EMPIRICAL MODE DECOMPOSITION APPLIED TO SPEAKER IDENTIFICATION

KEVIN BARTUSKA
SPRING 2016

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Electrical Engineering
with honors in Electrical Engineering

Reviewed and approved* by the following:

John Doherty
Professor of Electrical Engineering
Thesis Supervisor

Jeffrey Mayer
Associate Professor of Electrical Engineering
Honors Adviser

* Signatures are on file in the Schreyer Honors College.
ABSTRACT

This project applied the algorithm “Empirical Mode Decomposition” (EMD) to the problem of speaker identification – that is, recognizing a speaker by their unique tone of voice. A Gaussian mixture model was used to compare the mel frequency cepstral coefficients (MFCCs) of the speech files, after their processing with EMD. This experimenter determined that only the first several EMD modes contained the relevant information for speaker identification. Additionally, EMD was combined with singular value decomposition, but the benefits to recognition rate were not significant.
# TABLE OF CONTENTS

LIST OF FIGURES ........................................................................................................ iii

LIST OF TABLES ........................................................................................................ iv

Chapter 1 Introduction ................................................................................................. 1
   Motivation ................................................................................................................ 1
   Problem Solving Approach ...................................................................................... 2

Chapter 2 Signal Processing Tools .............................................................................. 4
   Empirical Mode Decomposition (EMD) ................................................................. 4
   Gaussian Mixture Models (GMM) ........................................................................... 6
   Mel Frequency Cepstral Coefficients (MFCCs) ....................................................... 7
   Preprocessing and Removing Silence .................................................................... 8
   Singular value Decomposition (SVD) ..................................................................... 8

Chapter 3 Methodology ............................................................................................... 10
   Speech Signals ....................................................................................................... 10
   General Setup ...................................................................................................... 10
   Using EMD as a Filter ........................................................................................... 12
   EMD applied to several IMFs ................................................................................ 12
   Using EMD and SVD ............................................................................................ 13

Chapter 4 Experimental Results ............................................................................... 15
   Using EMD as a Filter ........................................................................................... 15
   EMD applied to several IMFs ................................................................................ 19
   Using EMD and SVD ............................................................................................ 21

Chapter 5 Future Work and Conclusions ................................................................. 26
   Future Work ......................................................................................................... 26
   Conclusions ......................................................................................................... 26

Appendix A Implementation of EMD ....................................................................... 29

Appendix B Script for Removing Silences ............................................................... 37

Appendix C Script for Performing Tests in 4-1 and 4-2 .......................................... 39

Appendix D Script for Performing Tests in 4-3 ....................................................... 45
LIST OF FIGURES

Figure 1: General Testing Process .................................................................3
Figure 2: EMD used as a Lowpass filter .........................................................16
Figure 3: Testing with only one IMF ...............................................................16
Figure 4: Testing without the first IMF ............................................................17
Figure 5: Distribution of IMFs for all 232 Signals for qResol=30 dB ..................18
Figure 6: Finding MFCCs of Multiple IMFs .....................................................19
Figure 7: Comparison of methods for combining multiple IMFs .....................20
Figure 8: Entire Signal, Unweighted ...............................................................23
Figure 9: Each IMF, Unweighted .................................................................23
Figure 10: Entire Signal, Weighted ..............................................................24
Figure 11: Each IMF, Weighted .................................................................24
LIST OF TABLES

Table 1: Listing of Typical Numerical Parameters ................................................................. 11

Table 2: Selection of Results for EMD used as a Filter .......................................................... 17

Table 3: Summary of SVD Recognition Rates ........................................................................ 22
Chapter 1

Introduction

Every human possesses a unique vocal signature that is evident in all of his or her speech. Speaker recognition (or speaker identification) attempts to take advantage of this fact to match pairs of recordings that were spoken by the same person. This paper is concerned with text-independent speaker recognition, where the speaker uses completely unrelated words and phrases and must still be identified as a single speaker [1]. Theoretically, their unique personal intonation should allow the recordings to be matched correctly. Speaker identification has potential applications in many situations including access control and law enforcement [1].

Empirical Mode Decomposition (EMD) is a signal processing method that may be of some use in this domain. This paper attempts to assess its usefulness in a variety of ways. EMD is a natural way of breaking nonstationary signals into several “Intrinsic Mode Functions” (IMFs) in the time domain. It is used primarily as a filtering method prior to feature extraction and processing.

Motivation

Some experiments have already been conducted using EMD for speaker identification [2, 3]. It has also been applied to related problems, such as emotion classification [4]. This paper will work with those ideas and apply them to a much larger dataset, in addition to conducting some original tests. In [2], the notion of computing mel frequency cepstral coefficients (MFCCs)
for individual IMFs is introduced, and it is reapplied here. [3] introduced singular value decomposition as a way to collect and use the principle features of the voice signals. However, there are significant differences between those papers and the methods employed here. MFCCs are used instead of LPCCs, as in [3]. That paper also broke the signal into subbands before computing LPCC coefficients, whereas in this paper we find MFCCs on the level of individual IMFs. In addition, experiments in [2] vary the SNR of the speech signals, apply higher order statistics (HOS), and use a neural net in place of a GMM. In [2] delta MFCCs support vector machine are used in tests. These steps are not replicated here.

Although the tests performed here are somewhat simpler, they are based on a significantly larger set of data. This paper used 232 total recordings of 116 speakers. This is a significant increase over 30 speakers in [2] and 40 speakers in [3].

**Problem Solving Approach**

The testing process begins with a set of 232 recordings of human speech. Each of 116 individuals contributed two recordings. The set of individuals included both men and women. One recording from each subject is placed into a first group called the “training set”. The other 116 recordings become the “testing set”. Both the training and testing set signals are pre-processed in exactly the same fashion. Then the training set signals are used to build a Gaussian Mixture Model (GMM) and the testing set signals are compared to this model. The program outputs the closest match in the training set to each testing set signal. Note that the testing program assumes that each testing set signal does have a match.
A variety of pre-processing techniques are employed prior to building the GMM. First, silences are removed and all the signals are shortened to ten seconds in length. Then the signals are broken down into a series of intrinsic mode functions (IMFs or modes) using empirical mode decomposition. The features that are extracted from the signals are MFCCs. In some cases, the signals that are examined are a subset of the IMFs. In some other cases they are the sum of a subset of IMFs. For some tests the MFCCs of the entire speech signals are computed. Lastly, singular value decomposition is examined. This signal processing tool extracts principle component vectors from the MFCC matrices. These vectors are a useful representation of the information in the mel coefficients. All of these tools will be introduced in the next section. In these experiments, all processing is done using MATLAB.

![General Testing Process](image)

Figure 1: General Testing Process

The primary measure of performance was recognition rate. The result of comparing the training and testing sets is 116 pairs of signals. One signal in each pair is from each set. Either the two signals in a given pair have the same speaker or they do not. The recognition rate is the percentage of the time the speaker identification is correct i.e. the two signals have the same speaker.
Chapter 2

Signal Processing Tools

Empirical Mode Decomposition (EMD)

EMD is a signal processing method for breaking down signals into constituent parts. It is “empirical” in the sense that it can be applied directly to non-stationary signals and is entirely data-driven. The signal is decomposed into a number of “intrinsic mode functions” or IMFs. Ideal IMFs have three properties – they have the same number of zeros as local extrema or local extrema ±1, the average of their envelopes is always zero, and the sum of the modes is equal to the original signal.

EMD is performed on a signal using the following steps:

1) Identify all the local minima and maxima. Define an upper (and lower) envelope using these points and an interpolating function (here a cubic spline).
2) Average the envelopes and subtract this average from the signal.
3) If this residue is not a mode, repeat Steps 1 and 2
4) If this residue is a mode, then subtract this mode from the original signal and repeat Steps 1 through 3 to find the next mode.
5) Continuing sifting for modes. The final mode is either a constant or trend.

Because of the empirical nature of EMD, there are a number of computational details to examine. The specific implementation of EMD used in this paper can be found in [5] and in Appendix B. For example, local extrema are found by the following process: a data point that is
an extrema is identified. A parabolic interpolation through the points on either side of this extrema is found. The true extrema used for forming the envelopes is the vertex of this parabola.

The most important implementation detail is the stopping criteria. In practice, theoretically perfect IMFs are not found. Instead, the program sifts until the candidate IMF is within some acceptable error of the ideal. Here that error is defined in terms of energy ratios in decibels. Two parameters, called qResid and qResol in the code are used for this purpose. qResid controls the degree to which the program sifts when calculating IMFs, qResol controls the degree to which the program sifts on the entire signal before stopping. In both cases, sifting stops when the ratio grows above the allowed resolution.

\[
q_{\text{Resid}} = \frac{\text{Energy of the signal at the start of sifting}}{\text{Energy of envelope average}}
\]

\[
q_{\text{Resol}} = \frac{\text{Energy of the original signal}}{\text{Energy of the residue IMF}}
\]

Larger values for qResid and qResol tend to require more processing while smaller ones can sacrifice recognition accuracy. In our tests qResol was set to 40 dB, and qResid was typically set to 30 dB. These values seemed to strike a balance between the two extremes, though with more computational resources and longer processing times it may be fruitful to test with higher qResol settings.

The chief advantage of EMD is that no assumptions about the signal need to be made. In particular, the signals do not need to be stationary. This is fortunate as speech signals are non-stationary by their nature. Each phoneme or unit of sound will have its own Fourier frequency
spectra, so any spoken sentence will have a Fourier transform that changes with time, so the signal is non-stationary.

One theoretical problem with EMD is that the number of modes that a signal will decompose into is not knowable in advance and can vary significantly. It is dependent on the particular implementation of EMD being used, on the parameter settings, and of course on the signal in question. It was found in these experiments that the dominant modes are the first six. Modes beyond the first six do not contain much important information for the purposes of speaker identification, and so these higher order modes are often discarded.

The EMD algorithm can be thought of as an averaging method, the lower frequency envelopes are repeatedly removed, leaving only the higher frequency components. The result is that the first several modes contain the highest frequencies while the higher order modes contain lower frequencies. Moreover, the number of extrema and magnitudes of the first modes tend to be larger.

**Gaussian Mixture Models (GMM)**

Gaussian Mixture Models are used as the method for comparing the speech signals. Once features are extracted from the training set signals, a GMM is constructed. These experiments used 16 clusters. Each testing set signal is then compared to the model, and assigned a most likely speaker. Each speaker is modeled with a GMM, which represents the features of the signal as a mixture of normal distributions (Gaussians distributions). GMMs have been a standard way to represent speech data for some time, and their use and computation is highlighted and motivated in [6]. One interpretation of GMMs that hints at their usefulness is
that each Gaussian cluster represents a general feature of a person’s tone or intonation, and the combination of all of them is all but unique to that person [6].

**Mel Frequency Cepstral Coefficients (MFCCs)**

Mel Frequency Cepstral Coefficients are used as the features for the Gaussian Mixture Model. The motivation behind the MFCC method is the human ear. Humans perceive sound logarithmically, both in frequency and in intensity. The frequency spectrum is divided up logarithmically, and the logarithm of the energy in each bin is taken as well. MFCCs are computed with the following steps:

1) Divide the signal into short frames. For these experiments frames are 128 samples. Since all signals have a sampling frequency of 8000 Hz, this equates to frame lengths of 16ms. 8 ms of each frame is overlapping. The remaining steps are performed for each frame of the signal.

2) The frequency spectra is divided logarithmically into bins, and a periodogram estimate of the spectral density in each bin is calculated.

3) The coefficients are the DCT of the log of these energies.

There are three numerical parameters associated with this step. The first two parameters control the length of the frames and the amount which those frames overlap. The former is set to 128 samples and the latter to 64 samples. Since all the speech signals have a sampling rate of 8000Hz, these frames are 16ms in length. The third parameter controls how many bins the frequency spectra is divided into, and how many coefficients are ultimately computed. This is set to 20 in our experiments. However, the first coefficient is discarded,
leaving only 19. This is typical when working with mel coefficients. The 19 remaining coefficients are used as the features for our speech signals.

**Preprocessing and Removing Silence**

A MATLAB script was used to remove silences from speech files prior to processing. It can be found in Appendix B. All signals were shortened to a length of ten seconds using this MATLAB script. This was done prior to any other processing and is the first step. The program works by examining frames of the original signal, and removing any with energy below a certain threshold.

There are three numerical parameters associated with this step in processing. The volume threshold determines what constitutes a silence frame and a voiced frame. This took on a value of 0.02 V. The number of samples per frame controls how lengthy the silences can be before they are removed, for us it was 256 samples. At a sample rate of 8000Hz, this translates to 32ms. All silences shorter than 32ms are removed. Finally, the signal length controls how many seconds the post-processed signal will be. This was set to 10 seconds.

**Singular value Decomposition (SVD)**

Singular value Decomposition is a method for selecting the dominant features of signals. Before using SVD, the MFCCs are found for a given signal or IMF. The matrix of coefficients has $q$ rows and $J$ columns. $q$ is the number of mel coefficients (here, 19) and $J$ is the number of signal frames (1294). Then, SVD is a decomposition of the following form:

$$M_{q \times J} = U_{q \times q} A_{q \times J} V_{J \times J}^T$$
$U$ and $V$ are both orthogonal square matrices, and $A$ is a matrix with zeros on all entries except those along the main diagonal. The columns of $U$ are the dominant features that are of interest here. In practice, only some of the dominant features are needed, and so the last columns of $U$ are discarded prior to building the GMM and attempting to match signals.

For some experiments the principal feature vectors were weighted according to the following equation:

$$w_i = \frac{\sigma_i}{\sqrt{\sum (\sigma_j)^2}}$$

Where $w_i$ is the weight applied to the $i$th principal vector, and $\sigma_i = \sigma_{ii}$ is the singular value in the $A$ matrix. In practice, the first feature vector was larger than all other entries of $A$ by an order of magnitude, so the scaling affected the principal features. This weighting scheme was directly motivated by [3]. There are no user-set numerical parameters associated with SVD, and the built in MATLAB function was used.
Chapter 3
Methodology

Speech Signals

A set of 232 speech recordings were used to perform these experiments. 116 males and females each submitted two recordings. The original recordings vary in length but are several minutes of phone conversations. A signal from each speaker was used in the training set. The other 116 signals formed the testing set.

The signals were obtained from the switchboard 1 (release 2) collection of NIST telephone conversations. They are maintained by the Linguistic Data Consortium [7]. They are an excellent set of clean signals with an SNR of approximately 20 dB.

General Setup

The first step of the procedure was to shorten the training and testing set signals to 10 seconds and remove silences. This serves two purposes. It shortens them to a manageable length for EMD processing and removing superfluous silences improves the recognition accuracy.

Then, signals were broken down into their IMFs via the EMD algorithm. On the typical setting of qResol = 40 dB and qResid = 30 dB, the signals decomposed into approximately 13 modes on average. This step was the longest and most computationally expensive one.
Then additional modifications were performed on all of the signals. The experiments fell into one of three categories.

1) EMD was used as a filter. Some modes were subtracted away from the signals. MFCCs were found for each filtered signal.

2) EMD was used and MFCCs were found for each mode.

3) EMD was used and MFCCs were found for each mode. Then SVD was used to extract the principal feature vectors from each signal.

Since all of the processing was conducted using MATLAB scripts, there were many parameters which were chosen. A summary of the parameter choices used is in Table 1. This table also includes their usual value, which was taken for all tests unless otherwise mentioned.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Associated Step</th>
<th>Typical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Samples per frame</td>
<td>silence removal</td>
<td>256 samples</td>
</tr>
<tr>
<td>Signal Length</td>
<td>silence removal</td>
<td>10 seconds</td>
</tr>
<tr>
<td>Volume threshold</td>
<td>silence removal</td>
<td>0.02 volts</td>
</tr>
<tr>
<td>Residual threshold (qResid)</td>
<td>EMD</td>
<td>40 dB</td>
</tr>
<tr>
<td>Resolution threshold (qResol)</td>
<td>EMD</td>
<td>30 dB</td>
</tr>
<tr>
<td>Samples per frame</td>
<td>MFCC</td>
<td>128 samples</td>
</tr>
<tr>
<td>Samples of overlap</td>
<td>MFCC</td>
<td>64 samples</td>
</tr>
<tr>
<td>Number of mel coefficients</td>
<td>MFCC</td>
<td>20 coefficients</td>
</tr>
<tr>
<td>Number of GMM clusters</td>
<td>GMM</td>
<td>16 clusters</td>
</tr>
</tbody>
</table>

Table 1: Listing of Typical Numerical Parameters
Using EMD as a Filter

For these experiments a subset of the signal modes was selected and recombined. Typically the first N modes were used. In this case EMD is functioning as a high pass filter. If a particular mode (e.g. the second) mode was used, then EMD was functioning as a band pass filter. EMD was also used as a bandpass filter for a group of intermediate modes (e.g. modes two through six). Once a subset of modes was chosen, then all but those modes were removed from training and testing signals. Alternatively, only those selected modes were added together to form new training and testing set signals.

Once appropriately modified, the MFCCs of the training set signals were computed. These MFCCs were used to build a GMM, and comparisons were run. It is also possible to run a control test by selected all of the available modes. Then the experiment simplifies to finding the MFCCs of the entire signal, and effectively EMD was not used.

EMD applied to several IMFs

For these experiments the signals were decomposed with EMD on the usual settings: qResol = 40 dB and qResid = 30 dB. The first group of experiments showed that the higher order modes do not have a strong effect on recognition rate, so for these experiments only the initial modes were used, and the last modes were discarded. The signal that decomposed into the fewest IMFs was used completely. Any modes beyond this minimum number were discarded for other signals. This is justified by the results of the first set of experiments, which showed that those modes did not have much impact on the recognition rates of the GMM.
The MFCCs were then computed for each IMF under consideration for each signal. For all the tests in this section, only the first several modes from each signal were included. The feature matrix used to build and test the GMM had the following form: Each column was associated with an MFCC coefficient, while each row was associated with a particular frame of an IMF under consideration. In practice if \( N \) IMFs were included in a test, the matrix was dimension \( 1249N \times 19 \). The first IMF was listed in the first 1249 rows, the second IMF in the next 1249, the third (if present) in the next 1249 rows. 1249 was the number of frames obtained from each 10-second sound clip.

**Using EMD and SVD**

Signals in these experiments were broken into their IMFs using the EMD algorithm in the typical way. The signal that decomposed into the fewest IMFs was used completely. Any modes beyond this minimum number were discarded for other signals. This is justified by the results of the first set of experiments, which showed that those modes did not have much impact on the recognition rates of the GMM.

Mel coefficients were found for each of the modes under consideration for each signal. This meant that each mode was represented by a matrix \( M \), which had dimensions \( q \times J \), where \( q \) was the number of mel coefficients and \( J \) the number of signal frames. Here \( q \) was 19. Then, SVD was used to find a matrix \( U \) of dominant features. The dimension of \( U \) was \( q \times q \). In previous work, the first four dominant features were found to accurately represent the entire feature space [3], so for some runs only the first four columns of \( U \) were used, and the rest discarded. In other tests a different number of columns was included.
The data structure used to build the GMM can now be described. If the first $k$ dominant features (columns) of $U$ are included, then a new vector $G$ for each speech signal is constructed. $G$ is formed by concatenating columns of $U$ for different IMFs. Therefore, the first $k$ columns of $G$ are the first $k$ columns of the matrix $U$ resulting from the first IMF of a particular signal. The next $k$ columns of $G$ are the first $k$ columns of the matrix $U$ resulting from the second IMF of that signal. And so on for all the IMFs that are being considered. This collection of dominant features across several IMFs was presented to the GMM as a set of feature vectors.

In some of the tests, the columns of $U$ were weighted according to the following relationship:

$$w_i = \frac{\sigma_i}{\sqrt{\sum (\sigma_j)^2}}$$

Where $\sigma_i$ is the singular value associated with a particular principle component vector.
Chapter 4
Experimental Results

Using EMD as a Filter

For this first set of tests, signals were broken into their constituent modes via EMD. Then a subset of those modes were selected, and the rest were discarded. The selected modes were added together to produce a filtered version of each speech signal. The MFCCs of this modified signal were computed and then tested.

The tests fell into three main categories. For the first set, EMD was used as a lowpass filter. All modes past a certain number were discarded, and only the first N were kept. For the second set, only one intermediate IMF was used to generate the results. The third dataset emphasizes the importance of the first IMF by removing it.

In addition, these experiments were conducted with two different settings of the stopping parameter qResol. qResol controls the point when the program stops sifting for modes and declares the last IMF to be a constant or trendline. A larger value implies sifting to a higher degree of detail, while a smaller one implies that the program is more lax in its definition of a “trend”. The implementation of EMD used recommended 40 dB as a minimum value for this parameter, but the following tests show that 30 dB is acceptable, as the results are nearly identical across all tests.
Figure 2: EMD used as a Lowpass filter

Figure 3: Testing with only one IMF
Figure 4: Testing without the first IMF

The overall baseline recognition rate is approximately 90%. This is in line with what is expected for clean speech signals. Because the signals are so clean, many of the other results are significantly lower. This makes sense, because so many of the other tests are asking the GMM to perform the same task but with less information.
The first set of tests show that recognition rates with only the first IMF are around 68%, but rise quickly as additional modes are added. By the time one considers the first six modes, it appears to be functionally equivalent to examining the entire signal. All of the information needed for speaker identification appears to be contained in the first six modes. This is surprising, as the average number of modes on the qResol=30 dB setting is 14.3 modes. The frequency distribution of IMF count is shown in figure 5. Keeping only the first six IMFs implies discarding over two-thirds of the total number of modes for some signals, yet it causes little change in the ability of the GMM to match speakers.

![Distribution of the Number of IMFs]

**Figure 5: Distribution of IMFs for all 232 Signals for qResol=30 dB**

What the tests show in figures 3 and 4 is that the first IMF is of greatest importance. Remaining IMFs may still contribute, but less so. The third set of tests also shows that the first mode is extremely important; removing it causes the recognition rate to decrease significantly.
EMD applied to several IMFs

Four separate runs were completed in this section. In each case, the first N modes of each speech file were kept, and the rest discarded. The MFCCs of each IMF were completed and presented to the GMM. The results are summarized in Figure 7, which compares the results of this section to those in the previous section. It is clear from this figure that finding the MFCCs of a sum of modes is more effective than finding the MFCCs of each mode individually.

![Figure 6: Finding MFCCs of Multiple IMFs](image-url)
Considering a larger and larger number of IMFs will likely improve recognition rates with this method. There is a very large jump in improvement when moving from only one IMF to considering the first two. However, it seems that adding further IMFs beyond this point is not helpful. This is in contrast to the previous section when adding further IMFs was beneficial until approximately mode four or five.

Partially because this method saturates so much sooner, it lags behind its counterpart in performance. These data show that mode for mode it makes more sense to consider the MFCCs of the sum of all IMFs than to compute MFCCs of each IMF individually. The likely cause is the greater importance of the first mode in providing important information for recognition. In this scheme all modes are considered equally important when the GMM is constructed, but in practice they are not so. A method for weighting the modes appropriately may lead to better
results in this section. A different attempt at weighting the results is considered in the next section.

**Using EMD and SVD**

SVD is a method for extracting feature vectors from the MFCCs. These vectors are designed to accurately represent the feature space of mel coefficients. Four distinct tests were conducted here, and for each of them a different number of principal feature vectors was considered. In the first experiments, the entire speech signal was processed into its mel coefficients. Then this matrix of coefficients was decomposed using singular value decomposition, and the principal feature vectors were used to train and test the signals.

A second set of tests was conducted using MFCCs found from each of the first nine IMFs. Nine IMFs were the fewest produced from the EMD decomposition of all training and testing signals, and remaining higher order modes were discarded. Based on the results from “Using EMD as a Filter”, these very high order modes can be safely discarded without influencing the results. This step was performed so that each signal would have the same number of IMFs under consideration, and would be the same “size” for the GMM.

The MFCC matrix of each IMF was broken into its principal vectors using SVD, and all but the first four to six principle vectors were discarded (with the exact number varying by experiment). Then, the GMM was built and trained using a concatenated matrix of the principal vectors of all of signal’s IMFs.

A third set of tests was run computing the MFCCs of entire speech signals, but then weighting the principal vectors to emphasize the dominate ones. A final set of tests was
performed by again taking the MFCCs of each mode, and weighting the relevant principal vectors. The weighted vectors of each IMF were the combined exactly in the same way as the second group of tests.

For each suite of tests the GMM was built and trained using only the first principal vector, using the first two vectors, the first three, first four, and first five. The results are summarized in table 3 and examined more closely in the following figures.

<table>
<thead>
<tr>
<th>Number of PCs Kept</th>
<th>Unweighted</th>
<th></th>
<th>Weighted</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Entire Signal</td>
<td>Each IMF (9)</td>
<td>Entire Signal</td>
<td>Each IMF (9)</td>
</tr>
<tr>
<td>1</td>
<td>4.3%</td>
<td>26%</td>
<td>4.3%</td>
<td>26%</td>
</tr>
<tr>
<td>2</td>
<td>0.86%</td>
<td>21%</td>
<td>0.86%</td>
<td>26%</td>
</tr>
<tr>
<td>3</td>
<td>1.7%</td>
<td>9.5%</td>
<td>4.3%</td>
<td>22%</td>
</tr>
<tr>
<td>4</td>
<td>16%</td>
<td>8.6%</td>
<td>20%</td>
<td>24%</td>
</tr>
<tr>
<td>5</td>
<td>22%</td>
<td>5.2%</td>
<td>29%</td>
<td>23%</td>
</tr>
</tbody>
</table>

Table 3: Summary of SVD Recognition Rates
Figure 8: Entire Signal, Unweighted

Figure 9: Each IMF, Unweighted
Figure 10: Entire Signal, Weighted

Recognition Rate using SVD on Entire Signal, Weighted

Recognition Rate using SVD on Each IMF, Weighted

Figure 11: Each IMF, Weighted
In all cases, the performance using SVD was significantly worse than the recognition rates using the raw MFCC data. The default rate from simply computing MFCCs of each signal was 90%, and the best results from this section are around 30% accuracy. It is possible SVD will still be useful for signals that are noisier, but it is clear that for clean signals this technique is inferior to the previous approaches.

One other point of interest is the trends as more columns of $U$ are included in the mixture model. If the MFCCs are taken at the level of the entire speech signal, including more components improves performance. However, if the MFCCs are computed for each IMF and then concatenated, then including more component vectors actually decreases performance. In the unweighted case, this can be explained by the fact that the first component is most important. Values of $\sigma$ are typically an order of magnitude larger for the first component than for all subsequent ones. This implies that the first principle component is the dominant one, and is the one which best represents the feature space by a wide margin. Adding in other components without weighting them decreases accuracy, as they are not truly as important as the first vector. Weighting the higher order components reduces this effect, but does not completely eliminate it.
Chapter 5
Future Work and Conclusions

Future Work

Experiments with noisy signals are an avenue that should be explored. Adding even relatively nominal noise can significantly reduce recognition rates. These experiments were unable to improve on the baseline for clean signals, but it is possible that they will improve recognition rates for noisy signals. Fundamentally, all of the processing methods used were discarding information. Modes and feature vectors were thrown away. In clean signals, this information was not harmful, and may have even been beneficial for the speaker identification program. Nevertheless, in signals that are less clean, not all of the signal’s information content will be useful. It is possible that removing some information by discarding IMFs will improve results in that case, because the removed information would be harmful.

Additionally, due to time and computational limitations a full sweep of possible parameter choices for the EMD step was not completed. In particular, exploring higher settings of the resolution setting (qResol) will have a potentially large impact. A higher resolution here would imply all IMFs found are closer to their ideal form. This may improve the tests that compute MFCCs of the modes directly.

Conclusions

The process of finding MFCCs and then using a GMM to perform speaker identification is a robust process on its own. Given a set of high quality signals recognition rates around and
above 90% are the norm. Those baseline rates were observed here as well. Including EMD as a preprocessing step did not improve on this benchmark, but it still a fruitful avenue of research.

The experiments summarized in Figure 2 show that EMD can be used effectively as a filter before performing speaker identification. Discarding everything after just the first four to six modes recovers the baseline recognition rate. Since so many of the IMFs are irrelevant to speaker identification, there should be potential for this filtering to improve recognition rate with noisier signals, if that noise is lower frequency. In large part lower frequency noise and interference should be removed by discarding large numbers of higher order IMFs.

Considering several IMFs improves recognition rate compared to examining the first IMF along. The results shown in Figure 7 imply that it is more fruitful to combine the IMFs under consideration into a new signal prior to extracting features. The alternative of collecting MFCCs for each mode and using that as the set of features is less successful. The hypothesized reason for this is twofold. First, since the first few IMFs already contain enough information to identify speakers, nothing is gained by splitting them apart with EMD. Including information about the nature of the decomposition (by providing the IMFs side by side instead of summed) does not appear to carry enough extra information to be worthwhile. Second, in these experiments all the feature vectors were simply concatenated. The first IMF is clearly the most important one for speaker identification, so considering the features of all IMFs on equal footing seems suboptimal. In theory this could be remedied with a weighting scheme.

Lastly, SVD does not seem useful as a method for improving the recognition rates for clean signals. The combination of SVD and examining the MFCCs of each IMF seems particularly unhelpful, as it deteriorated more as more principle component vectors were included. This is likely true for many of the same reasons that considering each IMF was
unproductive without SVD. Here a weighting scheme was attempted and results were improved, but only marginally. The weighting scheme is not enough to compensate the fact that principle vectors after the first do not contribute much useful information.

In summary EMD appears to be a fruitful method for approaching speaker recognition. It must be combined with other techniques for feature extraction, but the can be used as a high pass filter prior to processing signals. It does not appear to be particularly useful to combine EMD with singular value decomposition, or to evaluate and combine the feature vectors of various modes. However, EMD can be used profitably as a low pass filter and will remove significant portions of a signal without affecting recognition rate.
Appendix A

Implementation of EMD

% This program performs the Empirical Mode Decomposition accordingly to the paper
% “On the HHT, its problems, and some solutions”, Reference: Rato, R. T., Ortigueira, M. D., and Batista, A. G.,
%
% Authors: Raul Rato (rtr@uninova.DOT.pt) and Manuel Ortigueira
% (mdortigueira@uninova.pt or mdo@fct.unl.pt)
%---------------------------------------------------------------

% rParabEmd__L: Emd parabolic decomposition with extrapolated extrema
v1.01
Build
20070717001

% Usage: rParabEmd = rParabEmd__L(x,qResol, qResid, qAlfa);
% x - input signal - must be a real vector
% qResol - Resolution (in DBs: 10*log(WSignal/Bias energy))- normally between 40 and 60 dB
% qResid - Residual energy (in DBs: 10*log (WSignal/WqResidual))- normally between 40 and 60 dB
% qAlfa - Gradient step size (normally is set to 1)
% rParabEmd - relation matrix of IMF modes (each as a line) with residual in last line.
%
% Limitations: NaN is not trapped
%
% History: V1.00 First version
% V1.01 Count mismatch detection (Line 44) increased from 1 to 2
%
% WARNING: This software is a result of our research work and is supplied without any garanties.
% We would like to receive comments on the results and report on bugs.
%
% /* NoSPAM: Replace .DOT. with a dot (.) */
% (c) LaPAS-2007

function rParabEmd = rParabEmd__L (x, qResol, qResid, qAlfa)

dbstop if warning
if(nargin~=4), error('rParabEmd__L: Use with 4 inputs.'), end
if(nargout>1), error('rParabEmd__L: Use with just one output.'), end
ArgCheck_s(x, qResol, qResid, qAlfa)

% Actual computation ---------------------------------------------
kc = x(:); % ket copy of the input signal
Wx= kc'*kc; % Original signal energy
quntN = length(kc); % Signal length
% loop to decompose the input signal into successive IMFs
rParabEmd= []; % Matrix which will contain the successive IMFs, and the residue
rParabEmdCnt= 0;
qDbResid= 0; %Equal energies at start
quntOscCnt= quntNOsc_s(kc);
while ((qDbResid<qResid) && (quntOscCnt>2) ) % c has some energy and oscillates
    kImf = kc; % at the beginning of the sifting process, kImf is the signal
    rPMOri= rGetPMx_s(kImf); % rPM= [xM(M), yM(M)];
    rPmOri= rGetPMin_s(kImf); % rPm= [xm(m), ym(m)];
    rPM= rMaxExtrapol_s(rPMOri, rPmOri, quntN);
    rPm= rMinExtrapol_s(rPMOri, rPmOri, quntN);
    quntLM= length(rPM); quntLm= length(rPm);
    if (abs(quntLM-quntLm)>2), disp('Debug: Max-Min count mismatch.'),keyboard,end;
    if (sum(abs(diff(sign(rPM(1:min(quntLM,quntLm),1)-rPm(1:min(quntLM,quntLm),1))))>0)
        disp('Debug: Max-Min sequence mismatch.'),keyboard;
        disp('Debug: Max-Min sequence mismatch.');
    end
    if (sum(abs(diff(sign(rPm(1:min(quntLM,quntLm),1)-rPM(1:min(quntLM,quntLm),1))))>0)
        disp('Debug: Max-Min reverse sequence mismatch.'),keyboard;
        disp('Debug: Max-Min reverse sequence mismatch.');
    end
    bTenv= spline(rPM(:,1), rPM(:,2), 1:quntN); % Top envelop:
bTenv[n];
    bDenv= spline(rPm(:,1), rPm(:,2), 1:quntN); % Down envelop:
bDenv[n];
    bBias= (bTenv+bDenv)/2; % first bias estimate
    true(1) % inner loop to find each IMF
    WImf= kImf'*kImf; %current IMF energy
    WBias= bBias*bBias'; %bias energy
    if WBias*WImf<0 , warning('rParabEmd__L: Ooops, negative energy detected.'), end
    if WBias> 0, DbqResol= 10*log10(WImf/WBias); else DbqResol= Inf; end
    if (DbqResol>qResol), break, end %Resolution reached
    %Resolution not reached. More work is needed
    kImf = kImf- qAlfa*bBias'; % subtract qAlfa bias from
    kImf
    rPMOri= rGetPMx_s(kImf); % rPM= [xM(M), yM(M)];
    rPmOri= rGetPMin_s(kImf); % rPm= [xm(m), ym(m)];
    rPM= rMaxExtrapol_s(rPMOri, rPmOri, quntN);
    rPm= rMinExtrapol_s(rPMOri, rPmOri, quntN);
    bTenv= spline(rPM(:,1), rPM(:,2), 1:quntN); % Top envelop:
bTenv[n];
bDenv = spline(rPm(:,1), rPm(:,2), 1:quntN);       % Down envelop:
bDenv[n];

end % Wend true

% new bias estimate
bBias = (bTenv+bDenv)/2;

rParabEmd = [rParabEmd; kImf'];
% store the extracted rParabEmd
in the matrix rParabEmd
kc = kc - kImf;
% subtract the extracted rParabEmd from the
signal
quntOscCnt= quntNOsc_s(kc);

rParabEmdCnt=rParabEmdCnt+1;
if (kc'*kc)>0
    qDbResid= 10*log10(Wx/(kc'*kc));
else
    qDbResid = Inf
end

end % Wend ((DbR... ))

if ((kc'*kc)/Wx)>(10^(-12))
    rParabEmd=[rParabEmd; kc'];
    %The residual is the last IMF
    rParabEmdCnt=rParabEmdCnt+1;
    NumOscqResiduais = quntNOsc_s(kc);
end

rParabEmd= rParabEmd';

end %main function

%SubFunctions

function ArgCheck_s(x, qResol, qResid, qAlfa)

[qL, qC] = size(x);
if ((qL*qC)~= max(qL,qC)), error('rParabEmd__L: Input signal must be a one
dim vector.'), end
if ((qL*qC)<= 1), error('rParabEmd__L: Input signal
must be a vector.'), end

[qL,qC] = size(qResol);
if ( ~(qL==1)&(qC==1)) , error('rParabEmd__L: Input resolution must be a
calar.'), end
if ( qResol<=0 ), error('rParabEmd__L: Input resolution must strictly
positive.'), end

[qL,qC] = size(qResid);
if ( ~(qL==1)&(qC==1)) , error('rParabEmd__L: Input residual must be a
calar.'), end
if ( qResid<=0 ), error('rParabEmd__L: Input residual must strictly
positive.'), end

[qL,qC] = size(qAlfa);
if ( ~(qL==1)&(qC==1)) , error('rParabEmd__L: qAlfa step must be a
calar.'), end
if ( qAlfa<=0 ), error('rParabEmd__L: qAlfa step must be strictly positive.'), end
end

%--------------------------------------------------------------------------
%----------- make at 17-Jul-07 10:16:59.44
% quntNOsc_s     v1.01
%                build 20070409001
%                Returns the oscillation count, no steps
function quntNOsc = quntNOsc_s (x)

y=0;    qisTop= false; qisDown= false;

for i=2:(length(x)-1)
    if( ((x(i-1)) < (x(i))) && ((x(i+1))< (x(i))) ) %Max /-
        y=y+1;
    end
    if( ((x(i-1)) > (x(i))) && ((x(i+1))> (x(i))) ) %min \\_/
        y=y+1;
    end

    if( ((x(i-1)) < (x(i))) && ((x(i+1))== (x(i))) ) %StepL /-
        qisTop= true; qisDown= false;
    end
    if( ((x(i-1)) == (x(i))) && ((x(i+1))< (x(i))) ) %stepR \-
        if qisTop; y=y+1; end;
        qisTop= false;
    end

    if( ((x(i-1)) > (x(i))) && ((x(i+1))== (x(i))) ) %stepL \-
        qisTop= false; qisDown= true;
    end
    if( ((x(i-1)) == (x(i))) && ((x(i+1))> (x(i))) ) %StepR /-
        if qisDown; y=y+1; end;
        qisDown=false;
    end
end % for i=2:(length(x)-1)
quntNOsc= y;
end % function y = quntNOsc_s (x)

%--------------------------------------------------------------------------
%----------- make at 17-Jul-07 10:16:59.44
% rPMaxExtrapol_s                                             V1.00
%                                               build 2007407001
% Time-mirrored top extrema (Parabolic Maxs) extrapolation

%Init -------------------------------
rPM= sortrows(rPM); %assumes nothing on rPM sort order
rPm= sortrows(rPm); %assumes nothing on rPm sort order

kTopTim1= rPM(:,1); kTopVal= rPM(:,2);
kDwnTim1= rPm(:,1); kDwnVal= rPm(:,2);
function rPMaxExtrapol_s(rPM, rPm, quntL)
% rPMaxExtrapol_s                                           V1.00
%                                               build 2007407001
% Time-mirrored up extrema (Parabolic Mins) extrapolation

%Start extrapolation ---------------------
if ( (kTopTim1(1)== 1) && (kDwnTim1(1)== 1) )
    disp ('rPMaxExtrapol_s: Poliextrema at signal''s start');
else ( (kTopTim1(1)<1) || (kDwnTim1(1)< 1) )
    disp ('rPMaxExtrapol_s: Invalid extrema at signal''s start');
else
    kTopTim1=[2-kDwnTim1(1); kTopTim1]; % New first Top at the (one
    % based) specular Min
    kTopVal=[kTopVal(1); kTopVal]; % Same Val as old first Top
end

% End extrapolation ---------------------
if ( (kTopTim1(end)== quntL) && (kDwnTim1(end)== quntL) )
    disp ('rPMaxExtrapol_s: Poliextrema at signal''s end');
else ( (kTopTim1(end)> quntL) || (kDwnTim1(end)> quntL) )
    disp ('rPMaxExtrapol_s: Invalid extrema at signal''s end');
else
    kTopTim1=[kTopTim1; (2*quntL - kDwnTim1(end))]; % New last Top at the
    % specular Min
    kTopVal=[ kTopVal; kTopVal(end)]; % Same Val as old last Top
end

% return value ------------------------
rPMaxExtrapol= sortrows([kTopTim1, kTopVal]);
end

%------------------------ make at 17-Jul-07 10:16:59.44

function rPMinExtrapol_s(rPM, rPm, quntL)
%rPMinExtrapol_s                                           V1.00
%                                               build 2007407001
% Time-mirrored down extrema (Parabolic Mins) extrapolation

%Start extrapolation ---------------------
if ( (kTopTim1(end)== quntL) && (kDwnTim1(end)== quntL) )
    disp ('rPMinExtrapol_s: Poliextrema at signal''s end');
else ( (kTopTim1(end)> quntL) || (kDwnTim1(end)> quntL) )
    disp ('rPMinExtrapol_s: Invalid extrema at signal''s end');
else
    kTopTim1=[kTopTim1; (2*quntL - kDwnTim1(end))]; % New last Top at the
    % specular Min
    kTopVal=[ kTopVal; kTopVal(end)]; % Same Val as old last Top
end

% return value ------------------------
rPMinExtrapol= sortrows([kTopTim1, kTopVal]);
end

%------------------------ make at 17-Jul-07 10:16:59.44

function rPMinExtrapol_s(rPM, rPm, quntL)
disp ('rPMinExtrapol_s: Poliextrema at signal''s end');
elseif ( (kTopTim1(end) > quntL) || (kDwnTim1(end) > quntL) )
  disp ('rPMinExtrapol_s: Invalid extrema at signal''s end');
else
  kDwnTim1=[kDwnTim1; (2*quntL - kTopTim1(end))];  % New last Dwn at the specular Max
  kDwnVal=[ kDwnVal; kDwnVal(end)];  % Same Val as old last Dwn
end

% return value ------------------------
rPMinExtrapol= sortrows([kDwnTim1, kDwnVal]);
end

%------------------------ make at 17-Jul-07 10:16:59.44
function rPMax= rGetPMaxs_s(aS)         %Get Parabolic Maxs, plateaus out
%                                       build 20070612001
kS= aS(:);
quntLenS=length(kS);
quntMaxCnt=0;
kSMNdx1= []; kSMVal=[];         %signal S Maxima indices and values
kSPMTim1= []; kSPMVal=[];       %signal S Parabolic Maxima times and values
if (quntLenS>2)     %if signal has enough length
  for Cnt=2:(quntLenS-1)  %search the Maxs
    if ( ((kS(Cnt) > kS(Cnt+1)) && ((kS(Cnt) >= kS(Cnt-1))) || ((kS(Cnt) >= kS(Cnt+1)) && ((kS(Cnt) > kS(Cnt-1)))))
      quntMaxCnt=quntMaxCnt+1;
kSMNdx1= [kSMNdx1; Cnt];  kSMVal=[kSMVal; kS(Cnt)];
    end
  end
end

% Now we have the Maxs, lets get the Parabolic Maxs
oldxv= -Inf; oldyv= -Inf;
intGapMax= max(kS)-min(kS);
for jj=1:quntMaxCnt     %for all Maxs
  %xa= -1; xb= 0; xc= 1;
ya= kS(kSMNdx1(jj)-1);  % Sample point before
yb= kS(kSMNdx1(jj));    % Sample point, == kSMVal(jj)
yc= kS(kSMNdx1(jj)+1);  % Sample point after
D= (-4*yb+2*ya+2*yc);
if (D==0), xv= kSMNdx1(jj);
else xv= kSMNdx1(jj)+(ya-yc)/D; end; % Vertix abscissa
D= (-16*yb+ 8*ya+ 8*yc);
if (D==0), yv= yb;
else yv= yb+ (2*yc*ya- ya*ya- yc*yc)/D; end;
% Lets check for double maxima
if ( ((xD==oldxv)||(abs(yv-oldyv)/abs(xv-oldxv)) > (2*intGapMax) )
  xv= (xv+ oldxv)/2; yv= max(yv,oldyv);  %Double found
  kSPMTim1(length(kSPMTim1))= xv;  kSPMVal(length(kSPMVal))= yv;
else
  kSPMTim1= [kSPMTim1; xv];  kSPMVal=[kSPMVal; yv];
end
oldxv= xv; oldyv= yv;
end % for jj=1:quntMaxCnt
if quntMaxCnt>0
    if ( kS(1) >= kSPMVal(1) )
        kSPMTim1 = [1; kSPMTim1]; kSPMVal=[kS(1); kSPMVal ]; %Start must be included as a Max
    end
    if ( kS(end) >= kSPMVal(end))
        kSPMTim1 = [kSPMTim1; quntLenS]; kSPMVal=[kSPMVal; kS(end)]; %End must be included as a Max
    end
end
if quntMaxCnt==0
    if ( kS(1) > kS(2) )
        kSPMTim1 = [1; kSPMTim1]; kSPMVal=[kS(1); kSPMVal ]; %Start must be included as a Max
    end
    if ( kS(end) > kS(end-1))
        kSPMTim1 = [kSPMTim1; quntLenS]; kSPMVal=[kSPMVal; kS(end)]; %End must be included as a Max
    end
end
if quntMaxCnt<0
    error('rGetPMaxs_s: Invalid MaxCnt value');
end
rPMax= sortrows([kSPMTim1, kSPMVal]);
end

%---------------- make at 17-Jul-07 10:16:59.44
function rPMin= rGetPMins_s(aS) %Get Parabolic Mins, plateaus out
% build 20070612001
kS= aS(:);
quntLenS=length(kS);
quntMinCnt=0;
kSMNdx1= []; kSMVal=[]; %signal S Minima indices and values
kSPMTim1= []; kSPMVal=[]; %signal S Parabolic Minima times and values
if (quntLenS>2) %if signal has enough length
    for Cnt=2:(quntLenS-1) %search the Mins
        if ( ((kS(Cnt) < kS(Cnt+1))) && ((kS(Cnt) <= kS(Cnt-1)))||( (kS(Cnt) <= kS(Cnt+1))) && ((kS(Cnt) < kS(Cnt-1))) )
            quntMinCnt=quntMinCnt+1;
kSMNdx1 = [kSMNdx1; Cnt]; kSMVal=[kSMVal; kS(Cnt)];
        end
    end
end
% Now we have the Mins, lets get the Parabolic Mins
oldxv= -Inf; oldyv= -Inf;
intGapMax= max(kS)-min(kS);
for jj=1:quntMinCnt %for all Mins
    %xa= -1; xb= 0; xc= 1;
ya = kS(kSMNdx1(jj)-1);  % Sample point before
yb = kS(kSMNdx1(jj));  % Sample point, == kSMVal(jj)
yc = kS(kSMNdx1(jj)+1);  % Sample point after
D = (-4*yb+2*ya+2*yc);
if (D==0), xv = kSMNdx1(jj);
else xv = kSMNdx1(jj)+ (ya-yc)/D; end; % Vertix abscissa
D = (-16*yb+ 8*ya+ 8*yc);
if (D==0), yv = yb;
else yv = yb+ (2*yc*ya- ya*ya- yc*yc)/D; end;
% Lets check for double minima
if ( (xv==oldxv)||(abs(yv-oldyv)/abs(xv-oldxv))> (2*intGapMax) )
xv = (xv+ oldxv)/2; yv = min(yv,oldyv);  %Double found
kSPMTim1(length(kSPMTim1))= xv; kSPMVal(length(kSPMVal))= yv;
else
kSPMTim1= [kSPMTim1; xv];  kSPMVal=[kSPMVal; yv];
end
oldxv= xv; oldyv= yv;
end % for jj=1:quntMinCnt

if quntMinCnt>0
  if ( kS(1) <= kSPMVal(1) )
    kSPMTim1= [1; kSPMTim1];  kSPMVal=[kS(1); kSPMVal ];  %Start must
    be included as a Min
  end
  if ( kS(end) <= kSPMVal(end))
    kSPMTim1= [kSPMTim1; quntLenS];  kSPMVal=[kSPMVal; kS(end)];  %End
    must be included as a Min
  end
end

if quntMinCnt==0
  if ( kS(1) < kS(2) )
    kSPMTim1= [1; kSPMTim1];  kSPMVal=[kS(1); kSPMVal];  %Start must be
    included as a Min
  end
  if ( kS(end) < kS(end-1))
    kSPMTim1= [kSPMTim1; quntLenS];  kSPMVal=[kSPMVal; kS(end)];  %End
    must be included as a Min
  end
else
  error(‘rGetPMins_s: Invalid MinCnt value’);
end

rPMin= sortrows([kSPMTim1, kSPMVal]);
end
%---------- make at 17-Jul-07 10:16:59.44
function [Y, YY, frameVoiced] = preprocessflag(signal, thresh, Fs, x, t)
% PREPROCESS takes an input signal and removes any silence present as well % as shortens it to a length of t seconds
% % INPUTS
% signal : wav file
% thresh : if maxval > thesh, signal considered voiced
% Fs : sampling frequency of the wav file
% x : length of frame (in # of samples)
% t : length of post-processed signal (in seconds)
% % OUTPUT
% Y : signal with silence removed (first t seconds)
% YY : "  " (second t seconds)
% frameVoiced: vector containing a 1 if frame is voiced, 0 otherwise
% % This program was written for use in processing telephone data, where the % speaker on one end of the line is silenced. In addition, the signals % could be upwards of 4-5 minutes long when silence is removed so is % shortens them to a desired length
% 27 June 2013 R.A. Metzger
% UPDATE 8/09/2013: added a second output to the function. This output is % shifted by length t. Also padded outputs with zeros
%=========================================================================
frameLength = x;
signalLength = length(signal);
nFrames = floor(signalLength/frameLength);

newSignal = zeros(signalLength,1);
noSilence = 0;

for k = 1:nFrames
    % Extract each frame from the speech signal
    frame = signal((k-1)*frameLength+1:frameLength*k);

    % Determines weather frame as an amplitude greater than the threshold
    maxValue = max(frame);

    if (maxValue > thresh)
        % If frames has amplitude above threshold, assign it to new signal
        noSilence = noSilence+1;
        newSignal((noSilence-1)*frameLength+1:frameLength*... 
        noSilence)=frame;
        frameVoiced(k) = 1;
    else
        frameVoiced(k) = 0;
    end
end
% Remove excess zeros
newSignal(noSilence*frameLength:end)=[];

SEGMENT_LENGTH = t*Fs;

% Takes the first segment of length t
Y = [newSignal((1-1)*SEGMENT_LENGTH+1:SEGMENT_LENGTH*1); zeros(3,1)];
% Y = [newSignal((1-1)*SEGMENT_LENGTH+1:SEGMENT_LENGTH*1)];
% Takes the segment delayed by t
% YY = [newSignal((2-1)*SEGMENT_LENGTH+1:SEGMENT_LENGTH*2)];
YY = [zeros(3,1); newSignal((2-1)*SEGMENT_LENGTH+1:SEGMENT_LENGTH*2)];
Appendix C

Script for Performing Tests in 4-1 and 4-2

%This program takes a text file with full path directories to all training %set signals and a second text file with full path directories to all %testing set signals. It then breaks them down with EMD, computes MFCCs of %modes and of the entire signal, and uses a Gaussian Mixture Model to %compare and record recognition rates.
%
%Results are output to the workspace and as .xls files in the local %directory
%
% 10 March 2016 Kevin Bartuska
%===============================================================================

%set parameters
qresol = 40;
qresid = 20;

mfcc_framelength = 128;
mfcc_overlap = 64;
mfcc_order = 20;

preproc_thresh = .02;
preproc_frame = 256;
preproc_length = 10;

gmm_K = 16;

%TRAINING SET----------

%process signals
[filename, pathname] = uigetfile('.txt', 'Please select a text file with directories');

%ensure they didn't click cancel
if not(isequal(filename, 0))
    directory = fullfile(pathname, filename);
    fid = fopen(directory, 'r');
    speakerArray = textscan(fid, '%s');
    speakerArray = speakerArray{:};
    TRAIN_nSpeakers = length(speakerArray);

trueindex = 1; %a safe variable in case we skip anything
%read in all the files, store signals in handles
for i = 1:TRAIN_nSpeakers
    iSpeakers = speakerArray{i};

if exist(iSpeakers, 'file') == 2
    [sig, ftemp] = audioread(iSpeakers);
    TRAIN_signals{trueindex} = sig;

    [~, name, ext] = fileparts(iSpeakers);
    fileName{trueindex} = strcat(name, ext);
    trueindex = trueindex + 1;
else
    warndlg({'The following file was not found:', iSpeakers})
end
end
TRAIN_nSpeakers = trueindex - 1;
TRAIN_fs = cell(1, TRAIN_nSpeakers);
for i = 1:TRAIN_nSpeakers
    TRAIN_fs{i} = 8000;
end
else
    msgbox({'Operation Cancelled', 'Current files retained in memory'})
    return
end

%TEST-----------------
%process signals
[filename, pathname] = uigetfile('.txt', 'Please select a text file with directories');

%ensure they didn't click cancel
if not(isequal(filename, 0))
    directory = fullfile(pathname, filename);
    fid = fopen(directory, 'r');
    speakerArray = textscan(fid, '%s');
    speakerArray = speakerArray{:};
    TEST_nSpeakers = length(speakerArray);
trueindex = 1; % a safe variable in case we skip anything
% read in all the files, store signals in handles
for i = 1:TEST_nSpeakers
    iSpeakers = speakerArray{i};
    if exist(iSpeakers, 'file') == 2
        [sig, ftemp] = audioread(iSpeakers);
        TEST_signals{trueindex} = preprocessflag(sig, preproc_thresh, 8000, preproc_frame, preproc_length);
    [~, name, ext] = fileparts(iSpeakers);
    fileName{trueindex} = strcat(name, ext);
    trueindex = trueindex + 1;
else

warndlg({'The following file was not found:', iSpeakers})
end

TEST_nSpeakers = trueindex-1;
TEST_fs = cell(1, TEST_nSpeakers);
for i=1:TEST_nSpeakers
    TEST_fs{i}=8000;
end
else
    msgbox({'Operation Cancelled', 'Current files retained in memory'})
    return
end

%TRAIN
%find modes of signal
for i=1:TRAIN_nSpeakers
    TRAIN_modes{i} = rParabEmd__L(TRAIN_signals{i}, qresol, qresid, 1);
end
size(TRAIN_modes{1})

%find out how many modes at a minimum
for i=1:TRAIN_nSpeakers
    TRAIN_numModes(i) = min(size(TRAIN_modes{i}));
end
TRAIN_minNumModes = min(TRAIN_numModes)

%TEST
%find modes of signal
for i=1:TEST_nSpeakers
    TEST_modes{i} = rParabEmd__L(TEST_signals{i}, qresol, qresid, 1);
end
size(TEST_modes{1})

%find out how many modes at a minimum
for i=1:TEST_nSpeakers
    TEST_numModes(i) = min(size(TEST_modes{i}));
end
TEST_minNumModes = min(TEST_numModes)

%Finding MFCCs-----------------------
globalminModes = min(TRAIN_minNumModes, TEST_minNumModes)

%find MFCCs of original signals
for i=1:TRAIN_nSpeakers
    TRAIN_wholeMFCCs{i} = mfcc(TRAIN_signals{i}, mfcc_framelength, mfcc_overlap, mfcc_order, TRAIN_fs{i})
end
size(TRAIN_wholeMFCCs)

%find mfccs of each IMF
for i = 1:TRAIN_nSpeakers
    for j = 1:globalminModes
        TRAIN_imfMFCCs{i,j} = mfcc(TRAIN_modes{i}(:, j), mfcc_framelength, mfcc_overlap, mfcc_order, TRAIN_fs{i})';
    end
end
size(TRAIN_imfMFCCs)

%find MFCCs of original signals
for i = 1:TEST_nSpeakers
    TEST_wholeMFCCs{i} = mfcc(TEST_signals{i}, mfcc_framelength, mfcc_overlap, mfcc_order, TEST_fs{i})';
end
size(TEST_wholeMFCCs)

%find mfccs of each IMF
for i = 1:TEST_nSpeakers
    for j = 1:globalminModes
        TEST_imfMFCCs{i,j} = mfcc(TEST_modes{i}(:, j), mfcc_framelength, mfcc_overlap, mfcc_order, TEST_fs{i})';
    end
end
size(TEST_imfMFCCs)

%build useful cell arrays to feed to GMM
clear TRAIN_GMM_INPUT_EACHIMF
clear TEST_GMM_INPUT_EACHIMF
for i = 1:TRAIN_nSpeakers
    temp1 = TRAIN_imfMFCCs{i,1}';
    temp2 = TRAIN_imfMFCCs{i,2}';
    temp3 = TRAIN_imfMFCCs{i,3}';
    temp4 = TRAIN_imfMFCCs{i,4}';
    TRAIN_GMM_INPUT_EACHIMF{i} = [temp1 temp2 temp3 temp4]';
end

for i = 1:TEST_nSpeakers
    temp1 = TEST_imfMFCCs{i,1}';
    temp2 = TEST_imfMFCCs{i,2}';
    temp3 = TEST_imfMFCCs{i,3}';
    temp4 = TEST_imfMFCCs{i,4}';
    TEST_GMM_INPUT_EACHIMF{i} = [temp1 temp2 temp3 temp4]';
end

% Build GMM, and run comparison
% Taking MFCCs of each mode
for i=1:TRAIN_nSpeakers
    mixm{i} = GaussianMixture(TRAIN_GMM_INPUT_EACHIMF{i}, gmm_K, gmm_K, true);
end
eachIMFSignalsMixture = mixm;

% Testing for MFCCs of each mode
LLike = 1;
clear matchspeaker;
for k = 1:TEST_nSpeakers
    clear LLike % Has to be cleared because it changes size each iteration
    for J = 1:length(eachIMFSignalsMixture)
        LLike(:,j) = GMClassLikelihood(eachIMFSignalsMixture{j}, TEST_GMM_INPUT_EACHIMF{k});
    end
    V = sum(LLike);
    [m, k1] = max(V);
    matchSpeaker{k,1} = TEST_filename{k};
    matchSpeaker{k,2} = 'matches';
    matchSpeaker{k,3} = TRAIN_filename{k1};
end
disp('EachIMF')
disp(matchSpeaker)
xlswrite(strcat(pwd, '\firstFourIMFMFCCsResults.xls'), matchSpeaker);

% Taking MFCCs of entire signal (baseline)
for i=1:TRAIN_nSpeakers
    mixm{i} = GaussianMixture(TRAIN_wholeMFCCs{i}, gmm_K, gmm_K, true);
end
wholeSignalsMixture = mixm;

% Testing for MFCCs of entire signal
LLike = 1;
for k = 1:TEST_nSpeakers
    clear LLike % Has to be cleared because it changes size each iteration
    for J = 1:length(wholeSignalsMixture)
        LLike(:,j) = GMClassLikelihood(wholeSignalsMixture{j}, TEST_wholeMFCCs{k});
    end
end
end
V = sum(LLike);

[m, k1] = max(V);

matchSpeaker{k,1} = TEST_filename{k};
matchSpeaker{k,2} = 'matches';
matchSpeaker{k,3} = TRAIN_filename{k1};
end
disp('Whole MFCCs')
disp(matchSpeaker)
xlswrite(strcat(pwd, '\wholeMFCCsResults.xls'), matchSpeaker);
Appendix D

Script for Performing Tests in 4-3

%This program takes a text file with full path directories to all training
%set signals and a second text file with full path directories to all
%testing set signals. It then breaks them down with EMD, computes MFCCs of
%modes and of the entire signal. The principle component vectors are
%extracted using SVD, and those vectors are used to train a Gaussian
%Mixture Model.
%
%Results of the mixture comparisons are output to the workspace and as
%.xls files in the localdirectory
%
% 10 March 2016 Kevin Bartuska
% =========================================================================

%set parameters

gresol = 40;
gresid = 30;

mfcc_framelength = 128;
mfcc_overlap = 64;
mfcc_order = 20;

preproc_thresh = .02;
preproc_frame = 256;
preproc_length = 10;

gmm_K = 16;

%TRAINING SET---------

%process signals
[filename, pathname] = uigetfile('.txt', 'Please select a text file with
directories');

%ensure they didn't click cancel
if not(isequal(filename, 0))
    directory = fullfile(pathname, filename);
    fid = fopen(directory, 'r');
    speakerArray = textscan(fid,'%s');
    speakerArray = speakerArray{:};
    TRAIN_nSpeakers = length(speakerArray);

    trueindex = 1; % a safe variable in case we skip anything
    % read in all the files, store signals in handles
for i = 1:TRAIN_nSpeakers
    iSpeakers = speakerArray{i};

    if exist(iSpeakers, 'file') == 2
        [sig, ftemp] = audioread(iSpeakers);
        TRAIN_signals{trueindex} = sig;
        [~, name, ext] = fileparts(iSpeakers);
        TRAIN_filename{trueindex} = strcat(name, ext);
        trueindex = trueindex + 1;
    else
        warndlg({'The following file was not found:', iSpeakers})
    end
end
TRAIN_nSpeakers = trueindex - 1;
TRAIN_fs = cell(1, TRAIN_nSpeakers);
for i = 1:TRAIN_nSpeakers
    TRAIN_fs{i} = 8000;
end
else
    msgbox({'Operation Cancelled', 'Current files retained in memory'})
    return
end

%TEST-----------------
%process signals
[filename, pathname] = uigetfile('.txt', 'Please select a text file with directories');

%ensure they didn’t click cancel
if not(isequal(filename, 0))
    directory = fullfile(pathname, filename);
    fid = fopen(directory, 'r');
    speakerArray = textscan(fid, '%s');
    speakerArray = speakerArray{:};
    TEST_nSpeakers = length(speakerArray);

    trueindex = 1; %a safe variable in case we skip anything
    %read in all the files, store signals in handles
    for i = 1:TEST_nSpeakers
        iSpeakers = speakerArray{i};

        if exist(iSpeakers, 'file') == 2
            [sig, ftemp] = audioread(iSpeakers);
            TEST_signals{trueindex} = preprocessflag(sig, preproc_thresh, 8000, preproc_frame, preproc_length);
            [~, name, ext] = fileparts(iSpeakers);
        end
    end
TEST_filename(trueindex) = strcat(name, ext);

trueindex = trueindex + 1;
else
    warndlg({'The following file was not found:', iSpeakers})
end
end

TEST_nSpeakers = trueindex-1;
TEST_fs = cell(1, TEST_nSpeakers);
for i=1:TEST_nSpeakers
    TEST_fs{1}=8000;
end
else
    msgbox({'Operation Cancelled', 'Current files retained in memory'})
    return
end

%TRAINING
%find modes of signal
for i=1:TRAIN_nSpeakers
    TRAIN_modes{i} = rParabEmd_L(TRAIN_signals{i}, qresol, qresid, 1);
    if mod(i, 10) == 0
        disp('num speakers')
        disp(i)
    end
end

size(TRAIN_modes{1})

%find out how many modes at a minimum
for i=1:TRAIN_nSpeakers
    TRAIN_numModes(i) = min(size(TRAIN_modes{i}));
end
TRAIN_minNumModes = min(TRAIN_numModes)

disp('finished training')

%TESTING
%find modes of signal
for i=1:TEST_nSpeakers
    TEST_modes{i} = rParabEmd_L(TEST_signals{i}, qresol, qresid, 1);
    if mod(i, 10) == 0
        disp('num speakers')
        disp(i)
    end
end
size(TEST_modes{1})

% find out how many modes at a minimum
for i=1:TEST_nSpeakers
    TEST_numModes(i) = min(size(TEST_modes{i}));
end
TEST_minNumModes = min(TEST_numModes)

disp('finished testing')

% Finding MFCCs ------------------
globalminModes = min(TRAIN_minNumModes, TEST_minNumModes)

% find MFCCs of original signals
for i=1:TRAIN_nSpeakers
    TRAIN_wholeMFCCs{i} = mfcc(TRAIN_signals{i}, mfcc_framelength, mfcc_overlap, mfcc_order, TRAIN_fs{i})';
    [U, lambda, ~] = svd(TRAIN_wholeMFCCs{i}');
    sumsquaresPC = sqrt(sum(diag(lambda).^2));
    % weighted
    for k=1:5
        if k==1
            TRAIN_wholeSigfourPCs{i} = U(:, k)*lambda(k, k)/sumsquaresPC;
        else
            TRAIN_wholeSigfourPCs{i} = [TRAIN_wholeSigfourPCs{i} U(:, k)*lambda(k, k)/sumsquaresPC];
        end
    end
    % unweighted
    % TRAIN_wholeSigfourPCs{i} = U(:, [1,2,3,4,5,6]);
end
size(TRAIN_wholeMFCCs)

% find MFCCs of each IMF
clear TRAIN_fourPCs
for i = 1:TRAIN_nSpeakers
    for j=1:globalminModes
        TRAIN_imfMFCCs{i,j} = mfcc(TRAIN_modes{i}(:, j), mfcc_framelength, mfcc_overlap, mfcc_order, TRAIN_fs{i})';
        [U, lambda, V] = svd(TRAIN_imfMFCCs{i, j}');
        sumsquaresPC = sqrt(sum(diag(lambda).^2));
        % weighted coefs
        for k=1:5
            if j==1 && k==1
                TRAIN_fourPCs{i} = U(:, k)*lambda(k, k)/sumsquaresPC;
            else
                TRAIN_fourPCs{i} = [TRAIN_fourPCs{i} U(:, k)*lambda(k, k)/sumsquaresPC];
            end
        end
        % unweighted coefs
        % if j==1
% TRAIN_fourPCs{i} = U(:, [1,2,3,4,5,6]);
% else
% end
end
size(TRAIN_imfMFCCs)

% find MFCCs of original signals
for i=1:TEST_nSpeakers
    TEST_wholeMFCCs{i} = mfcc(TEST_signals{i}, mfcc_framelength,
mfcc_overlap, mfcc_order, TEST_fs{i}');
    [U, lambda, ~] = svd(TEST_wholeMFCCs{i}');
    sumsquaresPC = sqrt(sum(diag(lambda).^2));
    % weighted
    for k=1:5
        if k==1
            TEST_wholeSigfourPCs{i} = U(:, k)*lambda(k, k)/sumsquaresPC;
        else
            TEST_wholeSigfourPCs{i} = [TEST_wholeSigfourPCs{i} U(:, k)*lambda(k, k)/sumsquaresPC];
        end
    end
    % unweighted
    % TEST_wholeSigfourPCs{i} = U(:, [1,2,3,4,5,6]);
end
size(TEST_wholeMFCCs)

% find mfccs of each IMF
clear TEST_fourPCs
for i = 1:TEST_nSpeakers
    for j=1:globalminModes
        TEST_imfMFCCs{i,j} = mfcc(TEST_modes{i}(:, j), mfcc_framelength,
mfcc_overlap, mfcc_order, TEST_fs{i}');
        [U, lambda, V] = svd(TEST_imfMFCCs{i, j}');
        sumsquaresPC = sqrt(sum(diag(lambda).^2));
        % weighted coefs
        for k = 1:5
            if j==1 && k==1
                TEST_fourPCs{i} = U(:, k)*lambda(k, k)/sumsquaresPC;
            else
                TEST_fourPCs{i} = [TEST_fourPCs{i} U(:, k)*lambda(k, k)/sumsquaresPC];
            end
        end
        % unweighted coefs
        % if j==1
        %   TEST_fourPCs{i} = U(:, [1,2,3,4,5,6]);
        % else
        %   TEST_fourPCs{i} = [TEST_fourPCs{i} U(:, [1,2,3,4,5,6])];
        % end
    end
end
end
size(TEST_imfMFCCs)

%Build GMM, run comparisons

%Training on first four PCs of whole signal
for i=1:TRAIN_nSpeakers
    mixm{i} = GaussianMixture(TRAIN_wholeSigfourPCs{i}, gmm_K, gmm_K, true);
end
trainingGMM = mixm;

%Testing on first four PCs of whole signal
LLike = 1;
for k = 1:TEST_nSpeakers
    clear LLike % Has to be cleared because it changes size each iteration
    for J = 1:length(trainingGMM)
        LLike(:,j) = GMClassLikelihood(trainingGMM{j},
            TEST_wholeSigfourPCs{k});
    end
    V = sum(LLike);
    [m, k1] = max(V);
    matchSpeaker{k,1} = TEST_filename{k};
    matchSpeaker{k,2} = 'matches';
    matchSpeaker{k,3} = TRAIN_filename{k1};
end
disp('EachIMF')
disp(matchSpeaker)
xlswrite(strcat(pwd, '\WEIGHTED_wholeSigFivePCsResults.xls'), matchSpeaker);

%Training on first four PCs
for i=1:TRAIN_nSpeakers
    mixm{i} = GaussianMixture(TRAIN_fourPCs{i}, gmm_K, gmm_K, true);
end
trainingGMM = mixm;

%Testing on first four PCs
LLike = 1;
for k = 1:TEST_nSpeakers
    clear LLike % Has to be cleared because it changes size each iteration
    for J = 1:length(trainingGMM)
        LLike(:,j) = GMClassLikelihood(trainingGMM{j},
            TEST_fourPCs{k});
    end
end
end
V = sum(LLike);

[m, k1] = max(V);

matchSpeaker{k,1} = TEST_filename{k};
matchSpeaker{k,2} = 'matches';
matchSpeaker{k,3} = TRAIN_filename{k1};

disp('EachIMF')
disp(matchSpeaker)
xlswrite(strcat(pwd, '\WEIGHTED_fivePCsResults.xls'), matchSpeaker);
BIBLIOGRAPHY


ACADEMIC VITA

Academic Vita of Kevin Bartuska
kzb5260@psu.edu

Education
Major and Minors:
B.S. in Electrical Engineering, with minors in Math and Physics
The Pennsylvania State University

Thesis Title: Empirical Mode Decomposition Applied to Speaker Identification
Thesis Supervisor: Dr. John Doherty

Work Experience:
January 2014 through May 2016
Mathnasium Math Tutor
Tutored students grade 2 through 12 in mathematics
Mathnasium, 273 Northland Center, State College, PA
Deb Cusatis

Summer 2015
National Instruments Hardware Intern
Designed prototype Compact RIO module with full time and intern engineers
National Instruments, 11500 N Mopac Expressway, Austin, TX

Grants Received:
Recipient of the H. Thomas and Dorothy Willits Hallowell Scholars Endowment through the Penn State College of Engineering
Recipient of the Penn State Schreyer Honors College Academic Excellence Scholarship

Certified LabVIEW Associate Developer (CLAD)

Language Proficiency: English