School of Electronic, Electrical and Systems Engineering

MSc Embedded Systems

Final project 2014-2015

Detection of Leitmotivs in audio recordings

Luciano Ernesto Juarez Rivera ID 1487897

Supervisor: Dr. Peter Jancovic

01/09/2015

ABSTRACT

The development, enhancement and access to computers and the Internet, has led to a dramatic increase on the available digital multimedia data, such as music, video, images and speech recordings. In recent years, the necessity of browsing through this data has reached high levels of demand, calling for the development of automatic intelligent systems to browse, categorize, and identify patterns or a specific characteristic of the digital information. Additionally, subject specialist systems are an area of interest for researchers and academics, such as music structure analysis, in example. This project focused on the specialist area of music analysis, as aimed to develop a automatic detection tool of leitmotivs in live audio recordings, which are small passages of music that give character to a piece, and are of high interest of music experts.

Two main audio feature extraction characterization methods are explored, Chroma and MFCCs, as well as different data classification methods, such as HMM, GMM, and SVM. As a final baseline, a HMM + GMM approach with MFCC features was selected. The experiments were conducted with the use of the HTK toolkit to design and test the system. Training and testing of four different leitmotivs, Grubel, Nibelungen, Ring, and Vertrags was performed with an excerpt of an opera recording of Wagner. Two main system training methods were performed, embedded-unit and isolated-unit, with different HMM parameters. The best overall performance of each could be measured with an F1 score 0.1013 for the embedded-unit, and 0.75 for the isolated-unit.

Results suggest that this type of approach is suitable for this application, as the majority of the leitmotivs were detected successfully. Further research involves the exploration of testing different audio feature representations, as well as other data classification techniques.

ACKNOWLEDGMENTS

I would like to thank:

Dr. Peter Jancovic for the support and very helpful advice provided during this project.

My fiancée, Rebecca Stanyer, for the unconditional moral support given to me throughout the whole process.

My family for their acceptance and unconditional support.

CONTENTS

List of Figures

List of Tables

1. INTRODUCTION

Audio recordings, particularly music recordings, are a vital element for modern society, as they form part of numerous occasions regularly. As digital technology develops, and with the growth of a unified World Wide Web, the available amount of multimedia data, such as music recordings, has increased considerably during the past recent years. This has created a necessity of the development of automatic tools with the capability of processing and analyzing this data, in order to extract relevant information or knowledge from it. To fulfill this need, a whole area of research has been focusing on this aspect, referred to as Music Information Retrieval (MIR). Derived from MIR, new applications are gaining more relevance, such as automatic music browsers, music genre classifiers, artist identification, chord recognition and melody and bass line estimation (Casey et al 2008).

On the other hand, other aspects such as music structure analysis are a subject of interest for musicians worldwide. Several works had centered on pattern detection within musical pieces (Logan and Chu, 2000, Dannenberg & Hu, 2003), but the complexity of making robust systems arise when dealing with polyphonic musical pieces (several instruments playing collectively).

From a music point of view, particular interest exists in understanding and identifying distinctive patterns that occur in some styles of music, referred to as motives. This identification is normally performed by a musician with careful and dedicated listening to recordings, which can be significantly tedious when analyzing long pieces, such as complete symphonies. Specifically, the motives of composer Richard Wagner are a topic that has been widely studied, as his denominated leitmotivs serve not only as a structural element in music, but as well as dramatic and character for the music itself (McShan, 1997). The importance of the leitmotivs in music is such that experiments revolving around them had been performed, such as Albrecht & Frieler (2014) in their work 'The perception and recognition of Wagnerian leitmotifs in multimodal conditions', in which they studied the capability of musicians and non-musicians to correctly identify four leitmotivs: Grubel, Nibelungen, Ring and Vertrags. A digital approach to this topic is been proved to be of interest in the work 'Using Digital Libraries in the Research of the Reception and Interpretation of Richard Wagner's Leitmotifs' from Dreyfus & Rindfleisch (2014), where they explore the application the use of digital databases in a musicology perspective, focusing on the reception and the processing of the different interpretations of Wagner's leitmotivs.

Based on the fact that there is a particular focus on the use of digital tools to the analysis of Richard Wagner's leitmotivs, the existence of MIR techniques for musical analysis, and the importance of the leitmotiv in musicology, this project focus on the development of an automatic system for the detection of leitmotivs across audio recordings. Although the concept of detecting musical motives has been previously addressed in another music genres (Ross et al, 2012, Ishwar et al, 2013), non existing work was found on particularly detecting the Wagnerian leitmotivs with an automatic tool. This project aims to develop such tool, for which the exploration of different audio representation features is first considered, and then the data classification techniques are analyzed. Subsequently, the development of the system is explained, the results are analyzed and finally, conclusions and further research are discussed.

2. AUDIO SIGNAL PROCESSING

The acoustic audio signal has a high information content, which needs to be extracted and converted to a domain that is mathematically workable. Three main approaches are categorized to describe musical content: Metadata, High Level Music Content Description and Low level audio features. The first one refers to the segment of the digital file that contains the relevant information about a musical piece, such as artist, year, album etc. The second one refers to music terminology descriptors such as pitch, harmony and rhythm. The third one refers to a signal processing point of view, by reading the information of the music signal itself (Casey et al, 2008). In this application, an analysis in the music recording itself is to be performed to identify the appearances of specific audio pieces (previously defined leitmotivs); therefore the selected approach is to perform low level audio feature extraction. According to Casey (2008), these features are generally classified in three different segmentation ways: frame-based, beat-synchronous and statistical measures, which are mostly based on the short-time spectrum of the signal. As a common starting point for the different low level audio feature extraction approaches, the Fast Fourier Transform (FFT, which in essence is a 'faster' version of a Discrete Fourier Transform) is applied to a windowed audio signal. Figure 2.1 depicts the steps followed on four common extraction feature methods.

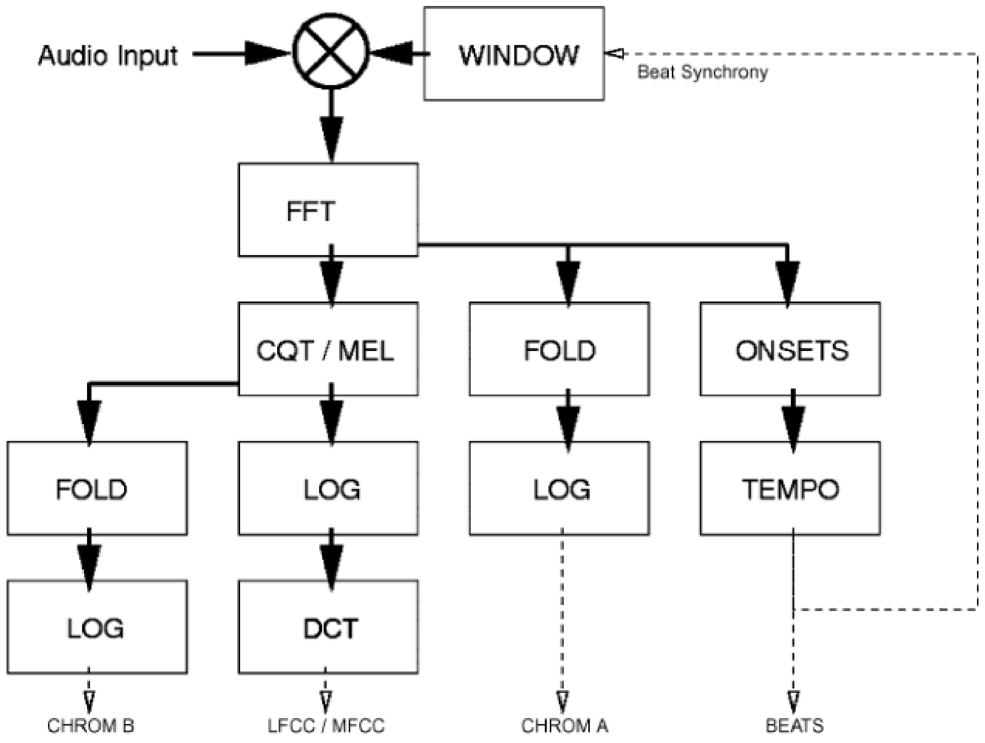


Figure 2.1 Common low level audio feature extraction methods for MIR (Taken from Casey et al, 2008 referenced in [5])

Two main features will be analyzed, MFCC and Chromas, in order to determine the suitability of each for the particular application.

2.1 Discrete Fourier Transform

The Fourier Transform decomposes a signal from the time domain into its ‘pieces’ in the frequency domain. The continuous Fourier Transform is defined as (Weisstein, 2002):

$$f(v) = \mathcal{F}_t[f(t)]\,(v) = \int_{-\infty}^{\infty} f(t)e^{-2\pi i v t}\,dt \tag{2.1}$$

Considering the case of a discrete function in which a sample f_k corresponds to a point of the continuous signal $f(t_k)$ with a time between samples $k\,\Delta$, with $k = 0, ..., N\text{-}1$, the discrete Fourier Transform is given as a result of $F_n \equiv \mathcal{F}_k\left[\{f_k\}_{k=0}^{N-1}\right](n)$ where (Weisstein, 2002):

$$F_n \equiv \sum_{k=0}^{N-1} f_k e^{-2\pi i n k/N} \tag{2.2}$$

Which from a signal spectrum is obtained. It is notice that for the process of feature extraction, the whole signal is segmented in small frames, in regular intervals defined by a time window, and for each frame the DFT is obtained, resulting in a set of spectrums of the processed signal, referred as spectrogram.

2.2 Mel Frequency Cepstral Coefficients (MFCC)

MFCCs come as a result of processing the spectral information of a signal. Once a FFT is performed in the different segmented frames, those are first wrapped around following the Mel-scale, to adapt to the properties of human pitch perception (Molau et al, 2001). The Mel-scale is a mapping between real frequency value and perceived frequency of the human hearing, which is linear approximately below 1 KHz and logarithmic above that value (Logan, 2000). This value is taken as a starting reference point (1 KHz = 1000 mels), from which any value in mels on the scale can be computed for any frequency *f* with the equation (Hasan & Rahman, 2004):

$$\mathrm{mel}(f)=2595\ \log_{10}\left(1+\frac{f}{700\ \mathrm{Hz}}\right) \tag{2.3}$$

Once transformed to the Mel correspondent value, the frames are passed through a set of filters, to be segmented into frequency bins. These filters have an overlap value to prevent information loss. Figure 2.2 shows the relation between the filter banks and the Mel-scale. Finally, the amplitude (phase discarded) of the output frames are taken logarithm, and the Discrete Cosine Transform is applied to obtain the final MFCC vector.

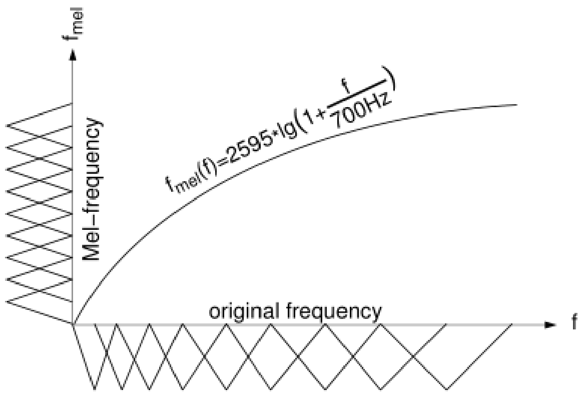


Figure 2.2 The behavior of the filter bank (triangular) corresponding to the Mel-scale (Taken from Molau et al, 2001 referenced in [22])

2.3 Chroma features (Pitch Class Profiles)

Pitch Class Profiles (PCP) are vectors which contain the distribution of the energy content of a given signal through a set of predefined classes, where each class represents the twelve semitones in traditional Western music (Cabral et al, 2005). Parting from obtaining the DFT of a windowed signal, the PCP calculation consists of segmenting the spectral components into regions (that can be seen as a window of fixed size), in which for each region every single frequency component is mapped to a correspondent PCP class, depending on its frequency value. Formally, the frequency value is calculated by the equation (Cabral et al, 2005):

$$\mathrm{p}\,(\,k\,) = \left[12\ \log_2\left(\frac{k}{N}\,\frac{f_{sr}}{f_{ref}}\right)\right] \bmod 12 \tag{2.4}$$

Where k corresponds to a frequency bin in the spectrum, N is the number of frequency bins, f_{sr} corresponds to the sampling rate and f_{ref} works as a reference frequency, allocated in PCP vector index 0. After each region is processed and allocated in the different classes, a set of PCP vectors is obtained by summing the magnitudes of all frequency bins corresponding to the same group or class, formally described as (Sheh & Ellis, 2003):

$$PCP[p] = \sum_{k:p(k)=p} |X[k]|^2 \tag{2.5}$$

The results in a 12-dimensional vector where each field contain the energy content for the different pitch classes. Finally, to classify the different notes in the same bin regardless of octave, folding of the PCP vector is applied.

2.4 Feature selection

Due to its effectiveness MFCCs had been widely used in speech recognition applications (Milner, & Shao, 2002, Hasan & Rahman, 2004). Although Chroma features had seem to be most commonly used in music information retrieval tasks such as chord and key recognition, music structure analysis (Casey et al, 2008) and audio thumb nailing (Bartsch & Wakefield, 2001), MFCCs had been proven to be suitable for music analysis (Logan, 2000) . Considering these factors, the MFCC features approach will be performed in this project; initially because the interest is in identifying a pattern as an overall picture rather than its components, and finally because MFCC features can be easily computed from audio recordings with the HTK toolkit.

3. MATHEMATICAL MODELING

3.1 Data classifying techniques

Music Information retrieval systems had been mathematically modelled popularly using approaches such as the Hidden Markov Models (HMM) (Foote, 1999, Sheh & Ellis, 2003), Gaussian Mixture Models (GMM) (Marlot, 2004, Ellis, 2007), and Support Vector Machines (SVM) (Guo & Li, 2003,Mandel & Ellis, 2005). One very important factor to consider modelling approach selection is the capability of sequence modelling, as temporal identification of a pattern is required. Previous work of a related application, chord and key recognition has achieved better results with HMMs, as music is modelled as a sequence over time (Casey et al 2008). Due to its relatively easy implementation, as well as for the availability of existing design tools, a combined approach of the properties of HMMs to model sequenced frames, and of GMMs for data clustering, the modelling will consist of a HMM + GMM integration, in which each HMM state will be associated with a GMM for robustness. Both models are described further.

3.1.1 Hidden Markov Models

A Markov Model can be perceived as a model of a network consisting of nodes (states) and arcs (transitions), in which each state generates an observation, therefore, a finite number of observation sequences can be generated from each model, depending on the number of states and transitions. According to Rabiner (1989) a Markov model consists of a set of N states S_1, S_2 ... S_N, in which a change of state occurs in the model at regular time intervals *t*, and the Markov assumption that the new state at time *t* will depend only on the previous observation, described as:

$$P\left[S_t = S_j \mid S_{t-1} = S_i, S_{t-2} = S_k, \ldots\right] = P\left[S_t = S_j \mid S_{t-1} = S_i\right] \tag{3.1}$$

Where only the time independent transitions are allowed, therefore, the transition probability matrix A = $\{a_{ij}\}$ is built up by:

$$a_{ij} = P\left[S_t = S_j \mid S_{t-1} = S_i\right],\ 1 \leq i, j \leq N \tag{3.2}$$

In which the transition matrix A has the property of:

$$a_{ij} \geq 0 \tag{3.3a}$$

$$\sum_{j=1}^{N} a_{ij} = 1 \tag{3.3b}$$

Since the coefficients follow standard stochastic limitations. The idea behind a Markov Model is that, given a sequence of observations, the model can tell the exact sequence of states the system had to follow to give that particular sequence as an output. This model is useful when managing sequences with a finite, small number of states and observations, since each state is associated to a particular observation. When an observation sequence can be generated by multiple state sequences, the Markov Model transforms into an unpractical solution, since every single possibility would be needed to create a transition matrix, generating a virtually infinite dimension matrix. Therefore, an alternative to this approach is the Hidden Markov Model (HMM). The HMM is a model that has the main characteristics of a Markov Model, but works under the principle that the states of the model can emit multiple observations. In other words, the HMM calculates the probability that the observation sequence *O* was generated by a state sequence *S*. In comparison, the HMM has an additional probability matrix, which contains the probabilities of a state generating a determined observation.

As formally described by Blunsom (2004), a Hidden Markov Model is represented by:

$$\lambda=(A, B, \pi) \tag{3.4}$$

Where A is the transition probability matrix, in which each element has the probability of state *i* being the state from which the system moves to state *j*, as described in (3.2).

B is the observation probability matrix, in which each element has the probability of state *j* generating observation *k*, time independent and described as:

$$B= [b_i\ (k)],\ b_i\ (k) = P(x_t= v_k\ /\ q_t= s_i) \tag{3.5}$$

Notice that s_i is an element of the state alphabet vector *S*, q_t is an element of the state sequence vector *Q*, and v_k is an element of the observation alphabet vector *V*.

Since the model is at an unknown state and requires initialization, π represents the initial probability matrix, defined as:

$$\pi = [\pi_i],\ \pi_i = P\ (q_1= s_i) \tag{3.6}$$

Finally, the model has two assumptions. The first, previously described as a single Markov model, which states that the current state only depends on the previous state. The second states that the observation output for a state at a time instant is completely independent to previous observations and states. Figure 3.1 shows an example of a simple but full example of a Hidden Markov Model.

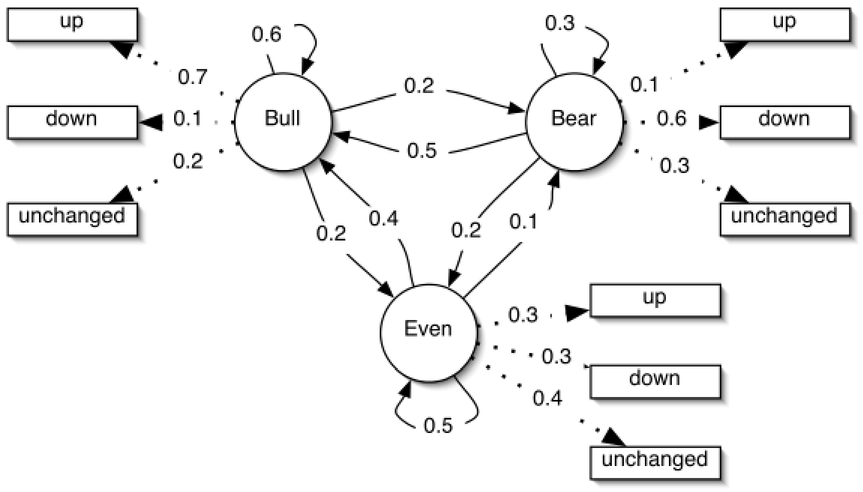


Figure 3.1 A full example of a Hidden Markov Model (Taken from Blunsom, 2004 referenced in [3])

The HMMs parameters can be re-estimated via Forward/Backward probabilities and Baum Welch algorithm; in where the forward probability of finding a HMM after a sequence of observations, α_j (t) for $1 < j < N$ and $1 < t \leq T$ is calculated by (Young et al, 2006):

$$\alpha_j(t) = \left[\sum_{i=2}^{N-1} \alpha_i(t-1)a_{ij}\right] b_j\,(o_t) \tag{3.7a}$$

With initial conditions denoted by:

$$\alpha_1(1)=1 \tag{3.7b}$$

$$\alpha_j\,(1)=a_1 b_j\,(o_1) \tag{3.7c}$$

And a final condition described as:

$$\alpha_N(T) = \sum_{i=2}^{N-1} \alpha_i(T)a_{iN} \tag{3.7d}$$

While the backward probability for β_i (t) $1 < i < N$ and $1 > t \geq T$, of finding observation sequence vector O given a HMM, is calculated by (Young et al, 2006):

$$\beta_i(t) = \sum_{j=2}^{N-1} a_{ij} b_j(o_{t+1})\;\beta_j(t+1) \tag{3.8a}$$

Whose initial conditions are given by:

$$\beta_i(T) = a_{iN} \tag{3.8b}$$

And final conditions:

$$\beta_1(1) = \sum_{j=2}^{N-1} a_{1j} b_j(o_1)\ \beta_j(1) \tag{3.8c}$$

The total probability *P* can be calculated by both probabilities with (Young et al, 2006):

$$P = p(O|\lambda) = \alpha_N(T) = \beta_1(1) \tag{3.9}$$

Once the probabilities are calculated, re-estimation is achieved by using the matched state sequences and updating the model parameters via Baum Welch Re-estimation. Notice that correct selection of the number of emitting states of a HMM can improve significantly the overall model effectiveness.

3.1.2 Gaussian Mixture Models

Gaussian Mixture models are a more fine distribution than a single Gaussian probabilistic distribution. Formally, a GMM is the weighted sum of *M* Gaussian mixture components, described by the equation (Reynolds, 2009):

$$p(x|\lambda) = \sum_{i=1}^{M} w_i\, g(x|\mu_i, \Sigma_i) \tag{3.10}$$

Where **x** is a D-dimensional data vector (data or samples), w_i, $i = 1, \ldots, M$, are the weights or influence of each mixture, and $g(x|\mu_i, \Sigma_i)$, $i = 1, \ldots, M$, are the number of mixture components. Each component follows the form (Reynolds, 2009):

$$g(x|\mu_i, \Sigma_i) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} \exp\left\{-\frac{1}{2}(x-\mu_i)' \Sigma_i^{-1} (x-\mu_i)\right\} \tag{3.11}$$

In which μ_i is the mean vector, and Σ_i is the covariance matrix. Parameters are represented by the notation (Reynolds, 2009):

$$\lambda = \{w_i, \mu_i, \Sigma_i\},\ i = 1, \ldots, M \tag{3.12}$$

The idea is to estimate a set of GMM parameters, λ, that best models or fits the data distribution along the values of the training data vectors. According to Reynolds (2009), the most popular approach is to use the Maximum Likelihood estimation, which has as objective to model the GMM parameters that maximize the likelihood of the GMM based on the training data; for a sequence *T* of training data vectors $X = \{x_1, \ldots, x_t\}$, the likelihood of the GMM can be written as:

$$p(X|\lambda)=\prod_{t=1}^{T}p(x_t|\lambda) \tag{3.13}$$

Since this is a non-linear relationship between the GMM parameters, equation 3.10 cannot be directly applied. To solve this problem, expectation-maximization (EM) algorithm is applied. Reynolds (2009) describes the algorithm as follows: EM takes as a basis the idea of starting with a reference GMM, with parameters λ, to estimate a new model λ', such that the condition $p(X|\lambda') \geq p(X|\lambda)$ is satisfied, and the new obtained model is set as the new reference, performing this iteratively until a convergence level is reached. The new parameters are calculated with the formulas:

Weights
$$w'_i=\frac{1}{T}\sum_{t=1}^{T}\Pr(i|x_t,\lambda) \tag{3.14}$$

Means
$$\mu'_i=\frac{\sum_{t=1}^{T}\Pr(i|x_t,\lambda)\ x_t}{\sum_{t=1}^{T}\Pr(i|x_t,\lambda)} \tag{3.15}$$

Variances
$$\sigma'^2_i=\frac{\sum_{t=1}^{T}\Pr(i|x_t,\lambda)\ x_t^2}{\sum_{t=1}^{T}\Pr(i|x_t,\lambda)}-\mu'^2_i \tag{3.16}$$

Finally, the posterior probability for component *i*, is computed by (Reynolds, 2009):

$$\Pr(i|x_t,\lambda)=\frac{w_i\ g(x_t\mid\mu_i,\Sigma_i)}{\sum_{k=1}^{M}w_k\ g\left(x_t\mid\mu_k,\Sigma_k\right)} \tag{3.17}$$

Figure 3.2 shows an example of a Gaussian Mixture Model.

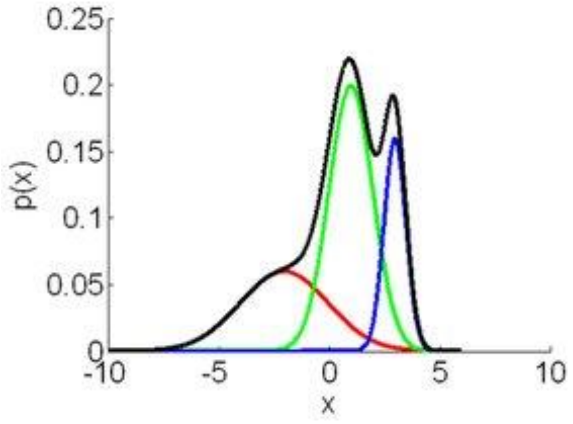


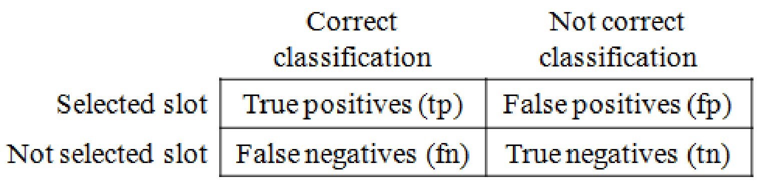
Figure 3.2 Gaussian Mixture Model (Taken from Doria, 2010 referenced in [7]).

3.2 Performance measures

Information retrieval systems extract relevant information with a determined level of performance, which is desired to be as high as possible, while minimizing the different type of errors that it can produce. Measuring a system's performance is elemental to tune its parameters and optimize the functionality. To evaluate the system's performance, the standard Precision, Recall and F-score will be used.

3.2.1 Precision and Recall

Precision and Recall are commonly used standard measures of performance for information retrieval systems, in which Precision focus evaluating the percentage of selected units that are correct (insertion errors), while Recall aims to measure the percentage of correct items that were selected (deletion errors) (Makhoul et al, 1999). The procedure consists of labeling different slots according to a 2 by 2 contingency table as described by Powers (2011), in which four classifications are shown: true positives, false positives, true negatives and false negatives.



	Correct classification	Not correct classification
Selected slot	True positives (tp)	False positives (fp)
Not selected slot	False negatives (fn)	True negatives (tn)

Figure 3.3 The Precision and Recall contingency table

Considering a hit = tp, the sum of substitutions and insertions = fp and the sum of substitutions and deletions = fn, the corresponding Precision and Recall values can be obtained by (Makhoul et al, 1999):

$$\mathrm{P} = \frac{tp}{tp + fp} \tag{3.18}$$

$$\mathrm{R} = \frac{tp}{tp + fn} \tag{3.19}$$

Due to the interest of having one single standardized measure, that takes into account these two factors, the F-measure is defined as the weighted harmonic mean of P and R and is computed by (Makhoul et al, 1999):

$$F = \left[\frac{\alpha}{P} + \frac{1\text{-}\alpha}{R}\right]^{-1} = \frac{PR}{(1\text{-}\alpha)P + \alpha R} \tag{3.20}$$

Where α is a weight constant value, between 0 and 1. A common value for α = 0.5, which defines equal weight for both P and R, reducing F to the equation (Makhoul et al, 1999):

$$F = \frac{2\,PR}{(1\text{-}\alpha)P + \alpha\, R} \tag{3.21}$$

Notice that a tradeoff exists between precision and recall, and depending on to which measure to pay attention to, α is modified accordingly.

4. DETECTION SYSTEM DEVELOPMENT

The selected classification method was the HMM approach with the use of MFCC features. To develop the system, the combined use of Matlab and the HTK Toolkit Version 3.4 was implemented. HTK is an open source set of applications designed to create and modify Hidden Markov Models, developed at the Machine Intelligence Laboratory (formerly known as the Speech Vision and Robotics Group) of the Cambridge University Engineering Department. The set of steps for building the system comprise the creation and configuration of specific HTK files, processing of the acoustic signal data to extract the features and training of the HMM. Subsequently testing the recognizer, and performing evaluation measures.

Recordings (data) are required both for training and testing, which were provided by the project supervisor.

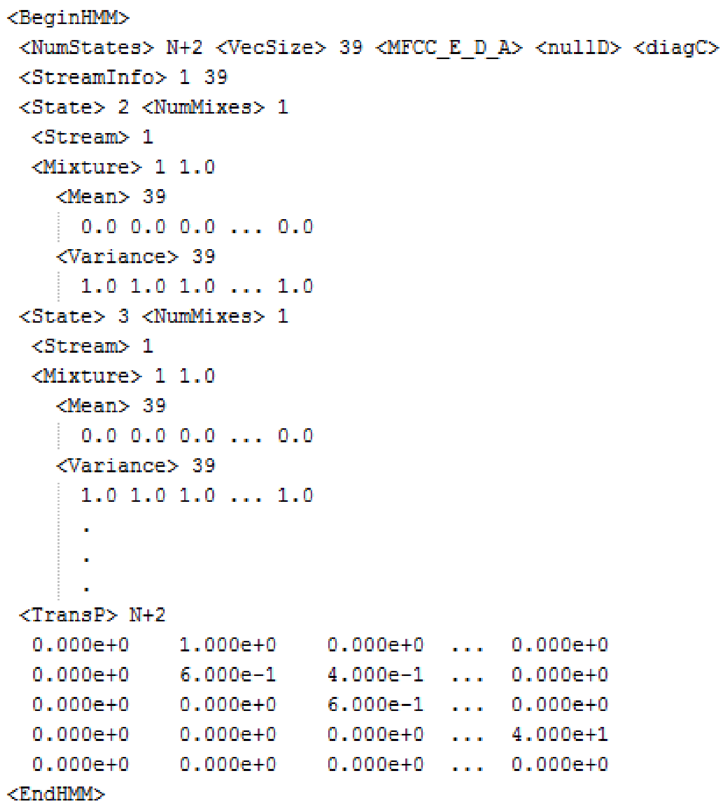
4.1 Building the detector

4.1.1 HTK file configuration

In order to build and construct HMMs with HTK, a series of files were created. The working environment of HTK requires the use of a HMM prototype definition file, a network for the detection process, a reference dictionary for labeling the recognized frames, training and testing label containing the categorization of the data, variable configuration file, and a series of lists containing the names of the elements to detect (commonly called word list), as well as the location in disk of the training and testing data.

Prototype file

This is a file containing the initial definitions for the HMMs. It works as a ‘skeleton’ to derive the initial parameter estimated HMM, based on the training data. Since the produced HMM will be composed by the same number of states as the prototype, this starting reference file will vary its number of states, mixtures and characteristics depending on the desired model to produce. The basic structure of any prototype file, which is manually created with a text editor, can be appreciated in Figure 4.1. In this file, N is the desired number of emitting states for a particular model. The first state has initial configurations including the number of states, vector size which represents the number of elements of the mean and variance vectors, and features parameter type which for this case MFCC_E_D_A is configured. Notice that the transition probability matrix must be of dimensions N+2 x N+2, where the first row is used as initial probability, and the last row contains zeros only. For each particular system setup, the number of states in the prototype file will be different.



```
<BeginHMM>
 <NumStates> N+2 <VecSize> 39 <MFCC_E_D_A> <nullD> <diagC>
 <StreamInfo> 1 39
 <State> 2 <NumMixes> 1
  <Stream> 1
  <Mixture> 1 1.0
    <Mean> 39
      0.0 0.0 0.0 ... 0.0
    <Variance> 39
      1.0 1.0 1.0 ... 1.0
 <State> 3 <NumMixes> 1
  <Stream> 1
  <Mixture> 1 1.0
    <Mean> 39
      0.0 0.0 0.0 ... 0.0
    <Variance> 39
      1.0 1.0 1.0 ... 1.0
      .
      .
      .
 <TransP> N+2
  0.000e+0    1.000e+0    0.000e+0  ...  0.000e+0
  0.000e+0    6.000e-1    4.000e-1  ...  0.000e+0
  0.000e+0    0.000e+0    6.000e-1  ...  0.000e+0
  0.000e+0    0.000e+0    0.000e+0  ...  4.000e+1
  0.000e+0    0.000e+0    0.000e+0  ...  0.000e+0
<EndHMM>
```

Figure 4.1 Prototype files general structure

Word list

The word list is a simple file which contains the names of all the different elements to model, and is used as a reference for the training and testing processes. To clarify, a quick example of the list could be as follows:

```
leitmotiv
leitmotiv
audio
sil
```

The names for each element are arbitrary, but when declaring silence or short pause models they are called for "sil" and "sp" for convenience. This file was created with a simple text editor.

Dictionary

In simple terms, the dictionary is a file that contains a "translation" for each element in the word list. Assume that a particular element was detected during recognition. Even if the recognizer matched a HMM to an existing pattern on the testing data, it needs an indication on how to label this token when outputting the results. The role of the dictionary is to provide the name for an output symbol after recognition. The name of the output label for an element does not have to be identical to the name to match on the word list. This means, a HMM definition can have the name 'model' for the whole training process, but the desired labeling name in the dictionary. A typical dictionary has the following structure:

```
leitmotiv wanderer
audio music
sil silence
$model $output_label
```

For convenience, the used scheme has identical names for the model name and the output label, and due to the small number of words to categorize, it was created manually as well.

Network

The network file is a set of nodes and arcs, where the nodes represent the different elements or classes to be identified and the arcs represent the possible transitions between them. In this particular case, audio recordings are to be analyzed to detect the presence of leitmotivs in large music files. For instance, the nodes represent the different leitmotivs, background music, silence and short pause. For this task, the network was configured for connected event recognition. This file can be manually constructed or with HTK tools HBUILD and HPARSE.

Due to its simplicity of usage, HPARSE was the choice to create the network. It takes as input a grammar syntax file and returns a Standard Lattice Format (SLF) network. An example of a used syntax was:

```
$motivs = grubel | nibelungen | ring | vertrags | audio;

( sil <$motivs [sp]> sil )
```

Giving as a result the network depicted in Figure 4.2. By adding another word to the variable $motivs, the network can be easily modified by running the tool with the modifications.

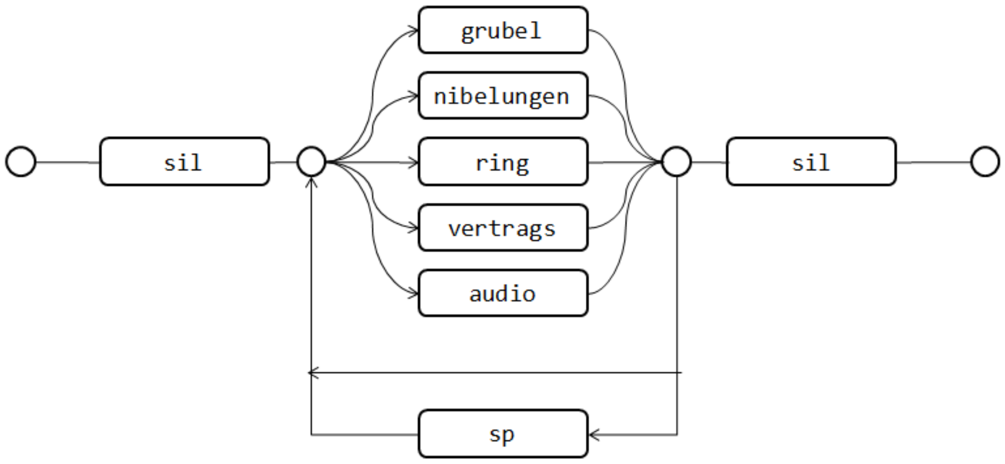


Figure 4.2 Generated recognition network

Configuration files

For training, testing and extracting features HTK can accept an optional configuration file for the different utilities included in the toolkit. A critical and necessary file in order to have a correct feature extraction is a file that will be used as an input parameter for the tool HCOPY of HTK. HCOPY can take waveform files and extract the features into MFCC vectors. The configuration files are simple text files with modifiable variables to operate the HTK environment. For this case, the configuration file for feature extraction can be observed in Figure 4.3

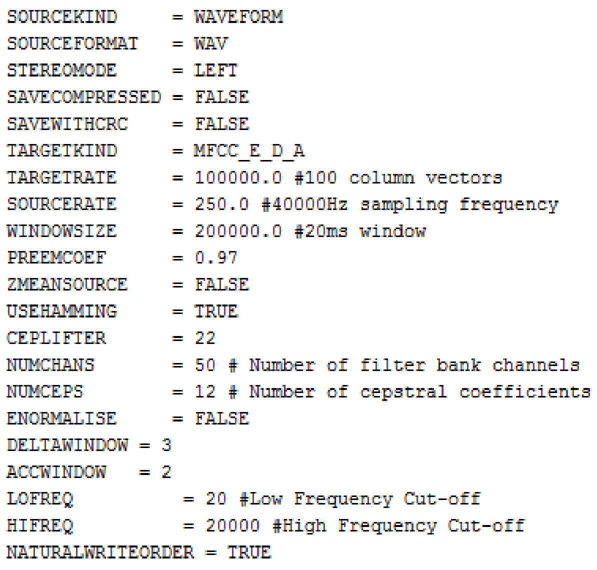

```
SOURCEKIND     = WAVEFORM
SOURCEFORMAT   = WAV
STEREOMODE     = LEFT
SAVECOMPRESSED = FALSE
SAVEWITHCRC    = FALSE
TARGETKIND     = MFCC_E_D_A
TARGETRATE     = 100000.0 #100 column vectors
SOURCERATE     = 250.0 #40000Hz sampling frequency
WINDOWSIZE     = 200000.0 #20ms window
PREEMCOEF      = 0.97
ZMEANSOURCE    = FALSE
USEHAMMING     = TRUE
CEPLIFTER      = 22
NUMCHANS       = 50 # Number of filter bank channels
NUMCEPS        = 12 # Number of cepstral coefficients
ENORMALISE     = FALSE
DELTAWINDOW = 3
ACCWINDOW   = 2
LOFREQ          = 20 #Low Frequency Cut-off
HIFREQ          = 20000 #High Frequency Cut-off
NATURALWRITEORDER = TRUE
```

Figure 4.3 Configuration file Variables

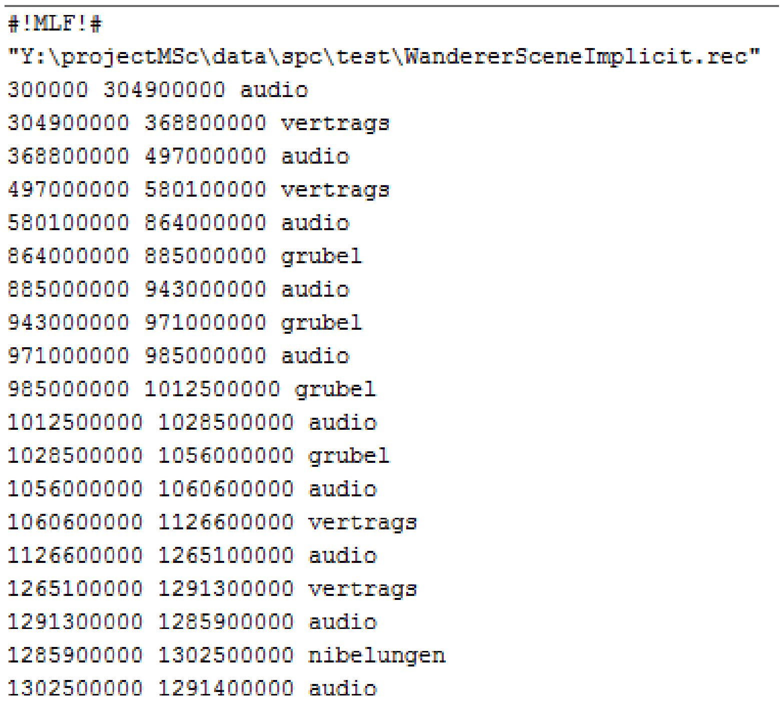
The selected parameters for feature extraction were targeted to signals with high frequency content, since music signals contain important information in both low (Hz) and high (kHz) range. Specification of the source audio file format is given by the parameters SOURCEKIND and SOURCEFORMAT, indicating the conversion of a waveform audio file. STEREMODE is set to read the left channel only. TARGETKIND specifies the type of feature to extract, which in this case is MFCC_E_D_A, setting delta to 3 and delta-delta to 2. It is observable that the SOURCERATE parameter indicates the sampling frequency of the waveform data, which has to correspond to the sampling rate of the actual recording. The parameters for MFCC conversion are set by TARGETRATE, which specifies the frame period output rate (window overlap), and WINDOWSIZE, which indicates the duration of each portion of audio to be processed at a time. These previous three parameters' units are expressed in hundreds on nanoseconds. For both training and testing, a small file containing the following parameters was created:

```
TARGETKIND = MFCC_E_D_A
DELTAWINDOW = 3
ACCWINDOW = 2
```

This is used to indicate the features type during training and testing. All of these files were manually created.

Label files

A label file is a description of the elements that appear in the training and testing data. It is a text file that contains a transcription of the elements that appear in the recordings, in order to have a reference point for identification and labeling of the different classes. The importance of the usage in training is to correctly categorize the acoustic speech frames and relate then to each relevant associated HMM. To create this file, the Matlab script xls2lab.m was developed to take information from an excel file containing the manually labeled appearances of leitmotivs in a recording, and convert it into standard HTK label format. The script is available in APPENDIX A. A portion of the generated label file can be appreciated in Figure 4.4. Notice that the start and end times of each element are included in the label. The original time slot in the excel file appears in a minute: seconds: hundredths of second (mm:ss: hs) format, in a floating point number.



```
#!MLF!#
"Y:\projectMSc\data\spc\test\WandererSceneImplicit.rec"
300000 304900000 audio
304900000 368800000 vertrags
368800000 497000000 audio
497000000 580100000 vertrags
580100000 864000000 audio
864000000 885000000 grubel
885000000 943000000 audio
943000000 971000000 grubel
971000000 985000000 audio
985000000 1012500000 grubel
1012500000 1028500000 audio
1028500000 1056000000 grubel
1056000000 1060600000 audio
1060600000 1126600000 vertrags
1126600000 1265100000 audio
1265100000 1291300000 vertrags
1291300000 1285900000 audio
1285900000 1302500000 nibelungen
1302500000 1291400000 audio
```

Figure 4.4 A fragment of the output label

Therefore, a small conversion procedure was implemented in the script with the following code:

```
%Convert value to 100ns
intpart = uint16(fix(startTime));
spart = startTime - double(intpart);
seconds = 60*intpart + 100*spart;
nstart = (uint64(seconds))*(10000000);
hs = str2double(timeparts(index1,3))/100;
nstart = nstart + (hs*10000000);
```

This converts the start and end times of each element, to subsequently be printed on the final file. Any other required label files can be manually typed, but taking into account that including the time slots is very important to a correct data encoding.

Path lists

Path lists are simple text files in .scp HTK format containing the exact file name, location in disk and file output destination for different HTK utilities. These could be manually created, but if the number of files begins to increase this becomes a tedious and susceptible to errors task. For this purpose, the Matlab script pathlistgenerator.m (available in APPENDIX B) was implemented. In script operation, a folder (containing the desired data to write on the list) is user selected, and the script generates three lists for it; a list containing the full name of the files in the folder, a list containing the full path of the files in the folder (used in training) and a list of the source path and destination path for each file (used in feature extraction).

4.1.2 Feature extraction and HMM initialization

The data sampling frequency was re-sampled from 44.1 KHz to 40KHz, for two reasons: The value of the period of 40KHz is an integer value (25 μs), and secondly, as 40KHz is enough to capture the high frequencies in the spectrum, storage space is significantly reduced when extracting the MFCC features with this sampling rate. Re-sampling was performed with a simple script in Matlab, audioFS.m (APPENDIX C). Extraction of the MFCC vectors for further system development is performed via the script feature_extract.m (APPENDIX D). A set of previously generated files are set into variables, and then the tool HCOPY is called via the system () function of Matlab. The main call is generated by the following code:

```
%Calling HCopy
    callHCopy = strcat(parHMM.dir_HTK,'HCopy');
    %Train files feature extraction
    command = [callHCopy,' ', ' -A -D -T 2 -C ', '"',copyConfig,'"',...
                                ' -S ', '"',copyTrainList,'"' ];
    %System command out
    [~,cmdout] = system(command);
```

The two inputs for HCOPY are the configuration file described previously, and the generated list containing the full path of source and destination. The same procedure is followed for the test files feature extraction, by changing the list of the –S parameter of the command. This will save a set of MFCC vector files in the specified location in the list. With the encoded data, it is possible to estimate the initial HMMs parameters, via the HINIT or HCOMPV tools. The difference between both is that HCOMPV creates a flat-start training based on the global values of the training data, while HINIT performs a more detailed parameter computation using Viterbi style estimation (Young et.al, 2006). Two versions of the script feature_extract.m are implemented, one using HCOMPV and the other using HINIT. Experiments with both tools will be described in further stages. In this stage, a seed HMM is created for each model with the use of one of the described tools. The call for HCOMPV is performed via the following piece of code:

```
command = [callHCompV,' ', ' -A -D -T 2 -C ', '"',trainConfig,'"',...

            ' -o ', 'hmmdef',...
            ' -f ', floorValue,...
            ' -m ',...
            ' -S ', '"',trainList,'"',...
            ' -M ','"',HMMout,'"',...
            ' ', '"',proto,'"' ];
```

In the case of calling HINIT, the code consists of:

```
command = [callHInit,' ', ' -A -D -T 1 -C ', '"',trainConfig,'"',...
            ' -o ', currentWord{i},...
            ' -i ', numIter,...
            ' -I ',label,...
            ' -l ',currentWord{i},...
            ' -m 1 ',...
            ' -v ',floorValue,...
            ' -S ', '"',trainList,'"',...
            ' -M ','"',HMMout,'"',...
            ' ', '"',proto,'"'
            ];
```

It is noticeable that both models accepts as input parameters the configuration file and prototype definition, as well as the list of the training files to perform the first estimation. By comparing both commands, it can be observed that HINIT makes the use of a label and specifies to which exact unit is going to create a HMM. Experiments will be performed with both styles of training and discussed further.

4.1.3 HMM parameter re-estimation

The resulting HMMs from the previous stage are ready to be tested, but parameter re-estimation is usually required to increase the robustness and accuracy of the recognizer. HTK provides two different tools to perform Baum-Welch re-estimation: HREST and HEREST. The main aspect to consider is that HREST performs isolated-unit training, while HEREST works on complete sets of HMMs following an embedded-unit training (Young et.al, 2006). For this purpose, two versions of the Matlab script leitmotivs_train.m (APPENDIX E) were developed to perform training using both schemes. Figure 4.5 shows the two main training schemes to be followed by the scripts.

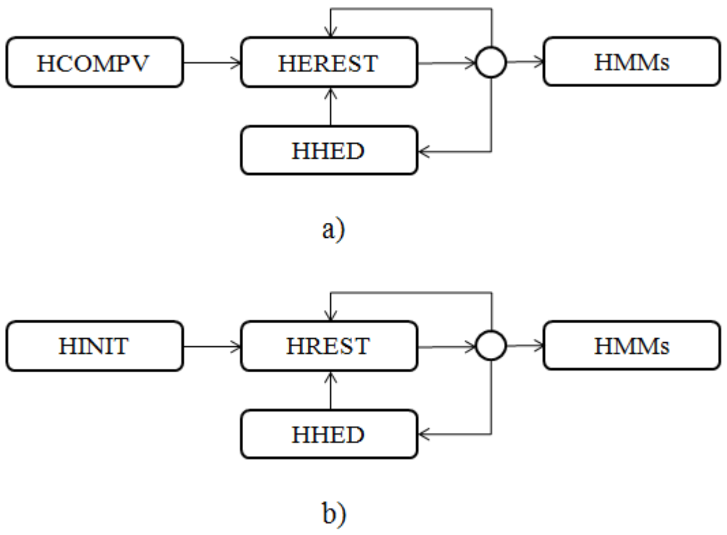


Figure 4.5 a) Model working on whole set of HMMs. b) Model working in isolated-unit HMM

In both cases, HHED is used to raise the number of Gaussian mixture components per state for each of the re-estimated models. In the case of the HCOMPV procedure, a single file containing the whole set of HMMs is processed with HEREST, while in the HINIT procedure each unit HMM is processed individually with HREST. In both cases, the training consists of performing several re-estimations iterations, then increasing the number of mixture components with HHED, and then re-estimating with several iterations. This loop is followed until the desired characteristics of the HMMs are reached. The command call for HEREST is executed by:

```
command = [callHERest,' ', ' -A -D -T 8 -D -C ', '"',trainConfig,'"',...
            ' -I ', '"',labelNSP,'"',...
            ' -t ',threshold,...
            ' -S ', '"',trainList,'"',...
            ' -H ', '"',bmacros,'"',...
            ' -H ', '"',bmodels,'"',...
            ' -M ', '"',nextHMM,'"',...
            ' -m ', '"',mindata,'"',' ',...
            ' -w ', '"','1','"',' ',...
            '"',wordListNSP,'"',...
            ];
```

In this case, HEREST takes as input arguments an existing set of HMM with the –H command, and its associated macro file with common values for a variance vector, the label file via the –I option, a pruning threshold value via –t, the list of training files via –S, the minimum number of training samples per model via –m, and the word list of the models to train. In the case of HREST, the command is called by:

```
command = [callHRest,' ', ' -A -D -T 1 -D -C ', '"',trainConfig,'"',...
            ' -l ', '"',currentWord{i},'"',...
            ' -t ',...
            ' -I ', '"',labelNSP,'"',...
```

```
               ' -S ', '"',trainList,'"',...
               ' -H ', '"',bmodels,'"',...
               ' -M ', '"',nextHMM,'"',...
               ' -m ', '"',mindata,'"',' ',...
               ' -w ', '"','1','"',' ',...
               bmodels,...
               ];
```

In which the main difference is the addition of the –l parameter, which specifies the name of the model to be trained. Another aspect to consider is that one single call of HEREST performs the training for the whole models, while in the HREST procedure a loop must be performed to train each model (since each is individually trained), resulting in a longer training process. In both cases, HHED is called by the general structure of:

```
command = [callHHEd,' ', ' -A -D -T 2',...
               ' -H ', '"',bmacros,'"',...
               ' -H ', '"',bmodels,'"',...
               ' -M ', '"',nextHMM,'"',' ',...
               '"',mixN,'"',' ',...
               '"',wordListSP,'"'
               ];
```

In which one or more HMM files are loaded via the –H option. The variable mixN contains the configuration file indicating how many mixture components per state for each model the tool is going to perform increasing of. Also, HHED creates a tied model between the silence model and short pause, meaning that the short pause model is modeled as the central state of the silence model. After completion, a set of HMMs for each unit will be obtained.

4.2 Data testing

4.2.1 Performing a HMM test

The detection is performed by calling the HTK tool HVITE. This tool performs classification by running the Viterbi algorithm, taking the produced HMMs as a reference for token labeling. Two versions of the script leitmotiv_test.m (APPENDIX F) were developed to perform this command. The general structure of calling HVITE is depicted on the following code:

```
command = [callHVite,' ', ' -A -D -T 1 -C ', '"',testConfig,'"',...
               ' -H ', '"',macros,'"',...
               ' -H ', '"',models,'"',...
               ' -S ', '"',testList,'"',...
               ' -w ', '"',network,'"',...
               ' -o SW -m -i ','"',testRes,'"',...
               ' -p ', penalty,...
               ' -s ', scaling,' ',...
               '"',dictionary,'"',' ',...
               '"',wordList,'"'
               ];
```

In which the –H options allows to load multiple HMM files, -w takes the word network for allowed transitions between units, -o controls the output file contents, -i contains the address for storing the test results, -p is the word insertion penalty (WIP) value, -s is the scaling factor value, and finally the dictionary and word list which allows to perform transcriptions.. The resulting file is a standard HTK label file in the style described before in file configuration, which contains each recognized unit and its associated time frame. It is to point that by tuning the WIP value, the performance of the system can be increased without the need of redefining a new set of HMMs.

4.2.2 Test results evaluation

The generated test result file is not in a convenient format to analyze; therefore a tool is required to perform the evaluation measures to assess the system's performance. HTK provides the tool HRESULT to compare two standard label files, in which the usual input would be a reference label with the real frames, and the result from HVITE. For the detection system assessment, HRESULT does not fulfill the requirements since it checks and compare a sequence of labels, but does not analyze temporal information. Therefore, the Matlab script hmmResult.m (APPENDIX G) was developed to provide more accurate evaluations. The script reads two standard HTK label files, tagging one as the reference label and the other one as the recognition result label. The aim is to count the number of hits, substitutions, insertions and deletions, and subsequently calculate percentages of precision, recall and the F score.

Definitions on what is considered a hit, substitution, deletion and insertion are required. Figure 4.6 shows a diagram illustrating under what conditions an identified label is considered a hit. If the start time of a detected frame falls between the hit-range, it is then considered a hit. The hit range covers any point between the reference leitmotiv start time and half its duration, and a shift tolerance of detection of 0.5s. The remaining scores are derived from this initial definition, in which a substitution is considered a frame that falls in the hit-range but has a different label than the reference, an insertion is considered a frame that appears on the results file that is not part of the reference file, and a deletion is considered a frame that exists in the reference file but never appeared on the results.

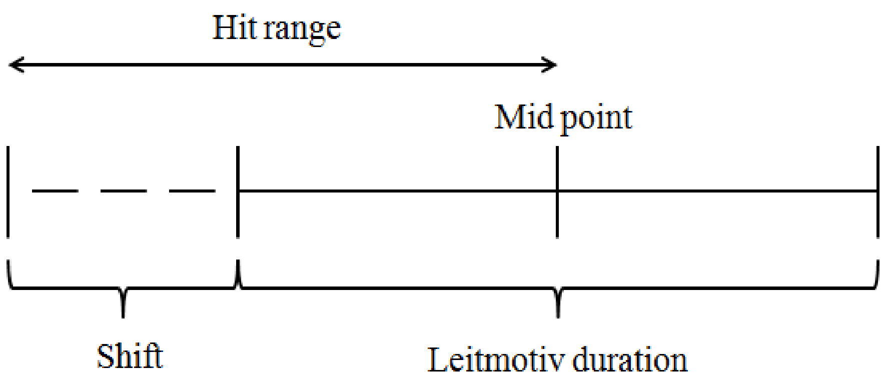


Figure 4.6 Hit consideration scheme

After computing the scores, the script plots a graph containing both reference and test result classes in a chronological appearance. An example of this graph is shown in Figure 4.7. In the plot, Detected indicates what the recognizer identified, and Original corresponds to the actual appearance of the leitmotiv in the recording.

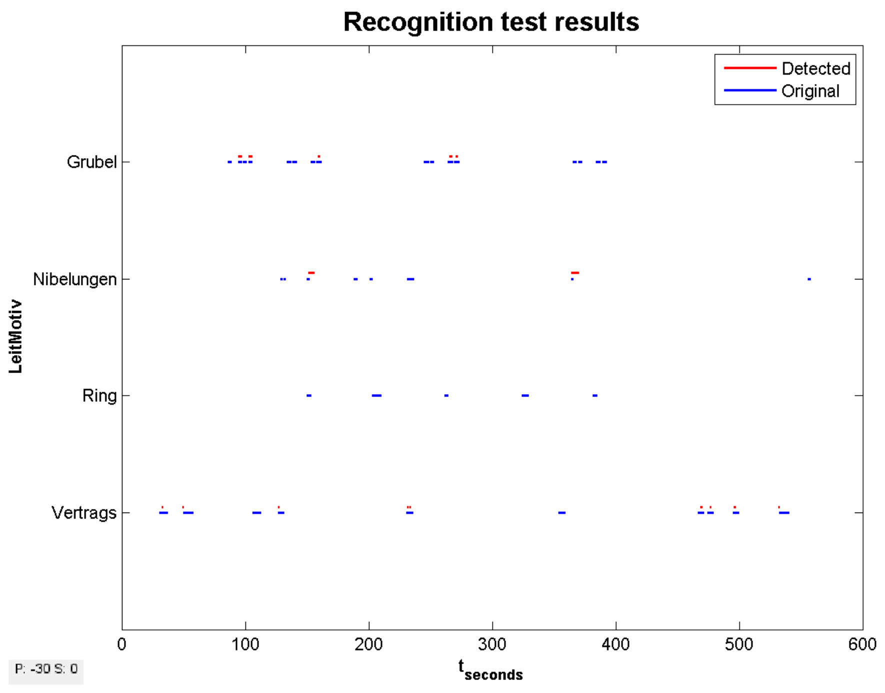


Figure 4.7 Recognition test results graph example

Additionally, the script writes the computed scores in a text file with its corresponding test parameters. Each test result is appended to the same file, to form a collective database, which will allow extracting information for further processing and results analysis.

5. RESULTS AND ANALYSIS

The obtained results are derived from two major experiments: tests with the embedded-unit training style system and tests the isolated-unit training style system. For each experiment, the HMMs were designed with different number of states, Gaussian mixture components per state for each model, as well as tuning of the word insertion penalty value. Each system was trained with an excerpt of an opera of Wagner, Wanderer Scene Implicit, and the tests were performed on the same file. In this file, 46 appearances distributed between 4 different leitmotivs, Grubel, Nibelungen, Ring and Vertrags is the target. Notice that for each experiment, any frame that is not part of a leitmotiv was labeled as 'audio'.

5.1 Embedded-unit training results

As explained in the training procedure in section 4.1.3, the embedded unit scheme performed training on whole sets of HMMs. The initial training parameters were a 10 state HMM for each leitmotiv, a 3 state audio and silence model and a single Gaussian mixture component for all the models. Such a simple setup gave poor results as only a 3% precision (P) and a 13.95% recall (R) were achieved. Increasing the number of mixtures for the audio model only from 1 to 2 and then from 2 to 4, conserving the same number of states resulted in a precision of 4.18% and a recall of 14.47%. Thus not a big difference, an increase in performance can be observed. A comparative series of tests were performed in a fixed state HMM, varying the number of mixtures for both leitmotiv and audio models. As a practical approach, the F1-score is used as a reference to compare both configurations results, displayed in Figure 5.1 for visualization. It can be clearly seen that isolated management of the number of mixture components improves the functionality of the detector; hence the effect of modifying the number of states is to be explored.

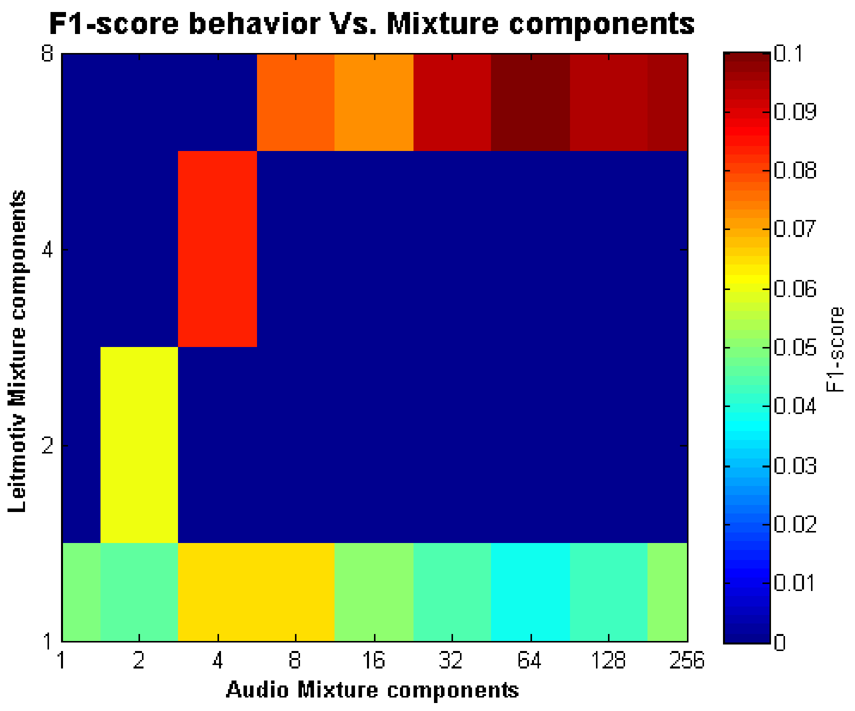


Figure 5.1 The effect in the system's performance of increasing the number of Gaussian mixture components

Since finding better parameters is highly dependent on an empirical method, different parameter sets of HMMs were configured in order to find a best performing recognition. For each set (i.e. 10 states for Leitmotivs and 3 states for audio models) several tests with different number of mixtures, increasing in powers of two were performed. The overall results of can be observed in Figure 5.2, in which the legend for LM indicates the number of state and maximum number of mixtures, where maximum number of mixtures refers to the highest number of mixture components that the training of the system performed. The tendency of the pattern shows that in a general scenario, increasing the number of states "shifts" down the desired score, as setups with 6 and 10 states for the audio model and 20+ states for the leitmotivs performed less efficiently. By taking the best result setup and observing the number classification numbers (hits, substitutions, insertions, deletions); it was noticeable that increasing the number of mixture components for the audio model reduces significantly the number of insertion errors. On the other hand, increasing the number of mixture components for the Leitmotivs showed that this modification improves the balancing of insertion and deletion errors, but with a side effect of reducing the number of hits or correct classifications, as a 4 Gaussian mixture component per state model gave values of H = 11, S = 25, I = 42 and D = 149, while a 64 Gaussian mixture component gave H = 8, S = 16, I = 54 and D = 59, respectively.

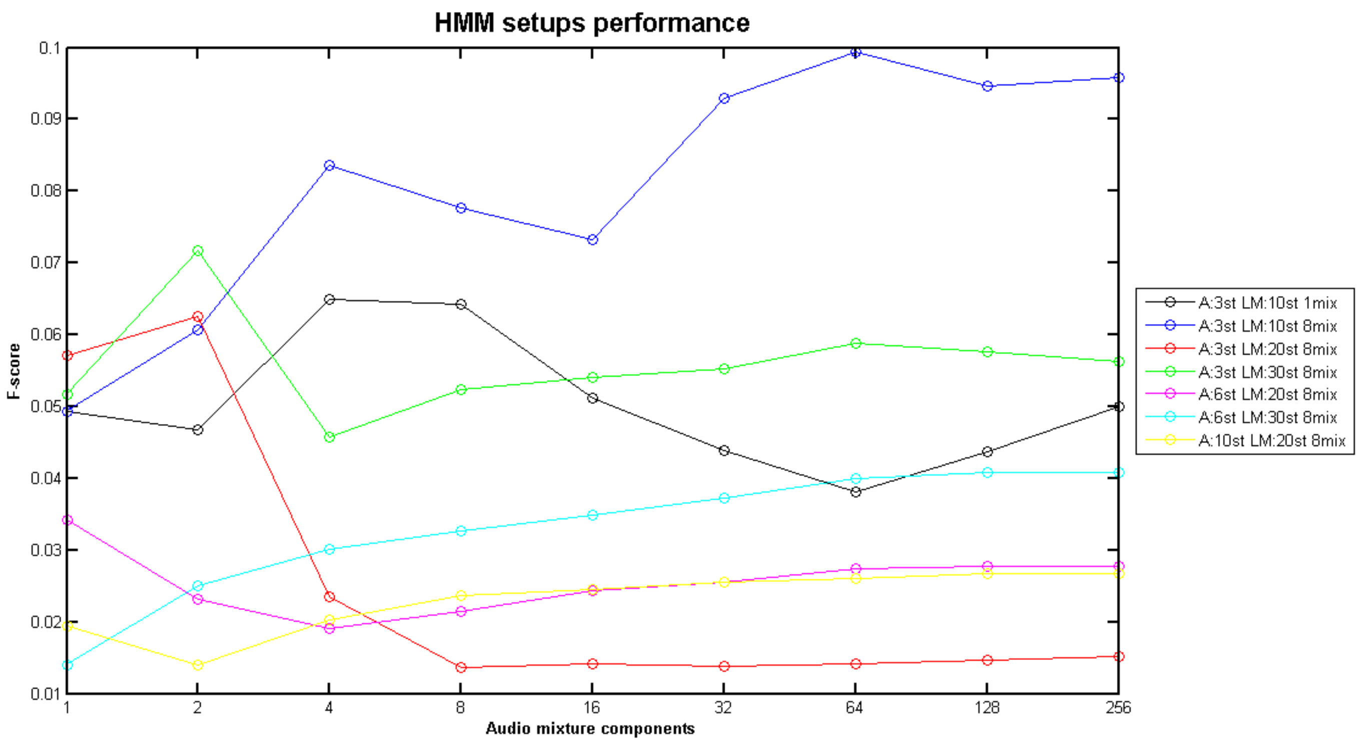


Figure 5.2 System performance with different embedded unit HMM training approach

However, increasing the audio mixture components to a very high number (such as 256) does not guarantee an optimum performance, as in not all of the cases the F1-score value peak is found in this configuration. Although the reduction of errors can be managed by changing the HMMs, the overall performance of the embedded-unit training systems are not accurate enough for this application, as the maximum achieved performance levels no do not surpass an F1-score of 0.15, even with word insertion penalty tuning. These observations apply to the embedded-unit training approach, due to this reason, the isolated-unit training approach was explored.

5.2 Isolated-unit training results

By segmenting each individual model and training the HMMs separately, parameter estimation is based only in the frames that are labeled with the desired model to train in the reference file. The newly trained system, taken as a starting point from analyzing its exponential like increasing behavior in Figure 5.2, had an initial setup of 6 state audio model and 30 state leitmotivs, with a single Gaussian mixture component for all models. A huge improvement was achieved as the system had a precision of 13.55% and a recall of 45.31%, giving a balanced F1-score of 0.2086. Considering the effect of increasing the Gaussian mixture components per state per model, a similar procedure to the one in section 5.1 was followed. Table 5.1 shows the results for the initial HMM scheme, with an increment in powers of two for the mixture components.

Table 5.1 A 6 state audio and 30 state leitmotiv HMM test results

H	S	I	D	Prec	Rec	F	p	s	AS	AMIX	LS	LMIX
28	3	160	34	14.66%	43.08%	0.2188	-10	0	6	1	30	1
29	3	182	32	13.55%	45.31%	0.2086	0	0	6	1	30	1
29	3	201	32	12.45%	45.31%	0.1953	10	0	6	1	30	1
37	1	121	26	23.27%	57.81%	0.3318	-10	0	6	2	30	2
37	1	142	26	20.56%	57.81%	0.3033	0	0	6	2	30	2
38	2	168	25	18.27%	58.46%	0.2784	10	0	6	2	30	2
39	3	82	22	31.45%	60.94%	0.4149	-10	0	6	4	30	4
39	3	90	22	29.55%	60.94%	0.3980	0	0	6	4	30	4
39	3	111	21	25.49%	61.90%	0.3611	10	0	6	4	30	4
40	2	59	20	39.60%	64.52%	0.4908	-10	0	6	8	30	8
40	2	74	20	34.48%	64.52%	0.4494	0	0	6	8	30	8
41	3	94	18	29.71%	66.13%	0.4100	10	0	6	8	30	8
40	2	43	20	47.06%	64.52%	0.5442	-10	0	6	16	30	8
40	2	51	20	43.01%	64.52%	0.5161	0	0	6	16	30	8
41	3	59	18	39.81%	66.13%	0.4970	10	0	6	16	30	8
40	2	30	20	55.56%	64.52%	0.5970	-10	0	6	32	30	8
40	2	31	20	54.79%	64.52%	0.5926	0	0	6	32	30	8
41	3	40	18	48.81%	66.13%	0.5616	10	0	6	32	30	8
40	2	15	20	70.18%	64.52%	0.6723	-10	0	6	64	30	8
40	2	17	20	67.80%	64.52%	0.6612	0	0	6	64	30	8
40	2	21	20	63.49%	64.52%	0.6400	10	0	6	64	30	8
40	2	5	20	85.11%	64.52%	0.7339	-10	0	6	128	30	8
40	2	6	20	83.33%	64.52%	0.7273	0	0	6	128	30	8
40	2	6	20	83.33%	64.52%	0.7273	10	0	6	128	30	8
40	2	5	20	85.11%	64.52%	0.7339	0	0	6	256	30	8
40	2	5	20	85.11%	64.52%	0.7339	10	0	6	256	30	8
40	2	5	20	85.11%	64.52%	0.7339	-10	0	6	256	30	8

In these results, a tuning of the word insertion penalty can be observed in column p, as its value oscillates between -10, 0 and 10. The effect of this is a slight fine tuning of a specific model. As the number of mixture components for the audio model increase, the numbers of hits increase as well, while the insertion errors drop to almost non-existent. The deletion errors decrease accordingly, but by analyzing the results it can be observed that this number depends mostly on increasing the number of mixture components for the leitmotivs models. Notice that changing from 128 to 256 mixtures in the audio model does not make any relevant changes; therefore the next tests were only performed up to 128 mixtures per state.

A different set of HMM parameters for the isolated-unit training was configured and different tests were performed accordingly. Figure 5.3 summarizes these results. High levels of performance were obtained as the highest number of hits was 42 out of the 46 targeted units. From the setup that marked 42 hits, the best performance reached a precision of 85.71% and a recall of 66.66%, with a balanced F1-score of 0.75. This means that the number of insertion errors was lower than the insertions errors, which for this application is preferable.

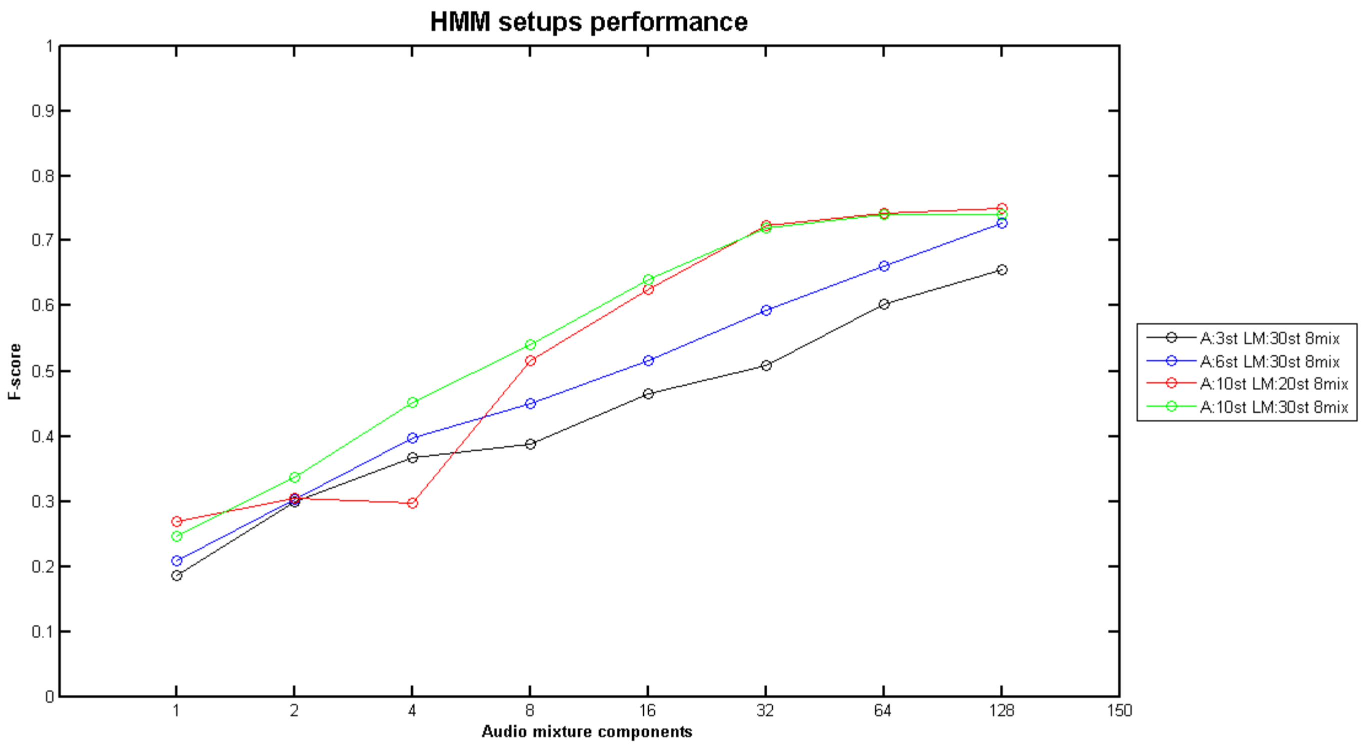


Figure 5.3 Isolated unit training approach performance results

It is sensible to observe the setup of the highest achieved F1-score, as the parameters need to be identified. From Figure 5.3 can be observed the highest point is reached in the red line at 64 and 128 mixtures in the horizontal axis. No changes were present when transition from 64 to 128 mixtures, therefore the best performance for these experiments was the setup of the 10 state audio, 20 state leitmotiv models, with 64 and 8 mixture components per state respectively.

5.3 Training style methods comparison

In order to make a formal judgment of both methods performance, a comparison between the best results of each method is performed. The best setup for embedded-unit was a 3state audio, 10 state leitmotiv models with 32 and 8 mixture components respectively, while the isolated-unit best setup has been previously mentioned. A comparative table (Table 5.2) shows the statistical values of each model, and percentage difference is specified between both models is calculated.

Table 5.2 Embedded-unit Vs. Isolated-unit best results models

Model	**H**	**S**	**I**	**D**	**Precision**	**Recall**	**F1-score**	**p**	**s**
Embedded-unit	8	13	55	61	9.76%	10.53%	0.1013	-10	0
Isolated-unit	42	3	4	18	85.71%	66.67%	0.7500	10	0
Change	425.00%	-76.92%	-92.73%	-70.49%	778.57%	533.33%	640.63%		

It is noticeable that the overall F1-score improved in 640.63% from the embedded-unit model to the isolated-unit model. The scores do not specify how 'accurate' the hits were, since the length of each identified frame is not depicted. Figures 5.4 and 5.5 serve for this purpose. Notice that in Figure 5.4 only 2 out of the 8 detected leitmotivs are within a reasonable length, which correspond to the Grubel leitmotiv. 2 classifications from the Nibelungen leitmotiv exceed the frame length, while the remaining 4 are extremely short. This is an important point to take into account, since the recognizer specifies between which times in the recording a leitmotiv appears. In contrast, Figure 5.5 shows a more healthy result, as the majority of the identified frames are significantly close to the exact duration of the reference frame. This indicates that not only the number of correct identified classes is better for the isolated-unit model, but the accuracy with which it labels the duration of a hit is superior as well. An analysis of these results suggests that the utterances are categorized or separated by a soft decision making of the tool HEREST, which in an audio recording analysis that has polyphonic tracks within, is not a convenient method for HMM parameter re-estimation. By having a hard decision, time influenced labeled examples of the desired classes to identify, HREST performs a more accurate alignment, and it can be observed by the performance improvement from one model to the other.

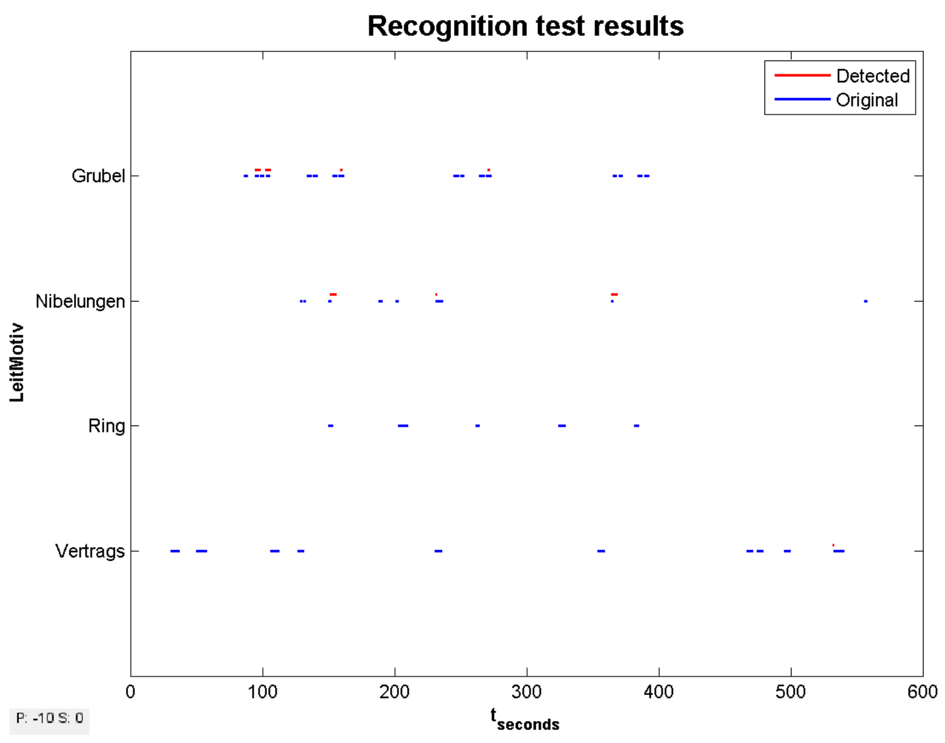


Figure 5.4 Embedded unit model time displayed results

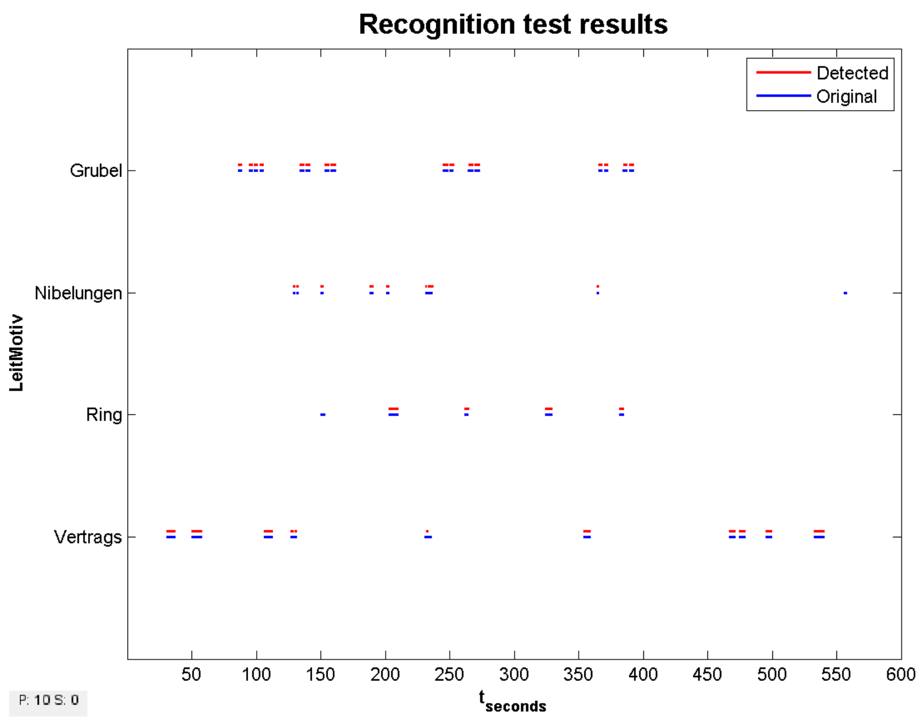


Figure 5.5 Isolated unit model time displayed results

6. CONCLUSIONS

In this project, the objective was to develop an automatic tool to detect specific leitmotivs of composer Richard Wagner in audio recordings. Different audio feature representations were explored, narrowing the decision to two main features, Chroma and MFCC, using MFCC as an option since they were proved to be suitable for music applications (Logan, 2000), and could be obtained in a practical way with the HTK toolkit (developed at the Cambridge University Engineering Department). The analysis of widely used methods in Music Information Retrieval systems was performed, narrowing the selection to HMM, GMM and SVM, taking a combination of HMM + GMM as a modeling technique, due to its relatively simple modeling via the use of the HTK toolkit. Two major approaches were taken in the design of the HMMs: isolated unit and embedded unit training. Both paths used an opera excerpt from Wagner as a set of training and testing data. For each operation of training the system, several configurations for the HMM parameters were set to identify the initial response of the selected features to the identification process.

Results with the embedded-unit training approach weren't satisfactory enough as the overall achieved F-score did not surpass a score of 0.15. Initially it appeared that the modification of the number of HMM states, and the number of GMM components did not made any significant improvement, with these results suggesting that this style of classification was not appropriate for this experiment.

Afterwards, experiments with the isolated-unit style of training were performed, with a significant improvement comparing the initial first training scheme results. Testing with different configurations for this scheme as well, allowed to observe that the majority of the targeted units were successfully detected, in both duration of frame and time of appearance in the recording. The best system with this setup achieved an F-score of 0.75, which represents an increase of 640.63% of the best achieved F-score with the embedded unit training.

The new results suggests that the selected approach appear to be sufficiently effective, as the insertion errors were basically reduced to almost non-existent, which for this application is far more important than minimizing the deletion errors. It is to mention that the ideal scenario would be to detect 100% of the leitmotivs in the recording, but there are complications due to the nature of the recording, as is a polyphonic audio signal, which means multiple instruments and voices, and even two leitmotivs, are occurring simultaneously.

The overall findings indicate that using the proposed method can be in fact an effective way of developing the automatic recognition tool, and could be improved with by performing more experiments with more extensive data corpora, as well as exploring more deeply the different configurations of the HMM parameters. An approach of training the HMMs individually proved to be more effective than a global approach using the overall mean and variance of the training data.

Further research could explore the possibility of implementing the tool development with the use of Chroma features, and combinations of Chromas and MFCCs, as well as the use of Support Vector Machines, Neural Networks or combinations of them.

All the minimum aims of the project were achieved, which were to develop the tool with at least one modeling approach and one feature representation, and testing in a small corpus of recordings.

Finally, the acquired knowledge about Music Information Retrieval, audio signal processing, mathematical modeling and programming techniques during the project formed part of a skill development process that allowed achieving successful results, making it an invaluable experience that will bring benefits in future research projects.

REFERENCES

[1] Albrecht, H., & Frieler, K. (2014). The perception and recognition of Wagnerian leitmotifs in multimodal conditions. In *International Conference of Students of Systematic Musicology*.

[2] Bartsch, M., & Wakefield, G. H. (2001). To catch a chorus: Using chroma-based representations for audio thumbnailing. In *Applications of Signal Processing to Audio and Acoustics, 2001 IEEE Workshop on the* (pp. 15-18). IEEE.

[3] Blunsom, P. (2004). Hidden markov models. *Lecture notes*, *15*, 18-19.

[4] Cabral, G., Briot, J. P., & Pachet, F. (2005). Impact of distance in pitch class profile computation. In *Proceedings of the Brazilian Symposium on Computer Music* (pp. 319-324).

[5] Casey, M., Veltkamp, R., Goto, M., Leman, M., Rhodes, C., & Slaney, M. (2008). Content-based music information retrieval: Current directions and future challenges. *Proceedings of the IEEE*, *96*(4), 668-696.

[6] Dannenberg, R. B., & Hu, N. (2003). Pattern discovery techniques for music audio. *Journal of New Music Research*, *32*(2), 153-163.

[7] Doria, D. (2010). Expectation Maximization of Gaussian Mixture Models in VTK. VTK Journal (ISSN 2328-3459) . Available from http://hdl.handle.net/10380/3218

[8] Dreyfus, L., & Rindfleisch, C. (2014, September). Using Digital Libraries in the Research of the Reception and Interpretation of Richard Wagner's Leitmotifs. In *Proceedings of the 1st International Workshop on Digital Libraries for Musicology* (pp. 1-3). ACM.

[9] Ellis, D. P. (2007). Classifying music audio with timbral and chroma features. In *ISMIR 2007: Proceedings of the 8th International Conference on Music Information Retrieval: September 23-27, 2007, Vienna, Austria* (pp. 339-340). Austrian Computer Society.

[10] Foote, J. (1999). An overview of audio information retrieval. *Multimedia systems*, *7*(1), 2-10.

[11] Guo, G., & Li, S. Z. (2003). Content-based audio classification and retrieval by support vector machines. *Neural Networks, IEEE Transactions on*, *14*(1), 209-215.

[12] Hasan, M. R., Jamil, M., & Rahman, M. G. R. M. S. (2004). Speaker identification using Mel frequency cepstral coefficients. *variations*, *1*, 4.

[13] Ishwar, V., Dutta, S., Bellur, A., & Murthy, H. A. (2013, November). Motif Spotting in an Alapana in Carnatic Music. In *ISMIR* (pp. 499-504).

[14] Logan, B. (2000, October). Mel Frequency Cepstral Coefficients for Music Modeling. In *ISMIR*.

[15] Logan, B., & Chu, S. (2000). Music summarization using key phrases. In *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on* (Vol. 2, pp. II749-II752). IEEE.

[16] Makhoul, J., Kubala, F., Schwartz, R., & Weischedel, R. (1999, February). Performance measures for information extraction. In *Proceedings of DARPA broadcast news workshop* (pp. 249-252).

[17] Mandel, M. I., & Ellis, D. P. (2005). Song-level features and support vector machines for music classification. In *ISMIR 2005: 6th International Conference on Music Information Retrieval: Proceedings: Variation 2: Queen Mary, University of London & Goldsmiths College, University of London, 11-15 September, 2005* (pp. 594-599). Queen Mary, University of London.

[18] Marolt, M. (2004, October). Gaussian Mixture Models For Extraction Of Melodic Lines From Audio Recordings. In *ISMIR*.

[19] McCowan, I. A., Moore, D., Dines, J., Gatica-Perez, D., Flynn, M., Wellner, P., & Bourlard, H. (2004). *On the use of information retrieval measures for speech recognition evaluation* (No. EPFL-REPORT-83156). IDIAP.

[20] McShan, J. (1997). Wagner and the Leitmotiv. University of Michigan. Retrieved from http://www.umich.edu/~umfandsf/symbolismproject/symbolism.html/Teutonic_Mythology/wagleit.html

[21] Milner, B., & Shao, X. (2002, September). Speech reconstruction from mel-frequency cepstral coefficients using a source-filter model. In *INTERSPEECH*.

[22] Molau, S., Pitz, M., Schluter, R., & Ney, H. (2001). Computing mel-frequency cepstral coefficients on the power spectrum. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on* (Vol. 1, pp. 73-76). IEEE.

[23] Powers, D. M. (2011). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation.

[24] Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE, 77*(2), 257-286.

[25] Reynolds, D. (2009). Gaussian mixture models. In *Encyclopedia of Biometrics* (pp. 659-663). Springer US.

[26] Ross, J. C., Vinutha, T. P., & Rao, P. (2012, October). Detecting Melodic Motifs from Audio for Hindustani Classical Music. In *ISMIR* (pp. 193-198).

[27] Sheh, A., & Ellis, D. P. (2003). Chord segmentation and recognition using EM-trained hidden Markov models. *ISMIR 2003*, 185-191.

[28] Weisstein, E. W. (2002). Discrete fourier transform. Extracted via web from the website http://mathworld.wolfram.com/DiscreteFourierTransform.html

[29] Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Liu, X., Moore, G., ... & Woodland, P. (2006). *The HTK book* (Vol. 2). Cambridge: Entropic Cambridge Research Laboratory.

APPENDIX A

```
%Read excel file and convert to standard HTK label file
clear
clc;
%% Open file for reading
disp('XLS to standard HTK label conversion');
disp('Select file');
[xlsName,xlsPath] = uigetfile({'*.xls;*.xlsx',...
    'Excel Files (*.xls,*.xlsx)'},...
    'Select spreadsheet file');
xlsFile = fullfile(xlsPath,xlsName);
%% File configuration
disp('Retrieving data...');
[~,data,~] = xlsread(xlsFile,1);
disp('Data retrieved');
temptimes = char(data(2:47,6:7));
names = data(2:47,1);
%Y = char(temptimes(1,1));
%Time extraction
delimiter = size(temptimes);
times = zeros(46,2);
timeparts = cell(delimiter(1),3);
for i=1:delimiter(1)
    timeparts(i,1:3) = strsplit(temptimes(i,:),{':','.'});
end

for k=1:(delimiter(1)/2)
    tempval = str2double(timeparts(k,2));
    cmpres = str2double(timeparts(k,1)) + (tempval/100);
    tempval = str2double(timeparts(k+46,2));
    cmpres2 = str2double(timeparts(k+46,1)) + (tempval/100);
    times(k,1) = cmpres;
    times(k,2) = cmpres2;
end
%% HTK Label init
mlfName = 'labelout.mlf';
masterlabel = fopen(fullfile(xlsPath,mlfName), 'wt');
[labName,path] = uigetfile('*.*','Select file to name internal label');
fprintf(masterlabel,'#!MLF!#\n');
[~,name,~] = fileparts(labName);
internalLabName = ['"',path, name, '.rec', '"'];
fprintf(masterlabel,'%s\n',internalLabName);
disp('Label file created');
prompt = 'Label with sp? Y/N: ';
question = lower(input(prompt,'s'));
printsp = strcmp(question,'y');
disp('Analyzing file...');
%% Conversion algorithm
%Initialise decision variables
decisionindex = size(times);
decisionMatrix = zeros(decisionindex(1),decisionindex(2));
refvalue = times(1,1);
```

```
temp = times(1,1);
condition = 0;
lock = 0;
count = 0;
%Initial check for NaN cells
for k=1:decisionindex(1)
    for l=1:decisionindex(2)
        temp = times(k,l);
        type = isnan(temp); %Check for NaN cells
        if type ~= 0
            index1 = k;
            index2 = l;
            decisionMatrix(index1,index2) = 1; %Start time
            decisionMatrix(index1,(index2+1)) = 1; %End time
        end
    end
end
%Restart index values
index1 = 1;
index2 = 1;
%Motiv detection
while(condition == 0)
    for a=1:decisionindex(1)
        for b=1:decisionindex(2);
            flag = decisionMatrix(a,b);
            if flag == 0
                refvalue = times(a,b);
                break;
            end
        end
        if flag == 0
            break;
        end
    end

    for i=1:decisionindex(1)
        for j=1:1
            lock = decisionMatrix(i,j);
            if lock == 0
                temp = times(i,j);
                if temp <= refvalue
                    refvalue = temp;
                    index1 = i;
                    index2 = j;
                end
            end
        end
    end
    %Min value check end
    decisionMatrix(index1,index2) = 1; %Start time
    decisionMatrix(index1,(index2+1)) = 1; %End time
    %Write to label file
    startTime = times(index1,index2);
    endTime = times(index1,(index2+1));
    %Convert value to 100ns
    intpart = uint16(fix(startTime));
    spart = startTime - double(intpart);
    seconds = 60*intpart + 100*spart;
    nstart = (uint64(seconds))*(10000000);
    hs = str2double(timeparts(index1,3))/100;
    nstart = nstart + (hs*10000000);
    intpart = uint16(fix(endTime));
    spart = endTime - double(intpart);
    seconds = 60*intpart + 100*spart;
    nend = (uint64(seconds))*(10000000);
    hs = str2double(timeparts(index1+46,3))/100;
    nend = nend + (hs*10000000);
    motiv = lower(names{index1,1});
    astart = 300000;
    count = count+1;
    %Print time and motiv name
    %fprintf(masterlabel,'%f %f %s\n',startTime, endTime, motiv);
    if count == 1
        fprintf(masterlabel,'%d %d audio\n',astart, nstart);
    else
        fprintf(masterlabel,'%d %d audio\n',prevend, nstart);
    end
    if printsp == 1
        fprintf(masterlabel,'sp\n');
        disp('sp');
    end
    fprintf(masterlabel,'%d %d %s\n',nstart, nend, motiv);
    disp(motiv);
    if printsp == 1
        fprintf(masterlabel,'sp\n');
        disp('sp');
    end
    %Store previous end
    prevend = nend;
    %Check if all values have been met
    condition = all(decisionMatrix(:) > 0);
end
%% Close file
disp('Label file conversion completed');
fprintf(masterlabel,'.');
fclose(masterlabel);
```

APPENDIX B

```
%Script for generating HTK lists
%Search path
audiopath = uigetdir;
addpath(audiopath);
%Array containing name of all the files in the folder
listing = dir(audiopath);
fid = fopen( 'listTrainHCopy_Siegfried.scp', 'wt' );
fid2 = fopen( 'listTrainFullPath_Siegfried.scp', 'wt');
fid3 = fopen('listTrain_Siegfried.scp', 'wt');
%Iterate from here
for k=3:length(listing)
    %Goes to Lists
    audiofile = listing(k).name;
    %Get full path
    fullpath = [audiopath '\' audiofile];
    A = char(1,100);
    %Detect and insert double \\ for file writing
    w = 1;
    for i=1:length(fullpath)
        currentchar = fullpath(i);
        A(w) = fullpath(i);
        w = w + 1;
        if currentchar == '\'
            A(w) = '\';
            w = w + 1;
        end
    end
    %Create HCopy new path
    B = char(1,100);
    for j=1:length(A)+1
        currentchar = A(j);
        B(j) = A(j);
        if j > 3
            check = strcat(A(j-2),A(j-1),A(j));
            value = strcmp(check,'wav');
            if value == 1;
                B(j-2) = 's';
                B(j-1) = 'p';
                B(j) = 'c';
            end
        end
        if currentchar == '.'
            B(j+1) = 'm';
            B(j+2) = 'f';
            B(j+3) = 'c';
            B(j+4) = 'c';
            break;
        end
    end
    %Write to Hcopy file
    fprintf(fid,A);
    fprintf(fid,' ');
    fprintf(fid,B);
    fprintf(fid,'\n');
    %Write to Full path file
    fprintf(fid2,B);
    fprintf(fid2,'\n');
    %Write to simple list
    fprintf(fid3, audiofile);
    fprintf(fid3, '\n');
end
%Close files
fclose(fid);
fclose(fid2);
fclose(fid3);
```

APPENDIX C

```
%%Audio editing script
clear
clc;
%% Audio read
[audName,path] = uigetfile('*.wav*','Select audiofile');
audio = fullfile(path,audName);
[data,Fs] = audioread(audio);
%Channel separation
value = size(data);
left = data(:,1);
if value(2) > 1
    right = data(:,2);
    disp('Two channel audio separated');
else
    disp('Single channel audio readed');
end
%Sampling frequency obtention
message =['File sampling frequency is: ' num2str(Fs) 'Hz'];
disp(message);
%Display 100ns units
tFs = 1 /((100*10^-9)*Fs);
message =['100ns period value is: ' num2str(tFs)];
disp(message);
%Period estimation
prompt = 'Type in desired sampling frequency: ';
freq = input(prompt,'s');
freq = str2double(freq);
tFs = 1 /((100*10^-9)*freq);
message =['100ns estimated period value of ' num2str(freq) 'Hz is: '
num2str(tFs)];
disp(message);
%Plot readed audio
plot(left);
%Plot audio spectrogram
%specgram(left,1024,Fs);
%% Chop data to the following duration 6:44
prompt = 'Skip audio modification? Y/N: ';
question = input(prompt,'s');
checkcondition = strcmp(question,'Y');
if checkcondition ~= 1
    prompt = 'Downsample? Y/N: ';
    question = input(prompt,'s');
    checkcondition2 = strcmp(question,'Y');
    if checkcondition2 ==1
        newaudio = fullfile(path,'newAudio.wav');
        [downchan, freq2] = resample(left,400,441);
        freqz(freq2);
        %downleft = downsample(left,factor);
        audiowrite(newaudio,downchan,freq);
        disp('New downsampled audio file created');
    else
        trimmed = left(1:17816400);
        newaudio = fullfile(path,'newAudio.wav');
        audiowrite(newaudio,trimmed,Fs);
        disp('New trimmed audio file created');
    end
else
```

```
    disp('Audio not modified');
end
```

APPENDIX D

```
%%Perform MFCC feature extraction via HTK V1
clear
clc;
%% Parameter configuration
disp('Setup test');
disp('Select root folder');
%Root folder selection
rootfolder = uigetdir('C:\','Select root folder');
%Directory assignment
parHMM.dir_HTK = [rootfolder '\HTK3.2bin\'];
parHMM.dir_HMM = [rootfolder '\hmmsTrained\'];
parHMM.dir_lists = [rootfolder '\list\'];
parHMM.dir_config = [rootfolder '\config\'];
parHMM.dir_lib = [rootfolder '\lib\'];
parHMM.dir_label = [rootfolder '\label\'];
parHMM.dir_result = [rootfolder '\result\'];
%% File setup
copyConfig = [parHMM.dir_config 'config_HCopy_MFCC_E_D_A'];
copyTrainList = [parHMM.dir_lists 'listTrainHCopy_Wanderer.scp'];
copyTrainList2 = [parHMM.dir_lists 'listTrainHCopy_Wanderer.scp'];
copyTestList = [parHMM.dir_lists 'listTestHCopy_Siegfried2.scp'];
trainConfig = [parHMM.dir_config 'config_train_MFCC_E_D_A'];
trainList = [parHMM.dir_lists 'listTrainFullPath_Wanderer.scp'];
proto = [parHMM.dir_lib 'proto_s1d39_st20m1_MFCC_E_D_A'];
HMMout = [parHMM.dir_HMM 'hmm0'];
%Variable setup
numCoef = '39';
parType = 'MFCC_E_D_A';
floorValue = '0.01';
disp('Setup complete');
%% Log file initialisation
%Open log file
checkfile = exist([parHMM.dir_result 'HMMfeature.log'],'file'); %Check if
file exists
if checkfile ~= 2
    logfile = fopen(fullfile(parHMM.dir_result,'HMMfeature.log'), 'wt' );
%create and write
else
    logfile = fopen(fullfile(parHMM.dir_result,'HMMfeature.log'), 'at' );
%append to existing data on file
end
fprintf(logfile, 'Feautre extraction log file\n');
c = clock; %Get current time/date
fprintf(logfile, 'Date/Time: %s\n', datestr(c)); %Print to file
fprintf(logfile, 'Selected root folder: %s\n', rootfolder);
%% Feature extraction
prompt = 'Skip feature extraction? Y/N: ';
question = input(prompt,'s');
checkcondition = strcmp(question,'Y');
if checkcondition ~= 1
    disp('Extracting features...');
    %Calling HCopy
    callHCopy = strcat(parHMM.dir_HTK,'HCopy');
    %Train files feature extraction

    copysweep = copyTrainList;
    command = [callHCopy,' ', ' -A -D -T 2 -C ', '"',copyConfig,'"',...
        ' -S ', '"',copysweep,'"' ];
    %System command out
    [~,cmdout] = system(command);
    if(~isempty(strfind(cmdout,'ERROR')))
        error(cmdout);
    end
    fprintf(logfile, '%s\n', command);
    fprintf(logfile, '%s\n', cmdout);
    disp(cmdout);

    %Test files feature extraction
    command = [callHCopy,' ', ' -A -D -T 2 -C ', '"',copyConfig,'"',...
        ' -S ', '"',copyTestList,'"' ];
    %System command out
    [~,cmdout] = system(command);
    if(~isempty(strfind(cmdout,'ERROR')))
        error(cmdout);
    end
    fprintf(logfile, '%s\n', command);
    fprintf(logfile, '%s\n', cmdout);
    disp(cmdout);
    disp('Features extracted succesfuly');
else
    disp('Feature extraction skipped');
end
%% HMM definition
%Calling HCompV
callHCompV = strcat(parHMM.dir_HTK,'HCompV');
command = [callHCompV,' ', ' -A -D -T 2 -C ', '"',trainConfig,'"',...
    ' -o ', 'hmmdef',...
    ' -f ', floorValue,...
    ' -m ',...
    ' -S ', '"',trainList,'"',...
    ' -M ','"',HMMout,'"',...
    ' ', '"',proto,'"'
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
disp('Seed HMM produced');
%Macro file
callMacro = strcat(parHMM.dir_HTK,'macro');
vFloors = [HMMout '\vFloors'];
macroOut = [HMMout '\macros'];
command = [callMacro,...
    ' ', numCoef,' ',...
    '"',parType,'"',' ',...
    '"',vFloors,'"',' ',...
    '"',macroOut,'"',' '...
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
disp('Macro file produced');
%Create HMM models file
% Get hmmdef and word list No SP paths
[hmmName,pathHMM] = uigetfile('*.*','Select HMM definition file');
[wordName,pathWord] = uigetfile('*.*','Select word list file');
%Full file paths
gethmm = fullfile(pathHMM,hmmName);
getwords = fullfile(pathWord,wordName);
%Open files
[hmmFile,errormsg] = fopen(gethmm,'rt');
disp(errormsg);
[wordFile,errormsg] = fopen(getwords,'r');
disp(errormsg);
%Create file for writing
models = fopen(fullfile(pathHMM,'models'), 'wt' );
%Read until end of file
i = 0;
wordline = fgetl(wordFile);
%Algorithm for HMM model definition
%Define number of states for the audio model
prompt = 'Type in # of States for audio model (3/6/10): ';
prestates = input(prompt,'s');
audiostates = str2double(prestates);
while ischar(wordline)
    frewind(hmmFile); %Point to start of file
    hmmline = fgetl(hmmFile);
    silcheck = strcmp(wordline,'sil');
    audiocheck = strcmp(wordline,'audio');
    statelock = strcmp(hmmline,'<STATE> 5');
    statemodify = strcmp(hmmline,'<NUMSTATES> 12');
    i = 0;
    %Sil model specification
    if(silcheck ~= 0)
        while statelock ~= 1
            i = i+1;
            comp = strcmp(hmmline,'~h "hmmdef"');
            if i > 3
                disp(hmmline);
                %Num of states
                if(comp ~= 0)
                    wordmodel = ['~h ', '"', wordline, '"'];
                    fprintf(models, '%s\n', wordmodel);
                else
                    if(statemodify ~= 0)
                        fprintf(models, '<NUMSTATES> 5\n');
                    else
                        fprintf(models, '%s\n', hmmline);
                    end

                end
            end
            hmmline = fgetl(hmmFile);
            statelock = strcmp(hmmline,'<STATE> 5');
            statemodify = strcmp(hmmline,'<NUMSTATES> 12');
        end
        %TRANSP Matrix
        fprintf(models, '<TRANSP> 5\n');
        fprintf(models, '0.000000e+000 1.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000\n');
        fprintf(models, '0.000000e+000 6.000000e-001 4.000000e-001 
0.000000e+000 0.000000e+000\n');
        fprintf(models, '0.000000e+000 0.000000e+000 6.000000e-001 4.000000e-
001 0.000000e+000\n');
        fprintf(models, '0.000000e+000 0.000000e+000 0.000000e+000 7.000000e-
001 3.000000e-001\n');
        fprintf(models, '0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000\n');
        fprintf(models, '<ENDHMM>\n');
        %Audio model specification
    elseif(audiocheck ~= 0)
        while statelock ~= 1
            i = i+1;
            comp = strcmp(hmmline,'~h "hmmdef"');
            if i > 3
                disp(hmmline);
                %Num of states
                if(comp ~= 0)
                    wordmodel = ['~h ', '"', wordline, '"'];
                    fprintf(models, '%s\n', wordmodel);
                else
                    if(statemodify ~= 0)
                        fprintf(models, '<NUMSTATES> 5\n');
                    else
                        fprintf(models, '%s\n', hmmline);
                    end

                end
            end
            hmmline = fgetl(hmmFile);
            %Depending on how many states the audio model need
            switch audiostates
                case 6
                    statelock = strcmp(hmmline,'<STATE> 8');
                    statemodify = strcmp(hmmline,'<NUMSTATES> 12');
                case 10
                    statelock = strcmp(hmmline,'<STATE> 12');
                    statemodify = strcmp(hmmline,'<NUMSTATES> 12');
                otherwise
                    statelock = strcmp(hmmline,'<STATE> 5');
                    statemodify = strcmp(hmmline,'<NUMSTATES> 12');
            end
        end
        %TRANSP Matrix
        switch audiostates
            case 6
```

```
                fprintf(models, '<TRANSP> 8\n');
                fprintf(models, '0.000000e+000 1.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000\n');
                fprintf(models, '0.000000e+000 6.000000e-001 4.000000e-001 
0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000\n');
                fprintf(models, '0.000000e+000 0.000000e+000 6.000000e-001 
4.000000e-001 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000\n');
                fprintf(models, '0.000000e+000 0.000000e+000 0.000000e+000 
6.000000e-001 4.000000e-001 0.000000e+000 0.000000e+000 0.000000e+000\n');
                fprintf(models, '0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 6.000000e-001 4.000000e-001 0.000000e+000 0.000000e+000\n');
                fprintf(models, '0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000 6.000000e-001 4.000000e-001 0.000000e+000\n');
                fprintf(models, '0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000 0.000000e+000 6.000000e-001 4.000000e-001\n');
                fprintf(models, '0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000\n');
                fprintf(models, '<ENDHMM>\n');
            case 10
                fprintf(models, '<TRANSP> 12\n');
                fprintf(models, '0.000000e+000 1.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000\n');
                fprintf(models, '0.000000e+000 6.000000e-001 4.000000e-001 
0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000\n');
                fprintf(models, '0.000000e+000 0.000000e+000 6.000000e-001 
4.000000e-001 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000\n');
                fprintf(models, '0.000000e+000 0.000000e+000 0.000000e+000 
6.000000e-001 4.000000e-001 0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000\n');
                fprintf(models, '0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 6.000000e-001 4.000000e-001 0.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000\n');
                fprintf(models, '0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000 6.000000e-001 4.000000e-001 0.000000e+000 
0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000\n');
                fprintf(models, '0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000 0.000000e+000 6.000000e-001 4.000000e-001 
0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000\n');
                fprintf(models, '0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 6.000000e-001 
4.000000e-001 0.000000e+000 0.000000e+000 0.000000e+000\n');
                fprintf(models, '0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 
6.000000e-001 4.000000e-001 0.000000e+000 0.000000e+000\n');
                fprintf(models, '0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 6.000000e-001 4.000000e-001 0.000000e+000\n');
                fprintf(models, '0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000 6.000000e-001 4.000000e-001\n');
                fprintf(models, '0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000\n');
                fprintf(models, '<ENDHMM>\n');
            otherwise
```

```
                fprintf(models, '<TRANSP> 5\n');
                fprintf(models, '0.000000e+000 1.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000\n');
                fprintf(models, '0.000000e+000 6.000000e-001 4.000000e-001 
0.000000e+000 0.000000e+000\n');
                fprintf(models, '0.000000e+000 0.000000e+000 6.000000e-001 
4.000000e-001 0.000000e+000\n');
                fprintf(models, '0.000000e+000 0.000000e+000 0.000000e+000 
6.000000e-001 4.000000e-001\n');
                fprintf(models, '0.000000e+000 0.000000e+000 0.000000e+000 
0.000000e+000 0.000000e+000\n');
                fprintf(models, '<ENDHMM>\n');
        end
        %Any other word
    else
        while ischar(hmmline)
            i = i+1;
            comp = strcmp(hmmline,'~h "hmmdef"');
            if i > 3
                disp(hmmline);
                if(comp ~= 0)
                    wordmodel = ['~h ', '"', wordline, '"'];
                    fprintf(models, '%s\n', wordmodel);
                else
                    fprintf(models, '%s\n', hmmline);
                end
            end
            hmmline = fgetl(hmmFile);
        end
    end
    wordline = fgetl(wordFile);
end

disp('HMM model created for each word on the list');
fprintf(logfile,'HMM model created for each word on the list');
%% Close files
fprintf(logfile, '\n');
fclose(logfile);
fclose(hmmFile);
fclose(wordFile);
fclose(models);
```

APPENDIX D.2

```
%%Perform MFCC feature extraction via HTK V2
clear
clc;
%% Parameter configuration
disp('Setup test');
disp('Select root folder');
%Root folder selection
%rootfolder = uigetdir('C:\','Select root folder');
rootfolder = 'Y:\projectMSc';
%Directory assignment
parHMM.dir_HTK = [rootfolder '\HTK3.2bin\'];
parHMM.dir_HMM = [rootfolder '\hmmsTrained\'];
parHMM.dir_lists = [rootfolder '\list\'];
parHMM.dir_config = [rootfolder '\config\'];
parHMM.dir_lib = [rootfolder '\lib\'];
parHMM.dir_label = [rootfolder '\label\'];
parHMM.dir_result = [rootfolder '\result\'];
%% File setup
copyConfig = [parHMM.dir_config 'config_HCopy_MFCC_E_D_A'];
copyTrainList = [parHMM.dir_lists 'listTrainHCopy_Wanderer.scp'];
copyTrainList2 = [parHMM.dir_lists 'listTrainHCopy_Wanderer.scp'];
copyTestList = [parHMM.dir_lists 'listTestHCopy_Siegfried2.scp'];
trainConfig = [parHMM.dir_config 'config_train_MFCC_E_D_A'];
trainList = [parHMM.dir_lists 'listTrainFullPath_Wanderer.scp'];
proto = [parHMM.dir_lib 'proto_s1d39_st10m1_MFCC_E_D_A'];
HMMout = [parHMM.dir_HMM 'hmm0'];
label = [parHMM.dir_label '\trainlabelnoSP.mlf'];
%Variable setup
numCoef = '39';
parType = 'MFCC_E_D_A';
floorValue = '0.01';
numIter = '50';
currentWord = {'grubel','nibelungen','ring','vertrags','audio','sil'};
disp('Setup complete');
%% Log file initialisation
%Open log file
checkfile = exist([parHMM.dir_result 'HMMfeature.log'],'file'); %Check if
file exists
if checkfile ~= 2
    logfile = fopen(fullfile(parHMM.dir_result,'HMMfeature.log'), 'wt' );
%create and write
else
    logfile = fopen(fullfile(parHMM.dir_result,'HMMfeature.log'), 'at' );
%append to existing data on file
end
fprintf(logfile, 'Feautre extraction log file\n');
c = clock; %Get current time/date
fprintf(logfile, 'Date/Time: %s\n', datestr(c)); %Print to file
fprintf(logfile, 'Selected root folder: %s\n', rootfolder);
%% Feature extraction
prompt = 'Skip feature extraction? Y/N: ';
question = input(prompt,'s');
checkcondition = strcmp(question,'Y');
if checkcondition ~= 1
    disp('Extracting features...');
    %Calling HCopy
    callHCopy = strcat(parHMM.dir_HTK,'HCopy');
    %Train files feature extraction
    command = [callHCopy,' ', ' -A -D -T 2 -C ', '"',copyConfig,'"',...
        ' -S ', '"',copyTrainList,'"' ];
    %System command out
    [~,cmdout] = system(command);
    if(~isempty(strfind(cmdout,'ERROR')))
        error(cmdout);
    end
    fprintf(logfile, '%s\n', command);
    fprintf(logfile, '%s\n', cmdout);
    disp(cmdout);

    %Test files feature extraction
    command = [callHCopy,' ', ' -A -D -T 2 -C ', '"',copyConfig,'"',...
        ' -S ', '"',copyTestList,'"' ];
    %System command out
    [~,cmdout] = system(command);
    if(~isempty(strfind(cmdout,'ERROR')))
        error(cmdout);
    end
    fprintf(logfile, '%s\n', command);
    fprintf(logfile, '%s\n', cmdout);
    disp(cmdout);
    disp('Features extracted succesfuly');
else
    disp('Feature extraction skipped');
end
%% HMM definition
%Changed this from HCompV to HInit (Viterbi alignment)
%Calling HInit
%Isolated word training
disp('Calling HInit...');
callHInit = strcat(parHMM.dir_HTK,'HInit');
command = [callHInit,' ', ' -A -D -T 1 -C ', '"',trainConfig,'"',...
    ' -o ', currentWord{5},...
    ' -i ', numIter,...
    ' -I ',label,...
    ' -l ',currentWord{5},...
    ' -m 1 ',...
    ' -v ',floorValue,...
    ' -S ', '"',trainList,'"',...
    ' -M ','"',HMMout,'"',...
    ' ', '"',proto,'"'
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
disp('Seed HMM produced');

%% Close files
fprintf(logfile, '\n');
fclose(logfile);
%fclose(hmmFile);
%fclose(wordFile);
%fclose(models);
```

APPENDIX E

```
%%Script for training and develop a set of HMMs with the HTK toolkit V1
clear
clc;
%% Parameter configuration
disp('Setup train');
disp('Select root folder');
%Root folder selection
rootfolder = uigetdir('C:\','Select root folder');
%Directory assignment
```

```
parHMM.dir_HTK = [rootfolder '\HTK3.2bin\'];
parHMM.dir_HMM = [rootfolder '\hmmsTrained\'];
parHMM.dir_lists = [rootfolder '\list\'];
parHMM.dir_config = [rootfolder '\config\'];
parHMM.dir_lib = [rootfolder '\lib\'];
parHMM.dir_label = [rootfolder '\label\'];
parHMM.dir_result = [rootfolder '\result\'];
%% File setup
trainConfig = [parHMM.dir_config 'config_train_MFCC_E_D_A'];
trainList = [parHMM.dir_lists 'listTrainFullPath_Wanderer.scp'];
wordListSP = [parHMM.dir_lib 'LeitmotivListSP2'];
wordListNSP = [parHMM.dir_lib 'LeitmotivListnoSP2'];
fixmodel = [parHMM.dir_lib 'leitfixer.hed'];
mix2 = [parHMM.dir_lib 'mix2_st20_a6.hed'];
mix3 = [parHMM.dir_lib 'mix3_st20_a6.hed'];
mix8 = [parHMM.dir_lib 'mix8_st20_a6.hed'];
mix16 = [parHMM.dir_lib 'mix16_st20_a6.hed'];
mix32 = [parHMM.dir_lib 'mix32_st20_a6.hed'];
mix64 = [parHMM.dir_lib 'mix64_st20_a6.hed'];
mix128 = [parHMM.dir_lib 'mix128_st20_a6.hed'];
mix256 = [parHMM.dir_lib 'mix256_st20_a6.hed'];
labelSP = [parHMM.dir_label 'trainlabelnoSP.mlf'];
labelNSP = [parHMM.dir_label 'trainlabelnoSP.mlf'];
labelaudSP = [parHMM.dir_label 'label_leitmotiv_noSP.mlf'];
labelaudNSP = [parHMM.dir_label 'label_audio_noSP.mlf'];
masterlabel = [parHMM.dir_label 'masterlabel.mlf'];
%Pruning Threshold values
threshold = '250.0 150.0 1000.0';
%Ask user for custom parameters
prompt = 'Type in minimum number of samples for training per model: ';
%prompt2 = 'Type in scaling factor value: ';
%Convert to string
mindata = input(prompt, 's');
%scaling = input(prompt2, 's');
disp('Setup complete');
disp('Creating directories...');
%% Log file initialisation
%Open log file
checkfile = exist([parHMM.dir_result 'HMMtrain.log'],'file'); %Check if file
exists
if checkfile ~= 2
    logfile = fopen(fullfile(parHMM.dir_result,'HMMtrain.log'), 'wt' );
%create and write
else
    logfile = fopen(fullfile(parHMM.dir_result,'HMMtrain.log'), 'at' );
%append to existing data on file
end
fprintf(logfile, 'Audio recognition train log file\n');
c = clock; %Get current time/date
fprintf(logfile, 'Date/Time: %s\n', datestr(c)); %Print to file
fprintf(logfile, 'Selected root folder: %s\n', rootfolder);
%% Directory creation
for k=0:100
    check = [parHMM.dir_HMM, 'hmm', num2str(k)];
    condition = exist(check,'dir');
    if(condition ~= 7);
        foldercmd = ['md', ' ', parHMM.dir_HMM, 'hmm', num2str(k)];
        [~,cmdout] = system(foldercmd);
        disp(['HMM ', num2str(k), ' folder created']);
        fprintf(logfile,'HMM %s folder created\n',num2str(k));
    else
        disp('Directory already exists');
        fprintf(logfile,'Directory already exists\n');
    end
end
%% Training
disp('Performing training...');
%Calling HERest
callHERest = strcat(parHMM.dir_HTK,'HERest');
%Perform three iterations
for k=1:3
    bmacros = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\macros'];
    bmodels = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\models'];
    nextHMM = [parHMM.dir_HMM, 'hmm', num2str(k)];
    command = [callHERest,' ', ' -A -D -T 8 -D -C ', '"',trainConfig,'"',...
        ' -I ', '"',labelNSP,'"',...
        ' -t ',threshold,...
        ' -S ', '"',trainList,'"',...
        ' -H ', '"',bmacros,'"',...
        ' -H ', '"',bmodels,'"',...
        ' -M ', '"',nextHMM,'"',...
        ' -m ', '"',mindata,'"',' ',...
        ' -w ', '"','1','"',' ',...
        '"',wordListNSP,'"',...
        ];
    %System command out
    [~,cmdout] = system(command);
    if(~isempty(strfind(cmdout,'ERROR')))
        error(cmdout);
    end
    fprintf(logfile, '%s\n', command);
    fprintf(logfile, '%s\n', cmdout);
    disp(cmdout);
end
%% Model Fixing;
bmodels = [parHMM.dir_HMM, 'hmm', num2str(3), '\models'];
nextHMM = [parHMM.dir_HMM, 'hmm', num2str(4)];
command = ['copy ' parHMM.dir_HMM, 'hmm3', ' ', parHMM.dir_HMM, 'hmm4'];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%SP model gen
callSP = strcat(parHMM.dir_HTK,'spmodel_gen');
command = [callSP,' ', bmodels, ' ', nextHMM, '\models'];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
```

```
disp(cmdout);
%HHEd
bmacros = [parHMM.dir_HMM, 'hmm', num2str(4), '\macros'];
bmodels = [parHMM.dir_HMM, 'hmm', num2str(4), '\models'];
nextHMM = [parHMM.dir_HMM, 'hmm', num2str(5)];
callHHEd = strcat(parHMM.dir_HTK,'HHEd');
command = [callHHEd,' ', ' -A -D -T 2',...
    ' -H ', '"',bmacros,'"',...
    ' -H ', '"',bmodels,'"',...
    ' -M ', '"',nextHMM,'"',' ',...
    '"',fixmodel,'"',' ',...
    '"',wordListSP,'"'
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%% Re estimation
for k=6:8
    bmacros = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\macros'];
    bmodels = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\models'];
    nextHMM = [parHMM.dir_HMM, 'hmm', num2str(k)];
    command = [callHERest,' ', ' -A -D -T 8 -C ', '"',trainConfig,'"',...
        ' -I ', '"',labelSP,'"',...
        ' -S ', '"',trainList,'"',...
        ' -H ', '"',bmacros,'"',...
        ' -H ', '"',bmodels,'"',...
        ' -M ', '"',nextHMM,'"',...
        ' -m ', '"',mindata,'"',' ',...
        ' -w ', '"','1','"',' ',...
        '"',wordListSP,'"'
        ];
    %System command out
    [~,cmdout] = system(command);
    if(~isempty(strfind(cmdout,'ERROR')))
        error(cmdout);
    end
    fprintf(logfile, '%s\n', command);
    fprintf(logfile, '%s\n', cmdout);
    disp(cmdout);
end
%% 2 Gaussians per mixture
bmacros = [parHMM.dir_HMM, 'hmm', num2str(8), '\macros'];
bmodels = [parHMM.dir_HMM, 'hmm', num2str(8), '\models'];
nextHMM = [parHMM.dir_HMM, 'hmm', num2str(9)];
callHHEd = strcat(parHMM.dir_HTK,'HHEd');
command = [callHHEd,' ', ' -A -D -T 2',...
    ' -H ', '"',bmacros,'"',...
    ' -H ', '"',bmodels,'"',...
    ' -M ', '"',nextHMM,'"',' ',...
    '"',mix2,'"',' ',...
    '"',wordListSP,'"'
    ];
%System command out
```

```
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%% Re estimation
for k=10:12
    bmacros = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\macros'];
    bmodels = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\models'];
    nextHMM = [parHMM.dir_HMM, 'hmm', num2str(k)];
    command = [callHERest,' ', ' -A -D -T 8 -C ', '"',trainConfig,'"',...
        ' -I ', '"',labelSP,'"',...
        ' -S ', '"',trainList,'"',...
        ' -H ', '"',bmacros,'"',...
        ' -H ', '"',bmodels,'"',...
        ' -M ', '"',nextHMM,'"',...
        ' -m ', '"',mindata,'"',' ',...
        ' -w ', '"','1','"',' ',...
        '"',wordListSP,'"'
        ];
    %System command out
    [~,cmdout] = system(command);
    if(~isempty(strfind(cmdout,'ERROR')))
        error(cmdout);
    end
    fprintf(logfile, '%s\n', command);
    fprintf(logfile, '%s\n', cmdout);
    disp(cmdout);
end
%% 4 Gaussians per mixture
bmacros = [parHMM.dir_HMM, 'hmm', num2str(12), '\macros'];
bmodels = [parHMM.dir_HMM, 'hmm', num2str(12), '\models'];
nextHMM = [parHMM.dir_HMM, 'hmm', num2str(13)];
callHHEd = strcat(parHMM.dir_HTK,'HHEd');
command = [callHHEd,' ', ' -A -D -T 2',...
    ' -H ', '"',bmacros,'"',...
    ' -H ', '"',bmodels,'"',...
    ' -M ', '"',nextHMM,'"',' ',...
    '"',mix3,'"',' ',...
    '"',wordListSP,'"'
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%% Re estimation
for k=14:20
    bmacros = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\macros'];
    bmodels = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\models'];
    nextHMM = [parHMM.dir_HMM, 'hmm', num2str(k)];
    command = [callHERest,' ', ' -A -D -T 8 -C ', '"',trainConfig,'"',...
        ' -I ', '"',labelSP,'"',...
        ' -S ', '"',trainList,'"',...
        ' -H ', '"',bmacros,'"',...
        ' -H ', '"',bmodels,'"',...
        ' -M ', '"',nextHMM,'"',...
        ' -m ', '"',mindata,'"',' ',...
        ' -w ', '"','1','"',' ',...
        '"',wordListSP,'"'
        ];
    %System command out
    [~,cmdout] = system(command);
    if(~isempty(strfind(cmdout,'ERROR')))
        error(cmdout);
    end
    fprintf(logfile, '%s\n', command);
    fprintf(logfile, '%s\n', cmdout);
    disp(cmdout);
end
%% 8 Gaussians per mixture
bmacros = [parHMM.dir_HMM, 'hmm', num2str(20), '\macros'];
bmodels = [parHMM.dir_HMM, 'hmm', num2str(20), '\models'];
nextHMM = [parHMM.dir_HMM, 'hmm', num2str(21)];
callHHEd = strcat(parHMM.dir_HTK,'HHEd');
command = [callHHEd,' ', ' -A -D -T 2',...
    ' -H ', '"',bmacros,'"',...
    ' -H ', '"',bmodels,'"',...
    ' -M ', '"',nextHMM,'"',' ',...
    '"',mix8,'"',' ',...
    '"',wordListSP,'"'
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%% Re estimation
for k=22:28
    bmacros = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\macros'];
    bmodels = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\models'];
    nextHMM = [parHMM.dir_HMM, 'hmm', num2str(k)];
    command = [callHERest,' ', ' -A -D -T 8 -C ', '"',trainConfig,'"',...
        ' -I ', '"',labelSP,'"',...
        ' -S ', '"',trainList,'"',...
        ' -H ', '"',bmacros,'"',...
        ' -H ', '"',bmodels,'"',...
        ' -M ', '"',nextHMM,'"',...
        ' -m ', '"',mindata,'"',' ',...
        ' -w ', '"','1','"',' ',...
        '"',wordListSP,'"'
        ];
    %System command out
    [~,cmdout] = system(command);
    if(~isempty(strfind(cmdout,'ERROR')))
        error(cmdout);
    end
    fprintf(logfile, '%s\n', command);
    fprintf(logfile, '%s\n', cmdout);
    disp(cmdout);
end
%% 16 Gaussians per mixture
bmacros = [parHMM.dir_HMM, 'hmm', num2str(28), '\macros'];
bmodels = [parHMM.dir_HMM, 'hmm', num2str(28), '\models'];
nextHMM = [parHMM.dir_HMM, 'hmm', num2str(29)];
callHHEd = strcat(parHMM.dir_HTK,'HHEd');
command = [callHHEd,' ', ' -A -D -T 2',...
    ' -H ', '"',bmacros,'"',...
    ' -H ', '"',bmodels,'"',...
    ' -M ', '"',nextHMM,'"',' ',...
    '"',mix16,'"',' ',...
    '"',wordListSP,'"'
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%% Re estimation
for k=30:36
    bmacros = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\macros'];
    bmodels = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\models'];
    nextHMM = [parHMM.dir_HMM, 'hmm', num2str(k)];
    command = [callHERest,' ', ' -A -D -T 8 -C ', '"',trainConfig,'"',...
        ' -I ', '"',labelSP,'"',...
        ' -S ', '"',trainList,'"',...
        ' -H ', '"',bmacros,'"',...
        ' -H ', '"',bmodels,'"',...
        ' -M ', '"',nextHMM,'"',...
        ' -m ', '"',mindata,'"',' ',...
        ' -w ', '"','1','"',' ',...
        '"',wordListSP,'"'
        ];
    %System command out
    [~,cmdout] = system(command);
    if(~isempty(strfind(cmdout,'ERROR')))
        error(cmdout);
    end
    fprintf(logfile, '%s\n', command);
    fprintf(logfile, '%s\n', cmdout);
    disp(cmdout);
end
%% 32 Gaussians per mixture
bmacros = [parHMM.dir_HMM, 'hmm', num2str(36), '\macros'];
bmodels = [parHMM.dir_HMM, 'hmm', num2str(36), '\models'];
nextHMM = [parHMM.dir_HMM, 'hmm', num2str(37)];
callHHEd = strcat(parHMM.dir_HTK,'HHEd');
command = [callHHEd,' ', ' -A -D -T 2',...
    ' -H ', '"',bmacros,'"',...
    ' -H ', '"',bmodels,'"',...
    ' -M ', '"',nextHMM,'"',' ',...
    '"',mix32,'"',' ',...
    '"',wordListSP,'"'
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%% Re estimation
for k=38:45
    bmacros = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\macros'];
    bmodels = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\models'];
    nextHMM = [parHMM.dir_HMM, 'hmm', num2str(k)];
    command = [callHERest,' ', ' -A -D -T 8 -C ', '"',trainConfig,'"',...
        ' -I ', '"',labelSP,'"',...
        ' -S ', '"',trainList,'"',...
        ' -H ', '"',bmacros,'"',...
        ' -H ', '"',bmodels,'"',...
        ' -M ', '"',nextHMM,'"',...
        ' -m ', '"',mindata,'"',' ',...
        ' -w ', '"','1','"',' ',...
        '"',wordListSP,'"'
        ];
    %System command out
    [~,cmdout] = system(command);
    if(~isempty(strfind(cmdout,'ERROR')))
        error(cmdout);
    end
    fprintf(logfile, '%s\n', command);
    fprintf(logfile, '%s\n', cmdout);
    disp(cmdout);
end
%% 64 Gaussians per mixture
bmacros = [parHMM.dir_HMM, 'hmm', num2str(45), '\macros'];
bmodels = [parHMM.dir_HMM, 'hmm', num2str(45), '\models'];
nextHMM = [parHMM.dir_HMM, 'hmm', num2str(46)];
callHHEd = strcat(parHMM.dir_HTK,'HHEd');
command = [callHHEd,' ', ' -A -D -T 2',...
    ' -H ', '"',bmacros,'"',...
    ' -H ', '"',bmodels,'"',...
    ' -M ', '"',nextHMM,'"',' ',...
    '"',mix64,'"',' ',...
    '"',wordListSP,'"'
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%% Re estimation
for k=47:54
    bmacros = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\macros'];
    bmodels = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\models'];
    nextHMM = [parHMM.dir_HMM, 'hmm', num2str(k)];
    command = [callHERest,' ', ' -A -D -T 8 -C ', '"',trainConfig,'"',...
        ' -I ', '"',labelSP,'"',...
        ' -S ', '"',trainList,'"',...
        ' -H ', '"',bmacros,'"',...
        ' -H ', '"',bmodels,'"',...
        ' -M ', '"',nextHMM,'"',...
        ' -m ', '"',mindata,'"',' ',...
        ' -w ', '"','1','"',' ',...
        '"',wordListSP,'"'
        ];
    %System command out
    [~,cmdout] = system(command);
    if(~isempty(strfind(cmdout,'ERROR')))
        error(cmdout);
    end
    fprintf(logfile, '%s\n', command);
    fprintf(logfile, '%s\n', cmdout);
    disp(cmdout);
end
%% 128 Gaussians per mixture
bmacros = [parHMM.dir_HMM, 'hmm', num2str(54), '\macros'];
bmodels = [parHMM.dir_HMM, 'hmm', num2str(54), '\models'];
nextHMM = [parHMM.dir_HMM, 'hmm', num2str(55)];
callHHEd = strcat(parHMM.dir_HTK,'HHEd');
command = [callHHEd,' ', ' -A -D -T 2',...
    ' -H ', '"',bmacros,'"',...
    ' -H ', '"',bmodels,'"',...
    ' -M ', '"',nextHMM,'"',' ',...
    '"',mix128,'"',' ',...
    '"',wordListSP,'"'
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%% Re estimation
for k=56:63
    bmacros = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\macros'];
    bmodels = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\models'];
    nextHMM = [parHMM.dir_HMM, 'hmm', num2str(k)];
    command = [callHERest,' ', ' -A -D -T 8 -C ', '"',trainConfig,'"',...
        ' -I ', '"',labelSP,'"',...
        ' -S ', '"',trainList,'"',...
        ' -H ', '"',bmacros,'"',...
        ' -H ', '"',bmodels,'"',...
        ' -M ', '"',nextHMM,'"',...
        ' -m ', '"',mindata,'"',' ',...
        ' -w ', '"','1','"',' ',...
        '"',wordListSP,'"'
        ];
    %System command out
    [~,cmdout] = system(command);
    if(~isempty(strfind(cmdout,'ERROR')))
        error(cmdout);
    end
    fprintf(logfile, '%s\n', command);
    fprintf(logfile, '%s\n', cmdout);
    disp(cmdout);
end
%% 256 Gaussians per mixture
bmacros = [parHMM.dir_HMM, 'hmm', num2str(63), '\macros'];
bmodels = [parHMM.dir_HMM, 'hmm', num2str(63), '\models'];
nextHMM = [parHMM.dir_HMM, 'hmm', num2str(64)];
callHHEd = strcat(parHMM.dir_HTK,'HHEd');
command = [callHHEd,' ', ' -A -D -T 2',...
    ' -H ', '"',bmacros,'"',...
    ' -H ', '"',bmodels,'"',...
    ' -M ', '"',nextHMM,'"',' ',...
    '"',mix256,'"',' ',...
    '"',wordListSP,'"'
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%% Re estimation
for k=65:70
    bmacros = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\macros'];
    bmodels = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\models'];
    nextHMM = [parHMM.dir_HMM, 'hmm', num2str(k)];
    command = [callHERest,' ', ' -A -D -T 8 -C ', '"',trainConfig,'"',...
        ' -I ', '"',labelSP,'"',...
        ' -S ', '"',trainList,'"',...
        ' -H ', '"',bmacros,'"',...
        ' -H ', '"',bmodels,'"',...
        ' -M ', '"',nextHMM,'"',...
        ' -m ', '"',mindata,'"',' ',...
        ' -w ', '"','1','"',' ',...
        '"',wordListSP,'"'
        ];
    %System command out
    [~,cmdout] = system(command);
    if(~isempty(strfind(cmdout,'ERROR')))
        error(cmdout);
    end
    fprintf(logfile, '%s\n', command);
    fprintf(logfile, '%s\n', cmdout);
    disp(cmdout);
end
%% File close
disp('Training complete');
fprintf(logfile, 'Training complete');
fprintf(logfile,'\n');
fclose(logfile);
```

APPENDIX E.2

```
%%Script for training and develop a set of HMMs with the HTK toolkit V2
```

```
clear
clc;
%% Parameter configuration
disp('Setup train');
disp('Select root folder');
%Root folder selection
%rootfolder = uigetdir('C:\','Select root folder');
rootfolder = 'Y:\projectMSc';
%Directory assignment
parHMM.dir_HTK = [rootfolder '\HTK3.2bin\'];
parHMM.dir_HMM = [rootfolder '\hmmsTrained\'];
parHMM.dir_lists = [rootfolder '\list\'];
parHMM.dir_config = [rootfolder '\config\'];
parHMM.dir_lib = [rootfolder '\lib\'];
parHMM.dir_label = [rootfolder '\label\'];
parHMM.dir_result = [rootfolder '\result\'];
%% File setup
trainConfig = [parHMM.dir_config 'config_train_MFCC_E_D_A'];
trainList = [parHMM.dir_lists 'listTrainFullPath_Wanderer.scp'];
wordListSP = [parHMM.dir_lib 'LeitmotivListSP2'];
wordListFix = [parHMM.dir_lib 'LeitmotivListSP3'];
wordListNSP = [parHMM.dir_lib 'LeitmotivListnoSP2'];
fixmodel = [parHMM.dir_lib 'leitfixer.hed'];
mix2 = [parHMM.dir_lib 'mix2_st20_a6.hed'];
mix3 = [parHMM.dir_lib 'mix3_st20_a6.hed'];
mix8 = [parHMM.dir_lib 'mix8_st20_a6.hed'];
mix16 = [parHMM.dir_lib 'mix16_st20_a6.hed'];
mix32 = [parHMM.dir_lib 'mix32_st20_a6.hed'];
mix64 = [parHMM.dir_lib 'mix64_st20_a6.hed'];
mix128 = [parHMM.dir_lib 'mix128_st20_a6.hed'];
mix256 = [parHMM.dir_lib 'mix256_st20_a6.hed'];
labelSP = [parHMM.dir_label 'trainlabelnoSP.mlf'];
labelNSP = [parHMM.dir_label 'trainlabelnoSP.mlf'];
%Pruning Threshold values
threshold = '250.0 150.0 1000.0';
%Ask user for custom parameters
prompt = 'Type in minimum number of samples for training per model: ';
%prompt2 = 'Type in scaling factor value: ';
%Convert to string
mindata = input(prompt, 's');
currentWord = {'grubel','nibelungen','ring','vertrags','audio','sil'};
fcontrol = 1;
%scaling = input(prompt2, 's');
disp('Setup complete');
disp('Creating directories...');
%% Log file initialisation
%Open log file
checkfile = exist([parHMM.dir_result 'HMMtrain.log'],'file'); %Check if file
exists
if checkfile ~= 2
    logfile = fopen(fullfile(parHMM.dir_result,'HMMtrain.log'), 'wt' );
%create and write
else
    logfile = fopen(fullfile(parHMM.dir_result,'HMMtrain.log'), 'at' );
%append to existing data on file
end
fprintf(logfile, 'Audio recognition train log file\n');
c = clock; %Get current time/date
```

```
fprintf(logfile, 'Date/Time: %s\n', datestr(c)); %Print to file
fprintf(logfile, 'Selected root folder: %s\n', rootfolder);
%% Temporary folders creation
for l=1:2
    check = [parHMM.dir_HMM, 'hmmTemp' num2str(l)];
    condition = exist(check,'dir');
    if(condition ~= 7);
        foldercmd = ['md', ' ', parHMM.dir_HMM, 'hmmTemp' num2str(l)];
        [~,cmdout] = system(foldercmd);
        disp('HMM Temp folder created');
        fprintf(logfile,'HMM Temp folder created\n');
    else
        disp('HMM Temp already exists');
        fprintf(logfile,'Directory already exists\n');
    end
end

%% Training
disp('Performing training...');
%Calling HRest
callHRest = strcat(parHMM.dir_HTK,'HRest');
command = ['copy ' parHMM.dir_HMM, 'hmm0', ' ', parHMM.dir_HMM, 'hmmTemp1'];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
%Perform first iterations
for i=1:length(currentWord)
    for k=1:5
        %bmacros = [parHMM.dir_HMM, 'hmm', num2str(k-1), '\macros'];
        %imodels = [parHMM.dir_HMM, 'hmm', num2str(0), '\' currentWord{i}];
        bmodels = [parHMM.dir_HMM, 'hmmTemp1', '\' currentWord{i}];
        nextHMM = [parHMM.dir_HMM, 'hmmTemp2'];
        command = [callHRest,' ', ' -A -D -T 1 -D -C ',
'"',trainConfig,'"',...
            ' -l ', '"',currentWord{i},'"',...
            ' -t ',...
            ' -I ', '"',labelNSP,'"',...
            ' -S ', '"',trainList,'"',...
            ' -H ', '"',bmodels,'"',...
            ' -M ', '"',nextHMM,'"',...
            ' -m ', '"',mindata,'"',' ',...
            ' -w ', '"','1','"',' ',...
            bmodels,...
            ];
        %System command out
        [~,cmdout] = system(command);
        if(~isempty(strfind(cmdout,'ERROR')))
            error(cmdout);
        end
        fprintf(logfile, '%s\n', command);
        fprintf(logfile, '%s\n', cmdout);
        disp(cmdout);
        command = ['copy ' parHMM.dir_HMM, 'hmmTemp2\',currentWord{i},...
            ' ', parHMM.dir_HMM, 'hmmTemp1'];
        %System command out
        [~,cmdout] = system(command);
    end
end
%Output and copy iterated reestimations
check = [parHMM.dir_HMM, 'hmmREst' num2str(fcontrol)];
condition = exist(check,'dir');
if(condition ~= 7);
    foldercmd = ['md', ' ', parHMM.dir_HMM, 'hmmREst' num2str(fcontrol)];
    [~,cmdout] = system(foldercmd);
    disp('HMM Temp folder created');
    fprintf(logfile,'HMM Temp folder created\n');
else
    disp('HMM Temp already exists');
    fprintf(logfile,'Directory already exists\n');
end
command = ['copy ' parHMM.dir_HMM, 'hmmTemp2', ' ', parHMM.dir_HMM,
'hmmREst', num2str(fcontrol)];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
%Increase control var
fcontrol = fcontrol+1;
%% Model Fixing;
foldercmd = ['md', ' ', parHMM.dir_HMM, 'hmmREst' num2str(fcontrol)];
[~,cmdout] = system(foldercmd);
bmodels = [parHMM.dir_HMM, 'hmmREst', num2str(fcontrol-1), '\sil'];
nextHMM = [parHMM.dir_HMM, 'hmmREst', num2str(fcontrol)];
command = ['copy ' parHMM.dir_HMM, 'hmmREst1', ' ', parHMM.dir_HMM,
'hmmREst2'];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%SP model gen
callSP = strcat(parHMM.dir_HTK,'spmodel_gen');
command = [callSP,' ', bmodels, ' ', nextHMM, '\sil'];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%Increase control var
fcontrol = fcontrol+1;
foldercmd = ['md', ' ', parHMM.dir_HMM, 'hmmREst' num2str(fcontrol)];
[~,cmdout] = system(foldercmd);
command = ['copy ' parHMM.dir_HMM, 'hmmREst2', ' ', parHMM.dir_HMM,
'hmmREst3'];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
```

```
    error(cmdout);
end
%Increase control var
fcontrol = fcontrol+1;
foldercmd = ['md', ' ', parHMM.dir_HMM, 'hmmREst' num2str(fcontrol)];
[~,cmdout] = system(foldercmd);
command = ['copy ' parHMM.dir_HMM, 'hmmREst3', ' ', parHMM.dir_HMM,
'hmmREst4'];
%System command out
[~,cmdout] = system(command);
%HHEd
bmodels = [parHMM.dir_HMM, 'hmmREst', num2str(fcontrol-1), '\sil'];
nextHMM = [parHMM.dir_HMM, 'hmmREst', num2str(fcontrol)];
callHHEd = strcat(parHMM.dir_HTK,'HHEd');
command = [callHHEd,' ', ' -A -D -T 2',...
    ' -H ', '"',bmodels,'"',...
    ' -M ', '"',nextHMM,'"',' ',...
    '"',fixmodel,'"',' ',...
    '"',wordListFix,'"'
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%Next model
fcontrol = fcontrol+1;
%% 2 mix Raise Number of Mixture components
%2 Gaussians per mixture
mixControl = 2;
command = ['copy ' parHMM.dir_HMM, 'hmmREst4', ' ', parHMM.dir_HMM,
'hmmTemp2'];
%System command out
[~,cmdout] = system(command);
%Output and copy iterated reestimations
foldercmd = ['md', ' ', parHMM.dir_HMM, 'hmmREst_' num2str(mixControl)
'mix_INIT'];
[~,cmdout] = system(foldercmd);
%Parameters
bmodels = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{1}];
bmodels2 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{2}];
bmodels3 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{3}];
bmodels4 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{4}];
bmodels5 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{5}];
bmodels6 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{6}];
nextHMM = [parHMM.dir_HMM, 'hmmREst_', num2str(mixControl) 'mix_INIT'];
callHHEd = strcat(parHMM.dir_HTK,'HHEd');
command = [callHHEd,' ', ' -A -D -T 2',...
    ' -H ', '"',bmodels,'"',...
    ' -H ', '"',bmodels2,'"',...
    ' -H ', '"',bmodels3,'"',...
    ' -H ', '"',bmodels4,'"',...
    ' -H ', '"',bmodels5,'"',...
    ' -H ', '"',bmodels6,'"',...
    ' -M ', '"',nextHMM,'"',' ',...
    '"',mix2,'"',' ',...
    '"',wordListSP,'"'
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%Increase control var
fcontrol = fcontrol+1;
%% RE EST 2
command = ['copy ' nextHMM, ' ', parHMM.dir_HMM, 'hmmTemp1'];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
for i=1:length(currentWord)-1
    for k=1:5
        bmodels = [parHMM.dir_HMM, 'hmmTemp1', '\' currentWord{i}];
        nextHMM = [parHMM.dir_HMM, 'hmmTemp2'];
        command = [callHRest,' ', ' -A -D -T 1 -D -C ',
'"',trainConfig,'"',...
            ' -l ', '"',currentWord{i},'"',...
            ' -t ',...
            ' -I ', '"',labelNSP,'"',...
            ' -S ', '"',trainList,'"',...
            ' -H ', '"',bmodels,'"',...
            ' -M ', '"',nextHMM,'"',...
            ' -m ', '"',mindata,'"',' ',...
            ' -v ', '"','0.000001','"',' ',...
            ' -w ', '"','1','"',' ',...
            bmodels,...
            ];
        %System command out
        [~,cmdout] = system(command);
        if(~isempty(strfind(cmdout,'ERROR')))
            error(cmdout);
        end
        fprintf(logfile, '%s\n', command);
        fprintf(logfile, '%s\n', cmdout);
        disp(cmdout);
        command = ['copy ' parHMM.dir_HMM, 'hmmTemp2\',currentWord{i},...
            ' ', parHMM.dir_HMM, 'hmmTemp1'];
        %System command out
        [~,cmdout] = system(command);
    end
end
%Output and copy
command = ['copy ' parHMM.dir_HMM, 'hmmTemp2' ' ', parHMM.dir_HMM,
'hmmREst_2mix_INIT'];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
```

```
end
%% 4 mix Raise Number of Mixture components
%4 Gaussians per mixture
mixControl = 4;
command = ['copy ' parHMM.dir_HMM, 'hmmREst_2mix_INIT', ' ', parHMM.dir_HMM,
'hmmTemp2'];
%System command out
[~,cmdout] = system(command);
%Output and copy iterated reestimations
foldercmd = ['md', ' ', parHMM.dir_HMM, 'hmmREst_' num2str(mixControl)
'mix_INIT'];
[~,cmdout] = system(foldercmd);
%Parameters
bmodels = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{1}];
bmodels2 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{2}];
bmodels3 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{3}];
bmodels4 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{4}];
bmodels5 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{5}];
bmodels6 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{6}];
nextHMM = [parHMM.dir_HMM, 'hmmREst_', num2str(mixControl) 'mix_INIT'];
callHHEd = strcat(parHMM.dir_HTK,'HHEd');
command = [callHHEd,' ', ' -A -D -T 2',...
    ' -H ', '"',bmodels,'"',...
    ' -H ', '"',bmodels2,'"',...
    ' -H ', '"',bmodels3,'"',...
    ' -H ', '"',bmodels4,'"',...
    ' -H ', '"',bmodels5,'"',...
    ' -H ', '"',bmodels6,'"',...
    ' -M ', '"',nextHMM,'"',' ',...
    '"',mix3,'"',' ',...
    '"',wordListSP,'"'
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%Increase control var
fcontrol = fcontrol+1;
%% RE EST 4
command = ['copy ' nextHMM, ' ', parHMM.dir_HMM, 'hmmTemp1'];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
for i=1:length(currentWord)-1
    for k=1:5
        bmodels = [parHMM.dir_HMM, 'hmmTemp1', '\' currentWord{i}];
        nextHMM = [parHMM.dir_HMM, 'hmmTemp2'];
        command = [callHRest,' ', ' -A -D -T 1 -D -C ',
'"',trainConfig,'"',...
            ' -l ', '"',currentWord{i},'"',...
            ' -t ',...
            ' -I ', '"',labelNSP,'"',...
            ' -S ', '"',trainList,'"',...
            ' -H ', '"',bmodels,'"',...
            ' -M ', '"',nextHMM,'"',...
            ' -m ', '"',mindata,'"',' ',...
            ' -v ', '"','0.000001','"',' ',...
            ' -w ', '"','1','"',' ',...
            bmodels,...
            ];
        %System command out
        [~,cmdout] = system(command);
        if(~isempty(strfind(cmdout,'ERROR')))
            error(cmdout);
        end
        fprintf(logfile, '%s\n', command);
        fprintf(logfile, '%s\n', cmdout);
        disp(cmdout);
        command = ['copy ' parHMM.dir_HMM, 'hmmTemp2\',currentWord{i},...
            ' ', parHMM.dir_HMM, 'hmmTemp1'];
        %System command out
        [~,cmdout] = system(command);
    end
end
%Output and copy
command = ['copy ' parHMM.dir_HMM, 'hmmTemp2' ' ', parHMM.dir_HMM,
'hmmREst_4mix_INIT'];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
%% 8 mix Raise Number of Mixture components
%8 Gaussians per mixture
mixControl = 8;
command = ['copy ' parHMM.dir_HMM, 'hmmREst_4mix_INIT', ' ', parHMM.dir_HMM,
'hmmTemp2'];
%System command out
[~,cmdout] = system(command);
%Output and copy iterated reestimations
foldercmd = ['md', ' ', parHMM.dir_HMM, 'hmmREst_' num2str(mixControl)
'mix_INIT'];
[~,cmdout] = system(foldercmd);
%Parameters
bmodels = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{1}];
bmodels2 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{2}];
bmodels3 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{3}];
bmodels4 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{4}];
bmodels5 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{5}];
bmodels6 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{6}];
nextHMM = [parHMM.dir_HMM, 'hmmREst_', num2str(mixControl) 'mix_INIT'];
callHHEd = strcat(parHMM.dir_HTK,'HHEd');
command = [callHHEd,' ', ' -A -D -T 2',...
    ' -H ', '"',bmodels,'"',...
    ' -H ', '"',bmodels2,'"',...
    ' -H ', '"',bmodels3,'"',...
    ' -H ', '"',bmodels4,'"',...
    ' -H ', '"',bmodels5,'"',...
    ' -H ', '"',bmodels6,'"',...
    ' -M ', '"',nextHMM,'"',' ',...
    '"',mix8,'"',' ',...
    '"',wordListSP,'"'
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%Increase control var
fcontrol = fcontrol+1;
%% RE EST 8
command = ['copy ' nextHMM, ' ', parHMM.dir_HMM, 'hmmTemp1'];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
for i=1:length(currentWord)-1
    for k=1:5
        bmodels = [parHMM.dir_HMM, 'hmmTemp1', '\' currentWord{i}];
        nextHMM = [parHMM.dir_HMM, 'hmmTemp2'];
        command = [callHRest,' ', ' -A -D -T 1 -D -C ',
'"',trainConfig,'"',...
            ' -l ', '"',currentWord{i},'"',...
            ' -t ',...
            ' -I ', '"',labelNSP,'"',...
            ' -S ', '"',trainList,'"',...
            ' -H ', '"',bmodels,'"',...
            ' -M ', '"',nextHMM,'"',...
            ' -m ', '"',mindata,'"',' ',...
            ' -v ', '"','0.000001','"',' ',...
            ' -w ', '"','1','"',' ',...
            bmodels,...
            ];
        %System command out
        [~,cmdout] = system(command);
        if(~isempty(strfind(cmdout,'ERROR')))
            error(cmdout);
        end
        fprintf(logfile, '%s\n', command);
        fprintf(logfile, '%s\n', cmdout);
        disp(cmdout);
        command = ['copy ' parHMM.dir_HMM, 'hmmTemp2\',currentWord{i},...
            ' ', parHMM.dir_HMM, 'hmmTemp1'];
        %System command out
        [~,cmdout] = system(command);
    end
end
%Output and copy
command = ['copy ' parHMM.dir_HMM, 'hmmTemp2' ' ', parHMM.dir_HMM,
'hmmREst_8mix_INIT'];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
```

```
end
%% 16 mix Raise Number of Mixture components
%16 Gaussians per mixture
mixControl = 16;
command = ['copy ' parHMM.dir_HMM, 'hmmREst_8mix_INIT', ' ', parHMM.dir_HMM,
'hmmTemp2'];
%System command out
[~,cmdout] = system(command);
%Output and copy iterated reestimations
foldercmd = ['md', ' ', parHMM.dir_HMM, 'hmmREst_' num2str(mixControl)
'mix_INIT'];
[~,cmdout] = system(foldercmd);
%Parameters
bmodels = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{1}];
bmodels2 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{2}];
bmodels3 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{3}];
bmodels4 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{4}];
bmodels5 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{5}];
bmodels6 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{6}];
nextHMM = [parHMM.dir_HMM, 'hmmREst_', num2str(mixControl) 'mix_INIT'];
callHHEd = strcat(parHMM.dir_HTK,'HHEd');
command = [callHHEd,' ', ' -A -D -T 2',...
    ' -H ', '"',bmodels,'"',...
    ' -H ', '"',bmodels2,'"',...
    ' -H ', '"',bmodels3,'"',...
    ' -H ', '"',bmodels4,'"',...
    ' -H ', '"',bmodels5,'"',...
    ' -H ', '"',bmodels6,'"',...
    ' -M ', '"',nextHMM,'"',' ',...
    '"',mix16,'"',' ',...
    '"',wordListSP,'"'
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%Increase control var
fcontrol = fcontrol+1;
%% RE EST 16
command = ['copy ' nextHMM, ' ', parHMM.dir_HMM, 'hmmTemp1'];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
for i=5:length(currentWord)-1
    for k=1:5
        bmodels = [parHMM.dir_HMM, 'hmmTemp1', '\' currentWord{i}];
        nextHMM = [parHMM.dir_HMM, 'hmmTemp2'];
        command = [callHRest,' ', ' -A -D -T 1 -D -C ',
'"',trainConfig,'"',...
            ' -l ', '"',currentWord{i},'"',...
            ' -t ',...
            ' -I ', '"',labelNSP,'"',...
            ' -S ', '"',trainList,'"',...
            ' -H ', '"',bmodels,'"',...
            ' -M ', '"',nextHMM,'"',...
            ' -m ', '"',mindata,'"',' ',...
            ' -v ', '"','0.000001','"',' ',...
            ' -w ', '"','1','"',' ',...
            bmodels,...
            ];
        %System command out
        [~,cmdout] = system(command);
        if(~isempty(strfind(cmdout,'ERROR')))
            error(cmdout);
        end
        fprintf(logfile, '%s\n', command);
        fprintf(logfile, '%s\n', cmdout);
        disp(cmdout);
        command = ['copy ' parHMM.dir_HMM, 'hmmTemp2\',currentWord{i},...
            ' ', parHMM.dir_HMM, 'hmmTemp1'];
        %System command out
        [~,cmdout] = system(command);
    end
end
%Output and copy
command = ['copy ' parHMM.dir_HMM, 'hmmTemp2' ' ', parHMM.dir_HMM,
'hmmREst_16mix_INIT'];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
%% 32 mix Raise Number of Mixture components
%32 Gaussians per mixture
mixControl = 32;
command = ['copy ' parHMM.dir_HMM, 'hmmREst_16mix_INIT', ' ', parHMM.dir_HMM,
'hmmTemp2'];
%System command out
[~,cmdout] = system(command);
%Output and copy iterated reestimations
foldercmd = ['md', ' ', parHMM.dir_HMM, 'hmmREst_' num2str(mixControl)
'mix_INIT'];
[~,cmdout] = system(foldercmd);
%Parameters
bmodels = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{1}];
bmodels2 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{2}];
bmodels3 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{3}];
bmodels4 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{4}];
bmodels5 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{5}];
bmodels6 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{6}];
nextHMM = [parHMM.dir_HMM, 'hmmREst_', num2str(mixControl) 'mix_INIT'];
callHHEd = strcat(parHMM.dir_HTK,'HHEd');
command = [callHHEd,' ', ' -A -D -T 2',...
    ' -H ', '"',bmodels,'"',...
    ' -H ', '"',bmodels2,'"',...
    ' -H ', '"',bmodels3,'"',...
    ' -H ', '"',bmodels4,'"',...
    ' -H ', '"',bmodels5,'"',...
    ' -H ', '"',bmodels6,'"',...
    ' -M ', '"',nextHMM,'"',' ',...
    '"',mix32,'"',' ',...
    '"',wordListSP,'"'
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%Increase control var
fcontrol = fcontrol+1;
%% RE EST 32
command = ['copy ' nextHMM, ' ', parHMM.dir_HMM, 'hmmTemp1'];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
for i=5:length(currentWord)-1
    for k=1:5
        bmodels = [parHMM.dir_HMM, 'hmmTemp1', '\' currentWord{i}];
        nextHMM = [parHMM.dir_HMM, 'hmmTemp2'];
        command = [callHRest,' ', ' -A -D -T 1 -D -C ',
'"',trainConfig,'"',...
            ' -l ', '"',currentWord{i},'"',...
            ' -t ',...
            ' -I ', '"',labelNSP,'"',...
            ' -S ', '"',trainList,'"',...
            ' -H ', '"',bmodels,'"',...
            ' -M ', '"',nextHMM,'"',...
            ' -m ', '"',mindata,'"',' ',...
            ' -v ', '"','0.000001','"',' ',...
            ' -w ', '"','1','"',' ',...
            bmodels,...
            ];
        %System command out
        [~,cmdout] = system(command);
        if(~isempty(strfind(cmdout,'ERROR')))
            error(cmdout);
        end
        fprintf(logfile, '%s\n', command);
        fprintf(logfile, '%s\n', cmdout);
        disp(cmdout);
        command = ['copy ' parHMM.dir_HMM, 'hmmTemp2\',currentWord{i},...
            ' ', parHMM.dir_HMM, 'hmmTemp1'];
        %System command out
        [~,cmdout] = system(command);
    end
end
%Output and copy
command = ['copy ' parHMM.dir_HMM, 'hmmTemp2' ' ', parHMM.dir_HMM,
'hmmREst_32mix_INIT'];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
```

```
end
%% 64 mix Raise Number of Mixture components
%64 Gaussians per mixture
mixControl = 64;
command = ['copy ' parHMM.dir_HMM, 'hmmREst_32mix_INIT', ' ', parHMM.dir_HMM,
'hmmTemp2'];
%System command out
[~,cmdout] = system(command);
%Output and copy iterated reestimations
foldercmd = ['md', ' ', parHMM.dir_HMM, 'hmmREst_' num2str(mixControl)
'mix_INIT'];
[~,cmdout] = system(foldercmd);
%Parameters
bmodels = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{1}];
bmodels2 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{2}];
bmodels3 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{3}];
bmodels4 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{4}];
bmodels5 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{5}];
bmodels6 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{6}];
nextHMM = [parHMM.dir_HMM, 'hmmREst_', num2str(mixControl) 'mix_INIT'];
callHHEd = strcat(parHMM.dir_HTK,'HHEd');
command = [callHHEd,' ', ' -A -D -T 2',...
    ' -H ', '"',bmodels,'"',...
    ' -H ', '"',bmodels2,'"',...
    ' -H ', '"',bmodels3,'"',...
    ' -H ', '"',bmodels4,'"',...
    ' -H ', '"',bmodels5,'"',...
    ' -H ', '"',bmodels6,'"',...
    ' -M ', '"',nextHMM,'"',' ',...
    '"',mix64,'"',' ',...
    '"',wordListSP,'"'
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%Increase control var
fcontrol = fcontrol+1;
%% RE EST 64
command = ['copy ' nextHMM, ' ', parHMM.dir_HMM, 'hmmTemp1'];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
for i=5:length(currentWord)-1
    for k=1:5
        bmodels = [parHMM.dir_HMM, 'hmmTemp1', '\' currentWord{i}];
        nextHMM = [parHMM.dir_HMM, 'hmmTemp2'];
        command = [callHRest,' ', ' -A -D -T 1 -D -C ',
'"',trainConfig,'"',...
            ' -l ', '"',currentWord{i},'"',...
            ' -t ',...
            ' -I ', '"',labelNSP,'"',...
```

```
                ' -S ', '"',trainList,'"',...
                ' -H ', '"',bmodels,'"',...
                ' -M ', '"',nextHMM,'"',...
                ' -m ', '"',mindata,'"',' ',...
                ' -v ', '"','0.000001','"',' ',...
                ' -w ', '"','1','"',' ',...
                bmodels,...
                ];
            %System command out
            [~,cmdout] = system(command);
            if(~isempty(strfind(cmdout,'ERROR')))
                error(cmdout);
            end
            fprintf(logfile, '%s\n', command);
            fprintf(logfile, '%s\n', cmdout);
            disp(cmdout);
            command = ['copy ' parHMM.dir_HMM, 'hmmTemp2\',currentWord{i},...
                ' ', parHMM.dir_HMM, 'hmmTemp1'];
            %System command out
            [~,cmdout] = system(command);
        end
    end
    %Output and copy
    command = ['copy ' parHMM.dir_HMM, 'hmmTemp2' ' ', parHMM.dir_HMM,
    'hmmREst_64mix_INIT'];
    %System command out
    [~,cmdout] = system(command);
    if(~isempty(strfind(cmdout,'ERROR')))
        error(cmdout);
    end
    %% 128 mix Raise Number of Mixture components
    %128 Gaussians per mixture
    mixControl = 128;
    command = ['copy ' parHMM.dir_HMM, 'hmmREst_64mix_INIT', ' ', parHMM.dir_HMM,
    'hmmTemp2'];
    %System command out
    [~,cmdout] = system(command);
    %Output and copy iterated reestimations
    foldercmd = ['md', ' ', parHMM.dir_HMM, 'hmmREst_' num2str(mixControl)
    'mix_INIT'];
    [~,cmdout] = system(foldercmd);
    %Parameters
    bmodels = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{1}];
    bmodels2 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{2}];
    bmodels3 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{3}];
    bmodels4 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{4}];
    bmodels5 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{5}];
    bmodels6 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{6}];
    nextHMM = [parHMM.dir_HMM, 'hmmREst_', num2str(mixControl) 'mix_INIT'];
    callHHEd = strcat(parHMM.dir_HTK,'HHEd');
    command = [callHHEd,' ', ' -A -D -T 2',...
        ' -H ', '"',bmodels,'"',...
        ' -H ', '"',bmodels2,'"',...
        ' -H ', '"',bmodels3,'"',...
        ' -H ', '"',bmodels4,'"',...
        ' -H ', '"',bmodels5,'"',...
        ' -H ', '"',bmodels6,'"',...
        ' -M ', '"',nextHMM,'"',' ',...
        '"',mix128,'"',' ',...
        '"',wordListSP,'"'
        ];
    %System command out
    [~,cmdout] = system(command);
    if(~isempty(strfind(cmdout,'ERROR')))
        error(cmdout);
    end
    fprintf(logfile, '%s\n', command);
    fprintf(logfile, '%s\n', cmdout);
    disp(cmdout);
    %Increase control var
    fcontrol = fcontrol+1;
    %% RE EST 128
    command = ['copy ' nextHMM, ' ', parHMM.dir_HMM, 'hmmTemp1'];
    %System command out
    [~,cmdout] = system(command);
    if(~isempty(strfind(cmdout,'ERROR')))
        error(cmdout);
    end
    for i=5:length(currentWord)-1
        for k=1:5
            bmodels = [parHMM.dir_HMM, 'hmmTemp1', '\' currentWord{i}];
            nextHMM = [parHMM.dir_HMM, 'hmmTemp2'];
            command = [callHRest,' ', ' -A -D -T 1 -D -C ',
    '"',trainConfig,'"',...
                ' -l ', '"',currentWord{i},'"',...
                ' -t ',...
                ' -I ', '"',labelNSP,'"',...
                ' -S ', '"',trainList,'"',...
                ' -H ', '"',bmodels,'"',...
                ' -M ', '"',nextHMM,'"',...
                ' -m ', '"',mindata,'"',' ',...
                ' -v ', '"','0.000001','"',' ',...
                ' -w ', '"','1','"',' ',...
                bmodels,...
                ];
            %System command out
            [~,cmdout] = system(command);
            if(~isempty(strfind(cmdout,'ERROR')))
                error(cmdout);
            end
            fprintf(logfile, '%s\n', command);
            fprintf(logfile, '%s\n', cmdout);
            disp(cmdout);
            command = ['copy ' parHMM.dir_HMM, 'hmmTemp2\',currentWord{i},...
                ' ', parHMM.dir_HMM, 'hmmTemp1'];
            %System command out
            [~,cmdout] = system(command);
        end
    end
    %Output and copy
    command = ['copy ' parHMM.dir_HMM, 'hmmTemp2' ' ', parHMM.dir_HMM,
    'hmmREst_128mix_INIT'];
    %System command out
    [~,cmdout] = system(command);
    if(~isempty(strfind(cmdout,'ERROR')))
        error(cmdout);
```

```
end
%% 256 mix Raise Number of Mixture components
%256 Gaussians per mixture
mixControl = 256;
command = ['copy ' parHMM.dir_HMM, 'hmmREst_128mix_INIT', ' ',
parHMM.dir_HMM, 'hmmTemp2'];
%System command out
[~,cmdout] = system(command);
%Output and copy iterated reestimations
foldercmd = ['md', ' ', parHMM.dir_HMM, 'hmmREst_' num2str(mixControl)
'mix_INIT'];
[~,cmdout] = system(foldercmd);
%Parameters
bmodels = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{1}];
bmodels2 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{2}];
bmodels3 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{3}];
bmodels4 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{4}];
bmodels5 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{5}];
bmodels6 = [parHMM.dir_HMM, 'hmmTemp2\' currentWord{6}];
nextHMM = [parHMM.dir_HMM, 'hmmREst_', num2str(mixControl) 'mix_INIT'];
callHHEd = strcat(parHMM.dir_HTK,'HHEd');
command = [callHHEd,' ', ' -A -D -T 2',...
    ' -H ', '"',bmodels,'"',...
    ' -H ', '"',bmodels2,'"',...
    ' -H ', '"',bmodels3,'"',...
    ' -H ', '"',bmodels4,'"',...
    ' -H ', '"',bmodels5,'"',...
    ' -H ', '"',bmodels6,'"',...
    ' -M ', '"',nextHMM,'"',' ',...
    '"',mix256,'"',' ',...
    '"',wordListSP,'"'
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%Increase control var
fcontrol = fcontrol+1;
%% RE EST 256
command = ['copy ' nextHMM, ' ', parHMM.dir_HMM, 'hmmTemp1'];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
for i=5:length(currentWord)-1
    for k=1:5
        bmodels = [parHMM.dir_HMM, 'hmmTemp1', '\' currentWord{i}];
        nextHMM = [parHMM.dir_HMM, 'hmmTemp2'];
        command = [callHRest,' ', ' -A -D -T 1 -D -C ',
'"',trainConfig,'"',...
            ' -l ', '"',currentWord{i},'"',...
            ' -t ',...
            ' -I ', '"',labelNSP,'"',...
            ' -S ', '"',trainList,'"',...
            ' -H ', '"',bmodels,'"',...
            ' -M ', '"',nextHMM,'"',...
            ' -m ', '"',mindata,'"',' ',...
            ' -v ', '"','0.000001','"',' ',...
            ' -w ', '"','1','"',' ',...
            bmodels,...
            ];
        %System command out
        [~,cmdout] = system(command);
        if(~isempty(strfind(cmdout,'ERROR')))
            error(cmdout);
        end
        fprintf(logfile, '%s\n', command);
        fprintf(logfile, '%s\n', cmdout);
        disp(cmdout);
        command = ['copy ' parHMM.dir_HMM, 'hmmTemp2\',currentWord{i},...
            ' ', parHMM.dir_HMM, 'hmmTemp1'];
        %System command out
        [~,cmdout] = system(command);
    end
end
%Output and copy
command = ['copy ' parHMM.dir_HMM, 'hmmTemp2' ' ', parHMM.dir_HMM,
'hmmREst_256mix_INIT'];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
%% File close
disp('Training complete');
fprintf(logfile, 'Training complete');
fprintf(logfile,'\n');
fclose(logfile);
```

APPENDIX F

```
%%Script for testing a developed set of HMMs with the HTK toolkit V1
clear
clc;
%% Parameter configuration
disp('Setup test');
disp('Select root folder');
%Root folder selection
%rootfolder = uigetdir('C:\','Select root folder');
rootfolder = 'Y:\projectMSc';
%Directory assignment
parHMM.dir_HTK = [rootfolder '\HTK3.2bin\'];
parHMM.dir_HMM = [rootfolder '\HCOMPV' '\hmmsTrained_A_10_256_LM_20_8\'];
parHMM.dir_lists = [rootfolder '\list\'];
parHMM.dir_config = [rootfolder '\config\'];
parHMM.dir_lib = [rootfolder '\lib\'];
parHMM.dir_label = [rootfolder '\label\'];
parHMM.dir_result = [rootfolder '\result\'];
%% File setup
```

```
testConfig = [parHMM.dir_config 'config_test_MFCC_E_D_A'];
testList = [parHMM.dir_lists 'listTestFullPath_Siegfried.scp'];
wordList = [parHMM.dir_lib 'LeitmotivListSP2'];
network = [parHMM.dir_lib 'genNetwork'];
dictionary = [parHMM.dir_lib 'LeitmotivDictionary2'];
var = '70';
models = [parHMM.dir_HMM 'hmm' var '\models'];
macros = [parHMM.dir_HMM 'hmm' var '\macros'];
testRes = [parHMM.dir_result 'testResultsCOMPV.mlf'];
%Ask user for the flags values
prompt = 'Type in insertion penalty value: ';
prompt2 = 'Type in scaling factor value: ';
%Convert to string
penalty = input(prompt, 's');
scaling = input(prompt2, 's');
disp('Setup complete');
disp('Performing test...');
%% Log file initialisation
%Open log file
checkfile = exist([parHMM.dir_result 'HMMtest.log'],'file'); %Check if file
exists
if checkfile ~= 2
    logfile = fopen(fullfile(parHMM.dir_result,'HMMtest.log'), 'wt' );
%create and write
else
    logfile = fopen(fullfile(parHMM.dir_result,'HMMtest.log'), 'at' );
%append to existing data on file
end
%Open test parameters file
testPars = fopen(fullfile(parHMM.dir_result,'testParsCOMP.txt'), 'wt' );
%open file to record p and s values
modelPars = fopen(fullfile(models), 'rt' ); %open file to record p and s
values
%Write to log file
fprintf(logfile, 'Audio recognition test log file\n');
c = clock; %Get current time/date
fprintf(logfile, 'Date/Time: %s\n', datestr(c)); %Print to file
fprintf(logfile, 'Selected root folder: %s\n', rootfolder);
fprintf(logfile, 'Insertion penalty p: %s\n', penalty);
fprintf(logfile, 'Scaling factor s: %s\n', scaling);
%Get current Test Parameters
modelline = fgetl(modelPars);
lcount = 0;
flag1 = 0;
flag2 = 0;
%Obtain NumMixes and NumStates for current test
while ischar(modelline)
    lcount = lcount+1;
    lineparts = strsplit(modelline);
    index = size(lineparts);
    if index(2) > 1;
        cond = strcmp(lineparts{2},'"grubel"');
        condaud = strcmp(lineparts{2},'"audio"');
        if cond == 1
            flag1 = 1;
            flag2 = 0;
        end
        if condaud == 1
```

```
            flag2 = 1;
            flag1 = 0;
        end
        if flag1 == 1
            cond2 = strcmp(lineparts{1},'<NUMSTATES>');
            if cond2 == 1
                motivState = fix(str2double(lineparts{2})-2);
                cond2 = 0;
            end
            cond3 = strcmp(lineparts{1},'<NUMMIXES>');
            if cond3 == 1
                motivMix = fix(str2double(lineparts{2}));
                flag1 = 0;
                cond3 = 0;
            else
                motivMix = 1;
            end
        end
        if flag2 == 1
            cond2 = strcmp(lineparts{1},'<NUMSTATES>');
            if cond2 == 1
                audioState = str2double(lineparts{2})-2;
                cond2 = 0;
            end
            cond3 = strcmp(lineparts{1},'<NUMMIXES>');
            if cond3 == 1
                audioMix = fix(str2double(lineparts{2}));
                flag2 = 0;
                cond3 = 0;
            else
                audioMix = 1;
            end
        end
    end
    modelline = fgetl(modelPars); %Get next line
end
%Write value to files
fprintf(testPars, '%s\n', penalty);
fprintf(testPars, '%s\n', scaling);
fprintf(testPars, '%s\n', num2str(audioState));
fprintf(testPars, '%s\n', num2str(audioMix));
fprintf(testPars, '%s\n', num2str(motivState));
fprintf(testPars, '%s\n', num2str(motivMix));
%% Testing
%Calling HVite
callHVite = strcat(parHMM.dir_HTK,'HVite');
command = [callHVite,' ', ' -A -D -T 1 -C ', '"',testConfig,'"',...
    ' -H ', '"',macros,'"',...
    ' -H ', '"',models,'"',...
    ' -S ', '"',testList,'"',...
    ' -w ', '"',network,'"',...
    ' -o SW -m -i ','"',testRes,'"',...
    ' -p ', penalty,...
    ' -s ', scaling,' ',...
    '"',dictionary,'"',' ',...
    '"',wordList,'"'
    ];
%System command out
```

```
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%% Close file
fprintf(logfile, '\n');
fclose(logfile);
fclose(testPars);
%% Run results script
getResults = fullfile('hmmResult_v1.m');
run(getResults);
```

APPENDIX F.2

```
%%Script for testing a developed set of HMMs with the HTK toolkit V1
clear
clc;
%% Parameter configuration
disp('Setup test');
disp('Select root folder');
%Root folder selection
%rootfolder = uigetdir('C:\','Select root folder');
rootfolder = 'Y:\projectMSc';
%Directory assignment
parHMM.dir_HTK = [rootfolder '\HTK3.2bin\'];
parHMM.dir_HMM = [rootfolder '\HCOMPV' '\hmmsTrained_A_10_256_LM_20_8\'];
parHMM.dir_lists = [rootfolder '\list\'];
parHMM.dir_config = [rootfolder '\config\'];
parHMM.dir_lib = [rootfolder '\lib\'];
parHMM.dir_label = [rootfolder '\label\'];
parHMM.dir_result = [rootfolder '\result\'];
%% File setup
testConfig = [parHMM.dir_config 'config_test_MFCC_E_D_A'];
testList = [parHMM.dir_lists 'listTestFullPath_Siegfried.scp'];
wordList = [parHMM.dir_lib 'LeitmotivListSP2'];
network = [parHMM.dir_lib 'genNetwork'];
dictionary = [parHMM.dir_lib 'LeitmotivDictionary2'];
var = '70';
models = [parHMM.dir_HMM 'hmm' var '\models'];
macros = [parHMM.dir_HMM 'hmm' var '\macros'];
testRes = [parHMM.dir_result 'testResultsCOMPV.mlf'];
%Ask user for the flags values
prompt = 'Type in insertion penalty value: ';
prompt2 = 'Type in scaling factor value: ';
%Convert to string
penalty = input(prompt, 's');
scaling = input(prompt2, 's');
disp('Setup complete');
disp('Performing test...');
%% Log file initialisation
%Open log file
checkfile = exist([parHMM.dir_result 'HMMtest.log'],'file'); %Check if file
exists
if checkfile ~= 2
```

```
    logfile = fopen(fullfile(parHMM.dir_result,'HMMtest.log'), 'wt' );
%create and write
else
    logfile = fopen(fullfile(parHMM.dir_result,'HMMtest.log'), 'at' );
%append to existing data on file
end
%Open test parameters file
testPars = fopen(fullfile(parHMM.dir_result,'testParsCOMP.txt'), 'wt' );
%open file to record p and s values
modelPars = fopen(fullfile(models), 'rt' ); %open file to record p and s
values
%Write to log file
fprintf(logfile, 'Audio recognition test log file\n');
c = clock; %Get current time/date
fprintf(logfile, 'Date/Time: %s\n', datestr(c)); %Print to file
fprintf(logfile, 'Selected root folder: %s\n', rootfolder);
fprintf(logfile, 'Insertion penalty p: %s\n', penalty);
fprintf(logfile, 'Scaling factor s: %s\n', scaling);
%Get current Test Parameters
modelline = fgetl(modelPars);
lcount = 0;
flag1 = 0;
flag2 = 0;
%Obtain NumMixes and NumStates for current test
while ischar(modelline)
    lcount = lcount+1;
    lineparts = strsplit(modelline);
    index = size(lineparts);
    if index(2) > 1;
        cond = strcmp(lineparts{2},'"grubel"');
        condaud = strcmp(lineparts{2},'"audio"');
        if cond == 1
            flag1 = 1;
            flag2 = 0;
        end
        if condaud == 1
            flag2 = 1;
            flag1 = 0;
        end
        if flag1 == 1
            cond2 = strcmp(lineparts{1},'<NUMSTATES>');
            if cond2 == 1
                motivState = fix(str2double(lineparts{2})-2);
                cond2 = 0;
            end
            cond3 = strcmp(lineparts{1},'<NUMMIXES>');
            if cond3 == 1
                motivMix = fix(str2double(lineparts{2}));
                flag1 = 0;
                cond3 = 0;
            else
                motivMix = 1;
            end
        end
        if flag2 == 1
            cond2 = strcmp(lineparts{1},'<NUMSTATES>');
            if cond2 == 1
                audioState = str2double(lineparts{2})-2;
                cond2 = 0;
            end
            cond3 = strcmp(lineparts{1},'<NUMMIXES>');
            if cond3 == 1
                audioMix = fix(str2double(lineparts{2}));
                flag2 = 0;
                cond3 = 0;
            else
                audioMix = 1;
            end
        end
    end
    modelline = fgetl(modelPars); %Get next line
end
%Write value to files
fprintf(testPars, '%s\n', penalty);
fprintf(testPars, '%s\n', scaling);
fprintf(testPars, '%s\n', num2str(audioState));
fprintf(testPars, '%s\n', num2str(audioMix));
fprintf(testPars, '%s\n', num2str(motivState));
fprintf(testPars, '%s\n', num2str(motivMix));
%% Testing
%Calling HVite
callHVite = strcat(parHMM.dir_HTK,'HVite');
command = [callHVite,' ', ' -A -D -T 1 -C ', '"',testConfig,'"',...
    ' -H ', '"',macros,'"',...
    ' -H ', '"',models,'"',...
    ' -S ', '"',testList,'"',...
    ' -w ', '"',network,'"',...
    ' -o SW -m -i ','"',testRes,'"',...
    ' -p ', penalty,...
    ' -s ', scaling,' ',...
    '"',dictionary,'"',' ',...
    '"',wordList,'"'
    ];
%System command out
[~,cmdout] = system(command);
if(~isempty(strfind(cmdout,'ERROR')))
    error(cmdout);
end
fprintf(logfile, '%s\n', command);
fprintf(logfile, '%s\n', cmdout);
disp(cmdout);
%% Close file
fprintf(logfile, '\n');
fclose(logfile);
fclose(testPars);
%% Run results script
getResults = fullfile('hmmResult_v1.m');
run(getResults);
```

APPENDIX G

```
 %%Script for compraing HVite results with custom label
clear
clc;
%% File opening
disp('Setup recogniton test results');
%disp('Select root folder');
%Root folder selection
%rootfolder = uigetdir('C:\','Select root folder');
rootfolder = 'Y:\projectMSc';
parHMM.dir_result = [rootfolder '\result\'];
parHMM.dir_data = [rootfolder '\data\'];
disp('Opening label files...');
%HVite label result
%[viteName,vitePath] = uigetfile({'*.mlf',...
%    'MLF Files (*.mlf)'},...
%'Select Viterbi file result');
viteName = 'testResultsINIT.mlf';
vitePath = parHMM.dir_result;
viteFile = fullfile(vitePath,viteName);
%Custom label
%[leitName,leitPath] = uigetfile({'*.mlf',...
%    'MLF Files (*.mlf)'},...
%    'Select custom label file');
leitName = 'labelout.mlf';
leitPath = rootfolder;
leitFile = fullfile(leitPath,leitName);
testPars = fullfile(parHMM.dir_result,'testPars.txt');
%Open files
[viteID,errormsg] = fopen(viteFile,'rt');
disp(errormsg);
[leitID,errormsg] = fopen(leitFile,'r');
disp(errormsg);
[testParsID,errormsg] = fopen(testPars,'r');
disp('Files opened');
%Read and store file elements
viteline = fgetl(viteID);
leitline = fgetl(leitID);
testline = fgetl(testParsID);
count = 0;
vcount = 0;
lcount = 0;
parcount = 0;
vitecmp1 = 0;
vitecmp2 = 0;
vitecmp3 = 0;
%Get penalty and scaling values
while ischar(testline)
    parcount = parcount+1;
    testline = fgetl(testParsID);
end
testValues = cell(parcount,1);
testValuesN = zeros(parcount,1);
frewind(testParsID);
testline = fgetl(testParsID);
i=1;
while ischar(testline)
    testValues{i} = testline;
    i=i+1;
    testline = fgetl(testParsID);
end
%Cycle to count number of lines for optimizing storage
while ischar(viteline)
    viteArray = strsplit(viteline,' ');
    count = count+1;
    vitend = strcmp(viteArray(1,1), '.');
    if (count > 2)&&(vitend ~= 1)
        vitecmp1 = strcmp(viteArray(1,3),'sil');
        vitecmp2 = strcmp(viteArray(1,3),'sp');
        %vitecmp3 = strcmp(viteArray(1,3),'audio');
        if (vitecmp1 == 1)||(vitecmp2 == 1)
            %disp('Ignored line for array allocation');
        else
            vcount = vcount+1;
        end
    end
    viteline = fgetl(viteID);
end
%Initialise array sizes
vitemotivs = cell(vcount,1);
vitetimes = zeros(vcount,2);
%Restart Counters
count = 0;
vcount = 0;
frewind(viteID);
viteline = fgetl(viteID);
%Store the data
while ischar(viteline)
    viteArray = strsplit(viteline,' ');
    count = count+1;
    vitend = strcmp(viteArray(1,1), '.');
    if (count > 2)&&(vitend ~= 1)
        vitecmp1 = strcmp(viteArray(1,3),'sil');
        vitecmp2 = strcmp(viteArray(1,3),'sp');
        vitecmp3 = strcmp(viteArray(1,3),'audio');
        if ((vitecmp1 == 1)||(vitecmp2 == 1))
            %disp('Not applicable');
        else
            vcount = vcount+1;
            vitemotivs(vcount) = viteArray(1,3);
            vitetimes(vcount,1) = str2double(viteArray(1,1));
            vitetimes(vcount,2) = str2double(viteArray(1,2));
            %disp(viteline);
        end
    end
    viteline = fgetl(viteID);
end
```

```
%Restart Counter
count = 0;
%Cycle to count number of lines for optimizing storage
while ischar(leitline)
    leitArray = strsplit(leitline,' ');
    count = count+1;
    leitend = strcmp(leitArray(1,1), '.');
    if (count > 2)&&(leitend ~= 1)
        leitcmp1 = strcmp(leitArray(1,3),'sil');
        leitcmp2 = strcmp(leitArray(1,3),'sp');
        leitcmp3 = strcmp(leitArray(1,3),'audio');
        %(leitcmp3 == 1)
        if (leitcmp1 == 1)||(leitcmp2 == 1)||(leitend ==1)
            %disp('Ignored line');
        else
            lcount = lcount+1;
        end
    end
    leitline = fgetl(leitID);
end
%Initialise Arrays
leitmotivs = cell(lcount,1);
leittimes = zeros(lcount,2);
%Restart Counters
count = 0;
lcount = 0;
frewind(leitID);
leitline = fgetl(leitID);
%Store the data for test label file
while ischar(leitline)
    leitArray = strsplit(leitline,' ');
    count = count+1;
    leitend = strcmp(leitArray(1,1), '.');
    if (count > 2)&&(leitend ~= 1)
        leitcmp1 = strcmp(leitArray(1,3),'sil');
        leitcmp2 = strcmp(leitArray(1,3),'sp');
        leitcmp3 = strcmp(leitArray(1,3),'audio');
        if (leitcmp1 == 1)||(leitcmp2 == 1)||(leitend ==1)
            %disp('Not applicable');
        else
            lcount = lcount+1;
            leitmotivs(lcount) = leitArray(1,3);
            leittimes(lcount,1) = str2double(leitArray(1,1));
            leittimes(lcount,2) = str2double(leitArray(1,2));
        end
    end
    leitline = fgetl(leitID);
end
%% Evaluation application
%Vitetimes and leittimes are compared with a matching or closest value
%Each time has a name associated with it
%Init time comparison variables
disp('Search initiated...');
currsTime = vitetimes(1,1);
curreTime = vitetimes(1,2);
leitrefst = leittimes(1,1);
leitrefend = leittimes(1,2);
alphaLock = 0;
```

```
control = 1;
vindex = 1;
lindex = 1;
hits = 0;
subs = 0;
leitMask = zeros(length(leitmotivs),1);
viteMask = zeros(length(vitemotivs),1);
ltimeMask = zeros(length(leittimes),1);
vtimeMask = zeros(length(vitetimes),1);
subVMask = zeros(length(vitemotivs),1);
subLMask = zeros(length(leitmotivs),1);
insVMask = zeros(length(vitemotivs),1);
delLMask = zeros(length(leitmotivs),1);
i=1;
%Number of Hits detection
while alphaLock == 0
    currentDuration = curreTime - currsTime;
    leitDuration = leitrefend - leitrefst;
    leitMid = leitrefst + leitDuration/2;
    shift = 5000000;
    backref = leitrefst-shift;
    namecheck = strcmp(vitemotivs{vindex},leitmotivs{lindex});
    modelisaudio = strcmp(vitemotivs{vindex},'audio');
    %If names match
    if namecheck == 1
        %Check if time shift is correct
        if (currsTime >= backref) && (currsTime <= leitMid)
            if modelisaudio ~=1 %Audio will not be considered a hit class
                %Mark hits
                viteMask(vindex) = 1;
                leitMask(lindex) = 1;
                hits = hits+1;
                lindex = lindex+1; %Displace index to next motiv
                vindex = 1; %Restart
                currsTime = vitetimes(vindex,1);
                curreTime = vitetimes(vindex,2);
                leitrefst = leittimes(lindex,1);
                leitrefend = leittimes(lindex,2);
                continue; %Start next loop iteration
            else
                viteMask(vindex) = 2;
                leitMask(lindex) = 2;
                lindex = lindex+1; %Displace index to next motiv
                vindex = 1; %Restart
                currsTime = vitetimes(vindex,1);
                curreTime = vitetimes(vindex,2);
                leitrefst = leittimes(lindex,1);
                leitrefend = leittimes(lindex,2);
                continue; %Start next loop iteration
            end
        else %Name match, time incorrect
            vindex = vindex+1;  %Displace current Index
            if vindex > length(vitemotivs)
                vindex = 1; %Prevent out of index array accessing
                lindex = lindex+1; %Shift to compare next leitmotiv
                currsTime = vitetimes(vindex,1);
                curreTime = vitetimes(vindex,2);
                leitrefst = leittimes(lindex,1);
                leitrefend = leittimes(lindex,2);
                continue;
            else %Not out of bounds, continue comparing
                currsTime = vitetimes(vindex,1);
                curreTime = vitetimes(vindex,2);
                if currsTime > leitrefend
                    vindex = 1;
                    lindex = lindex+1;
                    leitrefst = leittimes(lindex,1);
                    leitrefend = leittimes(lindex,2);
                    continue; %If frame higher than reference
                end
            end
        end %End shift check
    else %Else names don't match
        vindex = vindex+1;  %Displace current Index
        if vindex > length(vitemotivs)
            vindex = 1; %Prevent out of index array accessing
            lindex = lindex+1; %Shift to compare next leitmotiv
        else %Not out of bounds, next value
            currsTime = vitetimes(vindex,1);
            curreTime = vitetimes(vindex,2);
        end
    end %End if names match
    %While loop release lock
    if lindex >= length(leitmotivs)
        alphaLock = 1;
    end
    %Increment control value
    control = control+1;
end
%Number of Substitutions detection
lindex = 0;
leitrefst = leittimes(1,1);
leitrefend = leittimes(1,2);
leitName = leitmotivs{1};
for k=2:length(leitmotivs)
    lindex = lindex+1;
    for l=1:length(vitemotivs)
        currsTime = vitetimes(l,1);
        curreTime = vitetimes(l,2);
        currentDuration = curreTime - currsTime;
        leitDuration = leitrefend - leitrefst;
        leitMid = leitrefst + leitDuration/2;
        shift = 5000;
        leitName2 = vitemotivs{l};
        backref = leitrefst-shift;
        if (currsTime >= backref) && (currsTime <= leitMid)
            modelisaudio = strcmp(leitName,'audio');
            if modelisaudio ~= 1
                namecheck = strcmp(leitName2,leitName);
                if namecheck ~=1
                    %Mark Substitutions
                    subs = subs+1; %Same time frame, different name
                    subVMask(l) = 1;
                    subLMask(lindex) = 1;
                end
            else
```

```
                namecheck = strcmp(leitName2,leitName);
                if namecheck ~=1
                    %Mark Substitutions
                    subVMask(l) = 2;
                    subLMask(lindex) = 2;
                end

            end
        end
        if currsTime >= leitrefend
            leitrefst = leittimes(k,1);
            leitrefend = leittimes(k,2);
            leitName = leitmotivs{k};
            break; %Go to next outer loop iteration
        end
    end
end
%Number of insertions detection
for i=1:length(viteMask)
    if viteMask(i) ~= 0 || subVMask(i) ~= 0
        insVMask(i) = 0;
    else
        insVMask(i) = 1;
    end
end
%Number of deletions detection
for i=1:length(leitMask)
    if leitMask(i) ~= 0 || subLMask(i) ~= 0
        delLMask(i) = 0;
    else
        delLMask(i) = 1;
    end
end
insref = cumsum(insVMask);
delref = cumsum(delLMask);
ins = insref(length(insref));
dels = delref(length(delref));
precision = hits/(hits+subs+ins); %tp/tp+(fp)
recall = hits/(hits+subs+dels); %tp/tp+(fn)
F1 = 2*precision*recall/(precision+recall);
disp('Search completed');
% finms = ['Penalty Value: ' testValues{1}];
% disp(finms);
% finms = ['Score Value: ' testValues{2}];
% disp(finms);
finms = ['Total number of Hits: ' num2str(hits)];
disp(finms);
finms = ['Total number of Substitutions: ' num2str(subs)];
disp(finms);
finms = ['Total number of Insertions: ' num2str(ins)];
disp(finms);
finms = ['Total number of Deletions: ' num2str(dels)];
disp(finms);
finms = ['Precision: ' num2str(precision*100) '%'];
disp(finms);
finms = ['Recall: ' num2str(recall*100) '%'];
disp(finms);
finms = ['F1 Score: ' num2str(F1)];
```

```
disp(finms);
finms = ['Length vitemotivs: ' num2str(length(vitemotivs))];
disp(finms);
finms = ['Sum vitemotivs: ' num2str(hits+subs+ins)];
disp(finms);
finms = ['Length leitmotivs: ' num2str(length(leitmotivs))];
disp(finms);
finms = ['Sum leitmotivs: ' num2str(hits+subs+dels)];
disp(finms);
finms = ['penalty: ' num2str(testValues{1})];
disp(finms);
finms = ['scaling: ' num2str(testValues{2})];
disp(finms);
%Change array type
for i = 1:length(testValues);
    testValuesN(i) = str2double(testValues{i});
end

%% Plot results
limit = size(hits);
code = zeros(hits,2);
code2 = zeros(length(leitmotivs),2);
%Small cycles to transfer data for plotting
lsum = 0;
for i=1:length(leitmotivs)
    modelisaudio = strcmp(leitmotivs{i},'audio');
    if modelisaudio ~=1
        lsum = lsum+1;
    end
end
lnames = cell(lsum,1);
vnames = cell(hits,1);
lstarts = zeros(lsum*2,1);
vstarts = zeros(hits*2,1);
a=1;
b=1;
for i=1:length(vitemotivs)
    if viteMask(i) == 1
        vnames{a} = vitemotivs{i};
        vstarts(b) = (vitetimes(i,1)*100*(10^-9));
        vstarts(b+1) = (vitetimes(i,2)*100*(10^-9));
        a=a+1;
        b=b+2;
    end
end
a=1; %Restart counter
b=1;
for i=1:length(leitmotivs)
    modelisaudio = strcmp(leitmotivs{i},'audio');
    if modelisaudio ~=1
        lnames{a} = leitmotivs{i};
        lstarts(b) = (leittimes(i,1)*100*(10^-9));
        lstarts(b+1) = (leittimes(i,2)*100*(10^-9));
        a=a+1;
        b=b+2;
    end
end
%Cycle for data preparation
```

```
for i = 1:hits %length(vitemotivs)
    c1 = strcmp(vnames{i},'grubel');
    c2 = strcmp(vnames{i},'nibelungen');
    c3 = strcmp(vnames{i},'ring');
    c4 = strcmp(vnames{i},'vertrags');
    if c1 == 1
        code(i,:) = 4.05;
    end
    if c2 == 1
        code(i,:) = 3.05;
    end
    if c3 == 1
        code(i,:) = 2.05;
    end
    if c4 == 1
        code(i,:) = 1.05;
    end
end
for i = 1:length(lnames)
    c1 = strcmp(lnames{i},'grubel');
    c2 = strcmp(lnames{i},'nibelungen');
    c3 = strcmp(lnames{i},'ring');
    c4 = strcmp(lnames{i},'vertrags');
    if c1 == 1
        code2(i,:) = 4;
    end
    if c2 == 1
        code2(i,:) = 3;
    end
    if c3 == 1
        code2(i,:) = 2;
    end
    if c4 == 1
        code2(i,:) = 1;
    end
end
k = 1;
j = 1;
for i = 1:2:length(lstarts*2)
    if j <= hits*2
        g1 = plot(vstarts(i:i+1),[code(k) code(k)],'r');
        set(g1(1),'LineWidth',1.5);
        hold on;
        j = j+2;
    end
    %if i<length(lstarts)
    g2 = plot(lstarts(i:i+1),[code2(k) code2(k)],'b');
    set(g2(1),'LineWidth',1.5);
    %end
    hold on;
    k=k+1;
    xlim([0 600]);
    ylim([0 5]);
    axis xy;
    xlabel('\bft_{seconds}');
    ylabel('\bfLeitMotiv');
    set(gca,'XTick',[50 100 150 200 250 300 350 400 450 500 550 600]);
    set(gca,'YTick',[1 2 3 4]);
    set(gca,'YTickLabel',{'Vertrags','Ring','Nibelungen','Grubel'});
    title('\bf\fontsize{16}Recognition test results');
    legend('Detected','Original');
    mText = uicontrol('style','text');
    TexPos = get(mText,'Position');
    NText = [TexPos(1)/2 TexPos(2) TexPos(3) TexPos(4)];
    set(mText,'Position',NText);
    boxVal = ['P: ' testValues{1} ' ' 'S: ' testValues{2}];
    set(mText,'String',boxVal);
end
lmMix = '8';
imageName = ['LM_' lmMix '_' testValues{4} '_A_' testValues{3} '_'
testValues{5} '_ps' testValues{1} testValues{2}];
imagePath = [rootfolder '\newTestPlots\' imageName];
print(imagePath,'-dpng');
%% Final writings
checkfile = exist([parHMM.dir_data 'MatrixData_New.txt'],'file'); %Check if
file exists
if checkfile ~= 2
    logfile = fopen(fullfile(parHMM.dir_data,'MatrixData_New.txt'), 'wt' );
%create and write
    message = ['H','     ', 'S', '     ', 'I','     ','D','     ',...
        'Prec','            ',...
        'Rec','              ',...
        'F','                ','p','      ',...
        's','        ','AS','     ', 'AMIX','     ', 'LS','     ',...
        'LMIX\n'];
    fprintf(logfile,message);
else
    logfile = fopen(fullfile(parHMM.dir_data,'MatrixData_New.txt'), 'at' );
%append to existing data on file
end
fprintf(logfile,'%s     %s    %s    %s   %s     %s     %s     %s       %s
%s       %s       %s     %s\n',...
    num2str(hits),num2str(subs),num2str(ins),num2str(dels),...
    precision,recall,F1,testValues{1},testValues{2},...
    testValues{5},testValues{3},testValues{4},lmMix);

%% Close files
hmmtest = [parHMM.dir_data 'HMMtest.mat'];
datatest = [parHMM.dir_data 'DataTest.txt'];
save(hmmtest,'precision','recall','F1','testValuesN','-append');
save(datatest,'precision','recall','F1','testValuesN','-append','-ascii');
fclose(viteID);
fclose(leitID);
fclose(testParsID);
fclose(logfile);
disp('Closing figure...');
close all hidden;
disp('Results completed');
```