mirdata

Jan 14, 2021

Contents

This library provides tools for working with common MIR datasets, including tools for:

- downloading datasets to a common location and format
- validating that the files for a dataset are all present
- loading annotation files to a common format, consistent with the format required by *mir_eval*
- parsing track level metadata for detailed evaluations.

This libary was presented in our ISMIR 2019 paper

CHAPTER 1

Installation

```
pip install mirdata
```

CHAPTER 2

API documentation

2.1 API documentation

2.1.1 Dataset Loaders

mirdata.beatles

mirdata.guitarset

mirdata.ikala

mirdata.medleydb_melody

mirdata.medleydb_pitch

mirdata.medley_solos_db

mirdata.orchset

mirdata.rwc_classical

mirdata.rwc_genre

mirdata.rwc_jazz

mirdata.rwc_popular

mirdata.salami

mirdata.dali

2.1.2 Utilities

mirdata.utils

mirdata.download_utils

mirdata.jams_utils

CHAPTER 3

Examples

3.1 Examples

First of all, you can install mirdata using *pip*.

```
$ pip install mirdata
```

With *mirdata*, the first thing you would do is to download the dataset. Depending on the availability of the audio files, *mirdata* may only show you an instruction about how to download the dataset. Fortunately, we can download Orchset dataset directly.

```
import mirdata
# Download the Orchset Dataset
mirdata.orchset.download()
# Orchset_dataset_0.zip?download=1: 1.00B [03:05, 185s/B]
```

Once downloading is done, you can find the the dataset folder.

```
$ ls ~/mir_datasets/Orchset/
GT
Orchset - Predominant Melodic Instruments.csv
README.txt
audio
midi
```

The ID's and annotation data can be loaded as below.

```
# Load the dataset
orchset_data = mirdata.orchset.load()
orchset_ids = mirdata.orchset.track_ids()

# todo: add __str__() method and print(orchset_data)
```

If we wanted to use Orchset to evaluate the performance of a melody extraction algorithm (in our case, *very_bad_melody_extractor*), and then split the scores based on the metadata, we could do the following:

```python
import mir_eval
import mirdata
import numpy as np
import sox

def very_bad_melody_extractor(audio_path):
    duration = sox.file_info.duration(audio_path)
    time_stamps = np.arange(0, duration, 0.01)
    melody_f0 = np.random.uniform(low=80.0, high=800.0, size=time_stamps.shape)
    return time_stamps, melody_f0

# Evaluate on the full dataset
orchset_scores = {}
orchset_data = mirdata.orchset.load()
for track_id, track_data in orchset_data.items():
    est_times, est_freqs = very_bad_melody_extractor(track_data.audio_path_mono)

    ref_melody_data = track_data.melody
    ref_times = ref_melody_data.times
    ref_freqs = ref_melody_data.frequencies

    score = mir_eval.melody.evaluate(ref_times, ref_freqs, est_times, est_freqs)
    orchset_scores[track_id] = score

# Split the results by composer and by instrumentation
composer_scores = {}
strings_no_strings_scores = {True: {}, False: {}}
for track_id, track_data in orchset_data.items():
    if track_data.composer not in composer_scores.keys():
        composer_scores[track_data.composer] = {}

    composer_scores[track_data.composer][track_id] = orchset_scores[track_id]
    strings_no_strings_scores[track_data.contains_strings][track_id] = \
        orchset_scores[track_id]
```

This is the result of the example above.

```
# strings_no_strings_scores

{True: {'Beethoven-S3-I-ex1': OrderedDict([('Voicing Recall', 1.0),
                ('Voicing False Alarm', 1.0),
                ('Raw Pitch Accuracy', 0.029798422436459245),
                ('Raw Chroma Accuracy', 0.08063102541630149),
                ('Overall Accuracy', 0.0272654370489174)]),
  'Beethoven-S3-I-ex2': OrderedDict([('Voicing Recall', 1.0),
                ('Voicing False Alarm', 1.0),
                ('Raw Pitch Accuracy', 0.009221311475409836),
                ('Raw Chroma Accuracy', 0.07377049180327869),
                ('Overall Accuracy', 0.008754863813229572)]),

...

  'Wagner-Tannhauser-Act2-ex2': OrderedDict([('Voicing Recall', 1.0),
            ('Voicing False Alarm', 1.0),
            ('Raw Pitch Accuracy', 0.03685636856368564),
            ('Raw Chroma Accuracy', 0.08997289972899729),
            ('Overall Accuracy', 0.036657681940700806)])}}
```

very_bad_melody_extractor performs very badly!

3.2 FAQ

Q How do I add a new loader?

A Take a look at our instructions!

Q How do I get access to a dataset if the download function says it's not available?

A We don't distribute data ourselves, so unfortunately it is up to you to find the data yourself. We strongly encourage you to favor datasets which are currently available.

Q Can you send me the data for a dataset which is not available?

A No, we do not host or distribute datasets.

Q What do I do if my data fails validation?

A Very often, data fails vaildation because of how the files are named or how the folder is structured. If this is the case, try renaming/reorganizing your data to match what mirdata expects. If your data fails validation because of the checksums, this means that you are using data which is different from what most people are using, and you should try to get the more common dataset version, for example by using the data loader's download function. If you want to use your data as-is and don't want to see the annoying validation logging, you can set *silence_validator=True* when calling *.load()*.

Q How do you choose the data that is used to create the checksums?

A Whenever possible, the data downloaded using *.download()* is the same data used to create the checksums. If this isn't possible, we did our best to get the data from the original source (the dataset creator) in order to create the checksum. If this is again not possible, we found as many versions of the data as we could from different users of the dataset, computed checksums on all of them and used the version which was the most common amongst them.

Q Why didn't you release a version of this library in matlab/C/java/R?

A The creators of this library are python users, so we made a libray in python. We'd be very happy to provide guidance to anyone who wants to create a version of this library in another programming languages.

Q The download link is broken in a loader. What do I do?

A Please open an issue and tag it with the "broken link" label.

Q Why the name, mirdata?

A mir = mir + data. MIR is an acronym for Music Information Retrieval, and the library was built for working with data.

Q If I find a mistake in an annotation, should I fix it in the loader?

A No. All datasets have "mistakes", and we do not want to create another version of each dataset ourselves. The loaders should load the data as released. After that, it's up to the user what they want to do with it.

CHAPTER 4

Contribute

- Issue Tracker
- Source Code