dsdtools Documentation

Release 0.1.3

Fabian-Robert Stöter

July 20, 2016

Contents

A python package to parse and process the **demixing secrets dataset (DSD)** as part of the MUS task of the Signal Separation Evaluation Campaign (SISEC)

Contents:

CHAPTER 1

Installation

```
pip install dsdtools
```

1.1 DSD100 Dataset / Subset

The complete dataset (~14 GB) can be downloaded here. For testing and development we provide a subset of the DSD100 for direct download here. It has the same file and folder structure as well as the same audio file formats but consists of only 4 tracks of 30s each.

CHAPTER 2

Usage

This package should nicely integrate with your existing python code, thus makes it easy to participate in the SISEC MUS tasks. The core of this package is calling a user-provided function that separates the mixtures from the DSD into several estimated target sources.

2.1 Providing a compatible function

The core of this package consists of calling a user-provided function which separates the mixtures from the dsdtools into estimated target sources.

- The function will take an dsdtools `Track` object which can be used from inside your algorithm.
- Participants can access
- `Track.audio`, representing the stereo mixture as an `np.ndarray` of `shape=(nun_sampl, 2)`
- `Track.rate`, the sample rate
- `Track.path`, the absolute path of the mixture which might be handy to process with external applications, so that participants don't need to write out temporary wav files.
- The function needs to return a python `Dict` which consists of target name (`key`) and the estimated target as audio arrays with same shape as the mixture (`value`).
- It is the users choice which target sources they want to provide for a given mixture. Supported targets are `['vocals', 'accompaniment', 'drums', 'bass', 'other']`.
- Please make sure that the returned estimates do have the same sample rate as the mixture track.

Here is an example for such a function separating the mixture into a **vocals** and **accompaniment** track.

```
def my_function(track):

    # get the audio mixture as numpy array shape=(nun_sampl, 2)
    track.audio

    # compute voc_array, acc_array
    # ...

    return {
        'vocals': voc_array,
        'accompaniment': acc_array
    }
```

2.2 Create estimates for SiSEC evaluation

2.2.1 Setting up dsdtools

Simply import the dsdtools package in your main python function:

```
import dsdtools

dsd = dsdtools.DB(
    root_dir='path/to/dsdtools/',
)
```

The `root_dir` is the path to the dsdtools dataset folder. Instead of `root_dir` it can also be set system-wide. Just `export DSD_PATH=/path/to/dsdtools` inside your terminal environment.

2.2.2 Test if your separation function generates valid output

Before you run the full DSD100, which might take very long, participants can test their separation function by running:

```
dsd.test(my_function)
```

This test makes sure the user provided output is compatible to the dsdtools framework. The function returns `True` if the test succeeds.

2.2.3 Processing the full DSD100

To process all 100 DSD tracks and saves the results to the `estimates_dir`:

```
dsd.run(my_function, estimates_dir="path/to/estimates")
```

2.2.4 Processing training and testing subsets separately

Algorithms which make use of machine learning techniques can use the training subset and then apply the algorithm on the test data:

```
dsd.run(my_training_function, subsets="Dev")
dsd.run(my_test_function, subsets="Test")
```

If you want to exclude tracks from the training you can specify track ids as `dsdtools.DB(..., valid_ids=[1, 2]` object. Those tracks are then not included in `Dev` but are returned for `subsets="Valid"`.

2.2.5 Processing single or multiple DSD100 tracks

```
dsd.run(my_function, ids=30)
dsd.run(my_function, ids=[1, 2, 3])
dsd.run(my_function, ids=range(90, 99))
```

Note, that the provided list of ids can be overridden if the user sets a terminal environment variable `DSD_ID=1`.

2.2.6 Use multiple cores

Python Multiprocessing

To speed up the processing, `run` can make use of multiple CPUs:

```
dsd.run(my_function, parallel=True, cpus=4)
```

Note: We use the python builtin multiprocessing package, which sometimes is unable to parallelize the user provided function to PicklingError.

GNU Parallel

> GNU parallel is a shell tool for executing jobs in parallel using one or more computers. A job can be a single command or a small script that has to be run for each of the lines in the input. The typical input is a list of files, a list of hosts, a list of users, a list of URLs, or a list of tables. A job can also be a command that reads from a pipe. GNU parallel can then split the input and pipe it into commands in parallel.

By running only one `id` in each python process the dsdtools set can easily be processed with GNU parallel using multiple CPUs without any further modifications to your code:

```
parallel --bar 'DSD_ID={0} python main.py' ::: {1..100}
```

Compute the bss_eval measures

The official SISEC evaluation relies on *MATLAB* because currently there does not exist a bss_eval implementation for python which produces indentical results. Therefore please run `dsd100_eval_only.m` from the DSD100 Matlab scripts after you have processed and saved your estimates with *dsdtoolspy*.

2.3 Evaluation in python

> **Warning:** Evaluation in python is not supported yet

CHAPTER 3

Example

```
import dsdtools

def my_function(track):
    '''My fancy BSS algorithm'''

    # get the audio mixture as numpy array shape=(num_sampl, 2)
    track.audio

    # get the mixture path for external processing
    track.path

    # get the sample rate
    track.rate

    # return any number of targets
    estimates = {
        'vocals': vocals_array,
        'accompaniment': acc_array,
    }
    return estimates


# initiate dsdtools
dsd = dsdtools.DB(root_dir="./Volumes/Data/dsdtools")

# verify if my_function works correctly
if dsd.test(my_function):
    print "my_function is valid"

# this might take 3 days to finish
dsd.run(my_function, estimates_dir="path/to/estimates")
```

CHAPTER 4

Modules

class dsdtools.**DB**(*root_dir=None*, *setup_file=None*, *evaluation=None*, *valid_ids=None*)

Bases: object

The dsdtools DB Object

Parameters **root_dir** : str, optional

dsdtools Root path. If set to *None* it will be read from the *DSD_PATH* environment variable

subsets : str or list, optional

select a _dsdtools_ subset *Dev* or *Test* (defaults to both)

setup_file : str, optional

dsdtools Setup file in yaml format. Default is provided *dsd100.yaml*

evaluation : str, {None, 'bss_eval', 'mir_eval'}

Setup evaluation module and starts matlab if bsseval is enabled

valid_ids : list[int] or int, optional

select single or multiple _dsdtools_ items by ID that will be used for validation data (ie not included in the *Dev* set)

Attributes

setup_file	(str) path to yaml file. default: *setup.yaml*
root_dir	(str) dsdtools Root path. Default is *DSD_PATH* env
evaluation	(bool) Setup evaluation module
mixtures_dir	(str) path to Mixture directory
sources_dir	(str) path to Sources directory
sources_names	(list[str]) list of names of sources
targets_names	(list[str]) list of names of targets
evaluator	(BSSeval) evaluator used for evaluation of estimates
setup	(Dict) loaded yaml configuration

Methods

load_dsd_tracks()	Iterates through the dsdtools folder structure and returns `Track` objects
test(user_function)	Test the dsdtools processing
evaluate()	Run the evaluation
run(user_function=None, estimates_dir=None, evaluate=False)	Run the dsdtools processing, saving the estimates and optionally evaluate them

evaluate(*user_function=None*, *estimates_dir=None*, **args*, ***kwargs*)

Run the dsdtools evaluation

shortcut to ‘‘run(

user_function=None, estimates_dir=estimates_dir, evaluate=True

)‘‘

load_dsd_tracks(*subsets=None*, *ids=None*)

Parses the dsdtools folder structure and returns *Track* objects

Parameters **subsets** : list[str], optional

select a _dsdtools_ subset *Dev* or *Test*. Defaults to both

ids : list[int] or int, optional

select single or multiple _dsdtools_ items by ID

Returns list[Track]

return a list of `Track` Objects

run(*user_function=None*, *estimates_dir=None*, *evaluate=False*, *subsets=None*, *ids=None*, *parallel=False*, *cpus=4*)

Run the dsdtools processing

Parameters **user_function** : callable, optional

function which separates the mixture into estimates. If no function is provided (default in *None*) estimates are loaded from disk when *evaluate is True*.

estimates_dir : str, optional

path to the user provided estimates. Directory will be created if it does not exist. Default is *none* which means that the results are not saved.

evaluate : bool, optional

evaluate the estimates by using. Default is False

subsets : list[str], optional

select a _dsdtools_ subset *Dev* or *Test*. Defaults to both

ids : list[int] or int, optional

select single or multiple _dsdtools_ items by ID

parallel: bool, optional

activate multiprocessing

cpus: int, optional

set number of cores if *parallel* mode is active, defaults to 4

Raises **RuntimeError**

If the provided function handle is not callable.

See also:

test Test the user provided function

test(*user_function*)
Test the dsdtools processing

Parameters **user_function** : callable, optional

function which separates the mixture into estimates. If no function is provided (default in *None*) estimates are loaded from disk when *evaluate is True*.

Raises **TypeError**

If the provided function handle is not callable.

ValueError

If the output is not compliant to the bsseval methods

See also:

run Process the dsdtools

dsdtools.**init_worker**()

dsdtools.**process_function_alias**(*obj*, **args*, ***kwargs*)

4.1 Audio Classes

class dsdtools.audio_classes.**Source**(*name=None*, *path=None*)
Bases: object

An audio Target which is a linear mixture of several sources

Attributes

name	(str) Name of this source
path	(str) Absolute path to audio file
gain	(float) Mixing weight for this source

audio
array_like: [shape=(num_samples, num_channels)]

rate
int: sample rate in Hz

class dsdtools.audio_classes.**Target**(*sources*)
Bases: object

An audio Target which is a linear mixture of several sources

Attributes

sources	(list[Source]) list of `Source` objects for this `Target`

audio
array_like: [shape=(num_samples, num_channels)]

mixes audio for targets on the fly

class `dsdtools.audio_classes.`**`Track`**(*filename*, *track_id=None*, *track_artist=None*, *track_title=None*, *subset=None*, *path=None*)

Bases: `object`

An audio Track which is mixture of several sources and provides several targets

Attributes

name	(str) Track name
path	(str) Absolute path of mixture audio file
subset	({'Test', 'Dev'}) belongs to subset
targets	(OrderedDict) OrderedDict of mixted Targets for this Track
sources	(Dict) Dict of `Source` objects for this `Track`

audio
array_like: [shape=(num_samples, num_channels)]

rate
int: sample rate in Hz

CHAPTER 5

References

If you use this package, please reference the following paper

```
@inproceedings{SiSEC2015,
  TITLE = {{The 2015 Signal Separation Evaluation Campaign}},
  AUTHOR = {N. Ono and Z. Rafii and D. Kitamura and N. Ito and A. Liutkus},
  BOOKTITLE = {{International Conference on Latent Variable Analysis and Signal Separation  (LVA/ICA)
  ADDRESS = {Liberec, France},
  SERIES = {Latent Variable Analysis and Signal Separation},
  VOLUME = {9237},
  PAGES = {387-395},
  YEAR = {2015},
  MONTH = Aug,
}
```

d

A

D

E

I

L

P

R

S

T