KTH Computer Science and Communication

DT2119 Lab1: Feature extraction

1 Objective

The objective is to experiment with different features commonly used for speech analysis and recognition. The lab is designed in Python, but the same functions can be obtained in Matlab/Octave or using the Hidden Markov Toolkit (HTK). In Appendix A, a reference table is given indicating the correspondence between different systems.

2 Task

- compute Mel Filterbank and MFCC features step-by-step
- examine features
- evaluate correlation between feature
- compare utterances with Dynamic Time Warping
- illustrate the discriminative power of the features with respect to words
- perform hierarchical clustering of utterances
- train and analyze a Gaussian Mixture Model of the feature vectors.

In order to pass the lab, you will need to follow the steps described in this document, and present your results to a teaching assistant. Use Canvas to book a time slot for the presentation. Remember that the goal is not to show your code, but rather to show that you have understood all the steps.

Note for VT2020: Because of the current quarantine situation, it is recommended that you use one of the cloud services (for example Google Colab[1]) and solve the exercise in a shared notebook.

3 Data

The files `lab1_data.npz` and `lab1_example.npz` contain the data to be used for this exercise. The files contains two arrays: `data` and `example`[2].

[1] `https://colab.research.google.com`

[2] If you wish to use Matlab/Octave instead of Python, use the provided `py2mat.py` script to convert to Matlab format. Load the file with `load lab1_data` or `lab1_example`. You will load two cell arrays with the corresponding data stored in structures.

3.1 example

The array `example` can be used for debugging because it contains calculations of all the steps in Section 4 for one utterance. It can be loaded with:

```
import numpy as np
example = np.load('lab1_example.npz', allow_pickle=True)['example'].item()
```

The element `example` is a dictionary with the following keys:

`samples`:	speech samples for one utterance
`samplingrate`:	sampling rate
`frames`:	speech samples organized in overlapping frames
`preemph`:	pre-emphasized speech samples
`windowed`:	hamming windowed speech samples
`spec`:	squared absolute value of Fast Fourier Transform
`mspec`:	natural log of `spec` multiplied by Mel filterbank
`mfcc`:	Mel Frequency Cepstrum Coefficients
`lmfcc`:	Liftered Mel Frequency Cepstrum Coefficients

Figure 1 shows the content of the elements in `example`.

3.2 data

The array `data` contains a small subset of the TIDIGITS database (`https://catalog.ldc.upenn.edu/LDC93S10`) consisting of a total of 44 spoken utterances from one male and one female speaker[3]. The file was generated with the script `lab1_data.py`[4]. For each speaker, 22 speech files are included containing two repetitions of isolated digits (eleven words: "oh", "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine"). You can read the file from Python with:

```
data = np.load('lab1_data.npz', allow_pickle=True)['data']
```

The variable `data` is an array of dictionaries. Each element contains the following keys:

`filename`:	filename of the wave file in the database
`samplingrate`:	sampling rate of the speech signal (20kHz in all examples)
`gender`:	gender of the speaker for the current utterance (`man`, `woman`)
`speaker`:	speaker ID for the current utterance (`ae`, `ac`)
`digit`:	digit contained in the current utterance (`o, z, 1, ..., 9`)
`repetition`:	whether this was the first (`a`) or second (`b`) repetition
`samples`:	array of speech samples

4 Mel Frequency Cepstrum Coefficients step-by-step

Follow the steps below to computer MFCCs. Use the `example` array to double check that your calculations are right.

You need to implement the functions specified by the headers in `lab1_proto.py`. Once you have done this, you can use the function `mfcc` in `lab1_tools.py` to compute MFCC coefficients in one go.

[3]The complete database contains recordings from 225 speakers

[4]The script is included only for reference in case you need to use the full database in the future. In that case, you will need access to the KTH AFS file system.

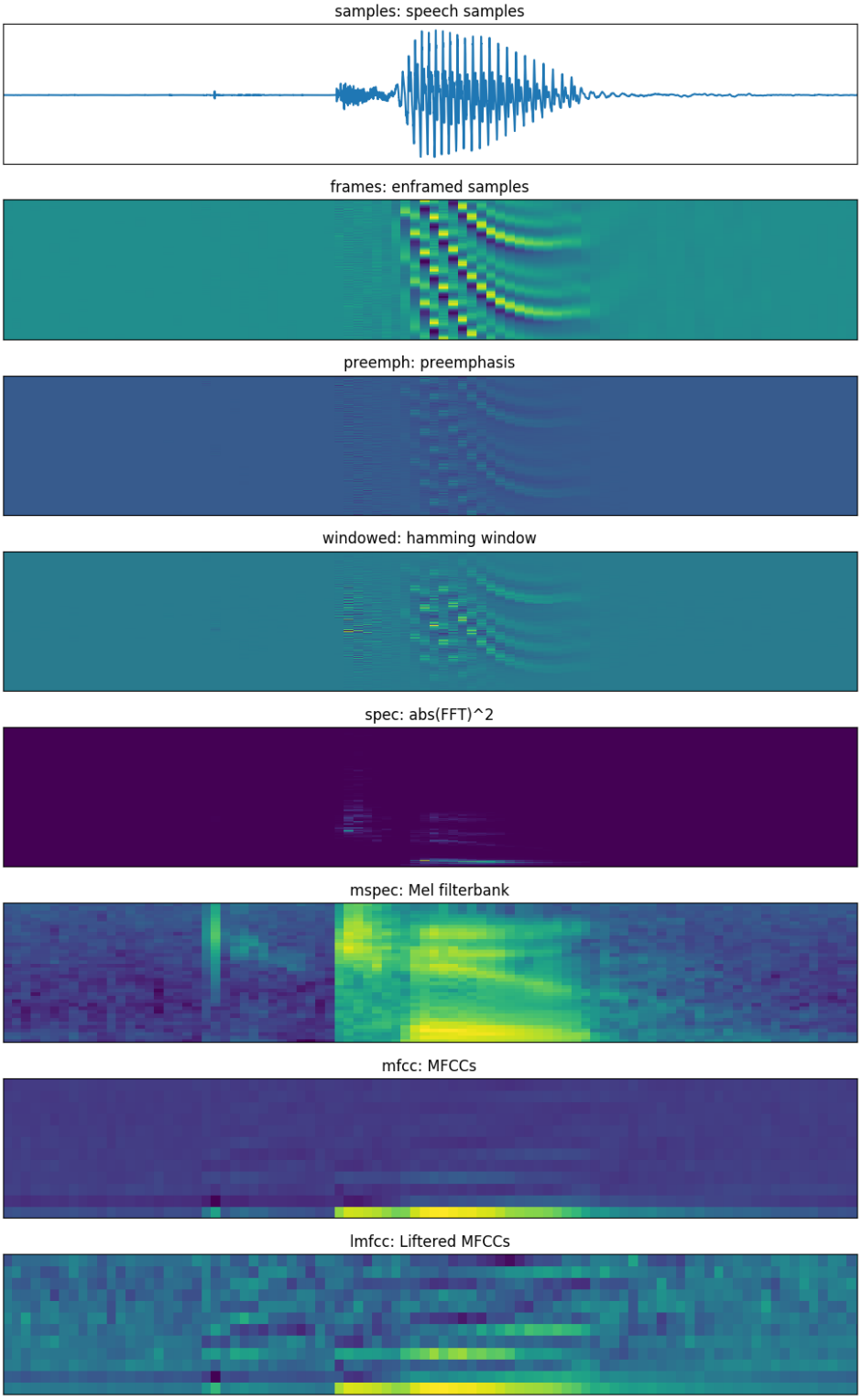

Figure 1. Evaluation of MFCCs step-by-step

4.1 Enframe

Implement the `enframe` function in `lab1_proto.py`. This will take as input speech samples, the frame length in samples and the number of samples overlap between consecutive frames and outputs a two dimensional array where each row is a frame of samples. Consider only the frames that fit into the original signal disregarding extra samples. Apply the enframe function to the utterance `example['samples']` with window length of 20 milliseconds and shift of 10 ms (figure out the length and shift in samples from the sampling rate, and write it in the lab report). Use the `pcolormesh` function from `matplotlib.pyplot` to plot the resulting array. Verify that your result corresponds to the array in `example['frames']`.

4.2 Pre-emphasis

Implement the `preemp` function in `lab1_proto.py`. To do this, define a pre-emphasis filter with pre-emphasis coefficient 0.97 using the `lfilter` function from `scipy.signal`. Explain how you defined the filter coefficients. Apply the filter to each frame in the output from the `enframe` function. This should correspond to the `example['preemph']` array.

4.3 Hamming Window

Implement the `windowing` function in `lab1_proto.py`. To do this, define a hamming window of the correct size using the `hamming` function from `scipy.signal` with extra option `sym=False`[5]. Plot the window shape and explain why this windowing should be applied to the frames of speech signal. Apply hamming window to the pre-emphasized frames of the previous step. This should correspond to the `example['windowed']` array.

4.4 Fast Fourier Transform

Implement the `powerSpectrum` function in `lab1_proto.py`. To do this, compute the Fast Fourier Transform (FFT) of the input from `scipy.fftpack` and then the squared modulus of the result. Apply your function to the windowed speech frames, with FFT length of 512 samples. Plot the resulting power spectrogram with `pcolormesh`. Beware of the fact that the FFT bins correspond to frequencies that go from 0 to $f_{\max}$ and back to 0. What is $f_{\max}$ in this case according to the Sampling Theorem? The array should correspond to `example['spec']`.

4.5 Mel filterbank log spectrum

Implement the `logMelSpectrum` function in `lab1_proto.py`. Use the `trfbank` function, provided in the `lab1_tools.py` file, to create a bank of triangular filters linearly spaced in the Mel frequency scale. Plot the filters in linear frequency scale. Describe the distribution of the filters along the frequency axis. Apply the filters to the output of the power spectrum from the previous step for each frame and take the natural log of the result. Plot the resulting filterbank outputs with `pcolormesh`. This should correspond to the `example['mspec']` array.

4.6 Cosine Transofrm and Liftering

Implement the `cepstrum` function in `lab1_proto.py`. To do this, apply the Discrete Cosine Transform (`dct` function from `scipy.fftpack.realtransforms`) to the outputs of the filterbank.

[5]The meaning of this option is beyond the scope of this course, but you should use it if you want to get the same results as in the example.

Use coefficients from 0 to 12 (13 coefficients). Note that using the `n=13` input parameter in `dct` is not the same as running without the argument and taking the first 13 elements in the results, try to explain why. Then apply liftering using the function `lifter` in `lab1_tools.py`. This last step is used to correct the range of the coefficients. Plot the resulting coefficients with `pcolormesh`. These should correspond to `example['mfcc']` and `example['lmfcc']` respectively.

Once you are sure all the above steps are correct, use the `mfcc` function (`lab1_tools.py`) to compute the liftered MFCCs for all the utterances in the `data` array. Observe differences for different utterances.

5 Feature Correlation

Concatenate all the MFCC frames from all utterances in the `data` array into a big feature $[N \times M]$ array where N is the total number of frames in the data set and M is the number of coefficients. Then compute the correlation coefficients between features[6] and display the result with `pcolormesh`. Are features correlated? Is the assumption of diagonal covariance matrices for Gaussian modelling justified? Compare the results you obtain for the MFCC features with those obtained with the Mel filterbank features (`'mspec'` features).

6 Explore Speech Segments with Clustering

Using the concatenated data from the previous section, train a Gaussian mixture model with `sklearn.mixture.GMM` (or `sklearn.mixture.GaussianMixture` depending on the version of `sklearn`). Vary the number of components for example: 4, 8, 16, 32. Consider utterances containing the same words and observe the evolution of the GMM posteriors. Can you say something about the classes discovered by the unsupervised learning method? Do the classes roughly correspond to the phonemes you expect to compose each word? Are those classes a stable representation of the word if you compare utterances from different speakers. As an example, plot and discuss the GMM posteriors for the model with 32 components for the four occurrences of the word "seven" (utterances 16, 17, 38, and 39).

7 Comparing Utterances

Given two utterances of length N and M respectively, compute an $[N \times M]$ matrix of local Euclidean distances between each MFCC vector in the first utterance and each MFCC vector in the second utterance.

Write a function called `dtw` (`lab1_proto.py`) that takes as input this matrix of local distances and outputs the result of the Dynamic Time Warping algorithm. The main output is the global distance between the two sequences (utterances), but you may want to output also the best path for debugging reasons.

For each pair of utterances in the `data` array:

1. compute the local Euclidean distances between MFCC vectors in the first and second utterance

2. compute the global distance between utterances with the `dtw` function you have written

[6] Note that we want the correlation between feature coefficients, and not between consecutive feature vectors, that is the resulting correlation matrix should be $M \times M$.

Store the global pairwise distances in a matrix D (44×44). Display the matrix with `pcolormesh`. Compare distances within the same digit and across different digits. Does the distance separate digits well even between different speakers?

Run hierarchical clustering on the distance matrix D using the `linkage` function from `scipy.cluster.hierarchy`. Use the "complete" linkage method. Display the results with the function `dendrogram` from the same library, and comment them. Use the `tidigit2labels` function (`lab1_tools.py`) to create labels to add to the dendrogram to simplify the interpretation of the results.

A Alternative Software Implementations

Although this lab has been designed for being carried out in Python, several implementations of speech related functions are available.

A.1 Matlab/Octave Instructions

The Matlab signal processing toolbox is one of the most complete signal processing piece of software available. Many speech related functions are however implemented in third party toolboxes. The most complete are the Voicebox[7] which is more oriented towards speech technology and the Auditory Toolbox[8] that is more focused on human auditory models.

If you use Octave instead of Matlab, make sure you have the following extra packages (in parentheses are the names of the corresponding `apt-get` packages for Debian based GNU Linux distributions, all packages are already installed on CSC Ubuntu machines):

- signal (`octave-signal`)

A.2 Hidden Markov Models Toolkit (HTK)

HTK is a powerful toolkit developed by Cambridge University for performing HMM-based speech recognition experiments. The HTK package is available at all CSC Ubuntu stations, or can be download for free at `http://htk.eng.cam.ac.uk/` after registration to the site. Its manual, the HTK Book, can be downloaded separately. In spite of being open source and free of charge, HTK, is unfortunately not free software in the Free Software Foundation sense because neither its original form nor its modifications can be freely distributed. Please refer to the license agreement for more information.

The HTK commands that are relevant to this exercise are the following:

HCopy: feature extraction tool. Can read audio files or feature files in HTK format and outputs HTK format files

HList: terminal based visualization of features. Reads HTK format feature files and displays information about them

General options are:

- `-C config`: reads configuration file `conf`
- `-S filelist`: reads list of files to process from `filelist`

for a complete list of options and usage information, run the commands without arguments.

Hint: `HList -r ...`: the `-r` option in `HList` will output the feature data in raw (ascii) format. This will make it easy to import the features in other programs such as python, Matlab or R.

Table 2 lists a number of possible spectral features and the corresponding HTK codes to be used in HCopy or HList.

[7] `http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html`
[8] http://amtoolbox.sourceforge.net/

Feature name	Matlab	Python
Linear filter	`filter`	`scipy.signal.lfilter`
Hamming window	`hamming`	`scipy.signal.hamming`
Fast Fourier Transform	`fft`	`scipy.fftpack.fft`
Discrete Cosine Transform	`dct`	`scipy.fftpack.realtransforms.dct`
Gaussian Mixture Model	`gmdistribution`	`sklearn.mixture.GMM`
Hierarchical clustering	`linkage`	`scipy.cluster.hierarchy.linkage`
Dendrogram	`dendrogram`	`scipy.cluster.hierarchy.dendrogram`
Plot lines	`plot`	`matplotlib.pyplot.plot`
Plot arrays	`image, imagesc`	`matplotlib.pyplot.pcolormesh`

Table 1. Mapping between Matlab and Python functions used in this exercise

Feature name	KTH code
linear filer-bank parameters	`MELSPEC`
log filter-bank parameters	`FBANK`
Mel-frequency cepstral coefficients	`MFCC`
linear prediction coefficients	`LPC`

Table 2. Feature extraction in HTK. The HCopy executable can be used to generate features from wave file to feature file. HList can be used to output the features in text format to stdout, for easy import in other systems