MSc in AI
NCSR Demokritos - University of Piraeus

Course: **Machine Learning for Multimodal Data**

Lesson 4
Audio Segmentation and Audio Fingerprinting

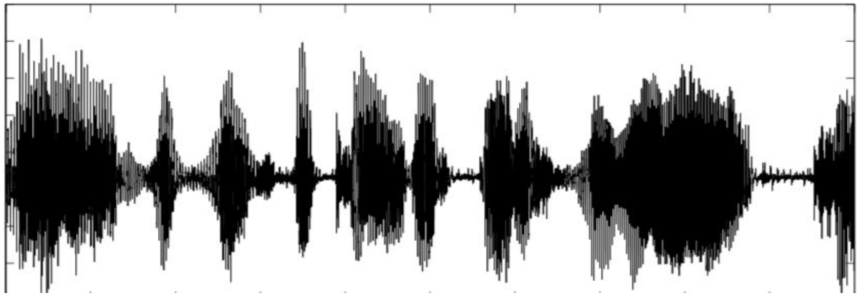
Theodoros Giannakopoulos

Previous

- **Represent** audio signals by features and segment statistics
- **Classify** each **segment** using the respective feature vector (statistics on short-term features)
- In the case of long recordings of homogeneous content (songs) ⟶ long term averaging of the features can be applied
- **But**
  - Content changes with time
  - Small segments (e.g. 2 or 3 seconds) can have a unique class label
  - Not the same segment size
  - Supervised knowledge is not always available
  - **Need for unsupervised / semisupervised methods to split the signal in terms of content**

Segmentation

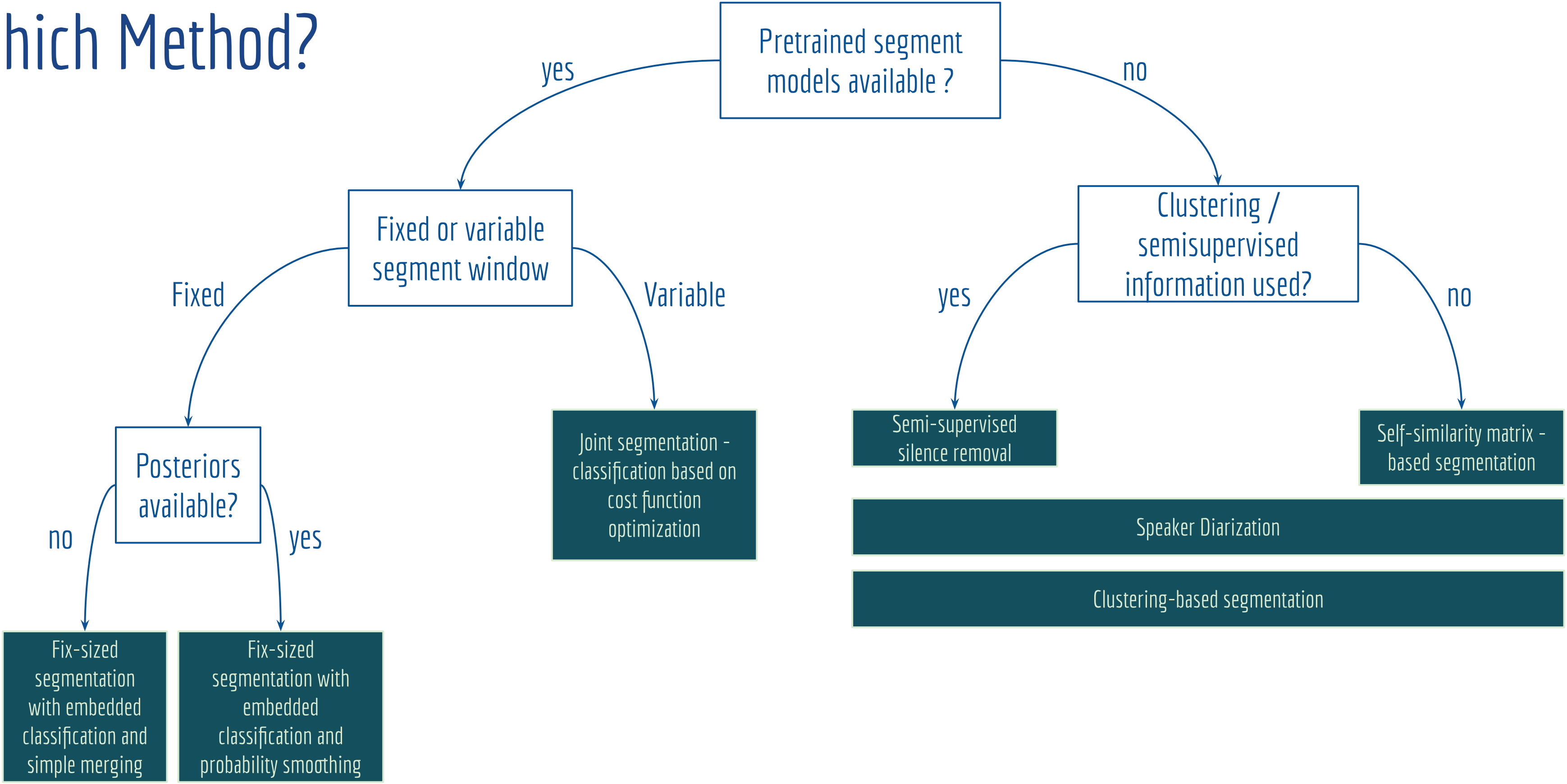
- Usually an unsupervised (or semi supervised) task
- Input:
  - long audio recording
  - (opt) prior information
- Output:
  - shorter segments of homogeneous content
- Homogeneity can be wrt:
  - Audio event class
  - Silence Vs Activity
  - Speaker identity
  - ...
- May or may not use supervised information (pretrained classifiers of audio segments)



Contents

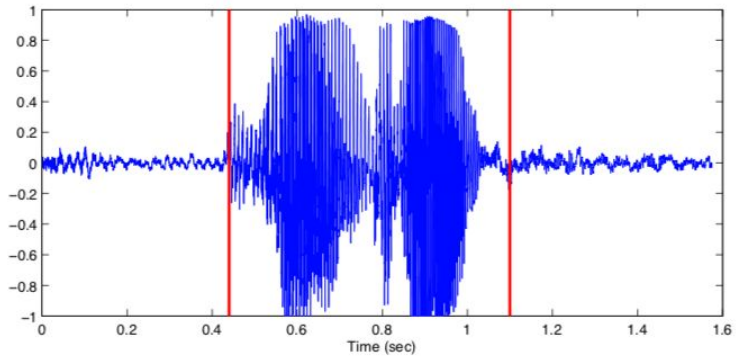
- Silence removal
- Speaker diarization
- Music segmentation
- Audio thumbnailing
- Model-based audio segmentation
- Joint segmentation - classification

Which Method?



Silence Removal / VAD (Voice Activity Detection

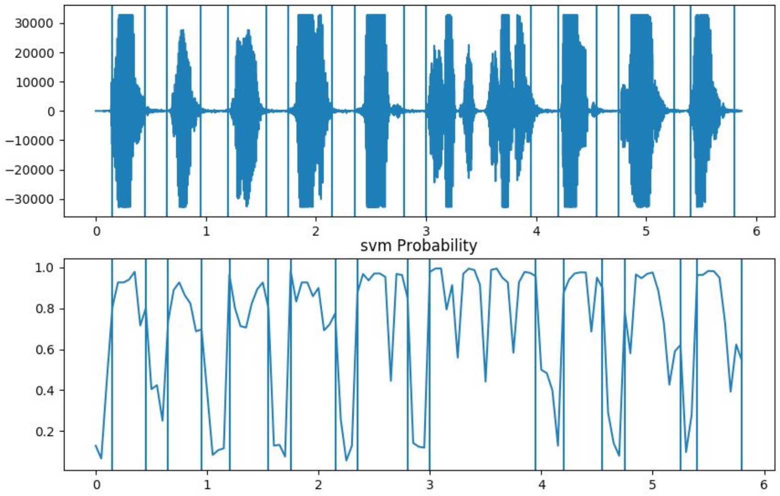
- Semi-supervised
- Implemented in pyAudioAnalysis
- Audio recording ⟶ segment endpoints of audio events (silence removed)
- Method:
    - Extract short-term features
    - Train SVM:
        - Binary
        - Use 10% of the highest-energy frames as positive (activity) and
        - 10% lowest as negative (silence)
    - Apply trained SVM with probabilistic output ⟶ sequence of probabilities
    - Apply dynamic threshold on probabilistic sequence
    - Extract segments



Silence Removal - example27.py

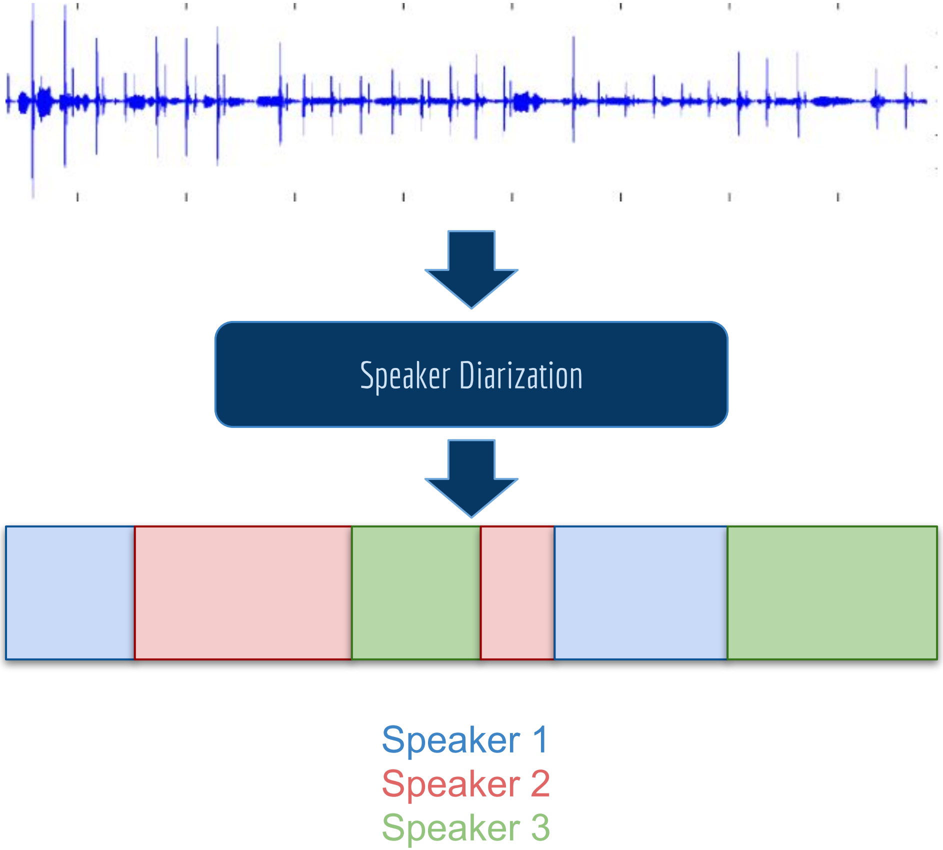
```
"""!
@brief Example 27
@details: Silence removal example (implemented in pyAudioAnalysis)
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import os, readchar
from pyAudioAnalysis.audioSegmentation import silence_removal as sR
from pyAudioAnalysis.audioBasicIO import read_audio_file

if __name__ == '__main__':
    # get non-silent segment limits:
    input_file = "../data/count.wav"
    fs, x = read_audio_file(input_file)
    seg_lims = sR(x, fs, 0.05, 0.05, 0.05, 0.5, True)
    # play each segment:
    for i_s, s in enumerate(seg_lims):
        print("Playing segment {0:d} of {1:d} "
              "({2:.2f} - {3:.2f} secs)".format(i_s, len(seg_lims), s[0], s[1]))
        # save the current segment to temp.wav
        os.system("avconv -i {} -ss {} -t {} temp.wav "
                  "-loglevel panic -y".format(input_file, s[0], s[1]-s[0]))
        # play segment and wait for input
        os.system("play temp.wav")
        readchar.readchar()
```

(also able to listen to individual non-silent segments)



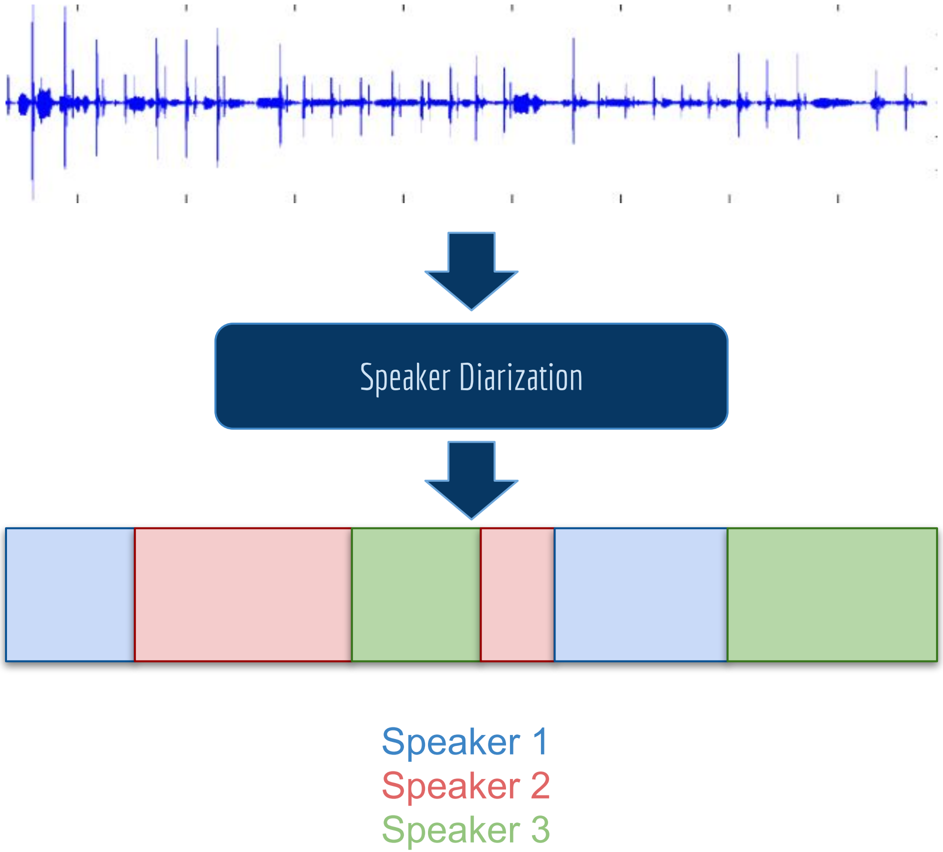
Speaker Diarization

- Input:
  - Audio signal recording
  - (optional) number of speakers
- Output:
  - Segment limits and respective speaker IDs
  - (if not provided) estimated number of speakers
- Answers the question: **“who spoke when”**
- Unsupervised or semi-supervised task
- Useful in:
  - Audio summarization
  - Emotion recognition
  - Speech Analytics
  - Automatic Speech Recognition (ASR)



Speaker Diarization - Challenges & Refs

- Number of speakers not always known
- Overlapping speakers
- High variability in noise / music / events
- High variability in speaker turn durations



[1] Tranter, Sue E., and Douglas A. Reynolds. "An overview of automatic speaker diarization systems." *IEEE Transactions on audio, speech, and language processing* 14.5 (2006): 1557-1565.

[2] Anguera, Xavier, Chuck Wooters, and Javier Hernando. "Acoustic beamforming for speaker diarization of meetings." *IEEE Transactions on Audio, Speech, and Language Processing*15.7 (2007): 2011-2022.

[3] Giannakopoulos, Theodoros, and Sergios Petridis. "Fisher linear semi-discriminant analysis for speaker diarization." *IEEE Transactions on Audio, Speech, and Language Processing* 20.7 (2012): 1913-1922.

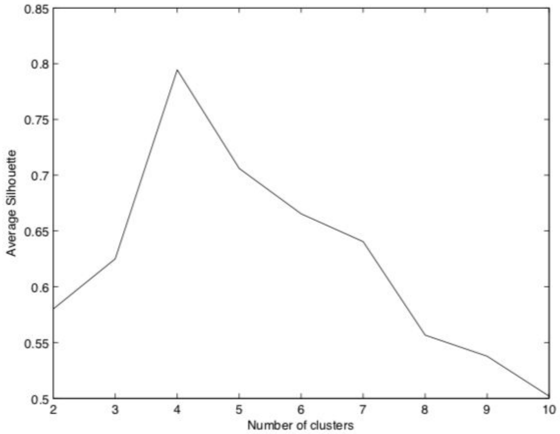
Speaker Diarization - example 28

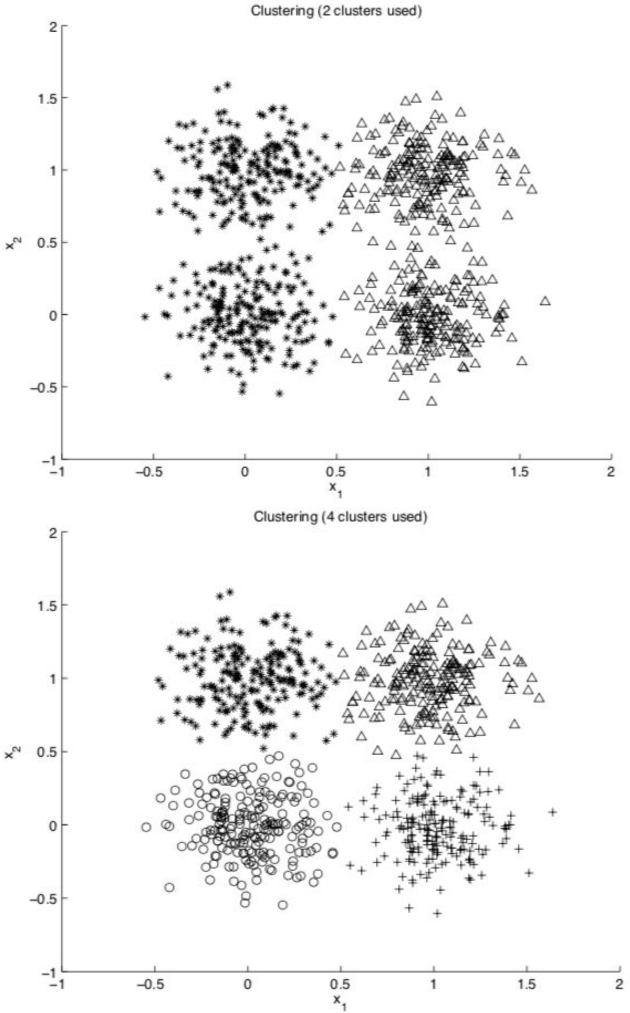
- Simplest approach:
    - Extract segment-level audio feature statistics
    - Huge segment overlap (to extract as many feature vectors as possible)
    - Clustering
    - Temporal smoothing (optional)
- Known number of speakers (clusters)
- Example 28:
    - 4 speakers (known)
    - Listen to all segments of each speakers

```python
"""!
@brief Example 28
@details: Speaker diarization example
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import os, readchar, sklearn.cluster
from pyAudioAnalysis.MidTermFeatures import mid_feature_extraction as mT
from pyAudioAnalysis.audioBasicIO import read_audio_file, stereo_to_mono
from pyAudioAnalysis.audioSegmentation import labels_to_segments
from pyAudioAnalysis.audioTrainTest import normalize_features

if __name__ == '__main__':
   # read signal and get normalized segment features:
   input_file = "../data/diarizationExample.wav"
   fs, x = read_audio_file(input_file)
   x = stereo_to_mono(x)
   mt_size, mt_step, st_win = 1, 0.1, 0.05
   [mt_feats, st_feats, _] = mT(x, fs, mt_size * fs, mt_step * fs,
                                round(fs * st_win), round(fs * st_win * 0.5))
   (mt_feats_norm, MEAN, STD) = normalize_features([mt_feats.T])
   mt_feats_norm = mt_feats_norm[0].T
   # perform clustering (k = 4)
   n_clusters = 4
   k_means = sklearn.cluster.KMeans(n_clusters=n_clusters)
   k_means.fit(mt_feats_norm.T)
   cls = k_means.labels_
   segs, c = labels_to_segments(cls, mt_step) # convert flags to segment limits
   for sp in range(n_clusters):            # play each cluster's segment
       for i in range(len(c)):
           if c[i] == sp and segs[i, 1]-segs[i, 0] > 1:
               # play long segments of current speaker
               print(c[i], segs[i, 0], segs[i, 1])
               cmd = "avconv -i {} -ss {} -t {} temp.wav " \
                     "-loglevel panic -y".format(input_file, segs[i, 0]+1,
                                                 segs[i, 1]-segs[i, 0]-1)
               os.system(cmd)
               os.system("play temp.wav -q")
               readchar.readchar()
```

Speaker Diarization - Estimate #speakers

- What if #speakers is unknown?
- Estimate using cluster - related metrics:
  - Gap statistic
  - Silhouette
- Silhouette: measures how "tightly" the data is grouped in each cluster (using a distance metric)





[1] Tibshirani, Robert, Guenther Walther, and Trevor Hastie. "Estimating the number of clusters in a data set via the gap statistic." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63.2 (2001): 411-423.
[2] Celeux, Gilles, and Gilda Soromenho. "An entropy criterion for assessing the number of clusters in a mixture model." *Journal of classification* 13.2 (1996): 195-212.
[3] Kodinariya, Trupti M., and Prashant R. Makwana. "Review on determining number of Cluster in K-Means Clustering." *International Journal* 1.6 (2013): 90-95.

Music Segmentation

- Automated structural analysis of music
- Efficient content-based music retrieval
- Indexing
- Audio summary
- Unsupervised: clustering will discover similar structural parts (not their labels)
- Need of supervised information or apriori knowledge to extract musical part labels

[1] Jensen, Kristoffer. "Multiple scale music segmentation using rhythm, timbre, and harmony." *EURASIP Journal on Applied Signal Processing* 2007.1 (2007): 159-159.
[2] Levy, Mark, Katy Noland, and Mark Sandler. "A comparison of timbral and harmonic music segmentation algorithms." *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*. Vol. 4. IEEE, 2007.

Music Segmentation

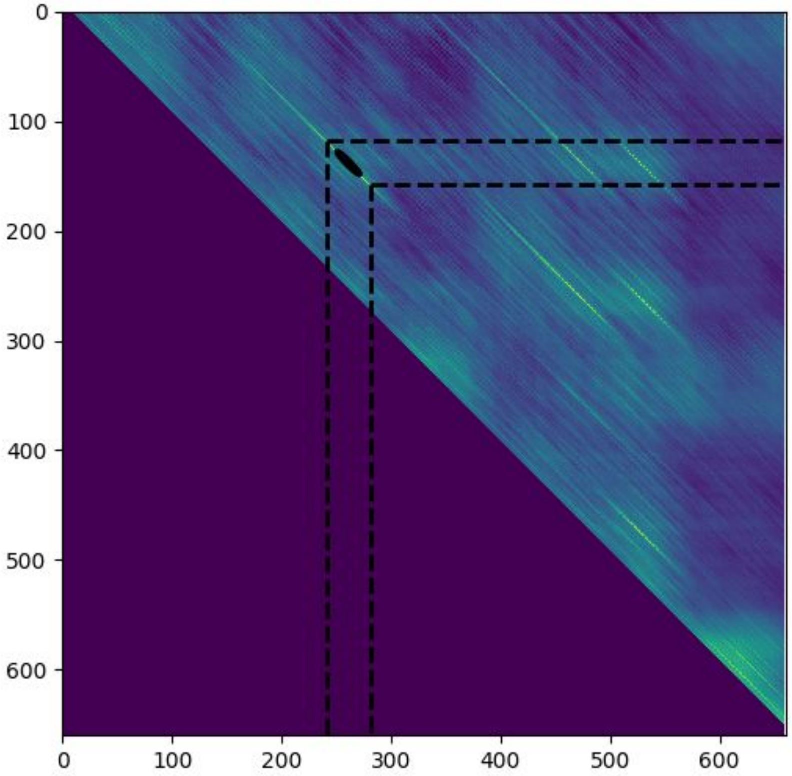
```
import os, readchar, sklearn.cluster
from pyAudioAnalysis.MidTermFeatures import mid_feature_extraction as mT
from pyAudioAnalysis.audioBasicIO import read_audio_file, stereo_to_mono
from pyAudioAnalysis.audioSegmentation import labels_to_segments
from pyAudioAnalysis.audioTrainTest import normalize_features

if __name__ == '__main__':
    # read signal and get normalized segment features:
    input_file = "../data/song1.mp3"
    fs, x = read_audio_file(input_file)
    x = stereo_to_mono(x)
    mt_size, mt_step, st_win = 5, 0.5, 0.05
    [mt_feats, st_feats, _] = mT(x, fs, mt_size * fs, mt_step * fs,
                                 round(fs * st_win), round(fs * st_win * 0.5))
    (mt_feats_norm, MEAN, STD) = normalize_features([mt_feats.T])
    mt_feats_norm = mt_feats_norm[0].T
    # perform clustering (k = 4)
    n_clusters = 4
    k_means = sklearn.cluster.KMeans(n_clusters=n_clusters)
    k_means.fit(mt_feats_norm.T)
    cls = k_means.labels_
    segs, c = labels_to_segments(cls, mt_step) # convert flags to segment limits
    for sp in range(n_clusters):            # play each cluster's segment
        for i in range(len(c)):
            if c[i] == sp and segs[i, 1]-segs[i, 0] > 5:
                # play long segments of current cluster (only win_to_play seconds)
                d = segs[i, 1]-segs[i, 0]
                win_to_play = 10
                if win_to_play > d:
                    win_to_play = d
                print(" * * * * CLUSTER {0:d} * * * * * {1:.1f} - {2:.1f}, "
                      "playing {3:.1f}-{4:.1f}".format(c[i], segs[i, 0],
                                                       segs[i, 1],
                                                       segs[i, 0] + d/2 - win_to_play/2,
                                                       segs[i, 0] + d/2 +
win_to_play/2))
                cmd = "avconv -i {} -ss {} -t {} temp.wav " \
                        "-loglevel panic -y".format(input_file,
                                                   segs[i, 0] + d/2 - win_to_play/2,
                                                   win_to_play)
                os.system(cmd)
                os.system("play temp.wav -q")
                readchar.readchar()
```

- Detected clusters may correlate to structural music parts
- Script plays each cluster's segments (only win_to_play seconds from the middle of each segment)

Music Segmentation - Audio Thumbnailing

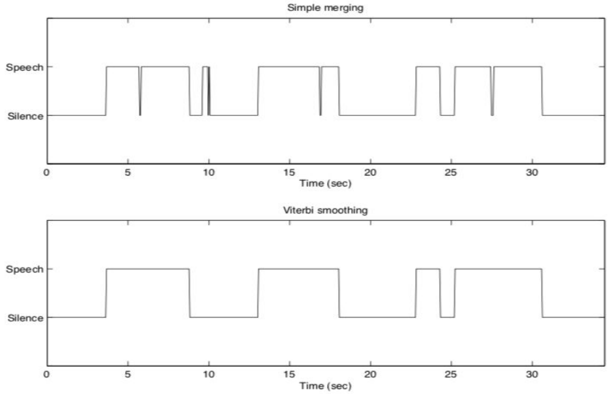
- Goal: detect the most representative part from a song
- Repeated
- Long enough
- In popular music: thumbnails → chorus
- Easiest part to memorize (commercial impact)
- Self-similarity matrix

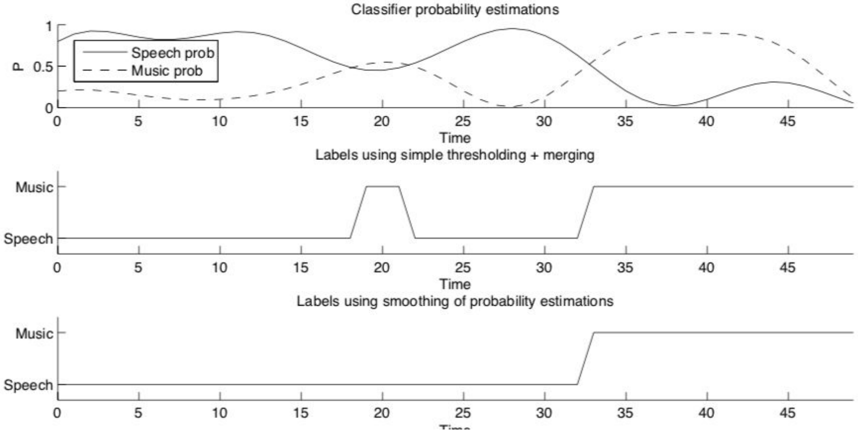
Music Segmentation - Audio Thumbnailing - example 30



Model-based segmentation

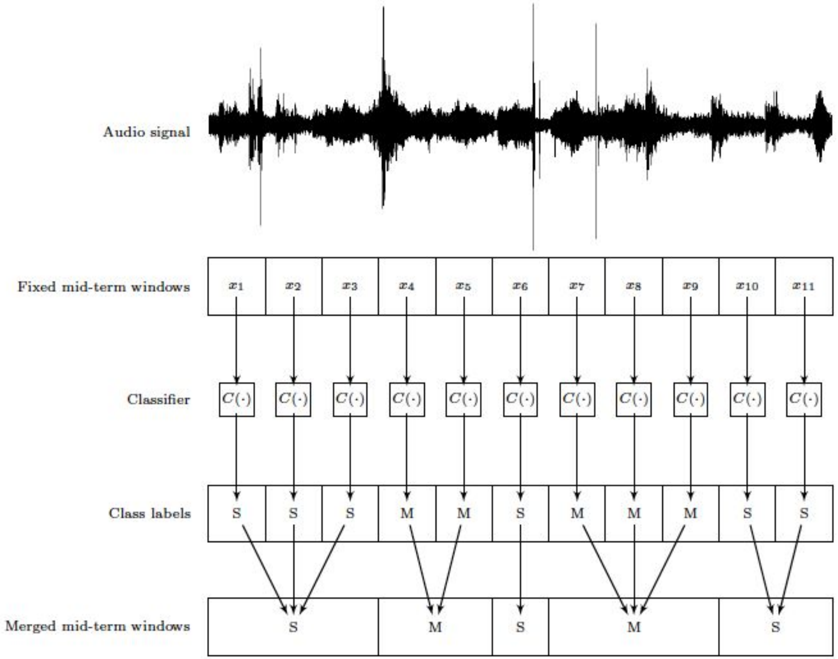
- Segmentation with embedded classification
- Train statistical models (classifiers) to each of the acoustic classes
- Divide the signal in segments:
  - Possibly overlapping
  - 1-5 second long depending on the application
- Use fix-sized segments as input to the pretrained models
- Post process:
  - Naive merging: merge successive segments that share the same class label **[next example]**
  - Probability smoothing (for soft-output classifiers): apply a Viterbi-based smoothing technique on the prosteriors
- Supervised knowledge not always available (or is partly!)





Supervised Segmentation

- Use a pre-trained model (e.g. an SVM trained to distinguish between speech and music audio segments)
- Segmentation input: a long audio stream that may contain both speech and music (non-overlapping)
- Classify each fix-sized segment of the audio stream using the trained model
- Merge successive segments that contain the same class label



Supervised Segmentation: Step A: Train Segment Classifier

```python
"""!
@brief Example 31A
@details: Train a speech-music classifier
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
from pyAudioAnalysis.audioTrainTest import extract_features_and_train

if __name__ == '__main__':
    mt = 1.0
    st = 0.05
    dir_paths = ["../data/speech_music/speech", "../data/speech_music/music"]
    extract_features_and_train(dir_paths, mt, mt, st, st,
                               "svm_rbf", "svm_speech_music")

```

`featureAndTrain()` results in an optimal (in terms of C) SVM model

The model is saved in `svm_speech_music` binary file

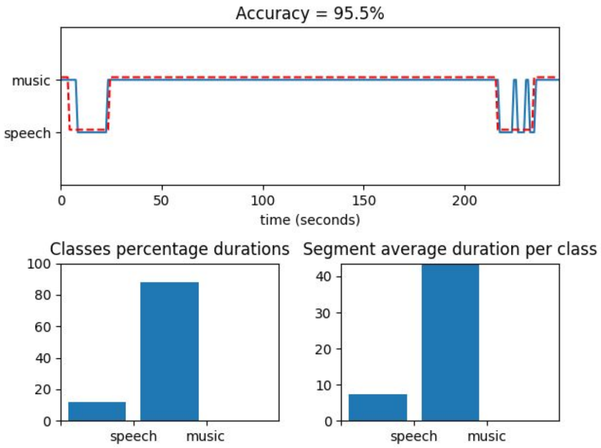
Also, file `svm_speech_musicMEANS` stores the MEAN/STD (normalization)

```
                  speech                     music          OVERALL
        C       PRE     REC     f1      PRE     REC     f1      ACC     f1
        0.001   93.5    84.8    89.0    86.1    94.1    89.9    89.5    89.5
        0.010   93.0    85.6    89.2    86.7    93.6    90.0    89.6    89.6
        0.500   95.3    91.9    93.6    92.2    95.5    93.8    93.7    93.7
        1.000   95.0    93.6    94.3    93.7    95.0    94.4    94.3    94.3    best f1         best Acc
        5.000   93.9    93.5    93.7    93.6    93.9    93.8    93.7    93.7
        10.000  93.2    95.0    94.1    95.0    93.1    94.0    94.1    94.1
        20.000  93.6    94.3    94.0    94.3    93.6    94.0    94.0    94.0
Confusion Matrix:
        spe     mus
spe     46.80   3.20
mus     2.48    47.52
Selected params: 1.00000
```

Supervised Segmentation: Step B: Apply Segment Classifier

```python
"""!
@brief Example 31B
@details: Speech music discrimination and segmentation (using a trained
speech - music segment classifier)
Important: Need to run 31A first to extract speech music model (stored
in svm_speech_music)
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
from pyAudioAnalysis.audioSegmentation import mid_term_file_classification

if __name__ == '__main__':
    au = "../data/scottish_radio.wav"
    gt = "../data/scottish_radio.segments"
#     au = "../data/musical_genres_small/hiphop/run_dmc_peter_riper.wav"
    mid_term_file_classification(au, "svm_speech_music", "svm_rbf", True, gt)
```

`mid_term_file_classification()` uses speech SVM model (`svm_speech_music`) and `svm_speech_musicMEANS` (feature normalization params) to classify and segment the input audio recording using fix-sized segments



If ground-truth file is provided → accuracy is also computed. (gt is plotted using red color)

GT file should have the format:

```
<seg_start_1>, <seg_end_1>, <class_label_1>
<seg_start_2>, <seg_end_2>, <class_label_2>
...
<seg_start_N>, <seg_end_N>, <class_label_N>
```

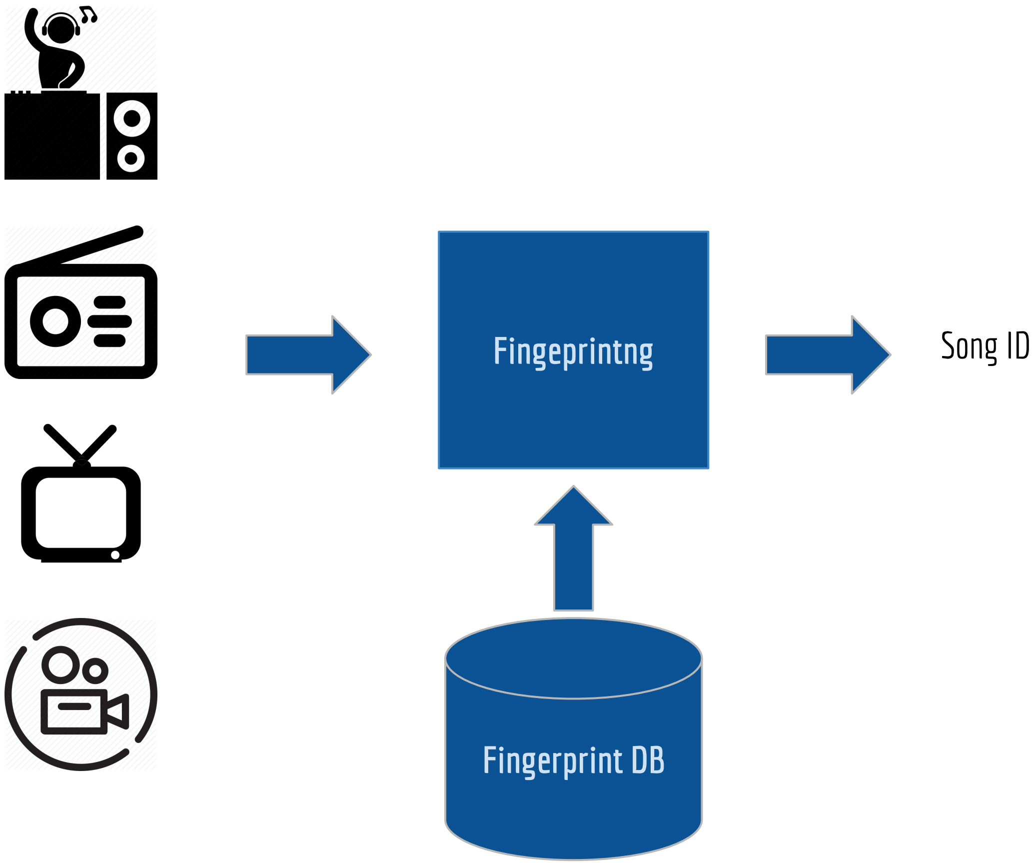
Joint Segmentation Classification (speech vs music)

- Audio **segmentation** treated as a **maximization task**
- Solution is obtained by means of **dynamic programming**
- ***find the sequence of segments and respective class labels that maximize the product of posterior class probabilities, given the audio features of the audio segments***

[1] Giannakopoulos, Theodoros, and Aggelos Pikrakis. *Introduction to audio analysis: a MATLAB® approach*. Academic Press, 2014.

[2] Kim, Hyoung-Gook, Nicolas Moreau, and Thomas Sikora. *MPEG-7 audio and beyond: Audio content indexing and retrieval.* John Wiley & Sons, 2006.

Fingerprinting



dejavu - Dependencies

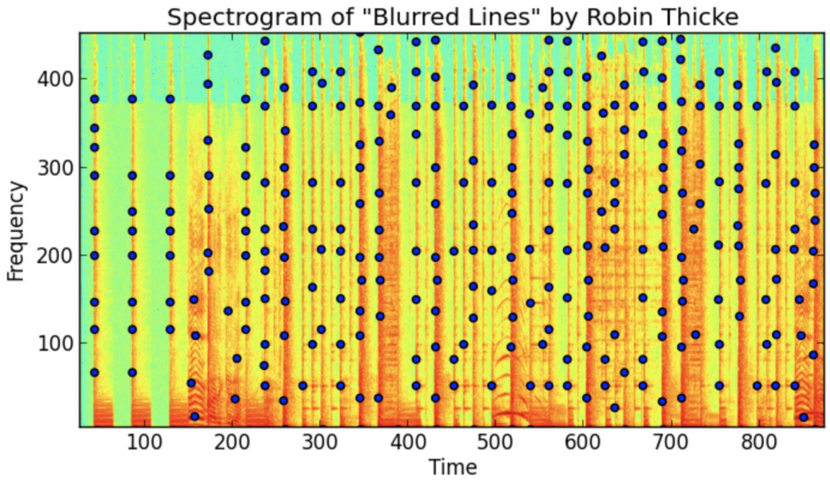
brew install (or apt-get install for ubuntu):

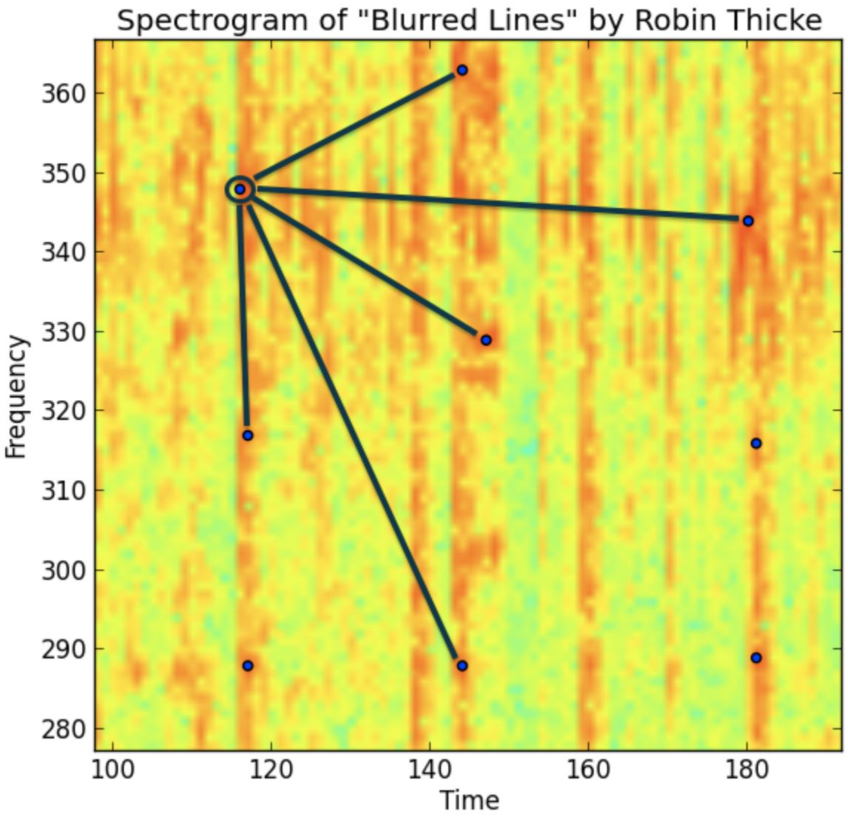
- Portaudio
- ffmpeg

Pip

- pydub
- numpy
- scipy
- matplotlib
- MySQL-python (requires mysql installed)

Fingerprinting - Core Method





[1] https://www.ee.columbia.edu/~dpwe/e4896/lectures/E4896-L13.pdf
[2] http://hpac.rwth-aachen.de/teaching/sem-mus-17/Final-slides/Froitzheim.pdf
[3] http://homepage.tudelft.nl/c7c8y/theses/PhDThesisDoets.pdf

dejavu - Handle DB

Start mysql:

⇒ mysqld

Create Database:

⇒ `mysql -u root`

⇒ `CREATE DATABASE IF NOT EXISTS dejavu;`

If database already exists and one needs to remove:

⇒ `mysql -u root`

⇒ `DROP DATABASE dejavu;`

dejavu - Add songs to database

```python
"""!
@brief Fingerprinting training using dejavu
@details: Audio fingerprinting training using dejavu
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""

import sys
sys.path.append('../')
import dejavu

config = {
   "database": {
       "host": "127.0.0.1",
       "user": "root",
       "passwd": "",
       "db": "dejavu",
   }
}

if __name__ == '__main__':
   djv = dejavu.Dejavu(config)
   n_workers = 3
   djv.fingerprint_directory("songs", [".wav"], n_workers)
```

Convert mp3 to 8K wav

⇒ python convertToWav.py mp3 16000 1

⇒ mkdir songs

⇒ mv mp3/*.wav songs

Add songs to database

⇒ python fingerprinting_add_dir.py

dejavu - Get database statistics

```python
"""!
@brief Fingeprinting: show database stats
@details: Audio fingerprinting using dejavu
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""

import MySQLdb as mysqldb

if __name__ == '__main__':
    db1 = mysqldb.connect("127.0.0.1", "root", "", 'dejavu')
    cursor = db1.cursor()
    a = cursor.execute("SELECT * FROM songs;")
    song_set = cursor.fetchall()
    for ir, r in enumerate(song_set):
        print "song {}\t {}".format(ir, r[1])
    n_fings = cursor.execute("SELECT * FROM fingerprints;")
    n_figs_per_songs = n_fings / float(len(song_set))
    print " - - - - - - - - - - - - - - - - -"
    print "NUMBER OF SONGS IN DATABASE: {}".format(len(song_set))
    print "NUMBER OF FINGERPRINTS IN DATABASE: {}".format(n_fings)
    print "NUMBER OF FINGERPRINTS PER SONG: {0:.1f}".format(n_figs_per_songs)
    print " - - - - - - - - - - - - - - - - -"
```

```
⇒ python fingerprinting_show_stats.py


…
…
…
song 93       05 Pearl Jam - Alive
song 94       80 Veruca Salt - Seether
song 95       37 Red Hot Chili Peppers - Give It
Away
song 96       65 The Wallflowers - One Headlight
song 97       83 Radiohead - Karma Police
song 98       86 Tool - 46 & 2
 - - - - - - - - - - - - - - - -
NUMBER OF SONGS IN DATABASE: 99
NUMBER OF FINGERPRINTS IN DATABASE: 2283127
NUMBER OF FINGERPRINTS PER SONG: 23061.9
 - - - - - - - - - - - - - - - -
```

dejavu - Search Song

```python
"""!
@brief fingerprinting search song
@details: Audio fingerprinting using dejavu
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""

import sys
sys.path.append('../')
import dejavu
from dejavu.recognize import FileRecognizer

config = {
   "database": {
        "host": "127.0.0.1",
        "user": "root",
        "passwd": "",
        "db": "dejavu",
   }
}

if __name__ == '__main__':
   djv = dejavu.Dejavu(config)
   print djv.recognize(FileRecognizer, "query.wav")
```

Download youtube song and search in DB

```
⇒ youtube-dl https://www.youtube.com/watch?v=YgSPaXgAdzE
⇒ avconv -i Beck\ -\ Loser-YgSPaXgAdzE.mp4 -ar 16000 -ac 1
query.wav
⇒ python fingerprinting_search_song.py
{'song_id': 75, 'song_name': '28 Beck - Loser', 'file_sha1':
'0E8F882C8203A56BCE8575CE00DA8ABC3D5A6F16', 'confidence': 1093,
'offset_seconds': -0.13932, 'match_time': 2.4883780479431152,
'offset': -3}
```

Using 10 seconds of data

```
⇒ avconv -i Beck\ -\ Loser-YgSPaXgAdzE.mp4 -ar 16000 -ac 1 -ss
100 -t 10 query.wav
⇒ python fingerprinting_search_song.py
{'song_id': 75, 'song_name': '28 Beck - Loser', 'file_sha1':
'0E8F882C8203A56BCE8575CE00DA8ABC3D5A6F16', 'confidence': 98,
'offset_seconds': 17.97224, 'match_time': 0.15426397323608398,
'offset': 387}
```

Using 1 sec of data

```
⇒ avconv -i Beck\ -\ Loser-YgSPaXgAdzE.mp4 -ar 16000 -ac 1 -ss
100 -t 10 query.wav
⇒ python fingerprinting_search_song.py
{'song_id': 75, 'song_name': '28 Beck - Loser', 'file_sha1':
'0E8F882C8203A56BCE8575CE00DA8ABC3D5A6F16', 'confidence': 3,
'offset_seconds': 17.97224, 'match_time': 0.018265962600708008,
'offset': 387}
```

dejavu - Search Song

```python
"""!
@brief fingerprinting search song
@details: Audio fingerprinting using dejavu
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""

import sys
sys.path.append('../')
import dejavu
from dejavu.recognize import FileRecognizer

config = {
    "database": {
        "host": "127.0.0.1",
        "user": "root",
        "passwd": "",
        "db": "dejavu",
    }
}

if __name__ == '__main__':
    djv = dejavu.Dejavu(config)
    print djv.recognize(FileRecognizer, "query.wav")
```

Download youtube song and search in DB (Loser Live)

⇒ youtube-dl https://www.youtube.com/watch\?v\=-l3_gwIOTGI

⇒ avconv -i Beck\ -\ Loser\ \(Live\ 2003\)--l3_gwIOTGI.mkv -ar 8000 -ac 1 -ss 50 -t 100 query.wav

⇒ python fingerprinting_search_song.py

{'song_id': 56, 'song_name': '48 Gin Blossoms - Hey Jealousy', 'file_sha1': '9336982AD0DBE906C9D8CDC8BF45BE86EC9C5EB3', 'confidence': 3, 'offset_seconds': 26.37787, 'match_time': 1.2293858528137207, 'offset': 568}

(wrong)

dejavu - Search Song (online from mic)

```python
"""!
@brief fingerprinting search song
@details: Audio fingerprinting using dejavu
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""

import sys
sys.path.append('../')
import dejavu
from dejavu.recognize import FileRecognizer

config = {
   "database": {
       "host": "127.0.0.1",
       "user": "root",
       "passwd": "",
       "db": "dejavu",
   }
}

if __name__ == '__main__':
   djv = dejavu.Dejavu(config)
   print djv.recognize(FileRecognizer, "query.wav")
```