains associated with each time-frequency bin. By calculating the STFT of the right ($Xr(k)$) and left ($Xl(k)$) channels, and combining them it´s possible to obtain a similarity measure given by

$$\varphi(k) = 2\frac{|X_l(k)X_r^*(k)|}{|X_l(k)|^2 + |X_r^*(k)|^2} \tag{4.1}$$

When the source is panned to the centre the value of the similarity measure is 1. When the source is panned to either side the value of the similarity measure will be 0. The only problem of this measure is the inability to determine if the source is panned to right or to the left. This issue can be resolved using the difference between the partial similarity measures ($\Delta(k)$), which are basically the calculation of the similarity measure for each independent channel. Positive values of $\Delta(k)$ correspond to signals panned towards the left and negative values correspond to signals panned to the right. Finally it´s possible to calculate the desired $SPS(k) = [1 - \varphi(k)] \times \Delta(k)$. In [20], a feature vector (θ) corresponding to 4 different frequency analysis windows is consider. The 4-dimensional feature vector corresponding to an analysis window t:

- Ptotal(t) - overall panning (0–22050 Hz).
- Plow(t) - low frequencies(0–250 Hz).
- Pmedium(t) – medium frequencies (250–2500 Hz).
- Phigh(t) - high frequencies (2500–22050 Hz).

The values of each of the 4 features are calculated by

$$P_{l,h} = \sqrt{\frac{1}{h-l+1}\sum_{k=l}^{h}[SPS(k)]^2}, \tag{4.2}$$

where "h" is the higher frequency of the window and "l" is the lowest frequency of the window. To capture the dynamics of panning information, the running mean and standard deviation over the past M frames are calculated by

$$m\theta(t) = mean[\theta(t - M + 1), \dots, \theta(t)], \quad (4.3)$$

and

$$s\theta(t) = std[\theta(t - M + 1), \dots, \theta(t)]. \quad (4.4)$$

These will be the main coefficients used to represent the panning features and were extracted using the procedure presented on the beginning of this section corresponding to the scheme of Figure 17. The Ellis data set was not used because the audio files are mono. For the extraction of these coefficients, the used command used was "bextract -spsf -st --SpectralCentroid instrument_collection.mf voice_collection.mf -ws 1024 –hp 512 -w panning.arff". This function considers four coefficients corresponding to the four frequency bands but since it calculates the mean and the standard deviation for every texture window, the resulting number of attributes is equal to 8.

4.1.3 LSP Features

LSP was first introduced by Itakura and Sugamura [28] as an alternative to the LPC. It was found that this new representation had some advantages over its predecessor and obtained good results for speech recognition. Because of this and since this function was available on MARSYAS this feature was tested for singing voice detection.

The LPC is another method of separating out the effects of source and filter from a speech signal. It encodes a signal by finding a set of weights on earlier signal values that can predict the next signal value. It can be understood as a set of coefficients that contain information about the glottal source filter, the lip radiation/preemphasis filter and the vocal tract itself. This means that LPC coefficients can simulate speech by sending an impulse signal, like the vibrations on the vocal chords through the filter. These coefficients were extracted using the procedure presented on the beginning of this section corresponding to the scheme of Figure 17. For the extraction of these coefficients, the used command was "bextract -lsp instrument_class.mf voice_class.mf -ws 1024 -hp 512 -w lsp.arff". This function considers 18 LSP coefficients but since it calculates the mean and the standard deviation for every texture window, the resulting number of attributes is equal to 36.

4.1.4 LPCC Features

The use of LPCC coefficients was motivated by the results presented in [27] and also by the fact that the extractor existed in MARSYAS. These coefficients derivate from the already mentioned LPC coefficients. After computing the LPC coefficients, they are converted to cepstral representation using linear transformation and finally they are warped using bilinear representation creating the LPCC coefficients.

These coefficients were extracted using the procedure presented on the beginning of this section corresponding to the scheme of Figure 17. For the extraction of these coefficients, the

used command used was “bextract -lpcc instrument_class.mf voice_class.mf -ws 1024 -hp 512 -w lpcc.arff”. This function considers 12 LPCC coefficients but since it calculates the mean and the standard deviation for every texture window, the resulting number of attributes is equal to 24.

On the next section the experiments for testing the influence of preprocessing in singing voice detection will be described.

4.2 Preprocessing

In this section some experiments using preprocessing techniques will be explained, as well as the respective theoretical background. Regarding preprocessing in this project, two different types are considered. The first one, peak clustering, deals with the music files before entering MARSYAS for feature extraction purposes. The second one, downsampling, deals with data in arff format that will be used for WEKA classification purposes as is depicted in Figure 18 . The downsampling is done resorting to MATLAB or WEKA functions.

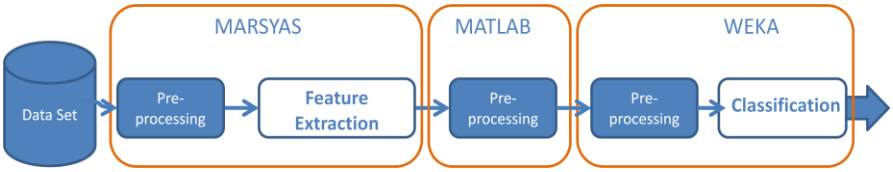

Figure 18: Block diagram explaining the location of the preprocessing blocks

On the next subsections some preprocessing techniques will be explained, first the downsampling and lastly the peak clustering technique.

4.2.1 Downsampling

The first preprocessing technique tested was the downsampling of the data. This experiment was firstly motivated by the need to reduce computational time and also because of memory limitations while using WEKA due to the size of the Greek data set. However, some interesting results were obtained and so some other experiment parameters were tested. On this experiment the effects of downsampling the data for voice/instrument classification will be tested.

On the first experiments, the Greek data set was the only one available. Due to its size, every analysis process took an enormous amount of time to compute. Also, some experiments could not be completed due to memory limitations associated with the java engine where the WEKA commands were processed. Therefore, two options were available, the first one was to use other data sets and the second was to reduce the data itself. Since both options were useful, both were adopted. Considering the reduction of the data set, it is not a trivial subject since it can be done in two ways:

- Use less data files from the data sets.
- Use less analysis points in each file(Downsampling).

The first solution is clearly the worst one because it would remove the diversity of the data set and therefore diminish the confidence of the final results. The objective is to have diversity in order to have a wider scope of classification. The second solution can be used because it will probably have no effect on the diversity of the data. The fact is that music has a variation standard of hundreds of milliseconds. This means that for example, in two consecutive analysis frames of 50ms the feature points won't be that different. If true, this means that we are working with repeated and worthless information that can be discarded. For this experiment two types of downsampling were considered:

- Selective downsampling.
- Random downsampling.

These operations deal with the removal of intermediate texture windows. In Figure 19, an example of texture windows consisted of 4 analysis windows each is represented.

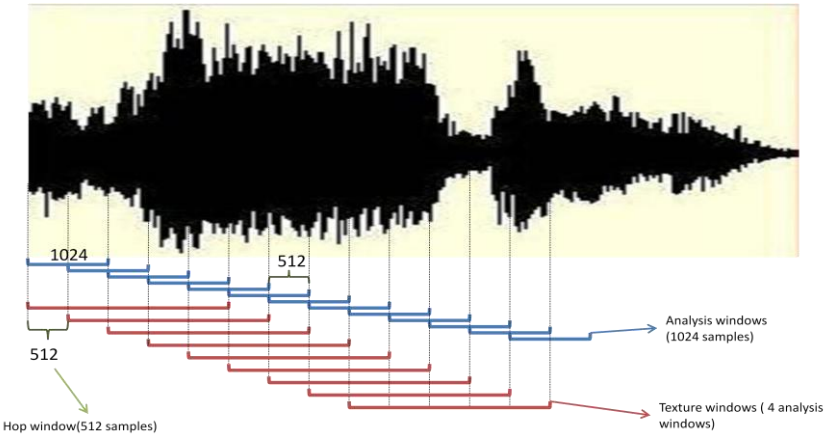

Figure 19: Texture window example for m=4 analysis windows

In fact each texture window by default (in bextract) contains 40 analysis windows but is only separated from the next texture window by the hop size that is equal to half of an analysis window size. This means that two sequential texture windows consisted of 40 analysis windows share 97,5 % of the samples. Having this as a fact it may be interesting to consider only some texture windows while varying their size.

4.2.1.1 Selective Downsampling

The selective downsampling refers to an operation executed using MATLAB functions. It occurs after the creation of the arff files in MARSYAS and before entering WEKA for classification as depicted in Figure 20 .

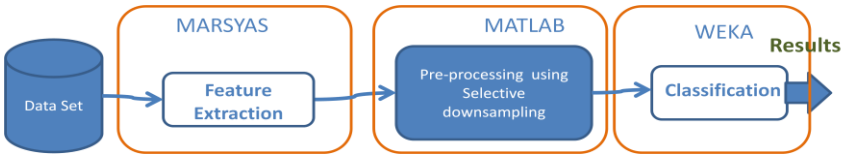

Figure 20: Selective downsampling framework

This process discards the intermediate texture windows until the number of shared samples between two texture windows is equal to the hop size. In Figure 21 an example of texture windows consisted of 4 analysis windows is presented. As can be seen, since the texture window size is equal to 4, the number of discarded texture windows is 3 (4 minus 1).

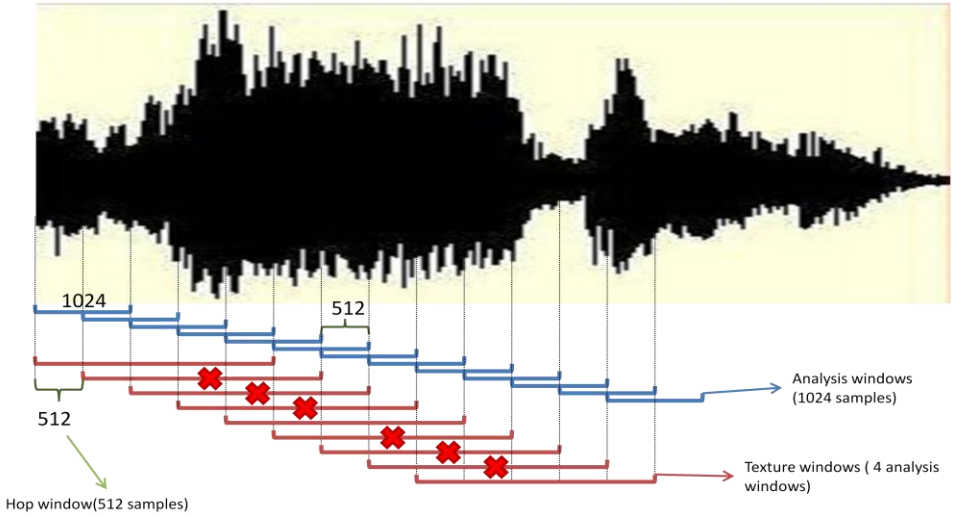

Figure 21: Selective downsampling- discarded texture windows

The resulting texture windows for this example are depicted in Figure 22. One can see that the percentage of common samples reduces, becoming for this case 20%.

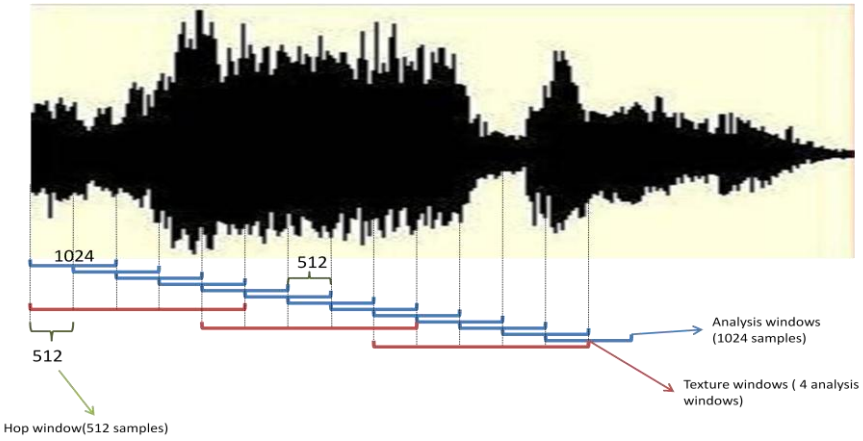

Figure 22: Selective downsampling- selected texture windows

4.2.1.2 Random Downsampling

The random downsampling refers to an operation executed before the classification process using the already mentioned WEKA supervised instance filter as depicted in Figure 23. In this filter one can select the percentage of the desired downsampled.

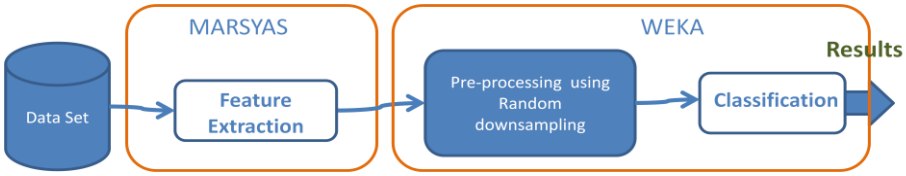

Figure 23: Random downsampling framework

The filter randomly discards instances of the input arff file. This is the same as randomly discard texture windows as depicted in Figure 24 for an example of texture windows consisted of 4 analysis windows each and with a downsampling of 50%.

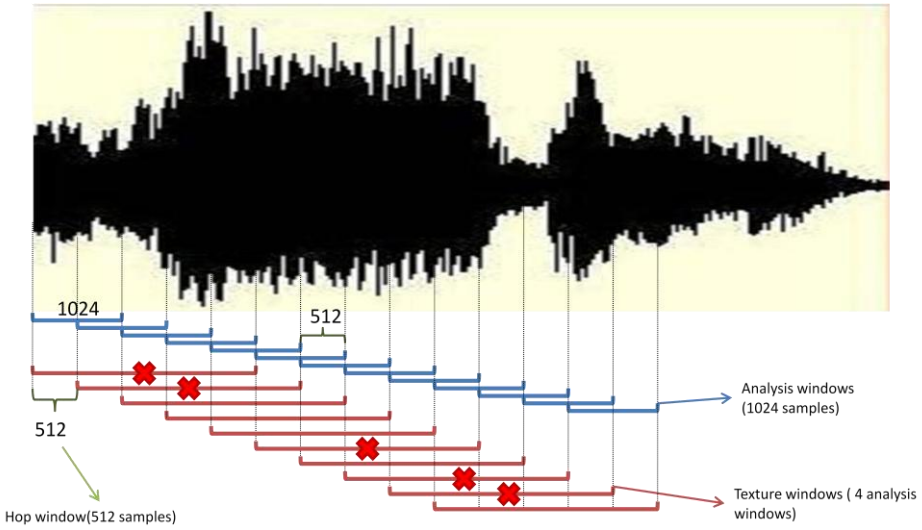

Figure 24: Random downsampling- discarded texture windows

The resulting texture windows for this example are depicted in Figure 25. The remaining texture windows are in this case close together but since this is a random operation this should not be seen as a rule.

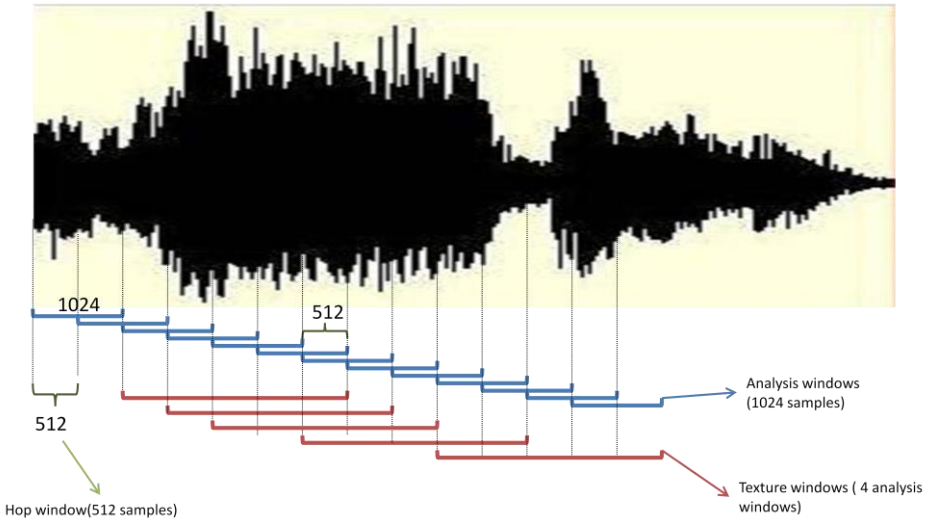

Figure 25: Random downsampling- remaining texture windows

4.2.1.3 Experimental Evaluation

As mentioned before, these experiments were first motivated by WEKA memory limitations but the major problem concerning big data sets is the processing time because if the data set has many instances, the processing time of feature extraction and classification becomes a nuisance. Also important is the repeated information because if excessive it may confuse the classifiers or have other malicious results in the classification process.

On this experiment the usefulness of this preprocessing algorithm for voice/instrument classification will be tested. It can be divided into three separate tests. On the first test the effects of downsampling and texture window size (memory size) will be tested. On the second test the value of the selective downsampling will be tested against the random downsampling while on the third test the value of the texture window size will be reviewed. The only data set used for these tests was the Greek data set because it enabled higher downsampling values since it has a much larger quantity of data than the others data sets. The other data sets could have been used but since for high downsampling values the remaining data was consisted of less than 500 instances the results would not be considered valid. The classifiers used for the three tests were:

- Naive Bayes – with the command "weka.classifiers.bayes.NaiveBayes".
- Support Vector Machine – the SMO version with the command" weka.classifiers.functions.SMO -L 0.0010 -P 1.0E-12". The "-L" option is the tolerance parameter while the "-P" option is the epsilon value for the round-off error.

A) The First Test

Considering the first test, Figure 26 shows the block diagram of the test sequence.

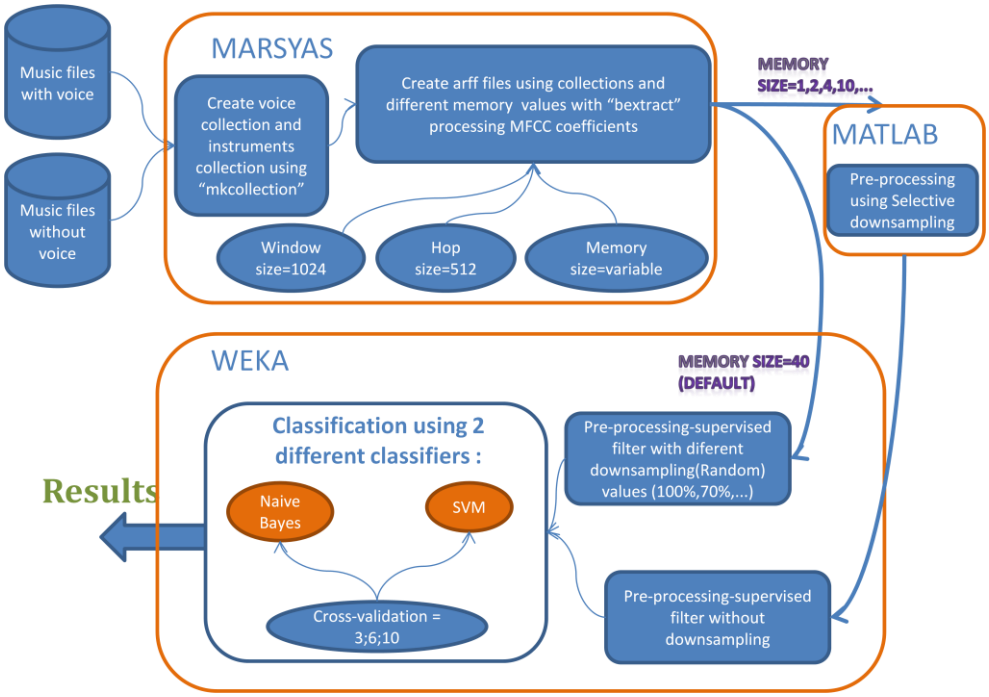

Figure 26: Framework for the first downsampling test

For the extraction of the MFCC features the bextract function processed the wav files supplied by the Greek data set working with an analysis window size of 1024, a hop size of 512 and several values for the memory size (texture window size). The resulting arff files for the different values of the memory size were loaded into MATLAB for the execution of the supervised

downsampling. When completed the downsampled arff files were inputted into WEKA and before applying a classifier with several values for the cross validation a resample filter was used so that the number of instances for each class was equal. The resulting arff from the extraction with memory size equal to 40 that is the default value was randomly downsampled for several different percentages with the supervised filter in WEKA. After being downsampled the resulting arff files were processed by the classifiers for the construction of the classification models using several values for cross validation. Once the classification model was constructed, the final stage, classification was executed.

B) The Second Test

The extraction of the MFCC coefficients was similar to the one presented on the first test. The resulting arff files for the different values of the memory size were loaded into MATLAB for the supervised downsampling and their duplicates were inputted into WEKA and were randomly downsampled for several different percentages with the supervised filter before being classified. The selective downsampled arff files were inputted into WEKA and before applying a classifier, a resample filter was used so that the number of instances for each class was equal. The scheme of this test can be seen in Figure 27. After being filtered, the resulting arff files were processed by the classifiers for the construction of the classification models using several values for cross validation. Once the classification model was constructed, the final stage, classification was executed.

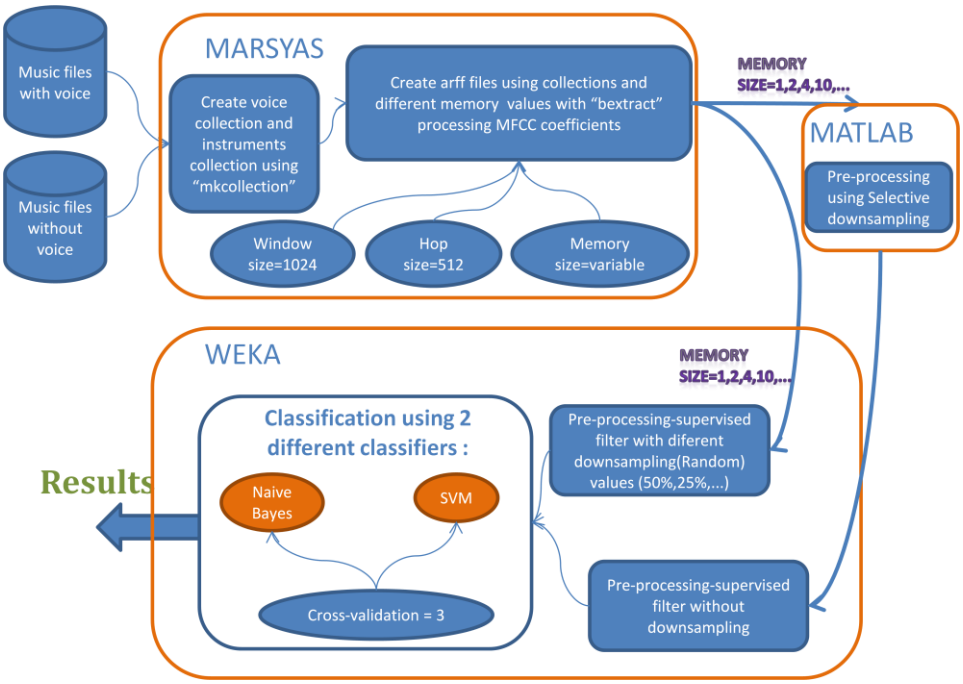

Figure 27: Framework for the second downsampling test

C) The Third Test

The Greek data set was used to supply the bextract function present in MARSYAS with data in order to extract the MFCC features. The used analysis window size was 1024, the hop size 512 and the memory size was tested for the extreme values of 1, 40, 100 and 200. The resulting arff files for the different values of the memory size were loaded into WEKA with no downsampling but a resample filter was used so that the number of instances for each class was equal. The scheme of this test can be seen in Figure 28. After being filtered the resulting arff files were processed by the classifiers for the construction of the classification models using several values for cross validation. Once the classification model was constructed, the final stage, classification was executed.

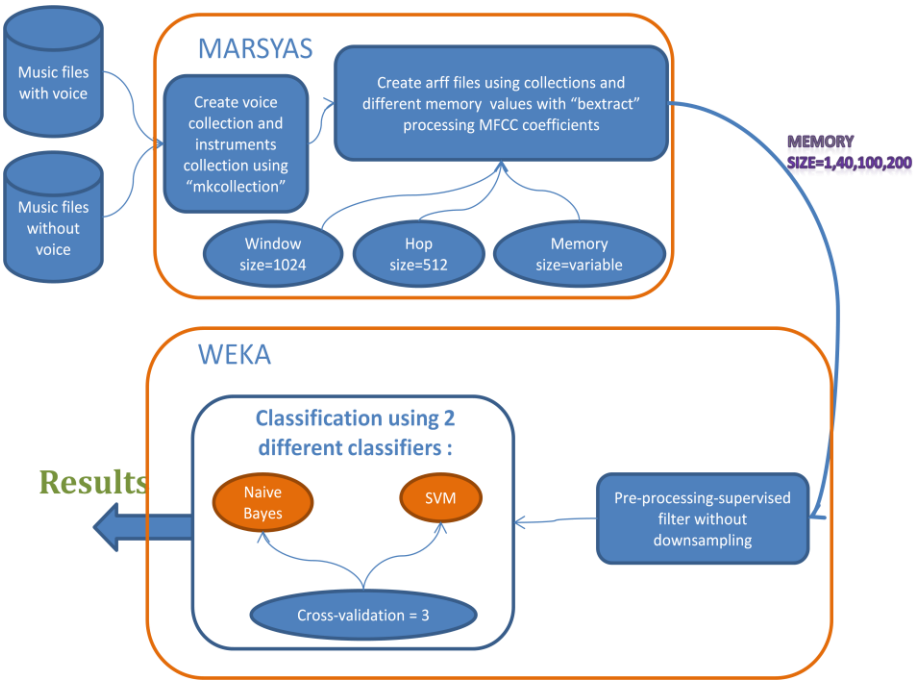

Figure 28: Framework for the third downsampling test

4.2.2 Peak Clustering

The implementation of this technique was motivated by the results obtained by Mathieu Lagrange, Luis Gustavo Martins, Jennifer Murdoch and George Tzanetakis, on their report [16] where a new preprocessing algorithm is proposed. This algorithm is based on Spectral Clustering and its goal is the direct separation of the predominant melodic sources from music signals. The main ideas behind this algorithm are the fact that human hearing "groups together" similar melodic sounds and also the fact that usually the most predominant melodic source is the human voice. A schematic of the process is showed in Figure 29. This algorithm was implemented in MARSYAS under the name of peakclustering and has several parameters that can be altered as will be explained further ahead. The algorithm has three steps, as input it receives an audio file and determines the spectral peaks, then it calculates the predominant melodic sources by spectral analysis and lastly, it reconstructs the signal.

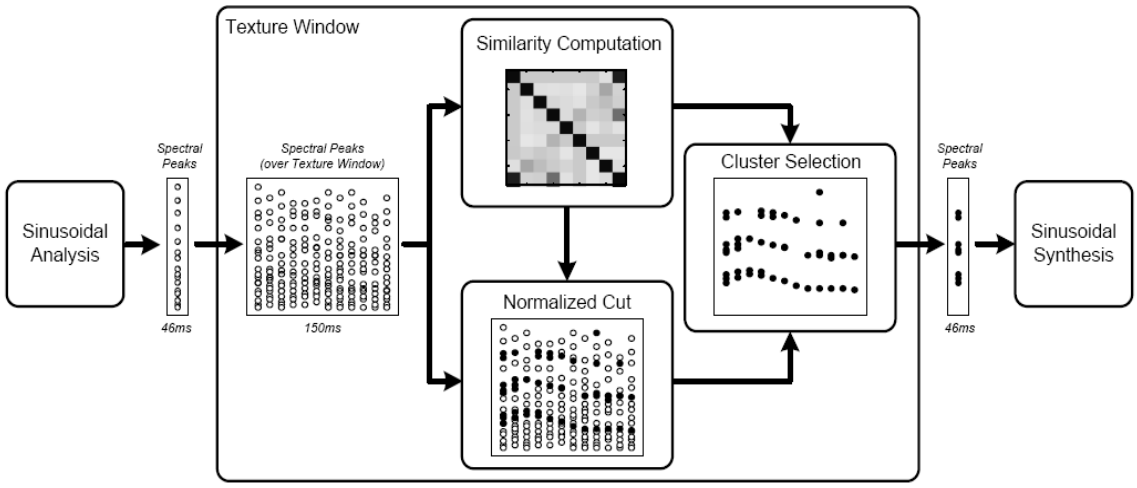

Figure 29 : Block diagram for the peak clustering process[Mathieu Lagrange, Luís Gustavo Martins, Jennifer Murdoch, George Tzanetakis, " Normalized Cuts for Predominant Melodic Source Separation",February,2008]

4.2.2.1 Sinusoidal Analysis

On the first step, the sinusoidal analysis, the objective is to separate the input file into sinusoidal spectral peaks. This is achieved by applying a STFT (Short-time Fourier transform) in each frame. The outputs of this stage are the amplitude, phase and frequency of each of the obtained peaks. The number of peak (sinusoids) to be calculated per frame can be altered by using the "-s" option.

4.2.2.2 Clustering

On the second step, the clustering, the peaks obtained on the first step are grouped in texture windows. Each texture window is formed by an integer number of frames which will help the partial tracking and the source formation. Similarity cues are used to calculate the similarity between sinusoidal peaks belonging to the same texture window. These cues are inspired by perceptual grouping cues such as amplitude, frequency proximity and harmonicity. Edges are formed both for peaks within a frame and peaks across frames. Each partition is a set of peaks that are grouped together such that the similarity within the partition is maximized and the similarity between different partitions is minimized. This means that each partition can contain several peaks from different frames thus forming a candidate for melodic source. The similarity is measured by the Harmonically Wrapped Peak Similarity (HWPS) which takes advantage of the similarity between peaks, not only in the same frame but also in other frames. The idea is to assign to each peak a spectral pattern that captures information about the spectrum in relation to the specific peak. As such, the degree of matching between these spectral patterns can be

used for similarity measure together with the amplitude and frequency between two peaks. However, from frame to frame the spectral pattern may change and also a specific frame may contain several harmonic sources. To solve these issues, the authors [16] use harmonically wrapped frequency space to align the two spectral patterns corresponding to the peaks. The objective is to obtain higher values of similarity for the peaks correspondent to the same melodic source and lower ones to peaks of other harmonic sources.

The number of desired clusters per texture window can be selected with the option "-c". Having found the clusters, it now becomes necessary to determine the most important ones and discard the others. This means that the most prominent melodic sources will be found by considering that they have more and better intra-cluster similarity measures. The number of selected clusters can be altered by the option "-k".

4.2.2.3 Sinusoidal Synthesis

On the third step, the sinusoidal synthesis, the peaks corresponding to the selected clusters are used to resynthesize the extracted signal using a bank of sinusoidal oscillators.

4.2.2.4 Experiment Conditions

On this experiment the usefulness of the peak clustering preprocessing algorithm for voice/instrument classification was tested. In Figure 30 the basic schematic of the preprocessing phase is described and is valid to all of the experiments although some conditions might change.

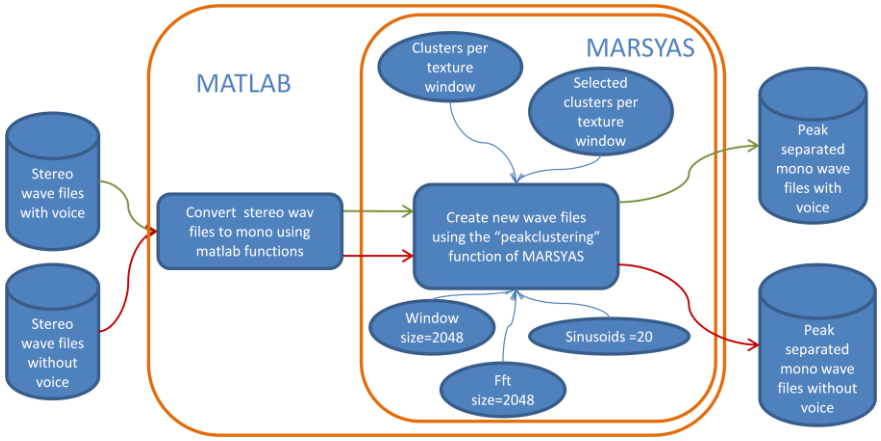

Figure 30: Block diagram of the data set preparation for the peak clustering experiment

The peakclustering function accepts mono wav files as input, so some music files were converted from stereo to mono using MATLAB functions. The mono conversion was used for the Greek and the Allegro data sets since the Ellis data set is formed by mono wav files. For the peakclustering function several parameters can be altered but only 5 were considered:

- Frame size "-w" option- The number of samples in each analysis frame.

- Sinusoids, "-s" option- maximum number of possible peaks calculated for each frame.
- Clusters per window "-c" option – number of peak clusters for each texture window.
- Selected Clusters per window "-k" option – number of peak clusters admitted as possible melodic source that are used by other texture windows.

The second phase is depicted in Figure 31 where the obtained data sets from the preprocessing phase were used to supply the bextract function present in MARSYAS with data in order to extract the MFCC features saving the result in arff file. The used analysis window size was 1024, the hop size 512 and the memory size equal to 40 frames. The resulting arff file was loaded into WEKA for classification purposes. Before applying a classifier, a resample filter was used to guarantee that the number of instances for each class was equal. The classifiers used were:

- Naive Bayes – with the command "weka.classifiers.bayes.NaiveBayes" and with cross validation equal to 3.
- Support Vector Machine – the SMO version with the command" weka.classifiers.functions.SMO -L 0.0010 -P 1.0E-12" and with cross validation equal to 3. The "-L" option is the tolerance parameter while the "-P" option is the epsilon value for the round-off error.

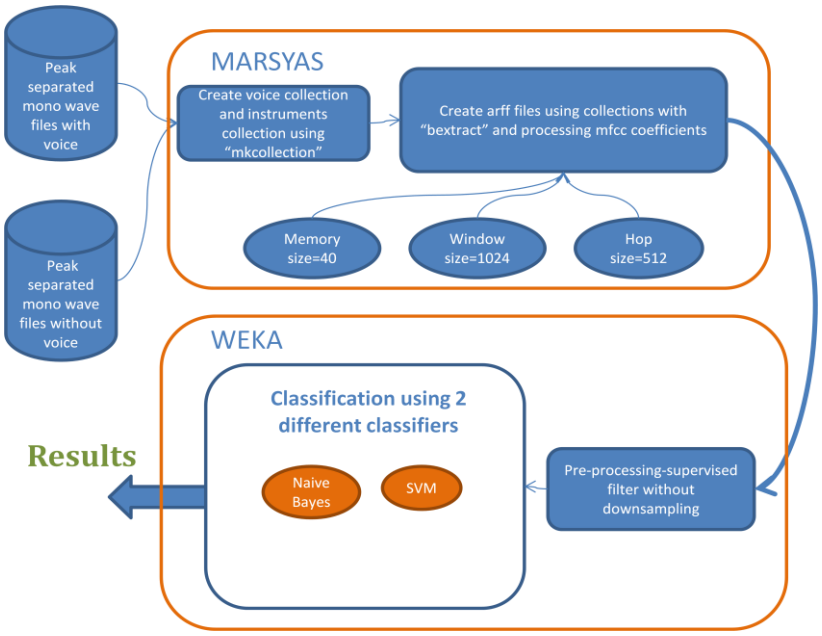

Figure 31: Block diagram of feature extraction and classification phases for the peak clustering experiment

For comparison the popular MFCC coefficients were used. These features were extracted from the same data sets and evaluated by the same classifiers in the same conditions. The experiment conditions are represented in Table 3.

Table 3 Peak Clustering experiment parameters

	Test nº							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Data set Used	Greek(after mono conversion)	Ellis	Ellis	Ellis	Ellis (corrected)	Allegro	Ellis	Allegro
Extracted Features	MFCC	MFCC	MFCC	MFCC	MFCC	MFCC	LSP	LSP
Clusters per texture window	3(default)	3(default)	5	5	5	5	5	5
Selected Clusters per texture window	2(default)	2(default)	3	3	3	3	3	3
Number of sinusoids	20	20	20	10	20	20	20	20

Although the experiment was presented only for the extraction of the MFCC coefficients, on posterior tests, the LSP coefficients were also extracted using the same methodology.

On the next section the experiments that combine some of the presented features for singing voice detection will described.

4.3 Feature Combination

On the following experiments, the MFCC, LSP, LPCC and panning coefficients were extracted and combined together for the construction and evaluation of the classification model. The arff file of each feature was merged by MATLAB in order to create only one arff. The structure of these experiments is similar to the one presented in section 4.1. The different resides in the fact that each independently extracted feature is merged with others creating an arff file that will be classified by WEKA as is depicted in Figure 32. This process enabled the classifiers to construct a classification model based on more than one feature. The classifiers and their options are similar to the ones used in section 4.1.

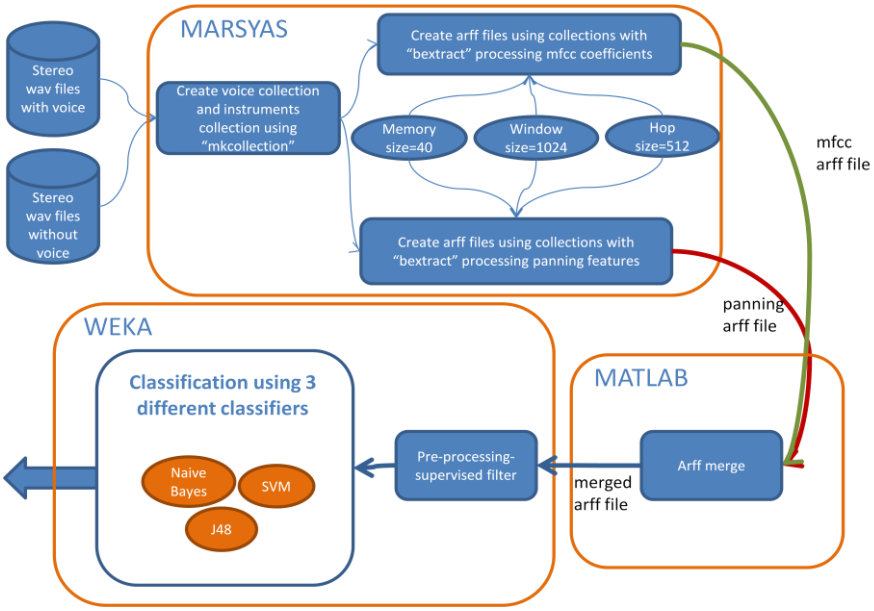

Figure 32: Block diagram of the panning and MFCC feature selection experiment

The block diagram presented in Figure 32 shows the test where the MFCC and panning coefficients were used. This structure was used for all the tests but using different combination of features as Table 4 shows.

Table 4: Feature combination experiment parameters

Features	Data sets	Number of attributes
MFCC +LSP	Allegro, Ellis, Greek	62
MFCC+LPCC	Allegro, Ellis, Greek	50
MFCC+Panning	Allegro, Greek	36
MFCC+LSP+LPCC	Ellis	86
MFCC+LSP+LPCC+Panning	Allegro, Greek	94

The number of attributes column represents the total number of attributes resulting from the merge operation. The Ellis data set was not used when the panning coefficients were extracted since it contains only mono files. For the last test where all the features were extracted a downsampling of 4% of the Greek data set was executed because the increase of the number of attributes caused an enormous augment of the required computational time for constructing the classification model of the SVM classifier.

Chapter 5

5. Results

This chapter is divided into three sections where the results of the experiments presented in chapter 4 will be discussed along with some preliminary conclusions. On the first section the influence of the selective and random downsampling as preprocessing techniques will be discussed. On the second section the peak clustering technique used as preprocessing in order to help on the extraction of some features will be evaluated. The last section will present the results of combining different features.

5.1 The influence of preprocessing using downsampling

The results for the first test of the experiment described in 4.2.1 are depicted in Table 5, and represent the number of correctly classified instances for both classes. As a remainder, the used features for the presented tests of this section are the MFCC coefficients extracted from the Greek data set.

Table 5: Results for the first test of the downsampling experiment

		Classifier							
		Naive Bayes			SVM				
		Cross validation							
	Downsampling (%)	3	6	10	3	6	10	Number of Instances	
Random downsampling for MFCC coefficients	100	75,76	75,75	75,75	79,35	79,36	79,36	977.927	window size = 1024 samples ; texture window size (memory size) = 40; hop size = 512 samples
	70	75,77	75,77	75,77	79,38	79,38	79,38	684.548	
	50	75,78	75,78	75,78	79,4	79,39	79,39	488.963	
	30	75,88	75,88	75,87	79,44	79,44	79,44	293.378	
	10	75,9	75,91	75,89	79,54	79,58	79,56	97.792	
	5	75,83	75,82	75,87	79,69	79,71	79,72	48.896	
	1	76,23	76,19	76,32	79,31	79,4	79,41	9.779	
	memory size								
Selective downsampling for MFCC coefficients	1	62,2	62,2	62,2	65,91	65,92	65,92	977.927	window size = 1024 samples ; hop size= 512 samples
	2	60,63	60,63	60,62	66,71	66,72	66,72	488.963	
	4	61,25	61,26	61,24	67,58	67,6	67,59	244.479	
	10	67,08	67,04	67	70,14	70,12	70,14	97.795	
	20	71,84	71,8	71,8	74,05	74,1	74,1	48.897	
	40	76,04	76,07	76,12	79,54	79,6	79,65	24.447	
	60	77,69	77,76	77,78	82,66	82,62	82,64	16.299	
	80	80,33	80,36	80,32	85,62	85,49	85,58	12.222	
	100	81,75	81,89	81,83	87,76	87,65	87,64	9.778	
	200	85,03	84,95	84,7	92,33	91,52	92,58	4.890	

From the obtained results presented in Table 5, it's possible to conclude that there´s no substantial difference in using 3, 6 and 10 as cross validation values and so for future experiments a value of 3 will be adopted.

Another conclusion is that for the random downsampling, the number of instances does not have a significant influence in the classification process. As can be seen, for a downsampling of 1% (9779 instances) the results are around 0,5% better for the Naive Bayes classifier and almost the same for the SVM classifier when compared with the 100% downsampling (977.927 instances) results. What can be concluded is that it's safe to randomly downsampling the data, although if greatly downsampled it may become too specific for the classifier thus making the classification model useless for other data. That is why the lowest used value for the downsampling was 1%.

For the selective downsampling the results improve with the reduction of the used number of instances and with the increase of the texture window. This can mean that either the selective downsampling is in fact better than the random downsampling, or that the size of the texture window has a great influence on the results. To understand the influence of both options the results of the second test of the experiment presented in 4.2.1 must be analyzed in Table 6. From the obtained results is possible to conclude that selective downsampling does not improve the classification results.

Table 6: Results for the second test of the downsampling experiment

	Memory size		Classifier		Number of Instances	
			Naive Bayes	SVM		
Selective downsampling for MFCC coefficients	**2**		60,63	66,71	488.963	window size =1024 samples ; hop size= 512 samples
	4		61,25	67,58	244.479	
	10		67,08	70,14	97.795	
	20		71,84	74,05	48.897	
	40		76,04	79,54	24.447	
	60		77,69	82,66	16.299	
	80		80,33	85,62	12.222	
	100		81,75	87,76	9.778	
	200		85,03	92,33	4.890	
Random downsampling for MFCC coefficients	**memory size**	**Downsampling (%)**				
	2	**50**	60,56	66,58	488.963	window size =1024 samples ; hop size= 512 samples
	4	**25**	61,52	67,82	244.481	
	10	**10**	67,2	70,45	97.792	
	20	**5**	71,86	74,33	48.896	
	40	**2,5**	75,89	79,74	24.448	
	60	**1,67**	78,45	83,01	16.331	
	80	**1,25**	80,41	85,82	12.224	
	100	**1**	82,2	88,08	9.779	
	200	**0,5**	87,32	93,7	4.889	

Since the selective downsampling is not responsible for the improvement in the classification results viewed in Table 5 and Table 6 one can assume that this improvement is due to the increase of the texture window size. In order to confirm this assumption the third test of the experiment presented in 4.2.1 was executed and the results are presented in Table 7.

Table 7: Results for the third test of the downsampling experiment

	Memory size (texture window)	Classifier		Number of Instances	
		Naive Bayes	SVM		
MFCC coefficients with no downsampling	**1**	62,2	65,91	977.927	window size=1024 samples ; hop size= 512 samples
	40	75,76	79,35	977.927	
	100	81,74	87,94	977.927	
	200	86,37	94	977.927	

From the analysis of the results presented in Table 7 it's finally possible to conclude that in fact the use of larger texture windows improves classification results. One important aspect to be considered is the meaning of the size of a texture window. The highest tested value for the texture windows was 200 because it represents a segment of roughly 2,3 seconds for the input

files that were sampled at 44 100Hz. This 2,3 seconds window is in truth to large because when dealing with original music files that are not separated into vocal and instrument segments it may ignore eventual segment changes. As an example let´s consider Figure 33 where an example of a texture window with a size of 200 samples using selective downsampling is depicted. The texture windows have approximately 2,3 seconds and are proportional in relation to the time scale presented.

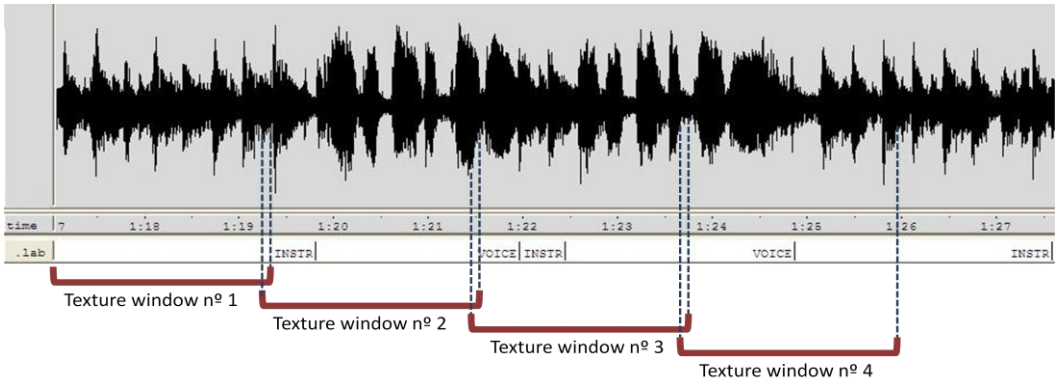

Figure 33: Example of texture window with size equal to 200 using selective downsampling

On this example, texture windows number 2, 3 and 4 contain samples from two different classes, voice and instrument. This will not be considered by the extractor since it will execute an average of the values of the analysis windows. This may cause for example that the considered texture windows are labelled as voice when in fact that is not true. For music genres that have a high rate of change between voice and instrument segments this does not appear to be the best approach. However for music genres that are constant and have low frequency of change between voice and instrument this approach can be very useful.

On the next section the influence of the peak clustering algorithm in singing voice detection will be discussed.

5.2 The influence of preprocessing using peak clustering

The obtained results for the experiment described in 4.2.2 can be seen in Table 8. The values refer to the percentage of correctly classified instances for both classes (instruments and voice).

Table 8: Peak Clustering experiment results

				Classification results		
		Data set used	Feature	Naive Bayes	SVM	Number of instances
Test nº	1	Greek(after mono conversion)	MFCC	59,14	64,7	978471
	2	Ellis	MFCC	68	73,88	39431
	3	Ellis	MFCC	68,4	73,63	39431
	4	Ellis	MFCC	66,62	70,91	39431
	5	Ellis (corrected)	MFCC	68,15	73,85	39121
	6	Allegro	MFCC	64,13	71,12	104421
	7	Ellis	LSP	49,9	69,13	39.451
	8	Allegro	LSP	53,35	71,19	104.421
MFCC with no preprocessing; memory size = 40; analysis window=1024 samples; hop size=512 samples		Greek	MFCC	75,76	79,35	977927
		Ellis	MFCC	72,77	84,96	38815
		Allegro	MFCC	74,71	78,83	101723
LSP with no preprocessing ; memory size = 40; analysis window=1024 samples; hop size=512 samples		Ellis	LSP	66,06	81,04	38.815
		Allegro	LSP	65,67	79,53	101.723

The obtained results for peak clustering preprocessing approach were disappointing and proved to be well below expectations. In comparison with the results presented in [16], the obtained results are quite inferior in the sense that the classification without preprocessing proved to be more effective for MFCC and LSP coefficients. In fact the results were opposite since in [16] the classification achieved higher values with the use of peak clustering for the MFCC coefficients. One could say this happened because of the data sets since on some songs the predominant melodic source is not the vocal melody. This means that the peak clustering operation could be extracting melodic sources that are not voice based and by doing so confuses the classifier causing the classification model to be less precise. This fact is more relevant for the Greek data set since the difference between the use of preprocessing and the MFCC and only MFCC is higher than for the other data sets. Another possible cause for this difference in the results may be caused by the feature extractor since the bextract function in MARSYAS is constantly updated causing variations in the results. In [16] the authors do not specify how they extracted the MFCC feature so this possibility cannot be confirmed.

Another possible cause for the deterioration of the results when using peak clustering may be the reconstruction of the signal. The music signals are created by the peakclustering function based on their most relevant melodic sources. When listening to the resulting files it's possible to notice that both voice and instrument classes share some kinds of sounds. Although it's true that the singing was reconstructed for the voice class, it's also true that some other sounds were created. These same sounds seem to be present in the reconstructed instrument class. This alone should not cause the deterioration of the results because although there are similar sounds for both classes, there are still unique ones that should be enough to distinguish them. One possible explanation is the way that the extractors extract the features from the recon-

structed signals. Analyzing the classification models constructed by the J48 classifier for the MFCC features with and without peak clustering it's possible to verify that the prominent coefficient differ when peak clustering is executed. Annex D and Annex E show the resulting tree models for MFCC without peak clustering while Annex F shows the same feature with peak clustering. The models in Annex D and Annex E have common coefficients resulting in an approximated tree model even though the data sets used for their construction were different. The common attributes for these models are the std_MFCC3, std_MFCC0 and mean_MFCC1 and only two of them are used on the construction of the tree model for the peak clustering experiment but with low relevance since they are low on the tree, as can be seen in Annex F.

The difference between the results of the Ellis data set and the corrected version of the same data set proved to be very small. This may indicate that the corrected segments aren´t relevant for classification purposes.

On the next section the influence of combining several of the presented features for singing voice detection will be discussed.

5.3 The influence of feature combination

In this section the results of the experiments described in sections 4.1 and 4.3 will be analyzed.

Table 9 shows the results of the experiments were the features were used independently.

Table 9: Results for the independent feature experiment

		Classifier								number of instances
		Naive Bayes	SVM	J48						
	Features			m=200	m=500	m=2500	m=3500	m=4500	m=6000	
Greek Data Set	MFCC (20% downsampling)	75,89	79,6	82,78	80,37	75,89	74,35	74,09	72,98	195585
	LSP (20% downsampling)	57,55	68,81	81,42	78,33	71,43	70,9	69,37	68,33	195585
	LPCC (20% downsampling)	54,95	53,9	64,6	63,02	60,32	59,56	59,25	59,07	195585
	Panning (30% downsampling)	57,41	55,62	67,43	65,87	62,91	62,48	61,91	61,37	293378
Allegro Data Set	MFCC	74,71	78,83	86,51	82,56	74,96	73,74	72,3	71,42	101723
	LSP	65,67	79,53	88,16	84,15	76,1	74,54	72,17	71,89	101723
	LPCC	58,25	61,24	76,49	72,71	67,71	65,96	65,84	64,37	101723
	Panning	59,24	58,57	73,52	70,9	64,2	62,46	61,93	61,14	101723
Ellis Data Set	MFCC	72,77	84,96	87,1	83,82	72,82	70,37	69,63	68,48	38815
	LSP	66,06	81,04	88,94	85,37	78,77	78,1	76,7	71,31	38815
	LPCC	60,04	73,62	81,8	78,05	75,45	71,68	71,03	69,73	38815

Regarding the features it is possible to conclude that in fact the MFCC coefficients seem the best independent feature for voice/instrument classification. The LSP coefficients also achieved

good results even surpassing the MFFC in some cases. The LPCC and the panning coefficients proved to be well below the LSP and MFCC coefficients in terms of classification results for all the classifiers.

On Table 10 the results of the experiments using independent features and a combination of them for voice/instrument classification are showed.

Table 10: Results for the feature combination experiment

		Classifier								number of instances
		Naive Bayes	SVM	J48						
	Features			m=200	m=500	m=2500	m=3500	m=4500	m=6000	
Greek Data Set	MFCC (20% downsampling)	75,89	79,6	82,78	80,37	75,89	74,35	74,09	72,98	195585
	LSP (20% downsampling)	57,55	68,81	81,42	78,33	71,43	70,9	69,37	68,33	195585
	LPCC (20% downsampling)	54,95	53,9	64,6	63,02	60,32	59,56	59,25	59,07	195585
	Panning (30% downsampling)	57,41	55,62	67,43	65,87	62,91	62,48	61,91	61,37	293378
	MFCC +LSP (20% downsampling)	70,49	81,51	84,84	81,85	76,31	75,09	74,45	73,89	195585
	MFCC+LPCC (20% downsampling)	68,71	79,63	83,14	81,02	76,12	74,84	74,28	73,41	195585
	MFCC+Panning (20% downsampling)	75,84	79,89	83,54	81,01	76,03	74,71	74,08	73,91	195585
	MFCC+LSP+LPCC+Panning (4% down-sampling)	66,48	81,41	79,14	76,86	71,53	69,87	70,65	69,73	39117
Allegro Data Set	MFCC	74,71	78,83	86,51	82,56	74,96	73,74	72,3	71,42	101723
	LSP	65,67	79,53	88,16	84,15	76,1	74,54	72,17	71,89	101723
	LPCC	58,25	61,24	76,49	72,71	67,71	65,96	65,84	64,37	101723
	Panning	59,24	58,57	73,52	70,9	64,2	62,46	61,93	61,14	101723
	MFCC +LSP	65,57	79,39	88,16	84,15	76,1	74,53	72,17	71,89	101723
	MFCC+LPCC	68,63	81,72	87,45	83,8	76,57	74,27	73,77	72,11	101723
	MFCC+Panning	76,27	80,25	87,49	83,11	77,3	73,87	72,3	71,42	101723
	MFCC+LSP+LPCC+Panning	69,45	88,51	91,02	87,78	79,69	78,3	78,56	75,32	101723
Ellis Data Set	MFCC	72,77	84,96	87,1	83,82	72,82	70,37	69,63	68,48	38815
	LSP	66,06	81,04	88,94	85,37	78,77	78,1	76,7	71,31	38815
	LPCC	60,04	73,62	81,8	78,05	75,45	71,68	71,03	69,73	38815
	MFCC +LSP	71,79	88,2	89,2	85,66	78,78	78,1	76,7	71,31	38815
	MFCC+LPCC	62,36	85,59	87,82	82,91	75,23	73,1	71,44	70,99	38815
	MFCC+LSP+LPCC	64,78	89,02	90,49	86,12	78,77	78,1	76,7	71,31	38815

The Naive Bayes classifier obtained the worst results for all of the feature combinations with the exception of the panning and LPCC coefficients when compared to the other classifiers. It is also interesting to notice that this classifier has an opposite trend comparing with the other classifiers when it comes to feature combination. With the increase of the number of features hence the attributes, the results of the classification deteriorate. This fact is most likely caused by the fact that all the attributes have the same weight for the Naive bayes since they are con-

sidered independently. This way, the classifier considers attributes that do not improve the construction of the classification model.

The SVM classifier dealt very well with the increase of the number of attributes used since the results also increased. The only inconvenient detail was the processing time required because it increased greatly when all of the features were used for classification.

The J48 classifier also behaved very well with the increase of the number of attributes since the results increased. As the minimum number of objects permitted per leaf (m) increased, the classification results decreased because the classification model became less and less adapted to the data set but the overfitting risk diminished. It´s hard to determine the ideal value for a good classification model with reduced overfitting. One could say from the obtained results that for m=200 the J48 classifier is the best classifier for voice/instrument classification but it´s not free of overfitting. It is also interesting to notice that with the increase of the number of objects permitted per leaf (m) the results tend for the same approximated values for every feature combination ranging from 70% to 74%. This happens because with the reduction of the tree caused by pruning (increasing m value) the resulting attributes of the tree models are roughly the same as showed in Annex G.

Regarding feature combination, some interesting results were obtained. For the SVM and J48 classifiers the results of the combined features were almost always higher when comparing with the results of the independent features. This means that for the LSP, LPCC and panning coefficients the results improve if they are combined with the MFCC coefficients. For the MFCC coefficients the combination with other features was always fruitful when using the SVM or J48 classifiers although for some cases the results are roughly the same. The best results are achieved when using all of the available features with the SVM and J48 (with low m value) classifiers reaching values around 89% of correctly classified instances for both classes. For the Allegro data set using MFCC, panning, LSP and the LPCC coefficients with the SVM classifier was possible to obtain 88,5% of correctly classified instances while for the Ellis data set 89% but using the MFCC, LSP and LPCC coefficients along with the SVM classifier. The results obtained with feature combination for the Ellis data set are higher than the ones obtained in [21] (around 80%) although the same data set was used.

Chapter 6

6. Conclusions

This dissertation addressed the detection of singing voice in music. Many other approaches were considered as a part of a state of the art study but the one presented here was believed to be advantageous.

Regarding the framework presented here, many parameters were tested so that one could understand their effects on singing voice detection. More specifically, the effects of preprocessing using the peak clustering technique proposed by [16] for singing voice detection were analyzed. The effects of downsampling the data sets were also tested using two different approaches, the selective and the random downsampling. Several features were tested either separately or in combination with others in order to understand their value as voice/instrument discriminators. The used features were the MFCC, LSP, LPCC and the panning coefficients. Some other curious tests were implemented such as the use of different sizes for the texture windows.

For the mentioned experiments, three different classifiers were used, the SVM, Naive Bayes and the J48. On some experiments where the J48 was used, several values of the minimum number of object per leaf were tested so that its influence on the construction of the classification models could be comprehended.

Several data sets were used so that the results did not become hostage to a determined set of data. The Allegro data set is a novelty and proved to be a reliable data set since the results obtained when it was used did not differ much from the other data sets.

On the next section, the main contributions of this dissertation will be described followed by some proposals for future work.

6.1 Summary

Regarding the downsampling, it can be concluded that if indeed a downsampling operation needs to be execute, either for data or processing time reduction it can be done without significant loss in classification results. The bigger the data set is the safer is the downsampling operation. This means for example that reducing a data set of 1 million samples to ten thousand samples is safer than reducing a data set of one thousand samples to one hundred samples although both correspond to a downsampling of 1%.

The selective downsampling did not prove to a better alternative to the random downsampling. The use of larger texture windows improves classification results but it can also cause the classifier to ignore eventual segment changes. For genres of song that have a high rate of change between voice and instrument segments this does not appear to be the best approach, however for low changing genres it may be very useful.

Considering the MFCC and LSP as features, the use of Peak Clustering showed a substantially worse performance and so didn't prove to be a worthy alternative.

The MFCC coefficients proved to be the best independent feature for singing voice detection. The combination of MFCC with other features increased the classification results especially when the SVM and the J48 (for low minNumObj) classifiers were used. The best results are achieved when using all of the available features with the SVM and J48 (with low minNumObj) classifiers reaching values around 89%.

A novel data set, the Allegro data set, proved to be well constructed judging by the similarity in the obtained results when using the other data sets. This is a useful contribution to MIR because this data set will be available for other investigators.

6.2 Future Work

As mentioned before, the used features were selected mainly because they were available in the MARSYAS software particularly on the bextract function. It would be interesting to test other features like the Vibrato [29] or the Harmonic Attenuated Log Frequency Power Coefficients (HA-LFPC) [27] either independently or combine them with some used on this dissertation. Along with these experiments several texture window sizes could be used thus combining the experiments presented on sections 4.2.1 and 4.3 of this thesis. Another interesting approach would be using other classifiers like the GMM or the HMM because they may work better for certain features than the ones used on the experiments. An example of this application was used in [30] where the author tries to detect singing voice in music signals using the MFCC coefficients and the GMM.

Regarding feature combination it was verified that for certain classifiers the use of several features improves the results. An interesting approach would be to distinguish which coefficients of the features are truly relevant for singing voice detection. For example, although in one of the tests of the experiment presented in subsection 4.3, 94 attributes representing several feature coefficients were used, not all were important for the classification process. This can be demonstrated by the analysis of the tree model constructed by the J48 classifier when the minimum number of objects was equal to 200. The constructed tree model was very large but still not all of the existing attribute were used. This means that there are some attributes that enable the classifier to better separate the characteristics of voice from instruments and some that are irrelevant. The study of feature contribution or value is called feature relevance. Some preliminary techniques were rehearsed but due to lack of time the results were not pre-

sented in this report. It's possible to use some techniques to rank the attributes before the construction of the classification models. This will enable the classifiers to construct the models based on the most relevant attributes which may help the classification process. This study can be applied for example to the panning coefficients because voice spans from 200 to 2000Hz which corresponds to the middle frequency band coefficient as explained in section 4.1.2.

References

[1]. **Alex Loscos, Pedro Cano, Jordi Bonada.** *Low-Delay Singing Voice Alignment to Text.* In Proceedings of the ICMC, 1999.

[2]. **Andre Holzapfel, Yannis Stylianou.** *Singer Identification in Rembetiko Music.* In proceedings of SMC07, 2007.

[3]. **Kim, Y. E. and Whitman.** *Singer identification in popular music recordings using voice coding features.* Proceedings of the 3rd International Conference on Music Information Retrieval. Paris, France: IRCAM, pp. 164-169. 2002.

[4]. Singstar. [Online] http://www.singstargame.com.

[5]. **George Tzanetakis, Georg Essl, Perry Cook.** *Automatic musical genre classification of audio signals.* In Proceedings of the International Symposium on Music Information Retrieval (ISMIR), Bloomington, Indiana, 2001.

[6]. **Wang, Wei-Ho Tsai and Hsin-Min.** *Towards Automatic Identification of Singing Language in Popular Music Recordings.* In Proc. of the 5th International Conference on Music Information Retrieval (ISMIR), 2004.

[7]. **Stephen V. Rice, Comparisonics Corporation.** findsounds.com. *findsounds.com.* [Online] http://www.findsounds.com/.

[8]. Foafing the Music. *Foafing the Music.* [Online] Music Technology Group of the Universitat Pompeu Fabra, 2004-2006. http://foafdevel.searchsounds.net/index.html.

[9]. MIR Systems. *Music Information Retrieval Systems.* [Online] http://mirsystems.info/index.php?id=mirsystems.

[10]. VISNET II. [Online] http://www.visnet-noe.org/index.html.

[11]. SID. [Online] http://www.cost-sid.org/.

[12]. Digitópia. *Casa da Música.* [Online] INESC Porto, UCP, ESMAE , Porto Digital and Casa da Música. http://www.casadamusica.com/Education/default.aspx?channelID=D0EE8B4D-7953-4955-A1A6-AA054EABDAD2&id=2E3C54AF-5764-453A-860B-F81F99A29E84&l=D0EE8B4D-7953-4955-A1A6-AA054EABDAD2.

[13]. MARSYAS. [Online] http://marsyas.sness.net/.

[14]. Audacity. [Online] http://audacity.sourceforge.net/?lang=en.

[15]. **Goto, Masataka and Fujihara,Hiromasa.** *A Music Information Retrieval System Based on Singing Voice Timbre.* ISMIR Vienna, Austria, 2007.

[16]. **Mathieu Lagrange, Luís Gustavo Martins, Jennifer Murdoch, George Tzanetakis.** *Normalized Cuts for Predominant Melodic Source Separation.* s.l. : IEEE Transactions on Audio, Speech and Language Processing, vol.16, no.2, pp.278-290, February,2008.

[17]. **Hanna Lukashevich, Matthias Gruhne and Christian Dittmar.** *Effective Singing Voice Detection in Popular Music using ARMA Filtering.* Proc. of the 10th Int. Conference on Digital Audio Effects (DAFx-07), Bordeaux, France, 2007.

[18]. **Chi Hang Wong, Wai Man Szeto ,Kin Hong Wong.** *Automatic Lyrics Alignment for Cantonese Popular Music.* in Multimedia Systems Journal 2006.

[19]. **Slaney, E. Scheirer and M.** *Construction and evaluation of a robust multifeature speech/music discriminator.* in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Munich, 1997.

[20]. **George Tzanetakis, Randy Jones, and Kirk McNally.** *Stereo Panning Features for Classifying Recording Production Style.* In Proceedings of the International Conference on Music Information Retrieval (ISMIR), Vienna, Austria, September 2007.

[21]. **Alan L. Berenzweig, Daniel P. W. Ellis.** *Locating Voice Segments Within Music Signals.* in Proceedings of the IEEE workshop on Applications on Signal Processing to Audio and Acoustics (WASPAA), 2001.

[22]. **Matín Rocamora, Perfecto Herrera.** *Comparing audio descriptors for singing voice detection in music audio files.* Proceedings of 11th Brazilian Symposium on Computer Music, Sao Paulo, Brazil, 2007.

[23]. **Holzapfel, Andre.** Andre Holzapfel. [Online] http://www.csd.uoc.gr/~hannover/tmp/labeled_remb.tar.gz.

[24]. **Allegro, Pedro Luis.** *Singing Voice Detection in Polyphonic Music Signals.* [Online] 2008. http://www.fe.up.pt/~ee02193.

[25]. **Ellis, Dan.** [Online] http://labrosa.ee.columbia.edu/sounds/musp/scheislan.html.

[26]. **Frank, Ian H. Witten and Eibe.** *Data Mining: Practical machine learning tools and techniques.* San Francisco : Morgan Kaufmann, 2005.

[27]. **Tin Lay Nwe, Arun Shenoy, Ye Wang.** *Singing Voice Detection in Popular Music.* Technical report,Department of Computer Science, University of Singapore, Singapore, October 2004.

[28]. **Itakura, N. Sugamura e F.** "Speech analysis and synthesis methods developed at ECL in NTT - from LPC to LSP". *Speech Communication.* vol. 5, 1986, pp. 199-215.

[29]. **Lay, Haizhou LI and Tin.** *Vibrato-Motivated Acoustic Feature for Singer Identification.* Institute for Infocomm Research. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP2006), 2006, Toulouse, France.

[30]. **Yipeng Li, DeLiang Wang.** *Separation of Singing Voice From Music Accompaniment for Monaural Recordings.* Department of Computer Science and Engineering& Center of Cognitive Science, The Ohio State University. IEEE Transactions on Audio, Speech, and Language Processing, 2007.

Annex A

```
% Created by Marsyas
@relation lsp_ws1024_hs512_weka.arff
@attribute Mean_Mem40_LSP_1 real
@attribute Mean_Mem40_LSP_2 real
@attribute Mean_Mem40_LSP_3 real
@attribute Mean_Mem40_LSP_4 real
@attribute Mean_Mem40_LSP_5 real
@attribute Mean_Mem40_LSP_6 real
@attribute Mean_Mem40_LSP_7 real
@attribute Mean_Mem40_LSP_8 real
@attribute Mean_Mem40_LSP_9 real
@attribute Mean_Mem40_LSP_10 real
@attribute Mean_Mem40_LSP_11 real
@attribute Mean_Mem40_LSP_12 real
@attribute Mean_Mem40_LSP_13 real
@attribute Mean_Mem40_LSP_14 real
@attribute Mean_Mem40_LSP_15 real
@attribute Mean_Mem40_LSP_16 real
@attribute Mean_Mem40_LSP_17 real
@attribute Mean_Mem40_LSP_18 real
@attribute Std_Mem40_LSP_1 real
@attribute Std_Mem40_LSP_2 real
@attribute Std_Mem40_LSP_3 real
@attribute Std_Mem40_LSP_4 real
@attribute Std_Mem40_LSP_5 real
@attribute Std_Mem40_LSP_6 real
@attribute Std_Mem40_LSP_7 real
@attribute Std_Mem40_LSP_8 real
@attribute Std_Mem40_LSP_9 real
@attribute Std_Mem40_LSP_10 real
@attribute Std_Mem40_LSP_11 real
@attribute Std_Mem40_LSP_12 real
@attribute Std_Mem40_LSP_13 real
@attribute Std_Mem40_LSP_14 real
@attribute Std_Mem40_LSP_15 real
@attribute Std_Mem40_LSP_16 real
@attribute Std_Mem40_LSP_17 real
@attribute Std_Mem40_LSP_18 real
@attribute output {instrument, voice}

@data
% e:\instrument2_class\10_2.wav
0.006202,0.009819,0.014867,0.017294,0.022264,0.025476,0.029703,0.032625,0.037310,0.041658,0.04
5960,0.050240,0.054594,0.057899,0.062057,0.066180,0.068772,0.072874,0.038734,0.061320,0.092844
,0.108003,0.139041,0.159095,0.185494,0.203744,0.233000,0.260152,0.287017,0.313750,0.340939,0.3
61580,0.387543,0.413295,0.429479,0.455097,instrument
0.010714,0.018790,0.028186,0.035002,0.044066,0.050595,0.058912,0.065607,0.074048,0.082821,0.09
1256,0.099454,0.108980,0.115561,0.124594,0.131879,0.137930,0.145892,0.047311,0.081991,0.123054
,0.152579,0.192090,0.220542,0.256801,0.285979,0.322780,0.361017,0.397788,0.433536,0.475032,0.5
03721,0.543096,0.574851,0.601224,0.635929,voice
```

Annex B

The original Ellis data set was separated in two folders, “Sing” and “NoSing”. The “NoSing” folder contained music files with instruments and silences, while the “Sing” folder contained music files with voiced sounds accompanied or not by instruments. After carefully analyzing the data set it was evident that some songs in the “Sing” folder contained segments that did not have voiced sounds thus compromising the integrity of this data set. The corrupt songs were corrected by erasing the instrument sections detected by hearing. The used program was audacity.

To better ascertain the erased segments, the detailed list featuring the songs and respective segments is presented in Table 11.

Table 11: Discarded voice segments for Ellis data set

	Discarded segment nº 1		Discarded segment nº 2	
File name	**start(seconds)**	**stop(seconds)**	**start(seconds)**	**stop(seconds)**
20_2	5,5	6,8		
26_1	9	10,1		
27_3	2,2	3,3	5,7	6,7
37_1	11,7	12,3		
50_3	6,1	7,4		

Annex C

To better understand the Allegro data set, the detailed description of each wav files will be listed.

Table 12: Allegro data set song descriptions

	Original Song	Artist	Album	Voice segment duration(seconds)	Instrument segment duration(seconds)
1	Here Without You	3 Doors Down	Away from the sun	26,9	16,5
2	You shook me all night long	AC/DC	Back In Black	8,5	14,5
3	Rastaman Live Up	Bob Marley	Confrontation	23	12,5
4	Georgia On My Mind	Ray Charles	His Greatest Hits CD1	40,5	16
5	Lágrima	Amalia Rodrigues	Lágrima	22,6	13,5
6	Let´s Dance	David Bowie	Let´s Dance	13	13,5
7	I Believe I Can Fly	R. Kelly	R	6,2	20
8	Época	Gotan Project	La Revancha del Tango	8,3	21,5
9	I Don't Feel Like Dancing	Scissor Sisters	Ta-Dah	26,9	19
10	Hit the Road Jack	Ray Charles	His Greatest Hits CD1	26,2	5,5
11	Jammin	Bob Marley	Legend: the Best	16,6	13,5
12	A Kind of Magic	Queen	Greatest Hits 2	32,7	21,2
13	Clandestino	Manu Chao	Clandestino	25,9	17,5
14	Garota do Ipanema	Toquinho & Vinicius	O Poeta e o Violão	50,3	0
15	The Girl from Ipanema	Antonio Carlos Jobim	Antonio Carlos Jobim - Brasileiro	0	49,8
16	Freestyler	Boomfunk MC´s	Freestyler	40,4	53,6
17	The Voice Within	Christina Aguilera	Stripped	72,7	6,8
18	The Final Countdown	Europe	The Final Countdown	24.9	78,7
19	Straight Ahead	Jimi Hendrix	The Cry of Love	15,3	20,7
20	Cosmic Girl	Jamiroquai	Travelling without moving	22,2	26
21	Diamonds on the inside	Ben Harper	Diamonds on the inside	26,9	17,5
22	Given Up	Linkin Park	Minutes to Midnight	28,5	23,9
23	Moonrise	Nitin Sawhney	All mixed up 2004	23,3	83,4
24	Quando um homem tem uma mangueira no quintal	Vanessa da Mata	Sim	17,8	12,7
25	Thriller	Michael Jackson	Thriller	19,9	13,6
			Total	594,6	591,4

Annex D

The next paragraph represents the tree model constructed for the J48 classifier using the MFCC coefficients extracted from the Allegro data set with minimum number of objects per leaf equal to 6000.

```
Std_Mem40_MFCC3_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= 0.874796
|   Mean_Mem40_MFCC3_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= 0.529636: voice
(6503.0/2002.0)
|   Mean_Mem40_MFCC3_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > 0.529636
|   |   Mean_Mem40_MFCC5_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= 0.171813: voice
(6456.0/2268.0)
|   |   Mean_Mem40_MFCC5_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > 0.171813
|   |   |   Std_Mem40_MFCC0_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= 5.280985
|   |   |   |   Std_Mem40_MFCC4_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= 0.808107
|   |   |   |   |   Mean_Mem40_MFCC1_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= 7.227011:
instrument (37935.0/10841.0)
|   |   |   |   |   Mean_Mem40_MFCC1_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > 7.227011:
voice (6109.0/2535.0)
|   |   |   |   Std_Mem40_MFCC4_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > 0.808107: voice
(6020.0/2345.0)
|   |   |   Std_Mem40_MFCC0_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > 5.280985: instrument
(8163.0/866.0)
Std_Mem40_MFCC3_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > 0.874796
|   Mean_Mem40_MFCC0_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= -70.256976: instrument
(6021.0/2408.0)
|   Mean_Mem40_MFCC0_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > -70.256976: voice
(24516.0/3662.0)
```

Annex E

The next paragraph represents the tree model constructed for the J48 classifier using the MFCC coefficients extracted from the Ellis data set with minimum number of objects per leaf equal to 4500.

```
Mean_Mem40_MFCC1_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= 5.606045
|   Std_Mem40_MFCC5_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= 0.710267
|   |   Std_Mem40_MFCC0_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= 4.355468
|   |   |   Std_Mem40_MFCC3_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= 0.600835
|   |   |   |   Mean_Mem40_MFCC0_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= -64.403143:
instrument (3624.0/1020.0)
|   |   |   |   Mean_Mem40_MFCC0_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > -64.403143: voice
(4953.0/1742.0)
|   |   |   Std_Mem40_MFCC3_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > 0.600835: voice
(9046.0/1891.0)
|   |   Std_Mem40_MFCC0_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > 4.355468: instrument
(3641.0/1344.0)
|   Std_Mem40_MFCC5_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > 0.710267: voice (3520.0/386.0)
Mean_Mem40_MFCC1_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > 5.606045
|   Std_Mem40_MFCC3_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= 0.744689: instrument
(10199.0/1298.0)
|           Std_Mem40_MFCC3_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0    >    0.744689:    voice
(3832.0/1651.0)
```

Annex F

The next paragraph represents the tree model constructed for the J48 classifier using the MFCC coefficients extracted from the Allegro data set with minimum number of objects per leaf equal to 4500 after being reconstructed using peak clustering.

```
Mean_Mem40_MFCC0_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= -95.347635: instrument
(15127.0/2393.0)
Mean_Mem40_MFCC0_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > -95.347635
|   Mean_Mem40_MFCC2_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= 1.959757
|   |   Mean_Mem40_MFCC10_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= -0.378332: voice
(6378.0/1649.0)
|   |   Mean_Mem40_MFCC10_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > -0.378332
|   |   |   Mean_Mem40_MFCC8_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= -0.882769: instru-
ment (14019.0/4941.0)
|   |   |   Mean_Mem40_MFCC8_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > -0.882769
|   |   |   |   Mean_Mem40_MFCC0_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= -85.885697
|   |   |   |   |   Std_Mem40_MFCC3_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= 2.626005
|   |   |   |   |   |   Mean_Mem40_MFCC0_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= -
90.925282: instrument (11846.0/4261.0)
|   |   |   |   |   |   Mean_Mem40_MFCC0_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > -
90.925282
|   |   |   |   |   |   |   Std_Mem40_MFCC0_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <=
5.629456: instrument (9329.0/3849.0)
|   |   |   |   |   |   |   Std_Mem40_MFCC0_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 >
5.629456: voice (6651.0/2495.0)
|   |   |   |   |   Std_Mem40_MFCC3_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > 2.626005:
voice (8480.0/3271.0)
|   |   |   |   Mean_Mem40_MFCC0_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > -85.885697: voice
(25710.0/8541.0)
|   Mean_Mem40_MFCC2_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > 1.959757: voice
(6881.0/1389.0)
```

Annex G

The next paragraph represents the tree model constructed for the J48 classifier using the MFCC, LSP and LPCC coefficients extracted from the Ellis data set with minimum number of objects per leaf equal to 2500.

```
Mean_Mem40_LSP_5 <= 0.739791: instrument (7921.0/532.0)
Mean_Mem40_LSP_5 > 0.739791
|   Mean_Mem40_LPCC_10 <= 90.060186
|   |   Mean_Mem40_LSP_18 <= 2.888908
|   |   |   Mean_Mem40_LSP_11 <= 1.807011
|   |   |   |   Std_Mem40_MFCC3_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 <= 0.828737
|   |   |   |   |   Mean_Mem40_LSP_4 <= 0.637198: instrument (3761.0/1240.0)
|   |   |   |   |   Mean_Mem40_LSP_4 > 0.637198
|   |   |   |   |   |   Std_Mem40_LSP_2 <= 0.024836: instrument (2578.0/1230.0)
|   |   |   |   |   |   Std_Mem40_LSP_2 > 0.024836: voice (4936.0/1225.0)
|   |   |   |   Std_Mem40_MFCC3_Power_FFT512_WinHamming_HopSize512_WinSize512_AudioCh0 > 0.828737: voice
(2890.0/253.0)
|   |   |   Mean_Mem40_LSP_11 > 1.807011: voice (7686.0/321.0)
|   |   Mean_Mem40_LSP_18 > 2.888908
|   |   |   Mean_Mem40_LPCC_7 <= -0.250009: voice (3482.0/1574.0)
|   |   |   Mean_Mem40_LPCC_7 > -0.250009: instrument (3030.0/353.0)
|   Mean_Mem40_LPCC_10 > 90.060186: instrument (2531.0/367.0)

Number of Leaves  :      9

    Size of the tree :   17
```

As the tree gets further pruned, the MFCC and LPCC coefficients tend to be ignored since the LSP coefficients are “higher” on the tree model.