Supported features:

* **python\_speech\_features.mfcc()** - Mel Frequency Cepstral Coefficients
* **python\_speech\_features.fbank()** - Filterbank Energies
* **python\_speech\_features.logfbank()** - Log Filterbank Energies
* **python\_speech\_features.ssc()** - Spectral Subband Centroids

To use MFCC features:

from python\_speech\_features import mfcc

from python\_speech\_features import logfbank

import scipy.io.wavfile as wav

(rate,sig) = wav.read("file.wav")

mfcc\_feat = mfcc(sig,rate)

fbank\_feat = logfbank(sig,rate)

print(fbank\_feat[1:3,:])

From here you can write the features to a file etc.

**Functions provided in python\_speech\_features module**

**python\_speech\_features.base.mfcc(*signal*, *samplerate=16000*, *winlen=0.025*, *winstep=0.01*, *numcep=13*, *nfilt=26*, *nfft=512*, *lowfreq=0*, *highfreq=None*, *preemph=0.97*, *ceplifter=22*, *appendEnergy=True*, *winfunc=<function <lambda>>*)**

Compute MFCC features from an audio signal.

|  |  |
| --- | --- |
| **Parameters:** | * **signal** – the audio signal from which to compute features. Should be an N\*1 array * **samplerate** – the samplerate of the signal we are working with. * **winlen** – the length of the analysis window in seconds. Default is 0.025s (25 milliseconds) * **winstep** – the step between successive windows in seconds. Default is 0.01s (10 milliseconds) * **numcep** – the number of cepstrum to return, default 13 * **nfilt** – the number of filters in the filterbank, default 26. * **nfft** – the FFT size. Default is 512. * **lowfreq** – lowest band edge of mel filters. In Hz, default is 0. * **highfreq** – highest band edge of mel filters. In Hz, default is samplerate/2 * **preemph** – apply preemphasis filter with preemph as coefficient. 0 is no filter. Default is 0.97. * **ceplifter** – apply a lifter to final cepstral coefficients. 0 is no lifter. Default is 22. * **appendEnergy** – if this is true, the zeroth cepstral coefficient is replaced with the log of the total frame energy. * **winfunc** – the analysis window to apply to each frame. By default no window is applied. You can use numpy window functions here e.g. winfunc=numpy.hamming |
| **Returns:** | A numpy array of size (NUMFRAMES by numcep) containing features. Each row holds 1 feature vector. |

**python\_speech\_features.base.fbank(*signal*, *samplerate=16000*, *winlen=0.025*, *winstep=0.01*, *nfilt=26*, *nfft=512*, *lowfreq=0*, *highfreq=None*, *preemph=0.97*, *winfunc=<function <lambda>>*)**

Compute Mel-filterbank energy features from an audio signal.

|  |  |
| --- | --- |
| **Parameters:** | * **signal** – the audio signal from which to compute features. Should be an N\*1 array * **samplerate** – the samplerate of the signal we are working with. * **winlen** – the length of the analysis window in seconds. Default is 0.025s (25 milliseconds) * **winstep** – the step between successive windows in seconds. Default is 0.01s (10 milliseconds) * **nfilt** – the number of filters in the filterbank, default 26. * **nfft** – the FFT size. Default is 512. * **lowfreq** – lowest band edge of mel filters. In Hz, default is 0. * **highfreq** – highest band edge of mel filters. In Hz, default is samplerate/2 * **preemph** – apply preemphasis filter with preemph as coefficient. 0 is no filter. Default is 0.97. * **winfunc** – the analysis window to apply to each frame. By default no window is applied. You can use numpy window functions here e.g. winfunc=numpy.hamming |
| **Returns:** | 2 values. The first is a numpy array of size (NUMFRAMES by nfilt) containing features. Each row holds 1 feature vector. The second return value is the energy in each frame (total energy, unwindowed) |

**python\_speech\_features.base.logfbank(*signal*, *samplerate=16000*, *winlen=0.025*, *winstep=0.01*, *nfilt=26*, *nfft=512*, *lowfreq=0*, *highfreq=None*, *preemph=0.97*)**

Compute log Mel-filterbank energy features from an audio signal.

|  |  |
| --- | --- |
| **Parameters:** | * **signal** – the audio signal from which to compute features. Should be an N\*1 array * **samplerate** – the samplerate of the signal we are working with. * **winlen** – the length of the analysis window in seconds. Default is 0.025s (25 milliseconds) * **winstep** – the step between successive windows in seconds. Default is 0.01s (10 milliseconds) * **nfilt** – the number of filters in the filterbank, default 26. * **nfft** – the FFT size. Default is 512. * **lowfreq** – lowest band edge of mel filters. In Hz, default is 0. * **highfreq** – highest band edge of mel filters. In Hz, default is samplerate/2 * **preemph** – apply preemphasis filter with preemph as coefficient. 0 is no filter. Default is 0.97. |
| **Returns:** | A numpy array of size (NUMFRAMES by nfilt) containing features. Each row holds 1 feature vector. |

**python\_speech\_features.base.ssc(*signal*, *samplerate=16000*, *winlen=0.025*, *winstep=0.01*, *nfilt=26*, *nfft=512*, *lowfreq=0*, *highfreq=None*, *preemph=0.97*, *winfunc=<function <lambda>>*)**

Compute Spectral Subband Centroid features from an audio signal.

|  |  |
| --- | --- |
| **Parameters:** | * **signal** – the audio signal from which to compute features. Should be an N\*1 array * **samplerate** – the samplerate of the signal we are working with. * **winlen** – the length of the analysis window in seconds. Default is 0.025s (25 milliseconds) * **winstep** – the step between successive windows in seconds. Default is 0.01s (10 milliseconds) * **nfilt** – the number of filters in the filterbank, default 26. * **nfft** – the FFT size. Default is 512. * **lowfreq** – lowest band edge of mel filters. In Hz, default is 0. * **highfreq** – highest band edge of mel filters. In Hz, default is samplerate/2 * **preemph** – apply preemphasis filter with preemph as coefficient. 0 is no filter. Default is 0.97. * **winfunc** – the analysis window to apply to each frame. By default no window is applied. You can use numpy window functions here e.g. winfunc=numpy.hamming |
| **Returns:** | A numpy array of size (NUMFRAMES by nfilt) containing features. Each row holds 1 feature vector. |

**python\_speech\_features.base.hz2mel(*hz*)**

Convert a value in Hertz to Mels

|  |  |
| --- | --- |
| **Parameters:** | **hz** – a value in Hz. This can also be a numpy array, conversion proceeds element-wise. |
| **Returns:** | a value in Mels. If an array was passed in, an identical sized array is returned. |

**python\_speech\_features.base.mel2hz(*mel*)**

Convert a value in Mels to Hertz

|  |  |
| --- | --- |
| **Parameters:** | **mel** – a value in Mels. This can also be a numpy array, conversion proceeds element-wise. |
| **Returns:** | a value in Hertz. If an array was passed in, an identical sized array is returned. |

**python\_speech\_features.base.get\_filterbanks(*nfilt=20*, *nfft=512*, *samplerate=16000*, *lowfreq=0*, *highfreq=None*)**

Compute a Mel-filterbank. The filters are stored in the rows, the columns correspond to fft bins. The filters are returned as an array of size nfilt \* (nfft/2 + 1)

|  |  |
| --- | --- |
| **Parameters:** | * **nfilt** – the number of filters in the filterbank, default 20. * **nfft** – the FFT size. Default is 512. * **samplerate** – the samplerate of the signal we are working with. Affects mel spacing. * **lowfreq** – lowest band edge of mel filters, default 0 Hz * **highfreq** – highest band edge of mel filters, default samplerate/2 |
| **Returns:** | A numpy array of size nfilt \* (nfft/2 + 1) containing filterbank. Each row holds 1 filter. |

**python\_speech\_features.base.lifter(*cepstra*, *L=22*)**

Apply a cepstral lifter the the matrix of cepstra. This has the effect of increasing the magnitude of the high frequency DCT coeffs.

|  |  |
| --- | --- |
| **Parameters:** | * **cepstra** – the matrix of mel-cepstra, will be numframes \* numcep in size. * **L** – the liftering coefficient to use. Default is 22. L <= 0 disables lifter. |

**python\_speech\_features.base.delta(*feat*, *N*)**

Compute delta features from a feature vector sequence.

|  |  |
| --- | --- |
| **Parameters:** | * **feat** – A numpy array of size (NUMFRAMES by number of features) containing features. Each row holds 1 feature vector. * **N** – For each frame, calculate delta features based on preceding and following N frames |
| **Returns:** | A numpy array of size (NUMFRAMES by number of features) containing delta features. Each row holds 1 delta feature vector. |

**Functions provided in sigproc module**

**python\_speech\_features.sigproc.framesig(*sig*, *frame\_len*, *frame\_step*, *winfunc=<function <lambda>>*, *stride\_trick=True*)**

Frame a signal into overlapping frames.

|  |  |
| --- | --- |
| **Parameters:** | * **sig** – the audio signal to frame. * **frame\_len** – length of each frame measured in samples. * **frame\_step** – number of samples after the start of the previous frame that the next frame should begin. * **winfunc** – the analysis window to apply to each frame. By default no window is applied. * **stride\_trick** – use stride trick to compute the rolling window and window multiplication faster |
| **Returns:** | an array of frames. Size is NUMFRAMES by frame\_len. |

**python\_speech\_features.sigproc.deframesig(*frames*, *siglen*, *frame\_len*, *frame\_step*, *winfunc=<function <lambda>>*)**

Does overlap-add procedure to undo the action of framesig.

|  |  |
| --- | --- |
| **Parameters:** | * **frames** – the array of frames. * **siglen** – the length of the desired signal, use 0 if unknown. Output will be truncated to siglen samples. * **frame\_len** – length of each frame measured in samples. * **frame\_step** – number of samples after the start of the previous frame that the next frame should begin. * **winfunc** – the analysis window to apply to each frame. By default no window is applied. |
| **Returns:** | a 1-D signal. |

**python\_speech\_features.sigproc.magspec(*frames*, *NFFT*)**

Compute the magnitude spectrum of each frame in frames. If frames is an NxD matrix, output will be Nx(NFFT/2+1).

|  |  |
| --- | --- |
| **Parameters:** | * **frames** – the array of frames. Each row is a frame. * **NFFT** – the FFT length to use. If NFFT > frame\_len, the frames are zero-padded. |
| **Returns:** | If frames is an NxD matrix, output will be Nx(NFFT/2+1). Each row will be the magnitude spectrum of the corresponding frame. |

**python\_speech\_features.sigproc.powspec(*frames*, *NFFT*)**

Compute the power spectrum of each frame in frames. If frames is an NxD matrix, output will be Nx(NFFT/2+1).

|  |  |
| --- | --- |
| **Parameters:** | * **frames** – the array of frames. Each row is a frame. * **NFFT** – the FFT length to use. If NFFT > frame\_len, the frames are zero-padded. |
| **Returns:** | If frames is an NxD matrix, output will be Nx(NFFT/2+1). Each row will be the power spectrum of the corresponding frame. |

**python\_speech\_features.sigproc.logpowspec(*frames*, *NFFT*, *norm=1*)**

Compute the log power spectrum of each frame in frames. If frames is an NxD matrix, output will be Nx(NFFT/2+1).

|  |  |
| --- | --- |
| **Parameters:** | * **frames** – the array of frames. Each row is a frame. * **NFFT** – the FFT length to use. If NFFT > frame\_len, the frames are zero-padded. * **norm** – If norm=1, the log power spectrum is normalised so that the max value (across all frames) is 0. |
| **Returns:** | If frames is an NxD matrix, output will be Nx(NFFT/2+1). Each row will be the log power spectrum of the corresponding frame. |

**python\_speech\_features.sigproc.preemphasis(*signal*, *coeff=0.95*)**

perform preemphasis on the input signal.

|  |  |
| --- | --- |
| **Parameters:** | * **signal** – The signal to filter. * **coeff** – The preemphasis coefficient. 0 is no filter, default is 0.95. |
| **Returns:** | the filtered signal. |