pydrobert-speech

Sean Robertson

Aug 18, 2022

CONTENTS:

This pure-python library allows for flexible computation of speech features.

For example, given feature configuration called `fbanks.json`:

```
{
  "name": "stft",
  "bank": "fbank",
  "frame_length_ms": 25,
  "include_energy": true,
  "pad_to_nearest_power_of_two": true,
  "window_function": "hanning",
  "use_power": true
}
```

You can compute triangular, overlapping filters like Kaldi or HTK with the commands

```
import json
from pydrobert.speech import *
# get the feature computer ready
params = json.load(open('fbank.json'))
computer = util.alias_factory_subclass_from_arg(compute.FrameComputer, params)
# assume "signal" is a numpy float array
feats = computer.compute_full(signal)
```

If you plan on using a PyTorch `DataLoader` or Kaldi tables in your ASR pipeline, you can compute all a corpus' features by using the commands `signals-to-torch-feat-dir` (requires *pytorch* package) or `compute-feats-from-kaldi-tables` (requires *pydrobert-kaldi* package).

This package can compute much more than f-banks, with many different permutations. Consult the documentation for a more in-depth discussion of how to use it.

CHAPTER

ONE

DOCUMENTATION

- Latest

CHAPTER

TWO

INSTALLATION

pydrobert-speech is available via both PyPI and Conda.

```
conda install -c sdrobert pydrobert-speech
pip install pydrobert-speech
pip install git+https://github.com/sdrobert/pydrobert-speech # bleeding edge
```

CHAPTER

THREE

LICENSING AND HOW TO CITE

Please see the pydrobert page for more details on how to cite this package.

`util.read_signal` can read NIST SPHERE files. To do so, code was adapted from NIST sph2pipe program and put into `pydrobert.speech._sphere`. License information can be found in `LICENSE_sph2pipe`. Please note that the license only permits the use of their code to decode the "shorten" file type, not encode it.

CHAPTER

FOUR

OVERVIEW

This section provides an overview of how *pydrobert-speech* is organized so that you can get your feature representation just right.

The input to *pydrobert-speech* are (acoustic) signals. The output are features. We call the operator that transforms the signal to some feature representation a computer. Operators that act on the signal and produce a signal, like random dithering, are preprocessors, and operators that act on features and produce features, like unit normalization, are post-processors. The latter two operators exist in the submodules *pydrobert.speech.pre* and *pydrobert.speech.post* and follow the usage pattern

```
>>> y = Op().apply(x)
```

A computer, which can be found in *pydrobert.speech.compute*, will require more complicated initialization. The standard feature representation, which is a 2D time-log-frequency matrix of energy, derives from *pydrobert.speech.compute.LinearFilterBankFrameComputer*. It calculates coefficients over uniform time slices (frames) using a bank of filters. Children of *pydrobert.speech.compute.LinearFilterBankFrameComputer* all have similar representations and all use linear banks of filters, but can be computed in different ways. The classic method of computation is the *pydrobert.speech.compute.ShortTimeFourierTransformFrameComputer*.

Banks of filters are derived from *pydrobert.speech.filters.LinearFilterBank*. Children of the parent class, such as *pydrobert.speech.filters.ComplexGammatoneFilterBank*, will decide on the shape of the filters.

pydrobert.speech.compute.LinearFilterBankFrameComputer instances compute coefficients at uniform intervals in time. However, the distribution over frequencies is decided by the distribution of filter frequency responses from the filter bank, which, in turn, depends on a scaling function. Scaling functions can be found in *pydrobert.speech.scales* such as *pydrobert.speech.scales.MelScaling*. Scaling functions transform the frequency domain into some other real domain. In *that* domain, filter frequency bandwidths are distributed uniformly which, when translated back to the frequency domain, could be quite non-uniform.

In sum, you build a computer by first choosing a scale from *pydrobert.speech.scales*. You then pass that as an argument to a filter bank that you've chosen from *pydrobert.speech.filters*. Finally, you past that as an argument to your computer of choice. For example:

```
>>> from pydrobert.speech import *
>>> scale = scales.MelScaling()
>>> bank = filters.ComplexGammatoneFilterBank(scale)
>>> computer = compute.ShortTimeFourierTransformFrameComputer(bank)
>>> # preprocess the signal
>>> feats = computer.compute_full(signal)
>>> # postprocess the signal
```

This is a bit different from the syntax described in the README. There, we use aliases. Aliases are a simple mechanism for unpacking hierarchies of parameters, such as the hierarchy between these computers, filter banks, and scales. We can streamline the above initialization as

```
>>> computer = compute.ShortTimeFourierTransformFrameComputer(
...     {"name": "tonebank", "scaling_function": "mel"})
```

or even

```
>>> computer = compute.FrameComputer.from_alias("stft",
...     {"name": "tonebank", "scaling_function": "mel"})
```

The dictionaries are merely keyword argument dictionaries with the special key `"name"` or `"alias"` referring to an alias of the subclass you wish to initialize (unless you just pass a string, at which point it's considered the alias with no arguments). Aliases are listed in each subclass' `alias` class member. Besides for brevity, aliases provide a principled way of storing hierarchies on disk via JSON. Thus, it's possible to access most of *pydrobert-speech*'s flexibility from the provided command-line hooks.

Finally, there are some visualization functions in the *pydrobert.speech.vis* module (requires `matplotlib`), some extensions to *pydrobert-kaldi* data iterators in *pydrobert.speech.corpus*.

CHAPTER

FIVE

COMMAND-LINE INTERFACE

5.1 compute-feats-from-kaldi-tables

```
compute-feats-from-kaldi-tables -h
usage: compute-feats-from-kaldi-tables [-h] [-v VERBOSE] [--config CONFIG]
                                       [--print-args PRINT_ARGS]
                                       [--min-duration MIN_DURATION]
                                       [--channel CHANNEL]
                                       [--preprocess PREPROCESS]
                                       [--postprocess POSTPROCESS]
                                       [--seed SEED]
                                       wav_rspecifier feats_wspecifier
                                       computer_config

Store features from a kaldi archive in a kaldi archive

    This command is intended to replace Kaldi's (https://kaldi-asr.org/) series of
    "compute-<something>-feats" scripts in a Kaldi pipeline.


positional arguments:
  wav_rspecifier        Input wave table rspecifier
  feats_wspecifier      Output feature table wspecifier
  computer_config       JSON file or string to configure a
                        'pydrobert.speech.compute.FrameComputer' object to
                        calculate features with

optional arguments:
  -h, --help            show this help message and exit
  -v VERBOSE, --verbose VERBOSE
                        Verbose level (higher->more logging)
  --config CONFIG
  --print-args PRINT_ARGS
  --min-duration MIN_DURATION
                        Min duration of segments to process (in seconds)
  --channel CHANNEL     Channel to draw audio from. Default is to assume mono
  --preprocess PREPROCESS
                        JSON list of configurations for
                        'pydrobert.speech.pre.PreProcessor' objects. Audio
                        will be preprocessed in the same order as the list
```

(continues on next page)

(continued from previous page)

```
  --postprocess POSTPROCESS
                        JSON List of configurations for
                        'pydrobert.speech.post.PostProcessor' objects.
                        Features will be postprocessed in the same order as
                        the list
  --seed SEED           A random seed used for determinism. This affects
                        operations like dithering. If unset, a seed will be
                        generated at the moment
```

5.2 signals-to-torch-feat-dir

```
usage: signals-to-torch-feat-dir [-h] [--channel CHANNEL]
                                 [--preprocess PREPROCESS]
                                 [--postprocess POSTPROCESS]
                                 [--force-as {npz,wav,table,hdf5,pt,soundfile,aiff,ogg,
↪sph,flac,file,npy,kaldi}]
                                 [--seed SEED] [--file-prefix FILE_PREFIX]
                                 [--file-suffix FILE_SUFFIX]
                                 [--num-workers NUM_WORKERS]
                                 map [computer_config] dir

Convert a map of signals to a torch SpectDataSet

    This command serves to process audio signals and convert them into a format that can
    be leveraged by "SpectDataSet" in "pydrobert-pytorch"
    (https://github.com/sdrobert/pydrobert-pytorch). It reads in a text file of
    format

        <utt_id_1> <path_to_signal_1>
        <utt_id_2> <path_to_signal_2>
        ...

    computes features according to passed-in settings, and stores them in the
    target directory as

        dir/
            <file_prefix><utt_id_1><file_suffix>
            <file_prefix><utt_id_2><file_suffix>
            ...

    Each signal is read using the utility "pydrobert.speech.util.read_signal()", which
    is a bit slow, but very robust to different file types (such as wave files, hdf5,
    numpy binaries, or Pytorch binaries). A signal is expected to have shape (C, S),
    where C is some number of channels and S is some number of samples. The
    signal can have shape (S,) if the flag "--channels" is set to "-1".

    Features are output as "torch.FloatTensor" of shape "(T, F)", where "T" is some
    number of frames and "F" is some number of filters.

    No checks are performed to ensure that read signals match the feature computer's
```

(continues on next page)

(continued from previous page)

```
    sampling rate (this info may not even exist for some sources).


positional arguments:
  map                   Path to the file containing (<utterance>, <path>)
                        pairs
  computer_config       JSON file or string to configure a
                        pydrobert.speech.compute.FrameComputer object to
                        calculate features with. If unspecified, the audio
                        (with channels removed) will be stored directly with
                        shape (S, 1), where S is the number of samples
  dir                   Directory to output features to. If the directory does
                        not exist, it will be created

optional arguments:
  -h, --help            show this help message and exit
  --channel CHANNEL     Channel to draw audio from. Default is to assume mono
  --preprocess PREPROCESS
                        JSON list of configurations for
                        'pydrobert.speech.pre.PreProcessor' objects. Audio
                        will be preprocessed in the same order as the list
  --postprocess POSTPROCESS
                        JSON List of configurations for
                        'pydrobert.speech.post.PostProcessor' objects.
                        Features will be postprocessed in the same order as
                        the list
  --force-as {npz,wav,table,hdf5,pt,soundfile,aiff,ogg,sph,flac,file,npy,kaldi}
                        Force the paths in 'map' to be interpreted as a
                        specific type of data. table: kaldi table (key is
                        utterance id); wav: wave file; hdf5: HDF5 archive (key
                        is utterance id); npy: Numpy binary; npz: numpy
                        archive (key is utterance id); pt: PyTorch binary;
                        sph: NIST SPHERE file; kaldi: kaldi object; file:
                        numpy.fromfile binary. soundfile: force soundfile
                        processing.
  --seed SEED           A random seed used for determinism. This affects
                        operations like dithering. If unset, a seed will be
                        generated at the moment
  --file-prefix FILE_PREFIX
                        The file prefix indicating a torch data file
  --file-suffix FILE_SUFFIX
                        The file suffix indicating a torch data file
  --num-workers NUM_WORKERS
                        The number of workers simultaneously computing
                        features. Should not affect determinism when used in
                        tandem with --seed. '0' means all work is done on the
                        main thread
```

CHAPTER

SIX

PYDROBERT.SPEECH API

Speech processing library

6.1 pydrobert.speech.alias

Functionality to do with alias factories

class pydrobert.speech.alias.**AliasedFactory**

Bases: ABC

An abstract interface for initialing concrete subclasses with aliases

aliases = {}

class aliases for *from_alias()*

classmethod from_alias(*alias*, **args*, ***kwargs*)

Factory method for initializing a subclass that goes by an alias

All subclasses of this class have the class attribute *aliases*. This method matches *alias* to an element in some subclass' *aliases* and initializes it. Aliases of this class are included in the search. Alias conflicts are resolved by always trying to initialize the last registered subclass that matches the alias.

Parameters

- **alias** (str) – Alias of the subclass
- ***args** – Positional arguments to initialize the subclass
- ****kwargs** – Keyword arguments to initialize the subclass

Raises

ValueError – Alias can't be found

pydrobert.speech.alias.**alias_factory_subclass_from_arg**(*factory_class*, *arg*)

Boilerplate for getting an instance of an AliasedFactory

Rather than an instance itself, a function could receive the arguments to initialize an *AliasedFactory* with *AliasedFactory.from_alias()*. This function uses the following strategy to try and do so

1. If *arg* is an instance of *factory_class*, return *arg*
2. If *arg* is a str, use it as the alias
3. a. Copy *arg* to a dictionary
   b. Pop the key `'alias'` and treat the rest as keyword arguments
   c. If the key `'alias'` is not found, try `'name'`

This function is intentionally limited in order to work nicely with JSON config files.

6.2 pydrobert.speech.compute

Compute features from speech signals

class pydrobert.speech.compute.**FrameComputer**

Bases: *AliasedFactory*

Construct features from a signal from fixed-length segments

A signal is treated as a (possibly overlapping) time series of frames. Each frame is transformed into a fixed-length vector of coefficients.

Features can be computed one at a time, for example:

```
>>> chunk_size = 2 ** 10
>>> while len(signal):Z
>>>     segment = signal[:chunk_size]
>>>     feats = computer.compute_chunk(segment)
>>>     # do something with feats
>>>     signal = signal[chunk_size:]
>>> feats = computer.finalize()
```

Or all at once (which can be much faster, depending on how the computer is optimized):

```
>>> feats = computer.compute_full(signal)
```

The k-th frame can be roughly localized to the signal offset to about `signal[k * computer.frame_shift]`. The signal's exact region of influence is dictated by the *frame_style* property.

abstract compute_chunk(*chunk*)

Compute some coefficients, given a chunk of audio

Parameters
chunk (ndarray) – A 1D float array of the signal. Should be contiguous and non-overlapping with any previously processed segments in the audio stream

Returns
chunk (numpy.ndarray) – A 2D float array of shape `(num_frames, num_coeffs)`. `num_frames` is nonnegative (possibly 0). Contains some number of feature vectors, ordered in time over axis 0.

compute_full(*signal*)

Compute a full signal's worth of feature coefficients

Parameters
signal (ndarray) – A 1D float array of the entire signal

Returns
spec (numpy.ndarray) – A 2D float array of shape `(num_frames, num_coeffs)`. `num_frames` is nonnegative (possibly 0). Contains some number of feature vectors, ordered in time over axis 0.

Raises
ValueError – If already begin computing frames (`started=True`), and *finalize()* has not been called

abstract finalize()

Conclude processing a stream of audio, processing any stored buffer

Returns

chunk (numpy.ndarray) – A 2D float array of shape (num_frames, num_coeffs). num_frames is either 1 or 0.

abstract property frame_length

Number of samples which dictate a feature vector

Type

int

property frame_length_ms

Number of milliseconds of audio which dictate a feature vector

Type

float

abstract property frame_shift

Number of samples absorbed between successive frame computations

Type

int

property frame_shift_ms

Number of milliseconds between succecssive frame computations

Type

float

abstract property frame_style

Dictates how the signal is split into frames

If 'causal', the k-th frame is computed over the indices signal[k * frame_shift:k * frame_shift + frame_length] (at most). If 'centered', the k-th frame is computed over the indices signal[k * frame_shift - (frame_length + 1) // 2 + 1:k * frame_shift + frame_length // 2 + 1]. Any range beyond the bounds of the signal is generated in an implementation-specific way.

Type

str

abstract property num_coeffs

Number of coefficients returned per frame

Type

int

abstract property sampling_rate

Number of samples in a second of a target recording

Type

float

abstract property started

Whether computations for a signal have started

Becomes True after the first call to *compute_chunk()*. Becomes False after call to *finalize()*

Type

bool

class pydrobert.speech.compute.**LinearFilterBankFrameComputer**(*bank*, *include_energy=False*)

Bases: *FrameComputer*

Frame computers whose features are derived from linear filter banks

Computers based on linear filter banks have a predictable number of coefficients and organization. Like the banks, the features with lower indices correspond to filters with lower bandwidths. *num_coeffs* will be simply `bank.num_filts + int(include_energy)`.

Parameters

- **bank** (Union[*LinearFilterBank*, Mapping, str]) – Each filter in the bank corresponds to a coefficient in a frame vector. Can be a `LinearFilterBank` or something compatible with *pydrobert.speech.alias.alias_factory_subclass_from_arg()*
- **include_energy** (bool) – Whether to include a coefficient based on the energy of the signal within the frame. If `True`, the energy coefficient will be inserted at index 0.

property **bank**

The LinearFilterBank from which features are derived

Type
LinearFilterBank

property **includes_energy**

Whether the first coefficient is an energy coefficient

Type
bool

pydrobert.speech.compute.**SIFrameComputer**

alias of *ShortIntegrationFrameComputer*

pydrobert.speech.compute.**STFTFrameComputer**

alias of *ShortTimeFourierTransformFrameComputer*

class pydrobert.speech.compute.**ShortIntegrationFrameComputer**(*bank*, *frame_shift_ms=10*, *frame_style=None*, *include_energy=False*, *pad_to_nearest_power_of_two=True*, *window_function=None*, *use_power=False*, *use_log=True*)

Bases: *LinearFilterBankFrameComputer*

Compute features by integrating over the filter modulus

Each filter in the bank is convolved with the signal. A pointwise nonlinearity pushes the frequency band towards zero. Most of the energy of the signal can be captured in a short time integration. Though best suited to processing whole utterances at once, short integration is compatable with the frame analogy if the frame is assumed to be the cone of influence of the maximum-length filter.

For computational purposes, each filter's impulse response is clamped to zero outside the support of the largest filter in the bank, making it a finite impulse response filter. This effectively decreases the frequency resolution of the filters which aren't already FIR. For better frequency resolution at the cost of computational time, increase *pydrobert.speech.config.EFFECTIVE_SUPPORT_THRESHOLD*.

Parameters

- **bank** (Union[*LinearFilterBank*, Mapping, str]) – Each filter in the bank corresponds to a coefficient in a frame vector. Can be a `LinearFilterBank` or something compatible with *pydrobert.speech.alias.alias_factory_subclass_from_arg()*

- **frame_shift_ms** (float) – The offset between successive frames, in milliseconds. Also the length of the integration
- **frame_style** (Optional[Literal['causal', 'centered']]) – Defaults to `'centered'` if *bank.is_zero_phase*, `'causal'` otherwise. If `'centered'` each filter of the bank is translated so that its support lies in the center of the frame
- **include_energy** (bool) –
- **pad_to_nearest_power_of_two** (bool) – Pad the DFTs used in computation to a power of two for efficient computation
- **window_function** (*pydrobert.speech.filters.WindowFunction*, dict, or str) – The window used to weigh integration. Can be a `WindowFunction` or something compatible with `pydrobert.speech.alias_factory_subclass_from_arg()`. Defaults to *pydrobert.speech.filters.GammaWindow* when `frame_style` is `'causal'`, otherwise *pydrobert.speech.filters.HannWindow*.
- **use_power** (bool) – Whether the pointwise linearity is the signal's power or magnitude
- **use_log** (bool) – Whether to take the log of the integration

aliases = {'si'}

class pydrobert.speech.compute.**ShortTimeFourierTransformFrameComputer**(*bank*, *frame_length_ms=None*, *frame_shift_ms=10*, *frame_style=None*, *include_energy=False*, *pad_to_nearest_power_of_two=True*, *window_function=None*, *use_log=True*, *use_power=False*, *kaldi_shift=False*)

Bases: *LinearFilterBankFrameComputer*

Compute features of a signal by integrating STFTs

Computations are per frame and as follows:

1. The current frame is multiplied with some window (rectangular, Hamming, Hanning, etc)
2. A DFT is performed on the result
3. For each filter in the provided input bank:
   a. Multiply the result of 2. with the frequency response of the filter
   b. Sum either the pointwise square or absolute value of elements in the buffer from 3a.
   c. Optionally take the log of the sum

Warning: This behaviour differs from that of [povey2011] or [young] in three ways. First, the sum (3b) comes after the filtering (3a), which changes the result in the squared case. Second, the sum is over the full power spectrum, rather than just between 0 and the Nyquist. This doubles the value at the end of 3c. if a real filter is used. Third, frame boundaries are calculated diffferently.

Parameters

- **bank** (Union[*LinearFilterBank*, Mapping, str]) – Each filter in the bank corresponds to a coefficient in a frame vector. Can be a LinearFilterBank or something compatible with *pydrobert.speech.alias.alias_factory_subclass_from_arg()*
- **frame_length_ms** (Optional[float]) – The length of a frame, in milliseconds. Defaults to the length of the largest filter in the bank
- **frame_shift_ms** (float, *optional*) – The offset between successive frames, in milliseconds
- **frame_style** (Optional[Literal['causal', 'centered']]) – Defaults to 'centered' if bank.is_zero_phase, 'causal' otherwise.
- **include_energy** (bool) –
- **pad_to_nearest_power_of_two** (bool) – Whether the DFT should be a padded to a power of two for computational efficiency
- **window_function** (Union[*WindowFunction*, Mapping, str, None]) – The window used in step 1. Can be a WindowFunction or something compatible with pydrobert.speech.alias_factory_subclass_from_arg(). Defaults to *pydrobert.speech.filters.GammaWindow* when *frame_style* is 'causal', otherwise *pydrobert.speech.filters.HannWindow*.
- **use_log** (bool) – Whether to take the log of the sum from 3b.
- **use_power** (bool) – Whether to sum the power spectrum or the magnitude spectrum
- **kaldi_shift** (bool) – If True, the k-th frame will be computed using the signal between signal[k - frame_length // 2 + frame_shift // 2:k + (frame_length + 1) // 2 + frame_shift // 2]. These are the frame bounds for Kaldi [povey2011].

aliases = {'stft'}

pydrobert.speech.compute.**frame_by_frame_calculation**(*computer*, *signal*, *chunk_size=1024*)

Compute feature representation of entire signal iteratively

This function constructs a feature matrix of a signal through successive calls to computer.compute_chunk. Its return value should be identical to that of calling computer.compute_full(signal), but is possibly much slower. computer.compute_full should be favoured.

Parameters

- **computer** (*FrameComputer*) –
- **signal** (ndarray) – A 1D float array of the entire signal
- **chunk_size** (int) – The length of the signal buffer to process at a given time

Returns

spec (numpy.ndarray) – A 2D float array of shape (num_frames, num_coeffs). num_frames is nonnegative (possibly 0). Contains some number of feature vectors, ordered in time over axis 0.

Raises

ValueError – If already begin computing frames (computer.started == True)

6.3 pydrobert.speech.config

Package constants used throughout pydrobert.speech

pydrobert.speech.config.**EFFECTIVE_SUPPORT_THRESHOLD = 0.0005**

Value considered roughly zero for support computations

No function is compactly supported in both the time and Fourier domains, but large regions of either domain can be very close to zero. This value serves as a threshold for zero. The higher it is, the more accurate computations will be, but the longer they will take

Type
float

pydrobert.speech.config.**LOG_FLOOR_VALUE = 1e-05**

Value used as floor when taking log in computations

Type
float

pydrobert.speech.config.**SOUNDFILE_SUPPORTED_FILE_TYPES**

A list of the types of files SoundFile will be responsible for reading

If `soundfile` can be imported, it's the intersection of `soundfile.available_formats()` with the set "wav", "ogg", "flac", and "aiff".

See also:

pydrobert.speech.util.read_signal
Where this flag is used

Type
set

pydrobert.speech.config.**USE_FFTPACK**

Whether to use `scipy.fftpack`

The scipy implementation of the FFT can be much faster than the numpy one. This is set automatically to `True` if `scipy.fftpack` can be imported. It can be set to `False` to use the numpy implementation.

Type
bool

6.4 pydrobert.speech.corpus

Submodule for corpus iterators

pydrobert.speech.corpus.**post_process_wrapper**(*cls*)

Wrap a pydrobert-kaldi Data object for post-processing

This function returns a class that wraps the *cls* class, performing some post-processing after batching

6.5 pydrobert.speech.filters

Filters and filter banks

class pydrobert.speech.filters.**BartlettWindow**

Bases: *WindowFunction*

A unit-normalized triangular window

See also:

numpy.bartlett

aliases = {'bartlett', 'tri', 'triangular'}

class pydrobert.speech.filters.**BlackmanWindow**

Bases: *WindowFunction*

A unit-normalized Blackman window

See also:

numpy.blackman

aliases = {'black', 'blackman'}

class pydrobert.speech.filters.**ComplexGammatoneFilterBank**(*scaling_function*, *num_filts=40*, *high_hz=None*, *low_hz=20.0*, *sampling_rate=16000*, *order=4*, *max_centered=False*, *scale_l2_norm=False*, *erb=False*)

Bases: *LinearFilterBank*

Gammatone filters with complex carriers

A complex gammatone filter [flanagan1960] [aertsen1981] can be defined as

$$h(t) = ct^{n-1}e^{-\alpha t + i\xi t}u(t)$$

in the time domain, where α is the bandwidth parameter, ξ is the carrier frequency, n is the order of the function, $u(t)$ is the step function, and c is a normalization constant. In the frequency domain, the filter is defined as

$$H(\omega) = \frac{c(n-1)!}{(\alpha + i(\omega - \xi))^n}$$

For large ξ, the complex gammatone is approximately analytic.

scaling_function is used to split up the frequencies between *high_hz* and *low_hz* into a series of filters. Every subsequent filter's width is scaled such that, if the filters are all of the same height, the intersection with the precedent filter's response matches the filter's Equivalent Rectangular Bandwidth (`erb == True`) or its 3dB bandwidths (`erb == False`). The ERB is the width of a rectangular filter with the same height as the filter's maximum frequency response that has the same L^2 norm.

Parameters

- **scaling_function** (Union[*ScalingFunction*, Mapping, str]) – Dictates the layout of filters in the Fourier domain. Can be a `ScalingFunction` or something compatible with *pydrobert.speech.alias.alias_factory_subclass_from_arg()*
- **num_filts** (int) – The number of filters in the bank

- **high_hz** (Optional[float]) – The topmost edge of the filter frequencies. The default is the Nyquist for *sampling_rate*.
- **low_hz** (float) – The bottommost edge of the filter frequences.
- **sampling_rate** (float, *optional*) – The sampling rate (cycles/sec) of the target recordings
- **order** (int) – The n parameter in the Gammatone. Should be positive. Larger orders will make the gammatone more symmetrical.
- **max_centered** (bool) – While normally causal, setting *max_centered* to true will shift all filters in the bank such that the maximum absolute value in time is centered at sample 0.
- **scale_l2_norm** (bool) – Whether to scale the l2 norm of each filter to 1. Otherwise the frequency response of each filter will max out at an absolute value of 1.
- **erb** (bool) –

See also:

pydrobert.speech.config.EFFECTIVE_SUPPORT_THRESHOLD
The absolute value below which counts as zero

aliases = {'gammatone', 'tonebank'}

property centers_hz

The point of maximum gain in each filter's frequency response, in Hz

This property gives the so-called "center frequencies" - the point of maximum gain - of each filter.

Type
int

class pydrobert.speech.filters.**GaborFilterBank**(*scaling_function*, *num_filts=40*, *high_hz=None*, *low_hz=20.0*, *sampling_rate=16000*, *scale_l2_norm=False*, *erb=False*)

Bases: *LinearFilterBank*

Gabor filters with ERBs between points from a scale

Gabor filters are complex, mostly analytic filters that have a Gaussian envelope in both the time and frequency domains. They are defined as

$$f(t) = C\sigma^{-1/2}\pi^{-1/4}e^{\frac{-t^2}{2\sigma^2}+i\xi t}$$

in the time domain and

$$\widehat{f}(\omega) = C\sqrt{2\sigma}\pi^{1/4}e^{\frac{-\sigma^2(\xi-\omega)^2}{2}}$$

in the frequency domain. Though Gaussians never truly reach 0, in either domain, they are effectively compactly supported. Gabor filters are optimal with respect to their time-bandwidth product.

scaling_function is used to split up the frequencies between *high_hz* and *low_hz* into a series of filters. Every subsequent filter's width is scaled such that, if the filters are all of the same height, the intersection with the precedent filter's response matches the filter's Equivalent Rectangular Bandwidth (`erb == True`) or its 3dB bandwidths (`erb == False`). The ERB is the width of a rectangular filter with the same height as the filter's maximum frequency response that has the same L^2 norm.

Parameters

- **scaling_function** (Union[*ScalingFunction*, Mapping, str]) – Dictates the layout of filters in the Fourier domain. Can be a ScalingFunction or something compatible with *pydrobert.speech.alias.alias_factory_subclass_from_arg()*
- **num_filts** (int) – The number of filters in the bank
- **high_hz** (Optional[float]) – The topmost edge of the filter frequencies. The default is the Nyquist for *sampling_rate*.
- **low_hz** (float) – The bottommost edge of the filter frequences.
- **sampling_rate** (float) – The sampling rate (cycles/sec) of the target recordings
- **scale_l2_norm** (bool) – Whether to scale the l2 norm of each filter to 1. Otherwise the frequency response of each filter will max out at an absolute value of 1.
- **erb** (bool) –

See also:

pydrobert.speech.config.EFFECTIVE_SUPPORT_THRESHOLD
The absolute value below which counts as zero

aliases = {'gabor'}

property centers_hz

The point of maximum gain in each filter's frequency response, in Hz

This property gives the so-called "center frequencies" - the point of maximum gain - of each filter.

Type
tuple

class pydrobert.speech.filters.**GammaWindow**(*order=4*, *peak=0.75*)

Bases: *WindowFunction*

A lowpass filter based on the Gamma function

A Gamma function is defined as:

$$p(t; \alpha, n) = t^{n-1} e^{-\alpha t} u(t)$$

Where n is the order of the function, α controls the bandwidth of the filter, and u is the step function.

This function returns a window based off a reflected Gamma function. α is chosen such that the maximum value of the window aligns with *peak*. The window is clipped to the width. For reasonable values of *peak* (i.e. in the last quarter of samples), the majority of the support should lie in this interval anyways.

Parameters

- **order** (int) –
- **peak** (float) – `peak * width`, where `width` is the length of the window in samples, is where the approximate maximal value of the window lies

aliases = {'gamma'}

order

peak

class pydrobert.speech.filters.**HammingWindow**

Bases: *WindowFunction*

A unit-normalized Hamming window

See also:

numpy.hamming

aliases = {'hamming'}

class pydrobert.speech.filters.**HannWindow**

Bases: *WindowFunction*

A unit-normalized Hann window

See also:

numpy.hanning

aliases = {'hann', 'hanning'}

class pydrobert.speech.filters.**LinearFilterBank**

Bases: *AliasedFactory*

A collection of linear, time invariant filters

A *LinearFilterBank* instance is expected to provide factory methods for instantiating a fixed number of LTI filters in either the time or frequency domain. Filters should be organized lowest frequency first.

abstract get_frequency_response(*filt_idx*, *width*, *half=False*)

Construct filter frequency response in a fixed-width buffer

Construct the 2pi-periodized filter in the frequency domain. Zero-phase filters are returned as 8-byte float arrays. Otherwise, they will be 16-byte complex floats.

Parameters

- **filt_idx** (int) – The index of the filter to generate. Less than *num_filts*
- **width** (int) – The length of the DFT to output
- **half** (bool) – Whether to return only the DFT bins between [0,pi]

Returns

fr (numpy.ndarray) – If *half* is False, returns a 1D float64 or complex128 numpy array of length *width*. If *half* is True and *width* is even, the returned array is of length width // 2 + 1. If *width* is odd, the returned array is of length (width + 1) // 2.

abstract get_impulse_response(*filt_idx*, *width*)

Construct filter impulse response in a fixed-width buffer

Construct the filter in the time domain.

Parameters

- **filt_idx** (int) – The index of the filter to generate. Less than *num_filts*
- **width** (int) – The length of the buffer, in samples. If less than the support of the filter, the filter will alias.

Returns

ir (numpy.ndarray) – 1D float64 or complex128 numpy array of length *width*

abstract get_truncated_response(*filt_idx*, *width*)

Get nonzero region of filter frequency response

Many filters will be compactly supported in frequency (or approximately so). This method generates a tuple *(bin_idx, buf)* of the nonzero region.

In the case of a complex filter, `bin_idx + len(buf)` may be greater than *width*; the filter wraps around in this case. The full frequency response can be calculated from the truncated response by:

```
>>> bin_idx, trnc = bank.get_truncated_response(filt_idx, width)
>>> full = numpy.zeros(width, dtype=trnc.dtype)
>>> wrap = min(bin_idx + len(trnc), width) - bin_idx
>>> full[bin_idx:bin_idx + wrap] = trnc[:wrap]
>>> full[:len(trnc) - wrap] = tnc[wrap:]
```

In the case of a real filter, only the nonzero region between `[0, pi]` (half-spectrum) is returned. No wrapping can occur since it would inevitably interfere with itself due to conjugate symmetry. The half-spectrum can easily be recovered by:

```
>>> half_width = (width + width % 2) // 2 + 1 - width % 2
>>> half = numpy.zeros(half_width, dtype=trnc.dtype)
>>> half[bin_idx:bin_idx + len(trnc)] = trnc
```

And the full spectrum by:

```
>>> full[bin_idx:bin_idx + len(trnc)] = trnc
>>> full[width - bin_idx - len(trnc) + 1:width - bin_idx + 1] = \
...     trnc[:None if bin_idx else 0:-1].conj()
```

(the embedded if-statement is necessary when bin_idx is 0, as the full fft excludes its symmetric bin)

Parameters

- **filt_idx** (int) – The index of the filter to generate. Less than *num_filts*
- **width** (int) – The length of the DFT to output

Returns

tfr (tuple of int, array)

abstract property is_analytic

Whether the filters are (approximately) analytic

An analytic signal has no negative frequency components. A real signal cannot be analytic.

Type

bool

abstract property is_real

Whether the filters are real or complex

Type

bool

abstract property is_zero_phase

Whether the filters are zero phase or not

Zero phase filters are even functions with no imaginary part in the fourier domain. Their impulse responses center around 0.

Type
bool

abstract property num_filts

Number of filters in the bank

Type
int

abstract property sampling_rate

Number of samples in a second of a target recording

Type
float

abstract property supports

Boundaries of effective support of filter impulse resps, in samples

Returns a tuple of length *num_filts* containing pairs of integers of the first and last (effectively) nonzero samples.

The boundaries need not be tight, i.e. the region inside the boundaries could be zero. It is more important to guarantee that the region outside the boundaries is approximately zero.

If a filter is instantiated using a buffer that is unable to fully contain the supported region, samples will wrap around the boundaries of the buffer.

Noncausal filters will have start indices less than 0. These samples will wrap to the end of the filter buffer when the filter is instantiated.

Type
tuple

abstract property supports_hz

Boundaries of effective support of filter freq responses, in Hz.

Returns a tuple of length *num_filts* containing pairs of floats of the low and high frequencies. Frequencies outside the span have a response of approximately (with magnitude up to `pydrobert.speech.EFFECTIVE_SUPPORT_SIGNAL`) zero.

The boundaries need not be tight, i.e. the region inside the boundaries could be zero. It is more important to guarantee that the region outside the boundaries is approximately zero.

The boundaries ignore the Hermitian symmetry of the filter if it is real. Bounds of `(10, 20)` for a real filter imply that the region `(-20, -10)` could also be nonzero.

The user is responsible for adjusting the for the periodicity induced by sampling. For example, if the boundaries are `(-5, 10)` and the filter is sampled at 15Hz, then all bins of an associated DFT could be nonzero.

Type
tuple

property supports_ms

Boundaries of effective support of filter impulse resps, in ms

Type
tuple

class pydrobert.speech.filters.**TriangularOverlappingFilterBank**(*scaling_function*, *num_filts=40*, *high_hz=None*, *low_hz=20.0*, *sampling_rate=16000*, *analytic=False*)

Bases: *LinearFilterBank*

Triangular frequency response whose vertices are along the scale

The vertices of the filters are sampled uniformly along the passed scale. If the scale is nonlinear, the triangles will be asymmetrical. This is closely related to, but not identical to, the filters described in [povey2011] and [young].

Parameters

- **scaling_function** (Union[*ScalingFunction*, Mapping, str]) – Dictates the layout of filters in the Fourier domain. Can be a `ScalingFunction` or something compatible with *pydrobert.speech.alias.alias_factory_subclass_from_arg()*
- **num_filts** (int) – The number of filters in the bank
- **high_hz** (Optional[float]) – The topmost edge of the filter frequencies. The default is the Nyquist for *sampling_rate*.
- **low_hz** (float) – The bottommost edge of the filter frequences.
- **sampling_rate** (float) – The sampling rate (cycles/sec) of the target recordings
- **analytic** (bool) – Whether to use an analytic form of the bank. The analytic form is easily derived from the real form in [povey2011] and [young]. Since the filter is compactly supported in frequency, the analytic form is simply the suppression of the `[-pi, 0)` frequencies

Raises

ValueError – If *high_hz* is above the Nyquist, or *low_hz* is below `0`, or `high_hz <= low_hz`

aliases = {'tri', 'triangular'}

property centers_hz

The point of maximum gain in each filter's frequency response, in Hz

This property gives the so-called "center frequencies" - the point of maximum gain - of each filter.

class pydrobert.speech.filters.**WindowFunction**

Bases: *AliasedFactory*

A real linear filter, usually lowpass

abstract get_impulse_response(*width*)

Write the filter into a numpy array of fixed width

Parameters

width (int) – The length of the window in samples

Returns

ir (numpy.ndarray) – A 1D vector of length *width*

6.6 pydrobert.speech.post

Classes for post-processing feature matrices

pydrobert.speech.post.**CMVN**

alias of *Standardize*

class pydrobert.speech.post.**Deltas**(*num_deltas*, *target_axis=-1*, *concatenate=True*, *context_window=2*, *pad_mode='edge'*, ***kwargs*)

Bases: *PostProcessor*

Calculate feature deltas (weighted rolling averages)

Deltas are calculated by correlating the feature tensor with a 1D delta filter by enumerating over all but one axis (the "time axis" equivalent).

Intermediate values are calculated with 64-bit floats, then cast back to the input data type.

Deltas will increase the size of the feature tensor when *num_deltas* is positive and passed features are non-empty.

If *concatenate* is `True`, *target_axis* specifies the axis along which new deltas are appended. For example,

```
>>> deltas = Deltas(num_deltas=2, concatenate=True, target_axis=1)
>>> features_shape = list(features.shape)
>>> features_shape[1] *= 3
>>> assert deltas.apply(features).shape == tuple(features_shape)
```

If *concatenate* is `False`, *target_axis* dictates the location of a new axis in the resulting feature tensor that will index the deltas (0 for the original features, 1 for deltas, 2 for double deltas, etc.). For example:

```
>>> deltas = Deltas(num_deltas=2, concatenate=False, target_axis=1)
>>> features_shape = list(features.shape)
>>> features_shape.insert(1, 3)
>>> assert deltas.apply(features).shape == tuple(features_shape)
```

Deltas act as simple low-pass filters. Flipping the direction of the real filter to turn the delta operation into a simple convolution, the first order delta is defined as

$$f(t) = \begin{cases} \frac{-t}{Z} & -W \leq t \leq W \\ 0 & \text{else} \end{cases}$$

where

$$Z = \sum_t f(t)^2$$

for some $W \geq 1$. Its Fourier Transform is

$$F(\omega) = \frac{-2i}{Z\omega^2} \left(W\omega \cos W\omega - \sin W\omega\right)$$

Note that it is completely imaginary. For $W \geq 2$, F is bound below $\frac{i}{\omega}$. Hence, F exhibits low-pass characteristics. Second order deltas are generating by convolving $f(-t)$ with itself, third order is an additional $f(-t)$, etc. By the convolution theorem, higher order deltas have Fourier responses that become tighter around $F(0)$ (more lowpass).

Parameters

- **num_deltas** (int) –
- **target_axis** (int) –
- **concatenate** (bool) –
- **context_window** (int) – The length of the filter to either side of the window. Positive.
- **pad_mode** (Union[str, Callable]) – How to pad the input sequence when correlating
- ****kwargs** – Additional keyword arguments to be passed to numpy.pad()

aliases = {'deltas'}

concatenate

num_deltas

class pydrobert.speech.post.**PostProcessor**

Bases: *AliasedFactory*

A container for post-processing features with a transform

abstract apply(*features*, *axis=-1*, *in_place=False*)

Applies the transformation to a feature tensor

Consult the class documentation for more details on what the transformation is.

Parameters

- **features** (ndarray) –
- **axis** (int) – The axis of *features* to apply the transformation along
- **in_place** (bool) – Whether it is okay to modify features (True) or whether a copy should be made (False)

Returns

out (numpy.ndarray) – The transformed features

class pydrobert.speech.post.**Stack**(*num_vectors*, *time_axis=0*, *pad_mode=None*, ***kwargs*)

Bases: *PostProcessor*

Stack contiguous feature vectors together

Parameters

- **num_vectors** (int) – The number of subsequent feature vectors in time to be stacked.
- **time_axis** (int) – The axis along which subsequent feature vectors are drawn.
- **pad_mode** (Union[str, Callable, None]) – Specified how the axis in time will be padded on the right in order to be divisible by *num_vectors*. Additional keyword arguments will be passed to numpy.pad(). If unspecified, frames will instead be discarded in order to be divisible by *num_vectors*.

aliases = {'stack'}

num_vectors

time_axis

class pydrobert.speech.post.**Standardize**(*rfilename=None*, *norm_var=True*, ***kwargs*)

Bases: *PostProcessor*

Standardize each feature coefficient

Though the exact behaviour of an instance varies according to below, the "goal" of this transformation is such that every feature coefficient on the chosen axis has mean 0 and variance 1 (if *norm_var* is `True`) over the other axes. Features are assumed to be real; the return data type after `apply()` is always a 64-bit float.

If *rfilename* is not specified or the associated file is empty, coefficients are standardized locally (within the target tensor). If *rfilename* is specified, feature coefficients are standardized according to the sufficient statistics collected in the file. The latter implementation is based off [povey2011]. The statistics will be loaded with *pydrobert.speech.util.read_signal()*.

Parameters

- **rfilename** (Optional[str]) –
- **norm_var** (bool) –
- ****kwargs** –

Notes

Additional keyword arguments can be passed to the initializer if rfilename is set. They will be passed on to *pydrobert.speech.util.read_signal()*

See also:

pydrobert.speech.util.read_signal
Describes the strategy used for loading signals

accumulate(*features*, *axis=-1*)

Accumulate statistics from a feature tensor

Parameters

- **features** (ndarray) –
- **axis** (int) –

Raises
ValueError – If the length of *axis* does not match that of past feature vector lengths

aliases = {'cmvn', 'normalize', 'standardize', 'unit'}

property have_stats

Whether at least one feature vector has been accumulated

Type
bool

save(*wfilename*, *key=None*, *compress=False*, *overwrite=True*)

Save accumulated statistics to file

If *wfilename* ends in `'.npy'`, stats will be written using `numpy.save()`.

If *wfilename* ends in `'.npz'`, stats will be written to a numpy archive. If *overwrite* is `False`, other key-values will be loaded first if possible, then resaved. If *key* is set, data will be indexed by *key* in the archive. Otherwise, the data will be stored at the first unused key of the pattern `'arr_d+'`. If *compress* is `True`, `numpy.savez_compressed()` will be used over `numpy.savez()`.

Otherwise, data will be written using `numpy.ndarray.tofile()`

Parameters

- **wfilename** (str) –
- **key** (Optional[str]) –
- **compress** (bool) –
- **overwrite** (bool) –

Raises

ValueError – If no stats have been accumulated

6.7 pydrobert.speech.pre

Classes for pre-processing speech signals

class pydrobert.speech.pre.**Dither**(*coeff=1.0*)

Bases: *PreProcessor*

Add random noise to a signal tensor

The default axis of *apply* has been set to None, which will generate random noise for each coefficient. This is likely the desired behaviour. Setting axis to an integer will add random values along 1D slices of that axis.

Intermediate values are calculated as 64-bit floats. The result is cast back to the input data type.

Parameters

coeff (float) – Added noise will be in the range `[-coeff, coeff]`

aliases = {'dither', 'dithering'}

coeff

class pydrobert.speech.pre.**PreProcessor**

Bases: *AliasedFactory*

A container for pre-processing signals with a transform

abstract apply(*signal*, *axis=-1*, *in_place=False*)

Applies the transformation to a signal tensor

Consult the class documentation for more details on what the transformation is.

Parameters

- **signal** (ndarray) –
- **axis** (int) – The axis of *signal* to apply the transformation along
- **in_place** – Whether it is okay to modify *signal* (True) or whether a copy should be made (False)

Returns

out (numpy.ndarray) – The transformed features

class pydrobert.speech.pre.**Preemphasize**(*coeff=0.97*)

Bases: *PreProcessor*

Attenuate the low frequencies of a signal by taking sample differences

The following transformation is applied along the target axis

```
new[i] = old[i] - coeff * old[i-1] for i > 1
new[0] = old[0]
```

This is essentially a convolution with a Haar wavelet for positive *coeff*. It emphasizes high frequencies.

Intermediate values are calculated as 64-bit floats. The result is cast back to the input data type.

Parameters
coeff (float) –

aliases = {'preemph', 'preemphasis', 'preemphasize'}

coeff

6.8 pydrobert.speech.scales

Scaling functions

Scaling functions transform a scalar in the frequency domain to some other real domain (the "scale" domain). The scaling functions should be invertible. Their primary purpose is to define the bandwidths of filters in *pydrobert.speech.filters*.

class pydrobert.speech.scales.**BarkScaling**

Bases: *ScalingFunction*

Psychoacoustic scaling function

Based on a collection experiments briefly mentioned in [zwicker1961] involving masking to determine critical bands. The functional approximation to the scale is implemented with the formula from [traunmuller1990] (being honest, from Wikipedia):

$$s = \begin{cases} z + 0.15(2 - z) & \text{if } z < 2 \\ z + 0.22(z - 20.1) & \text{if } z > 20.1 \end{cases}$$

where

$$z = 26.81 f / (1960 + f) - 0.53$$

Where s is the scale and f is the frequency in Hertz.

aliases = {'bark'}

class pydrobert.speech.scales.**LinearScaling**(*low_hz*, *slope_hz=1.0*)

Bases: *ScalingFunction*

Linear scaling between high and low scales/frequencies

Parameters

- **low_hz** (float) – The frequency (in Hertz) corresponding to scale 0.
- **slope_hz** (float) – The increase in scale corresponding to a 1 Hertz increase in frequency.

aliases = {'linear', 'uniform'}

low_hz

slop_hz

class pydrobert.speech.scales.**MelScaling**

Bases: *ScalingFunction*

Psychoacoustic scaling function

Based of the experiment in [stevens1937] wherein participants adjusted a second tone until it was half the pitch of the first. The functional approximation to the scale is implemented with the formula from [oshaughnessy1987] (being honest, from Wikipedia):

$$s = 1127 \ln\left(1 + \frac{f}{700}\right)$$

Where s is the scale and f is the frequency in Hertz.

aliases = {'mel'}

class pydrobert.speech.scales.**OctaveScaling**(*low_hz*)

Bases: *ScalingFunction*

Uniform scaling in log2 domain from low frequency

Parameters

low_hz (float) – The positive frequency (in Hertz) corresponding to scale 0. Frequencies below this value should never be queried.

aliases = {'octave'}

low_hz

class pydrobert.speech.scales.**ScalingFunction**

Bases: *AliasedFactory*

Converts a frequency to some scale and back again

abstract hertz_to_scale(*hertz*)

Convert frequency (in Hertz) to scalar

abstract scale_to_hertz(*scale*)

Convert scale to frequency (in Hertz)

6.9 pydrobert.speech.util

Miscellaneous utility functions

pydrobert.speech.util.**angular_to_hertz**(*angle*, *samp_rate*)

Convert radians/sec to cycles/sec

pydrobert.speech.util.**circshift_fourier**(*filt*, *shift*, *start_idx=0*, *dft_size=None*, *copy=True*)

Circularly shift a filter in the time domain, from the fourier domain

A simple application of the shift theorem

$$DFT(T_u x)[k] = DFT(x)[k] e^{-2i\pi k u}$$

Where we set `u = shift / dft_size`

Parameters

- **filt** (ndarray) – The filter, in the fourier domain

- **shift** (float) – The number of samples to be translated by.
- **start_idx** (int) – If *filt* is a truncated frequency response, this parameter indicates at what index in the dft the nonzero region starts
- **dft_size** (Optional[int]) – The dft_size of the filter. Defaults to `len(filt) + start_idx`
- **copy** (bool) – Whether it is okay to modify and return *filt*

Returns

out (numpy.ndarray) – The 128-bit complex filter frequency response, shifted by *u*

pydrobert.speech.util.**gauss_quant**(*p*, *mu=0*, *std=1*)

Gaussian quantile function

Given a probability from a univariate Gaussian, determine the value of the random variable such that the probability of drawing a value l.t.e. to that value is equal to the probability. In other words, the so-called inverse cumulative distribution function.

If *scipy* can be imported, this function uses `scipy.norm.ppf()` to calculate the result. Otherwise, it uses the approximation from Odeh & Evans 1974 (thru Brophy 1985)

Parameters

- **p** (float) – The probability
- **mu** (float) – The Gaussian mean
- **std** (float) – The Gaussian standard deviation

Returns

q (float) – The random variable value

pydrobert.speech.util.**hertz_to_angular**(*hertz*, *samp_rate*)

Convert cycles/sec to radians/sec

pydrobert.speech.util.**read_signal**(*rfilename*, *dtype=None*, *key=None*, *force_as=None*, ***kwargs*)

Read a signal from a variety of possible sources

Though the goal of this function is to return an array representing a signal of some sort, the way it goes about doing so depends on the setting of *rfilename*, processed in the following order:

1. If *rfilename* starts with the regular expression `r'^(ark|scp)(,w+)*:'`, the file is treated as a Kaldi table and opened with the kaldi data type *dtype* (defaults to `BaseMatrix`). The package `pydrobert.kaldi` will be imported to handle reading. If *key* is set, the value associated with that key is retrieved. Otherwise the first listed value is returned.
2. If *rfilename* ends with a file type listed in *pydrobert.speech.config.SOUNDFILE_SUPPORTED_FILE_TYPES* (requires `soundfile`), the file will be opened with that audio file type.
3. If *rfilename* ends with `'.wav'`, the file is assumed to be a wave file. The function will rely on the `scipy` package to load the file if `scipy` can be imported. Otherwise, it uses the standard `wave` package. The type of data encodings each package can handle varies, though neither can handle compressed data.
4. If *rfilename* ends with `'.hdf5'`, the file is assumed to be an HDF5 file. HDF5 and `h5py` must be installed on the host system to read this way. If *key* is set, the data will assumed to be indexed by *key* on the archive. Otherwise, a depth-first search of the archive will be performed for the first data set. If set, data will be cast to as the numpy data type *dtype*
5. If *rfilename* ends with `'.npy'`, the file is assumed to be a binary in Numpy format. If set, the result will be cast as the numpy data type *dtype*.

6. If *rfilename* ends with `'.npz'`, the file is assumed to be an archive in Numpy format. If *key* is swet, the data indexed by *key* will be loaded. Otherwise the data indexed by the key `'arr_0'` will be loaded. If set, the result will be cast as the numpy data type *dtype*.
7. If *rfilename* ends with `'.pt'`, the file is assumed to be a binary in PyTorch format. If set, the results will be cast as the numpy data type *dtype*.
8. If *rfilename* ends with `'.sph'`, the file is assumed to be a NIST SPHERE file. If set, the results will be cast as the numpy data type *dtype*
9. If *rfilename* ` ends with `'|'`, it will try to read an object of kaldi data type *dtype* (defaults to `BaseMatrix`) from a basic kaldi input stream.
10. Otherwise, we throw an `IOError`

Additional keyword arguments are passed along to the associated open or read operation.

Parameters

- **rfilename** (`str`) –
- **dtype** (`Optional`[`dtype`]) –
- **key** (`Optional`[`Any`]) –
- **force_as** (`Optional`[`str`]) – If not `None`, forces *rfilename* to be interpreted as a specific file type, bypassing the above selection strategy. `'tab'`: Kaldi table; `'wav'`: wave file; `'hdf5'`: HDF5 file; `'npy'`: Numpy binary; `'npz'`: Numpy archive; `'pt'`: PyTorch binary; `'sph'`: NIST sphere; `'kaldi'` Kaldi object; `'file'` read via `numpy.fromfile()`. The types in `SOUNDFILE_SUPPORTED_FILE_TYPES` are also valid values. *'soundfile'* will use `soundfile` to read the file regardless of the suffix.
- ****kwargs** –

Returns

signal (`numpy.ndarray`)

Warning: Post v 0.2.0, the behaviour after step 8 changed. Instead of trying to read first as Kaldi input, and, failing that, via `numpy.fromfile()`, we try to read as Kaldi input if the file name ends with `'|'` and error otherwise. The catch-all behaviour was disabled due to the interaction with *pydrobert.speech.config.SOUNDFILE_SUPPORTED_FILE_TYPES* whose value depends on the existence of `soundfile` and the underlying version of libsndfile.

Notes

Python code for reading SPHERE files (not via :mod:soundfile`) was based off of sph2pipe v 2.5. That code can only suppport the "shorten" audio format up to version 2.

6.10 pydrobert.speech.vis

Visualization functions

raises ImportError
This submodule requires `matplotlib`

pydrobert.speech.vis.**compare_feature_frames**(*computers*, *signal*, *axes=None*, *figure_height=None*, *figure_width=None*, *plot_titles=None*, *positions=None*, *post_ops=None*, *title=None*, ***kwargs*)

Compare features from frame computers via spectrogram-like heat map

Direct comparison of `FrameComputer` objects is possible because all subclasses of this abstract data type share a common interpretation of frame boundaries (according to `FrameComputer.frame_style()`).

Additional keyword args will be passed to the plotting routine.

Parameters

- **computers** (Union[*FrameComputer*, Sequence[*FrameComputer*]]) – One or more frame computers to compare
- **signal** (ndarray) – A 1D array of the raw speech. Assumed to be valid with respect to computer settings (e.g. sample rate).
- **axes** (Optional[int]) – By default, this function creates a new figure and subplots. Setting one *axes* value for every *computers* value will plot feature representations from *computers* into each ordered `Axes`. If *axes* do not belong to the same figure, a `ValueError` will be raised
- **figure_height** (Optional[float]) – If a new figure is created, this sets the figure height (in inches). This value is determined dynamically according to *figure_width* by default. A `ValueError` will be raised if both *figure_height* and *axes* are set
- **figure_width** (Optional[float]) – If a new figure is created, this set the figure width (in inches). This value defaults to 3.33 inches if all subplots are positioned vertically, and to 7 inches if there are at least two columns of plots. A `ValueError` will be raised if both *figure_width* and *axes* are set
- **plot_titles** (Optional[Tuple[str, ...]]) – An ordered list of strings specifying the titles of each subplot. The default is to not display subplot titles
- **positions** (Optional[Tuple[Union[int, Tuple[int, int]], ...]]) – If a new figure is created, *positions* decides how the subplots should be positioned relative to one another. Can contain only ints (describing the position on only the row-axis) or pairs of ints (describing the row-col positions). Positions must be contiguous and start from index 0 or 0,0 (top or top-left). *positions* cannot be specified if *axes* is specified
- **post_ops** (Union[*PostProcessor*, Sequence[*PostProcessor*], None]) – One or more post-processors to apply (in order) to each computed feature representation. If a simple list of post-processors is provided, each operation is applied to the default axis (the feature coefficient axis). To explicitly set the axis, pairs of `(op, axis)` can be specified in the list. No op is allowed to change the shape of the feature representation (e.g. `Deltas`), or a `ValueError` will be thrown
- **title** (Optional[str]) – The title of the whole figure. Default is to display no title

Returns
fig (matplotlib.figure.Figure) – The containing figure

pydrobert.speech.vis.**plot_frequency_response**(*banks*, *axes=None*, *dft_size=None*, *half=None*, *title=None*, *x_scale='hz'*, *y_scale='dB'*, *cmap=None*)

Plot frequency response of filters in a filter bank

Parameters

- **bank** –
- **axes** (Optional[Axes]) – An `Axes` object to plot on. Default is to generate a new figure
- **dft_size** (Optional[int]) – The size of the Discrete Fourier Transform to plot. Defaults to `max(max(bank.supports), 2 * bank.sampling_rate // min(bank.supports_hz)`
- **half** (Optional[bool]) – Whether to plot the half or full spectrum. Defaults to `bank.is_real`
- **title** (Optional[str]) – What to call the graph. The default is not to show a title
- **x_scale** (Literal['hz', 'ang', 'bins']) – The frequency coordinate scale along the x axis. Hertz (`'hz'`) is cycles/sec, angular frequency (`'ang'`) is radians/sec, and `'bins'` is the sample index within the DFT
- **y_scale** (Literal['dB', 'power', 'real', 'imag', 'both']) – How to express the frequency response along the y axis. Decibels (`'dB'`) is the log of a ratio of the maximum quantity in the bank. The range between 0 and -20 decibels is displayed. Power spectrum (`'power'`) is the squared magnitude of the frequency response. `'real'` is the real part of the response, `'imag'` is the imaginary part of the response, and `'both'` displays both `'real'` and `'imag'` as separate lines
- **cmap** (Optional[Colormap]) – A `Colormap` to pull colours from. Defaults to matplotlib's default colormap

Returns

fig (matplotlib.figure.Figure) – The containing figure

CHAPTER

SEVEN

REFERENCES

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[stevens1937] S. S. Stevens, J. Volkmann, and E. B. Newman, "A Scale for the Measurement of the Psychological Magnitude Pitch," The Journal of the Acoustical Society of America, vol. 8, no. 3, pp. 185-190, 1937.

[flanagan1960] J. L. Flanagan, "Models for approximating basilar membrane displacement," The Bell System Technical Journal, vol. 39, no. 5, pp. 1163-1191, Sep. 1960.

[zwicker1961] E. Zwicker, "Subdivision of the Audible Frequency Range into Critical Bands (Frequenzgruppen)," The Journal of the Acoustical Society of America, vol. 33, no. 2, pp. 248-248, 1961.

[aertsen1981] A. M. H. J. Aertsen, J. H. J. Olders, and P. I. M. Johannesma, "Spectro-temporal receptive fields of auditory neurons in the grassfrog," Biological Cybernetics, vol. 39, no. 3, pp. 195-209, Jan. 1981.

[oshaughnessy1987] D. O'Shaughnessy, Speech communication: human and machine. Addison-Wesley Pub. Co., 1987.

[traunmuller1990] H. Traunmüller, "Analytical expressions for the tonotopic sensory scale," The Journal of the Acoustical Society of America, vol. 88, no. 1, pp. 97-100, Jul. 1990.

[povey2011] D. Povey et al., "The Kaldi Speech Recognition Toolkit," in IEEE 2011 Workshop on Automatic Speech Recognition and Understanding, Hilton Waikoloa Village, Big Island, Hawaii, US, 2011.

[young] S. Young et al., "The HTK book (for HTK version 3.4)," Cambridge university engineering department, vol. 2, no. 2, pp. 2-3, 2006.

PYTHON MODULE INDEX

p

INDEX

E

F

G

H

I

L

M

N

O

P

R

S

T

U

W