musdb Documentation

Release 0.2.1

Fabian-Robert Stöter

Jun 30, 2019

Contents

A python package to parse and process the **sigsep musdb18** dataset as part of the MUS task of the Signal Separation Evaluation Campaign (SISEC)

Contents:

CHAPTER 1

Installation

Decoding

As the MUSDB18 is encoded as STEMS, it relies on ffmpeg to read the multi-stream files. We provide a python wrapper called stempeg that allows to easily parse the dataset and decode the stem tracks on-the-fly. Before you install musdb (that includes the stempeg requirement), it is therefore required to install ffmpeg. The installation differ among operating systems.

E.g. if you use Anaconda you can install ffmpeg on Windows/Mac/Linux using the following command:

```
conda install -c conda-forge ffmpeg
```

Alternatively you can install ffmpeg manually as follows:

Mac: use homebrew: brew install ffmpeg Ubuntu Linux: sudo apt-get install ffmpeg

Use a decoded version

If you have trouble installing stempeg or ffmpeg we also support parse and process the pre-decoded PCM/wav files. We provide docker based scripts to decode the dataset to wav files. If you want to use the decoded musdb dataset, use the is_wav parameter when initialising the dataset.

Package installation

You can install the musdb parsing package using pip:

pip install musdb

```
pip install musdb
```

1.1 SIGSEP-MUS Dataset / Subset

The dataset can be downloaded here.

CHAPTER 2

Usage

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

- The function will take an musdb `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
```

(continues on next page)

(continued from previous page)

```
    # compute voc_array, acc_array
    # ...

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

2.2 Create estimates for SiSEC evaluation

2.2.1 Setting up musdb

Simply import the musdb package in your main python function:

```
import musdb

mus = musdb.DB(
    root_dir='path/to/musdb/',
)
```

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

2.2.2 Test if your separation function generates valid output

Before you run the full 150 tracks, which might take very long, participants can test their separation function by running:

```
mus.test(my_function)
```

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

2.2.3 Processing the full musdb

To process all 150 musdb tracks and saves the results to the `estimates_dir`:

```
mus.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:

```
mus.run(my_training_function, subsets="train")
mus.run(my_test_function, subsets="test")
```

2.2.5 Access the reference signals / targets

For supervised learning you can use the provided reference sources by loading the *track.targets* dictionary. E.g. to access the vocal reference from a track:

```
track.targets['vocals'].audio
```

2.2.6 Use multiple cores

Python Multiprocessing

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

```
mus.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.

CHAPTER 3

Example

```python
import musdb

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 musdb
mus = musdb.DB(root_dir="./Volumes/Data/musdb")

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

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

CHAPTER 4

Modules

4.1 Audio Classes

CHAPTER 5

References

If you use this package, please reference the following paper