UMONS
Université de Mons

Faculté Polytechnique

SIGNAL PROCESSING'S PROJECT

How Useful Is Machine Learning ?
A Speaker Classification Project

LEKEMO ATONFACK Rosabelle
JIMENEZ MILLAN Sebastian

POLYTECH MONS

Under the direction of Mr DUTOIT and of Mr EL HADDAD

Academic Year: 2018-2019

ACADÉMIE UNIVERSITAIRE WALLONIE-BRUXELLES

0.1 Introduction

Machine learning is a kind of artificial intelligence based on the construction of computer programs that can understand, deduce and improve themselves through the experience of the data provided to it.

This project thus consist of taking over the concept of machine learning which is building systems based on data. In order to best acquire this concept, we will focus on the implementation rule-based systems using signal processing techniques and then compare them to machine learning-based techniques.

Within the limit of our study, we will therefore be focused on speaker classification, which means trying to recognize some speaker's voices.

0.2 SIGNAL PRE-PROCESSING

The libraries used for the functions :

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as sig
from scipy.io.wavfile import read
from numpy import random as rnd
from scikit_talkbox_lpc import lpc_ref
```

Function that returns normalized data input

```
def normalize(data):
    x=np.max(np.abs(data))
    amp1=data/x
    return amp1
```

Function that splits given that into frames

- signal = data to split
- ms = size of each frame in ms
- step = shifting step in ms

```
def frames(signal,ms,step):
    wsize=ms*16
    split=[]
    etape=step*16
    for i in range(0,len(signal),etape):
        window=[]
        for j in range(i,wsize+i,1):
            if j<len(signal):
                window.append(signal[j])
        if len(window)==wsize:
            split.append(window)
        else:
            for k in range(wsize-len(window)):
                window.append(0)
            split.append(window)
    return split
```

FEATURES EXTRACTION ALGORITHMS

Signal Energy

Function which outputs the energy of a given signal, using the formula $E = \Sigma|x(i)|^2$:

```
def energy(signal):
    Energy=sum(np.abs(np.array(signal))**2)
    return Energy
```

Pitch

Autocorrelation-Based Pitch Estimation System

Function that estimates pitch for frames of a signal with autocorrelation, calculating the energy of each frame and classifying it as voiced or invoiced depending on the threshold given. Returns energy, voiced as 1 unvoiced as 0, and frequency of each frame.

```
def acorrpitch(sound,threshold,wsize,step):
    ns=normalize(sound)          #start normalizing signal
    sp=frames(ns,wsize,step)     #divide it in frames
    Energy=[]; voice=[]; f0=[]   #energy, voiced/unvoiced and frequency
    for e in range(len(sp)):
        en=energy(sp[e])         #calculate energy of the frame
        Energy.append(en)        #append energy to energy vector
    Energy=normalize(Energy)     #normalize energy
    for i in range(len(sp)):
        if Energy[i]>threshold:
            voice.append(1)                          #voiced frame==1
            c=plt.xcorr(sp[i],sp[i],maxlags=320)     #autocorrelation
            pks=sig.find_peaks(c[1])     #find peaks on autocorrelation vector
            p=[]     #vector p which will have the real values, not the index
            for j in range(len(pks[0])):
                if pks[0][j]<287 or pks[0][j]>351:   #frequency range
                    p.append(c[1][pks[0][j]])        #append to vector p
            s=np.sort(p)                 #sort the peaks from lowest to highest
            p2=s[-1]                             #second highest peak
            for k in range(320):
                if p2==c[1][k]:          #find position of second highest peak
                    d=320-k              #highest peak is always in position 320
            f=(16000)/d          #frequency=fe/N, N is distance between peaks
            f0.append(f)                 #append frequency to frequency vector
        else:
            voice.append(0)              #unvoiced frame==0
            f0.append(0)                 #frequency is 0 for unvoiced frames
    return np.array(voice),np.array(f0)
```

Cepstrum-Based Pitch Estimation System

Function that estimates pitch for frames of a window with cepstrum, calculating the energy of each frame and classifying it as voiced or invoiced depending on the threshold given. Calculates frames on a rectangular window and on a hamming window
Returns energy, voiced as 1/unvoiced as 0, and frequency of each frame on a rectangular window and a hamming window

```
def cepstrumpitch(sound,threshold,wsize,step):
    ns=normalize(sound)                         #start normalizing signal
    sp=frames(ns,wsize,step)                    #divide it in frames
    Energy=[]; voice=[]; f0=[]; hf0=[]          #energy, voiced/unvoiced and frequency
    hamm=np.hamming(wsize*16)                   #hamming window with size of a frame
    for e in range(len(sp)):
        en=energy(sp[e])                        #calculate energy of the frame
        Energy.append(en)                       #append energy to energy vector
    Energy=normalize(Energy)                    #normalize energy
    for i in range(len(sp)):
        hf=hamm*sp[i]                           #hamming window on frame
        if Energy[i]>threshold:
            voice.append(1)                     #voiced frame==1
            w=np.fft.fft(sp[i])                 #FFT on rect window frame
            wlog=np.log10(np.abs(w))            #log spectrum on rect window frame
            hw=np.fft.fft(hf)                   #FFT on hamming window frame
            hwlog=np.log10(np.abs(hw))          #log spectrum on hamming window frame
            cepstrum=np.fft.ifft(wlog)          #cepstrum of rect window frame
            hcep=np.fft.ifft(hwlog)             #cepstrum of hamming window frame
            cut=cepstrum[50:500]                #to cut from 50 to 500Hz
            hcut=hcep[50:500]                   #cut in hamming window frame
            for j in range(len(cut)):
                if max(cut)==cut[j]:            #find peak in cut vector
                    f=j+50                      #frequency is position of the max value
            f0.append(f)                        #append frequency to frequency vector
            for k in range(len(hcut)):
                if max(hcut)==hcut[k]:           #find peak in cut vector
                    hf=k+50                      #position of the max value
            hf0.append(hf)                        #append frequency to frequency vector
        else:
            voice.append(0)                      #unvoiced frame==0
            f0.append(0)                         #frequency is 0 for unvoiced frames
            hf0.append(0)                        #hamming frequency vector
    return Energy,np.array(voice),np.array(f0),np.array(hf0)
```

Other functions

Function that selects a random sentence from a given gender either 'male' or 'female'

```
def selectsound(gender):
    ab=rnd.choice(['a','b'])
    if ab=='a':
        rm0=rnd.randint(1,594)
        selection=ab,rm0
        if rm0<10:
            fs,Mono=read('/Users/Sebas/Documents/UMONS/'+gender+'/wav/
arctic_a000'+str(rm0)+'.wav')
```

```
        if rm0>=10 and rm0<100:
            fs,Mono=read('/Users/Sebas/Documents/UMONS/'+gender+'/wav/arctic_a00'+str(rm0)+'.wav')
        if rm0>=100:
            fs,Mono=read('/Users/Sebas/Documents/UMONS/'+gender+'/wav/arctic_a0'+str(rm0)+'.wav')
    else:
        rm1=rnd.randint(1,540)
        selection=ab,rm1
        if rm1<10:
            fs,Mono=read('/Users/Sebas/Documents/UMONS/'+gender+'/wav/arctic_b000'+str(rm1)+'.wav')
        if rm1>=10 and rm1<100:
            fs,Mono=read('/Users/Sebas/Documents/UMONS/'+gender+'/wav/arctic_b00'+str(rm1)+'.wav')
        if rm1>=100:
            fs,Mono=read('/Users/Sebas/Documents/UMONS/'+gender+'/wav/arctic_b0'+str(rm1)+'.wav')
    return fs,Mono,selection
```

Function that select a random sound from a given gender 'male' or 'female', normalizes signal, splits it in frames and calculates energy.
Plots the signal in temporal domain and normalized energy by frames.
Returns normalized signal, energy and the number of the wav file

```
def randsound(gender,wsize,step):
    fs,Mono,sound=selectsound(gender)
    t=np.linspace(0,(len(Mono)-1)/fs,len(Mono))
    Mono=normalize(Mono)
    sp=frames(Mono,wsize,step)
    E=[]
    print('Number of frames: '+str(len(sp)))
    for i in range(len(sp)):
        en=energy(sp[i])
        E.append(en)
    Energy=normalize(E)
    plt.figure()
    plt.suptitle(gender+' '+sound[0]+str(sound[1]),fontsize=16)
    plt.subplot(211)
    plt.plot(t,Mono)
    plt.xlabel('time')
    plt.ylabel('pressure')
    plt.title('Signal')
    plt.subplot(313)
    plt.plot(Energy)
    plt.xlabel('frame')
    plt.ylabel('energy')
    plt.title('Energy')
    return Mono,Energy,sound
```

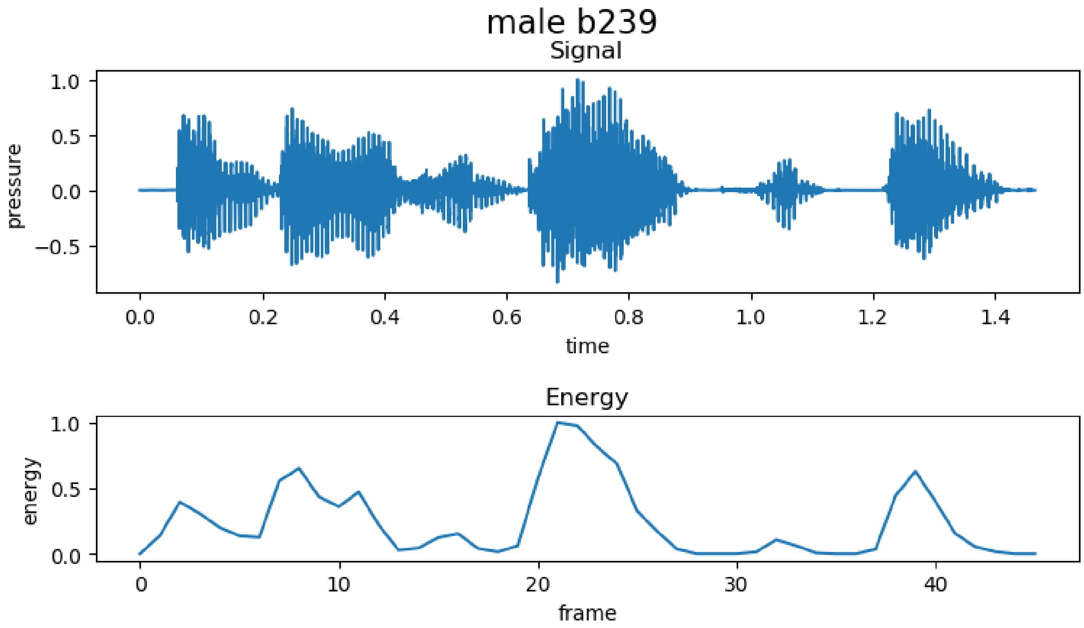

Figure 1 – Example of a random sound chosen from BLT with plots of temporal domain and energy, using the function randsound

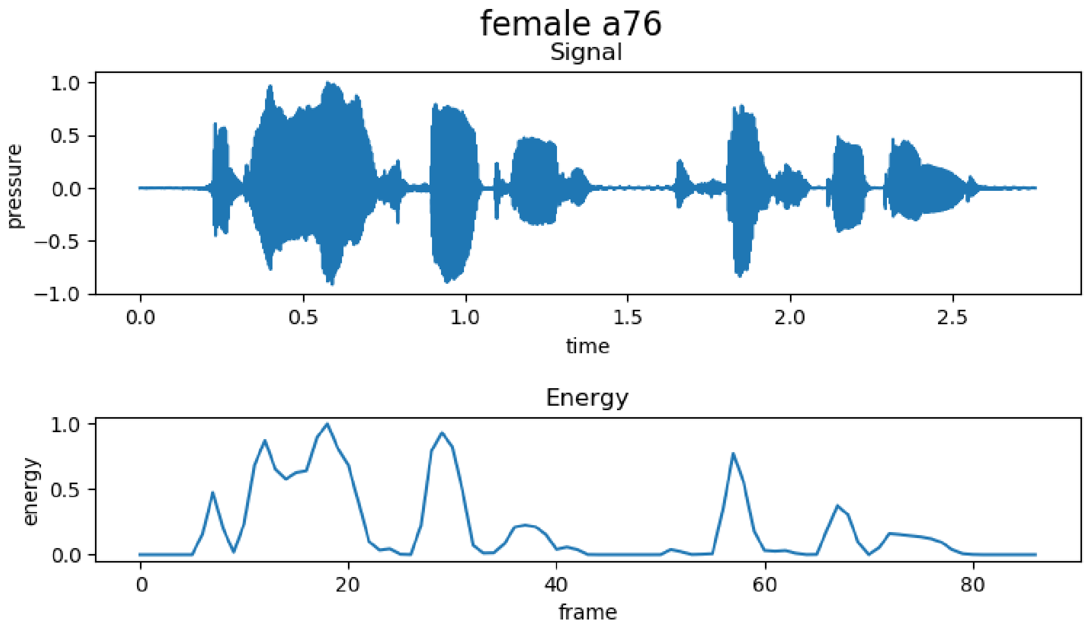

Figure 2 – Example of random sound chosen from SLT with plots of temporal domain and energy, using the function randsound

Formants

Function that calculates the formants of the frames of a given signal, with wsize beign the size of each frame in *ms* and the step the shifting step.
Returns a list with the formants on each frame.

```
def formants(signal,wsize,step):
    sp=frames(signal,wsize,step)          #divide signal in frames
    hamm=np.hamming(wsize*16)             #hamming window with size of a frame
    coeff=[]; roots=[]; raiz=[]; angle=[]; freq=[];
    for i in range(len(sp)):
        filtered=sig.lfilter([1,-0.67],[1],sp[i])   #high pass filter on frame
        hf=hamm*filtered                            #hamming window on frame
        coeff.append(lpc_ref(hf,10))                #LPC coefficients of each frame
        roots.append(np.roots(coeff[i]))            #roots of the LPC
    for j in range(len(roots)):
        l=len(roots[j])                             #number of roots
        erase=[k for k in range(1,l,2)]             #delete half of roots
        r=np.delete(roots[j],erase)
        raiz.append(r)       #save roots
        angle.append([])     #this empty lists are to append here the formants
        freq.append([])      #and then append the list of formants of each frame
        for z in range(int(l/2)):                   #angle for each root
            angle[j].append(np.arctan(np.imag(raiz[j][z])/np.real(raiz[j][z])))
            fn=abs((angle[j][z]*16000)/(2*np.pi))   #calculate frequency
            freq[j].append(fn)                      #save frequency on formant list
        freq[j].sort()           #sort frequencies from lowest ot highest
    return np.array(freq)
```

0.2.1 MFCC

Function that outputs the power spectrum of the signal, the energy and finally the MFCCs which are the coefficients representing the vocal tract activities.

```
import scipy as sp
from scipy.fftpack import dct

def MFCC(signal,ms,step,f0):

    mfccs=[]
    P=[]
    y=[]
    const_a=0.97
    a=[]
    a[0]=1
    a[1]=0
    b=[]
    b[0]=1
    b[1]=-const_a
    NTFD=512
    fbanks=[]

    sigFiltered=scipy.signal.lfilter(a,b,signal)
    split=frames(sigFiltered,ms,step)

```

```
    for i in range(len(split)):
        fenetre=scipy.signal.hamming(len(split[i]))    #each window taken has the same lenght as the frame
        yframe=split[i]*fenetre                        #apply each window to each frame
        #y.append(yframe)
        P.append((numpy.linalg.norm(numpy.fft(yframe))^2)/NTFD)


    pow_frames=frames(P,ms,step)

    fbanks=filter_banks(pow_frames, f0, nfilt = 40, NFFT=512)
    y=scipy.fftpack.dct(fbanks, type=2, axis=1, norm='ortho')


    en=y[0]         #the first output is the energy
    for i in range(1,12,1):
        mfccs[i-1]=y[i]       #the 12 oders are the mfccs


    return P,en,mfccs
```

Rule Based System

To implement the classification system, we create a set of rules for classifying the sentences of the speakers. For the parameters, we use size of a frame = $50ms$, shifting step = $16ms$ and threshold = 0.10. The threshold chosen at 0.10, is after observing the plotted energy along with the temporal plot of the sound, finding that the sounds we want to find frequencies from are mostly above this level.

We select randomly 15 sentences from each speaker, we calculate the average frequency on each sentence selecting only the voiced frames. We then calculate the average frequency for each speaker. Finally the average frequency of both speakers is calculated.

```
'Rule Based System'


import speaker as spk
import numpy as np


#Parameters
wsize=50            #size of each frame in ms, real size is wsize*16
step=16             #shifting step in ms, real step is step*16
threshold=0.10      #threshold for clasifying frame as voiced or unvoiced

#Lists for the selected sounds and extracted features
speaker_male=[]
speaker_female=[]
num_male=[]
num_female=[]
voice_male=[]
voice_female=[]
freq_male=[]
```

```
freq_female=[]

#Select 15 random sentences from each speaker
for i in range(15):                                                 #Sentences
    MonoM,energiaF,numeroM=spk.randsound('Male',wsize,step)         #Male
    MonoF,energiaF,numeroF=spk.randsound('Female',wsize,step)       #Female
    speaker_male.append(MonoM)       #Signal of selected male sentence
    speaker_female.append(MonoF)     #Signal of selected female sentence
    num_male.append(numeroM)         #File number for male sentences
    num_female.append(numeroF)       #File number for female sentences
    voiceM,frequencyM=spk.acorrpitch(MonoM,0.10,wsize,step) #autocorrelation
    voiceF,frequencyF=spk.acorrpitch(MonoF,0.10,wsize,step) #autocorrelation
    voice_male.append(voiceM)        #voiced or unvoiced classification
    voice_female.append(voiceF)      #for each frame
    freq_male.append(frequencyM)     #List of frequencies for male sentences
    freq_female.append(frequencyF)   #list of frequencies for male sentences

#Will calculate average frequency for each sentence
average_freq_male=[]
average_freq_female=[]

for i in range(15):
    f=0; v=0                 #frequency, number of voiced frames
    l=len(voice_male[i])     #number of frames of the signal
    for j in range(l):
        if voice_male[i][j]==1:      #only select the voiced frames
            f=f+freq_male[i][j]      #sum of all frequencies
            v=v+1                    #number of frequencies summed
    average_freq_male.append(f/v)   #average frequency

#same for female sentences
for i in range(15):
    f=0; v=0
    l=len(voice_female[i])
    for j in range(l):
        if voice_female[i][j]==1:
            f=f+freq_female[i][j]
            v=v+1
    average_freq_female.append(f/v)

#Calculate total average frequency for male and female sentences
tot_average_male=np.mean(average_freq_male)
tot_average_female=np.mean(average_freq_female)

print('Average frequency of male sentences: '+str(tot_average_male))
print('Average frequency of female sentences: '+str(tot_average_female))

#Calculate average frequency between both speakers
average_ensemble=(tot_average_male+tot_average_female)/2

print('Average frequency of all sentences: '+str(average_ensemble))
```

Classification System 1

For the first rule based system, we work on the average frequency obtained and we use this as the threshold to classify new randomly chosen sentences. If the average frequency of a sentence is lower than this threshold, we classify it as male, and if it is higher we classify it as female. We then calculate the accuracy of the system.

```
'System 1'

#System 1's threshold will rely on the average frequency of all the sentences
#After calculating the average frequency of all sentences in both speakers,
#it will classify a sentence as BLT(male) speaker if the average frequency
#is below the total average frequency, and as SLT(female) if it is higher

#New selection of sentences
sentence_male_1=[]
sentence_female_1=[]
voice_male_1=[]
voice_female_1=[]
freq_male_1=[]
freq_female_1=[]

for i in range(15):                     #Select new random sentences
    MonoM,energiaF,numeroM=spk.randsound('Male',wsize,step)      #male
    MonoF,energiaF,numeroF=spk.randsound('Female',wsize,step)    #female
    sentence_male_1.append(MonoM)                                #Signal
    sentence_female_1.append(MonoF)                              #Lists
    voiceM,frequencyM=spk.acorrpitch(MonoM,0.10,wsize,step) #Pitch estimation
    voiceF,frequencyF=spk.acorrpitch(MonoF,0.10,wsize,step) #by autocorrelation
    voice_male_1.append(voiceM)       #voiced
    voice_female_1.append(voiceF)     #or unvoiced
    freq_male_1.append(frequencyM)    #Frequencies
    freq_female_1.append(frequencyF)  #lists

#Will calculate average frequency for each sentence
average_freq_male_1=[]
average_freq_female_1=[]

for i in range(15):
    f=0; v=0                     #frequency, number of voiced frames
    l=len(voice_male_1[i])       #number of frames of the signal
    for j in range(l):
        if voice_male_1[i][j]==1:     #only select the voiced frames
            f=f+freq_male_1[i][j]     #sum of all frequencies
            v=v+1                     #number of frequencies summed
    average_freq_male_1.append(f/v)   #average frequency

#same for female sentences
for i in range(15):
    f=0; v=0
    l=len(voice_female_1[i])
    for j in range(l):
        if voice_female_1[i][j]==1:
            f=f+freq_female_1[i][j]
            v=v+1
    average_freq_female_1.append(f/v)
```

```python

#Classification lists
classification_male_1=[]
classification_female_1=[]


#go through all sentences
for i in range(15):
    if average_freq_male_1[i]<=average_ensemble:    #if freq < average,
        classification_male_1.append(1)             #1==true
    else:                                           #otherwise 0==false
        classification_male_1.append(0)
    if average_freq_female_1[i]>average_ensemble:   #for female
        classification_female_1.append(1)           #the condition
    else:                                           #freq > average
        classification_female_1.append(0)

#accuracy of male and female classification, and system total
accuracy_male_1=np.mean(classification_male_1)
accuracy_female_1=np.mean(classification_female_1)
accuracy1=np.mean(classification_male_1+classification_female_1)

print('System 1')
print('Accuracy for male speaker: '+str(accuracy_male_1))
print('Accuracy for female speaker: '+str(accuracy_female_1))
print('System Accuracy: '+str(accuracy1))
```

These are some results from the classification of the first system, testing on 15 sentences of each speaker :

```
Accuracy for male speaker: 0.8666666666666667
Accuracy for female speaker: 1.0
System Accuracy: 0.9333333333333333
```

FIGURE 3 – Results 1

```
Accuracy for male speaker: 0.9333333333333333
Accuracy for female speaker: 1.0
System Accuracy: 0.9666666666666667
```

FIGURE 4 – Results 2

0.3 Conclusion

In this project, our study was about the implementation of rule-based systems in order to figure out the machine learning concept. We therefore focused on speaker classification which meant recognizing speaker's voices.

All a long this project, we first of all had to pre-process the signals by normalizing them and splitting them into frames. Then, we had to extract some of their features which were their energy and an estimation of the pitch. After that, we had to characterize the shape of the signal at a certain instant by outputting the resonances frequencies of the vocal tract. And finally, we define rules to build a speaker recognition system which are, in this case, rules to classify males and females.

Arrived to the end of this project, we were certainly able to classify speakers. An important point to remind is that, our rules classification had been based on our data and will then obviously showed a good accuracy. This might not be the case on others data that have never been seen by our algorithm.