# **Yaafe - audio features extraction**

*Yaafe* is an audio features extraction toolbox.

**Easy to use**

The user can easily declare the features to extract and their parameters in a text file. Features can be extracted in a batch mode, writing CSV or H5 files. The user can also extract features with Python or matlab.

**Efficient**

*Yaafe* automatically identifies common intermediate representations (spectrum, envelope, autocorrelation, ...) and computes them only once. Extraction is processed block per block so that arbitrarily long files can be processed, and memory occupation is low.

If you wonder about the *Yaafe* acronym, it’s just *Yet Another Audio Feature Extractor*.

### Frames

*class* yaafefeatures.Frames

Segment input signal into frames.

First frame has zeros on left half so that it is centered on time 0s, then consecutive frames are equally spaced. Consequently, frame *i* (starting from 0) is centered on sample *i* \* *stepSize*.

**Parameters**:

* blockSize (default=1024): output frames size
* stepSize (default=512): step between consecutive frames

**Declaration example**:

Frames blockSize=1024 stepSize=512

### LPC

*class* yaafefeatures.LPC

Compute the Linear Predictor Coefficients (LPC) of a signal frame. It uses autocorrelation and Levinson-Durbin algorithm.

|  |  |
| --- | --- |
|  |  |

**Parameters**:

* LPCNbCoeffs (default=2): Number of Linear Predictor Coefficients to compute
* blockSize (default=1024): output frames size
* stepSize (default=512): step between consecutive frames

**Declaration example**:

LPC LPCNbCoeffs=2 blockSize=1024 stepSize=512

LSF

*class* yaafefeatures.LSF

Compute the Line Spectral Frequency (LSF) coefficients of a signal frame.

|  |  |
| --- | --- |
|  |  |

**Parameters**:

* blockSize (default=1024): output frames size
* stepSize (default=512): step between consecutive frames

**Declaration example**:

LSF blockSize=1024 stepSize=512

### Loudness

*class* yaafefeatures.Loudness

The loudness coefficients are the energy in each Bark band, normalized by the overall sum.

|  |  |
| --- | --- |
|  |  |

**Parameters**:

* FFTLength (default=0): Frame’s length on which perform FFT. Original frame is padded with zeros or truncated to reach this size. If 0 then use original frame length.
* FFTWindow (default=Hanning): Weighting window to apply before fft. Hanning|Hamming|None
* LMode (default=Relative): “Specific” computes loudness without normalization, “Relative” normalize each band so that they sum to 1, “Total” just returns the sum of Loudness in all bands.
* blockSize (default=1024): output frames size
* stepSize (default=512): step between consecutive frames

**Declaration example**:

Loudness FFTLength=0 FFTWindow=Hanning LMode=Relative blockSize=1024 stepSize=512

### MFCC

*class* yaafefeatures.MFCC

Compute the Mel-frequencies cepstrum coefficients.

Mel filter bank is built as 40 log-spaced filters according to the following mel-scale:

melfreq = 1127 * log(1 + \frac{freq}{700})

Each filter is a triangular filter with height 2/(f_{max}-f_{min}). Then MFCCs are computed as following, using DCT II:

mfcc = dct(log(abs(fft(hanning(N).x)).MelFilterBank))

|  |  |
| --- | --- |
|  |  |

**Parameters**:

* CepsIgnoreFirstCoeff (default=1): 0 keeps the first cepstral coeffcient, 1 ignore it
* CepsNbCoeffs (default=13): Number of cepstral coefficient to keep.
* FFTWindow (default=Hanning): Weighting window to apply before fft. Hanning|Hamming|None
* MelMaxFreq (default=6854.0): Maximum frequency of the mel filter bank
* MelMinFreq (default=130.0): Minimum frequency of the mel filter bank
* MelNbFilters (default=40): Number of mel filters
* blockSize (default=1024): output frames size
* stepSize (default=512): step between consecutive frames

**Declaration example**:

MFCC CepsIgnoreFirstCoeff=1 CepsNbCoeffs=13 FFTWindow=Hanning MelMaxFreq=6854.0 MelMinFreq=130.0 MelNbFilters=40 blockSize=1024 stepSize=512

## Python interaction

*Yaafe* python bindings allow to easily extract features from Python with a great flexibility. The first step is always to build the DataFlow object corresponding to the audio features to extract (for example using a FeaturePlan object), and configure an Engine.

>>> from yaafelib import \*  
>>>  
>>> # Build a DataFlow object using FeaturePlan  
>>> fp = FeaturePlan(sample\_rate=16000)  
>>> fp.addFeature('mfcc: MFCC blockSize=512 stepSize=256')  
True  
>>> fp.addFeature('mfcc\_d1: MFCC blockSize=512 stepSize=256 > Derivate DOrder=1')  
True  
>>> fp.addFeature('mfcc\_d2: MFCC blockSize=512 stepSize=256 > Derivate DOrder=2')  
True  
>>> df = fp.getDataFlow()  
>>>  
>>> # or load a DataFlow from dataflow file.  
>>> df = DataFlow()  
>>> df.load(dataflow\_file)  
True  
>>>  
>>> # configure an Engine  
>>> engine = Engine()  
>>> engine.load(df)  
True  
>>> # extract features from an audio file using AudioFileProcessor  
>>> afp = AudioFileProcessor()  
>>> afp.processFile(engine,audiofile)  
0  
>>> feats = engine.readAllOutputs()  
>>> # and play with your features  
>>>  
>>> # extract features from an audio file and write results to csv files  
>>> afp.setOutputFormat('csv','output',{'Precision':'8'})  
True  
>>> afp.processFile(engine,audiofile)  
0  
>>> # this creates output/myaudio.wav.mfcc.csv,  
>>> # output/myaudio.wav.mfcc\_d1.csv and  
>>> # output/myaudio.wav.mfcc\_d2.csv files.  
>>>  
>>> # extract features from a numpy array  
>>> import numpy  
>>> audio = numpy.random.randn(1,100000)  
>>> feats = engine.processAudio(audio)  
>>> # and play with your features