Disvoice Documentation

Release 0.1.1

Camilo Vasquez

Mar 04, 2021

CONTENTS

DisVoice is a python framework designed to compute features from speech files. Disvoice computes glottal, phonation, articulation, prosody, phonological, and features representation learnig strategies using autoencders. The features can be computed both from sustained vowels and continuous speech utterances with the aim to recognize praliguistic aspects from speech.

The features can be used in classifiers to recognize emotions, or communication capabilities of patients with different speech disorders including diseases with functional origin such as larinx cancer or nodules; craneo-facial based disorders such as hipernasality developed by cleft-lip and palate; or neurodegenerative disorders such as Parkinson's or Hungtinton's diseases.

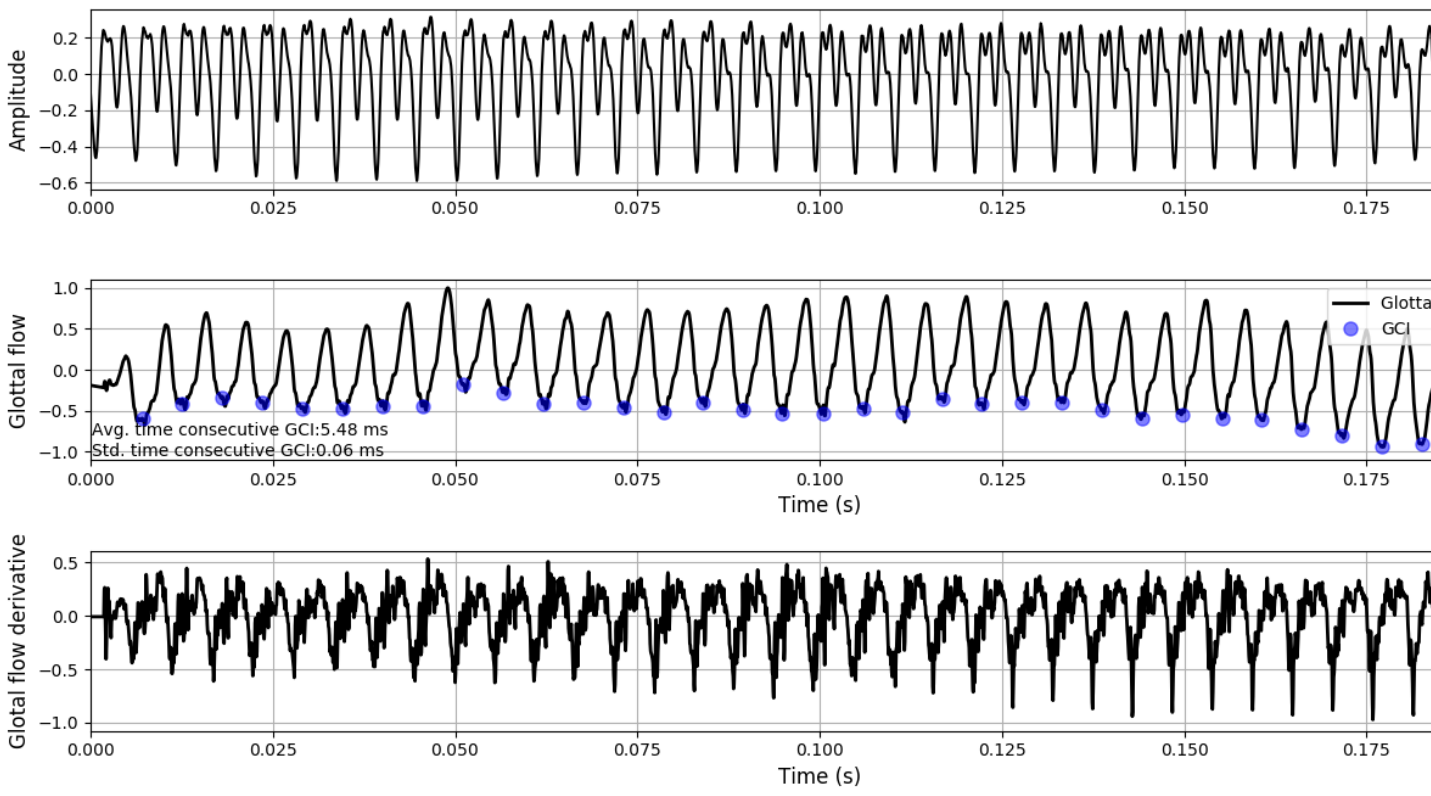
The features are also suitable to evaluate mood problems like depression based on speech patterns.

For additional details about each feature type, and how to use DisVoice, please check

CHAPTER

ONE

GLOTTAL FEATURES



class `glottal.`**`Glottal`**

Compute features based on the glottal source reconstruction from sustained vowels and continuous speech.

For continuous speech, the features are computed over voiced segments

Nine descriptors are computed:

1. Variability of time between consecutive glottal closure instants (GCI)
2. Average opening quotient (OQ) for consecutive glottal cycles-> rate of opening phase duration / duration of glottal cycle
3. Variability of opening quotient (OQ) for consecutive glottal cycles-> rate of opening phase duration /duration of glottal cycle
4. Average normalized amplitude quotient (NAQ) for consecutive glottal cycles-> ratio of the amplitude quotient and the duration of the glottal cycle

5. Variability of normalized amplitude quotient (NAQ) for consecutive glottal cycles-> ratio of the amplitude quotient and the duration of the glottal cycle
6. Average H1H2: Difference between the first two harmonics of the glottal flow signal
7. Variability H1H2: Difference between the first two harmonics of the glottal flow signal
8. Average of Harmonic richness factor (HRF): ratio of the sum of the harmonics amplitude and the amplitude of the fundamental frequency
9. Variability of HRF

Static or dynamic matrices can be computed:

Static matrix is formed with 36 features formed with (9 descriptors) x (4 functionals: mean, std, skewness, kurtosis)

Dynamic matrix is formed with the 9 descriptors computed for frames of 200 ms length with a time-shift of 50 ms.

Notes:

1. The fundamental frequency is computed using the RAPT algorithm.

```
>>> python glottal.py <file_or_folder_audio> <file_features> <dynamic_or_static>
↪<plots (true,  false)> <format (csv, txt, npy, kaldi, torch)>
```

Examples command line:

```
>>> python glottal.py "../audios/001_a1_PCGITA.wav" "glottalfeaturesAst.txt"
↪"static" "true" "txt"
>>> python glottal.py "../audios/098_u1_PCGITA.wav" "glottalfeaturesUst.csv"
↪"static" "true" "csv"
>>> python glottal.py "../audios/098_u1_PCGITA.wav" "glottalfeaturesUst.ark"
↪"dynamic" "true" "kaldi"
>>> python glottal.py "../audios/098_u1_PCGITA.wav" "glottalfeaturesUst.pt"
↪"dynamic" "true" "torch"
```

Examples directly in Python

```
>>> from disvoice.glottal import Glottal
>>> glottal=Glottal()
>>> file_audio="../audios/001_a1_PCGITA.wav"
>>> features=glottal.extract_features_file(file_audio, static, plots=True, fmt=
↪"numpy")
>>> features2=glottal.extract_features_file(file_audio, static, plots=True, fmt=
↪"dataframe")
>>> features3=glottal.extract_features_file(file_audio, dynamic, plots=True, fmt=
↪"torch")
```

```
>>> path_audios="../audios/"
>>> features1=glottal.extract_features_path(path_audios, static, plots=False, fmt=
↪"numpy")
>>> features2=glottal.extract_features_path(path_audios, static, plots=False, fmt=
↪"torch")
>>> features3=glottal.extract_features_path(path_audios, static, plots=False, fmt=
↪"dataframe")
```

extract_features_file (*audio*, *static=True*, *plots=False*, *fmt='npy'*, *kaldi_file=''*)

Extract the glottal features from an audio file

Parameters

- **audio** – .wav audio file.
- **static** – whether to compute and return statistic functionals over the feature matrix, or return the feature matrix computed over frames
- **plots** – timeshift to extract the features
- **fmt** – format to return the features (npy, dataframe, torch, kaldi)
- **kaldi_file** – file to store kaldi features, only valid when fmt==”kaldi”

Returns features computed from the audio file.

```
>>> glottal=Glottal()
>>> file_audio="../audios/001_a1_PCGITA.wav"
>>> features1=glottal.extract_features_file(file_audio, static=True,␣
↪plots=True, fmt="npy")
>>> features2=glottal.extract_features_file(file_audio, static=True,␣
↪plots=True, fmt="dataframe")
>>> features3=glottal.extract_features_file(file_audio, static=False,␣
↪plots=True, fmt="torch")
>>> glottal.extract_features_file(file_audio, static=False, plots=False, fmt=
↪"kaldi", kaldi_file="./test.ark")
```

extract_features_path (*path_audio*, *static=True*, *plots=False*, *fmt='npy'*, *kaldi_file=''*)
Extract the glottal features for audios inside a path

Parameters

- **path_audio** – directory with (.wav) audio files inside, sampled at 16 kHz
- **static** – whether to compute and return statistic functionals over the feature matrix, or return the feature matrix computed over frames
- **plots** – timeshift to extract the features
- **fmt** – format to return the features (npy, dataframe, torch, kaldi)
- **kaldi_file** – file to store kaldifeatures, only valid when fmt==”kaldi”

Returns features computed from the audio file.

```
>>> glottal=Glottal()
>>> path_audio="../audios/"
>>> features1=glottal.extract_features_path(path_audio, static=True,␣
↪plots=False, fmt="npy")
>>> features2=glottal.extract_features_path(path_audio, static=True,␣
↪plots=False, fmt="csv")
>>> features3=glottal.extract_features_path(path_audio, static=False,␣
↪plots=True, fmt="torch")
>>> glottal.extract_features_path(path_audio, static=False, plots=False, fmt=
↪"kaldi", kaldi_file="./test.ark")
```

extract_glottal_signal (*x*, *fs*)
Extract the glottal flow and the glottal flow derivative signals

Parameters

- **x** – data from the speech signal.
- **fs** – sampling frequency

Returns glottal signal

Returns derivative of the glottal signal

Returns glottal closure instants

```
>>> from scipy.io.wavfile import read
>>> glottal=Glottal()
>>> file_audio="../audios/001_a1_PCGITA.wav"
>>> fs, data_audio=read(audio)
>>> glottal, g_iaif, GCIs=glottal.extract_glottal_signal(data_audio, fs)
```

`plot_glottal`(*data_audio*, *fs*, *GCI*, *glottal_flow*, *glottal_sig*)

Plots of the glottal features

Parameters

- **`data_audio`** – speech signal.
- **`fs`** – sampling frequency
- **`GCI`** – glottal closure instants
- **`glottal_flow`** – glottal flow
- **`glottal_sig`** – reconstructed glottal signal

Returns plots of the glottal features.

`glottal.`**`SE_VQ_varF0`**(*x*, *fs*, *f0=None*)

Function to extract GCIs using an adapted version of the SEDREAMS algorithm which is optimised for non-modal voice qualities (SE-VQ). Ncand maximum peaks are selected from the LP-residual signal in the interval defined by the mean-based signal.

A dynamic programming algorithm is then used to select the optimal path of GCI locations. Then a post-processing method, using the output of a resonator applied to the residual signal, is carried out to remove false positives occurring in creaky speech regions.

Note that this method is slightly different from the standard SE-VQ algorithm as the mean based signal is calculated using a variable window length.

This is set using an f0 contour interpolated over unvoiced regions and heavily smoothed. This is particularly useful for speech involving large f0 excursions (i.e. very expressive speech).

Parameters

- **`x`** – speech signal (in samples)
- **`fs`** – sampling frequency (Hz)
- **`f0`** – f0 contour (optional), otherwise its computed using the RAPT algorithm

Returns GCI Glottal closure instants (in samples)

References: Kane, J., Gobl, C., (2013) 'Evaluation of glottal closure instant detection in a range of voice qualities', Speech Communication 55(2), pp. 295-314.

ORIGINAL FUNCTION WAS CODED BY JOHN KANE AT THE PHONETICS AND SPEECH LAB IN TRINITY COLLEGE DUBLIN ON 2013.

THE SEDREAMS FUNCTION WAS CODED BY THOMAS DRUGMAN OF THE UNIVERSITY OF MONS

THE CODE WAS TRANSLATED TO PYTHON AND ADAPTED BY J. C. Vasquez-Correa AT PATTERN RECOGNITION LAB UNIVERSITY OF ERLANGEN NUREMBER- GERMANY AND UNIVERSTY OF ANTIOQUIA, COLOMBIA JCAMILO.VASQUEZ@UDEA.EDU.CO https//jcvasquezc.github.io `glottal.`**`IAIF`**(*x*, *fs*, *GCI*)

Function to carry out iterative and adaptive inverse filtering (Alku et al 1992).

Parameters

- **`x`** – speech signal (in samples)
- **`fs`** – sampling frequency (in Hz)
- **`GCI`** – Glottal closure instants (in samples)

Returns glottal flow derivative estimate

Function Coded by John Kane @ The Phonetics and Speech Lab Trinity College Dublin, August 2012

THE CODE WAS TRANSLATED TO PYTHON AND ADAPTED BY J. C. Vasquez-Correa AT PATTERN RECOGNITION LAB UNIVERSITY OF ERLANGEN NUREMBER- GERMANY AND UNIVERSTY OF ANTIOQUIA, COLOMBIA JCAMILO.VASQUEZ@UDEA.EDU.CO https//jcvasquezc.github.io

`glottal.`**`get_vq_params`**(*gf*, *gfd*, *fs*, *GCI*)

Function to estimate the glottal parameters: NAQ, QOQ, H1-H2, and HRF

This function can be used to estimate a range of conventional glottal source parameters often used in the literature. This includes: the normalized amplitude quotient (NAQ), the quasi-open quotient (QOQ), the difference in amplitude of the first two harmonics of the differentiated glottal source spectrum (H1-H2), and the harmonic richness factor (HRF)

Parameters

- **`gf`** – [samples] [N] Glottal flow estimation
- **`gfd`** – [samples] [N] Glottal flow derivative estimation
- **`fs`** – [Hz] [1] sampling frequency
- **`GCI`** – [samples] [M] Glottal closure instants

Returns NAQ [s,samples] [Mx2] Normalised amplitude quotient

Returns QOQ[s,samples] [Mx2] Quasi-open quotient

Returns H1H2[s,dB] [Mx2] Difference in glottal harmonic amplitude

Returns HRF[s,samples] [Mx2] Harmonic richness factor

References: [1] Alku, P., B ackstrom, T., and Vilkman, E. Normalized amplitude quotient for parameterization of the glottal flow. Journal of the Acoustical Society of America, 112(2):701-710, 2002.

[2] Hacki, T. Klassifizierung von glottisdysfunktionen mit hilfe der elektroglottographie. Folia Phoniatrica, pages 43-48, 1989.

[3] Alku, P., Strik, H., and Vilkman, E. Parabolic spectral parameter - A new method for quantification of the glottal flow. Speech Communication, 22(1):67-79, 1997.

[4] Hanson, H. M. Glottal characteristics of female speakers: Acoustic correlates. Journal of the Acoustical Society of America, 10(1):466-481, 1997.

[5] Childers, D. G. and Lee, C. K. Voice quality factors: Analysis, synthesis and perception. Journal of the Acoustical Society of America, 90(5):2394-2410, 1991.

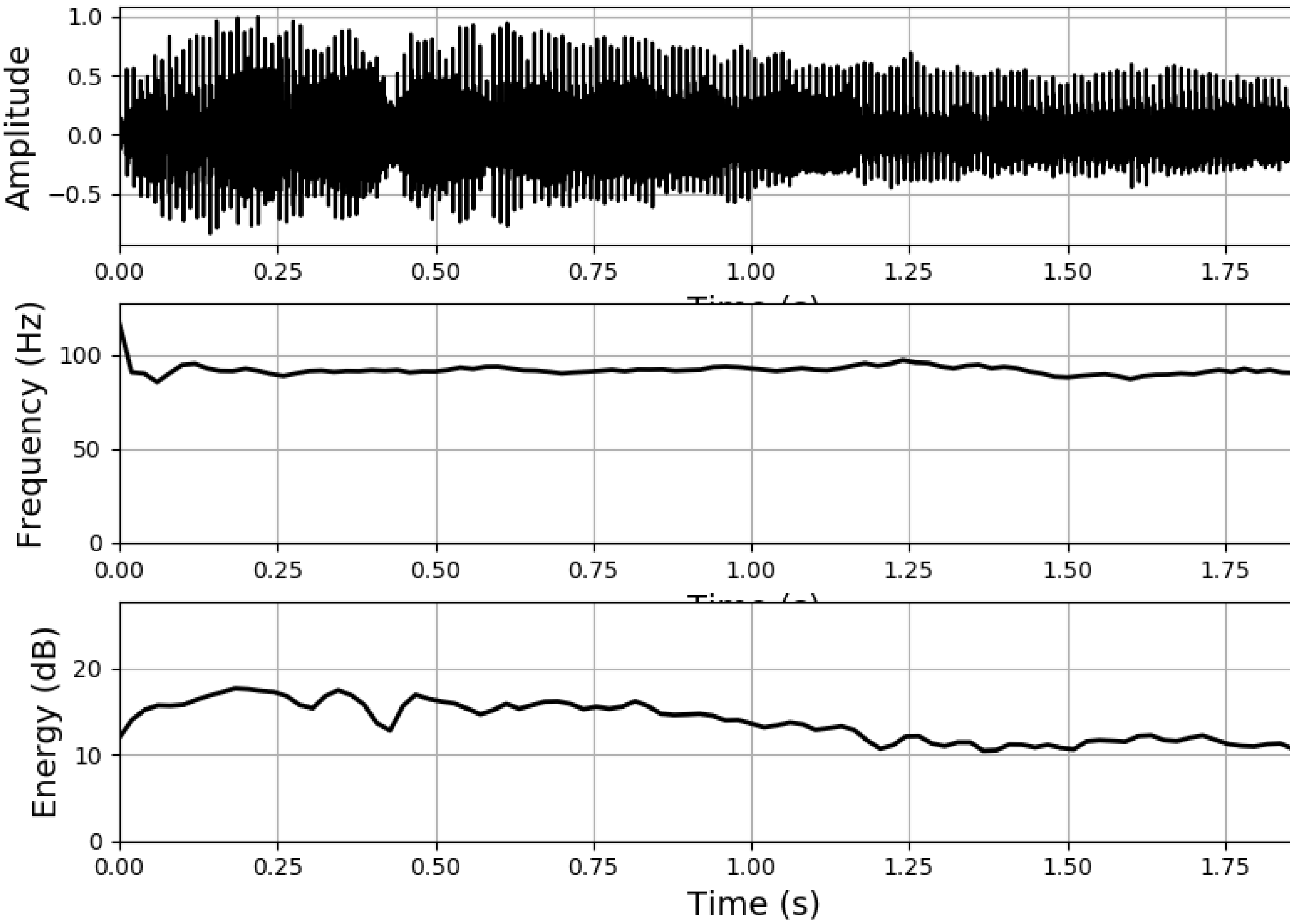
Function Coded by John Kane @ The Phonetics and Speech Lab Trinity College Dublin, August 2012

THE CODE WAS TRANSLATED TO PYTHON AND ADAPTED BY J. C. Vasquez-Correa AT PATTERN RECOGNITION LAB UNIVERSITY OF ERLANGEN NUREMBERGER- GERMANY AND UNIVERSTY OF ANTIOQUIA, COLOMBIA JCAMILO.VASQUEZ@UDEA.EDU.CO https//jcvasquezc.github.io

CHAPTER

TWO

PHONATION FEATURES



Created on Jul 21 2017

@author: J. C. Vasquez-Correa

class `phonation.`**`Phonation`**

Compute phonation features from sustained vowels and continuous speech.

For continuous speech, the features are computed over voiced segments

Seven descriptors are computed:

1. First derivative of the fundamental Frequency
2. Second derivative of the fundamental Frequency
3. Jitter
4. Shimmer
5. Amplitude perturbation quotient
6. Pitch perturbation quotient
7. Logaritmic Energy

Static or dynamic matrices can be computed:

Static matrix is formed with 29 features formed with (seven descriptors) x (4 functionals: mean, std, skewness, kurtosis) + degree of Unvoiced

Dynamic matrix is formed with the seven descriptors computed for frames of 40 ms.

Notes:

1. In dynamic features the first 11 frames of each recording are not considered to be able to stack the APQ and PPQ descriptors with the remaining ones.
2. The fundamental frequency is computed the RAPT algorithm. To use the PRAAT method, change the "self.pitch method" variable in the class constructor.

Script is called as follows

```
>>> python phonation.py <file_or_folder_audio> <file_features> <static (true or␣
↪false)> <plots (true or false)> <format (csv, txt, npy, kaldi, torch)>
```

Examples command line:

```
>>> python phonation.py "../audios/001_a1_PCGITA.wav" "phonationfeaturesAst.txt"
↪"true" "true" "txt"
>>> python phonation.py "../audios/098_u1_PCGITA.wav" "phonationfeaturesUst.csv"
↪"true" "true" "csv"
>>> python phonation.py "../audios/098_u1_PCGITA.wav" "phonationfeaturesUdyn.pt"
↪"false" "true" "torch"
```

```
>>> python phonation.py "../audios/" "phonationfeaturesst.txt" "true" "false" "txt
↪"
>>> python phonation.py "../audios/" "phonationfeaturesst.csv" "true" "false" "csv
↪"
>>> python phonation.py "../audios/" "phonationfeaturesdyn.pt" "false" "false"
↪"torch"
```

Examples directly in Python

```
>>> from disvoice.phonation import Phonation
>>> phonation=Phonation()
>>> file_audio="../audios/001_a1_PCGITA.wav"
>>> features=phonation.extract_features_file(file_audio, static, plots=True, fmt=
↪"numpy")
>>> features2=phonation.extract_features_file(file_audio, static, plots=True, fmt=
↪"dataframe")
>>> features3=phonation.extract_features_file(file_audio, dynamic, plots=True,␣
↪fmt="torch")
```

```
>>> path_audios="../audios/"
>>> features1=phonation.extract_features_path(path_audios, static, plots=False,␣
↪fmt="numpy")
>>> features2=phonation.extract_features_path(path_audios, static, plots=False,␣
↪fmt="torch")
>>> features3=phonation.extract_features_path(path_audios, static, plots=False,␣
↪fmt="dataframe")
```

extract_features_file (*audio*, *static=True*, *plots=False*, *fmt='npy'*, *kaldi_file=''*)
Extract the phonation features from an audio file

Parameters

- **audio** – .wav audio file.
- **static** – whether to compute and return statistic functionals over the feature matrix, or return the feature matrix computed over frames
- **plots** – timeshift to extract the features
- **fmt** – format to return the features (npy, dataframe, torch, kaldi)
- **kaldi_file** – file to store kaldi features, only valid when fmt=="kaldi"

Returns features computed from the audio file.

```
>>> phonation=Phonation()
>>> file_audio="../audios/001_a1_PCGITA.wav"
>>> features1=phonation.extract_features_file(file_audio, static=True,␣
↪plots=True, fmt="npy")
>>> features2=phonation.extract_features_file(file_audio, static=True,␣
↪plots=True, fmt="dataframe")
>>> features3=phonation.extract_features_file(file_audio, static=False,␣
↪plots=True, fmt="torch")
>>> phonation.extract_features_file(file_audio, static=False, plots=False,␣
↪fmt="kaldi", kaldi_file="./test")
```

extract_features_path (*path_audio*, *static=True*, *plots=False*, *fmt='npy'*, *kaldi_file=''*)
Extract the phonation features for audios inside a path

Parameters

- **path_audio** – directory with (.wav) audio files inside, sampled at 16 kHz
- **static** – whether to compute and return statistic functionals over the feature matrix, or return the feature matrix computed over frames
- **plots** – timeshift to extract the features
- **fmt** – format to return the features (npy, dataframe, torch, kaldi)
- **kaldi_file** – file to store kaldifeatures, only valid when fmt=="kaldi"

Returns features computed from the audio file.

```
>>> phonation=Phonation()
>>> path_audio="../audios/"
>>> features1=phonation.extract_features_path(path_audio, static=True,␣
↪plots=False, fmt="npy")
>>> features2=phonation.extract_features_path(path_audio, static=True,␣
↪plots=False, fmt="csv")
```

```
>>> features3=phonation.extract_features_path(path_audio, static=False,␣
↪plots=True, fmt="torch")
>>> phonation.extract_features_path(path_audio, static=False, plots=False,␣
↪fmt="kaldi", kaldi_file="./test.ark")
```

`plot_phon` (*data_audio*, *fs*, *F0*, *logE*)

Plots of the phonation features

Parameters

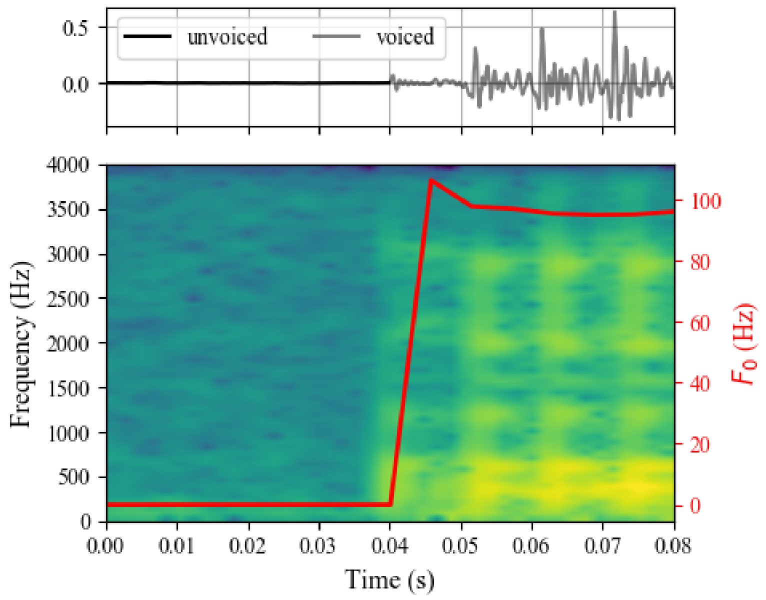
- **`data_audio`** – speech signal.
- **`fs`** – sampling frequency
- **`F0`** – contour of the fundamental frequency
- **`logE`** – contour of the log-energy

Returns plots of the phonation features.

CHAPTER

THREE

ARTICULATION FEATURES



Created on Jul 21 2017

@author: J. C. Vasquez-Correa

class `articulation.`**`Articulation`**

Compute articulation features from continuous speech.

122 descriptors are computed:

1-22. Bark band energies in onset transitions (22 BBE).

23-34. Mel frequency cepstral coefficients in onset transitions (12 MFCC onset)

35-46. First derivative of the MFCCs in onset transitions (12 DMFCC onset)

47-58. Second derivative of the MFCCs in onset transitions (12 DDMFCC onset)

59-80. Bark band energies in offset transitions (22 BBE).

81-92. MFCCC in offset transitions (12 MFCC offset)

93-104. First derivative of the MFCCs in offset transitions (12 DMFCC offset)

105-116. Second derivative of the MFCCs in offset transitions (12 DMFCC offset)

117. First formant Frequency

118. First Derivative of the first formant frequency

119. Second Derivative of the first formant frequency
120. Second formant Frequency
121. First derivative of the Second formant Frequency
122. Second derivative of the Second formant Frequency

Static or dynamic matrices can be computed:

Static matrix is formed with 488 features formed with (122 descriptors) x (4 functionals: mean, std, skewness, kurtosis)

Dynamic matrix are formed with the 58 descriptors (22 BBEs, 12 MFCC, 12DMFCC, 12 DDMFCC) computed for frames of 40 ms with a time-shift of 20 ms in onset transitions.

The first two frames of each recording are not considered for dynamic analysis to be able to stack the derivatives of MFCCs

Notes: 1. The first two frames of each recording are not considered for dynamic analysis to be able to stack the derivatives of MFCCs 2. The fundamental frequency is computed the PRAAT algorithm. To use the RAPT method, change the "self.pitch method" variable in the class constructor.

Script is called as follows

```
>>> python articulation.py <file_or_folder_audio> <file_features> <static (true␣
↪or false)> <plots (true or false)> <format (csv, txt, npy, kaldi, torch)>
```

Examples command line:

```
>>> python articulation.py "../audios/001_ddk1_PCGITA.wav" "articulation_
↪featuresDDKst.txt" "true" "true" txt
>>> python articulation.py "../audios/001_ddk1_PCGITA.wav" "articulation_
↪featuresDDKst.csv" "true" "true" csv
>>> python articulation.py "../audios/001_ddk1_PCGITA.wav" "articulation_
↪featuresDDKst.pt" "true" "true" torch
>>> python articulation.py "../audios/001_ddk1_PCGITA.wav" "articulation_
↪featuresDDKdyn.txt" "false" "true" txt
>>> python articulation.py "../audios/001_ddk1_PCGITA.wav" "articulation_
↪featuresDDKdyn.csv" "false" "true" csv
>>> python articulation.py "../audios/001_ddk1_PCGITA.wav" "articulation_
↪featuresDDKdyn.pt" "false" "true" torch
```

Examples directly in Python

```
>>> articulation=Articulation()
>>> file_audio="../audios/001_ddk1_PCGITA.wav"
>>> features1=articulation.extract_features_file(file_audio, static=True,␣
↪plots=True, fmt="npy")
>>> features2=articulation.extract_features_file(file_audio, static=True,␣
↪plots=True, fmt="dataframe")
>>> features3=articulation.extract_features_file(file_audio, static=False,␣
↪plots=True, fmt="torch")
>>> articulation.extract_features_file(file_audio, static=False, plots=False, fmt=
↪"kaldi", kaldi_file="./test")
```

extract_features_file (*audio, static=True, plots=False, fmt='npy', kaldi_file=''*)

Extract the articulation features from an audio file

Parameters

- **audio** – .wav audio file.

- **static** – whether to compute and return statistic functionals over the feature matrix, or return the feature matrix computed over frames
- **plots** – timeshift to extract the features
- **fmt** – format to return the features (npy, dataframe, torch, kaldi)
- **kaldi_file** – file to store kaldi features, only valid when fmt==”kaldi”

Returns features computed from the audio file.

```
>>> articulation=Articulation()
>>> file_audio="../audios/001_ddk1_PCGITA.wav"
>>> features1=articulation.extract_features_file(file_audio, static=True,␣
↪plots=True, fmt="npy")
>>> features2=articulation.extract_features_file(file_audio, static=True,␣
↪plots=True, fmt="dataframe")
>>> features3=articulation.extract_features_file(file_audio, static=False,␣
↪plots=True, fmt="torch")
>>> articulation.extract_features_file(file_audio, static=False, plots=False,␣
↪fmt="kaldi", kaldi_file="./test")
```

```
>>> path_audio="../audios/"
>>> features1=articulation.extract_features_path(path_audio, static=True,␣
↪plots=False, fmt="npy")
>>> features2=articulation.extract_features_path(path_audio, static=True,␣
↪plots=False, fmt="csv")
>>> features3=articulation.extract_features_path(path_audio, static=False,␣
↪plots=True, fmt="torch")
>>> articulation.extract_features_path(path_audio, static=False, plots=False,␣
↪fmt="kaldi", kaldi_file="./test.ark")
```

extract_features_path(*path_audio, static=True, plots=False, fmt='npy', kaldi_file=''*)
Extract the articulation features for audios inside a path

Parameters

- **path_audio** – directory with (.wav) audio files inside, sampled at 16 kHz
- **static** – whether to compute and return statistic functionals over the feature matrix, or return the feature matrix computed over frames
- **plots** – timeshift to extract the features
- **fmt** – format to return the features (npy, dataframe, torch, kaldi)
- **kaldi_file** – file to store kaldifeatures, only valid when fmt==”kaldi”

Returns features computed from the audio file.

```
>>> articulation=Articulation()
>>> path_audio="../audios/"
>>> features1=articulation.extract_features_path(path_audio, static=True,␣
↪plots=False, fmt="npy")
>>> features2=articulation.extract_features_path(path_audio, static=True,␣
↪plots=False, fmt="csv")
>>> features3=articulation.extract_features_path(path_audio, static=False,␣
↪plots=True, fmt="torch")
>>> articulation.extract_features_path(path_audio, static=False, plots=False,␣
↪fmt="kaldi", kaldi_file="./test.ark")
```

plot_art (*data_audio*, *fs*, *F0*, *F1*, *F2*, *segmentsOn*, *segmentsOff*)
Plots of the articulation features

Parameters

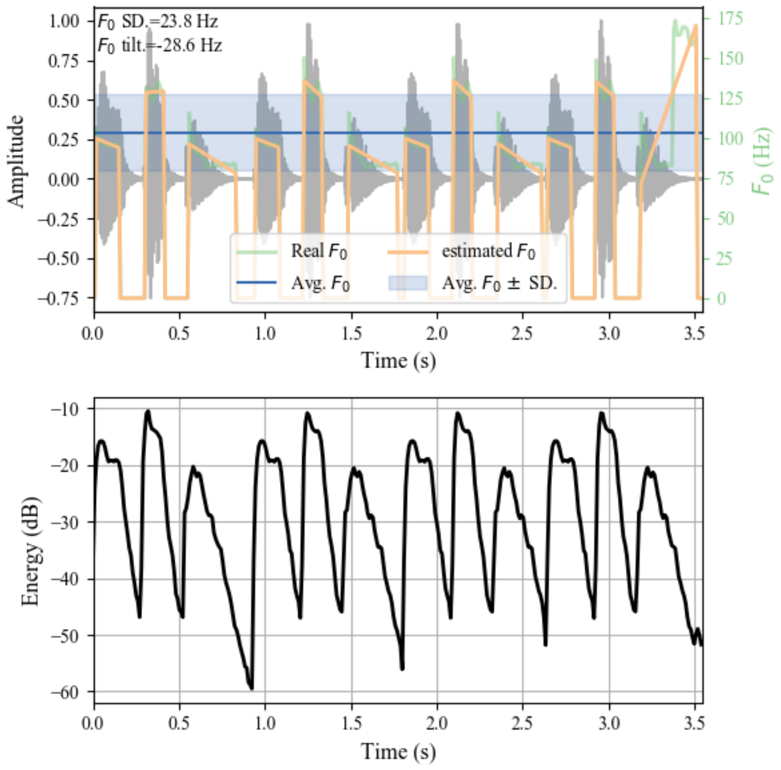
- **data_audio** – speech signal.
- **fs** – sampling frequency
- **F0** – contour of the fundamental frequency
- **F1** – contour of the 1st formant
- **F2** – contour of the 2nd formant
- **segmentsOn** – list with the onset segments
- **segmentsOff** – list with the offset segments

Returns plots of the articulation features.

CHAPTER

FOUR

PROSODY FEATURES



Created on Jul 21 2017, Modified Apr 10 2018.

@author: J. C. Vasquez-Correa, T. Arias-Vergara, J. S. Guerrero

class `prosody.`**`Prosody`**

Compute prosody features from continuous speech based on duration, fundamental frequency and energy. Static or dynamic matrices can be computed: Static matrix is formed with 103 features and include

1-6 F0-contour: Avg., Std., Max., Min., Skewness, Kurtosis

7-12 Tilt of a linear estimation of F0 for each voiced segment: Avg., Std., Max., Min., Skewness, Kurtosis

13-18 MSE of a linear estimation of F0 for each voiced segment: Avg., Std., Max., Min., Skewness, Kurtosis

19-24 F0 on the first voiced segment: Avg., Std., Max., Min., Skewness, Kurtosis

25-30 F0 on the last voiced segment: Avg., Std., Max., Min., Skewness, Kurtosis

31-34 energy-contour for voiced segments: Avg., Std., Skewness, Kurtosis

35-38 Tilt of a linear estimation of energy contour for V segments: Avg., Std., Skewness, Kurtosis

39-42 MSE of a linear estimation of energy contour for V segment: Avg., Std., Skewness, Kurtosis

43-48 energy on the first voiced segment: Avg., Std., Max., Min., Skewness, Kurtosis

49-54 energy on the last voiced segment: Avg., Std., Max., Min., Skewness, Kurtosis

55-58 energy-contour for unvoiced segments: Avg., Std., Skewness, Kurtosis

59-62 Tilt of a linear estimation of energy contour for U segments: Avg., Std., Skewness, Kurtosis

63-66 MSE of a linear estimation of energy contour for U segments: Avg., Std., Skewness, Kurtosis

67-72 energy on the first unvoiced segment: Avg., Std., Max., Min., Skewness, Kurtosis

73-78 energy on the last unvoiced segment: Avg., Std., Max., Min., Skewness, Kurtosis

79 Voiced rate: Number of voiced segments per second

80-85 Duration of Voiced: Avg., Std., Max., Min., Skewness, Kurtosis

86-91 Duration of Unvoiced: Avg., Std., Max., Min., Skewness, Kurtosis

92-97 Duration of Pauses: Avg., Std., Max., Min., Skewness, Kurtosis

98-103 Duration ratios: Pause/(Voiced+Unvoiced), Pause/Unvoiced, Unvoiced/(Voiced+Unvoiced),Voiced/(Voiced+Unvoiced), Voiced/Puase, Unvoiced/Pause

Dynamic matrix is formed with 13 features computed for each voiced segment and contains

1-6. Coefficients of 5-degree Lagrange polynomial to model F0 contour

7-12. Coefficients of 5-degree Lagrange polynomial to model energy contour

13. Duration of the voiced segment

Dynamic prosody features are based on Najim Dehak, “Modeling Prosodic Features With Joint Factor Analysis for Speaker Verification”, 2007

Script is called as follows

```
>>> python prosody.py <file_or_folder_audio> <file_features> <static (true or␣
↪false)> <plots (true or false)> <format (csv, txt, npy, kaldi, torch)>
```

Examples command line:

```
>>> python prosody.py "../audios/001_ddk1_PCGITA.wav" "prosodyfeaturesAst.txt"
↪"true" "true" "txt"
>>> python prosody.py "../audios/001_ddk1_PCGITA.wav" "prosodyfeaturesUst.csv"
↪"true" "true" "csv"
>>> python prosody.py "../audios/001_ddk1_PCGITA.wav" "prosodyfeaturesUdyn.pt"
↪"false" "true" "torch"
```

```
>>> python prosody.py "../audios/" "prosodyfeaturesst.txt" "true" "false" "txt"
>>> python prosody.py "../audios/" "prosodyfeaturesst.csv" "true" "false" "csv"
>>> python prosody.py "../audios/" "prosodyfeaturesdyn.pt" "false" "false" "torch"
>>> python prosody.py "../audios/" "prosodyfeaturesdyn.csv" "false" "false" "csv"
```

Examples directly in Python

```
>>> prosody=Prosody()
>>> file_audio="../audios/001_ddk1_PCGITA.wav"
>>> features1=prosody.extract_features_file(file_audio, static=True, plots=True,␣
↪fmt="npy")
```

```
>>> features2=prosody.extract_features_file(file_audio, static=True, plots=True,␣
↪fmt="dataframe")
>>> features3=prosody.extract_features_file(file_audio, static=False, plots=True,␣
↪fmt="torch")
>>> prosody.extract_features_file(file_audio, static=False, plots=False, fmt=
↪"kaldi", kaldi_file="./test")
```

```
>>> path_audio="../audios/"
>>> features1=prosody.extract_features_path(path_audio, static=True, plots=False,␣
↪fmt="npy")
>>> features2=prosody.extract_features_path(path_audio, static=True, plots=False,␣
↪fmt="csv")
>>> features3=prosody.extract_features_path(path_audio, static=False, plots=True,␣
↪fmt="torch")
>>> prosody.extract_features_path(path_audio, static=False, plots=False, fmt=
↪"kaldi", kaldi_file="./test.ark")
```

extract_features_file (*audio*, *static=True*, *plots=False*, *fmt='npy'*, *kaldi_file=''*)
Extract the prosody features from an audio file

Parameters

- **audio** – .wav audio file.
- **static** – whether to compute and return statistic functionals over the feature matrix, or return the feature matrix computed over frames
- **plots** – timeshift to extract the features
- **fmt** – format to return the features (npy, dataframe, torch, kaldi)
- **kaldi_file** – file to store kaldi features, only valid when fmt=="kaldi"

Returns features computed from the audio file.

```
>>> prosody=Prosody()
>>> file_audio="../audios/001_ddk1_PCGITA.wav"
>>> features1=prosody.extract_features_file(file_audio, static=True,␣
↪plots=True, fmt="npy")
>>> features2=prosody.extract_features_file(file_audio, static=True,␣
↪plots=True, fmt="dataframe")
>>> features3=prosody.extract_features_file(file_audio, static=False,␣
↪plots=True, fmt="torch")
>>> prosody.extract_features_file(file_audio, static=False, plots=False, fmt=
↪"kaldi", kaldi_file="./test")
```

extract_features_path (*path_audio*, *static=True*, *plots=False*, *fmt='npy'*, *kaldi_file=''*)
Extract the prosody features for audios inside a path

Parameters

- **path_audio** – directory with (.wav) audio files inside, sampled at 16 kHz
- **static** – whether to compute and return statistic functionals over the feature matrix, or return the feature matrix computed over frames
- **plots** – timeshift to extract the features
- **fmt** – format to return the features (npy, dataframe, torch, kaldi)
- **kaldi_file** – file to store kaldifeatures, only valid when fmt=="kaldi"

Returns features computed from the audio file.

```
>>> prosody=Prosody()
>>> path_audio="../audios/"
>>> features1=prosody.extract_features_path(path_audio, static=True,␣
↪plots=False, fmt="npy")
>>> features2=prosody.extract_features_path(path_audio, static=True,␣
↪plots=False, fmt="csv")
>>> features3=prosody.extract_features_path(path_audio, static=False,␣
↪plots=True, fmt="torch")
>>> prosody.extract_features_path(path_audio, static=False, plots=False, fmt=
↪"kaldi", kaldi_file="./test.ark")
```

plot_pros (*data_audio*, *fs*, *F0*, *segmentsV*, *segmentsU*, *F0_features*)

Plots of the prosody features

Parameters

- **data_audio** – speech signal.
- **fs** – sampling frequency
- **F0** – contour of the fundamental frequency
- **segmentsV** – list with the voiced segments
- **segmentsU** – list with the unvoiced segments
- **F0_features** – vector with f0-based features

Returns plots of the prosody features.

prosody_dynamic (*audio*)

Extract the dynamic prosody features from an audio file

Parameters **audio** – .wav audio file.

Returns array (N,13) with the prosody features extracted from an audio file. N= number of voiced segments

```
>>> prosody=Prosody()
>>> file_audio="../audios/001_ddk1_PCGITA.wav"
>>> features=prosody.prosody_dynamic(file_audio)
```

prosody_static (*audio*, *plots*)

Extract the static prosody features from an audio file

Parameters

- **audio** – .wav audio file.
- **plots** – timeshift to extract the features

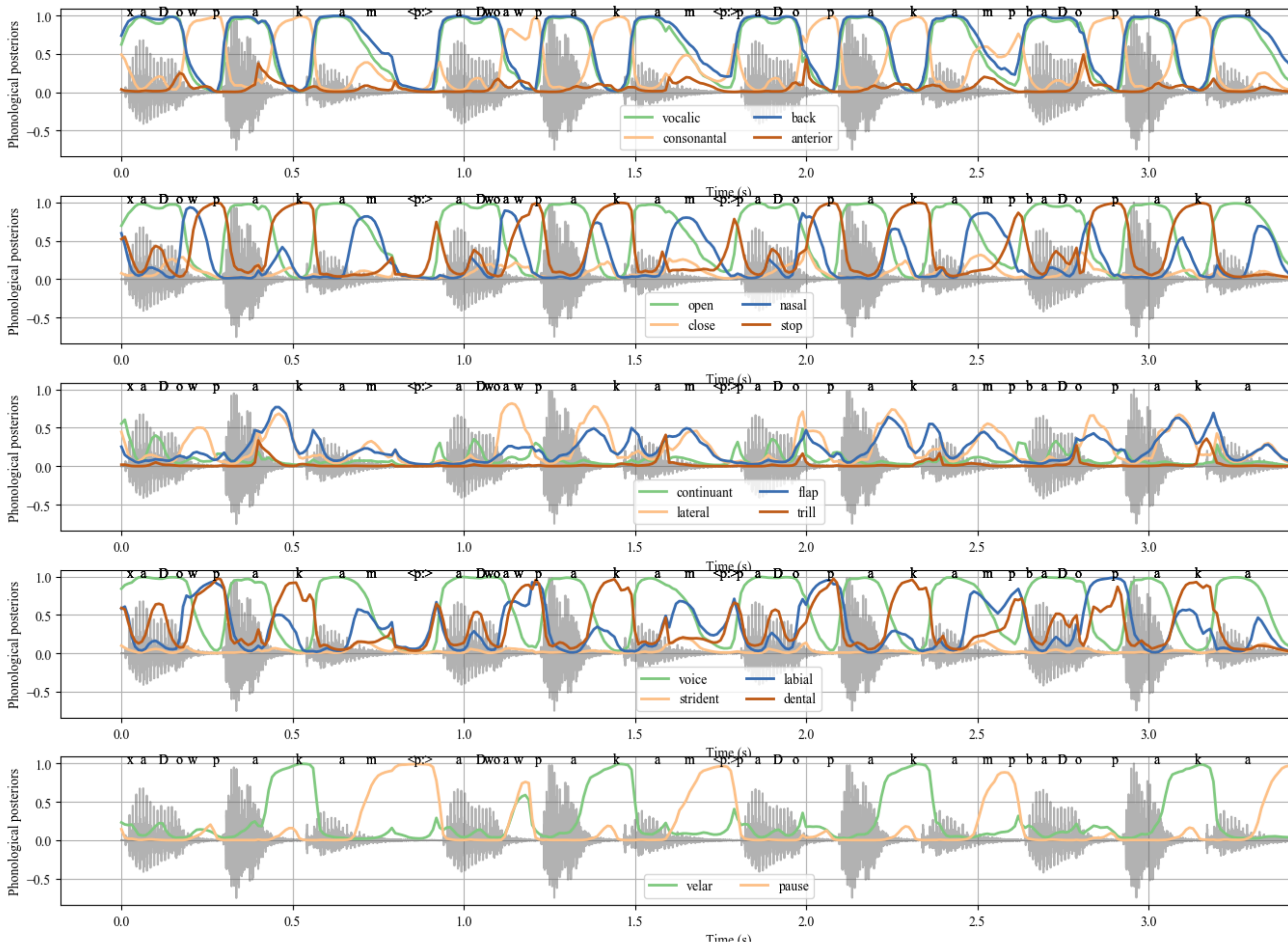
Returns array with the 103 prosody features

```
>>> prosody=Prosody()
>>> file_audio="../audios/001_ddk1_PCGITA.wav"
>>> features=prosody.prosody_static(file_audio, plots=True)
```

CHAPTER

FIVE

PHONOLOGICAL FEATURES



Created on Jun 24 2020

@author: J. C. Vasquez-Correa

class `phonological.`**`Phonological`**

Compute phonological features from continuous speech files.

18 descriptors are computed, bases on 18 different phonological classes from the phonet toolkit https://phonet.readthedocs.io/en/latest/?badge=latest

It computes the phonological log-likelihood ratio features from phonet

Static or dynamic matrices can be computed:

Static matrix is formed with 108 features formed with (18 descriptors) x (6 functionals: mean, std, skewness, kurtosis, max, min)

Dynamic matrix is formed with the 18 descriptors computed for frames of 25 ms with a time-shift of 10 ms.

Script is called as follows

```
>>> python phonological.py <file_or_folder_audio> <file_features> <static (true␣
↪or false)> <plots (true or false)> <format (csv, txt, npy, kaldi, torch)>
```

Examples command line:

```
>>> python phonological.py "../audios/001_ddk1_PCGITA.wav"
↪"phonologicalfeaturesAst.txt" "true" "true" "txt"
>>> python phonological.py "../audios/001_ddk1_PCGITA.wav"
↪"phonologicalfeaturesUst.csv" "true" "true" "csv"
>>> python phonological.py "../audios/001_ddk1_PCGITA.wav"
↪"phonologicalfeaturesUdyn.pt" "false" "true" "torch"
```

```
>>> python phonological.py "../audios/" "phonologicalfeaturesst.txt" "true" "false
↪" "txt"
>>> python phonological.py "../audios/" "phonologicalfeaturesst.csv" "true" "false
↪" "csv"
>>> python phonological.py "../audios/" "phonologicalfeaturesdyn.pt" "false"
↪"false" "torch"
>>> python phonological.py "../audios/" "phonologicalfeaturesdyn.csv" "false"
↪"false" "csv"
```

Examples directly in Python

```
>>> phonological=Phonological()
>>> file_audio="../audios/001_ddk1_PCGITA.wav"
>>> features1=phonological.extract_features_file(file_audio, static=True,␣
↪plots=True, fmt="npy")
>>> features2=phonological.extract_features_file(file_audio, static=True,␣
↪plots=True, fmt="dataframe")
>>> features3=phonological.extract_features_file(file_audio, static=False,␣
↪plots=True, fmt="torch")
>>> phonological.extract_features_file(file_audio, static=False, plots=False, fmt=
↪"kaldi", kaldi_file="./test")
```

`extract_features_file` (*audio*, *static=True*, *plots=False*, *fmt='npy'*, *kaldi_file=''*)
Extract the phonological features from an audio file

Parameters

- **`audio`** – .wav audio file.
- **`static`** – whether to compute and return statistic functionals over the feature matrix, or return the feature matrix computed over frames
- **`plots`** – timeshift to extract the features
- **`fmt`** – format to return the features (npy, dataframe, torch, kaldi)
- **`kaldi_file`** – file to store kaldi features, only valid when fmt=="kaldi"

Returns features computed from the audio file.

```
>>> phonological=Phonological()
>>> file_audio="../audios/001_ddk1_PCGITA.wav"
    >>> features1=phonological.extract_features_file(file_audio, static=True,␣
    ↪plots=True, fmt="npy")
    >>> features2=phonological.extract_features_file(file_audio, static=True,␣
    ↪plots=True, fmt="dataframe")
    >>> features3=phonological.extract_features_file(file_audio, static=False,␣
    ↪plots=True, fmt="torch")
    >>> phonological.extract_features_file(file_audio, static=False, plots=False,␣
    ↪fmt="kaldi", kaldi_file="./test")
```

```
>>> phonological=Phonological()
>>> path_audio="../audios/"
>>> features1=phonological.extract_features_path(path_audio, static=True,␣
↪plots=False, fmt="npy")
>>> features2=phonological.extract_features_path(path_audio, static=True,␣
↪plots=False, fmt="csv")
>>> features3=phonological.extract_features_path(path_audio, static=False,␣
↪plots=True, fmt="torch")
>>> phonological.extract_features_path(path_audio, static=False, plots=False,␣
↪fmt="kaldi", kaldi_file="./test.ark")
```

extract_features_path (*path_audio*, *static=True*, *plots=False*, *fmt='npy'*, *kaldi_file=''*)
Extract the phonological features for audios inside a path

Parameters

- **path_audio** – directory with (.wav) audio files inside, sampled at 16 kHz
- **static** – whether to compute and return statistic functionals over the feature matrix, or return the feature matrix computed over frames
- **plots** – timeshift to extract the features
- **fmt** – format to return the features (npy, dataframe, torch, kaldi)
- **kaldi_file** – file to store kaldifeatures, only valid when fmt=="kaldi"

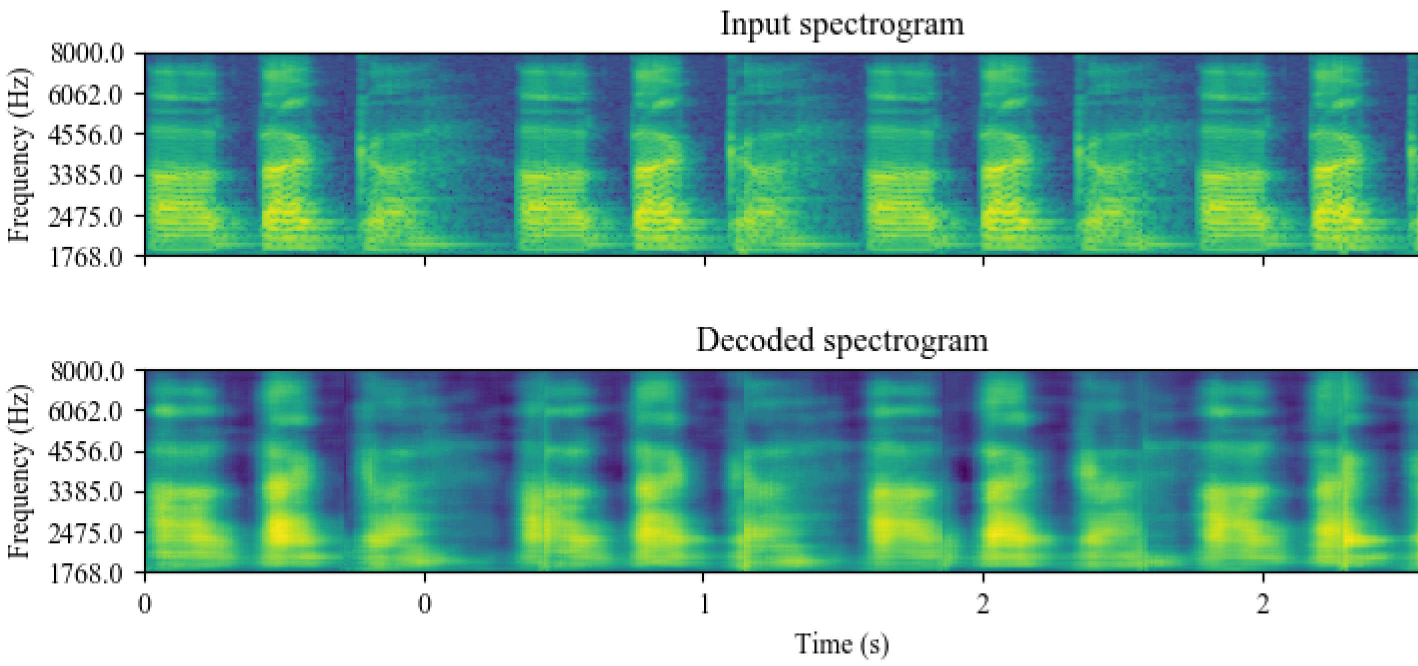
Returns features computed from the audio file.

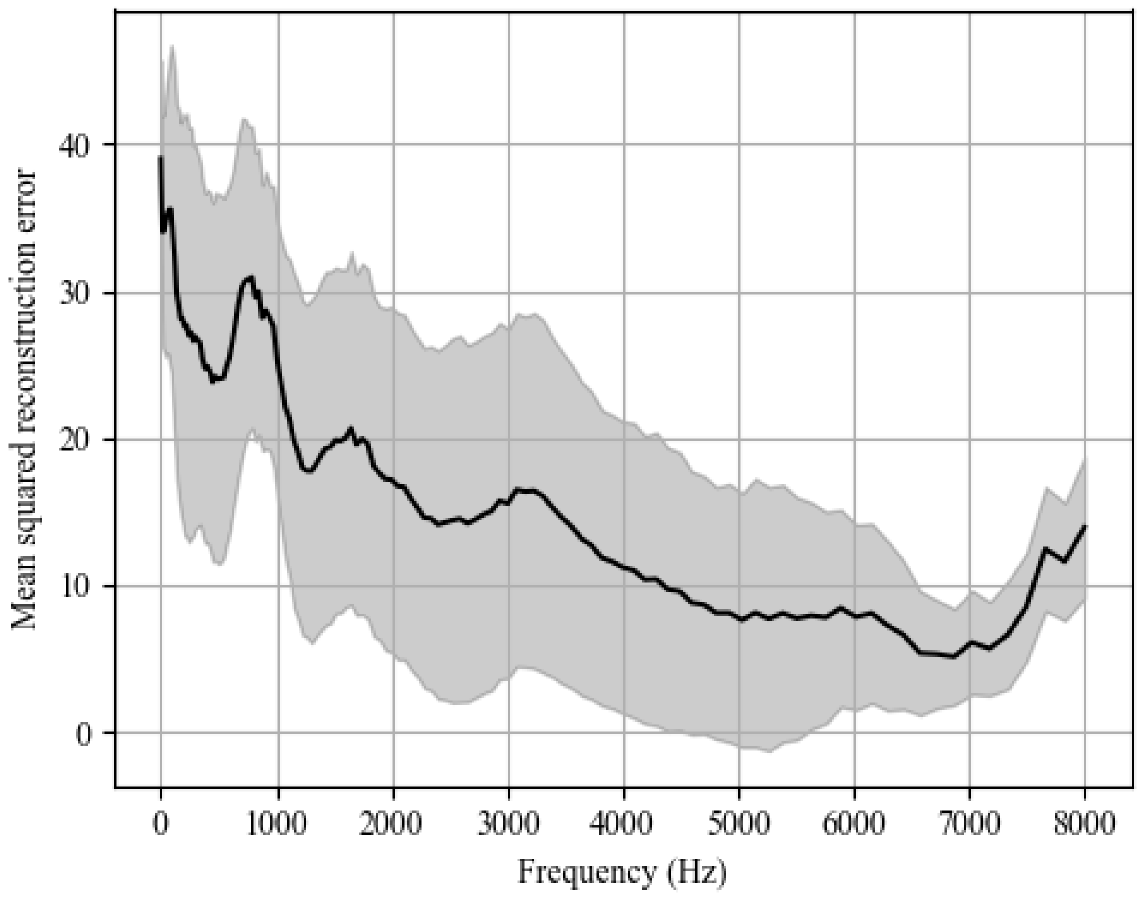
```
>>> phonological=Phonological()
>>> path_audio="../audios/"
>>> features1=phonological.extract_features_path(path_audio, static=True,␣
↪plots=False, fmt="npy")
>>> features2=phonological.extract_features_path(path_audio, static=True,␣
↪plots=False, fmt="csv")
>>> features3=phonological.extract_features_path(path_audio, static=False,␣
↪plots=True, fmt="torch")
>>> phonological.extract_features_path(path_audio, static=False, plots=False,␣
↪fmt="kaldi", kaldi_file="./test.ark")
```

CHAPTER

SIX

REPRESENTATION LEARNING FEATURES





class `replearning.`**`RepLearning`**(*model*)

Feature extraction from speech signals based on representation learning strategies using convolutional and recurrent autoencoders

Two types of features are computed

1. 256 features extracted from the bottleneck layer of the autoencoders
2. 128 features based on the MSE between the decoded and input spectrograms of the autoencoder in different frequency regions

Additionally, static (for all utterance) or dynamic (for each 500 ms speech segments) features can be computed: - The static feature vector is formed with 1024 features and contains (384 descriptors) x (4 functionals: mean, std, skewness, kurtosis) - The dynamic feature matrix is formed with the 384 descriptors computed for speech segments with 500ms length and 250ms time-shift - You can choose between features computed from a convolutional or recurrent autoencoder

Script is called as follows

```
>>> python replearning.py <file_or_folder_audio> <file_features> <static (true or␣
↪false)> <plots (true or false)> <format (csv, txt, npy, kaldi, torch)> <model␣
↪(CAE, RAE)>
```

Examples command line:

```
>>> python replearning.py "../audios/001_ddk1_PCGITA.wav"
↪"replearningfeaturesDDKst.txt" "true" "true" "txt" "CAE"
>>> python replearning.py "../audios/001_ddk1_PCGITA.wav"
↪"replearningfeaturesDDKdyn.pt" "false" "true" "torch" "CAE"
```

```
>>> python replearning.py "../audios/" "replearningfeaturesst.txt" "true" "false"
↪"txt" "CAE"
>>> python replearning.py "../audios/" "replearningfeaturesst.csv" "true" "false"
↪"csv" "CAE"
>>> python replearning.py "../audios/" "replearningfeaturesdyn.pt" "false" "false
↪" "torch" "CAE"
```

Examples directly in Python

```
>>> from replearning import RepLearning
>>> replearning=RepLearning('CAE')
>>> file_audio="../audios/001_a1_PCGITA.wav"
>>> features1=replearning.extract_features_file(file_audio, static=True,␣
↪plots=True, fmt="npy")
>>> features2=replearning.extract_features_file(file_audio, static=True,␣
↪plots=True, fmt="dataframe")
>>> features3=replearning.extract_features_file(file_audio, static=False,␣
↪plots=True, fmt="torch")
>>> replearning.extract_features_file(file_audio, static=False, plots=False, fmt=
↪"kaldi", kaldi_file="./test")
```

extract_features_file(*audio*, *static=True*, *plots=False*, *fmt='npy'*, *kaldi_file=''*)

Extract the representation learning features from an audio file

Parameters

- **audio** – .wav audio file.
- **static** – whether to compute and return statistic functionals over the feature matrix, or return the feature matrix computed over frames
- **plots** – timeshift to extract the features
- **fmt** – format to return the features (npy, dataframe, torch, kaldi)
- **kaldi_file** – file to store kaldi features, only valid when fmt=="kaldi"

Returns features computed from the audio file.

```
>>> replearning=RepLearning('CAE')
>>> file_audio="../audios/001_ddk1_PCGITA.wav"
>>> features1=replearning.extract_features_file(file_audio, static=True,␣
↪plots=True, fmt="npy")
>>> features2=replearning.extract_features_file(file_audio, static=True,␣
↪plots=True, fmt="dataframe")
>>> features3=replearning.extract_features_file(file_audio, static=False,␣
↪plots=True, fmt="torch")
>>> replearning.extract_features_file(file_audio, static=False, plots=False,␣
↪fmt="kaldi", kaldi_file="./test")
```

```
>>> replearning=RepLearning('CAE')
>>> path_audio="../audios/"
>>> features1=replearning.extract_features_path(path_audio, static=True,␣
↪plots=False, fmt="npy")
>>> features2=replearning.extract_features_path(path_audio, static=True,␣
↪plots=False, fmt="csv")
```

```
>>> features3=replearning.extract_features_path(path_audio, static=False,␣
↪plots=True, fmt="torch")
>>> replearning.extract_features_path(path_audio, static=False, plots=False,␣
↪fmt="kaldi", kaldi_file="./test.ark")
```

extract_features_path(*path_audio*, *static=True*, *plots=False*, *fmt='npy'*, *kaldi_file=''*)
Extract the representation learning features for audios inside a path

Parameters

- **path_audio** – directory with (.wav) audio files inside, sampled at 16 kHz
- **static** – whether to compute and return statistic functionals over the feature matrix, or return the feature matrix computed over frames
- **plots** – timeshift to extract the features
- **fmt** – format to return the features (npy, dataframe, torch, kaldi)
- **kaldi_file** – file to store kaldifeatures, only valid when fmt=="kaldi"

Returns features computed from the audio file.

```
>>> replearning=RepLearning('CAE')
>>> path_audio="../audios/"
>>> features1=phonological.replearning(path_audio, static=True, plots=False,␣
↪fmt="npy")
>>> features2=phonological.replearning(path_audio, static=True, plots=False,␣
↪fmt="csv")
>>> features3=phonological.replearning(path_audio, static=False, plots=True,␣
↪fmt="torch")
>>> replearning.extract_features_path(path_audio, static=False, plots=False,␣
↪fmt="kaldi", kaldi_file="./test.ark")
```

Feature extraction from speech signals based on representation learning strategies

class AEspeech.**AEspeech**(*model*, *units*)

compute_bottleneck_features(*wav_file*, *return_numpy=True*)
Compute the the bottleneck features of the autoencoder

Parameters

- **wav_file** – .wav file with a sampling frequency of 16kHz
- **return_numpy** – return the features in a numpy array (True) or a Pytorch tensor (False)

Returns Pytorch tensor (nf, h) or numpy array (nf, h) with the extracted features. nf: number of frames, size of the bottleneck space

compute_dynamic_features(*wav_directory*)
Compute both the bottleneck and the reconstruction error features from the autoencoder for wav files inside a directory

Parameters **wav_directory** – .wav file with a sampling frequency of 16kHz

Returns dictionary with the extracted bottleneck and error features, and with information about which frame coresponds to which wav file in the directory.

compute_global_features(*wav_directory*, *stack_feat=False*)
Compute global features (1 vector per utterance) both for the bottleneck and the reconstruction error features from the autoencoder for wav files inside a directory

Parameters

- **`wav_directory`** – .wav file with a sampling frequency of 16kHz
- **`stack_feat`** – if True, returns also a feature matrix with the stack of the bottleneck and error features

Returns pandas dataframes with the bottleneck and error features.

`compute_rec_error_features`(*wav_file*, *return_numpy=True*)
Compute the reconstruction error features from the autoencoder

Parameters

- **`wav_file`** – .wav file with a sampling frequency of 16kHz
- **`return_numpy`** – return the features in a numpy array (True) or a Pytorch tensor (False)

Returns Pytorch tensor (nf, 128) or numpy array (nf, 128) with the extracted features. nf: number of frames

`compute_rec_spectrogram`(*wav_file*, *return_numpy=True*)
Compute the reconstructed spectrogram from the autoencoder

Parameters

- **`wav_file`** – .wav file with a sampling frequency of 16kHz
- **`return_numpy`** – return the features in a numpy array (True) or a Pytorch tensor (False)

Returns Pytorch tensor (N, C, F, T). N: batch of spectrograms extracted every 500ms, C: number of channels (1), F: number of Mel frequencies (128), T: time steps (126)

`compute_spectrograms`(*wav_file*)
Compute the tensor of Mel-scale spectrograms to be used as input for the autoencoders from a wav file

Parameters **`wav_file`** – .wav file with a sampling frequency of 16kHz

Returns Pytorch tensor (N, C, F, T). N: batch of spectrograms extracted every 500ms, C: number of channels (1), F: number of Mel frequencies (128), T: time steps (126)

`destandard`(*tensor*)
destandardize input tensor from the autoencoders

Parameters **`tensor`** – standardized input tensor for the AEs (N, 128,126)

Returns destandardized tensor for the AEs (N, 128,126)

`plot_spectrograms`(*wav_file*)
Figure of the decoded spectrograms by the AEs

Parameters **`wav_file`** – .wav file with a sampling frequency of 16kHz

`show_spectrograms`(*spectrograms*)
Visualization of the computed tensor of Mel-scale spectrograms to be used as input for the autoencoders from a wav file

Parameters **`spectrograms`** – tensor of spectrograms obtained from `compute_spectrograms(wav-file)`

`standard`(*tensor*)
standardize input tensor for the autoencoders

Parameters **`tensor`** – input tensor for the AEs (N, 128,126)

Returns standardize tensor for the AEs (N, 128,126)

CHAPTER SEVEN

NEED HELP?

If you have trouble with Disvoice, please write to Camilo Vasquez at: juan.vasquez@fau.de

CHAPTER EIGHT

REFERENCES

If you use Disvoice for research purposes, please cite the following papers, depending on the features you use:

8.1 glottal features

[1] Belalcázar-Bolaños, E. A., Orozco-Arroyave, J. R., Vargas-Bonilla, J. F., Haderlein, T., & Nöth, E. (2016, September). Glottal Flow Patterns Analyses for Parkinson's Disease Detection: Acoustic and Nonlinear Approaches. In International Conference on Text, Speech, and Dialogue (pp. 400-407). Springer.

8.2 phonation features

[1] T. Arias-Vergara, J. C. Vásquez-Correa, J. R. Orozco-Arroyave, Parkinson's Disease and Aging: Analysis of Their Effect in Phonation and Articulation of Speech, Cognitive computation, (2017).

[2] Vásquez-Correa, J. C., et al. "Towards an automatic evaluation of the dysarthria level of patients with Parkinson's disease." Journal of communication disorders 76 (2018): 21-36.

8.3 articulation features

[1] Vásquez-Correa, J. C., et al. "Towards an automatic evaluation of the dysarthria level of patients with Parkinson's disease." Journal of communication disorders 76 (2018): 21-36.

[2]. J. R. Orozco-Arroyave, J. C. Vásquez-Correa et al. "NeuroSpeech: An open-source software for Parkinson's speech analysis." Digital Signal Processing (2017).

8.4 prosody features

[1]. N., Dehak, P. Dumouchel, and P. Kenny. "Modeling prosodic features with joint factor analysis for speaker verification." IEEE Transactions on Audio, Speech, and Language Processing 15.7 (2007): 2095-2103.

[2] Vásquez-Correa, J. C., et al. "Towards an automatic evaluation of the dysarthria level of patients with Parkinson's disease." Journal of communication disorders 76 (2018): 21-36.

8.5 phonological features

[1] Vásquez-Correa, J. C., Klumpp, P., Orozco-Arroyave, J. R., & Nöth, E. (2019). Phonet: a Tool Based on Gated Recurrent Neural Networks to Extract Phonological Posteriors from Speech. Proc. Interspeech 2019, 549-553.

8.6 Representation learning features

[1] Vasquez-Correa, J. C., et al. (2020). Parallel Representation Learning for the Classification of Pathological Speech: Studies on Parkinson's Disease and Cleft Lip and Palate. Speech Communication, 122, 56-67.

CHAPTER NINE

INSTALLATION

From the source file:

```
git clone https://github.com/jcvasquezc/disvoice
cd disvoice
bash install.sh
```

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER ELEVEN

HELP

If you have trouble with Disvoice, please write to Camilo Vasquez at: juan.vasquez@fau.de

PYTHON MODULE INDEX

INDEX