EE627 – Term Project : Jul. 2013 Semester

August 12, 2013

Title : Build a speaker verification system

Assigned to : Batch No. 10

TAs Assigned : Waquar Ahmad. He is Available @ACES 203 MiPS Lab EE Department. email : wahmad@iitk.ac.in

Objective :

The objective of this project is to build a speaker verification system. A speaker verification system needs to be built using the NIST2004 database. Experimental results for speaker verification, in terms of equal error rate (EER) also need to be provided.

Methodology To Be Followed :

The methodlolgy need to be followed for performing the recogniton experiments are as follows:

- Building the task grammar (a "language model")
- Feature extraction.
- Model Preparation .
- Model Re-estimation
- Evaluating the recognizer against the test data
- Reporting recognition results

Database for Training

The 2004 NIST Speaker Recognition evaluation is part of an ongoing series of yearly evaluations conducted by NIST (National Institute of Standards and Technology). These evaluations provide an important contribution to the direction of research efforts and the calibration of technical capabilities. They are intended to be of interest to all researchers working on the general problem of text-independent speaker recognition. To this end the evaluation was designed to be simple, to focus on core technology issues, to be fully supported, and to be accessible.NIST has been coordinating Speaker Recognition Evaluations since 1996. Each evaluation begins with the announcement of the official evaluation plan which clearly states the rules and tasks involved with the evaluation. The evaluation culminates with a follow-up workshop, where NIST reports the official results and researchers share in their findings.The data consists of conversational telephone speech collected by the LDC.Additional documentation is available from the NIST website at http://www.itl.nist.gov/iad/mig/tests/sre/2004/index.html.The NIST 2004 database is available at MiPS Lab. Please contact the TA assigned for the same.

Tools To Be Used

The Tools to be used to implement this term project is the HTK Toolkit and matlab. The details of download, installation, and usage are available at the following URL :

http://htk.eng.cam.ac.uk/

Deliverables/ Submissions

The deliverables or submission procedures for the term project are as follows:

- Presentations and Report : Two set of presentations is required for every batch in this term project. The first presentation will be scheduled before mid sem and the second presentation will be scheduled before end sem. The marks will be distributed separately for two presentation. Additionally a report needs to be submitted detailing the project.
- System Demo in real time : The demonstration of the project is need to be carried out by each batch. The demonstration includes the real time presentation of the working model for recognition system.
- Code/script submissions : The code or script has to be submitted by each batch which will include complete details of the project.

Other Useful Links

- Matlab Code : speaker-recognition.googlecode.com/files/project.pdf

SPEAKER IDENTIFICATION: A DEMONSTRATION USING MATLAB

Anil Alexander, Andrzej Drygajlo

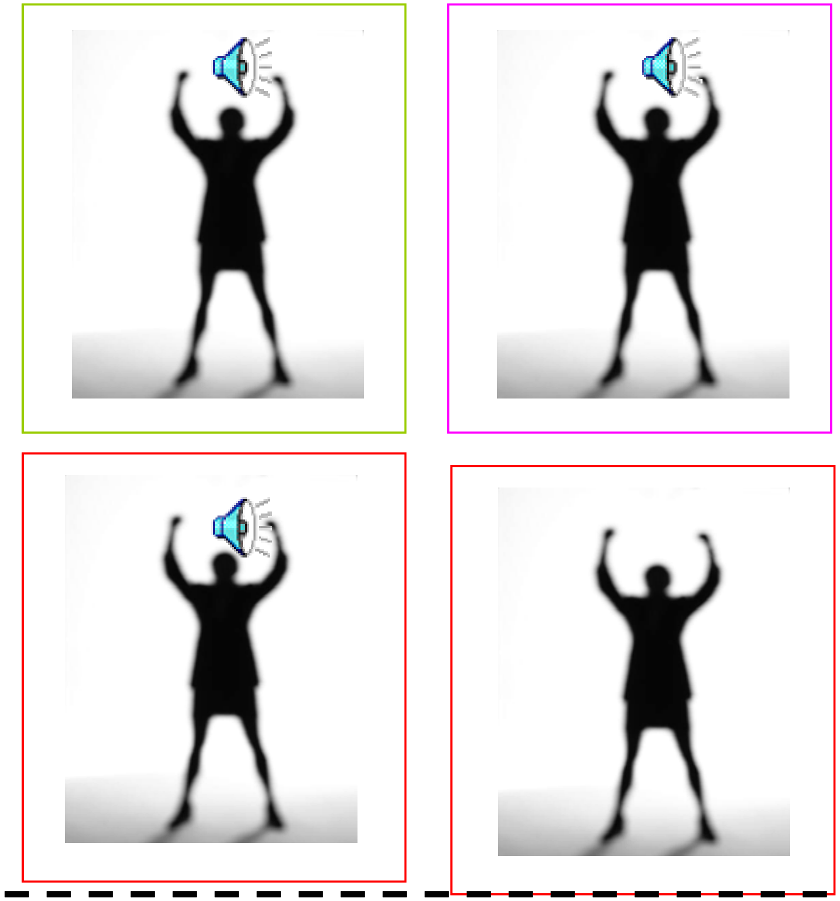
Swiss Federal Institute of Technology, Lausanne
Signal Processing Institute

Demo and description available at:
http://scgwww.epfl.ch/courses/Biometrics-Lectures-2005-2006-pdf/03-Biometrics-Exercise-3-2005/

Problem

Unknown Speaker requesting access

Speakers known to the recognition system



Which of the speakers in the recognition system is the unknown speaker ?

What needs to be done ?

- To identify the voices of the unknown speaker we need to:
  - Extract characteristic features of the speech of the known speakers
  - Create models of the features of the known speakers
  - Compare the features from the unknown speaker’s utterances with the statistical models of the voices of the speakers known to the system.
  - Make decision when we have identified that test utterance belongs to a certain speaker.

A speaker recognition system

- We can construct a speaker recognition system using some of the tools

  - Feature extraction using MFCC (Mel Frequency Cepstral Coefficients)

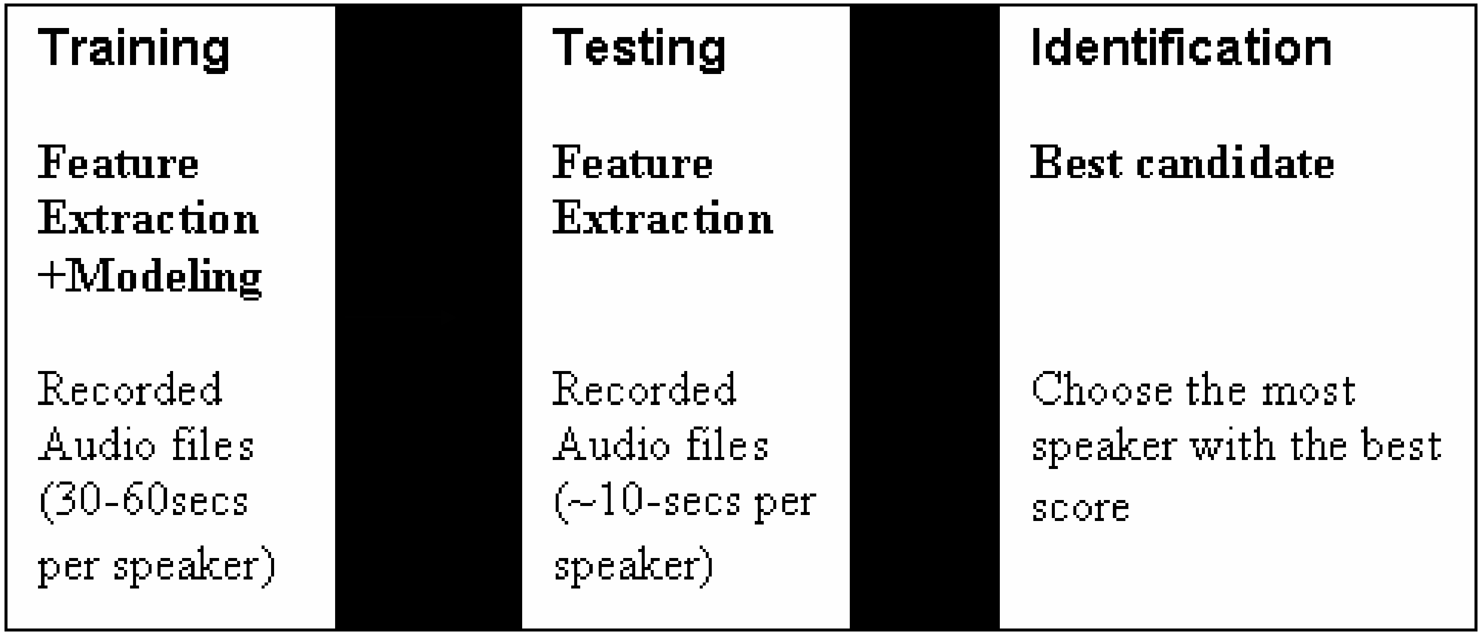
  - Statistical modeling using Gaussian Mixture Modeling (Expectation-Maximization algorithm)

Outline of tasks to be performed

The following steps have to be done:

- Read in the training and testing files
- Feature extraction for both the training and testing files
- Statistical modeling of the features of the training files
- Testing each of the test files with the models created
- Choosing the ‘best candidate’ test file corresponding to a training model, and verifying whether both of them came from the same speaker.

Outline



Training	Testing	Identification
Feature Extraction +Modeling	**Feature Extraction**	**Best candidate**
Recorded Audio files (30-60secs per speaker)	Recorded Audio files (~10-secs per speaker)	Choose the most speaker with the best score

Matlab tools

- In order to make a simple speaker recognition system, you can use a few publicly available tools
  - Feature Extraction using Mel Frequency Cepstral Coefficients (MFCC) available from Voicebox, Imperial College, England

  http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
  - Statistical modeling using Gaussian Mixture Modeling (GMM), from our lab's website

  http://scgwww.epfl.ch/matlab/student_labs/2004/labs/Lab08_Speaker_Recognition/

Read in training and testing files

- We can use the Matlab function wavread

WAVREAD Read Microsoft WAVE (".wav") sound file.

From Matlab help:

Y=WAVREAD(FILE) reads a WAVE file specified by the string FILE, returning the sampled data in Y.

Ex: `training_data1=wavread('01_train.wav');`

Feature extraction

- For MFCC feature extraction, we use the melcepst function from Voicebox.

- MELCEPST Calculate the mel cepstrum of a signal

Simple use: c=melcepst(s,fs)   % calculate mel cepstrum with 12 coefs, 256 sample frames

Ex: training_features1=melcepst(training_data1,8000);

Statistical modeling

- For statistical modeling we use the ‘gmm_evaluate’ function

  Performs statistical modeling of the features, using the Gaussian Mixture Modeling algorithm, and returns the means, variances and weights of the models created.

  [mu,sigma,c]=gmm_estimate(X,M,<iT,mu,sigm,c,Vm>)

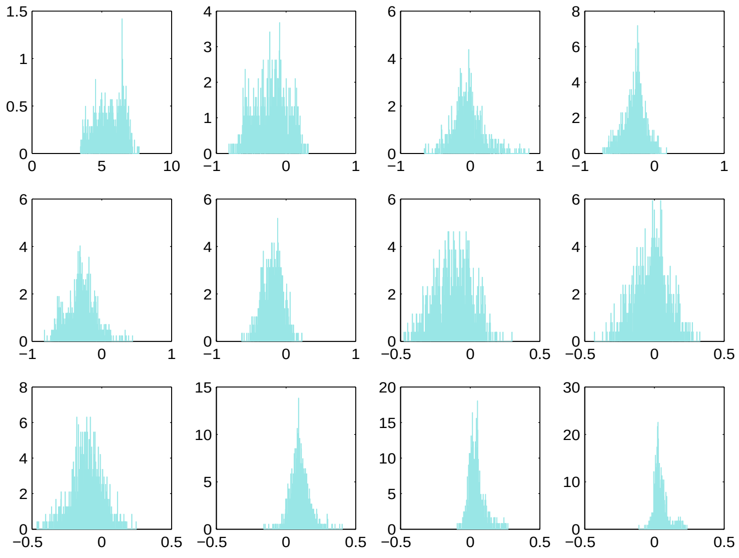
  X : the column by column data matrix (LxT)

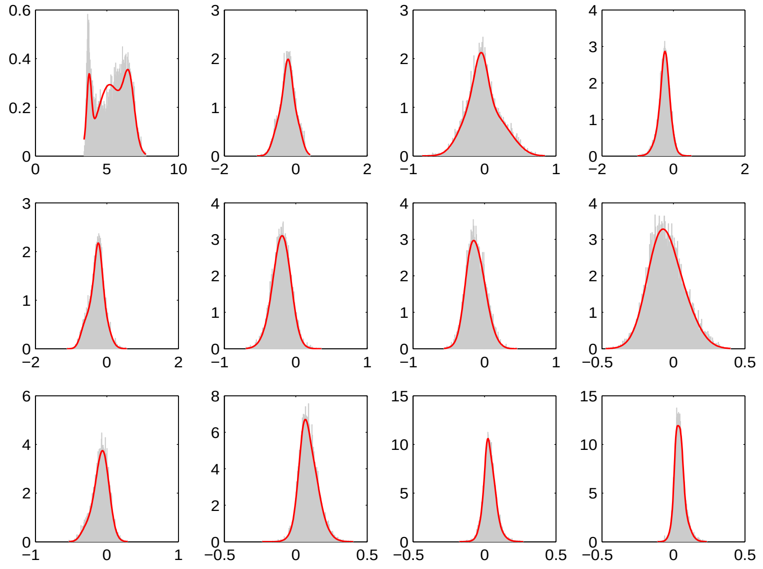
  M : number of gaussians

  iT : number of iterations, by default 10

Ex:[mu_train1,sigma_train1,c_train1]=gmm_estimate(training_features, No_of_Gaussians);

Statistical modeling





Features of the speaker

Model of the features of the speaker

Testing

- For testing, we use the lmultigauss function.

[lYM,lY]=lmultigauss(X,mu,sigma,c)

computes multigaussian log-likelihood, using the test data (X), the means (mu), the diagonal covariance matrix of the model (sigma) and the matrix representing the weights of each of the models (c).

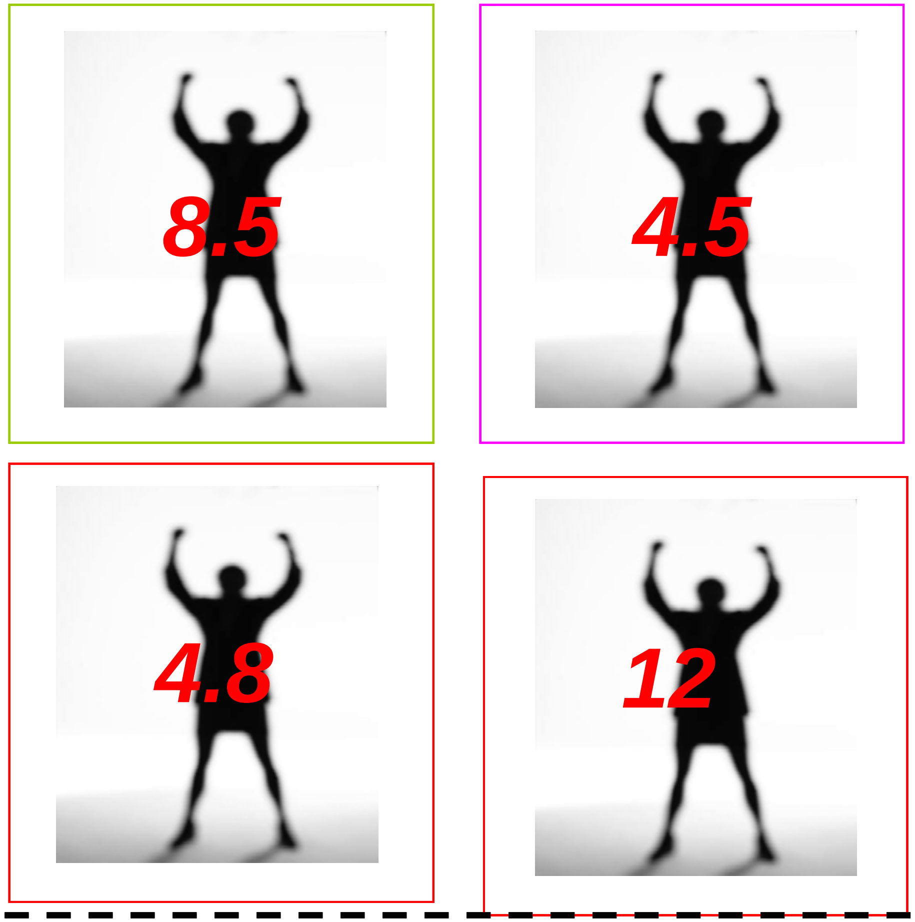
Ex:

[lYM,lY]=lmultigauss(testing_features1,mu_train1,sigm_train1, c_train1);

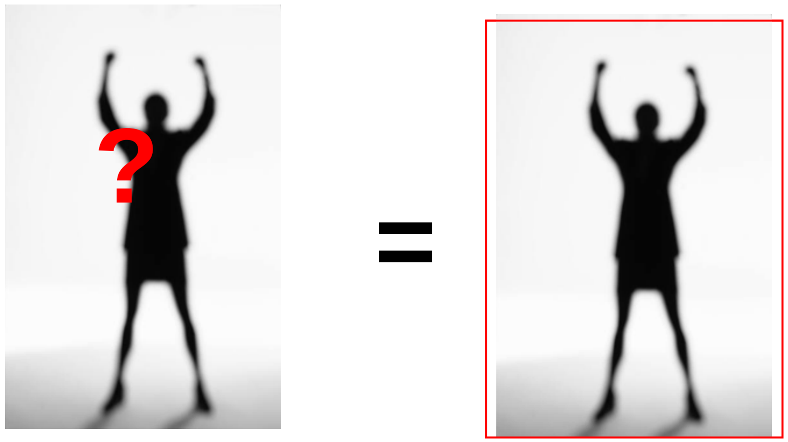
mean(lYM);

Solution for the problem ?

Scores obtained for the models of the speakers



Unknown Speaker requesting access



The speaker who has obtained the maximum score is assigned to the unknown speaker

Additional resources and references

Feature extraction (other possible options)

- **Use code for LPC already in Matlab**
- **RASTA-PLP coefficients from Dan Ellis' website**
  - **http://www.ee.columbia.edu/~dpwe/resources/matlab/rastamat/**

Gaussian Mixture Modeling of the features

- **Douglas A. Reynolds, Thomas F. Quatieri, and Robert B. Dunn. Speaker verifi*cation using adapted Gaussian mixture models*. Digital Signal Processing, 10(103), 2000.**

SGN-1656 Signal Processing Laboratory
Voice biometrics: Automatic speaker identification

1 General information

In this project work, we build a Matlab program for speaker recognition. There are two phases: in the first phase – *training phase* – we learn a statistical model for the spectral characteristics of each speaker in the training data and in the second phase – *test phase* – we use the models to identify a speaker of previously unseen test sentences.

The main concepts of this work are frame-wise feature extraction for speech signals (here line spectral frequencies derived from linear prediction filter coefficients) and classification using Gaussian mixture models. More information about speech signal processing is available in SGN-4010 `http://www.cs.tut.fi/kurssit/SGN-4010/`. Pattern classification is discussed in more details in SGN-2506 `http://www.cs.tut.fi/kurssit/SGN-2506/`.

To pass this exercise, you should write the required MATLAB codes and a report of the work. You can include the MATLAB codes in the report as an appendix. Send the report in a pdf form to `hanna.silen@tut.fi` by 10.5.2013.

2 Speaker identification based on spectral features

Automatic speaker recognition can be divided into speaker identification and speaker verification. The first one refers to the process of finding out who the speaker of a speech segment is while the latter one refers to the process of checking whether the speaker is who he claims to be.

In this exercise, we build a MATLAB program for text independent speaker identification and evaluate its performance for new data. The text-independence means that no attention is paid to the verbal content message of speech. Instead, we investigate the characteristics of the speech spectra an model the speaker-specific characteristics using one of the most widely used approaches in speaker identification, Gaussian mixture model (GMM) based speaker identification [1]. An extensive overview of the methods for speaker recognition is given in [2] (available at `http://cs.joensuu.fi/pages/tkinnu/webpage`).

The recognition process consists of two phases. In the first phase, a training phase, we learn parameters for the statistical GMM models based on some training material. In the second phase, a test phase, the identification accuracy of the learned models is evaluated using data that was not included in the model training. Both the training and testing phases work on spectral feature vectors extracted frame-wise from speech waveforms. Line spectral frequency (LSF) coefficients used in this work are derived from linear prediction coefficients and are described in the following.

2.1 Frame-wise feature extraction

Typically automatic speaker identification (as well as speaker verification, speech recognition, etc.) approaches rely on the use of some spectral features extracted frame-wise from audio signals. In the feature extraction, the speech signal is processed in short frames of 20-30 ms in which the speech can be assumed to remain stationary. Typically windowing by some smooth window, such as Hanning window, is used. For each windowed frame, we extract a feature vector that is later used as an observation vector in the speaker modeling.

Typical features used in speaker identification are related to the speech spectrum. The commonly used examples include mel-frequency cepstral coefficients (MFCCs) as well as linear prediction (LP) based features such line spectral frequency (LSF) coefficients – MFCCs probably being the most common in speaker recognition.

In this exercise, we use the LP-based LSFs as frame-wise spectral features in speaker modeling and identification. Computation of LSF coefficients starts from LP analysis that is one of the most important tools in digital speech processing. MATLAB provides a straightforward conversion between LP and LSF coefficients. We will review the estimation of LSF coefficients next.

Extraction of LSF coefficients

Speech production can be modeled as a source-filter system where a source signal produced by the vocal cords is filtered by a vocal tract filter with resonances at formant frequencies (for recap, check `http://www.cs.tut.fi/kurssit/SGN-4010/LP_en.pdf`). The frequencies of the formants are both phoneme and speaker-specific, the first formants being related to the phoneme identity while the high-frequency formants being more speaker-dependent.

LP analysis provides a tool for the analysis of the vocal tract characteristics. Vocal tract can be considered to be pth order all-pole filter $1/A(z)$:

$$\frac{1}{A(z)} = \frac{1}{1 + a_1 z^{-1} \ldots + a_p z^{-p}} \tag{1}$$

where the filter coefficients $a_1 \ldots a_p$ are estimated using linear prediction. Figure 1 (a) illustrates the spectrum of the all-pole filter $1/A(z)$ estimated from a short speech frame from a signal representing a Finnish vowel *y*. The thin line represents the amplitude spectrum (absolute value of discrete Fourier transform, DFT) of the frame.

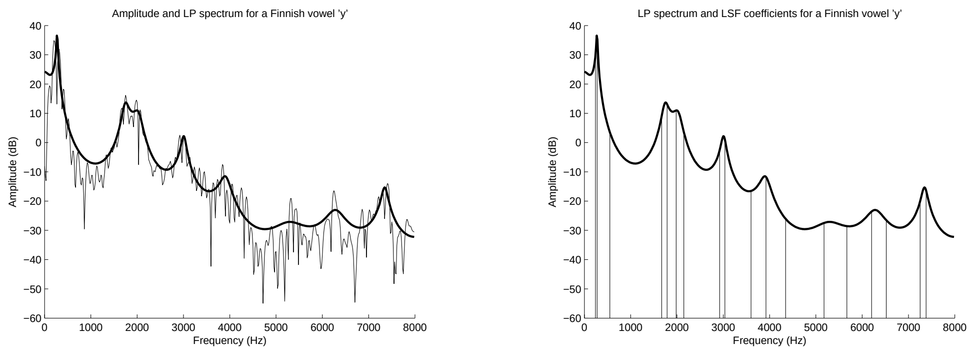


Figure 1: *LP spectrum (thick line) for a frame extracted from a speech signal and (a) the amplitude spectrum (thin line) and (b) the LSF values for the same frame.*

In MATLAB, LP coefficient of a given order are estimated by the function `lpc`:

```
a = lpc(frame,order); % Estimate LP coefficients
```

Coefficients of an LP filter have poor interpolation properties: a small change in the coefficients can even make the all-pole filter $1/A(z)$ unstable! For speech analysis purposes, LP polynomial $A(z)$ is often decomposed into line spectral frequencies (LSF). LSFs have good quantization and interpolation properties and are thus widely used in many speech applications such as speech coding. The LSF representation of the previous LP spectrum is given in Figure 1 (b). Again the thick line represents the LP spectrum and the frequency values of the thin lines represent the LSF coefficient values. As seen in the figure, two closely spaced LSF coefficients tend to indicate a peak in the LP spectrum.

A more detailed derivation for the estimation of LSF coefficients is given for example in `http://www.cs.tut.fi/sgn/arg/8003102/syn_en.pdf`. The basic idea is to decompose the LP polynomial $A(z)$ into two polynomials that have their roots on the unit circle. The resulting LSF vector Ω corresponding to $A(z)$ (of order p) consists of p angle values (or equivalently, frequency values) ω_i:

$$\Omega = (\omega_1, \omega_2, \ldots, \omega_p).$$

In this exercise, you can use MATLAB's LP-to-LSF conversion of the function `poly2lsf`:

```
x = poly2lsf(a); % Convert LP coefficients into LSF coefficients
```

Note that the values of vector `x` are between 0 and π (whereas in the figure the LSF values are scaled to the range 0 – 8 kHz). In the following, frame-wise LSF vectors ω are used as observation vectors $\mathbf{x}$ in speaker modeling and identification.

2.2 Statistical modeling of speaker characteristics

Speaker modeling with Gaussian mixture models (GMMs) [1] is a classical approach used in automatic speaker identification. A GMM consists of several Gaussian components representing general speaker-specific characteristics. The two phases – modeling (training) and identification (testing) – are described in the following.

Speaker modeling with GMMs

The most commonly used approach for speaker identification uses Gaussian mixture models (GMM) to model the characteristics of each speaker's spectral features. A Gaussian mixture density $p(\mathbf{x}|\lambda)$ is defined as a sum of M Gaussian components:

$$p(\mathbf{x}|\lambda) = \sum_{i=1}^{M} p_i \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i), \tag{2}$$

where $\mathbf{x}$ is an observation vector of size $D \times 1$, p_i is the prior probability or mixing weight of ith Gaussian component $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ and $\sum_{i=1}^{M} p_i = 1$. The ith multivariate Gaussian distribution with mean $\boldsymbol{\mu}_i$ and covariance $\boldsymbol{\Sigma}_i$ is defined as:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_i|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)\right]. \tag{3}$$

Here $\boldsymbol{\mu}_i$ is a vector of size $D \times 1$ and $\boldsymbol{\Sigma}_i$ a matrix of size $D \times D$. $|\boldsymbol{\Sigma}_i|$ is the determinant of $\boldsymbol{\Sigma}_i$. To decrease the number of model parameters, diagonal covariance matrices with non-zero values only on the main diagonal are typically used instead of full covariance matrices.

The values for the GMM model parameters $\lambda = \{p_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$ with $i = 1, \ldots, M$ are estimated based on training data. For a training vector sequence $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$, the likelihood of $\mathbf{X}$ given the model λ is

$$p(\mathbf{X}|\lambda) = \prod_{t=1}^{T} p(\mathbf{x}_t|\lambda). \tag{4}$$

The likelihood function in (4) is a nonlinear function of λ and we need an iterative *expectation maximization* (EM) algorithm [3] to find the values for the parameters in the model λ that maximize the likelihood function. The details of EM algorithm are omitted here but there is more information available for instance in the material of SGN-2506. In this work, you can use an estimation function already available in MATLAB.

In the modeling phase (*training phase*), the parameters of a speaker-dependent GMM model λ_s of (2) are estimated separately for each of the speakers $s = \{1, \ldots, S\}$ in the training material. In the simplest case, the model λ_s is learned only from the training vectors of the sth speaker. The EM algorithm for the model parameter estimation is implemented also in the Statistics Toolbox of MATLAB. To estimate the GMM parameters (mixing weights, means, and diagonal covariances), you can use the following code snippet:

```
% Xs - Training data of the sth speaker
% M  - Number of Gaussian components in the GMM

% Options - Maximum number of EM iterations 100
options = statset('Display','final','MaxIter',100);

speakerModel = gmdistribution.fit(Xs,M,'CovType','diagonal',...
                                  'Options',options);
```

Speaker identification based on GMMs

In the speaker identification phase (*test phase*), speaker-specific models learned in the training phase are used to identify the speaker of the test vector sequences. The identification is made based on the maximum *a posteriori* probability: the speaker of a test vector sequence is decided to be the one whose GMM model gives the highest posterior probability.

The maximum a posteriori estimate for the speaker of a test vector sequence $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$ is:

$$\hat{s} = \arg\max_{1 \le j \le S} p(\lambda_j|\mathbf{X}) = \arg\max_{1 \le j \le S} \frac{p(\mathbf{X}|\lambda_j)p(\lambda_j)}{p(\mathbf{X})}. \tag{5}$$

The term $p(\lambda_j)$ is a prior probability of the jth speaker, and can be assumed to be equal $(1/S)$ for all speakers. The denominator $p(\mathbf{X})$ is also equal for all the speakers, hence we can write:

$$\hat{s} = \arg\max_{1 \le j \le S} p(\mathbf{X}|\lambda_j). \tag{6}$$

By assuming the observations to be independent this can be written as a sum of log-densities:

$$\hat{s} = \arg \max_{1 \le j \le S} \sum_{t=1}^{T} \log p(\mathbf{x}_t | \lambda_j). \tag{7}$$

The probability density $p(\mathbf{x}_t|\lambda_j)$ is the same as in (2). The segment is identified to be spoken by the speaker whose model λ_s maximizes (7).

For the implementation of the identification step, there are tools available in the MATLAB Statistics toolbox. You can use them or write your own code.

3 Assignment: MATLAB program for speaker identification

In the previous section, you got a review of frame-wise LSF feature extraction and speaker modeling and identification using GMMs. You should now write your own MATLAB program for speaker identification and evaluate its recognition accuracy for new data.

3.1 Data

The set we use in this work is a subset of a CHiME speech database (available at `http://spandh.dcs.shef.ac.uk/projects/chime/PCC/datasets.html`). The set we use for this exercise is `PCCdata16kHz_devel_clean.tar.gz` (size 51M) from:

`http://spandh.dcs.shef.ac.uk/projects/chime/PCC/data/`

The selected subset consists of 16-bit stereo WAV files with a sampling frequency of 16 kHz from 34 different speakers. The files are named to indicate directly the speaker: e.g. files s1_*.wav are spoken by the speaker 1 and s23_*.wav by the speaker 23.

The first task is to download the data and extact the zip package. Divide the sentences into training and test sentences. The training sentences are used for learning the parameters of the speaker-specific GMM models and the test sentences for evaluating the identification accuracy of the learned models. For each speaker, use 10 randomly chosen sentences for training (roughly half of the data) and the rest for testing. Operate on mono-signals: to convert the original stereo signals into mono signals, use averaging of the left and right channels.

3.2 MATLAB implementation for training and testing

Write a MATLAB code that implements the speaker identification system described in Section 2. Use training data to learn the parameters of speaker-specific GMMs and test data to evaluate the identification accuracy.

For both training and test data, extract frame-wise LSFs of order 18 (sampling frequency of the speech signals is 16 kHz). Use Hanning windowing of frames and a frame length of 30 ms and a frame overlap of 50%. Before the windowing and feature extraction, use pre-filtering of the audio signals with a filter $y(n) = x(n) - 0.97x(n-1)$ to emphasize the high frequency content.

Build a speaker-specific GMM model for each speaker using the LSF feature vectors extracted from the speaker's training data. Set the number of Gaussian components to $M = 8$. Use diagonal covariance matrices.

Evaluate the identification accuracy by using the test data. Identify the speaker in each test sentence by using the GMM models learned from the training data. Compute the identification accuracy as in [1] for (a) each speaker as and (b) for the whole data by using the formula:

$$\text{\% correct identification} = 100 \cdot \frac{\text{number of correctly identified segments}}{\text{total number of segments}}. \tag{8}$$

Are some of the speakers more difficult to identify than the others? How does the accuracy change if you increase/decrease the number of Gaussian components in GMM modeling? What happens if the number of Gaussian components is too low/high? Are there some bottlenecks in your implementation or in the general approach? Include the evaluation results in the report for instance as a table or a figure.

4 Writing the report

Write a report of the work including the description of the implementation details and evaluation results for the given data set. Include the MATLAB code as an appendix of the report.

Since the exercise is new, feedback is also more than welcome (!) – for instance, were the instructions easy/hard to follow; are the tools applied in the exercise useful for further studies/work; and do you feel you learned something/nothing by completing this exercise? Include the feedback in the report.

Send the report in pdf form to `hanna.silen@tut.fi` by 10.5.2013.

References

[1] Reynolds, D. and Rose, R., Robust Text-Independent Speaker Identification Using Gaussian Mixture Speaker Models. *IEEE Transactions on Speech and Audio Processing*, Vol. 3, No. 1, 1995.

[2] Kinnunen, T. and Li, H., An Overview of Text-Independent Speaker Recognition: from Features to Supervectors, *Speech Communication*, Vol. 52, No. 1, 2010.

[3] Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the EM algorithm. In *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*. Vol. 39, No. 1, 1977.

Building a Simple Speaker Identification System

1. Introduction

1.1. We will be using the Hidden Markov Model Toolkit (HTK). HTK is installed under linux on the lab machines. Your path should already be set, but you can check this by looking for a htk-related path element in your `.bashrc` file in your home directory.

1.2. The HTK Book can be found at http://www.computing.dcu.ie/~john/htk/htkbook.pdf. Have a *quick* browse thru it. Familiarise yourself with where the tutorial section and the reference section can be found.

1.3. What follows is a step-by step tutorial on how to create a simple speaker recogniser. If there are any problems with the tutorial, or the way HTK is configured under linux, please notify your tutor and/or mail me.

1.4. The tutorial is like a basic *recipe*. As your skills develop you should experiment with other *ingredients*. The HTK book will give you ideas.

2. The Grammar

2.1. HTK uses a finite state grammar that consists of variables defined by regular expressions. Create a file called `gram.txt` and place the following in it.

```
$speaker = John | Mary;
($speaker)
```

2.2. A word network must be created from the grammar. This can be done using `HParse`:

```
HParse gram.txt wdnet
```

2.2.1. Have a look at the resulting file `wdnet`. Does it make any sense?

3. Recording

3.1. Record both John and Mary uttering the vowel /a/ as in "father" (as you may have done at the doctor's at some point in your life). Make four recordings of the vowel for each speaker. Place them in files `ja{1-4}.wav` and `ma{1-4}.wav`. Note the sampling rate at which the recordings were made.

4. Parameterisation

4.1. We must first extract relevant information from the speech spectra. This is called parameterisation. We will use Mel-frequency cepstral coefficients (MFCCs).

4.2. We must build a configuration file. Call it `config_wav2mfc`. In it place the following:

```
# Coding parameters
SOURCEKIND      = WAVEFORM
SOURCEFORMAT    = WAV
SOURCERATE      = Sampling Period in units of 10^-7s
TARGETKIND = MFCC_0_D_A
TARGETRATE = Frame skip duration in units of 10^-7s
SAVECOMPRESSED = T
SAVEWITHCRC = T
WINDOWSIZE = Frame duration in 10^-7s
USEHAMMING = T
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
NUMCEPS = 12
ENORMALISE = T
```

4.3. Note that in your configuration file, you must replace the italicised pieces with actual values. Use any frame skip in the range 10-20ms. Use any frame duration in the range 20-40ms.

4.4. We can list the `wav` files (and corresponding targets) we wish to parameterise in a file. Let's call it `convert.scp` and assume that our data is stored in a directory named `Data`:

```
Data\ja1.wav Data\ja1.mfc
Data\ja2.wav Data\ja2.mfc
Data\ja3.wav Data\ja3.mfc
Data\ja4.wav Data\ja4.mfc
Data\ma1.wav Data\ma1.mfc
Data\ma2.wav Data\ma2.mfc
Data\ma3.wav Data\ma3.mfc
Data\ma4.wav Data\ma4.mfc
```

4.5. MFCCs are extracted from the `.wav` files with `HCopy`:

```
HCopy -T 1 -C config_wav2mfc -S convert.scp
```

4.6. Use `HList` to view the contents of any of the `mfc` files. Do they make sense?

5. Model Preparation

5.1. We will now initialise two speaker models with our training data.

5.2. First we will create another configuration file, `config_mfc`, to let the HTK tools know about our `mfc` files:

```
# Coding parameters
TARGETKIND = MFCC_0_D_A
TARGETRATE = ??
SAVECOMPRESSED = T
SAVEWITHCRC = T
WINDOWSIZE = ??
USEHAMMING = T
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
NUMCEPS = 12
ENORMALISE = T
```

5.3. We will also list the `mfc` files we wish to use for training. We will use three tokens for training each model and retain the other for testing. List these in `training.scp`:

```
Data\ja1.mfc
Data\ja2.mfc
Data\ja3.mfc
Data\ma1.mfc
Data\ma2.mfc
Data\ma3.mfc
```

5.4. Create a new directory `hmm0` and place the following prototype model in a file called `proto`:

```
~o <VecSize> 39 <MFCC_0_D_A>
~h "proto"
    <BeginHMM>
     <NumStates> 5
     <State> 2
        <Mean> 39
          0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0
        <Variance> 39
          1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0
     <State> 3
        <Mean> 39
          0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0
        <Variance> 39
          1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
```

```
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0
     <State> 4
        <Mean> 39
          0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0
        <Variance> 39
          1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0
     <TransP> 5
      0.0 1.0 0.0 0.0 0.0
      0.0 0.6 0.4 0.0 0.0
      0.0 0.0 0.6 0.4 0.0
      0.0 0.0 0.0 0.7 0.3
      0.0 0.0 0.0 0.0 0.0
    <EndHMM>
```

5.5. Now use HCompV to initialise the models with the training data:

```
HCompV -C config_mfc -f 0.01 -m -S training.scp -M
hmm0 proto
```

5.6. This results in the creation of two files – `proto` and `vFloors` – in the directory `hmm0`. These files must be edited in the following way:

5.6.1. An error occurs at this point which rearranges the order of the parts of the `MFCC_0_D_A` label as `MFCC_D_A_0`. This must be corrected.

5.6.2. The first three lines of proto must be cut and pasted into `vFloors`, which is then saved as `macros`.

5.6.3. A file called `hmmdefs` is created by copying and pasting the rest of the proto file once for each HMM and renaming the copies accordingly. Note that each HMM begis with ~h "*model_name*" and ends with `<EndHMM>`. For instance you could call your models "ja" and "ma".

6. Model Re-estimation

6.1. Now that we have initialised our two speaker-vowel models with global means and variances, we will use HERest to perform Baum-Welch training.

6.2. For this step we need to create two more files: `speakertrainphones0.mlf` and `phonemodels0`. The former contains what the HTK book refers to as 'phone-level transcriptions'. These specify the phones in each of the sound files to be used in training the HMMs. The latter simply lists the phone-level models.

6.3. `speakertrainmodels0.mlf` will look like:

```
#!MLF!#
"*/ja1.lab"
ja
.
"*/ja2.lab"
ja
.
"*/ja3.lab"
ja
.
"*/ma1.lab"
ma
.
"*/ma2.lab"
ma
.
"*/ma3.lab"
ma
```

6.4. `phonemodels0` …

```
ja
ma
```

6.5. We then re-estimate our models

```
HERest -C config_mfc -I speakertrainmodels0.mlf -t
250.0 150.0 1000.0 -S training.scp -H hmm0/macros -H
hmm0/hmmdefs -M hmm1 phonemodels0
```

6.6. Do this three times, each time putting the re-estimated models in a new directory: `hmm1` (as above), `hmm2`, `hmm3`.

7. Recognition Testing

7.1. The Dictionary

7.1.1. The dictionary defines (in alphabetical order) speaker models by their constituent parts, i.e. by each HMM associated with them. For this speaker recogniser each model consists of only one HMM. A word model might consist of several phone states. (For example the word "one" might be defined as "w uh n" – and John's model for the word one might be "jw juh jn"). Similarly, you could include SENT-START and SENT-END can be defined by a silence model ('sil'), but 'sil' need not be included in this simple system. See the tutorial example in the HTK book for more on silence models

7.1.2. Your dictionary – call it `dict` – will look like this:

```
John            ja
Mary            ma
```

7.2. You also need to create a list of parameterised testing files and a list of models. The former will be of the same format as `training.scp` – call it `testing.scp`:

```
Data\ja4.mfc
Data\ma4.mfc
```

7.3. The latter is similar to `phonemodels0` but each model is enclosed in double quotes; call it `HmmList` and it will contain the following:

```
“ja”
“ma”
```

7.4. Now, having reminded yourself of the role of the grammar, we have all the elements in place to perform speaker recognition. Our recognised labels will be output to file `recout.mlf` when we carry out the recognition using `HVite` (which performs recognition using the Viterbi algorithm):

```
HVite -H hmm3/macros -H hmm3/hmmdefs -S testing.scp
-i recout.mlf -w wdnet -p 0.0 -s 5.0 dict HmmList
```

8. Results

8.1. We now want to gauge the accuracy of the recognition.

8.2. List the contents of the test files. The former will be similar in format to `speakertrainmodels0.mlf` except the models listed are at the speaker level rather than the phone level. Call it `speakertestmodels0.mlf`.

```
#!MLF!#
"*/ja4.lab"
John
.
"*/ma4.lab"
Mary
```

8.3. HResults is then used:

```
HResults -I speakertestmodels0.mlf HmmList
recout.mlf
```

8.4. Look up the HTK book to see how to:

8.4.1. interpret the results;

8.4.2. output a confusion matrix.