Spring 2022 Term Project: Speaker Recognition
ECE 466 Digital Signal Processing

March 18, 2022

1 Introduction

Speaker recognition (not speech recognition), is a key area of research based on human voice signals. Given a set of individuals and their corresponding audio recordings, the aim is to identify the speaker in the audio signal automatically after tuning (or training) a digital system for the process. This process allows computers to verify the identity of speakers and control access to personalized, or even, secure information such as; banking by phone, remote access to computers, database access services, home speakers, shopping with home speakers, etc. This task is achieved by learning and utilizing speaker-specific characteristics of input utterances, and inspired heavily from human ears, and how they perceive sounds.

In this project, you will:

1. Build a simple system for speaker recognition,
2. Implement a procedure to preprocess data such that the data is safe, clean and reliable,
3. Implement necessary machine learning building blocks, such as training, validation, testing
4. Utilize multiple models/methods for preprocessing and classification, and compare them.

2 Principals of Speaker Recognition

For a speaker recognition system, we might have two major tasks; identification and verification. Identification is the process of determining which registered speaker provides a given utterance. Verification, on the other hand, is the process of accepting or rejecting the identity claim of a speaker. The system that we will focus on is text-independent speaker identification system since its task is to identify the person who speaks regardless of what they are saying. This is akin to training and testing phases of a supervised machine learning system.

A speaker recognition system is built upon two modules: feature extraction and feature matching. Feature extraction is the process that extracts a small amount of data from the voice signal that can later be used to represent each speaker. This module is usually further tailored for other tasks as well, such as tailoring the features to help the feature matching, clean out data, anonymize the individuals, etc. Feature matching is the procedure to identify the unknown speaker by comparing extracted features from his/her voice input with the ones from a set of known speakers.

Most speaker recognition systems are supervised, or semi-supervised. Therefore, they have a training and a testing phase. In the training phase, each registered speaker has to provide samples of their speech so that the system can build or train a reference model for that speaker. In the testing phase, the input speech is matched with stored reference model(s) and a recognition decision is made.

3 Feature Extraction:

This module will process the speech waveform for further analysis. The speech signal is a slowly time-varying signal but when examined over a sufficiently short period of time (between 5-100 ms), it's characteristics

are often fairly stationary. However, over long periods of time (on the order of 1/5 seconds or more) the signal characteristic change to reflect the different speech sounds being spoken. Therefore, short-time Fourier analysis is the most common way to characterize the speech signal.

Different methods have been proposed to extract features from speech signals including Linear Prediction Coding (LPC), Mel-Frequency Cepstrum Coefficients (MFCC), and others. MFCC is perhaps the best known and most popular and will be the focus of this project. MFCC's are based on the known variation of the human ear's critical bandwidths with frequency, filters spaced linearly at low frequencies and logarithmically at high frequencies have been used to capture the phonetically important characteristics of speech. This is expressed in the mel-frequency scale, which is a linear frequency spacing below 1000 Hz and a logarithmic spacing above 1000 Hz. The process of computing MFCCs is described in more detail next. In addition to MFCC's, we will also use simple FFT and STFT as naive ways of extracting features and see if these approaches are viable.

3.1 Fast Fourier Transform Processor

A simple idea is extracting frequency magnitudes as features of the signal. Since we are only interested in certain frequencies, the size of the features for each recording will be fixed. Fast Fourier Transform converts the speech signal from the time domain into the frequency domain. The FFT is a fast algorithm to implement the Discrete Fourier Transform (DFT), which is defined on the set of values of speech signal $\{x_n\}$, as follows (see `scipy.fft.fft`):

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j2\pi kn/N}, \qquad k \in \{1, 2, \ldots, N-1\} \tag{1}$$

In general, X_k's are complex numbers and we only consider their absolute values, i.e. magnitude part of FFT. The resulting sequence $\{X_k\}$ is interpreted as follows, positive frequencies $0 \leq f < \frac{F_s}{2}$ correspond to values $0 \leq n \leq \frac{N}{2} - 1$, while negative frequencies $-\frac{F_s}{2} < f < 0$ correspond to $\frac{N}{2} \leq n \leq N - 1$. Here, F_s denotes the sampling frequency. As we are not interested in frequencies above 5kHz, we will discard the corresponding elements of the resulting vector.

3.2 Short Time Fourier Transform

As speech signals are generally very long vectors, and frequency information changes throughout time, a better way of representing the frequency information is by separating the signal into smaller parts in time. This separation is can be achieved with overlapping frames. The frames of the signal, are then converted into frequency domain to get frequency information for each part. Thus, the whole process is akin to a moving FFT of the signal and the output has time and frequency information together. This way of processing a signal is usually called Short Time Fourier Transform (STFT) and the result is called a periodogram. A block diagram for STFT is given in Fig. 1.

3.2.1 Frame Blocking

In this step, the continuous speech signal is blocked into frames of N samples, with adjacent frames being separated by $M (M < N)$. The first frame consists of the first N samples. The second frame begins M samples after the first frame, and overlaps it by $N - M$ samples and so on. This process continues until all the speech is accounted for within one or more frames. Typical values for N and M are $N = 256$ (which is equivalent to 30 msec windowing and facilitate the fast radix-2 FFT) and $M = 100$.

3.2.2 Windowing

The next step in the processing is to window each individual frame so as to minimize the signal discontinuities at the beginning and end of each frame. This is not a core part of STFT but it improves the results significantly. The concept here is to minimize the spectral distortion by using the window to taper the signal to zero at the beginning and end of each frame. If we define the window as $w[n], 0 \leq n \leq N - 1$, where N is the number of samples in each frame, then the result of windowing is the signal:

$$y_l[n] = x_l[n]w[n], \qquad 0 \leq n \leq N. \tag{2}$$

For this purpose, typically the Hamming window is used, which has the form (see `scipy.signal.hamming`):

$$w(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N-1}\right) \tag{3}$$

3.2.3 Fast Fourier Transform

The next processing step is converting each frame of N samples from the time domain into the frequency domain using FFT. This creates a frequency/feature vector of length N for each frame. We again consider only the magnitude part of FFT. Since we only consider the magnitude part, the output is the square root of the spectrogram of our signal. (*See the relationship between STFT and spectrogram in this StackOverflow question.*)

3.3 Mel-frequency cepstrum coefficients processor

A block diagram of the structure of an MFCC processor is given in Fig. 2. The speech input is typically recorded at a sampling rate above 10 kHz. This sampling frequency was chosen to minimize the effects of aliasing in the analog-to-digital conversion. These sampled signals can capture all frequencies up to 5 kHz, which cover most energy of sounds that are generated by humans. As been discussed previously, the main purpose of the MFCC processor is to mimic the behavior of the human ears. In addition, rather than the speech waveforms themselves, MFFC's are shown to be less susceptible to mentioned variations.

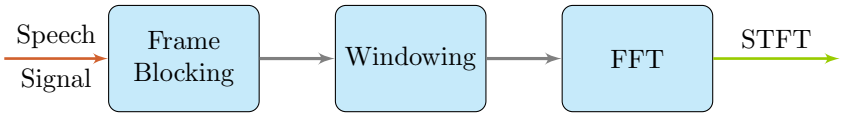

Figure 1: Block Diagram of Short Time Fourier Transform.

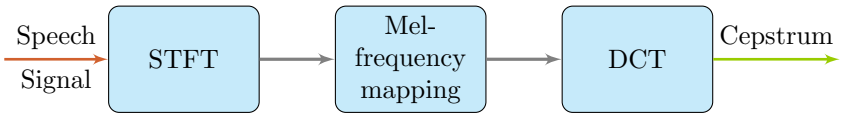

Figure 2: Block Diagram of Mel-frequency pre-processing.

3.3.1 Mel-frequency Wrapping

As mentioned above, psychophysical studies have shown that human perception of the frequency contents of sounds for speech signals does not follow a linear scale. Thus for each tone with an actual frequency, f, measured in Hz, a subjective pitch is measured on a scale called the 'mel' scale. The mel-frequency scale is a linear frequency spacing below 1000 Hz and a logarithmic spacing above 1000 Hz.

One approach to simulating the subjective spectrum is to use a filter bank, spaced uniformly on the mel-scale. That filter bank has a triangular bandpass frequency response, and the spacing as well as the bandwidth is determined by a constant mel frequency interval. The number of mel spectrum coefficients, K, is typically chosen as 20. Note that this filter bank is applied in the frequency domain, thus it simply amounts to applying the triangle-shape windows as shown in the figure above to the spectrum, to the STFT of the signal. A useful way of thinking about this mel-wrapping filter bank is to view each filter as a histogram bin (where bins have overlap) in the frequency domain. See `melfilter` function on Pytorch[1]. This function will create the triangular shape filters in the frequency domain. You need to filter each frame with each one of these filters to obtain the mel frequency cepstrum coefficients (MFCC) as discussed below.

[1]You can also use the melfilter provided by `librosa` library or create your own function based on the source codes of these libraries.

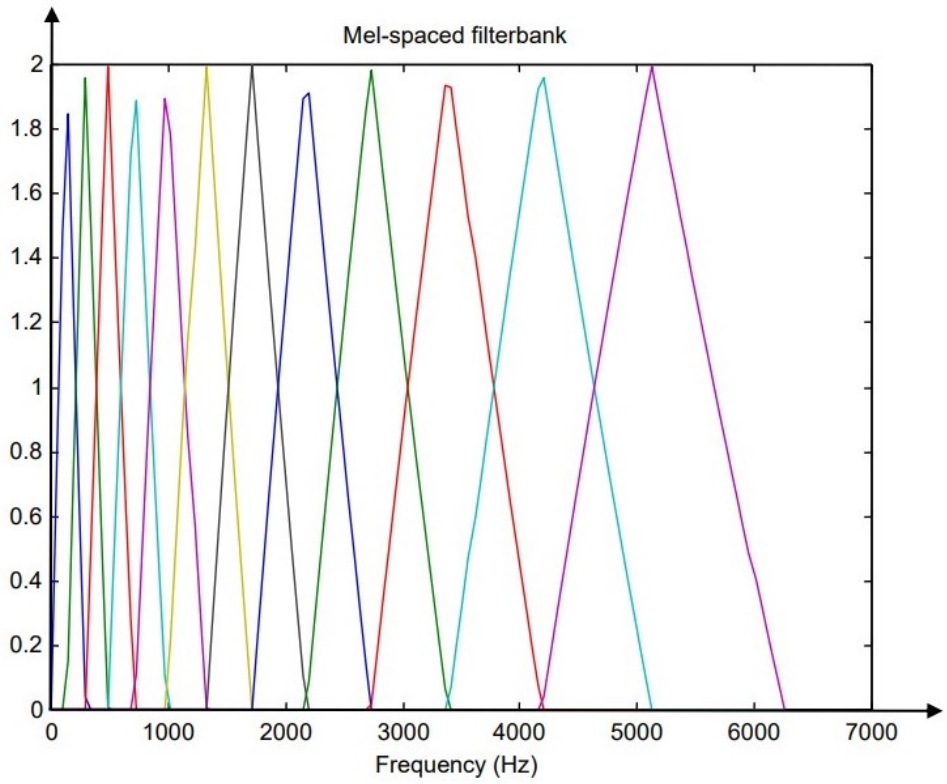

Figure 3: An example of mel-spaced filterbank.

3.3.2 Cepstrum

In this final step, we convert the log mel spectrum back to time. The result is called the mel frequency cepstrum coefficients (MFCC). The cepstral representation of the speech spectrum provides a good representation of the local spectral properties of the signal for the given frame analysis. Because the mel spectrum coefficients (and so their logarithm) are real numbers, we can convert them to the time domain using the Discrete Cosine Transform (DCT). Therefore if we denote those mel power spectrum coefficients that are the result of the last step as $\tilde{S}_k$, $k \in \{1, 2, \ldots, K-1\}$, where K is the number of filters, we can calculate the MFCC's, $\tilde{c}_n$, as:

$$\tilde{c}_n = \sum_{k=1}^{K} \log(\tilde{S}_k) \cos\left(n\left(k - \frac{1}{2}\right)\frac{\pi}{K}\right), \qquad n \in \{1, 2, \ldots, K-1\}. \tag{4}$$

Note that we exclude the first component, $\tilde{c}_0$, from the DCT since it represents the mean value of the input signal, which carries little speaker specific information.

By applying the procedure described above, for each speech frame of around 30 ms with overlap, a set of mel-frequency cepstrum coefficients is computed. These are the results of a cosine transform of the logarithm of the short-term power spectrum expressed on a mel-frequency scale. This set of coefficients is called an acoustic vector. Therefore each input utterance is transformed into a sequence of acoustic vectors. In the next section we will see how those acoustic vectors can be used to represent and recognize the voice characteristic of the speaker.

4 Feature Matching

The goal of speaker recognition is to classify the different speech samples into one of a number of categories or classes. The objects of interest are sequences of acoustic vectors that are extracted from an input speech

using the techniques described in the previous section. The classes here refer to individual speakers. Since the classification procedure in our case is applied on extracted features, it can be also referred to as feature matching. Furthermore, if there exists some set of patterns that the individual classes of which are already known, then one has a problem in supervised pattern recognition. These patterns comprise the training set and are used to derive a classification algorithm. The remaining patterns are then used to test the classification algorithm; these patterns are collectively referred to as the test set. If the correct classes of the individual patterns in the test set are also known, then one can evaluate the performance of the algorithm.

4.1 VQ Based Feature Matching

The state-of-the-art in feature matching techniques used in speaker recognition include Dynamic Time Warping (DTW), Hidden Markov Modeling (HMM), and Vector Quantization (VQ). In this project, the VQ approach will be used for STFT and MFCC based features, due to ease of implementation and high accuracy. VQ is a process of mapping vectors from a large vector space to a finite number of regions in that space. Each region is called a cluster and can be represented by its center called a codeword. The collection of all codewords is called a codebook.

Fig. 4 shows a conceptual diagram to illustrate this recognition process. In the figure, only two speakers and two dimensions of the feature space are shown. The circles refer to the feature vectors from the speaker 1 while the triangles are from the speaker 2. In the training phase, using the clustering algorithm described in Section 4.2, a speaker-specific VQ codebook is generated for each known speaker by clustering his/her training feature vectors. The result codewords (centroids) are shown by black circles and black triangles for speaker 1 and 2, respectively. The distance from a vector to the closest codeword of a codebook is called a VQ-distortion. In the recognition phase, an input utterance of an unknown voice is "vector-quantized" using each trained codebook and the total VQ distortion is computed. The speaker corresponding to the VQ codebook with smallest total distortion is identified as the speaker of the input utterance.

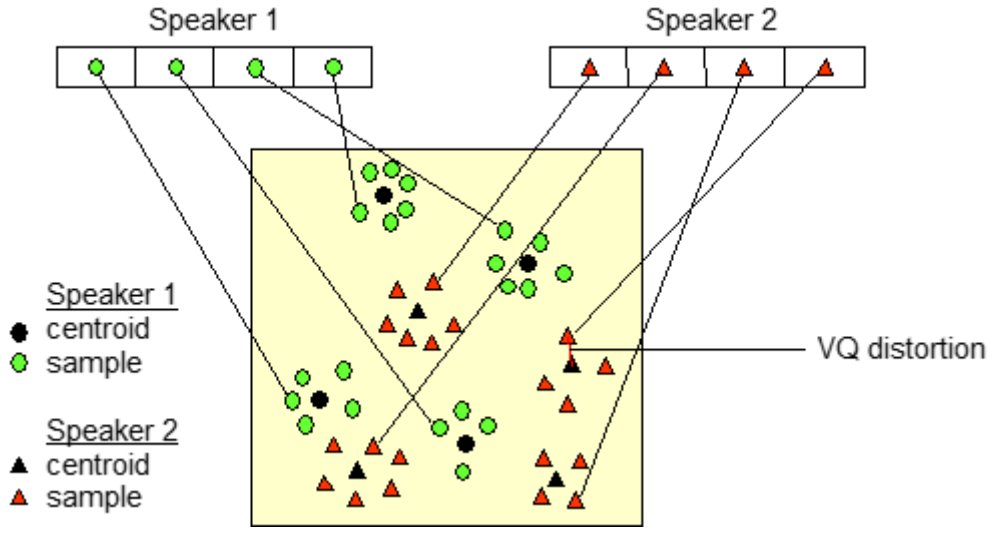

Figure 4: Conceptual diagram illustrating vector quantization codebook formation. One speaker can be discriminated from another based of the location of centroids. (Adapted from Song et al., 1987)

4.1.1 Clustering the Training Vectors

After the enrolment session, the feature vectors extracted from input speech of each speaker provide a set of training vectors for that speaker. As described above, the next important step is to build a speaker-specific VQ codebook for each speaker using those training vectors. There is a well-known algorithm, namely LBG algorithm [Linde, Buzo and Gray, 1980], for clustering a set of L training vectors into a set of M codebook vectors. The algorithm is formally implemented by the following recursive procedure:

1. Design a 1-vector codebook; this is the centroid of the entire set of training vectors (hence, no iteration is required here).

2. Double the size of the codebook by splitting each current codebook $\mathbf{y}_n$ according to the rule:

$$\mathbf{y}_n^+ = \mathbf{y}_n(1+\epsilon)$$
$$\mathbf{y}_n^- = \mathbf{y}_n(1-\epsilon)$$

   where n varies from 1 to the current size of the codebook, and ϵ is a splitting parameter (we choose $\epsilon = 0.01$).

3. Nearest-Neighbor Search: for each training vector, find the codeword in the current codebook that is closest (in terms of similarity measurement), and assign that vector to the corresponding cell (associated with the closest codeword).

4. Centroid Update: update the codeword in each cell using the centroid of the training vectors assigned to that cell.

5. Iteration 1: repeat steps 3 and 4 until the average distance falls below a preset threshold

6. Iteration 2: repeat steps 2, 3 and 4 until a codebook size of M is designed.

Intuitively, the LBG algorithm designs an M-vector codebook in stages. It starts first by designing a 1-vector codebook, then uses a splitting technique on the codewords to initialize the search for a 2-vector codebook, and continues the splitting process until the desired M-vector codebook is obtained.

Fig. 5 shows, in a flow diagram, the detailed steps of the LBG algorithm. "Cluster vectors" is the nearest-neighbor search procedure which assigns each training vector to a cluster associated with the closest codeword. "Find centroids" is the centroid update procedure. "Compute distortion" sums the distances of all training vectors in the nearest-neighbor search so as to determine whether the procedure has converged.

For FFT based features, however, this VQ based approach is not applicable and finding the centroid vectors of each class, i.e. the first block in Fig. 5 is enough for training as FFT based features are vectors, rather than a set of vectors.

5 Project

As stated before, in this project we will experiment with the building and testing of an automatic speaker recognition system. D2L Project folder provides a test database and some functions to ease the development process. You can find two main functions: `train` and `test`. Download all of these files from the project web page into your working folder. These files need to be thoroughly understood. Your tasks are to write missing parts for feature extraction and matching, which will be called from the given main functions. In order to accomplish that, follow each step in this section carefully and check your understanding by answering all the questions.

5.1 Speech Data

Download the ZIP file of the speech database from the project D2L page. After unzipping the file correctly, you will find two folders, TRAIN and TEST, each contains 8 files, named: S1.WAV, S2.WAV, . . . , S8.WAV; each is labeled after the ID of the speaker. These files were recorded in Microsoft WAV format. In Windows systems, you can listen to the recorded sounds by double clicking into the files.

Our goal is to train a voice model (or more specific, a VQ codebook in the MFCC vector space) for each speaker S1 - S8 using the corresponding sound file in the TRAIN folder. After this training step, the system would have knowledge of the voice characteristic of each (known) speaker. Next, in the testing phase, the system will be able to identify the (assumed unknown) speaker of each sound file in the TEST folder.

Question 1: *Play each sound file in the* `data/train` *folder. Can you distinguish the voices of the eight speakers in the database? Now play each sound in the* `data/test` *folder in a random order without looking at the file name (pretending that you do not known the speaker) and try to identify the speaker using your*

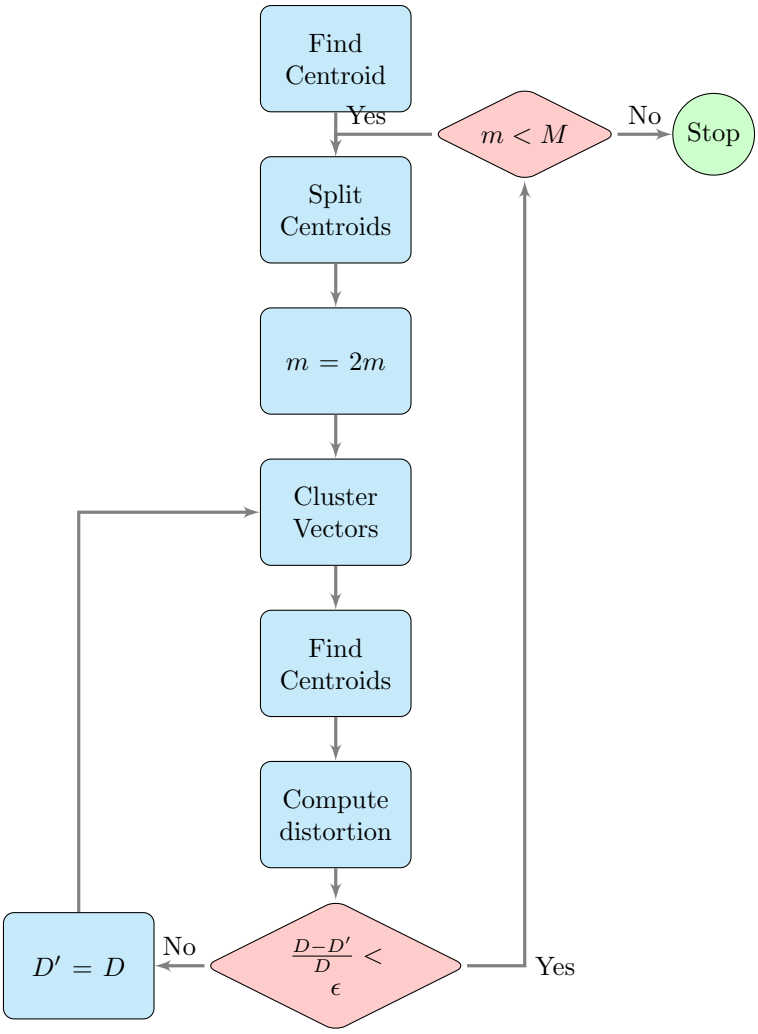

Figure 5: Flow diagram of LBG algorithm.

knowledge of their voices that you just learned from the `data/train` *folder. This is exactly what the computer will do in our system. What is your (human performance) recognition rate? Record this result so that it could be later on compared against the performance of our system.*

5.2 Speech Processing

In this phase you are required to write a function that reads a sound file and turns it into an array of FFT, a sequence of STFT, or a sequence of MFCC (acoustic vectors) using the speech processing steps described previously. Many of those tasks are already accomplished by standard functions in `scipy.signal` library. The functions that you might need are: `wave`[2], `scipy.signal.hamming`, `scipy.fft.fft`, `scipy.signal.stft`, `scipy.fft.dct` and `melfilter`.

Question 2: *Read a sound file. Check it by playing the sound file using* `scipy.io`*. What is the sampling rate? What is the highest frequency that the recorded sound can capture with fidelity? With that sampling rate, how many milliseconds of actual speech are contained in a block of 256 samples? Plot the signal to view it in the time domain. Convert it into frequency domain using* `scipy.fft.fft` *and plot the frequency information. It should be obvious that the raw signal in the time or frequency domain has a very high amount of elements and it is difficult to analyze the voice characteristic. Now cut the speech signal (a vector) into*

[2]You can utilize `scipy.io.wavfile` as well.

frames with overlap (refer to the frame section in the theory part). The result is a matrix where each column is a frame of N samples from original speech signal. Apply the steps "Windowing" and "FFT" to transform the signal into the frequency domain.

Question 3: *After successfully running the preceeding process, what is the interpretation of the result? Compute the power spectrum and plot it out using the* `plt.matshow` *command. Note that it is better to view the power spectrum on the log scale. Locate the region in the plot that contains most of the energy. Translate this location into the actual ranges in time (ms) and frequency (in Hz) of the input speech signal.*

Question 4: *Compute and plot the power spectrum of a speech file using different frame size: for example* $N = 128$*, 256 and 512. In each case, set the frame increment* M *to be about* $N/3$*. Can you describe and explain the differences among those spectra?*

The last step in speech processing is converting the power spectrum into mel-frequency cepstrum coefficients. The function `melfilter` can be useful for this task.

Question 5: *Check the link given above for more information about this function. Follow the tutorial to plot out the mel-spaced filter bank. What is the behavior of this filter bank? Compare it with the theoretical part.*

Question 6: *Compute and plot the spectrum of a speech file before and after the mel-frequency wrapping step. Describe and explain the impact of the* `melfilter` *program.*

Finally, complete the "Cepstrum" step and put all pieces together into a single function, which performs the preprocessing. For this part, it is always recommended to have a single function for preprocessing, which in turn calls other functions depending on the feature type you need to work with, such as STFT or MFCC. See `preprocess` function in the project folder for an example.

5.3 Vector Quantization

The result of the last section is that we transform speech signals into vectors in an acoustic space. In this section, we will apply the VQ-based pattern recognition technique to build speaker reference models from those vectors in the training phase and then can identify any sequences of acoustic vectors uttered by unknown speakers (see the paper 'Speaker Recognition Using MFCC and Vector Quantization' by Nijhawan and Soni, 2014).

Question 7: *To inspect the feature space (STFT or MFCC vectors) we can pick any two dimensions (say the 5th and the 6th) and plot the data points in a 2D plane. Use acoustic vectors of two different speakers and plot data points in two different colors. Do the data regions from the two speakers overlap each other? Are they in clusters?*

Now write a function, `vqlbg`, that trains a VQ codebook with 8 classes corresponding to the 8 speakers using the LGB algorithm described before. Use the utility function `scipy.spatial.distance.cdist` to compute the pairwise Euclidean distances between the codewords and training vectors in the iterative process.

5.4 Simulation and Evaluation

Now is the final part! Use the two supplied programs: train and test to simulate the training and testing procedure in a speaker recognition system, respectively.

Question 8: *What is recognition rate our system can perform? Compare this with the human performance. For the cases that the system makes errors, re-listen to the speech files and try to come up with some explanations.*