University of Colorado at Boulder **ECEN 4532**

Lab 1
Lab report due on January 28, 2019

This is a Python only lab. Each student needs to turn in her/his own lab report and own programs.

1 Introduction

The goal of this lab is to explore some of the recent techniques developed in the audio industry to organize, and search large music collections by content. The explosion of digital music has created a need to invent new tools to search and organize music. Several websites provide large database of musical tracks:

- http://magnatune.com/
- http://www.allmusic.com/
- http://www.last.fm/

and also allow users to find musical tracks and artists that are similar.

Companies such as gracenote and shazam have application to recognize a song based solely on the actual music. Other examples of automated music analysis include

1. Score following: Rock Prodigy, SmartMusic, Rockband
2. Automatic music transcription: Zenph
3. Music recommendation, playlisting: Google Music, Last.fm, Pandora, Spotify
4. Machine listening: Echonest

At the moment these tools are still improving. Fast computational methods continue to be needed to expand these tools beyond their current scope and integrate them on portable music players.

The goal of this lab is to develop digital signal processing tools to automatically extract features that characterize music.

2 Audio signal: from the sound to the wav file

Sounds are produced by the vibration of air; sound waves produce variations of air pressure. The sound waves can be measured using a microphone that converts the mechanical energy into electrical energy. Precisely, the microphone converts air pressure into voltage levels. The electrical signal is then sampled in time at a sampling frequency f_s, and quantized. The current standard for high quality audio and DVD is a sampling frequency of $f_s = 96$kHz, and a 24-bit depth for the quantization. The

CD quality is 16 bit sampled at $f_s = 44.1$ kHz.

Assignment

1. Dolphins can only hear sounds over the frequency range $[7 - 120]$ kHz. At what sampling frequency f_s should we sample digital audio signals for dolphins?

2.1 Segments, Frames, and Samples

In order to automatically analyze the music, the audio files are segmented into non overlapping segments of a few seconds. This provides the coarsest resolution for the analysis. The audio features are computed at a much finer resolution. The audio file is divided into frames of a few milliseconds over which the analysis is conducted. In some cases the frames may be chosen to overlap, whereas in other cases they will not.

In this lab the audio files are sampled at $f_s = 11,025$ Hz, and we consider intervals of size $N = 512$ samples, or 46 ms. This interval of 46 millisecond is called a **frame**. We will implement several algorithms that will return a set of features for each frame. While there are 512 samples per frame, we will usually return a feature vector of much smaller size.

3 The music

In the file `http://ecee.colorado.edu/~fmeyer/.private/` you will find 12 tracks of various length (username: 'music'; password: 'eCmsHPaJeGT8GBue5WmAiUld'). There are two examples of six different musical genres:

1. Classical
2. Electronic
3. Jazz
4. Metal and punk
5. Rock and pop
6. World

The name of the file indicates its genre.

The musical tracks are chosen because they are diverse but also have interesting characteristics, which should be revealed by your analysis.

- track201-classical is a part of a violin concerto by J.S. Bach (13-BWV 1001 : IV. Presto).
- track204-classical is a classical piano piece composed by Robert Schumann (Davidsbundlertanze, Op.6. XVI).
- track370-electronic is an example of synthetic music generated by software that makes it possible to create notes that do not have the regular structure of Western music. The pitches may be unfamiliar but the rhythm is quite predictable.
- track396-electronic is an example of electronic dance floor piece. The monotony and the simplicity of the synthesizer and the bass drum are broken by vocals.
- track437-jazz is another track by the same trio as track439-jazz
- track439-jazz is an example of (funk) jazz with a Hammond B3 organ, a bass, and a percussion. The song is characterized by a well defined rhythm and a simple melody.
- track463-metal is an example of rock and metal with vocals, bass, electric guitars and percussion.
- track492-metal is an example of heavy metal with heavily distorted guitars, drums with double bass drum.
- track547-rock is an example of new wave rock of the 80's. The music varies from edgy to hard. It has guitars, keyboards, and percussion.
- track550-rock is an example of pop with keyboard, guitar, bass, and vocals. There is a rich melody and the sound of steel-string guitar.
- track707-world: this is a Japanese flute, monophonic, with background sounds between the notes. The notes are held for a long time.
- track729-world: this is a piece with a mix of Eastern and Western influence using electric and acoustic sarod and on classical steel-string guitars.

Assignment

2. Write a Python function that extracts T seconds of music from a numpy array holding wav file data. The sample at which you start extracting should be an argument which you input as a variable. You will use the python function `scipy.io.wavfile.read` to read a track from a file. You will need to install the `scipy` package in order to have access to it

   In the lab you will use T =**24 seconds** and start your extraction at the sample halfway through each track to compare the different algorithms. Download the 12 audio files, and test your function.

4 Low level features: time domain analysis

The most interesting features will involve some sophisticated spectral analysis. There are, however, a few simple features that can be directly computed in the time domain, We describe some of the most popular features in the following.

4.1 Loudness

The standard deviation of the original audio signal $x[n]$ computed over a frame of size N provides a sense of the loudness,

$$\sigma(n) = \sqrt{\frac{1}{N}\sum_{m=0}^{N-1}\left[x(nN+m) - E[x_n]\right]^2} \quad \text{with} \quad E[x_n] = \frac{1}{N}\sum_{m=0}^{N-1} x(nN+m) \qquad (1)$$

4.2 Zero-crossing

The Zero Crossing Rate is the average number of times the audio signal crosses the zero amplitude line per time unit. The ZCR is related to the pitch height, and is also correlated to the noisiness of the signal. We use the following definition of

ZCR,

$$\mathrm{ZCR}(n) = \frac{1}{N-1}\sum_{m=1}^{N-1}\frac{1}{2}|\mathrm{sgn}(x(nN+m)) - \mathrm{sgn}(x(nN+m-1))|. \qquad (2)$$

Note that to ensure that each frame's ZCR rate can be computed independently of that of other frames, the lower index in the summation is one larger than it is for the standard deviation.

Assignment

3. Implement the loudness and ZCR and evaluate these features on the lowest-numbered track of each genre (6 tracks total). The frames should be non overlapping. Your python function should display each feature as a time series in a separate figure (see e.g. Fig. 1).
4. Comment on the specificity of the feature and its ability to separate different musical genres.

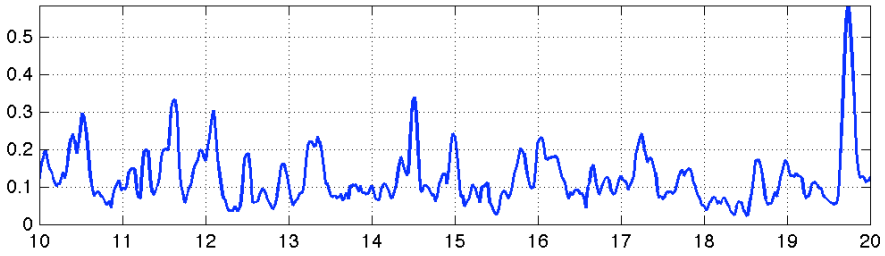

Figure 1: Zero crossing rate as a function of the frame number.

5 Low level features: spectral analysis

5.1 Windowing and spectral leaking

Music is made up of notes of different pitch. It is only natural that most of the automated analysis of music should be performed in the spectral (frequency) domain.

Our goal is the reconstruction of a musical score. Our analysis requires $N = 512$ samples to compute notes over a frame. The spectral analysis proceeds as follows. Each frame is smoothly extracted by multiplying the original audio signal by a taper window w. The Fourier transform of the windowed signal is then computed. If x_n denotes the n'th frame of size $N = 512$, and w is a window vector of size N, then the Fourier transform, X_n (of size $N/2+1$), for frame n is given conceptually by

$$\begin{aligned} Y &= \mathrm{FFT}(w * x_n); \text{(this is component-by-component multiplication)} \\ K &= N/2; \\ X_n &= Y[0 : K+1]; \end{aligned} \tag{3}$$

Note that we ignore negative frequencies in order to get a length $N/2 + 1$ set of frequency values (the FFT algorithm places negative frequencies at the upper coefficients which we are dropping). We normally convert the complex-valued FFT coefficients to power values by looking at their magnitudes. This gives a time-varying power spectrum which can be looked at as a color image called a spectrogram.

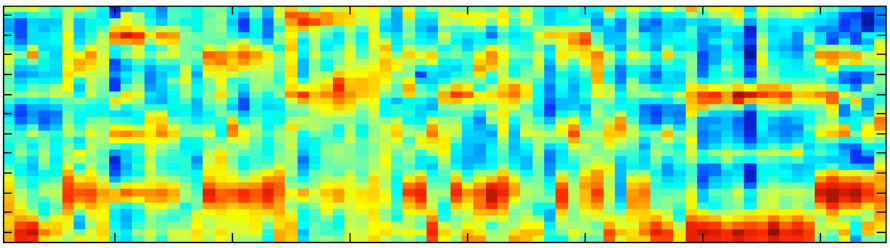

Figure 2: Spectrogram: square of the magnitude (color-coded) of the Fourier transform, $|X_n(k)|^2$, as a function of the frame index n (x-axis), and the frequency index k (y-axis).

Assignment

5. Let

$$x[n] = 1, \ -N/2 \leq n \leq N/2 \tag{4}$$
$$x[n] = 0, \text{ else} \tag{5}$$

   Derive the theoretical expression of the discrete time Fourier transform of x, given by

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n] e^{-j\omega n}. \tag{6}$$

   What is its magnitude?

6. In practice, we often multiply the signal $x[n]$ by a window $w[n]$, instead of using a window of all 1's as just analyzed above. We assume that the window w is non zero at times $n = -N/2, \ldots, N/2$, and we define

$$y[n] = x[n]w[n]. \tag{7}$$

   Derive the expression of the Fourier transform of $y[n]$ in terms of the Fourier transform of x and the Fourier transform of the window w.

Assignment

7. Compute the spectrogram of an audio track as follows:
   (a) Use a sequence of non-overlapping *frames* of size N
   (b) Use the function `scipy.signal.get_window` to compute windows of size N of type Hann and Blackman
   (c) Plot these windows
   (d) Use the function `scipy.signal.spectrogram` to generate histograms
   (e) Investigate the second 'classical' file and the 'second' world file using the Hann and Blackman windows. Do different windows produce different spectrograms?

There are several simple descriptors that can be used to characterize the spectral information provided by the Fourier transform over a frame. In the rest of the lab, we will be using a Blacman window to compute the Fourier transform.

5.2 Spectral centroid and spread

For these concepts we will treat the normalized magnitude of a spectral coefficient as if it were the *probability* that a particular frequency value occurs. In other words, for frame n, we have for the "probability" of frequency k

$$P_n(k) = \frac{|X_n(k)|}{\sum_{l=0}^{K} |X_n(l)|}. \tag{8}$$

We then define the spectral centroid (the "center of mass") of the spectrum as

$$\mu_n = \sum_{k=0}^{K} k P_n(k) \tag{9}$$

where we again think of k as the frequency.

The *spectral spread* for frame n is then the standard deviation given by

$$\sigma_n = \sqrt{\sum_{k=0}^{K} [k - \mu_n]^2 P_n(k)} \tag{10}$$

The spectral centroid can be used to quantify sound sharpness or brightness. The spread quantifies the width of the spectrum around the centroid, and thus helps differentiate between tone-like and noise-like sounds.

5.3 Spectral flatness

Spectral flatness is the ratio between the geometric and arithmetic means of the magnitude of the Fourier transform,

$$\mathrm{SF}(n) = \frac{\left(\prod_{k=0}^{K} |X_n(k)|\right)^{1/K}}{\frac{1}{K}\sum_{k=0}^{K} |X_n(k)|} \tag{11}$$

The flatness is always smaller than one since the geometric mean is always smaller than the arithmetic mean. The flatness is one, if all $|X(k)|$ are equal. This happens for a very noisy signal. A very small flatness corresponds to the presence of tonal components. In summary, this is a measure of the noisyness of the spectrum.

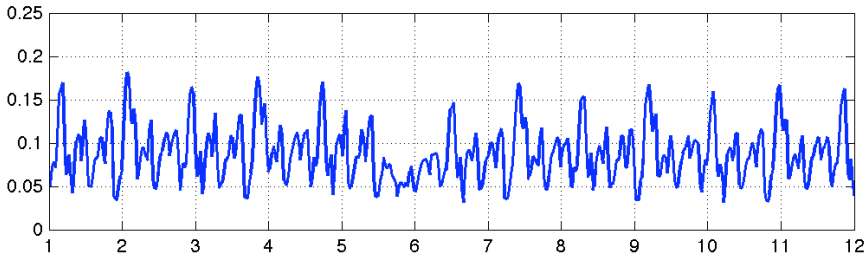

Figure 3: Spectral flatness as a function of the frame number.

5.4 Spectral flux

The spectral flux is a global measure of the spectral changes between two adjacent frames, $n-1$ and n,

$$F_n = \sum_{k=0}^{K} \left(P_n(k) - P_{n-1}(k)\right)^2 \tag{12}$$

where $P_n(k)$ is the normalized frequency distribution for frame n, given by (8).

Assignment

9. Implement all the low level spectral features and evaluate them on the first track of each genre. Your report should display each feature as a time series in a separate figure (e.g. see Fig. 3).
10. Comment on the specificity of the feature, and its ability to separate different musical genres.

5.5 Application: Mpeg7 Low Level Audio Descriptors

MPEG 7, also known as "Multimedia Content Description Interface," provides a standardized set of technologies for describing multimedia content. Part 4 of the standard specifies description tools that pertain to multimedia in the audio domain. The standard defines Low-level Audio Descriptors (LLDs) that consist of a collection of simple, low complexity descriptors to characterize the audio content. Some of the features are similar to the spectral features defined above.

6 Basic Psychoacoustic Quantities

In order to develop more sophisticated algorithms to analyze music based on its content, we need to define several subjective features such as timbre, melody, harmony, rhythm, tempo, mood, lyrics, etc. Some of these concepts can be defined formally, while others are more subjective and can be formalized using a wide variety of different algorithms. We will focus on the features that can be defined mathematically.

6.1 Psychoacoustic

Psychoacoustics involves the study of the human auditory system, and the formal quantification of the relationships between the physics of sounds, and our perception of audio. We will describe some key aspects of the human auditory system:

1. the perception of frequencies and pitch for pure and complex tones;
2. the frequency selectivity of the auditory system: our ability to perceive two similar frequencies as distinct;
3. the modeling of the auditory system as a bank of auditory filters;
4. the perception of loudness;
5. the perception of rhythm.

6.2 Perception of frequencies

The auditory system, like the visual system, is able to detect frequencies over a wide range of scales. In order to measure frequencies over a very large range, it operates using a logarithmic scale. Let us a consider a pure tone, modeled by a sinusoidal signal oscillating at a frequency f. If $f < 500$ Hz, then the perceived tone – or pitch – varies as a linear function of f. When $f > 1,000$ Hz, then the perceived pitch increases logarithmically with f.

Several frequency scales have been proposed to capture the logarithmic scaling of frequency perception.

6.3 The mel/Bark scale

The Bark scale (named after the German physicist Barkhausen) is defined as

$$z = 7 \operatorname{arcsinh}(f/650) = 7 \log\left(f/650 + \sqrt{1 + (f/650)^2}\right),$$

where f is measured in Hz. The mel-scale is defined by the fact that 1 bark = 100 mel.

In this lab we will use a slightly modified version of the mel scale defined by

$$m = 1127.01048 * \log(1 + f/700). \tag{13}$$

Note that, by construction, the m value corresponding to 1000 Hz is 1000 (the logarithm in the above formula is the natural log)

6.4 The cochlear filterbank

Finally, we need to account for the fact that the auditory system behaves as a set of filters, i.e. as a filterbank, whose filters have overlapping frequency responses. For each filter, the range of frequencies over which the filter response is significant is called a critical band. A critical band is a band of audio frequencies within which a second tone will interfere with the first via auditory masking.

Our perception of pitch can be quantified using the total energy at the output of each filter. All spectral energy that falls into one critical band is summed up, leading to a single number for that band.

We describe in the following a simple model of the cochlear filterbank. The filter bank is constructed using $N_B = 40$ logarithmically spaced triangle filters centered at the frequencies Ω_p, which are implicitly defined by

$$\text{mel}_p = 1127.01048 * \log(1 + \Omega_p/700) \tag{14}$$

The sequence of mel frequencies corresponding to the Ω_p are chosen to be equally spaced in the mel scale. Letting the indexing of the center frequencies of the filters start with $p = 1$ we have

$$\text{mel}_p = p\frac{\text{mel}_{\max} - \text{mel}_{\min}}{N_B + 1} + \text{mel}_{\min} \tag{15}$$

where N_B is the number of filters in the filterbank,

$$\text{mel}_{\text{min}} = 1127.01048 * \log\left(1 + 20/700\right), \tag{16}$$
$$\text{mel}_{\text{max}} = 1127.01048 * \log\left(1 + 0.5 * f_s/700\right), \tag{17}$$

and f_s is the sampling frequency of the audio being analyzed. Note that we have defined mel_{max} to be the m value corresponding to half the sampling frequency, which is the highest frequency preserved in the digital signal. Similarly, we have defined mel_{min} to correspond to a frequency of 20 Hz. (These min and max values are not used as center frequencies for any filters, rather, they are the upper and lower frequency limits that are covered by the filters in the filter bank.)

Each filter H_p is centered around the frequency Ω_p and is defined by

$$H_p(f) = \begin{cases} \dfrac{2}{\Omega_{p+1} - \Omega_{p-1}} \dfrac{f - \Omega_{p-1}}{\Omega_p - \Omega_{p-1}} & \text{if} \quad f \in [\Omega_{p-1}, \Omega_p), \\ \dfrac{2}{\Omega_{p+1} - \Omega_{p-1}} \dfrac{\Omega_{p+1} - f}{\Omega_{p+1} - \Omega_p} & \text{if} \quad f \in [\Omega_p, \Omega_{p+1}). \end{cases} \tag{18}$$

In the above, $\Omega_0 = 20$ and $\Omega_{N_B+1} = f_s/2$. Furthermore, Each triangular filter is normalized such that the integral of each filter is 1. In addition, the filters overlap so that the frequency at which the filter H_p is maximum is the starting frequency for the next filter H_{p+1}, and the ending frequency of H_{p-1}.

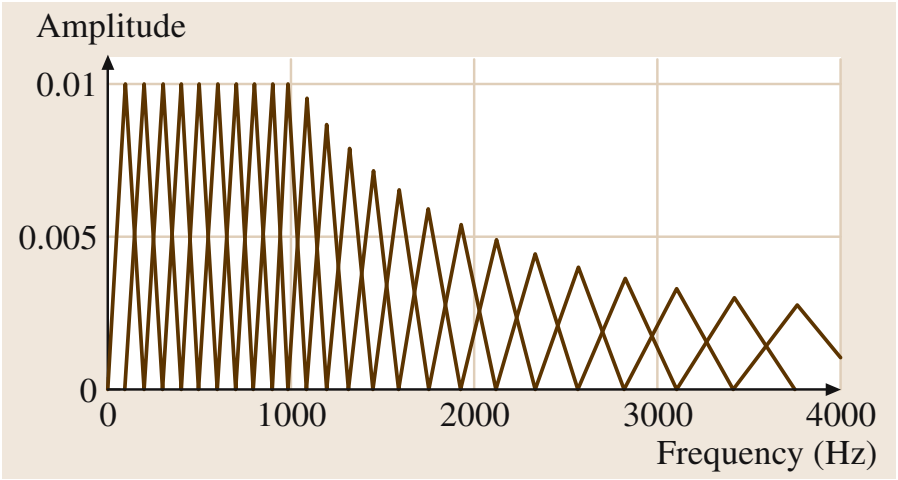

Figure 4: The filterbanks used to compute the mfcc.

Finally, when doing the calculation digitally using the FFT, the mel-spectrum (MFCC) coefficients of the n-th frame can be captured in an array defined for $p = 1, \ldots, N_B$ as

$$\texttt{mfccn}[p-1] = \sum_{k=0}^{K} |H_p(k) X_n(k)|^2 \tag{19}$$

where the $H_p(k)$ values are obtained by sampling the continuous filter defined by (18) at the frequencies corresponding to coefficient k, i.e., at $k\frac{f_s}{N}$. The discrete filter is normalized such that

$$\sum_{k=0}^{K} H_p(k) = 1 \tag{20}$$

The python code in Fig. 5 computes the sequence of frequencies Ω_p. Note that we need the frequencies Ω_0 and Ω_{N_B+1} in order to compute the filter response of the filters H_1 and H_{N_B}.

```
#!/usr/local/bin/python3
import numpy as np
import math

###########
# Here is our main block of code.  Execution starts here.
# We could EASILY compute the answer analytically in one or two lines
# of code. However, the code below illustrates
# several python functions/features that are good to know.
# In other words, it is needlessly complex but very instructional!
###########

Fs = 11025.0  # sampling frequency
nbanks = 40   # number of filters we will eventually need
mel_min = 1127.01048 * math.log(1 + 20/700)
omega_min = (math.e**(mel_min / 1127.01048) - 1) * 700
mel_max = 1127.01048 * math.log(1 + 0.5 * Fs/700)
omega_max = (math.e**(mel_max / 1127.01048) - 1) * 700
print(f"mel_min = {mel_min} and mel_max = {mel_max}")
print(f"omega_min = {omega_min} and omega_max = {omega_max}")

# Generate a numpy frequency array in Hz from omega_min to omega_max,
# including the endpoints, using a stepsize of 1
linfreq = np.arange(np.round(omega_min,0), np.round(omega_max, 0) + 1, 1)
print(f"type of linfreq is {type(linfreq)}")
print(f"linfreq = {linfreq}")

# This array holds the mel values corresponding to the
# linear frequencies in the array generated above
melfreq = 1127.01048 * np.log(1 + linfreq/700)
print(f"type of melfreq is {type(melfreq)}")
print(f"melfreq = {melfreq}")

# Generate nbanks+2 mel frequencies uniformly spaced between mel_min
# and mel_max, including mel_min and mel_max
mel = np.linspace(mel_min, mel_max, nbanks+2)
print(f"mel = {mel}")

# This array will end up holding the center frequencies of the filters
# For convenience, it will also hold omega_min at location 0 and omega_max
# at location nbanks + 1.  REMEMBER: indexing starts at 0
omega = np.zeros(nbanks + 2, np.float64)
```



```
# Now, populate the array.  The method is as follows:
# step through each
# frequency in the mel array.  Find the closest match to this frequency
# in the melfreq array.  Use the index of this closest match to look up
# the corresponding frequency in Hz in the linfreq array
for i in range(nbanks + 2) :
  idx = np.argmin( np.absolute( melfreq - mel[i] ) )
  omega[i] = linfreq[idx]
```

Assignment

11. Implement the computation of the triangular filterbanks $H_p, p = 1, \ldots, N_B$. Your function will return an array `fbank` of size $N_B \times K$ such that row $p-1$ contains the filter values for H_p. Generate a graph of the frequency responses similar to the one above
12. Implement the computation of the mfcc coefficients, as defined in (19). Include the code in your writeup. No results are necessary for now as this routine will be used in the next lab.