UMONS
Université de Mons

Faculté Polytechnique

Signal processing:

How useful is machine learning ? A speaker classification project

Third year engineering, IT section

ALI Hind
MÉLICE Océane

POLYTECH MONS

Under the guidance of:
Kevin EL HADDAD
Thierry DUTOIT

novembre 2020

pôle hainuyer

Contents

Introduction

This report seeks to answer the question "How useful is machine learning ?" doing so by taking a look at speaker classification with the help of python. Thanks to different parts of the human body, such as the glottis or the lungs, humans can make sounds. Air passes through various organs which can be likened to power, a source and a filter. So we can say that a signal is produced. A person's speech is made up of different sounds (unvoiced and voiced).
To achieve our goal of building a rule based system allowing for user classification, we will create different functions applying them to vocal recordings of people will allow us to study features that are meaningful when it comes to recognising the speaker. Furthermore, several elements in the treatment of the voice in a general way will be approached during this project. We will see for example,the signal's energy, the fundamental frequency (F0), the Mel-Frequency, Cepstrum Coefficients (MFCC), the Formant frequencies...
All of this will be done with the help of *python*.

Chapter 1

Signal Pre-processing

In order to get proper signals that could be used is later functions pre-processing is a crucial first step given that our signals can have different range of values but also some noise. Starting with the normalization. We had to modify the signal in order to have the signal values in the same range. We were able to test our code using a fairly simple function(a sine of amplitude equal to 3). As shown in the following figure, our signal has been transformed and the amplitude is now well between -1 and 1 (orange signal).

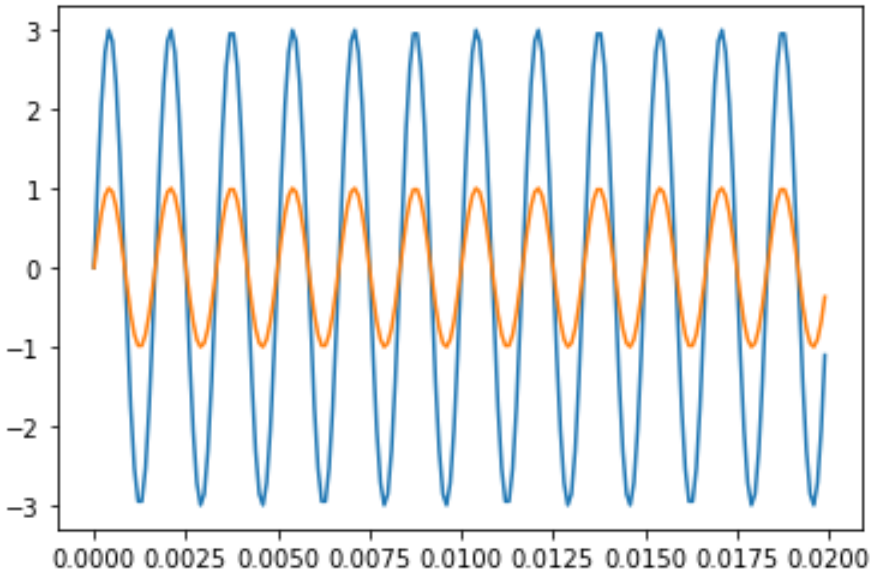

Figure 1.1: Graphic : *pre-processing* of *a sine*

The second step is splitting the signal into several frames (segment of the speech) because we need to extract features from each frame and not from the whole signal. To do this, it was necessary to take into account the step, the sampling frequency as well as the width. Indeed, it is necessary to separate the signal into several frames each having the same length. Another point to watch out for is overlapping or non-overlapping. Given that the width is supplied in milliseconds, it was important to also get the value of the sampling frequency in order to convert this values into samples and work with more ease. If the sampling step is smaller than the desired width of each frame, our frames will overlap, which is a desired effect given that it adds precision to the features we will extract from the frames later on. It is important to note that this method has also lead us to discard some information when the last bits of the signal seemed to have a width smaller than the necessary one.

Chapter 2

Features extraction algorithms

In this part, we will see different features or descriptors allowing us to create our rule-based system.

2.1 Signal energy

To compute the energy, as shown in the following formula, we take the sum of the squared value of the different values present in our signal or frame. The energy is an important feature as it allows us to determine whether a sound is voiced or unvoiced, as will be explained later on in this report.

$$E = \sum |x(i)|^2 \tag{2.1}$$

2.2 Pitch

The pitch corresponds to the lowest frequency of a signal in the spectral range, also known as the fundamental frequency. Its values ranging between 60 and 500 HZ. The pitch value for voiced sounds must be different from zero unlike the voiced unvoiced sounds which have a pitch of zero.

2.2.1 Autocorrelation-based pitch estimation system

In this section, we will discuss the implementation of an algorithm that allows us to find the distance between the highest peaks of an auto-correlation. In the time domain, this distance is equivalent to T_0 so it's possible to find f_0.
To make this algorithm, it was necessary to use the functions previously created.We start by normalizing our signal, then we separated into several frames. It shall be noted that we picked a width of 30 milliseconds, given that most our audio samples are 3 seconds long, this estimation was inspired by the example in the protocol, we then picked a step of 20 seconds, given that having an overlap ensures more accurate results. The next step was to calculate the energy for each frame.
After extracting the signal, sampling frequency and the energy of a particular sample which was randomly picked, we wanted to ensure that our graphs looked correct, so we preceded to check the profile of our signal thanks to the *Audacity* program. After the comparison we can see that two seem identical, we also checked the sampling frequency with the program and it was the identical to the one given by our algorithm. However, we realized that our scale had a problem. We believe that our results are too large, we should have values between -1 and 1

because the signal has been normalized before, and we have checked the normalization function. We noticed that when the results were displayed in the python console, they were righteously between -1 and 1. So we have a display error in our code.

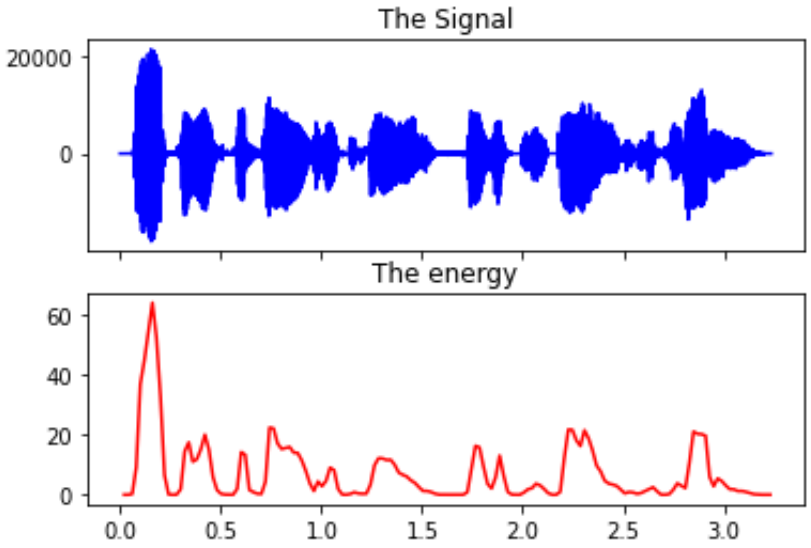

Figure 2.1: Graphic : *signal and energy* of a speaker

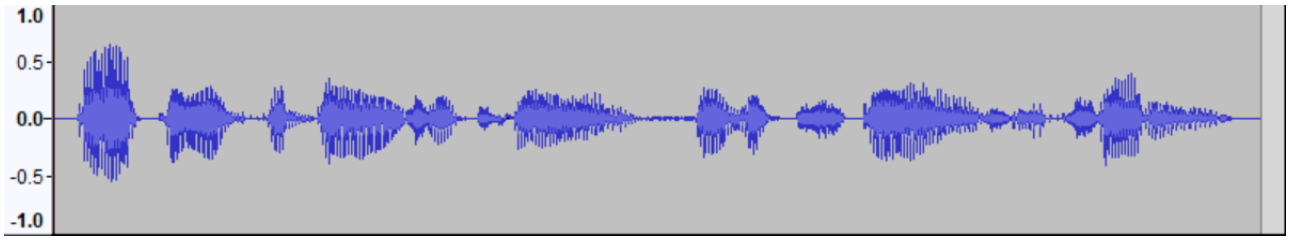

Figure 2.2: Graphic : *signal* of a speaker with *udacity*

In order to begin the actual auto-correlation, we had to separate the voiced and unvoiced frames in our signal, given that the unvoiced frames should give us a pitch of zero. For this we plotted the energy and temporal representation of five randomly selected signals. We looked on our energy subplots to find an appropriate threshold for the unvoiced sounds. This seemed to be around 6 and 8. So we decided to set our *treshold* to 7. To work with voiced sounds, we keep all the frames with an energy greater than the *treshold*(they will have a nonzero f_0).
The last step is to do the autocorrelation, which means to correlate each voiced frame with itself in order to find the distance between the peaks, often referred to as the lag. We used the *xcorr* function provided by the lab supervisor. We had to put the correct input arguments in order to use this function. We therefore had to calculate the *maxlag* beforehand. To get this value we need to find the minimum period, which in our case is the one corresponding to the minimum frequency of 50 Hz, the minimal period is therefor one over our minimal frequency, which is equal to 0.02 seconds. Then using sampling frequency (f_s) as means of a unit conversion, we can get the following maxlag ($maxlag = f_s * \frac{1}{50} = 320$) in samples.
Once we had the right maxlag, we preceded to compute the auto-correlation of each voiced frame,*xcorr* is a function which outputs the lags and the correlation values, as mentioned previously we are here looking to find the distance between the central lobe, which happens to always have the highest correlation index and thus highest peak, and which also happens

to always be at the lag position of 0, and a neighbouring peak. The next step is then to find the different peak positions of our correlation function, we however had to make sure that the distance does not exceed $max\ distance = (f_s * \frac{1}{500})$ to get coherent results, the logic behind this is similar to the one behind the *maxlag*. We then, out of curiosity tried plotting the correlation with respect to the lags, and what we noticed was that some of correlated frames only seemed to have one lobe, given that the work we're doing here is to find the distance between the central lobe and the neighbouring one it seemed necessary to remove those mono-lobbed correlation frames. Once all of that was done, and after converting our distance from frames to seconds and then talking one over that in order to find the fundamental frequency, or pitch we've been looking for. The results obtained after all these steps fall righteously between 60 and 500 HZ, which seems consistent.

2.2.2 Cepstrum-based pitch estimation system

Another method to calculate the pitch is to use the cepstrum-based algorithm. To implement this one, we first needed to create a function which selects 5 random vocals for each speaker. This function will therefore go through the entire list of available records and chose the return number of audio titles. After that another function will have to extract the data (signal, energy (per frame) and sampling frequency) from the selected signals in order for us to use them. We will start with the same functions as before. Which is to say by normalizing the signal, splitting it into frames and calculating the energy of each frame. After that, we need to get the temporal representation of the signal as well as its energy. Two example representations as shown in figures 2.3 and 2.4.
We also defined a treshold, using the same method described in the auto-correlation part (treshold =7). This allows us to separate unvoiced sound from voiced sound.
Now, we must compute the cepstrum. As defined in the protocol the cepstrum is the result of the *Fourier inverse transform* of a signal's log spectrum. Following this definition and the instructions in the protocol, we started by computing the signal's response and take its algorithmic value. Indeed, we know that f_0 is a multiple of harmonics. So we must be able to see a peak at the value corresponding to the spacing between these harmonics. This is the fundamental frequency which lies between 60 and 500 Hz. Another step is to apply a hamming window or a rectangular window to each voiced frame before calculating the energy in order to be able to compare which one gives the best results. The last step is to find the index of the highest value and use it to compute an estimation of the pitch. What we noticed while doing this work was that unfortunately our values in the cepstrum were minute, in such a way that when we tried to gather the part of the spectrum where the fundamental frequency should've been that array was returned to us empty, we therefore were not able to properly get an estimation of the pitch using this method.

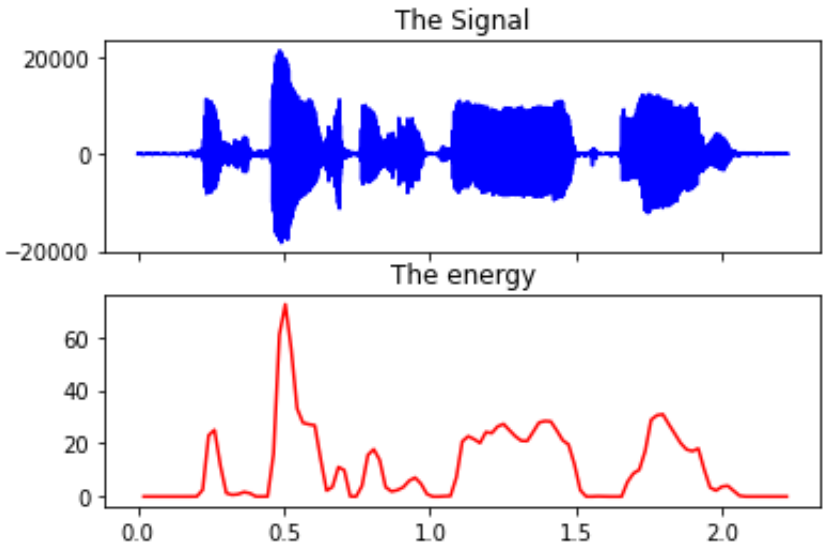

Figure 2.3: Graphic : *signal and energy* of a speaker

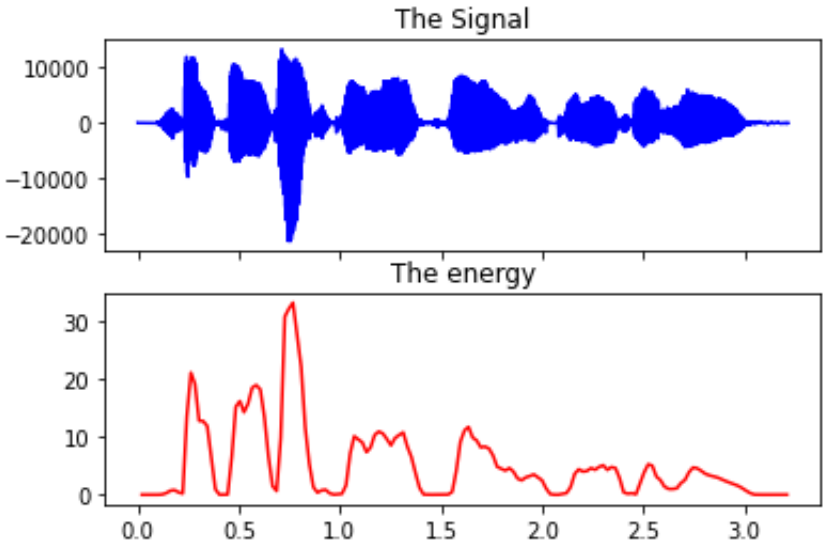

Figure 2.4: Graphic :*signal and energy* of a speaker

2.3 Formants

As defined in the protocol formants are the peaks found in the frequency response envelope. After doing some research on the internet we found out that several formants are to be found at various frequencies with an increase of around 1000Hz from one formant to the next. Which is similar to what we see in the graph shown in the protocol. According to our research it appears that different vowels have different formants, it appears that they are highly concentrated around values from 0 to around 4500Hz, yet we also see rare appearances for values near 8000Hz.

In order to extract this feature we started as usual by splitting our signal into various frames, followed by that we had to do the preamplasis step, which goes to say passing the various frames through a filter, the equation of which was given in the protocol (With the alpha parameter in our case 0.67):

$$y(t) = x(t) - \alpha x(t-1) \tag{2.2}$$

After passing it through this filter, we need to apply a hamming window to it in the same fashion as before. We also have to extract the LPC coefficients. These allow us to approximate the vocal tract. They correspond to the coefficients of the poles in our source-filter model.This step is possible thanks to the LPC function of the *scikits.talkbox* library. Unfortunately, we were unable to install it, which is why we imported the function code directly into our own code as instructed by the lab supervisor. The formants are represented by the roots of the LPC, which is why our next step was to find those roots, which were conjugated complex, we therefore had to ensure to only take one of the two parts of the conjugated complex. We made the choice to keep those whose imaginary part was positive. Doing so in order to have a phase between 0 and π as well as to avoid redundancy in our formants since two complex conjugates provide the same phase (same frequency). After this separation, it was necessary to sort them in ascending order and to convert the radians into hertz.

The values we get for the formants are more or less what we expected according to our research, which goes to say we notice that the formants we get do increase by around 1000Hz with each step and that our values appear to be in the appropriate range. With a higher concentration around 3000Hz to 4500Hz. The average value we get is around 3400Hz.

2.4 MFCC

MFCCs or Mel-Frequency Cepstral Coefficients is used to represent a person's vocal tract. For this algorithm, we will have to use a filter bank. But we must first pre-emphasize the signal in the same way as before but with an $\alpha = 0.97$. We must then separate the signal into several frames and apply a hamming window to each frame. We then have to calculate for each frame its frequency spectrum and compute the power spectrum. Using the following equation (2.3) in which NTFD=512, we can calculate the power spectrum (periodogram).

$$P = \frac{|FFT(x(t))|^2}{N_{TFD}} \tag{2.3}$$

We must pass our signal inside a Mel-filter in order to adjust the scale which is produced by the the Fourier transform. The Mel-filter imitates the human ear's perception of sounds. A function has been made available to us by the Lab supervisor to help us with this step. The outputs of this function correspond to the filter bank values. However, the outputs are highly correlated which can be a problem for the use of machine learning algorithm, so we need to flatten a *Discrete Cosine Transform* to decorrelate the filter bank. We keep 13 results of this vector because the others represent rapid changes in the filter bank coefficients and can be overlooked. The first result correspond to the energy and the others to the MFCCS. In this part we had difficulties using the command given in the protocol, when setting the axis to 1 in the dct function which allows the computation of a discrete cosine, we went with axis=-1 as suggested by the scipy documentation.
Despite our attempts to interpret the results, and use online documentation to find examples or reasonable outputs, the desired output remains nebulous, all we can confidently say is that the function we have created return an array of 13 elements as expected. The first of which is meant to be the energy and the 12 others are the MFCCs as mentioned previously.

Chapter 3

Building a rule-based system

In this part we are going to use the features extracted with the help of the previous functions in an appropriate way in order to build a speaker classification system.
We will no longer use in this part 5 random recordings but 15 for each speaker. We must take the 30 sentences in total and use the previous methods to compute and visualize the discriminating features. Under ideal circumstances we would have used the two different pitch estimations (autocorrelation and cepstrum), as well as the formants in order to pick the dividing thresholds.
Unfortunately, as our cepstrum function does not give us consistent and reliable results, we will only analyze the results of the autocorrelation.For the pitch we decided to average the pitch values in each frame and to take the average of that in order to get an estimate of an acceptable threshold, which for speakerA (SLT woman) is always appears to always be lower than 300. While for the speakerB (BDL man), it is greater than 300Hz. We can therefore conclude that if we have a pitch lower than 300 we have a sentence from the woman and an pitch greater than 300 comes from the man. We tried to average the formants in a similar fashion, however the values we get are very close values for speakerA and speakerB. In order to avoid false guesses we decided to mainly rely on the pitch in order to guess our speaker
We started by creating a first rule based function to test our reflections and observations, we did so by calculating the number of times the program guessed the wrong speaker. As you can see in the following figure(3.1), we get an identification accuracy of around 87%. Next we tried to create a few more rule based systems by modifying their parameters in order to get the most accurate system possible. As you can see on figures (3.1 and 3.2) we get better results in our first test. However, in our third test, we wanted to try to have the program find the threshold on it's own. As shown in figure(3.3) we get the same accuracy as in our first test.

```
AveragePitchA   284.94287696415796
AverageFormantsA 3541.1141152813493
AveragePitchB   333.74103746754014
AverageFormantsB 3623.603287331265
Number A guesses  11
Number B guesses  19
the rule based system has made  26  right guesses and  4
wrong ones.
Thus accuracy is  86.66666666666667 %
```

Figure 3.1: Results obtained in the *python console*

```
AveragePitchA   287.36086781095986
AverageFormantsA 3536.5204693594646
AveragePitchB   341.0167410834641
AverageFormantsB 3631.5272294187675
Number A guesses  12
Number B guesses  18
the rule based system has made  25  right guesses and  5
wrong ones.
Thus accuracy is  83.33333333333333 %
```

Figure 3.2: Results obtained in the *python console*

```
AveragePitchA   289.9488450698882
AverageFormantsA 3525.908904079055
AveragePitchB   345.2865727808881
AverageFormantsB 3609.269423032891
Number A guesses  11
Number B guesses  19
the rule based system has made  26  right guesses and  4
wrong ones.
Thus accuracy is  86.66666666666667 %
```

Figure 3.3: Results obtained in the *python console*

If we take a new data base collected from other people with possibly a different environment our "best" rule-based system will work but will not give the best results for this new data. Indeed, we can always use the functions of data extraction, calculation of the pitch, etc. However, we made a choice regarding the decisive values pitch as well as the formants allowing us to identify the speakers. This choice was made based on our observations of the results displayed. It would therefore be necessary to create a system allowing us to automatically calculate the new decision-making thresholds in the event of a change in the database used. Our curiosity lead us to "tease" the idea by computing a threshold using a first approximation done by our first rule based function, and then use it in a rule based 3 function ,which relied on that value, the results of this is shown in figure 3.3, and we can happily report that they seem promising.

Chapter 4

Building a machine learning-based system

Machine learning is an incredibly hip topic in the IT world nowadays. It is a much more robust, reliable and widely used algorithm than the rule-based one. In this approach we take a bunch of data, pick a decisive feature, in the example given by our lab supervisor, that feature was the color if we were to curate bananas and apples. Several samples of measurements would be necessary in order to "teach the system" how to decide, this goes to say, the system would have decision weights which would be corrected by each output. The value of the output is used to compute the error given that in the beginning stages the expected result is known before hands. The weights are corrected by the errors, so the accuracy of the algorithm improves the more guesses it has made. Unfortunately we were not able to implement a machine learning-based system due to time constraints.

Conclusion

This project allowed us to use concepts seen in the signal processing course and throughout the lab sessions and apply them in a more or less real life application. We also discovered many others concepts, ideas, and gained lots of knowledge in python, which is an incredibly popular and useful programming language nowadays, throughout the process. Indeed, over the past three weeks we had the opportunity to manipulate recordings, and extract features from these recordings. Unfortunately, some parts of the project had to be left out due to time constrains (machine learning, cepstrum) but we are overall happy to have been able to visualize and compute the majority of the requested features.

Appendix A

Bibliography

1.Formant, research by Priyanka A. Abhang, Suresh C. Mehrotra, in Introduction to EEG- and Speech-Based Emotion Recognition, 2016
https://www.sciencedirect.com/topics/medicine-and-dentistry/formant

2.What are formants?
https://person2.sol.lu.se/SidneyWood/praate/whatform.html:text=A

3.Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between
https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html