UMONS
Université de Mons

Faculté Polytechnique

Signal processing

Project of Signal processing

Tsolakis Georges, Vincent Stragier

POLYTECH MONS

December 2018

ACADÉMIE UNIVERSITAIRE WALLONIE-BRUXELLES

Contents

1 Introduction **1**

2 Functions **2**

2.0.1 Function wav import all 2
2.0.2 Normalize 2
2.0.3 Splicer 2
2.0.4 Signal energy 3
2.0.5 Pitch voiced autocorr 4

3 Rule based system **6**

4 Test bench **8**

5 Conclusion **10**

Chapter 1

Introduction

The goal of this given project is to preform speaker classification , recognise different speakers by using rule-based systems and machine learning-based techniques.First we are creating five general functions in order to use them in our test-bench for a specific data-set of different wav files , containing male and female voices.

Chapter 2

Functions

2.0.1 Function wav import all

With the following function wav import we are giving the name of the directory where the wav files are stored.The input of this function is the directory path and as output ,the "result" of the function" we have the samples (wav files) , the sampling frequencies and the file names

```
def wav_import_all(directory_path):
    samples_set = []
    sampling_frequencies = []
    file names = [f for f in listdir(directory_path)
    if isfile(join(directory_path, f))]

    for i in range(len(filenames)):
        samples = read((str(directory_path)+ "/" + filenames[i]))
        samples_set.append(np.array(samples[1],np.float))
        sampling_frequencies.append(np.array(samples[0],np.float))

    return samples_set, sampling_frequencies, filenames
```

2.0.2 Normalize

With the following function "normalize" we are taking the samples and we normalise them with the maximum value of the samples . The goal is to normalise the data in order to have to have signal values between -1 and 1 .As a result all the speech signals will have values in the same range , thus making easier the comparison.

```
def normalize(samples):
    return np.divide(samples, np.amax(np.absolute(samples)))
```

2.0.3 Splicer

Using this function we are separating a speech wav file into smaller sub parts (frames). In order to achieve the goal of this project we have to extract features that represent the signal as precisely as possible.Separating the signal into frames give us the chance to extract identifying features from a "smaller amount of data" ,making the output more precise. In this function we are using as input the data , the width of the window which represents the size of the data we are going to get from the original speech file (in ms) .With each step we are moving our window by a specific amount of ms inside the original speech file . For each step we obtain a different frame .

```
def splicer(samples, width, shifting_step, frequency):
    frames = []

    sampling_frequency = frequency/1000
    numeric_width = abs(int(width*sampling_frequency))
    numeric_shifting_step = abs(int(shifting_step*sampling_frequency))
    test_configuration = numeric_shifting_step-numeric_width

    # Determine the number of iteration to apply
    if test_configuration > 0:
        number_of_iterations =(len(samples)-numeric_width)/
        (test_configuration)
    elif test_configuration == 0:
        number_of_iterations = len(samples)/numeric_width
    else:
        number_of_iterations = -(len(samples)-numeric_width)/
        (test_configuration)

    # Split in frames
    for j in range(int(number_of_iterations)):
        frames.append(np.split(samples, [j*numeric_shifting_step,
        j*numeric_shifting_step + numeric_width])[1])

    return frames
```

2.0.4 Signal energy

In this function we are calculating the energy of a single frame by using the following expression

$E = \sum frame^2$

This function takes the frames calculated in the previous function as input and calculates their energy.A list called temp returns the values of the corresponding energies for each frame.

```
def signal_energy(frames):
    temp = []
    for i in range(len(frames)):
        temp.append(np.sum(np.square(frames[i])))
    return temp
```

2.0.5 Pitch voiced autocorr

Before using this function we have to determine a threshold.We are using a threshold in order to determine if we have a voiced or unvoiced sound.This determination is being made by looking at 3 random graphs for each gender (male , female). In order to determine its value we take a look at the peaks in the diagrams and we find that the lowest peak corresponds to a value of 0.3 .This function uses as input the frames calculated in the function splicer, the sampling frequency found in the wav import all function and the threshold which was determined as explained before. We are setting the limits (mix , max) for the lag parameter by dividing the sampling frequency by the minimum and maximum (60Hz , 500Hz).We are going through the list containing the energy for each frame and if the energy for a frame is bigger than the threshold calculated before , we are calling the xcorr function in order to find the auto-correlation for the specific energy of the frame. The auto-correlation is symmetrical so in order to speed up our code we are only getting the upper part. Then, we are assigning to a variable index to the first peak between 60Hz and 500 Hz.Taken under account that we only took the upper part of the auto-correlation we have to correct the index variable in order to get the right indicator.Finally we are using the sampling frequency and the index variable to calculate the pitch.

If the energy of each frame is smaller than the threshold we are placing a zero in the pitch list.

```
def pitch_voiced_autocorr(frames, sampling_frequency, threshold = 0.3):
    energy_of_each_frames = signal_energy(frames)

    maxlag = int(sampling_frequency/50.0)
    lag_min = int(sampling_frequency/500.0)
    lag_max = int(sampling_frequency/60.0)

    index = []
    pitch = []
    autocorr = 0

    for i in range(len(energy_of_each_frames)):
        if energy_of_each_frames[i] >= threshold:
            # Compute the autocorrelation of the
            voiced frame with maxlag = 50 Hz
            autocorr = plt.xcorr(frames[i], frames[i], maxlags=maxlag)[1]

            # Use only the upper side of the
            autocorrelation to compute de pitch
            upper_side = autocorr[int(len(autocorr)/2):]

            # Find the first peak between 60 Hz and 500 Hz
            index_ = np.argmax(upper_side[lag_min:lag_max])

            # Unbiase the index of the peak
            index_ += lag_min

            # Compute the picth frequency
            pitch.append(sampling_frequency/index_)
            #print(pitch[i])
            # Correct the index to display it on the full plot
            index.append(index_+int(len(autocorr)/2))

        else:
            # Correct the temporality of the vector
            pitch.append(0);
            index.append(0);
        #print(pitch)
    return np.array(pitch, np.float), index
```

Chapter 3

Rule based system

For this ruled-based system , we compute the mean value of the mean pitch value for both women and men for a known data-set. This mean value is then being used as a threshold, if the value is smaller than the threshold the speaker is a man , if not it is a woman.

```
# Compute the mean value of the pitch for each speaker
print("Compute␣the␣mean␣value␣of␣the␣pitch␣for␣each␣speaker")
start_time = time.time()
n_women = 0
n_man = 0
mean_women = 0
mean_man = 0

for i in range(number_of_samples):
    # The filename for women contain a "(2)" in their filename
    if(filenames[i].endswith("(2).wav")):
        n_women += 1
        mean_women += np.sum(pitch_of_voiced_frames_in_each_samples[i])
        /len(pitch_of_voiced_frames_in_each_samples[i])
    else:
        n_man += 1
        mean_man += np.sum(pitch_of_voiced_frames_in_each_samples[i])
        /len(pitch_of_voiced_frames_in_each_samples[i])
try:
    mean_women /= n_women
    mean_man /= n_man
except:
    print("Division␣by␣zero")

pitch_threshold = (mean_women + mean_man)/2
print("Done␣in␣(" + str(time.time() - start_time) + "␣seconds)")


# Test the proposed rule-based system
print("Test␣the␣proposed␣rule-based␣system␣on␣the␣database")
start_time = time.time()
speaker_kind = [] # 1 = Women, 0 = Man

for i in range(number_of_samples):
    if pitch_of_voiced_frames_in_each_samples[i])
    /len(pitch_of_voiced_frames_in_each_samples[i] < pitch_threshold
print("Done in (" + str(time.time() - start_time) + " seconds)")
print("Finished in " + str(time.time() - start_time_) + " seconds")
```

Chapter 4

Test bench

Until here we created the necessary functions for the project.This function are generic , we can use them in this project or in some other project that requires the same processing.Here, in the test bench section we are applying our functions to the specific data in which we are interested in . In order to do this we are using a loop statement for the whole data-set , for every sample .Here in this section we are also plotting the graphs .

```
# import matplotlib.pyplot as plt

# Import a set of samples
samples_set, sampling_frequencies, filenames = wav_import_all ("./samples")
number_of_samples = len(samples_set)

# 1. Auto-correlation-Based Pitch Estimation System:
# Normalize the set of samples
print("Normalize␣samples␣set")
start_time = time.time()
normalized_set = []
for i in range(number_of_samples):
    normalized_set.append(normalize(samples_set[i]))
print("Done␣in␣(" + str(time.time() - start_time) + "␣seconds)")

# Slice the normalized set of samples in frames
# Slicing parameters
print("Slices␣the␣normalized␣samples␣set")
start_time = time.time()
widths_arrays = np.ones(len(normalized_set))*30 # ms
shifting_steps_arrays = np.ones(len(normalized_set))*30 # ms

set_of_samples_frames  = []
for i in range(number_of_samples):
    set_of_samples_frames.append(splicer(normalized_set[i], widths_arrays[i],
print("Done␣in␣(" + str(time.time() - start_time) + "␣seconds)")

"""
```

```
print("Compute the pitch")
start_time = time.time()
# Compute the pitch of voiced frames of each sample
pitch_of_voiced_frames_in_each_samples = []
for i in range(number_of_samples):
    plt.figure(i+1)
    pitch_of_voiced_frames_in_each_samples.append
    (pitch_voiced_autocorr(set_of_samples_frames[i], sampling_frequencies[i],
    plt.close('all')
print("Done in (" + str(time.time() - start_time) + " seconds)")

#plt.close('all')
```

Chapter 5

Conclusion

On this given project we had the opportunity to practise the notions given on the theoretical course and also take a deeper look at machine learning-based techniques and rule-based systems .