Project: speaker classification

Signal Processing

Nov-Dec 2021

Introduction

Machine learning consists of building systems based on data. These types of systems are very popular nowadays. The goal of this project is to give you an intuition of the advantages of machine learning-based systems. You are going to implement a rule-based system using signal processing techniques and compare it to machine learning techniques. The subject of this project is an application that classifies speakers, which means that it recognizes different speakers. Your application will base its classification on the voices of the speakers. Speaker classification is different from speech recognition whose goal is to recognize what is being said by a speaker.

Speech production

Speech is composed of voiced (ex: a, o, i, e,...) and unvoiced (ex: f, h,...) sounds. The glottis (cf. Figure 1) either opens or closes while air is sent from the lungs. On the one hand, when the glottis is open, the air passes through it and then through the vocal tract producing unvoiced sounds. In this case, the air sent from the glottis can be represented as white noise and the glottis as an all-pole filter of which the parameters depend on the shape of the vocal tract. The latter filters the white noise to produce one of the unvoiced sounds. The parameters of this filter change for every new unvoiced sound. On the other hand, when the glottis is closed, it vibrates when the air arrives from the lungs and a signal similar to a Dirac comb is produced. This signal is also filtered by the vocal tract the way than before. This analogy of speech production to signals filtered is called *source-filter model*.

Rule-based systems

The rule-based systems rely on rules set by the developers in the form of hand-coded scripts using IF-ELSE conditions to obtain an outcome or take a decision. In this project, you will have to set some conditions that will be checked in the incoming speech to determine to which speaker it belongs to.

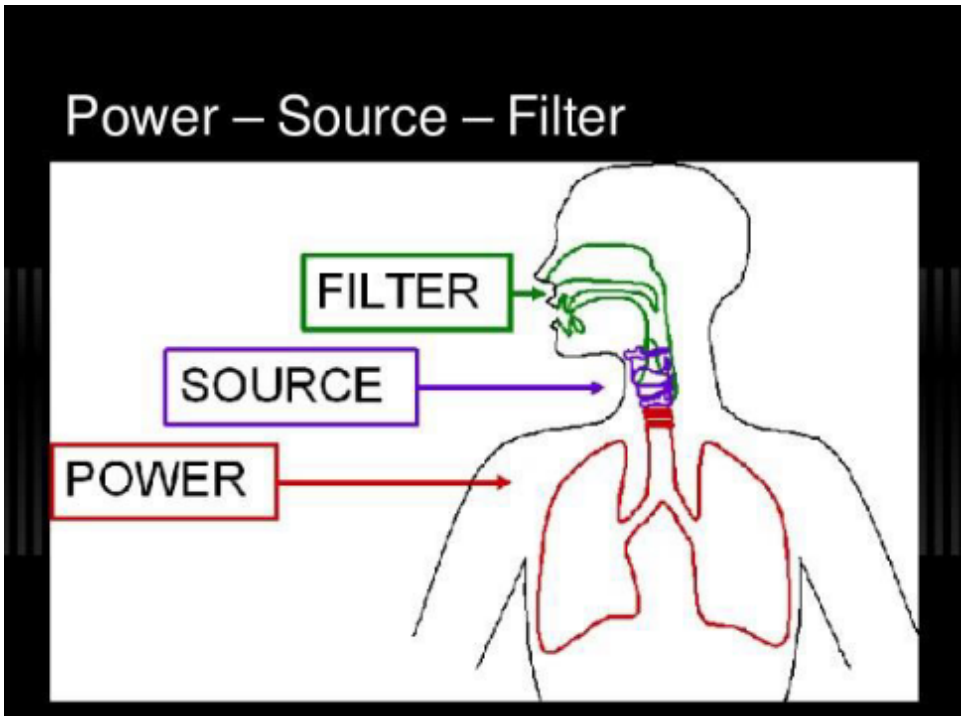


Figure 1: Schematic of speech production

To do so, **features** will be extracted from the speech signals. You will then analyze each of these features in order to define some rules with the aim to recognize different speakers. Among the mostly used features in speech processing, you will focus on:

- the signal's energy,
- the fundamental frequency,
- the Mel-Frequency Cepstrum Coefficients ($MFCC$),
- the Formant frequencies.

Database

For this project, you will use data from a database of recorded sentences, the **CMU-Artic DB** with speakers BDL (male) and SLT (female) available on this link. The data used to evaluate your system is usually different from the data used to build it. Indeed, the goal is to obtain accurate results with new and unseen data. If you were to use data the system was built upon, your results would have a certain bias and so the evaluation would give a fake score. Such system would have a high chance to perform poorly in runtime.

Procedure

In what follows, you will divide your project into several functions put in a single python file (referred to as module) and name this module UTILS.PY. Create a second module named MAIN.PY in which you will import your functions and use them, build your project and do your analysis as required later.

You are also asked to test each of the functions you build before using it. In this project, you have to verify that the outputs go along with your predicted results when injecting a specific input. For this, create a testing script for each function that you will save under the name <**function_name**>**_test.py**.

Your report will contain the analysis and interpretations required (which involve answering the questions asked here, analyzing and interpreting the results obtained and the tests you've made). Every result you obtain needs to be analyzed and interpreted even if you are not explicitly asked to do so (the important observations and interpretations are enough).

Useful Python libraries

Since Python has a lot of useful libraries, some of them have functions that you might want to use in order to implement this project. You can find the inputs and outputs as well as examples of each functions in the documentation available online.

- AUDIOFILE: contains the function **read** that allows one to read a wav file in Python.
- GLOB: contains the function **glob** that allows one to save multiple folder and/or file paths in the same list.
- RANDOM: contains the function **sample** that allows one to extract randomly the indexes of n utterances from a list.

1 Signal preprocessing

Speech data in a database can be messy or not organized. They need therefore some preprocessing before being used. For instance, they need to put in a certain format that fits the targeted application of filtered to remove part of a constant noise present in all of them.

1. NORMALIZATION: The signal's sample values are scaled between a certain range (in this case, we'll scale it between -1 and 1). **Write a function that would take a signal as an input and output a signal with value in range [-1, 1].**

2. SPLIT:: The speech segment is split in (non-)overlapping *frames*. Indeed, in order to obtain values that represents signals as precisely as possible, we need to extract features from slices (referred to as *frames*) of the signal. **Write a function that would take a signal and output a list of frames. See this operation as a certain size window sliding from samples to samples. According to the size of window and the overlapping expected, each step will output a new frame. The parameters of the function should be the width of the window (in [ms]), the sliding step (in [ms]), the sampling frequency and any other parameter you might need.**

2 Features extraction

This section gives the description of the features extraction algorithms that you have to implement in Python.

2.1 Signal energy

The signal energy is computed by applying:

$$E = \sum_i |x(i)|^2 \tag{1}$$

with i the indices of the elements of the signal x

1. **Write a function that would output the energy of a given signal.**

2.2 Pitch

The fundamental frequency f_0 is the lowest frequency that constitute a signal. It represent the source signal of a source-filter model. Its values range from 60 Hz for deep to 500 Hz for acute voices. The term pitch is related to subjective psychoacoustic perception. Voiced sounds contain pitch and therefore have a non-zero f_0 value unlike unvoiced sounds.

2.2.1 Autocorrelation-based pitch estimation system

In the section you will implement the autocorrelation-based algorithm to estimate the pitch. Since f_0 is the lowest frequency contained in a voiced signal, the fundamental period can be estimated by first computing the autocorrelation of the signal in the temporal domain and then measuring the distance between the highest peaks. This estimated fundamental period is the inverse of the f_0. Implement the following algorithm:

1. **Randomly select 5 utterances from each speaker**,
2. **normalize the signal and display it**,

3. **split the signal into 50 ms long frames**,
4. **compute the energy of each frame**,
5. **for each frame compare its energy to a threshold you will determined graphically**[1],
6. **for the voiced segment, compute the autocorrelation of the signal using the xcorr function in xcorr.py with a maxlag equal to 100**,
7. **find the distance between the two first peaks and estimate f_0**[2] **while for the unvoiced segments, $f_0 = 0Hz$.**

2.2.2 Cepstrum-based pitch estimation system

In this section, you will implement the cepstrum-based algorithm to estimate the pitch. By definition, a cepstrum results from the Inverse Fourier Transform (IFT) of a signal's log spectrum. Implement the following algorithm:

1. **Randomly select 5 utterances from each speaker**,
2. **normalize the signal and display it**,
3. **split the signal into 50 ms long frames**,
4. **compute the energy of each frame**,
5. **for each frame compare its energy to a threshold you will determined graphically**,
6. **on the voiced segments, compute the cepstrum. To obtain the power cepstrum, first compute the frequency response of the signal, then calculate the logarithmic value and finally its IFT** [3] **such as:**

$$C_p = F^{-1}(log_{10}(F(f(t)) \tag{2}$$

[1] If the energy is greater than the threshold, consider the frame as voiced, otherwise considered it unvoiced. Such voiced/unvoiced classification is a very well-studied domain and more accurate solutions can be found, but for the sake of this lab's pitch estimation, using the energy is enough as precision is not a concern.

[2] The lowest f_0 possible is 50 Hz.

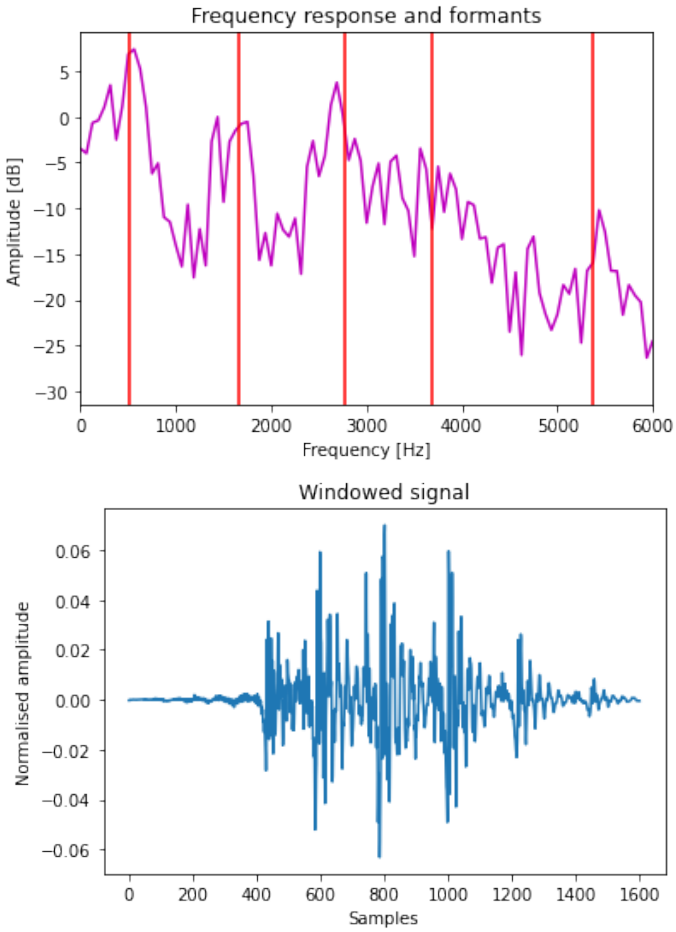
[3] The reason why it is related to the pitch is the following: the harmonics are multiples of the fundamental frequency f_0. Two consecutive harmonics are therefore separated from one another by $f_0 Hz$ in the spectral domain. The inverse FFT of the log amplitude spectrum (or log frequency response) will thus show a peak at the value corresponding to the space between these harmonics, i.e. the fundamental frequency. So once the cepstrums are computed for a certain frame, your algorithm should search for a peak value in the part of the cepstrum vector where the fundamental frequency should be (usually between 60 and 500 Hz).

7. **apply a Hamming window of the signal after comparing its energy to a threshold and before extracting the cepstra from it, and compare your results with the previous ones.**

8. **To obtain f_0 from the cepstra, find the index of the highest peak and convert it in Hz, while for the unvoiced segments $f_0 = 0Hz$.**

2.3 Formants

Formants are the peaks found in the frequency response envelope. They are therefore frequency values that represent the resonances of the vocal tract. In other terms, they characterize the way the signal is shaped at a certain instant and thus the sound produced. In order to estimate the formants, we will rely on a Linear Predictive Coding (LPC)-based algorithm. The LPC are coefficients that approximate the vocal tract as the coefficients of an all-pole filter in source-filter model. So the formants, with respect to this filter, can be seen as its poles since they are peaks at certain frequencies.



In order to implement the algorithm and estimate the formants, you are asked to follow these steps:

1. **Split the signal in frames**,
2. **pass the signal into a first order high-pass filter** [4]**. The filter follows this equation:**

$$y(n) = x(n) - \alpha x(n-1) \quad \text{with } \alpha = 0.67 \tag{3}$$

3. **apply a Hamming window onto the signal**,
4. **Extract the LPC coefficients on each windowed frame**[5]**. To determine the order of the LPC coefficients, it is common to use the rule:**

$$n_{coeff} = 2 + F_s/1000 \tag{4}$$

   but you can pick a number between 8 and 13,
5. **Find the roots of the LPC**[6],
6. Since the LPC are real values, the roots will be complex conjugate. **Keep only the roots with one of the conjugate complex numbers (either the positives or the negatives)**. Indeed, the goal is to find the angles of these number that represent the frequency values. So two complex conjugates will give the same angle and so the same frequencies.
7. **Deduce the corresponding Hz values from these complex conjugate (make sure to also sort them at the end in order to obtain increasing values so it is easier to associate them to formants:** $F_1 < F_2 < F_3$

2.4 MFCC

The MFCCs are coefficients representing the filter of a source-filter model. They therefore represent the vocal tract activities during speech production. You are asked to write an algorithm that compute the MFCCs:

1. **Pre-emphasize the signal: use the same filter equation than previously (Eq. 3) but with** $\alpha = 0.97$,
2. **split the signal into frames**,
3. **apply a Hamming window on every frame**,

[4] In order to balance the high and low frequencies of the signal, as the high frequencies usually have lower values and in order to improve the SNR as part of the noise is in the low frequencies.

[5] By using the LPC function from the file SCIKIT_TALKBOX_LPC.PY

[6] Which should represent the formants.

4. **compute the power spectrum of the signal** (*periodogram*):

$$P = \frac{|\mathbf{fft}(x(t))|^2}{N_{DFT}} \quad \text{with } N_{DFT} = 512. \tag{5}$$

5. **Transform the power spectrum's scale by passing it through a Mel-Filter Bank[7]. Use the function available in filterbanks.py to output the filter bank value with the power spectrum frames as input.**

6. As the filter bank coefficients begin very correlated, **you have to apply a Discrete Cosine Transform (DCT)[8] on the filter bank coefficients which will output the MFCC vectors.** The parameters of the dct function should be as follows:

$$dct(filter\_banks, norm =' ortho')$$

7. In general, only the first 13 values are kept from the MFCC list[9]. **Keep only the first 13 values from the list**.

3 Building a rule-based system

We will now analyze each of the features extracted from sentences of both speakers and define rules to build a speaker recognition system. From each speaker, randomly pick 15 sentences from each speaker on which the feature analysis and extraction will be made.

1. Use the algorithm you have written in the previous section to extract features for the 30 sentences. Then study and visualize the discriminatory power of these features by comparing their values for both speakers.

2. Based on what you have seen, propose a rule-based system by:

   (a) choosing a feature on which the system rely,

   (b) for each feature, state rules that might discriminate between the two speakers[10].

3. Implement in Python a rule-based system you proposed in the previous question and test it on the database.

[7] The idea behind this is to mimc the ear perception and adjust the scales calculated by the FFT.

[8] Use the dct function form SCIPY.FFT.

[9] The first is equivalent to the energy and the 12 others are the MFCCs.

[10] e.g.: if pitch $>$ 300 Hz and F1 $>$ 566 Hz $\Rightarrow$ The speaker is SLT.

4. To evaluate how well your system works, you will rely on the *accuracy* metric. This means that you will compare each output your system gives to the real class (speaker A or B). The accuracy is the percentage of well classified utterance. For instance, if the total of test sentences is 10 (distributed equally between A and B) and your system classifies correctly 3 and 4 utterances from speaker A and B respectively, then the accuracy is $7/10 = 70\%$.

5. For the final step, change the system or *fine-tune* it in order to obtain higher accuracy results. You will quickly notice that the number of possibilities to try in order to change the combinations rules/features is infinite. For the sake of this exercise you will define three systems and compare them. Finally, pick the one with the highest accuracy value.

6. Download data from other male and female speakers and adapt your code so that it classifies male vs female.

7. How well do you think your best system will work if applied on new data? Check your answer by trying to classify from the newly downloaded data.

4 Building a machine learning-based system

Nowadays, applications need to be robust and algorithms need to be generalized enough to be efficient with all the data distribution possible. Machine learning consists of tuning the parameters or *weights* of a system for a specific task. Data relevant for this task is used to *train* the system to do the required task. Using the same format as before, you are asked to explore different machine learning algorithms and report a comparison of their results while comparing them also with the rule-based system previously implemented.