# Speech Attendance System

# Attendance recognition using python

Sagar Makar

BT18GEC321

**INTRODUCTION:**

**Definition**:

Speaker attendance system is identification of a person from characteristics of his or her voices and mark him present or absent matching him or her to the test data containing voices of all. The term voice recognition can refer to speaker recognition or speech recognition.

**Types of speech recognition:**

* Text Dependent -

If the text must be the same for enrolment and verification this is called text-dependent recognition. In such systems it can either be common across all speakers. A common pass phrase or unique. In addition, the use of shared-secrets like password or pins or knowledge-based information can be employed in order to create a multi-factor verification.

* Text Independent -

Such systems are most often used for speaker identification as they require very little if any cooperation by the speaker. In the case the text during enrolment and test are different. In fact, the enrolment may happen without user’s knowledge, as in case for many forensic applications. As text independent technologies don’t compare what was said at enrolment and verification, verification applications tend to also employ speech recognition to determine what the user is saying at the point of authentication.

* Basics -

Next to speech recognition, there is we can do with sound fragments. While speech recognition focuses on converting speech (spoken words) to digital data, we can also use fragments to identify the person who is speaking. This is also known as voice recognition. Every individual has different characteristics when speaking, caused by differences in anatomy and behavioural patterns. Speaker verification and speaker identification are getting more attention in this digital age. For example, a home digital assistant can automatically detect which person is speaking.

**Methodology**

**How:**

ASR is done by extracting MFCCs and LPCs from each speaker and then forming a speaker-specific codebook of the same by using Vector Quantization (I like to think of it as a fancy name for NN-clustering). After that, the system is trained and tested for 8 different speakers.

Create virtualenv with:

virtualenv -p python3 .env

. .env/bin/activate

pip install -r requirements.txt

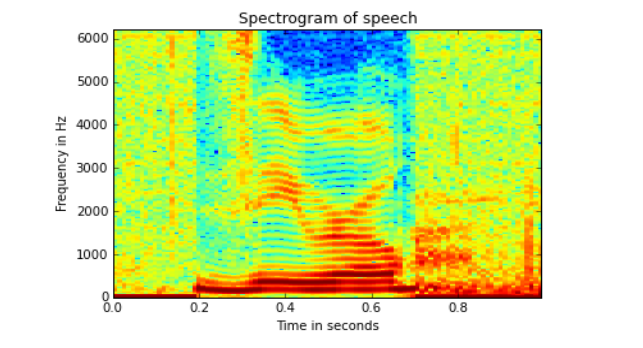
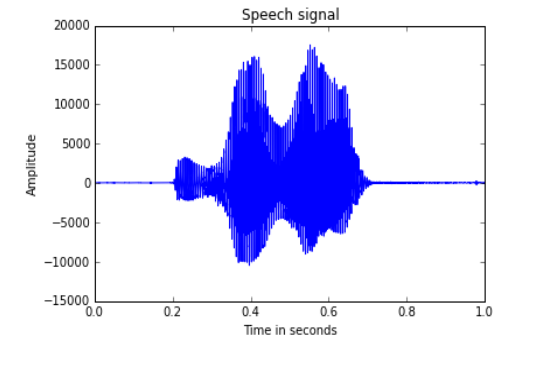
To test the algorithm, run test.py. Certain parameters are open to be changed, such as the order of LPC coefficients, the number of Mel filterbanks and the number of centroids in each codebook.

**Repository used:**

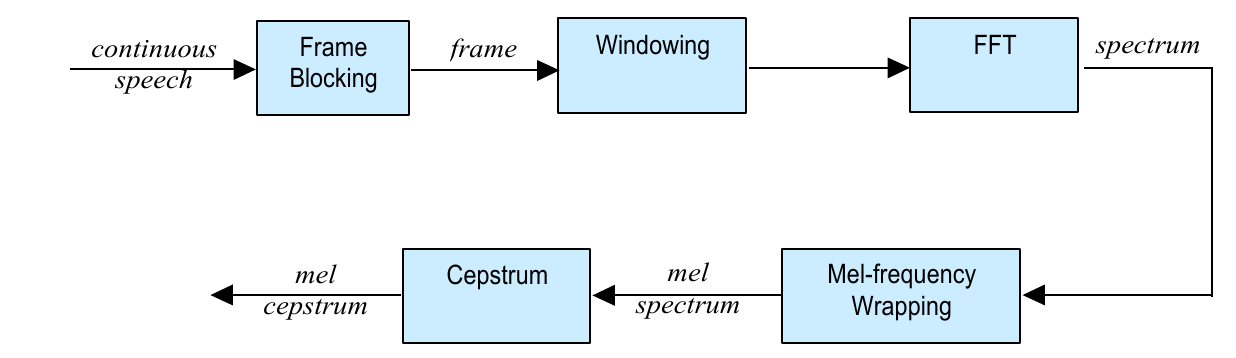
numpy

scipy

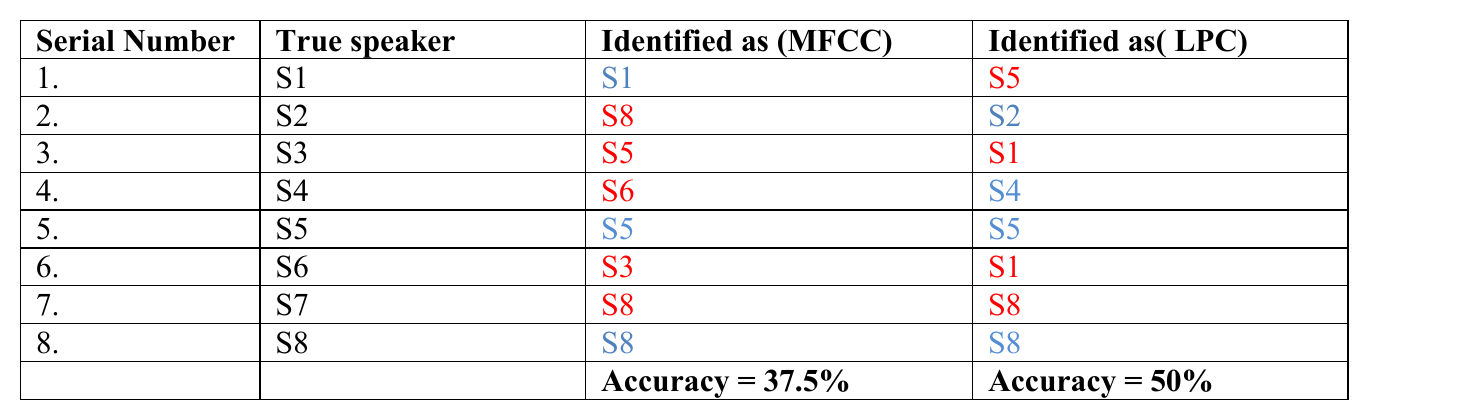
matplotlib

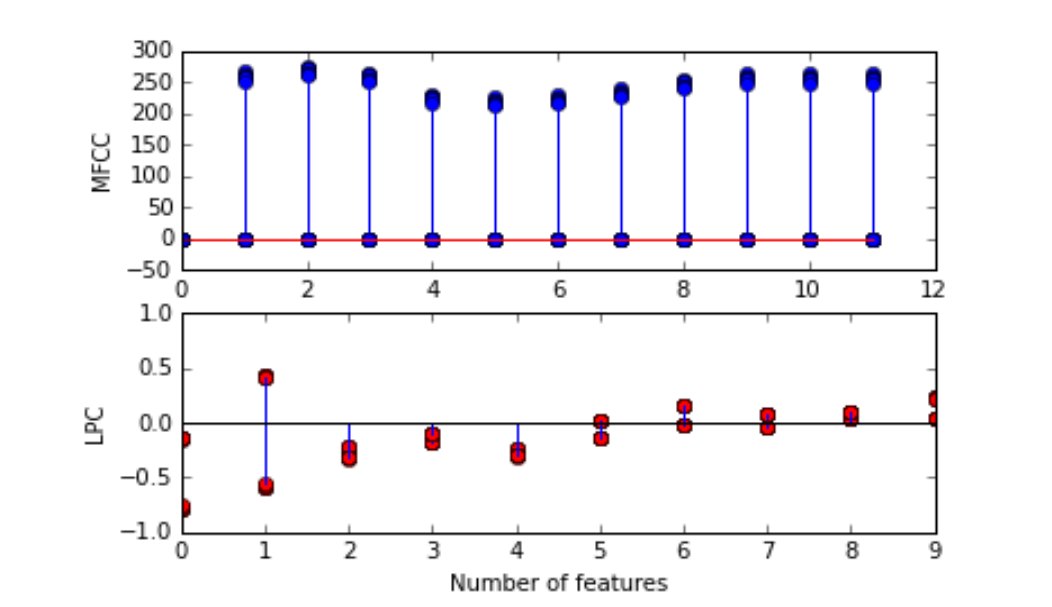


**MFCC Calculation schematic:**

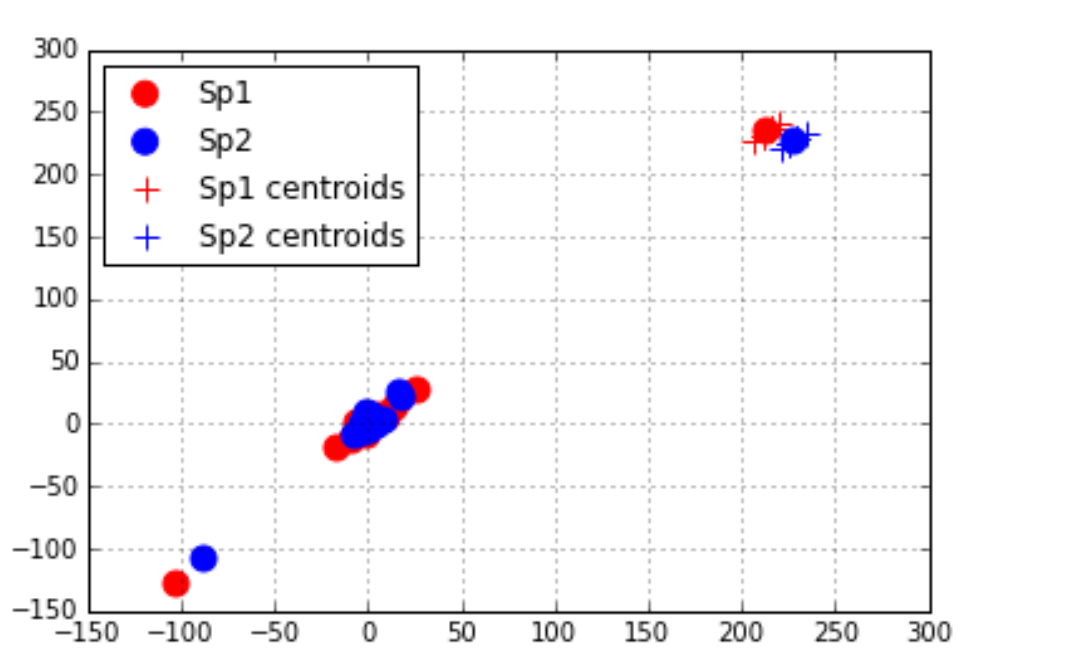


**Accuracy Percentage:**

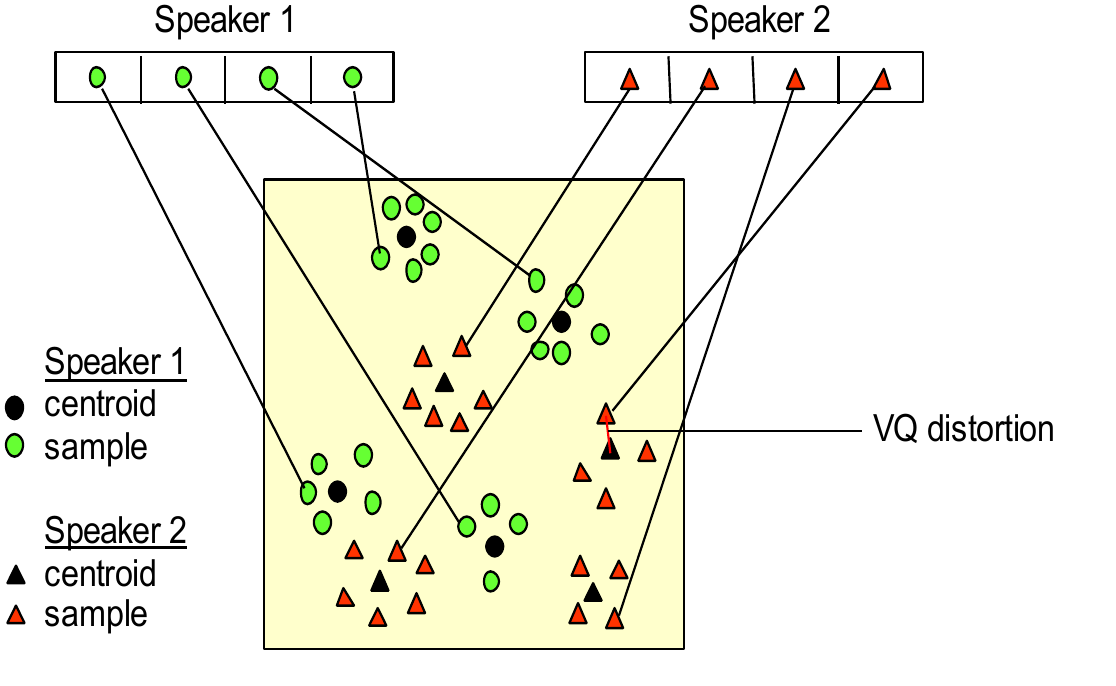


**Codebooks for LPC and MFCC:** 

**Codebooks for two speakers:**



**Conceptual Codebooks:**



**Mel-Frequency Cepstral Coefficients:**

Human hearing is not linear but logarithmic in nature. This implies that our ear acts as a filter. 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. The relationship between frequency in Hz and frequency in Mel scale is given by: m= 1125 ln(1 + f/700), f = 700(e^(m/1125) -1 )

To calculate MFCCs, the steps are as follows.

The speech signal is divided into frames of 25ms with an overlap of 10ms. Each frame is multiplied with a Hamming window. The periodogram of each frame of speech is calculated by first doing an FFT of 512 samples on individual frames, then taking the power spectrum as: P(k) = 1/N (S(k))^2

Where P(k) refers to power spectral estimate and S(k) refers to Fourier coefficients for the kth frame of speech and N is the length of the analysis window. The last 257 samples of the periodogram are preserved since it is an even function.

The entire frequency range is divided into ‘n’ Mel filter banks, which is also the number of coefficients we want. ‘For ‘n’ = 12, the filter bank is shown in Figure 3 - a number of overlapping triangular filters with increasing bandwidth as the frequency increases. To calculate filter bank energies we multiply each filter bank with the power spectrum, and add up the coefficients. Once this is performed we are left with ‘n’ numbers that give us an indication of how much energy was in each filter bank. We take the logarithm of these ‘n’ energies and compute its Discrete Cosine Transform to get the final MFCCs.

**Linear Prediction Coefficients:**

LPCs are another popular feature for speaker recognition. To understand LPCs, we must first understand the Autoregressive model of speech. Speech can be modelled as a pth order AR process, where each sample is given by: x(n) = k=1∑p a\_k x(n-k) + u(n)

Each sample at the nth instant depends on ‘p’ previous samples, added with a Gaussian noise u(n). This model comes from the assumption that a speech signal is produced by a buzzer at the end of a tube (voiced sounds), with occasional added hissing and popping sounds.

LPC coefficients are given by α. To estimate the coefficients, we use the Yule-Walker equations. It uses the autocorrelation function Rx. Autocorrelation at lag l is given by: R(l) = n=1∑N x(n)x(n-l)

While calculating ACF in Python, the Box-Jenkins method is used which scales the correlation at each lag by the sample variance so that the autocorrelation at lag 0 is unity.

**LBG Algorithm:**

The LBG algorithm [Linde, Buzo and Gray], is used 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).

y + n = yn ( 1 +e )

y - n = yn ( 1 - e )

where n varies from 1 to the current size of the codebook, and e is a splitting parameter.

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.

5.6.Iteration 1: repeat steps 3 and 4 until vector distortion for current iteration falls below a fraction of the pervious iteration’s distortion. This is to ensure that the process has converged.

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.

**It is then followed by Feature training and Testing.**