SpeechPy Documentation

Release master

Amirsina Torfi

Jul 27, 2018

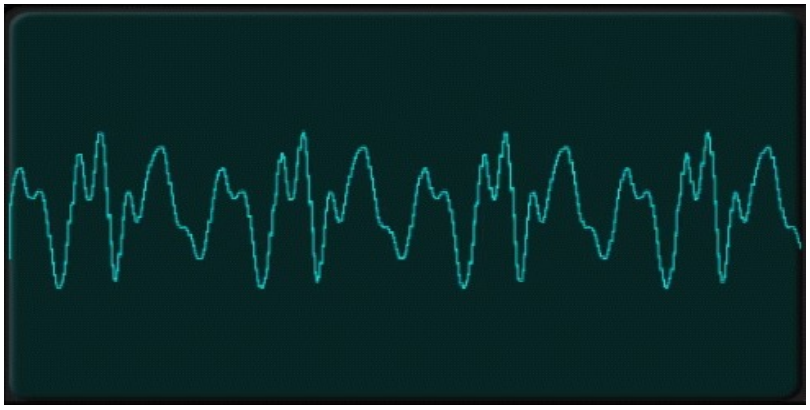
Preface

CHAPTER 1

Introduction

1.1 Foreword

The purpose of this project is to provide a package for speech processing and feature extraction. This library provides most frequent used speech features including MFCCs and filterbank energies alongside with the log-energy of filterbanks.



1.2 Motivation

There are different motivations for this open source project.

1.2.1 Deep Learning application

One of the main reasons for creating this package was to provide necessary features for deep learning applications such as ASR(Automatic Speech Recognition) or SR(Speaker Recognition). As a results, most of the features that are necessary are provided hear.

1.2.2 Pythonic Packaging

Another reason for creating this package was to have a Pythonic environment for speech recognition and feature extraction due to the fact that the Python language is becoming ubiquotous!

1.3 How to Install?

There are two possible ways for installation of this package: local installation and PyPi.

1.3.1 Local Installation

For local installation at first the repository must be cloned:

```
git clone https://github.com/astorfi/speech_feature_extraction.git
```

After cloning the reposity, root to the repository directory then execute:

```
python setup.py develop
```

1.3.2 Pypi

The package is available on PyPi. For direct installation simply execute the following:

```
pip install speechpy
```

1.4 Citation

If you used this package, please cite it as follows:

```
@misc{amirsina_torfi_2017_840395,
            author = {Amirsina Torfi},
            title = {{SpeechPy: Speech recognition and feature extraction}},
            month = aug,
            year = 2017,
            doi = {10.5281/zenodo.840395},
    url = {https://doi.org/10.5281/zenodo.840395}
}
```

CHAPTER 2

Preprocessing

Processing module for signal processing operations.

This module demonstrates documentation for the signal processing function which are required as internal computations in the package.

ivar preemphasis Preemphasising on the signal. This is a preprocessing step.

ivar stack_frames Create stacking frames from the raw signal.

ivar fft_spectrum Calculation of the Fast Fourier Transform.

ivar power_spectrum Power Spectrum calculation.

ivar log_power_spectrum Log Power Spectrum calculation.

ivar derivative_extraction Calculation of the derivative of the extracted featurs.

ivar cmvn Cepstral mean variance normalization. This is a post processing operation.

ivar cmvnw Cepstral mean variance normalization over the sliding window. This is a post processing operation.

2.1 Pre-emphasis

speechpy.processing.**preemphasis**(*signal*, *shift=1*, *cof=0.98*)

preemphasising on the signal.

Parameters

- **signal** (*array*) – The input signal.
- **shift** (*int*) – The shift step.
- **cof** (*float*) – The preemphasising coefficient. 0 equals to no filtering.

Returns The pre-emphasized signal.

Return type array

2.2 Stacking

speechpy.processing.**stack_frames**(*sig*, *sampling_frequency*, *frame_length=0.02*, *frame_stride=0.02*, *filter=<function <lambda>>*, *zero_padding=True*)

Frame a signal into overlapping frames.

Parameters

- **sig** (*array*) – The audio signal to frame of size (N,).
- **sampling_frequency** (*int*) – The sampling frequency of the signal.
- **frame_length** (*float*) – The length of the frame in second.
- **frame_stride** (*float*) – The stride between frames.
- **filter** (*array*) – The time-domain filter for applying to each frame. By default it is one so nothing will be changed.
- **zero_padding** (*bool*) – If the samples is not a multiple of frame_length(number of frames sample), zero padding will be done for generating last frame.

Returns Stacked_frames-Array of frames of size (number_of_frames x frame_len).

Return type array

2.3 FFT Spectrum

speechpy.processing.**fft_spectrum**(*frames*, *fft_points=512*)

This function computes the one-dimensional n-point discrete Fourier Transform (DFT) of a real-valued array by means of an efficient algorithm called the Fast Fourier Transform (FFT). Please refer to https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.rfft.html for further details.

Parameters

- **frames** (*array*) – The frame array in which each row is a frame.
- **fft_points** (*int*) – The length of FFT. If fft_length is greater than frame_len, the frames will be zero-padded.

Returns The fft spectrum. If frames is an num_frames x sample_per_frame matrix, output will be num_frames x FFT_LENGTH.

Return type array

2.4 Power Spectrum

speechpy.processing.**power_spectrum**(*frames*, *fft_points=512*)

Power spectrum of each frame.

Parameters

- **frames** (*array*) – The frame array in which each row is a frame.
- **fft_points** (*int*) – The length of FFT. If fft_length is greater than frame_len, the frames will be zero-padded.

Returns The power spectrum. If frames is an num_frames x sample_per_frame matrix, output will be num_frames x fft_length.

Return type array

2.5 Power Spectrum Log

speechpy.processing.**log_power_spectrum**(*frames*, *fft_points=512*, *normalize=True*)

Log power spectrum of each frame in frames.

Parameters

- **frames** (*array*) – The frame array in which each row is a frame.
- **fft_points** (*int*) – The length of FFT. If fft_length is greater than frame_len, the frames will be zero-padded.
- **normalize** (*bool*) – If normalize=True, the log power spectrum will be normalized.

Returns The power spectrum - If frames is an num_frames x sample_per_frame matrix, output will be num_frames x fft_length.

Return type array

2.6 Derivative Extraction

speechpy.processing.**derivative_extraction**(*feat*, *DeltaWindows*)

This function the derivative features.

Parameters

- **feat** (*array*) – The main feature vector(For returning the second order derivative it can be first-order derivative).
- **DeltaWindows** (*int*) – The value of DeltaWindows is set using the configuration parameter DELTAWINDOW.

Returns Derivative feature vector - A NUMFRAMESxNUMFEATURES numpy array which is the derivative features along the features.

Return type array

CHAPTER 3

Features

feature module.

This module provides functions for calculating the main speech features that the package is aimed to extract as well as the required elements.

Functions:

filterbanks: Compute the Mel-filterbanks The filterbanks must be created for extracting speech features such as MFCC.

mfcc: Extracting Mel Frequency Cepstral Coefficient feature.

mfe: Extracting Mel Energy feature.

lmfe: Extracting Log Mel Energy feature.

extract_derivative_feature: Extract the first and second derivative features. This finction, directly use the `derivative_extraction` function in the `processing` module.

3.1 MFCC

`speechpy.feature.`**`mfcc`**(*signal*, *sampling_frequency*, *frame_length=0.02*, *frame_stride=0.01*, *num_cepstral=13*, *num_filters=40*, *fft_length=512*, *low_frequency=0*, *high_frequency=None*, *dc_elimination=True*)

Compute MFCC features from an audio signal.

Parameters

- **`signal`** (*array*) – the audio signal from which to compute features. Should be an N x 1 array
- **`sampling_frequency`** (*int*) – the sampling frequency of the signal we are working with.
- **`frame_length`** (*float*) – the length of each frame in seconds. Default is 0.020s

- **frame_stride** (*float*) – the step between successive frames in seconds. Default is 0.02s (means no overlap)
- **num_filters** (*int*) – the number of filters in the filterbank, default 40.
- **fft_length** (*int*) – number of FFT points. Default is 512.
- **low_frequency** (*float*) – lowest band edge of mel filters. In Hz, default is 0.
- **high_frequency** (*float*) – highest band edge of mel filters. In Hz, default is samplerate/2
- **num_cepstral** (*int*) – Number of cepstral coefficients.
- **dc_elimination** (*bool*) – hIf the first dc component should be eliminated or not.

Returns A numpy array of size (num_frames x num_cepstral) containing mfcc features.

Return type array

3.2 Mel Frequency Energy

speechpy.feature.**mfe**(*signal*, *sampling_frequency*, *frame_length=0.02*, *frame_stride=0.01*, *num_filters=40*, *fft_length=512*, *low_frequency=0*, *high_frequency=None*)

Compute Mel-filterbank energy features from an audio signal.

Parameters

- **signal** (*array*) – the audio signal from which to compute features. Should be an N x 1 array
- **sampling_frequency** (*int*) – the sampling frequency of the signal we are working with.
- **frame_length** (*float*) – the length of each frame in seconds. Default is 0.020s
- **frame_stride** (*float*) – the step between successive frames in seconds. Default is 0.02s (means no overlap)
- **num_filters** (*int*) – the number of filters in the filterbank, default 40.
- **fft_length** (*int*) – number of FFT points. Default is 512.
- **low_frequency** (*float*) – lowest band edge of mel filters. In Hz, default is 0.
- **high_frequency** (*float*) – highest band edge of mel filters. In Hz, default is samplerate/2

Returns features - the energy of fiterbank of size num_frames x num_filters. The energy of each frame: num_frames x 1

Return type array

3.3 Log Mel Frequency Energy

speechpy.feature.**lmfe**(*signal*, *sampling_frequency*, *frame_length=0.02*, *frame_stride=0.01*, *num_filters=40*, *fft_length=512*, *low_frequency=0*, *high_frequency=None*)

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

Parameters

- **signal** (*array*) – the audio signal from which to compute features. Should be an N x 1 array
- **sampling_frequency** (*int*) – the sampling frequency of the signal we are working with.
- **frame_length** (*float*) – the length of each frame in seconds. Default is 0.020s
- **frame_stride** (*float*) – the step between successive frames in seconds. Default is 0.02s (means no overlap)
- **num_filters** (*int*) – the number of filters in the filterbank, default 40.
- **fft_length** (*int*) – number of FFT points. Default is 512.
- **low_frequency** (*float*) – lowest band edge of mel filters. In Hz, default is 0.
- **high_frequency** (*float*) – highest band edge of mel filters. In Hz, default is samplerate/2

Returns Features - The log energy of fiterbank of size num_frames x num_filters frame_log_energies. The log energy of each frame num_frames x 1

Return type array

3.4 Extract Derivative Features

speechpy.feature.**extract_derivative_feature**(*feature*)

This function extracts temporal derivative features which are first and second derivatives.

Parameters **feature** (*array*) – The feature vector which its size is: N x M

Returns The feature cube vector which contains the static, first and second derivative features of size: N x M x 3

Return type array

CHAPTER 4

postprocessing

Processing module for signal processing operations.

This module demonstrates documentation for the signal processing function which are required as internal computations in the package.

ivar preemphasis Preemphasising on the signal. This is a preprocessing step.

ivar stack_frames Create stacking frames from the raw signal.

ivar fft_spectrum Calculation of the Fast Fourier Transform.

ivar power_spectrum Power Spectrum calculation.

ivar log_power_spectrum Log Power Spectrum calculation.

ivar derivative_extraction Calculation of the derivative of the extracted featurs.

ivar cmvn Cepstral mean variance normalization. This is a post processing operation.

ivar cmvnw Cepstral mean variance normalization over the sliding window. This is a post processing operation.

4.1 Global Cepstral Mean and Variance Normalization

speechpy.processing.**cmvn**(*vec*, *variance_normalization=False*)

This function is aimed to perform global cepstral mean and variance normalization (CMVN) on input feature vector "vec". The code assumes that there is one observation per row.

Parameters

- **vec** (*array*) – input feature matrix (size:(num_observation,num_features))
- **variance_normalization** (*bool*) – If the variance normilization should be performed or not.

Returns The mean(or mean+variance) normalized feature vector.

Return type array

4.2 Local Cepstral Mean and Variance Normalization over Sliding Window

speechpy.processing.**cmvnw**(*vec*, *win_size=301*, *variance_normalization=False*)

This function is aimed to perform local cepstral mean and variance normalization on a sliding window. The code assumes that there is one observation per row.

Parameters

- **vec** (*array*) – input feature matrix (size:(num_observation,num_features))
- **win_size** (*int*) – The size of sliding window for local normalization. Default=301 which is around 3s if 100 Hz rate is considered(== 10ms frame stide)
- **variance_normalization** (*bool*) – If the variance normilization should be performed or not.

Returns The mean(or mean+variance) normalized feature vector.

Return type array

CHAPTER 5

test

5.1 Test Package

Once the package has been installed, a test file can be directly run to show the results. The test example can be seen in `test/test_package.py` as below:

```
import scipy.io.wavfile as wav
import numpy as np
import speechpy
import os

file_name = os.path.join(os.path.dirname(os.path.abspath(__file__)),'Alesis-Sanctuary-
↪QCard-AcoustcBas-C2.wav')
fs, signal = wav.read(file_name)
signal = signal[:,0]

# Example of pre-emphasizing.
signal_preemphasized = speechpy.processing.preemphasis(signal, cof=0.98)

# Example of staching frames
frames = speechpy.processing.stack_frames(signal, sampling_frequency=fs, frame_
↪length=0.020, frame_stride=0.01, filter=lambda x: np.ones((x,)),
     zero_padding=True)

# Example of extracting power spectrum
power_spectrum = speechpy.processing.power_spectrum(frames, fft_points=512)
print('power spectrum shape=', power_spectrum.shape)

############ Extract MFCC features #############
mfcc = speechpy.feature.mfcc(signal, sampling_frequency=fs, frame_length=0.020, frame_
↪stride=0.01,
         num_filters=40, fft_length=512, low_frequency=0, high_frequency=None)
mfcc_cmvn = speechpy.processing.cmvnw(mfcc,win_size=301,variance_normalization=True)
print('mfcc(mean + variance normalized) feature shape=', mfcc_cmvn.shape)
```

(continues on next page)

(continued from previous page)

```
mfcc_feature_cube = speechpy.feature.extract_derivative_feature(mfcc)
print('mfcc feature cube shape=', mfcc_feature_cube.shape)

############# Extract logenergy features #############
logenergy = speechpy.feature.lmfe(signal, sampling_frequency=fs, frame_length=0.020,␣
↪frame_stride=0.01,
         num_filters=40, fft_length=512, low_frequency=0, high_frequency=None)
logenergy_feature_cube = speechpy.feature.extract_derivative_feature(logenergy)
print('logenergy features=', logenergy.shape)
```

5.2 Test Local

There is an alternative local way of testing without the necessity to package installation. The local test example can be found in `test/test_package.py` as follows:

```
import scipy.io.wavfile as wav
import numpy as np
import os
import sys
lib_path = os.path.abspath(os.path.join('..'))
print(lib_path)
sys.path.append(lib_path)
import speechpy
import os

file_name = os.path.join(os.path.dirname(os.path.abspath(__file__)),'Alesis-Sanctuary-
↪QCard-AcoustcBas-C2.wav')
fs, signal = wav.read(file_name)
signal = signal[:,0]

# Example of pre-emphasizing.
signal_preemphasized = speechpy.processing.preemphasis(signal, cof=0.98)

# Example of staching frames
frames = speechpy.processing.stack_frames(signal, sampling_frequency=fs, frame_
↪length=0.020, frame_stride=0.01, filter=lambda x: np.ones((x,)),
         zero_padding=True)

# Example of extracting power spectrum
power_spectrum = speechpy.processing.power_spectrum(frames, fft_points=512)
print('power spectrum shape=', power_spectrum.shape)

############# Extract MFCC features #############
mfcc = speechpy.feature.mfcc(signal, sampling_frequency=fs, frame_length=0.020, frame_
↪stride=0.01,
             num_filters=40, fft_length=512, low_frequency=0, high_frequency=None)
mfcc_cmvn = speechpy.processing.cmvnw(mfcc,win_size=301,variance_normalization=True)
print('mfcc(mean + variance normalized) feature shape=', mfcc_cmvn.shape)

mfcc_feature_cube = speechpy.feature.extract_derivative_feature(mfcc)
print('mfcc feature cube shape=', mfcc_feature_cube.shape)

############# Extract logenergy features #############
```

(continues on next page)

(continued from previous page)

```
logenergy = speechpy.feature.lmfe(signal, sampling_frequency=fs, frame_length=0.020,␣
↪frame_stride=0.01,
             num_filters=40, fft_length=512, low_frequency=0, high_frequency=None)
logenergy_feature_cube = speechpy.feature.extract_derivative_feature(logenergy)
print('logenergy features=', logenergy.shape)
```

For ectracting the feature at first, the signal samples will be stacked into frames. The features are computed for each frame in the stacked frames collection.

5.3 Dependencies

Two packages of `Scipy` and `NumPy` are the required dependencies which will be installed automatically by running the `setup.py` file.

CHAPTER 6

Contributing

When contributing to this repository, you are more than welcome to discuss your feedback with any of the owners of this repository. *For typos, please do not create a pull request. Instead, declare them in issues or email the repository owner*. For technical and conceptual questions please feel free to **directly contact the repository owner**. Before asking general questions related to the concepts and techniques provided in this project, **please make sure to read and understand its associated paper**.

Please note we have a code of conduct, please follow it in all your interactions with the project.

6.1 Pull Request Process

Please consider the following criterions in order to help us in a better way:

1. The pull request is mainly expected to be a code script suggestion or improvement.
2. A pull request related to non-code-script sections is expected to make a significant difference in the documentation. Otherwise, it is expected to be announced in the issues section.
3. Ensure any install or build dependencies are removed before the end of the layer when doing a build and creating a pull request.
4. Add comments with details of changes to the interface, this includes new environment variables, exposed ports, useful file locations and container parameters.
5. You may merge the Pull Request in once you have the sign-off of at least one other developer, or if you do not have permission to do that, you may request the owner to merge it for you if you believe all checks are passed.

6.2 Final Note

We are looking forward to your kind feedback. Please help us to improve this open source project and make our work better. For contribution, please create a pull request and we will investigate it promptly. Once again, we appreciate your kind feedback and elaborate code inspections.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index