**Voiceprint Recognition**

Yongcheng Huang

Delft University of Technology



# Introduction

This project is implemented through python and mainly working as a usable tool to implement voiceprint recognition. Reference the implementation of SHAZAM, which is a widely used platform in reality.

# Music compression

A plot in pseudocolor of the magnitude spectrum of 100 audio frames (spectrogram). Horizontally the 100 frames, vertically the frequency (DFT index).

Background pattern

Description automatically generated

Since the sampling rate is 44100HZ, thus the length of frame in milliseconds is:

**512/44100\*1000=11.6ms**

The total number of frames of the file using is 857.

Since there is lot of noise in the whole audio signal, only the coefficients that have the largest magnitude are those which has the highest effect in the sound recognition for human beings. Thus, we have to keep the important audio signal and shrink the effect of the noise.

Chart

Description automatically generated

A plot in pseudocolor of the magnitude spectrum of 100 audio frames when keeping only the 3 largest DFT coefficients in the range [0,N/2]. Horizontally the frames, vertically the frequency (DFT index).

Chart

Description automatically generated

Show a plot in pseudocolor of the magnitude spectrum of 100 audio frames when keeping the 8 largest DFT coefficients in the range [0,N/2]. Horizontally the frames, vertically the frequency (DFT index).

From the plots about and by listening to the compressed signal, I choose 10 as the value of DFT coefficients to keep in the compressed signal. Since L = 10, 19 samples are kept in each frame, then the compression rate is 19/512 = 0.037.

For the whole file, it will be 857\*19/438784\*100%=3.7%

# Voiceprint Recognition

Chart

Description automatically generated

A plot in pseudocolor of the magnitude spectrum of 200 audio frames (spectrogram) of music file

‘query\_blurred\_lines\_1.wav’. Horizontally the 200 frames, vertically the frequency (DFT index).

For the length of each frame in milliseconds, like what we have done before, since the sample rate is 22050HZ, the length is

**512/22050\*1000=23.2ms**

The total number of frames in the file ‘query\_blurred\_lines\_1.wav’ is 257.

A screenshot of a computer

Description automatically generated with low confidence

A plot in pseudocolor of the bark scale energies of the music file ‘query\_blurred\_lines\_1.wav’. Horizontally the 200 frames, vertically the bark scale frequency bands (0-16).

A picture containing chart

Description automatically generated

A plot in pseudocolor of the fingerprint (0-1 pattern) of the music file ‘query\_blurred\_lines\_1.wav’. Horizontally the 200 frames, vertically the bits (0-15)

Finally, I choose to use bit error rate to compare the fingerprints of the files. Since the BER can show the difference of the provided files and the given fingerprint. The smaller the difference, the more similar the two fingerprints are. Thus, we can use the BER as a good method to compare the similarity of two fingerprints.

For a completely unrelated song, the BER should be around 0.5. And for a different version of the same song, it should be about 0.25.

# CODE-Compression

fs, mmdata = wv.read("data/musicclip.wav")

display(Audio(mmdata, rate=fs))

nicesignalplot(np.arange(0,len(mmdata)/fs,1/fs),mmdata,'audio file musicclip.wav')

N = 512

M = int(len(mmdata)/N)

mmdata.resize(N \* M, refcheck = False)

MMFRAMES = mmdata.reshape([M,N])

MMFRAMES = np.fft.fft(MMFRAMES)

print(MMFRAMES.shape)

plt.figure(figsize=[12,8])

plt.pcolor(np.log(np.abs(MMFRAMES[0:100,:int(N/2)].T + 0.0001)))

plt.title('Spectrogram')

plt.show()

#2\*L-1 = 15

L = 10

CMFRAMES = np.zeros(MMFRAMES.shape ,dtype = 'complex\_')

for i in range(MMFRAMES.shape[0]):

    indices = np.argsort(MMFRAMES[i,:])

    largest = indices[indices.shape[0] - 2\*L+1: ]

    for index in largest:

        CMFRAMES[i, index] = MMFRAMES[i, index]

plt.figure(figsize=[12,8])

plt.pcolor(np.log(np.abs(CMFRAMES[0:100,:int(N/2)].T + 0.0001)))

plt.title('Spectrogram')

plt.show()

plt.pcolor(np.log(np.abs(CMFRAMES[0:100,:int(N/8)].T)+0.0001))

CMFRAMES = np.fft.ifft(CMFRAMES)

signal = CMFRAMES.reshape(857\*512)

display(Audio(signal, rate=fs))

# CODE-Voiceprint Recognition

musicfiles = ['data/querysongs/query\_call\_me\_1.wav','data/querysongs/query\_call\_me\_2.wav',\

              'data/querysongs/query\_call\_me\_3.wav','data/querysongs/query\_call\_me\_4.wav',\

              'data/querysongs/query\_call\_me\_5.wav',\

              'data/querysongs/query\_get\_lucky\_1.wav','data/querysongs/query\_get\_lucky\_2.wav',\

              'data/querysongs/query\_get\_lucky\_3.wav','data/querysongs/query\_get\_lucky\_4.wav',\

              'data/querysongs/query\_get\_lucky\_5.wav',\

              'data/querysongs/query\_scream\_shout\_1.wav','data/querysongs/query\_scream\_shout\_2.wav',\

              'data/querysongs/query\_scream\_shout\_3.wav','data/querysongs/query\_scream\_shout\_4.wav',\

              'data/querysongs/query\_scream\_shout\_5.wav',\

              'data/querysongs/query\_locked\_out\_1.wav','data/querysongs/query\_locked\_out\_2.wav',\

              'data/querysongs/query\_locked\_out\_3.wav','data/querysongs/query\_locked\_out\_4.wav',\

              'data/querysongs/query\_locked\_out\_5.wav',\

              'data/querysongs/query\_blurred\_lines\_1.wav','data/querysongs/query\_blurred\_lines\_2.wav',\

              'data/querysongs/query\_blurred\_lines\_3.wav','data/querysongs/query\_blurred\_lines\_4.wav',\

              'data/querysongs/query\_blurred\_lines\_5.wav']

fs, xx = wv.read('data/querysongs/query\_blurred\_lines\_1.wav')

display(Audio(xx, rate=fs))

nicesignalplot(np.arange(0,len(xx)/fs,1/fs),xx,'audio file query\_blurred\_lines\_1.wav')

Nframelen = 512

Mframes   = int(len(xx)/Nframelen)

print('Input speech is divided into',Mframes,'frames of length',Nframelen)

xx.resize(Mframes\*Nframelen)

frames = xx.reshape([Mframes,Nframelen])

XX = np.fft.fft(frames)

print(XX.shape)

plt.pcolor(np.log(np.abs(XX[0:200,:int(Nframelen/8)].T)+0.001))

plt.show()

BarkScaleBandID = np.load("data/bark\_scale\_band\_id.pkl", allow\_pickle = True)

print(BarkScaleBandID)

BB   = np.zeros([Mframes,17])

for n in range(0,int(Nframelen/2)+1):

    BB[:,BarkScaleBandID[n]] += (np.abs(XX[:,n]))

plt.figure(figsize=[12,8])

plt.pcolor(np.log(BB[:200,:].T))

plt.title('Bark frequency-scale energies')

plt.show()

def e(BB, m, n):

    return (BB[m+1, n+1]-BB[m+1, n]) - (BB[m, n+1]-BB[m, n])

def b(m, n, BB):

    if (e(BB, m, n) >= 0): return 1

    else: return 0

bb = np.zeros((257,17))

for m in range(0, Mframes-1):

    for n in range(0, 15):

        bb[m, n] = b(m, n, BB)

plt.figure(figsize=[12,8])

plt.pcolor(bb[:200,:].T)

plt.title('Bark frequency-scale energies')

plt.show()

data = np.load('data/fingerprint.pkl',allow\_pickle=True)

BER = []

for name in musicfiles:

    fs, xx = wv.read(name)

    Nframelen = 512

    Mframes   = int(len(xx)/Nframelen)

    xx.resize(Mframes\*Nframelen)

    frames = xx.reshape([Mframes,Nframelen])

    XX = np.fft.fft(frames)

    BB   = np.zeros([Mframes,17])

    for n in range(0,int(Nframelen/2)+1):

        BB[:,BarkScaleBandID[n]] += (np.abs(XX[:,n]))

    bb = np.zeros((257,17))

    for m in range(0, Mframes-1):

        for n in range(0, 15):

            bb[m, n] = b(m, n, BB)

    D=0

    for m in np.arange(0, Mframes-1):

        for n in np.arange(0, 15):

            if(bb[m,n] != data[m,n]):

                D = D + 1

    D= D / (256 \* 16)

    BER.append(D)

print(np.argmin(np.array(BER)))

print(BER)