MSc in AI
NCSR Demokritos - University of Piraeus

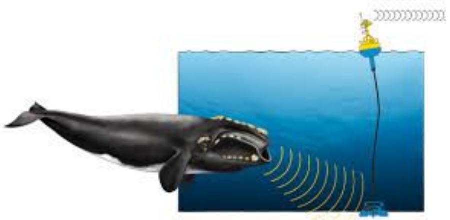
Course: **Machine Learning for Multimodal Data**

Lesson 2
Audio Representations and Feature Extraction

Theodoros Giannakopoulos

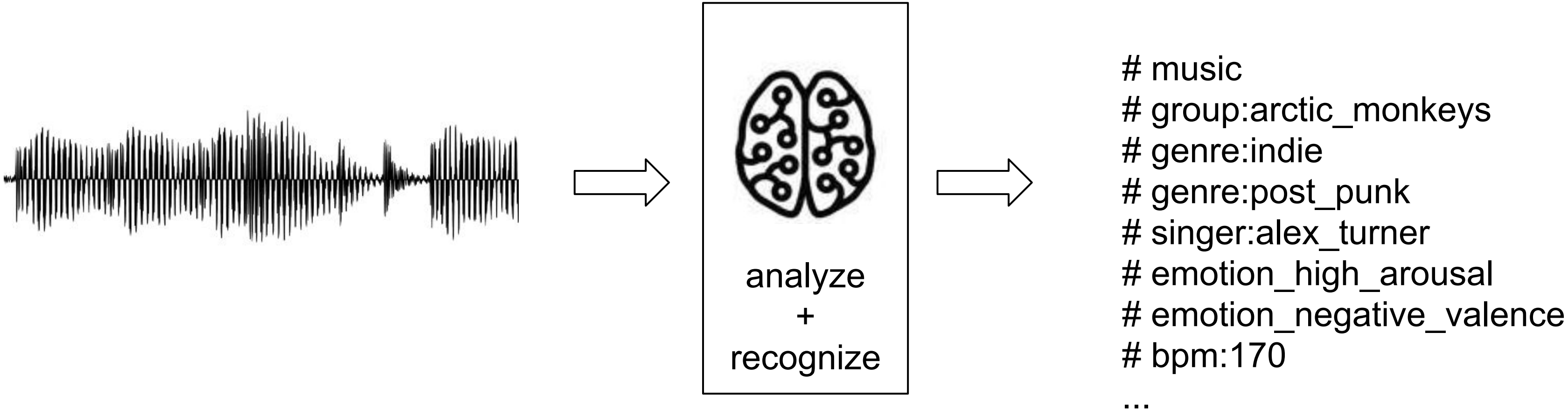
Audio Analysis Applications

Applications	Audio	Speech	Music
Automatic Speech Recognition (ASR)			
Virtual Assistants			
Music Search			
Surveillance			
Environmental Monitoring			
Recommendation			



Audio Analysis Goal & Applications

- Goal: extract high-level descriptions from raw audio signals (sounds)
- Using:
  - signal analysis: to extract features and representations
  - machine learning (supervised or unsupervised) to train models and to discover patterns
- Speech / Music / Audio
- Also referred to as “machine listening”

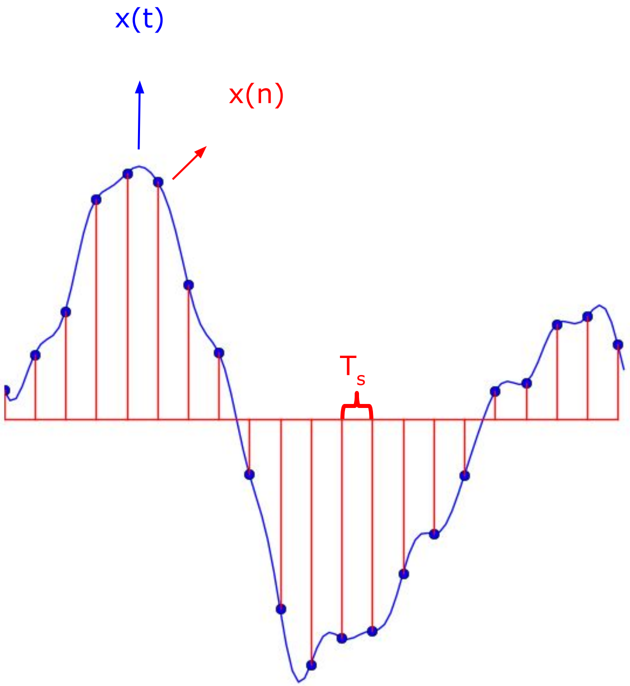


Demo code

Course's code samples will be available at this github repo:
https://github.com/tyiannak/multimodalAnalysis

What is sound?

- sound (physics):
  - a travelling vibration (wave)
  - through a medium (e.g. air)
  - transfers energy (particle to particle)
  - until "perceived" by our ears
- amplitude - loudness
- frequency - vibrations per sec
- analog sound ⟶ digital sound
  - sampling (sampling freq), fs
  - quantization (bits per sample)
  - example:
    - 44100 Hz
    - 16 bits per sample (sample resolution)
    - ~8 million integers for an average song! (single channel....)



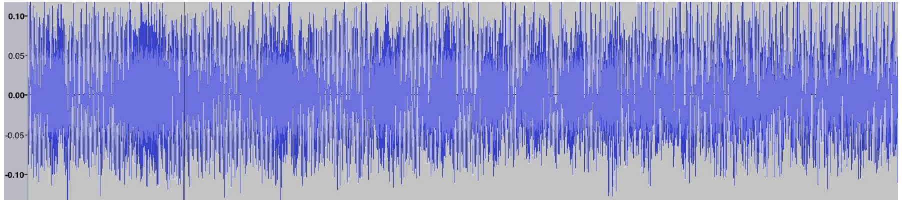
DFT

- Discrete Fourier Transform (DFT)
- A frequency domain representation of the signal (time domain)
- FFT: efficient implementation of the DFT
- Given x(n) (signal), DFT is $X(k) = \sum_{n=0}^{N-1} x(n) \exp(-j\frac{2\pi}{N}kn), k = 0, \ldots, N-1$
- Inverse $\longrightarrow$ $x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k) \exp(j\frac{2\pi}{N}kn), n = 0, \ldots, N-1,$
- Can be re-written in the form $\longrightarrow$ $x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k) y_k(n), n = 0, \ldots, N-1,$ $y_k(n) = \exp(j\frac{2\pi}{N}kn), n = 0, \ldots, N-1$
- I.e. original signal can be written as a weighted average of complex exponentials (weights are DFT coefficients)

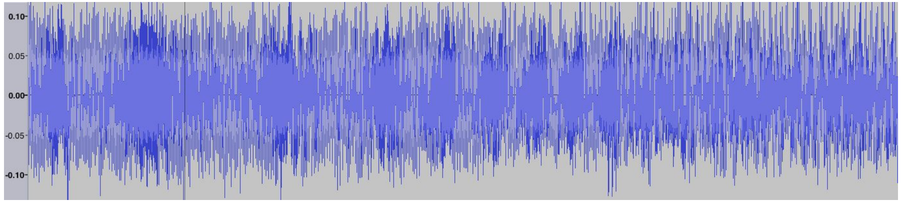
DFT is:

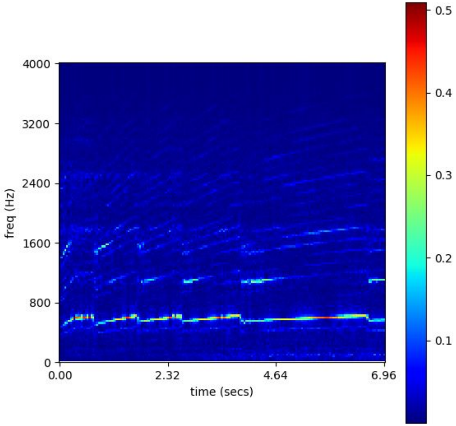
- *Defined in the range 0..fs*
- *Symmetric (center 0..fs/2)*

Representation: Time Vs Frequency



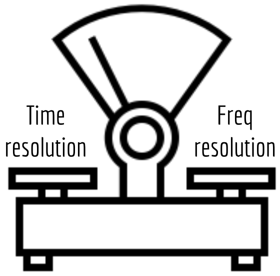
?





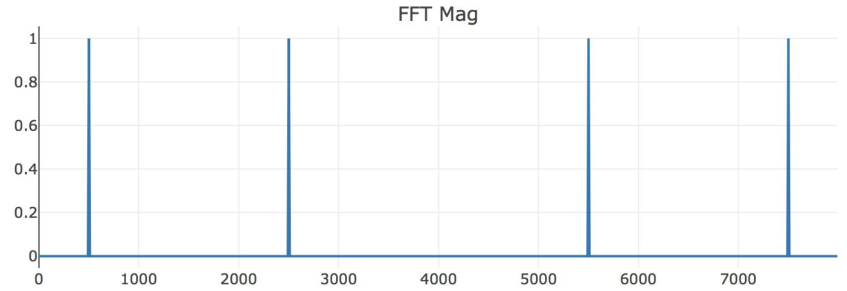
Frequency Representation: Spectrogram

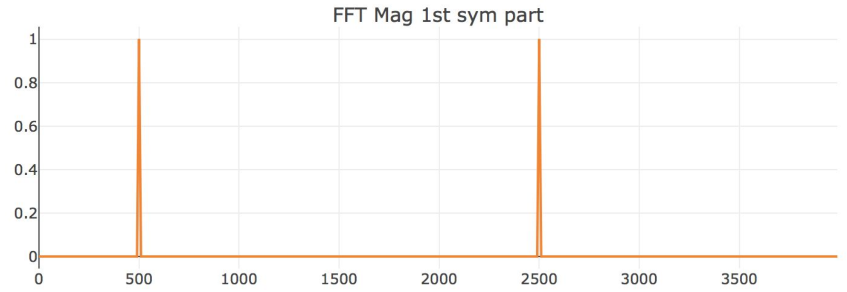
- Spectrogram: Time - Frequency 2D representation
- 1st step: Windowing
  - Signal broken into short-term windows (or frames)
  - Typically 20 to 100 mseconds
  - (Non)overlapping
  - E.g.: 50 msec frame size, 10 msec step: 80% overlap
- 2nd step: FFT
  - Fast Implementation of the Discrete Fourier Transform (DFT)
  - Not to be confused with Discrete Time Fourier Transform which:
    - Is continuous in the frequency domain
    - Is periodic in the range 0..fs
- FFT Resolution & window length:
  - longer windows → more dense representations → better frequency resolutions
  - but also losing time resolution (because of signals' non - stationarity)
  - trade off between time and frequency resolution



Frequency Representation: FFT Example

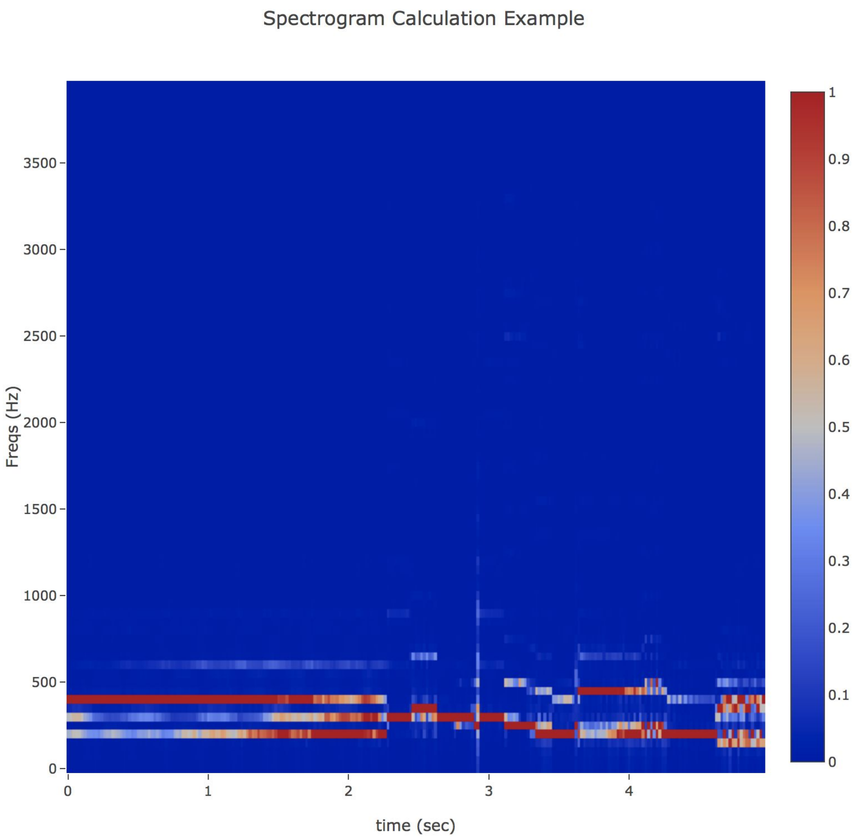
```python
"""!
@brief Example 01
@details FFT computation example
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import scipy.fftpack as scp
import numpy as np
import plotly
import plotly.graph_objs as go
if __name__ == '__main__':
    f1, f2, fs, duration = 500, 2500, 8000, 0.1
    # define time range
    t = np.arange(0, duration, 1.0/fs)
    # define signal as sum of cosines
    x = np.cos(2 * np.pi * t * f1) + np.cos(2 * np.pi * t * f2)
    # get mag of fft
    X = np.abs(scp.fft(x))
    # normalize FFT mag
    X = X / X.max()
    freqs = np.arange(0, 1, 1.0/len(X)) * fs
    # get 1st symmetric part
    freqs_1 = freqs[0:int(len(freqs)/2)]
    X_1 = X[0:int(len(X)/2)]
    figs = plotly.tools.make_subplots(rows=2, cols=1,
                                      subplot_titles=["FFT Mag",
                                                      "FFT Mag 1st sym part"])
    figs.append_trace(go.Scatter(x=freqs, y=X, showlegend=False), 1, 1)
    figs.append_trace(go.Scatter(x=freqs_1, y=X_1, showlegend=False), 2, 1)
    plotly.offline.plot(figs, filename="temp.html", auto_open=True)
```





Frequency Representation: Spectrogram Example

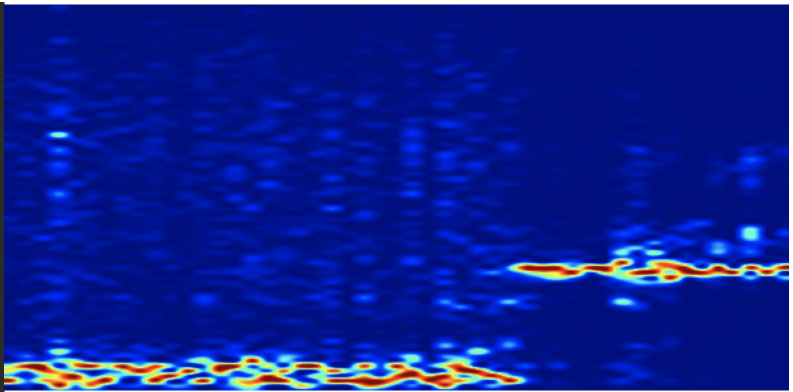
```python
"""!
@brief Example 02
@details Example of spectrogram computation for a wav file, using only scipy
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import scipy.fftpack as scp
import numpy as np
import scipy.io.wavfile as wavfile
import plotly
import plotly.graph_objs as go
layout = go.Layout(title='Spectrogram Calculation Example',
                   xaxis=dict(title='time (sec)',),
                   yaxis=dict(title='Freqs (Hz)',))


def get_fft_spec(signal, fs, win):
   frame_size, signal_len, spec, times = int(win * fs), len(signal), [], []
   # break signal into non-overlapping short-term windows (frames)
   frames = np.array([signal[x:x + frame_size] for x in
                      np.arange(0, signal_len - frame_size, frame_size)])
   for i_f, f in enumerate(frames): # for each frame
       times.append(i_f * win)
       # append mag of fft
       X = np.abs(scp.fft(f)) ** 2
       freqs = np.arange(0, 1, 1.0/len(X)) * (fs/2)
       spec.append(X[0:int(len(X)/2)] / X.max())
   return np.array(spec).T, freqs, times

if __name__ == '__main__':
   [Fs, s] = wavfile.read("../data/sample_music.wav")
   S, f, t = get_fft_spec(s, Fs, 0.02)
   heatmap = go.Heatmap(z=S, y=f, x=t)
   plotly.offline.plot(go.Figure(data=[heatmap], layout=layout),
                       filename="temp.html", auto_open=True)
```

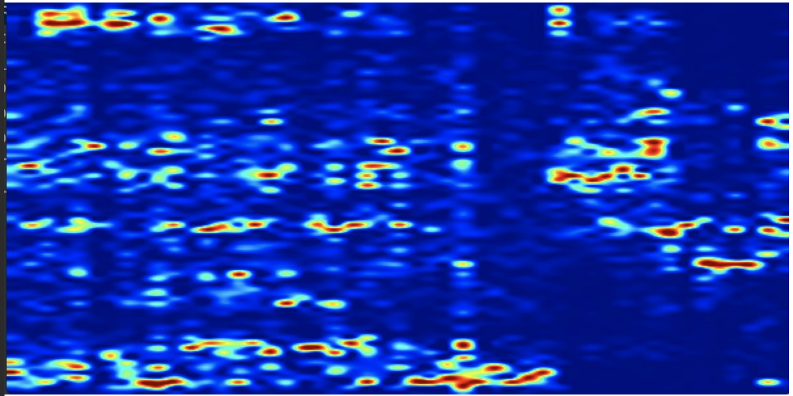


- **scipy.fftpack** used for fft

Frequency Representation: Spectrogram Recording Example

```python
"""!
@brief Example 03
@details Example of audio recording and spectrogram computation
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import numpy as np
import scipy.io.wavfile as wavfile
import example02, pyaudio, struct, cv2, signal, sys

fs = 8000
bufSize = int(fs * 1.0)
def signal_handler(signal, x):
    wavfile.write("output.wav", fs, np.int16(aggregated_buf))
    sys.exit(0)
signal.signal(signal.SIGINT, signal_handler)

if __name__ == '__main__':
    global aggregated_buf; aggregated_buf = np.array([])
    pa = pyaudio.PyAudio() # initialize recording
    stream = pa.open(format=pyaudio.paInt16 , channels=1, rate=fs,
                     input=True, frames_per_buffer=bufSize)
    while 1:
        # read recorded data, convert bytes to samples and then to numpy array
        block = stream.read(bufSize, exception_on_overflow=False)
        s = np.array(list(struct.unpack("%dh"%(len(block)/2), block))).astype(float)
        aggregated_buf = np.concatenate((aggregated_buf, s))
        s /= (2 ** 15)
        # get spectrogram and visualize it using opencv
        specgram, f, t = example02.get_fft_spec(s, fs, 0.02)
        iSpec = np.array(specgram[::-1] * 255, dtype=np.uint8)
        iSpec2 = cv2.resize(iSpec, (600, 350), interpolation=cv2.INTER_CUBIC)
        iSpec2 = cv2.applyColorMap(iSpec2, cv2.COLORMAP_JET)
        cv2.imshow('Spectrogram', iSpec2)
        cv2.moveWindow('Spectrogram', 100, 100)
        ch = cv2.waitKey(10)
```



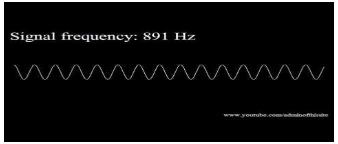


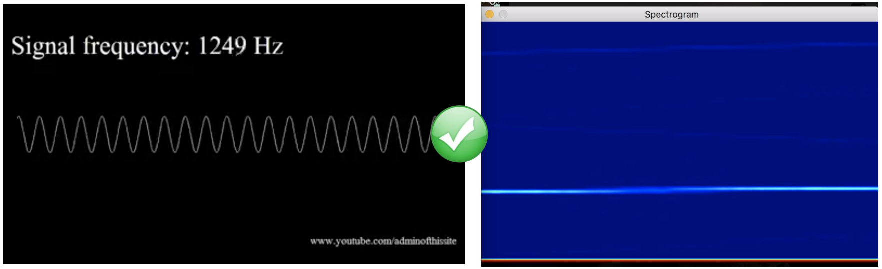
scipy.fftpack used for fft (imported from example2)
pyaudio used for data acquisition (recording)
opencv used for fast online, non-blocking visualization (matplotlib slower)

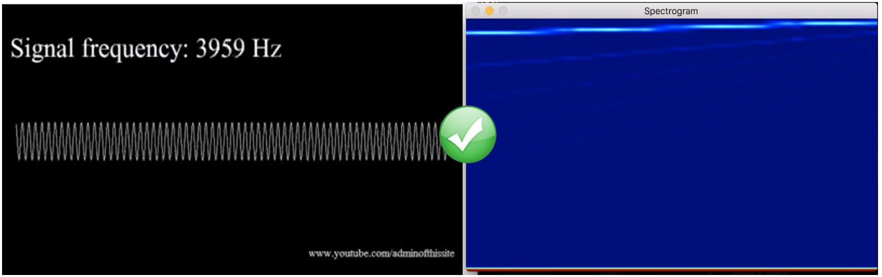
Frequency Representation: Spectrogram Recording Example

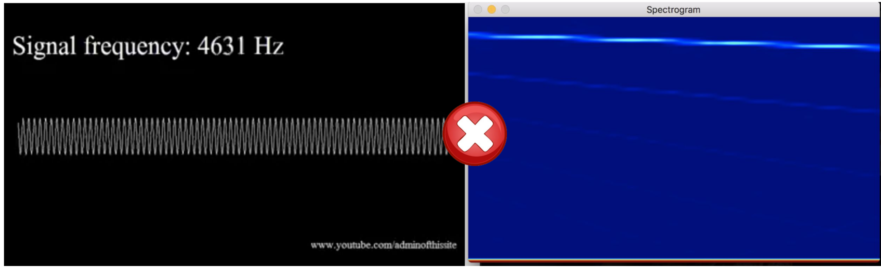
Sampling freq / Aliasing ?

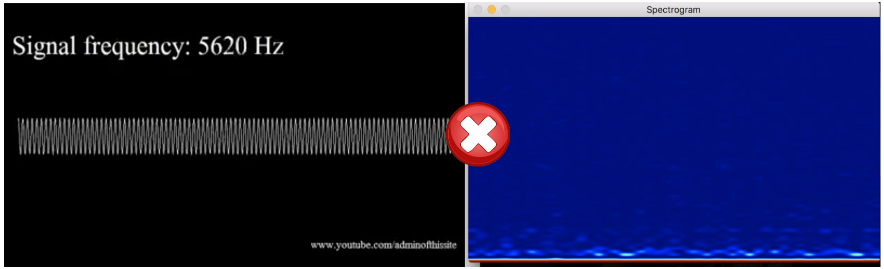
Check the example03.py script on this sound:











- Fs = 8000 Hz ⟶ $F_{nyquist}$ = 4000 Hz
- Freqs after the Nyquist Freq are not captured
- Run experiment on paura

Open-source libraries for audio signal analysis

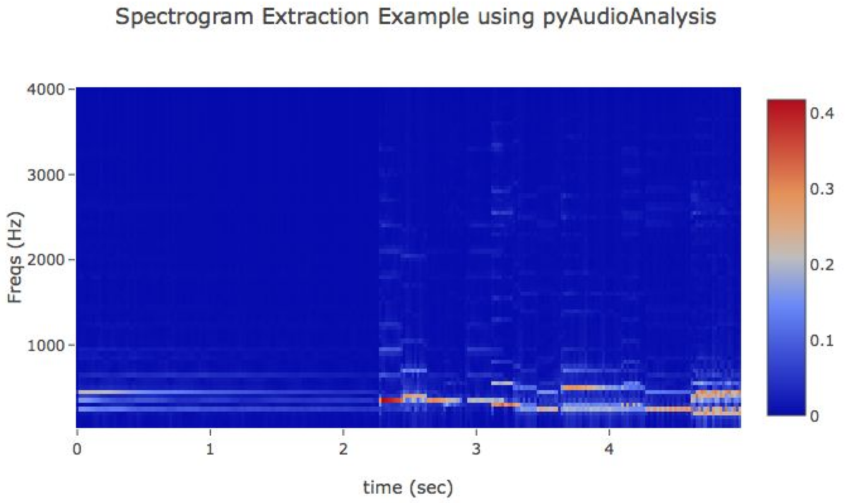
- librosa (Python)
  - https://librosa.github.io
  - Implements various audio features (mfccs, chroma, beat, etc)
  - Audio fx (e.g. pitch shift)
  - Some ML components
- pyAudioAnalysis (Python)
  - https://github.com/tyiannak/pyAudioAnalysis
  - Implements various audio features
  - Built-in training/testing of classifiers (using scikit-learn)
  - Clustering (speaker diarization)
  - Visualization
- Opensmile (C++)
  - https://audeering.com/technology/opensmile/
  - Richest set of audio features

Spectrogram calculation using pyAudioAnalysis

```
"""!
@brief Example 04
@details pyAudioAnalysis spectrogram calculation and visualization example
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import numpy as np
import scipy.io.wavfile as wavfile
import plotly
import plotly.graph_objs as go
from pyAudioAnalysis import ShortTermFeatures as aF
layout = go.Layout(title='Spectrogram Extraction Example using pyAudioAnalysis',
                   xaxis=dict(title='time (sec)',),
                   yaxis=dict(title='Freqs (Hz)',))

def normalize_signal(signal):
   signal = np.double(signal)
   signal = signal / (2.0 ** 15)
   return (signal - signal.mean()) / ((np.abs(signal)).max() + 0.0000000001)

if __name__ == '__main__':
   [Fs, s] = wavfile.read("../data/sample_music.wav")
   s = normalize_signal(s)
   [S, t, f] = aF.spectrogram(s, Fs, int(Fs * 0.020), int(Fs * 0.020))
   heatmap = go.Heatmap(z=S.T, y=f, x=t)
   plotly.offline.plot(go.Figure(data=[heatmap], layout=layout),
                       filename="temp.html", auto_open=True)
```

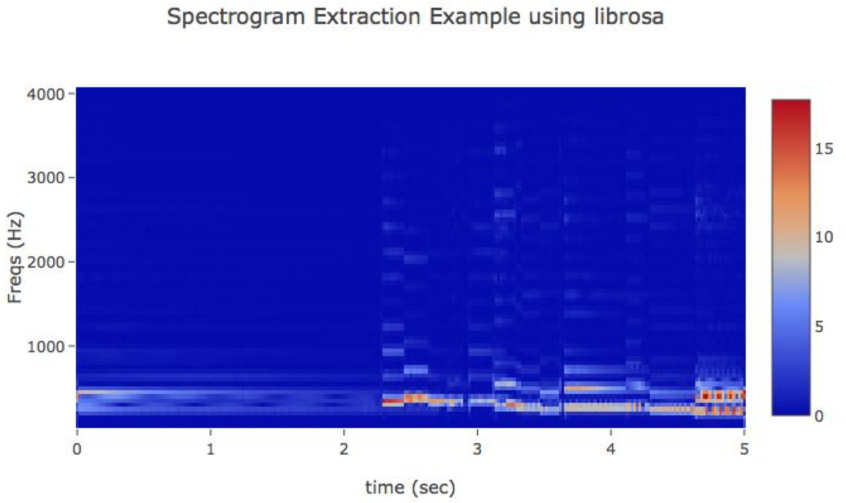
Only for 16-bit sample resolution

Also returns time and frequency scales (in Hz and secs respectively)



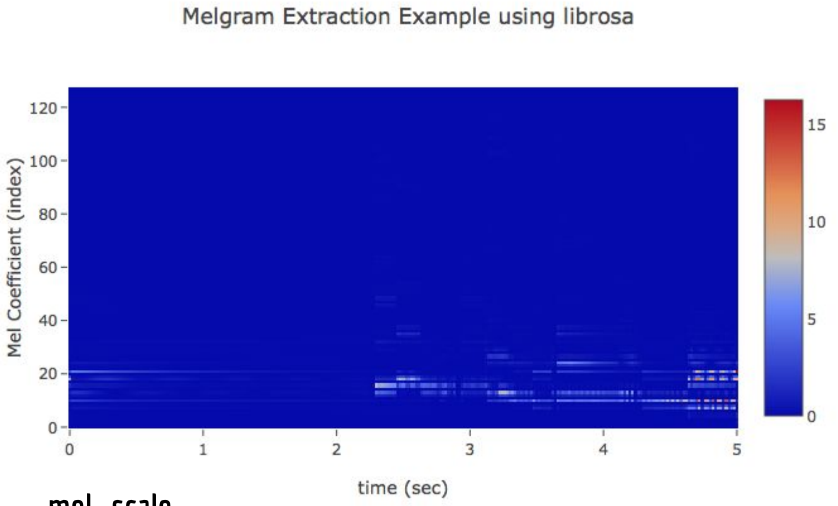
Spectrogram calculation using librosa

```python
"""!
@brief Example 05
@details librosa spectrogram calculation and visualization example
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import numpy as np
import scipy.io.wavfile as wavfile
import plotly
import librosa
import plotly.graph_objs as go
layout = go.Layout(title='Spectrogram Extraction Example using librosa',
                   xaxis=dict(title='time (sec)',),
                   yaxis=dict(title='Freqs (Hz)',))

def normalize_signal(signal):
    signal = np.double(signal)
    signal = signal / (2.0 ** 15)
    return (signal - signal.mean()) / ((np.abs(signal)).max() + 0.0000000001)

if __name__ == '__main__':
    [Fs, s] = wavfile.read("../data/sample_music.wav")
    s = normalize_signal(s)
    S = np.abs(librosa.stft(s, int(Fs * 0.020), int(Fs * 0.020)))
    # create frequency and time axes
    f = [float((f + 1) * Fs) / (int(Fs * 0.020)) for f in range(S.shape[0])]
    t = [float(t * int(Fs * 0.020)) / Fs for t in range(S.shape[1])]
    heatmap = go.Heatmap(z=S, y=f, x=t)
    plotly.offline.plot(go.Figure(data=[heatmap], layout=layout),
                        filename="temp.html", auto_open=True)
```



Melgram calculation using librosa

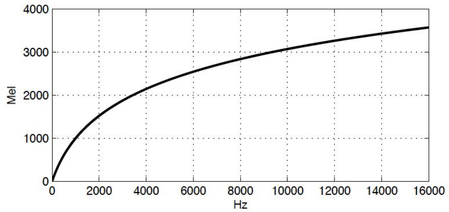
```python
"""!
@brief Example 06
@details librosa spectrogram calculation and visualization example
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import numpy as np
import scipy.io.wavfile as wavfile
import plotly
import librosa
import plotly.graph_objs as go
layout = go.Layout(title='Melgram Extraction Example using librosa',
                   xaxis=dict(title='time (sec)',),
                   yaxis=dict(title='Mel Coefficient (index)',))

def normalize_signal(signal):
   signal = np.double(signal)
   signal = signal / (2.0 ** 15)
   return (signal - signal.mean()) / ((np.abs(signal)).max() + 0.0000000001)

if __name__ == '__main__':
   [Fs, s] = wavfile.read("../data/sample_music.wav")
   s = normalize_signal(s)
   S = librosa.feature.melspectrogram(s, Fs, None, int(Fs * 0.020),
                                      int(Fs * 0.020), power=2)
   # create frequency and time axes
   f = range(S.shape[0])
   t = [float(t * int(Fs * 0.020)) / Fs for t in range(S.shape[1])]
   heatmap = go.Heatmap(z=S, y=f, x=t)
   plotly.offline.plot(go.Figure(data=[heatmap], layout=layout),
                       filename="temp.html", auto_open=True)
```



mel- scale

- Conform with psychoacoustic observations
- The human auditory system can distinguish neighboring frequencies more easily in the low frequency regions

$$f_w = 1127.01048 * \log(f/700 + 1)$$



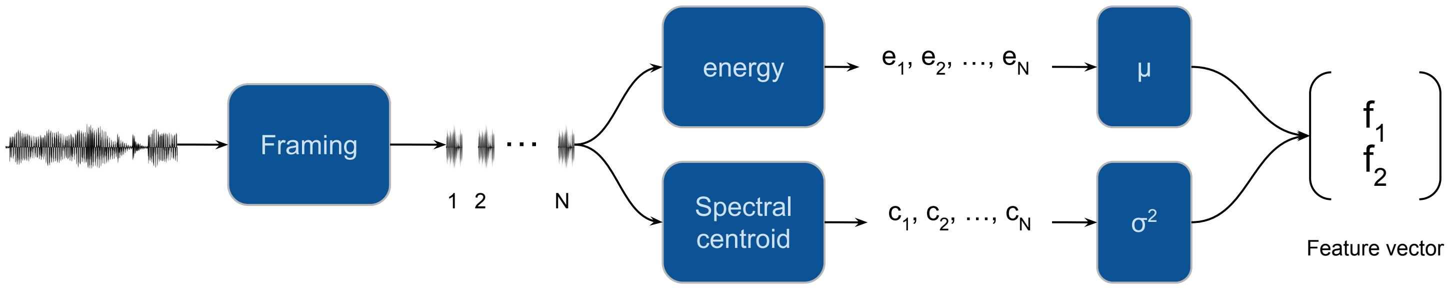
Why Mel?

```python
"""!
@brief Example 07
@details Frequency prerceived discrimination experiment
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
from __future__ import print_function
import os, time, scipy.io.wavfile as wavfile, numpy as np
from random import randint

def play_sound(freq, duration, fs):
    t = np.arange(0, duration, 1.0/fs);  x = 0.5*np.cos(2 * np.pi * t * freq)
    wavfile.write("temp.wav", fs, x); os.system("play temp.wav -q")

if __name__ == '__main__':
    freqs, thres, n_exp, fs = [250, 500, 1000, 2000, 3000], [2, 5, 10, 20], 10, 16000
    answers = [[] for i in range(len(freqs))]
    for i_f, f in enumerate(freqs):
        for t in thres:
            answers[i_f].append(0)
            for i in range(n_exp):
                sequel = randint(1, 2)
                if sequel == 2:
                    play_sound(f, 0.5, fs); time.sleep(0.5); play_sound(f+t, 0.5, fs)
                else:
                    play_sound(f+t, 0.5, fs); time.sleep(0.5); play_sound(f, 0.5, fs)
                ans = int(raw_input('Which was higher (1/2):'))
                if ans == sequel: answers[i_f][-1] += 1
    print("Freq\t", end='')
    for t in thres: print("{0:.1f}\t".format(t), end='')
    print("")
    for i_f, f in enumerate(freqs):
        print("{} Hz\t".format(f), end='')
        for i_t, t in enumerate(thres):
            print("{0:.1f}\t".format(answers[i_f][i_t] / float(n_exp)), end='')
        print("")
```

	Thresholds			
Freq	2 Hz	5 Hz	10 Hz	20 Hz
250 Hz	0.7	1	1	1
500 Hz	0.4	0.8	0.9	1
1000 Hz	0.6	0.8	1	0.9
2000 Hz	0.5	0.4	0.9	1
3000 Hz	0.5	0.5	0.6	1

Audio segment feature extraction

- Short-term windowing:
  - “frames”
  - extract features per frame (such as energy, or spectral centroid)
  - result: sequence of vectors (one vector for each frame)
- Segment windowing:
  - segments are either predefined or applied on long recordings (e.g. fix-sized)
  - each segment corresponds to a sequence of short-term feature vectors
  - common practice
    - extract segment (mid-term) statistics (μ, σ^2)
    - applied per sequence of short-term feature sequence (in the segment)



Audio features: Segment Statistics

- Each feature is extracted in a short-term basis
- Segment feature statistics capture temporal changes in short-term feature sequences
- Statistics:
    - mean value
    - std/var
    - percentiles
    - max / min
    - Skewness
- Examples:
    - average zero crossing rate
    - deviation of the spectral centroid

Time-domain features

- Energy
  - usually normalized by window length
  - high variation over successive speech frames (std statistic)

$$E(i) = \frac{1}{W_L}\sum_{n=1}^{W_L}|x_i(n)|^2$$

- Zero Crossing Rate
  - rate of sign changes during the frame
  - measure of noisiness
  - high values for noisy signals

$$sgn[x_i(n)] = \begin{cases} 1, & x_i(n) \geq 0 \\ -1, & x_i(n) < 0 \end{cases} \qquad Z(i) = \frac{1}{2W_L}\sum_{n=1}^{W_L} |\, sgn[x_i(n)] - sgn[x_i(n-1)] \,|$$

- Energy Entropy
  - measure of abrupt changes in the signal's energy
  - divide frames to K sub-frames and compute (normalized) sub-energies ($e_{subframe\_k}$)
  - compute entropy of $e_{subframe\_k}$ sequence

$$e_j = \frac{E_{subFrame_j}}{E_{shortFrame_i}} \qquad E_{shortFrame_i} = \sum_{k=1}^{K} E_{subFrame_k},$$

$$H(i) = -\sum_{j=1}^{K} e_j^2 \cdot \log_2(e_j^2)$$

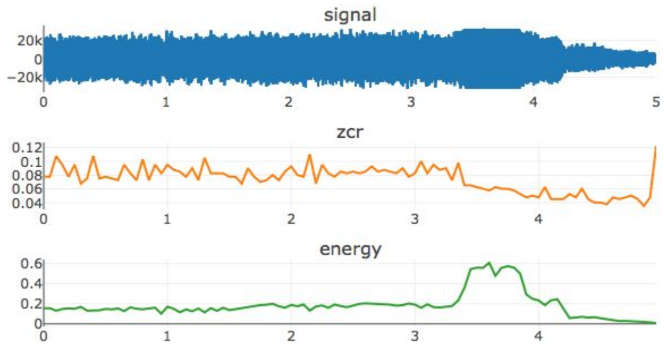
Time-domain features - Example

```python
"""!
@brief Example 08
@details pyAudioAnalysis feature extraction example 1
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import numpy as np
import plotly
import plotly.graph_objs as go
from pyAudioAnalysis import ShortTermFeatures as aF
from pyAudioAnalysis import audioBasicIO as aIO


if __name__ == '__main__':
    # read machine sound
    fs, s = aIO.read_audio_file("../data/general/objects/1-46744-A.ogg.wav")
    duration = len(s) / float(fs)
    # extract short term features and plot ZCR and Energy
    [f, fn] = aF.feature_extraction(s, fs, int(fs * 0.050), int(fs * 0.050))
    figs = plotly.tools.make_subplots(rows=3, cols=1,
                                      subplot_titles=["signal", fn[0], fn[1]])
    time = np.arange(0, duration - 0.050, 0.050)
    time_s = np.arange(0, duration, 1/float(fs))
    figs.append_trace(go.Scatter(x=time_s, y=s, showlegend=False), 1, 1)
    figs.append_trace(go.Scatter(x=time, y=f[0, :], showlegend=False), 2, 1)
    figs.append_trace(go.Scatter(x=time, y=f[1, :], showlegend=False), 3, 1)
    plotly.offline.plot(figs, filename="temp.html", auto_open=True)
```

Also returns list of short-term feature names

This is a #n_frames x #n_wins matrix

- using pyAudioAnalysis
- short-term feature sequences for ZCR / Energy
- sound: **vacuum cleaner**
- zcr is higher for “noisy” sounds



Audio features: Frequency Domain (Spectral) Features

- Let X be the abs(FFT)
- Spectral Centroid
  - Center of gravity of the spectrum

$$C_i = \frac{\sum_{k=1}^{Wf_L}(k+1)X_i(k)}{\sum_{k=1}^{Wf_L}X_i(k)}$$

- Spectral spread
  - 2nd central moment of the spectrum

$$S_i = \sqrt{\frac{\sum_{k=1}^{Wf_L}((k+1)-C_i)^2 X_i(k)}{\sum_{k=1}^{Wf_L}X_i(k)}}$$

- Spectral entropy
  - Divide spectrum into L sub-bands
  - Compute normalized sub-band energies (E_f)
  - Compute entropy

$$n_f = \frac{E_f}{\sum_{f=0}^{L-1}E_f},\ f = 0, \dots, L-1. \qquad H = -\sum_{f=0}^{L-1} n_f \cdot log_2(n_f)$$

- Spectral Flux
  - Spectral change between two successive frames

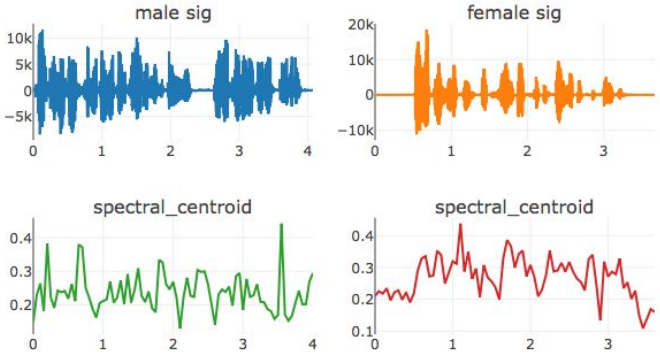
$$EN_i(k) = \frac{X_i(k)}{\sum_{l=1}^{Wf_L}X_i(l)} \qquad Fl_{(i,i-1)} = \sum_{k=1}^{Wf_L}(EN_i(k) - EN_{i-1}(k))^2$$

- Spectral Rolloff
  - Freq below which a percentage of the mag distribution of the spectrum is concentrated
  - If the m-th DFT coefficient is the spectral rolloff ⟶

$$\sum_{k=1}^{m} X_i(k) = C\sum_{k=1}^{Wf_L} X_i(k)$$

Audio features: Frequency Domain (Spectral) Features - Example

```python
"""!
@brief Example 09
@details pyAudioAnalysis feature extraction example for male / female speeches
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import numpy as np
import plotly
import plotly.graph_objs as go
from pyAudioAnalysis import ShortTermFeatures as aF
from pyAudioAnalysis import audioBasicIO as aIO


if __name__ == '__main__':
    win = 0.05
    fp1 = "../data/general/speech/m1_neu-m1-l1.wav.wav" # male
    fp2 = "../data/general/speech/f1_neu-f1-l2.wav.wav" # female
    # read machine sound
    fs1, s1 = aIO.read_audio_file(fp1)
    fs2, s2 = aIO.read_audio_file(fp2)
    dur1, dur2 = len(s1) / float(fs1), len(s2) / float(fs2)
    # extract short term features
    [f1, fn] = aF.feature_extraction(s1, fs1, int(fs1 * win), int(fs1 * win))
    [f2, fn] = aF.feature_extraction(s2, fs2, int(fs2 * win), int(fs2 * win))
    figs = plotly.tools.make_subplots(rows=2, cols=2,
                                      subplot_titles=["male sig", "female sig",
                                                      fn[3], fn[3]])
    t1 = np.arange(0, dur1 - win, win)
    ts_1 = np.arange(0, dur1, 1/float(fs1))
    t2 = np.arange(0, dur2 - win, win)
    ts_2 = np.arange(0, dur2, 1/float(fs2))
    figs.append_trace(go.Scatter(x=ts_1, y=s1, showlegend=False), 1, 1)
    figs.append_trace(go.Scatter(x=ts_2, y=s2, showlegend=False), 1, 2)
    figs.append_trace(go.Scatter(x=t1, y=f1[3, :], showlegend=False), 2, 1)
    figs.append_trace(go.Scatter(x=t2, y=f2[3, :], showlegend=False), 2, 2)
    plotly.offline.plot(figs, filename="temp.html", auto_open=True)
```



Audio features: Cepstral Domain

- Mel-Frequency Cepstral Coefficients
  - Compute DFT
  - Mel-scale filter bank application
  - Compute O_k as the power of the output of each filter
  - Compute MFCCs as the discrete cosine transform coefficients of the mel-scaled log-power spectrum
- Usually select the first 13 MFCCs (considered to carry enough discriminative information especially for speech classification tasks)
- Cepstrum in general (not mel):
  - Inverse fft of the log fft

$$f_w = 1127.01048 * \log(f/700 + 1)$$

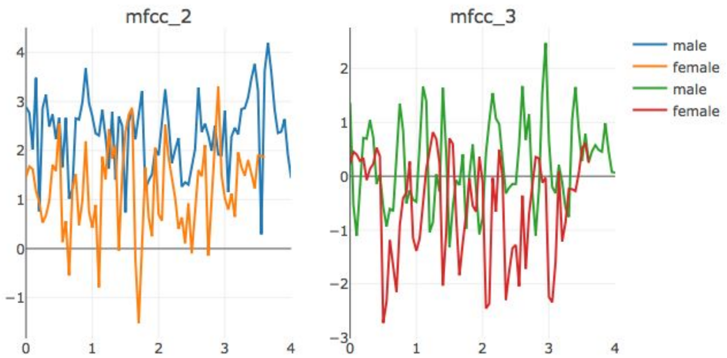
$$c_m = \sum_{k=1}^{L} (\log \widetilde{O_k}) \cos[m(k - \frac{1}{2})\frac{\pi}{L}], \quad m = 1, \ldots, L$$

$$\left|\mathcal{F}^{-1}\left\{\log\left(|\mathcal{F}\{f(t)\}|^2\right)\right\}\right|^2$$

signal → fft → log → ifft → cepstrum

Audio features: Cepstral Domain - Example

```python
"""!
@brief Example 10
@details pyAudioAnalysis feature extraction example  - mfccs for male/female
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import numpy as np
import plotly
import plotly.graph_objs as go
from pyAudioAnalysis import audioFeShortTermFeaturesatureExtraction as aF
from pyAudioAnalysis import audioBasicIO as aIO


if __name__ == '__main__':
   win = 0.05
   fp1 = "../data/general/speech/m1_neu-m1-l1.wav.wav" # male
   fp2 = "../data/general/speech/f1_neu-f1-l2.wav.wav" # female
   # read machine sound
   fs1, s1 = aIO.read_audio_file(fp1)
   fs2, s2 = aIO.read_audio_file(fp2)
   dur1, dur2 = len(s1) / float(fs1), len(s2) / float(fs2)
   # extract short term features
   [f1, fn] = aF.feature_extraction(s1, fs1, int(fs1 * win), int(fs1 * win))
   [f2, fn] = aF.feature_extraction(s2, fs2, int(fs2 * win), int(fs2 * win))
   figs = plotly.tools.make_subplots(rows=1, cols=2,
                                     subplot_titles=[fn[9], fn[10]])
   t1 = np.arange(0, dur1 - 0.050, 0.050)
   t2 = np.arange(0, dur2 - 0.050, 0.050)
   figs.append_trace(go.Scatter(x=t1, y=f1[9, :], name="male"), 1, 1)
   figs.append_trace(go.Scatter(x=t2, y=f2[9, :], name="female"), 1, 1)
   figs.append_trace(go.Scatter(x=t1, y=f1[10, :], name="male"), 1, 2)
   figs.append_trace(go.Scatter(x=t2, y=f2[10, :], name="female"), 1, 2)

   plotly.offline.plot(figs, filename="temp.html", auto_open=True)
```



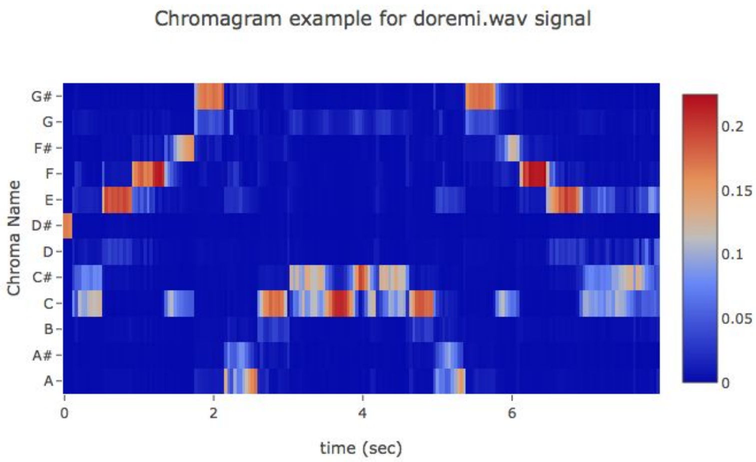
Audio features: Chroma Vector

- 12-element frequency representation
- In music applications
- Group the DFT coefficients of a window into 12 bins
- Each bin represents the 12 equal-tempered classes of western-type music
- Bins in semitone spacing
- Sk is the set of frequencies for the k-th bin (representing DFT coefficients)

$$v_k = \sum_{n \in S_k} \frac{X_i(n)}{N_k}, k \in 0..11$$

Audio features: Chroma Vector - Example

```python
"""!
@brief Example 11
@details pyAudioAnalysis chromagram example
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import plotly
import plotly.graph_objs as go
from pyAudioAnalysis import ShortTermFeatures as aF
from pyAudioAnalysis import audioBasicIO as aIO
layout = go.Layout(title='Chromagram example for doremi.wav signal',
                   xaxis=dict(title='time (sec)',),
                   yaxis=dict(title='Chroma Name',))


if __name__ == '__main__':
   win = 0.04
   fp = "../data/doremi.wav" # music sample
   # read machine sound
   fs, s = aIO.read_audio_file(fp)
   fs = float(fs)
   dur1 = len(s) / float(fs)
   spec, time, freq = aF.chromagram(s, fs, int(fs * win),
                                    int(fs * win), False)
   heatmap = go.Heatmap(z=spec.T, y=freq, x=time)
   plotly.offline.plot(go.Figure(data=[heatmap], layout=layout),
                       filename="temp.html", auto_open=True)
```



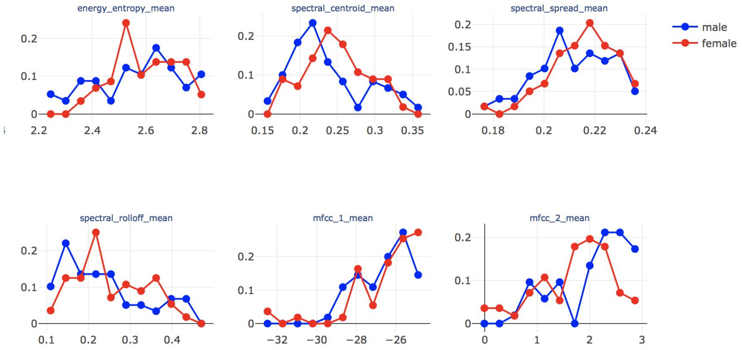
Plotly histogram representation (function in utilities.py)

```python
import plotly
import plotly.graph_objs as go
import numpy as np
import matplotlib.pyplot as plt

def plot_feature_histograms(list_of_feature_mtr, feature_names,
                            class_names, n_columns=5):
    '''
    Plots the histograms of all classes and features for a given
    classification task.
    :param list_of_feature_mtr: list of feature matrices
                                (n_samples x n_features) for each class
    :param feature_names:       list of feature names
    :param class_names:         list of class names, for each feature matr
    '''
    clr_map = plt.cm.get_cmap('jet')
    n_features = len(feature_names)
    n_bins = 12
    n_classes = len(class_names)
    n_rows = int(n_features / n_columns) + 1
    figs = plotly.tools.make_subplots(rows=n_rows, cols=n_columns,
                                      subplot_titles=feature_names)
    figs['layout'].update(height=(n_rows * 250))
    range_cl = range(int(int(255/n_classes)/2), 255, int(255/n_classes))
    clr = []
    for i in range(n_classes):
        clr.append('rgba({},{},{},{})'.format(clr_map(range_cl[i])[0],
                                              clr_map(range_cl[i])[1],
                                              clr_map(range_cl[i])[2],
                                              clr_map(range_cl[i])[3]))
    for i in range(n_features):
        # for each feature get its bin range (min:(max-min)/n_bins:max)
        f = np.vstack([x[:, i:i + 1] for x in list_of_feature_mtr])
        bins = np.arange(f.min(), f.max(), (f.max() - f.min()) / n_bins)
        for fi, f in enumerate(list_of_feature_mtr):
            # load the color for the current class (fi)
            mark_prop = dict(color=clr[fi], line=dict(color=clr[fi], width=3))
            # compute the histogram of the current feature (i) and normalize:
            h, _ = np.histogram(f[:, i], bins=bins)
            h = h.astype(float) / h.sum()
            cbins = (bins[0:-1] + bins[1:]) / 2
            scatter_1 = go.Scatter(x=cbins, y=h, name=class_names[fi],
                                   marker=mark_prop, showlegend=(i == 0))
            # (show the legend only on the first line)
            figs.append_trace(scatter_1, int(i/n_columns)+1, i % n_columns+1)
    for i in figs['layout']['annotations']:
        i['font'] = dict(size=10, color='#224488')
    plotly.offline.plot(figs, filename="report.html", auto_open=True)
```

https://plot.ly/python/

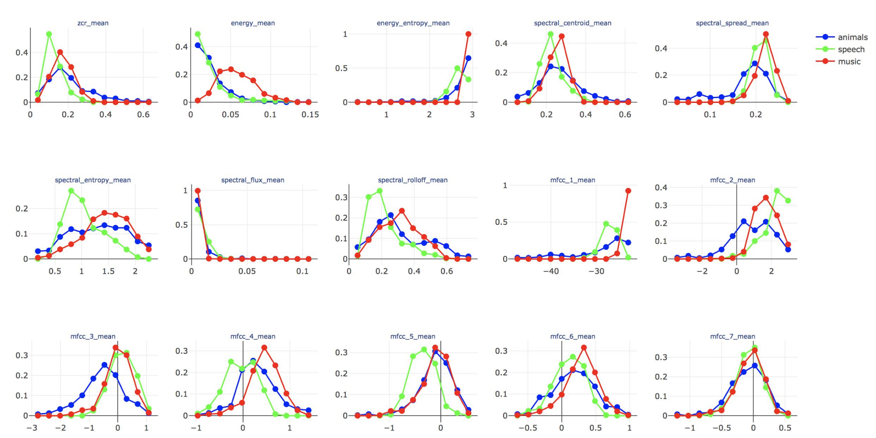
Feature discrimination example: male vs female segments

```python
"""!
@brief Example 12
@details pyAudioAnalysis feature extraction for classes organized in folders
and feature histogram representation (per feature and class).
Binary classification task: male vs female speech segments
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
from pyAudioAnalysis import MidTermFeatures as aF
import os.path
import utilities as ut

if __name__ == '__main__':
   dirs = ["../data/gender/male",
           "../data/gender/female"]
   class_names = [os.path.basename(d) for d in dirs]
   m_win, m_step, s_win, s_step = 1, 1, 0.1, 0.05
   features = []
   for d in dirs:
       # get feature matrix for each directory (class)
       f, files, fn = aF.directory_feature_extraction(d, m_win, m_step, s_win,
                                                      s_step)
       features.append(f)
   ut.plot_feature_histograms(features, fn, class_names)
```



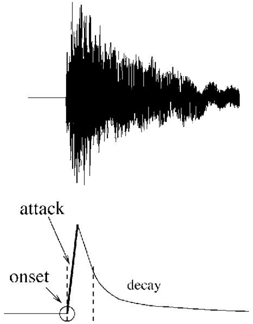
Feature discrimination example: 3-class task

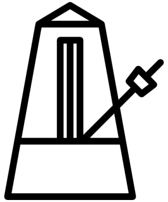
```
"""!
@brief Example 13
@details pyAudioAnalysis feature extraction for classes organized in folders
and feature histogram representation (per feature and class).
3-class classification task: animals vs speech vs music segments
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
from pyAudioAnalysis import MidTermFeatures as aF
import os.path
import utilities as ut

if __name__ == '__main__':
    dirs = ["../data/general/animals",
            "../data/general/speech",
            "../data/general/music"]
    class_names = [os.path.basename(d) for d in dirs]
    m_win, m_step, s_win, s_step = 1, 1, 0.1, 0.05
    features = []
    for d in dirs:
        # get feature matrix for each directory (class)
        f, files, fn = aF.directory_feature_extraction(d, m_win, m_step, s_win,
                                                        s_step)
        features.append(f)
    ut.plot_feature_histograms(features, fn, class_names)
```



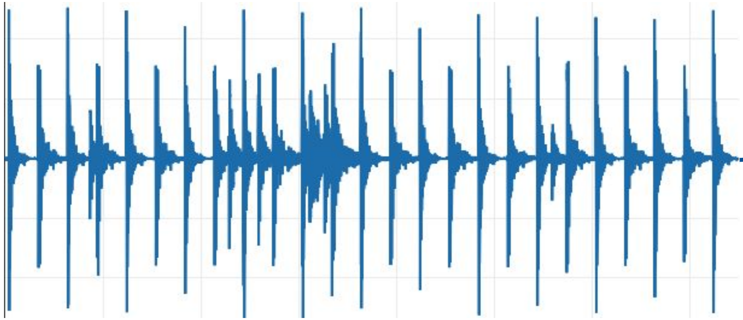
Beat tracking (1)

- Tempo / beat: fundamental properties of music
- Beat: a steady "pulse" that provides the temporal framework of a song
- Beat tracking: "tapping the foot when music plays"
- **onset** detection:
  - onset: time position of a significant signal change (e.g note)
  - change in signal's energy or frequency distribution
  - onset ⟶ attach ⟶ decay
- **tempo** estimation & beat **peaks** selection:
  - detect peaks that are (almost) equally spaced in time
  - detected peaks are (almost) consistent with est. tempo

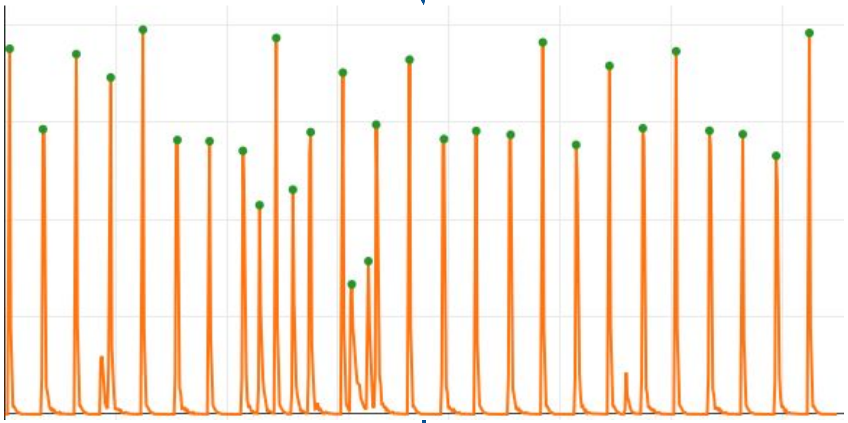




Beat tracking (2)



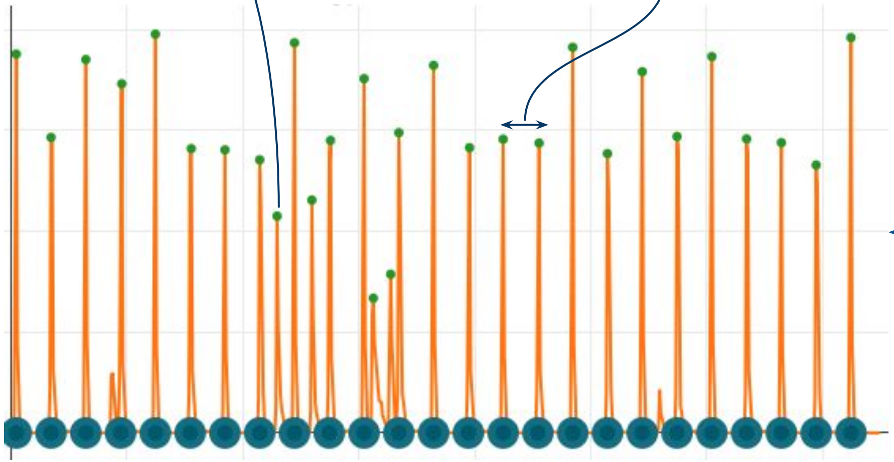
Feature Extraction & Onset Detection



Tempo estimation / beats detection

Certain peaks that are not consistent with the estimated tempo are discarded

period (seconds)
tempo = 60 / period bpms



check example14.py for this demo

Beat tracking (3)

- Perception of beat is
    - hierarchical
    - ambiguous
- Estimated tempo can be multiple of the “real” tempo
- E.g. rap song tracked at 170 bpms: true tempo is 170/2 = 85 bpms!
- External expert knowledge may be needed to put constraints in the last step of the tempo extraction method (see prev slide)

J. P. Bello, L. Daudet, S.Abdallah, C. Duxbury, M. Davies, M. B. Sandler, “A Tutorial on Onset Detection in Music Signals,” *IEEETr.Speech andAudio Proc.*,vol.13,no.5,pp.1035-1047,September 2005

P. Desain & H. Honing,“Computational models of beat induction:The rule-based approach,” *J. New Music Research*, vol. 28 no. 1, pp. 29-42, 1999.

Eric. D. Scheirer,“Tempo and beat analysis of acoustic musical signals,” *J.Acoust. Soc.Am.*, vol. 103, pp. 588-601, 1998.

Davies, M. E., & Plumbley, M. D. (2007). Context-dependent beat tracking of musical audio. *IEEE Transactions on Audio, Speech, and Language Processing*, *15*(3), 1009-1020.

McKinney, M. F., & Moelants, D. (2006). Ambiguity in tempo perception: What draws listeners to different metrical levels?. *Music Perception: An Interdisciplinary Journal*, *24*(2), 155-166.

Beat tracking - Example

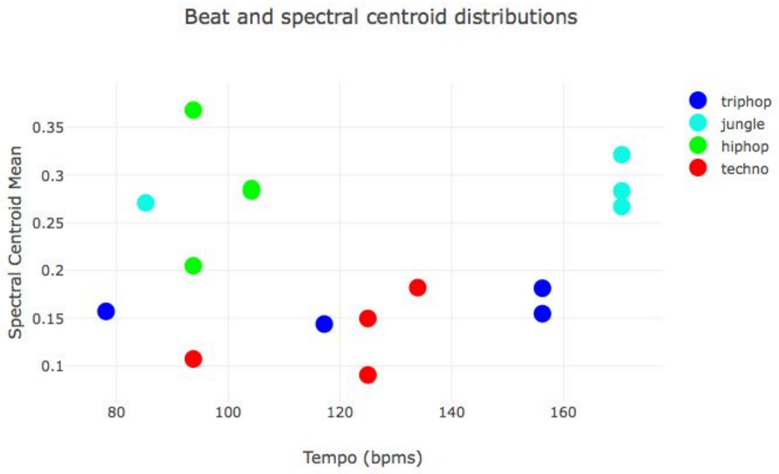
```python
"""!
@brief Example 15
@details librosa beattracking example
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import numpy as np
import scipy.io.wavfile as wavfile
import sys
import librosa

if __name__ == '__main__':
   # needs filepath as main argument:
   if len(sys.argv) != 2:
       sys.exit()
   # load file and extract tempo and beats:
   [Fs, s] = wavfile.read(sys.argv[1])
   tempo, beats = librosa.beat.beat_track(y=s, sr=Fs, units="time")
   beats -= 0.05
   # add small 220Hz sounds on the 2nd channel of the song on each beat
   s = s.reshape(-1, 1)
   s = np.array(np.concatenate((s, np.zeros(s.shape)), axis=1))
   for ib, b in enumerate(beats):
       t = np.arange(0, 0.2, 1.0 / Fs)
       amp_mod = 0.2 / (np.sqrt(t)+0.2) - 0.2
       amp_mod[amp_mod < 0] = 0
       x = s.max() * np.cos(2 * np.pi * t * 220) * amp_mod
       s[int(Fs * b):
         int(Fs * b) + int(x.shape[0]), 1] = x.astype('int16')
   wavfile.write("output.wav", Fs, np.int16(s))
```

Usage example:

```
audio|master⚡ ⇒ python3 example15.py
../data/musical_genres_small/hiphop/nwa_straght_outta_campton.wav
```

Beat tracking - Discrimination Example

```python
"""!
@brief Example 16
@details librosa beattracking example: extract tempo and spectral centroid for
songs from different musical genres
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import scipy.io.wavfile as wavfile, utilities as ut
import glob, os, librosa, plotly, numpy as np, plotly.graph_objs as go
from pyAudioAnalysis.MidTermFeatures import mid_feature_extraction as mt

layout = go.Layout(title='Beat and spectral centroid distributions',
                   xaxis=dict(title='Tempo (bpms)',),
                   yaxis=dict(title='Spectral Centroid Mean',))

def get_dir_features(dir_name):
   feats = []
   for f in glob.glob(os.path.join(dir_name, "*.wav")):
       [Fs, s] = wavfile.read(f)
       tempo, _ = librosa.beat.beat_track(y=s, sr=Fs)
       f, _, fn = mt(s, Fs, int(1.0*Fs), int(1.0*Fs), int(0.1*Fs), int(0.1*Fs))
       feats.append([tempo, np.mean(f[fn.index("spectral_centroid_mean")],
                                    axis=0)])
   return np.array(feats)

if __name__ == '__main__':
   g_paths = glob.glob("../data/musical_genres_small/*/")
   g_names = [p.split('/')[-2] for p in g_paths]
   clr = ut.get_color_combinations(len(g_paths))
   features = [get_dir_features(g) for g in g_paths]
   plots = [go.Scatter(x=features[i][:, 0], y=features[i][:, 1],
                       mode='markers', name=g_names[i], marker=dict(
                       color=clr[i], size=15))
            for i in range(len(g_paths))]
   plotly.offline.plot(go.Figure(data = plots, layout=layout),
                       filename="temp.html", auto_open=True)
```



"True" tempo values:

- Triphop: 90-110
- Jungle: 160-170
- Hiphop: 80-100
- Techno: 120-130

- Tempo is not always discriminative
- Tempo estimation has errors (e.g. triphop estimated values are double)

Need for multiple features to achieve accurate discrimination

Pitch tracking

- **f0**:
  - fundamental frequency
  - a **physical** property of sound:
    - speech: glottal pulses freq
    - music: most dominant freq of a note (eg freq of vibration of a string)
- **pitch**
  - a **subjective** phenomenon (f0 open to measurement)
  - perceptual
  - follows f0
  - speech:
    - not always clear
    - vad required
  - music:
    - note transcription
    - polyphony

Pitch tracking:

- Time / spectral domain
- Spectral:
  - simple argmax?
  - no! f0 not always the freq with the max freq in spectrogram

Pitch Tracking - Example using librosa

```python
"""!
@brief Example 17
@details librosa pitch tracking example
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import scipy.io.wavfile as wavfile
import librosa
import plotly
import numpy as np
import plotly.graph_objs as go
from scipy.signal import medfilt as mf

layout = go.Layout(title='Librosa pitch estimation',
                   xaxis=dict(title='time frame',),
                   yaxis=dict(title='freq (Hz)',))

def get_librosa_pitch(signal, fs, window):
   pitches, magnitudes = librosa.piptrack(y=signal, sr=fs, n_fft=int(window),
                                          hop_length=int(window/10))
   pitch_pos = np.argmax(magnitudes, axis=0)
   pitches_final = []
   for i in range(len(pitch_pos)):
       pitches_final.append(pitches[pitch_pos[i], i])
   pitches_final = np.array(pitches_final)
   pitches_final[pitches_final > 2000] = 0  # cut high pitches
   return mf(pitches_final, 3)              # use medfilt for smoothing

if __name__ == '__main__':
   [fs, s] = wavfile.read("../data/acapella.wav")
   p = get_librosa_pitch(s, fs, fs/20)
   plt = go.Scatter(x=np.arange(len(p)), y=p, mode='markers', showlegend=False)
   plotly.offline.plot(go.Figure(data=[plt], layout=layout),
                       filename="temp.html", auto_open=True)
```