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

Documentation

2.1 Table of supported datasets

This table is provided as a guide for users to select appropriate datasets. The list of annotations omits some metadata for brevity, and we document the dataset's primary annotations only. The number of tracks indicates the number of unique "tracks" in a dataset, but it may not reflect the actual size or diversity of a dataset, as tracks can vary greatly in length (from a few seconds to a few minutes), and may be homogeneous. For specific information about the contents of each dataset, click the link provided in the "Module" column.

"Downloadable" possible values:

- : Freely downloadable
- : Available upon request
- : Youtube Links only
- : Not available

2.1.1 Annotation Type Descriptions

The table above provides annotation types as a guide for choosing appropriate datasets, but it is difficult to generically categorize annotation types, as they depend on varying definitions and their meaning can change depending on the type of music they correspond to. Here we provide a rough guide to the types in this table, but we **strongly recommend** reading the dataset specific documentation to ensure the data is as you expect.

Beats

Musical beats, typically encoded as sequence of timestamps and corresponding beat positions. This implicitly includes *downbeat* information (the beginning of a musical measure).

Chords

Musical chords, e.g. as might be played on a guitar. Typically encoded as a sequence of labeled events, where each event has a start time, end time, and a label. The label taxonomy varies per dataset, but typically encode a chord's root and its quality, e.g. A:m7 for "A minor 7".

Drums

Transcription of the drums, typically encoded as a sequence of labeled events, where the labels indicate which drum instrument (e.g. cymbal, snare drum) is played. These events often overlap with one another, as multiple drums can be played at the same time.

F0

Musical pitch contours, typically encoded as time series indidcating the musical pitch over time. The time series typically have evenly spaced timestamps, each with a correspoinding pitch value which may be encoded in a number of formats/granularities, including midi note numbers and Hertz.

Genre

A typically global "tag", indicating the genre of a recording. Note that the concept of genre is highly subjective and we refer those new to this task to this article.

Instruments

Labels indicating which instrument is present in a musical recording. This may refer to recordings of solo instruments, or to recordings with multiple instruments. The labels may be global to a recording, or they may vary over time, indicating the presence/absence of a particular instrument as a time series.

Key

Musical key. This can be defined globally for an audio file or as a sequence of events.

Lyrics

Lyrics corresponding to the singing voice of the audio. These may be raw text with no time information, or they may be time-aligned events. They may have varying levels of granularity (paragraph, line, word, phoneme, character) depending on the dataset.

Melody

The musical melody of a song. Melody has no universal definition and is typically defined per dataset. It is typically enocoded as *F0* or as *Notes*. Other types of annotations such as Vocal F0 or Vocal Notes can often be considered as melody annotations as well.

Notes

Musical note events, typically encoded as sequences of start time, end time, label. The label typically indicates a musical pitch, which may be in a number of formats/granularities, including midi note numbers, Hertz, or pitch class.

Sections

Musical sections, which may be "flat" or "hierarchical", typically encoded by a sequence of timestamps indicating musical section boundary times. Section annotations sometimes also include labels for sections, which may indicate repetitions and/or the section type (e.g. Chorus, Verse).

Technique

The playing technique used by a particular instrument, for example "Pizzicato". This label may be global for a given recording or encoded as a sequence of labeled events.

Tempo

The tempo of a song, typical in units of beats-per-minute (bpm). This is often indicated globally per track, but in practice tracks may have tempos that change, and some datasets encode tempo as time-varying quantity. Additionally, there may be multiple reasonable tempos at any given time (for example, often 2x or 0.5x a tempo value will also be "correct"). For this reason, some datasets provide two or more different tempo values.

Vocal Activity

A time series or sequence of events indicating when singing voice is present in a recording. This type of annotation is implicitly available when Vocal *F0* or Vocal *Notes* annotations are available.

2.2 Examples

2.2.1 Basic Example

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.orchset
# 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:

```
import mir_eval
import mirdata.orchset
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),
```

(continues on next page)

(continued from previous page)

```
                ('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!

2.2.2 Using mirdata with local vs. remote data

When using mirdata on the same machine as where your datasets live, we do the loading for you.

For example, to load the melody annotations from Orchset into memory, we can simply call:

```
import mirdata.orchset

# Load a single track
track = mirdata.orchset.Track('Beethoven-S3-I-ex1')
melody_annotation = track.melody

print(melody_annotation)
# F0Data(times=array([0.000e+00, 1.000e-02, 2.000e-02, ..., 1.244e+01, 1.245e+01,
#   1.246e+01]), frequencies=array([  0.   ,   0.   ,   0.   , ..., 391.995, 391.995,
↪391.995]), confidence=array([0., 0., 0., ..., 1., 1., 1.]))
```

However, if your data lives somewhere else, accessing the annotation will return *None*. Instead, you can use the module's loading functions directly:

```
import mirdata.orchset

# Load a single track, specifying the remote location
track = mirdata.orchset.Track('Beethoven-S3-I-ex1', data_home='gs://my_custom/remote_
↪path')
melody_path = track.melody_path

print(melody_path)
# gs://my_custom/remote_path/GT/Beethoven-S3-I-ex1.mel
print(os.path.exists(melody_path))
# False

# write code here to locally download your path e.g. to a temporary file.
def my_downloader(remote_path):
    # the contents of this function will depend on where your data lives, and how
↪permanently you want the files to remain on the machine. We point you to libraries
↪handling common use cases below.
    # for data you would download via scp, you could use the [scp](https://pypi.org/
↪project/scp/) library
    # for data on google drive, use [pydrive](https://pythonhosted.org/PyDrive/)
    # for data on google cloud storage use [google-cloud-storage](https://pypi.org/
↪project/google-cloud-storage/)
    return local_path_to_downloaded_data
```

(continues on next page)

(continued from previous page)

```

temp_path = my_downloader(melody_path)

# call orchset's melody annotation loader
melody_annotation = orchset.load_melody(temp_path)

print(melody_annotation)
# F0Data(times=array([0.000e+00, 1.000e-02, 2.000e-02, ..., 1.244e+01, 1.245e+01,
#   1.246e+01]), frequencies=array([  0.   ,   0.   ,   0.   , ..., 391.995, 391.995,␣
↪391.995]), confidence=array([0., 0., 0., ..., 1., 1., 1.]))
```

2.2.3 Using mirdata with tf.data.Dataset

The following is a simple example of a generator that can be used to create a tensorflow Dataset

```
import mirdata.orchset
import numpy as np
import tensorflow as tf

def orchset_generator():
    # using the default data_home
    track_ids = mirdata.orchset.track_ids()
    for track_id in track_ids:
        track = mirdata.orchset.Track(track_id)
        audio_signal, sample_rate = track.audio_mono
        yield {
            "audio": audio_signal.astype(np.float32),
            "sample_rate": sample_rate,
            "annotation": {
                "times": track.melody.times.astype(np.float32),
                "freqs": track.melody.frequencies.astype(np.float32),
            },
            "metadata": {"track_id": track.track_id}
        }

dataset = tf.data.Dataset.from_generator(
    orchset_generator,
    {
        "audio": tf.float32,
        "sample_rate": tf.float32,
        "annotation": {"times": tf.float32, "freqs": tf.float32},
        "metadata": {'track_id': tf.string}
    }
)
```

2.3 API documentation

2.3.1 Dataset Loaders

mirdata.beatles

Beatles Dataset Loader

The Beatles Dataset includes beat and metric position, chord, key, and segmentation annotations for 179 Beatles songs. Details can be found in http://matthiasmauch.net/_pdf/mauch_omp_2009.pdf and http://isophonics.net/content/reference-annotations-beatles.

class `mirdata.beatles.`**`Track`**(*track_id*, *data_home=None*)

Beatles track class

Parameters

- **`track_id`** (*str*) – track id of the track
- **`data_home`** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

`audio_path`

track audio path

Type str

`beats_path`

beat annotation path

Type str

`chords_path`

chord annotation path

Type str

`keys_path`

key annotation path

Type str

`sections_path`

sections annotation path

Type str

`title`

title of the track

Type str

`track_id`

track id

Type str

`audio`

audio signal, sample rate

Type (np.ndarray, float)

`beats`

human-labeled beat annotation

Type BeatData

`chords`

chord annotation

Type ChordData

`key`

key annotation

Type KeyData

sections
section annotation

Type SectionData

to_jams()
Jams: the track's data in jams format

mirdata.beatles.**cite**()
Print the reference

mirdata.beatles.**download**(*data_home=None*, *force_overwrite=False*, *cleanup=True*)
Download the Beatles Dataset (annotations). The audio files are not provided due to copyright issues.

Parameters

- **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*
- **force_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

mirdata.beatles.**load**(*data_home=None*)
Load Beatles dataset

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns {*track_id*: track data}

Return type (dict)

mirdata.beatles.**load_audio**(*audio_path*)
Load a Beatles audio file.

Parameters **audio_path** (*str*) – path to audio file

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

mirdata.beatles.**load_beats**(*beats_path*)
Load Beatles format beat data from a file

Parameters **beats_path** (*str*) – path to beat annotation file

Returns loaded beat data

Return type (utils.BeatData)

mirdata.beatles.**load_chords**(*chords_path*)
Load Beatles format chord data from a file

Parameters **chords_path** (*str*) – path to chord annotation file

Returns loaded chord data

Return type (utils.ChordData)

mirdata.beatles.**load_key**(*keys_path*)
Load Beatles format key data from a file

Parameters **keys_path** (*str*) – path to key annotation file

Returns loaded key data

Return type (utils.KeyData)

`mirdata.beatles.load_sections(sections_path)`
Load Beatles format section data from a file

Parameters **sections_path** (*str*) – path to section annotation file

Returns loaded section data

Return type (utils.SectionData)

`mirdata.beatles.track_ids()`
Get the list of track IDs for this dataset

Returns A list of track ids

Return type (list)

`mirdata.beatles.validate(data_home=None, silence=False)`
Validate if a local version of this dataset is consistent

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns

List of file paths that are in the dataset index but missing locally

invalid_checksums (list): List of file paths where the expected file exists locally but has a different checksum than the reference

Return type missing_files (list)

mirdata.beatport_key

beatport_key Dataset Loader The Beatport EDM Key Dataset includes 1486 two-minute sound excerpts from various EDM subgenres, annotated with single-key labels, comments and confidence levels generously provided by Eduard Mas Marín, and thoroughly revised and expanded by Ángel Faraldo.

The original audio samples belong to online audio snippets from Beatport, an online music store for DJ's and Electronic Dance Music Producers (<http:www.beatport.com>). If this dataset were used in further research, we would appreciate the citation of the current DOI (10.5281/zenodo.1101082) and the following doctoral dissertation, where a detailed description of the properties of this dataset can be found:

Ángel Faraldo (2017). Tonality Estimation in Electronic Dance Music: A Computational and Musically Informed Examination. PhD Thesis. Universitat Pompeu Fabra, Barcelona.

This dataset is mainly intended to assess the performance of computational key estimation algorithms in electronic dance music subgenres.

Data License: Creative Commons Attribution Share Alike 4.0 International

class `mirdata.beatport_key.Track(track_id, data_home=None)`
beatport_key track class :param track_id: track id of the track :type track_id: str :param data_home: Local path where the dataset is stored.

If *None*, looks for the data in the default directory, *~/mir_datasets*

audio_path
track audio path

Type str

keys_path
key annotation path

Type str

metadata_path
sections annotation path

Type str

title
title of the track

Type str

track_id
track id

Type str

artists
artist annotation

Type Dict

audio
audio signal, sample rate

Type (np.ndarray, float)

genres
genre annotation

Type Dict

key
key annotation

Type String

tempo
tempo beatports crowdsourced annotation

Type int

to_jams()
Jams: the track's data in jams format

mirdata.beatport_key.**cite**()
Print the reference

mirdata.beatport_key.**download**(*data_home=None*, *force_overwrite=False*, *cleanup=True*, *partial_download=None*)
Download the beatport_key Dataset (annotations). The audio files are not provided due to copyright issues.

This dataset annotations have characters that doesnt correspond with json format. In particular, "bpm": nan doesn't correspond to json format. The function find_replace is used to fix this problem. input file :param data_home: Local path where the dataset is stored.

If *None*, looks for the data in the default directory, *~/mir_datasets*

Parameters

- **force_overwrite** (*bool*) – Whether to overwrite the existing downloaded data

- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.
- **partial_download** (*list of str*) – arguments can be 'audio' 'metadata' or/and 'keys'

`mirdata.beatport_key.find_replace`(*directory*, *find*, *replace*, *pattern*)
Replace in some directory all the songs with the format pattern find by replace :param directory (str) path to directory: :param find (str) string from replace: :param replace (str) string to replace: :param pattern (str) regex that must match the directories searrched:

`mirdata.beatport_key.load`(*data_home=None*)
Load beatport_key dataset :param data_home: Local path where the dataset is stored.

If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns {*track_id*: track data}

Return type (dict)

`mirdata.beatport_key.load_artist`(*metadata_path*)
Load beatport_key tempo data from a file :param metadata_path: path to metadata annotation file :type metadata_path: str

Returns list of artists involved in the track.

Return type (list of strings)

`mirdata.beatport_key.load_audio`(*audio_path*)
Load a beatport_key audio file. :param audio_path: path to audio file :type audio_path: str

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

`mirdata.beatport_key.load_genre`(*metadata_path*)
Load beatport_key genre data from a file :param metadata_path: path to metadata annotation file :type metadata_path: str

Returns with the list of strings with genres ['genres'] and list of strings with sub-genres ['sub_genres']

Return type (dict)

`mirdata.beatport_key.load_key`(*keys_path*)
Load beatport_key format key data from a file :param keys_path: path to key annotation file :type keys_path: str

Returns loaded key data

Return type (str)

`mirdata.beatport_key.load_tempo`(*metadata_path*)
Load beatport_key tempo data from a file :param metadata_path: path to metadata annotation file :type metadata_path: str

Returns loaded tempo data

Return type (str)

`mirdata.beatport_key.track_ids`()
Get the list of track IDs for this dataset :returns: A list of track ids :rtype: (list)

`mirdata.beatport_key.validate`(*data_home=None*, *silence=False*)
Validate if a local version of this dataset is consistent :param data_home: Local path where the dataset is stored.

If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns

List of file paths that are in the dataset index but missing locally

invalid_checksums (list): List of file paths where the expected file exists locally but has a different checksum than the reference

Return type missing_files (list)

mirdata.dali

DALI Dataset Loader

DALI contains 5358 audio files with their time-aligned vocal melody. It also contains time-aligned lyrics at four levels of granularity: notes, words, lines, and paragraphs.

For each song, DALI also provides additional metadata: genre, language, musician, album covers, or links to video clips.

For more details, please visit: https://github.com/gabolsgabs/DALI

class `mirdata.dali.`**`Track`**(*track_id*, *data_home=None*)

DALI melody Track class

Parameters

- **`track_id`** (*str*) – track id of the track
- **`data_home`** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

`album`

the track's album

Type str

`annotation_path`

path to the track's annotation file

Type str

`artist`

the track's artist

Type str

`audio_path`

path to the track's audio file

Type str

`audio_url`

youtube ID

Type str

`dataset_version`

dataset annotation version

Type int

`ground_truth`

True if the annotation is verified

Type bool

language
sung language

Type str

release_date
year the track was released

Type str

scores_manual
TODO

Type int

scores_ncc
TODO

Type float

title
the track's title

Type str

track_id
the unique track id

Type str

url_working
True if the youtube url was valid

Type bool

annotation_object
DALI Annotations object

Type DALI.Annotations

audio
audio signal, sample rate

Type (np.ndarray, float)

lines
line-aligned lyrics

Type LyricData

notes
note-aligned lyrics

Type NoteData

paragraphs
paragraph-aligned lyrics

Type LyricData

to_jams()
Jams: the track's data in jams format

words
word-aligned lyric

Type LyricData

`mirdata.dali.cite()`
Print the reference

`mirdata.dali.download(data_home=None, force_overwrite=False)`
DALI is not available for downloading directly. This function prints a helper message to download DALI through zenodo.org.

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

`mirdata.dali.load(data_home=None)`
Load DALI dataset

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns {*track_id*: track data}

Return type (dict)

`mirdata.dali.load_annotations_class(annotations_path)`
Load full annotations into the DALI class object

Parameters **annotations_path** (*str*) – path to a DALI annotation file

Returns DALI annotations object

`mirdata.dali.load_annotations_granularity(annotations_path, granularity)`
Load annotations at the specified level of granularity

Parameters

- **annotations_path** (*str*) – path to a DALI annotation file
- **granularity** (*str*) – one of 'notes', 'words', 'lines', 'paragraphs'

Returns NoteData for granularity='notes' or LyricData otherwise

`mirdata.dali.load_audio(audio_path)`
Load a DALI audio file.

Parameters **audio_path** (*str*) – path to audio file

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

`mirdata.dali.track_ids()`
Return track ids

Returns A list of track ids

Return type (list)

`mirdata.dali.validate(data_home=None, silence=False)`
Validate if the stored dataset is a valid version

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns

List of file paths that are in the dataset index but missing locally

invalid_checksums (list): List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

Return type missing_files (list)

mirdata.giantsteps_tempo

giantsteps_tempo Dataset Loader

name: GiantSteps (tempo+genre)

contact: Richard Vogl <richard.vogl@tuwien.ac.at> Peter Knees <peter.knees@tuwien.ac.at>

description: collection of annotations for 664 2min(1) audio previews from www.beatport.com

references: [1] Peter Knees, Ángel Faraldo, Perfecto Herrera, Richard Vogl, Sebastian Böck, Florian Hörschläger, Mickael Le Goff: "Two data sets for tempo estimation and key detection in electronic dance music annotated from user corrections", Proc. of the 16th Conference of the International Society for Music Information Retrieval (ISMIR'15), Oct. 2015, Malaga, Spain.

[2] Hendrik Schreiber, Meinard Müller: "A Crowdsourced Experiment for Tempo Estimation of Electronic Dance Music", Proc. of the 19th Conference of the International Society for Music Information Retrieval (ISMIR'18), Sept. 2018, Paris, France.

annotations: tempo (bpm), genre

notes:

The audio files (664 files, size ~1gb) can be downloaded from http://www.beatport.com/ using the bash script:

https://github.com/GiantSteps/giantsteps-tempo-dataset/blob/master/audio_dl.sh

To download the files manually use links of the following form: http://geo-samples.beatport.com/lofi/<name of mp3 file> e.g.: http://geo-samples.beatport.com/lofi/5377710.LOFI.mp3

To convert the audio files to .wav use (bash + sox):

./convert_audio.sh

To retrieve the genre information, the JSON contained within the website was parsed. The tempo annotation was extracted from forum entries of people correcting the bpm values (i.e. manual annotation of tempo). For more information please contact creators.

[2] found some files without tempo. There are:

3041381.LOFI.mp3 3041383.LOFI.mp3 1327052.LOFI.mp3

Their v2 tempo is denoted as 0.0 in tempo and mirex and has no annotation in the JAMS format.

(1): Most of the audio files are 120 seconds long. Exceptions are: name length 906760.LOFI.mp3 62 1327052.LOFI.mp3 70 4416506.LOFI.mp3 80 1855660.LOFI.mp3 119 3419452.LOFI.mp3 119 3577631.LOFI.mp3 119

mirdata.giantsteps_key

giantsteps_key Dataset Loader

The GiantSteps+ EDM Key Dataset includes 600 two-minute sound excerpts from various EDM subgenres, annotated with single-key labels, comments and confidence levels by Daniel G. Camhi, and thoroughly revised and expanded by Ángel Faraldo at MTG UPF. Additionally, 500 tracks have been thoroughly analysed, containing pitch-class set

descriptions, key changes, and additional modal changes. This dataset is a revision of the original GiantSteps Key Dataset, available in Github (<https://github.com/GiantSteps/giantsteps-key-dataset>) and initially described in:

Knees, P., Faraldo, Á., Herrera, P., Vogl, R., Böck, S., Hörschläger, F., Le Goff, M. (2015). Two Datasets for Tempo Estimation and Key Detection in Electronic Dance Music Annotated from User Corrections. In Proceedings of the 16th International Society for Music Information Retrieval Conference, 364–370. Málaga, Spain.

The original audio samples belong to online audio snippets from Beatport, an online music store for DJ's and Electronic Dance Music Producers (<http:www.beatport.com>). If this dataset were used in further research, we would appreciate the citation of the current DOI (10.5281/zenodo.1101082) and the following doctoral dissertation, where a detailed description of the properties of this dataset can be found:

Ángel Faraldo (2017). Tonality Estimation in Electronic Dance Music: A Computational and Musically Informed Examination. PhD Thesis. Universitat Pompeu Fabra, Barcelona.

This dataset is mainly intended to assess the performance of computational key estimation algorithms in electronic dance music subgenres.

All the data of this dataset is licensed with Creative Commons Attribution Share Alike 4.0 International.

class `mirdata.giantsteps_key.`**`Track`**(*track_id*, *data_home=None*)

giantsteps_key track class

Parameters

- **`track_id`** (*str*) – track id of the track
- **`data_home`** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

`audio_path`

track audio path

Type str

`keys_path`

key annotation path

Type str

`metadata_path`

sections annotation path

Type str

`title`

title of the track

Type str

`track_id`

track id

Type str

`artists`

artist annotation

Type Dict

`audio`

audio signal, sample rate

Type (np.ndarray, float)

genres
genre annotation

Type Dict

key
key annotation

Type String

tempo
tempo beatports crowdsourced annotation

Type int

to_jams()
Jams: the track's data in jams format

mirdata.giantsteps_key.**cite**()
Print the reference

mirdata.giantsteps_key.**download**(*data_home=None*, *force_overwrite=False*, *cleanup=True*, *partial_download=None*)
Download the giantsteps_key Dataset (annotations). The audio files are not provided due to copyright issues.

Parameters

- **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*
- **force_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.
- **partial_download** (*list of str*) – arguments can be 'audio' 'metadata' or/and 'keys'

mirdata.giantsteps_key.**load**(*data_home=None*)
Load giantsteps_key dataset

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns {*track_id*: track data}

Return type (dict)

mirdata.giantsteps_key.**load_artist**(*metadata_path*)
Load giantsteps_key tempo data from a file

Parameters **metadata_path** (*str*) – path to metadata annotation file

Returns list of artists involved in the track.

Return type (list of strings)

mirdata.giantsteps_key.**load_audio**(*audio_path*)
Load a giantsteps_key audio file.

Parameters **audio_path** (*str*) – path to audio file

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

mirdata.giantsteps_key.**load_genre**(*metadata_path*)
Load giantsteps_key genre data from a file

Parameters **metadata_path** (*str*) – path to metadata annotation file

Returns with the list of strings with genres ['genres'] and list of strings with sub-genres ['sub_genres']

Return type (dict)

`mirdata.giantsteps_key.load_key(keys_path)`

Load giantsteps_key format key data from a file

Parameters **keys_path** (*str*) – path to key annotation file

Returns loaded key data

Return type (str)

`mirdata.giantsteps_key.load_tempo(metadata_path)`

Load giantsteps_key tempo data from a file

Parameters **metadata_path** (*str*) – path to metadata annotation file

Returns loaded tempo data

Return type (str)

`mirdata.giantsteps_key.track_ids()`

Get the list of track IDs for this dataset

Returns A list of track ids

Return type (list)

`mirdata.giantsteps_key.validate(data_home=None, silence=False)`

Validate if a local version of this dataset is consistent

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns

List of file paths that are in the dataset index but missing locally

invalid_checksums (list): List of file paths where the expected file exists locally but has a different checksum than the reference

Return type missing_files (list)

mirdata.groove_midi

Groove MIDI Loader

The Groove MIDI Dataset (GMD) is composed of 13.6 hours of aligned MIDI and synthesized audio of human-performed, tempo-aligned expressive drumming. The dataset contains 1,150 MIDI files and over 22,000 measures of drumming.

To enable a wide range of experiments and encourage comparisons between methods on the same data, Gillick et al. created a new dataset of drum performances recorded in MIDI format. They hired professional drummers and asked them to perform in multiple styles to a click track on a Roland TD-11 electronic drum kit. They also recorded the aligned, high-quality synthesized audio from the TD-11 and include it in the release.

The Groove MIDI Dataset (GMD), has several attributes that distinguish it from existing ones:

- The dataset contains about 13.6 hours, 1,150 MIDI files, and over 22,000 measures of drumming.
- Each performance was played along with a metronome set at a specific tempo by the drummer.

- The data includes performances by a total of 10 drummers, with more than 80% of duration coming from hired professionals. The professionals were able to improvise in a wide range of styles, resulting in a diverse dataset.
- The drummers were instructed to play a mix of long sequences (several minutes of continuous playing) and short beats and fills.
- Each performance is annotated with a genre (provided by the drummer), tempo, and anonymized drummer ID.
- Most of the performances are in 4/4 time, with a few examples from other time signatures.
- Four drummers were asked to record the same set of 10 beats in their own style. These are included in the test set split, labeled eval-session/groove1-10.
- In addition to the MIDI recordings that are the primary source of data for the experiments in this work, the authors captured the synthesized audio outputs of the drum set and aligned them to within 2ms of the corresponding MIDI files.

A train/validation/test split configuration is provided for easier comparison of model accuracy on various tasks.

The dataset is made available by Google LLC under a Creative Commons Attribution 4.0 International (CC BY 4.0) License.

For more details, please visit: http://magenta.tensorflow.org/datasets/groove

class `mirdata.groove_midi.`**`Track`**(*track_id*, *data_home=None*)

Groove MIDI Track class

Parameters

- **`track_id`** (*str*) – track id of the track
- **`data_home`** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir_datasets*

`drummer`

Drummer id of the track (ex. 'drummer1')

Type str

`session`

Type of session (ex. 'session1', 'eval_session')

Type str

`track_id`

track id of the track (ex. 'drummer1/eval_session/1')

Type str

`style`

Style (genre, groove type) of the track (ex. 'funk/groove1')

Type str

`tempo`

Track tempo in beats per minute (ex. 138)

Type int

`beat_type`

Whether the track is a beat or a fill (ex. 'beat')

Type str

`time_signature`

Time signature of the track (ex. '4-4', '6-8')

Type str

midi_path
Path to the midi file

Type str

audio_path
Path to the audio file

Type str

duration
Duration of the midi file in seconds

Type float

split
Whether the track is for a train/valid/test set. One of 'train', 'valid' or 'test'.

Type str

audio
audio signal, sample rate

Type (np.ndarray, float)

beats
machine-generated beat annotation

Type BeatData

drum_events
annotated drum kit events

Type EventData

midi
prettyMIDI obj

Type (obj)

`mirdata.groove_midi.cite()`
Print the reference

`mirdata.groove_midi.download(data_home=None, force_overwrite=False, cleanup=True)`
Download Groove MIDI.

Parameters

- **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*
- **force_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

`mirdata.groove_midi.load(data_home=None)`
Load Groove MIDI dataset

Parameters **data_home** (*str*) – Local path where Groove MIDI is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns {*track_id*: track data}

Return type (dict)

`mirdata.groove_midi.load_audio(audio_path)`

Load a Groove MIDI audio file.

Parameters **audio_path** (*str*) – path to audio file

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

`mirdata.groove_midi.load_beats(midi_path, midi=None)`

Load beat data from the midi file.

Parameters

- **midi_path** (*str*) – path to midi file
- **midi** (*pretty_midi.PrettyMIDI*) – pre-loaded midi object or None if None, the midi object is loaded using midi_path

Returns beat_data (BeatData)

`mirdata.groove_midi.load_drum_events(midi_path, midi=None)`

Load drum events from the midi file.

Parameters

- **midi_path** (*str*) – path to midi file
- **midi** (*pretty_midi.PrettyMIDI*) – pre-loaded midi object or None if None, the midi object is loaded using midi_path

Returns drum_events (EventData)

`mirdata.groove_midi.load_midi(midi_path)`

Load a Groove MIDI midi file.

Parameters **midi_path** (*str*) – path to midi file

Returns pretty_midi object

Return type midi_data (pretty_midi.PrettyMIDI)

`mirdata.groove_midi.track_ids()`

Return track ids

Returns A list of track ids

Return type (list)

`mirdata.groove_midi.validate(data_home=None, silence=False)`

Validate if the stored dataset is a valid version

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns

List of file paths that are in the dataset index but missing locally

invalid_checksums (list): List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

Return type missing_files (list)

mirdata.gtzan_genre

GTZAN-Genre Dataset Loader

This dataset was used for the well known paper in genre classification "Musical genre classification of audio signals " by G. Tzanetakis and P. Cook in IEEE Transactions on Audio and Speech Processing 2002.

The dataset consists of 1000 audio tracks each 30 seconds long. It contains 10 genres, each represented by 100 tracks. The tracks are all 22050 Hz mono 16-bit audio files in .wav format.

class `mirdata.gtzan_genre.`**`Track`**(*track_id*, *data_home=None*)

gtzan_genre Track class

Parameters

- **track_id** (*str*) – track id of the track
- **data_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir_datasets*

audio_path

path to the audio file

Type str

genre

annotated genre

Type str

track_id

track id

Type str

audio

audio signal, sample rate

Type (np.ndarray, float)

to_jams()

Jams: the track's data in jams format

`mirdata.gtzan_genre.`**`cite`**()

Print the reference

`mirdata.gtzan_genre.`**`download`**(*data_home=None*, *force_overwrite=False*, *cleanup=True*)

Download the GTZAN-Genre dataset.

Parameters

- **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*
- **force_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

`mirdata.gtzan_genre.`**`load`**(*data_home=None*)

Load GTZAN-Genre

Parameters **data_home** (*str*) – Local path where GTZAN-Genre is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns {*track_id*: track data}

Return type (dict)

`mirdata.gtzan_genre.load_audio(audio_path)`
Load a GTZAN audio file.

Parameters **audio_path** (*str*) – path to audio file

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

`mirdata.gtzan_genre.track_ids()`
Return track ids

Returns A list of track ids

Return type (list)

`mirdata.gtzan_genre.validate(data_home=None, silence=False)`
Validate if the stored dataset is a valid version

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns

List of file paths that are in the dataset index but missing locally

invalid_checksums (list): List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

Return type missing_files (list)

mirdata.guitarset

GuitarSet Loader

GuitarSet provides audio recordings of a variety of musical excerpts played on an acoustic guitar, along with time-aligned annotations including pitch contours, string and fret positions, chords, beats, downbeats, and keys.

GuitarSet contains 360 excerpts that are close to 30 seconds in length. The 360 excerpts are the result of the following combinations:

- 6 players
- 2 versions: comping (harmonic accompaniment) and soloing (melodic improvisation)
- 5 styles: Rock, Singer-Songwriter, Bossa Nova, Jazz, and Funk
- 3 Progressions: 12 Bar Blues, Autumn Leaves, and Pachelbel Canon.
- 2 Tempi: slow and fast.

The tonality (key) of each excerpt is sampled uniformly at random.

GuitarSet was recorded with the help of a hexaphonic pickup, which outputs signals for each string separately, allowing automated note-level annotation. Excerpts are recorded with both the hexaphonic pickup and a Neumann U-87 condenser microphone as reference. 3 audio recordings are provided with each excerpt with the following suffix:

- hex: original 6 channel wave file from hexaphonic pickup
- hex_cln: hex wave files with interference removal applied
- mic: monophonic recording from reference microphone
- mix: monophonic mixture of original 6 channel file

Each of the 360 excerpts has an accompanying JAMS file which stores 16 annotations. Pitch:

- 6 pitch_contour annotations (1 per string)
- 6 midi_note annotations (1 per string)

Beat and Tempo:

- 1 beat_position annotation
- 1 tempo annotation

Chords:

- 2 chord annotations: instructed and performed. The instructed chord annotation

is a digital version of the lead sheet that's provided to the player, and the performed chord annotations are inferred from note annotations, using segmentation and root from the digital lead sheet annotation.

For more details, please visit: http://github.com/marl/guitarset/

class `mirdata.guitarset.`**`Track`**(*track_id*, *data_home=None*)

guitarset Track class

Parameters

- **`track_id`** (*str*) – track id of the track
- **`data_home`** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir_datasets*

`audio_hex_cln_path`

path to the debleeded hex wave file

Type str

`audio_hex_path`

path to the original hex wave file

Type str

`audio_mic_path`

path to the mono wave via microphone

Type str

`audio_mix_path`

path to the mono wave via downmixing hex pickup

Type str

`jams_path`

path to the jams file

Type str

`mode`

one of ['solo', 'comp'] For each excerpt, players are asked to first play in 'comp' mode and later play a 'solo' version on top of the already recorded comp.

Type str

`player_id`

ID of the different players. one of ['00', '01', . . . , '05']

Type str

style
one of [‘Jazz’, ‘Bossa Nova’, ‘Rock’, ‘Singer-Songwriter’, ‘Funk’]

Type str

tempo
BPM of the track

Type float

track_id
track id

Type str

audio_hex
raw hexaphonic audio signal, sample rate

Type (np.ndarray, float)

audio_hex_cln
bleed-removed hexaphonic audio signal, sample rate

Type (np.ndarray, float)

audio_mic
stereo microphone audio signal, sample rate

Type (np.ndarray, float)

audio_mix
stereo mix audio signal, sample rate

Type (np.ndarray, float)

beats
the track’s beat positions

Type BeatData

inferred_chords
the track’s chords inferred from played transcription

Type ChordData

key_mode
the track’s key and mode

Type KeyData

leadsheet_chords
the track’s chords as written in the leadsheet

Type ChordData

notes
a dict that contains 6 NoteData. From Low E string to high e string. {

‘E’: NoteData(...), ‘A’: NoteData(...), ... ‘e’: NoteData(...)

}

Type dict

pitch_contours
a dict that contains 6 F0Data. From Low E string to high e string. {

'E': F0Data(...), 'A': F0Data(...), ... 'e': F0Data(...)

}

Type (dict)

to_jams()

Jams: the track's data in jams format

mirdata.guitarset.**cite**()

Print the reference

mirdata.guitarset.**download**(*data_home=None*, *partial_download=None*, *force_overwrite=False*, *cleanup=True*)

Download GuitarSet.

Parameters

- **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*
- **force_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **partial_download** (*list*) –

  List indicating what to partially download. The list can include any of:

  - *'annotations'* the annotation files
  - *'audio_hex_original'* original 6 channel wave file from hexaphonic pickup
  - *'audio_hex_debleeded'* hex wave files with interference removal applied
  - *'audio_mic'* monophonic recording from reference microphone
  - *'audio_mix'* monophonic mixture of original 6 channel file

  If *None*, all data is downloaded.
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

mirdata.guitarset.**load**(*data_home=None*)

Load GuitarSet

Parameters **data_home** (*str*) – Local path where GuitarSet is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns {*track_id*: track data}

Return type (dict)

mirdata.guitarset.**load_audio**(*audio_path*)

Load a Guitarset audio file.

Parameters **audio_path** (*str*) – path to audio file

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

mirdata.guitarset.**load_chords**(*jams_path*, *leadsheet_version=True*)

Parameters

- **jams_path** (*str*) – Path of the jams annotation file
- **leadsheet_version** (*Bool*) – Whether or not to load the leadsheet version of the chord annotation If False, load the infered version.

Returns Chord data

Return type (ChordData)

`mirdata.guitarset.load_multitrack_audio(audio_path)`
Load a Guitarset multitrack audio file.

Parameters **audio_path** (*str*) – path to audio file

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

`mirdata.guitarset.load_note_ann(jams_path, string_num)`

Parameters **jams_path** (*str*) – Path of the jams annotation file

string_num (int), in range(6): Which string to load. 0 is the Low E string, 5 is the high e string.

`mirdata.guitarset.load_pitch_contour(jams_path, string_num)`

Parameters **jams_path** (*str*) – Path of the jams annotation file

string_num (int), in range(6): Which string to load. 0 is the Low E string, 5 is the high e string.

`mirdata.guitarset.track_ids()`
Return track ids

Returns A list of track ids

Return type (list)

`mirdata.guitarset.validate(data_home=None, silence=False)`
Validate if the stored dataset is a valid version

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns

List of file paths that are in the dataset index but missing locally

invalid_checksums (list): List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

Return type missing_files (list)

mirdata.ikala

iKala Dataset Loader

The iKala dataset is comprised of 252 30-second excerpts sampled from 206 iKala songs (plus 100 hidden excerpts reserved for MIREX). The music accompaniment and the singing voice are recorded at the left and right channels respectively and can be found under the Wavfile directory. In addition, the human-labeled pitch contours and timestamped lyrics can be found under PitchLabel and Lyrics respectively.

For more details, please visit: http://mac.citi.sinica.edu.tw/ikala/

class `mirdata.ikala.Track(track_id, data_home=None)`
ikala Track class

Parameters

- **track_id** (*str*) – track id of the track

- **data_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir_datasets*

audio_path
path to the track's audio file

Type str

f0_path
path to the track's f0 annotation file

Type str

lyrics_path
path to the track's lyric annotation file

Type str

section
section. Either 'verse' or 'chorus'

Type str

singer_id
singer id

Type str

song_id
song id of the track

Type str

track_id
track id

Type str

f0
The human-annotated singing voice pitch

Type F0Data

instrumental_audio
mono instrumental audio signal, sample rate

Type (np.ndarray, float)

lyrics
The human-annotated lyrics

Type LyricData

mix_audio
mono mixture audio signal, sample rate

Type (np.ndarray, float)

to_jams()
Jams: the track's data in jams format

vocal_audio
mono vocal audio signal, sample rate

Type (np.ndarray, float)

`mirdata.ikala.cite()`
Print the reference

`mirdata.ikala.download(data_home=None, force_overwrite=False)`
Download iKala Dataset. However, iKala dataset is not available for download anymore. This function prints a helper message to organize pre-downloaded iKala dataset.

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

`mirdata.ikala.load(data_home=None)`
Load iKala dataset

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns {*track_id*: track data}

Return type (dict)

`mirdata.ikala.load_instrumental_audio(audio_path)`
Load an ikala instrumental.

Parameters **audio_path** (*str*) – path to audio file

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

`mirdata.ikala.load_mix_audio(audio_path)`
Load an ikala mix.

Parameters **audio_path** (*str*) – path to audio file

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

`mirdata.ikala.load_vocal_audio(audio_path)`
Load an ikala vocal.

Parameters **audio_path** (*str*) – path to audio file

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

`mirdata.ikala.track_ids()`
Return track ids

Returns A list of track ids

Return type (list)

`mirdata.ikala.validate(data_home=None, silence=False)`
Validate if the stored dataset is a valid version

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns

List of file paths that are in the dataset index but missing locally

invalid_checksums (list): List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

Return type missing_files (list)

mirdata.maestro

MAESTRO Dataset Loader

MAESTRO (MIDI and Audio Edited for Synchronous TRacks and Organization) is a dataset composed of over 200 hours of virtuosic piano performances captured with fine alignment (~3 ms) between note labels and audio waveforms.

The dataset is created and released by Google's Magenta team.

The dataset contains over 200 hours of paired audio and MIDI recordings from ten years of International Piano-e-Competition. The MIDI data includes key strike velocities and sustain/sostenuto/una corda pedal positions. Audio and MIDI files are aligned with 3 ms accuracy and sliced to individual musical pieces, which are annotated with composer, title, and year of performance. Uncompressed audio is of CD quality or higher (44.1–48 kHz 16-bit PCM stereo).

A train/validation/test split configuration is also proposed, so that the same composition, even if performed by multiple contestants, does not appear in multiple subsets. Repertoire is mostly classical, including composers from the 17th to early 20th century.

The dataset is made available by Google LLC under a Creative Commons Attribution Non-Commercial Share-Alike 4.0 (CC BY-NC-SA 4.0) license.

This loader supports MAESTRO version 2.

For more details, please visit: https://magenta.tensorflow.org/datasets/maestro

class `mirdata.maestro.`**`Track`**(*track_id*, *data_home=None*)

MAESTRO Track class

Parameters

- **`track_id`** (*str*) – track id of the track
- **`data_home`** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir_datasets*

`audio_path`

Path to the track's audio file

Type str

`canonical_composer`

Composer of the piece, standardized on a single spelling for a given name.

Type str

`canonical_title`

Title of the piece. Not guaranteed to be standardized to a single representation.

Type str

`duration`

Duration in seconds, based on the MIDI file.

Type float

`midi_path`

Path to the track's MIDI file

Type str

`split`

Suggested train/validation/test split.

Type str

track_id
track id

Type str

year
Year of performance.

Type int

audio
track's audio signal, sample rate

Type (np.ndarray, float)

midi
description of output

Type output type

notes
annotated piano notes

Type NoteData

to_jams()
Jams: the track's data in jams format

mirdata.maestro.**cite**()
Print the reference

mirdata.maestro.**download**(*data_home=None*, *partial_download=None*, *force_overwrite=False*, *cleanup=True*)
Download the dataset.

Parameters

- **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*
- **force_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **partial_download** (*list*) –

  List indicating what to partially download. The list can include any of:

  - 'all': audio, midi and metadata
  - 'midi': midi and metadata only
  - 'metadata': metadata only

  If *None*, all data is downloaded.
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

mirdata.maestro.**load**(*data_home=None*)
Load MAESTRO dataset

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns {*track_id*: track data}

Return type (dict)

`mirdata.maestro.load_audio(audio_path)`
Load a MAESTRO audio file.

Parameters **audio_path** (*str*) – path to audio file

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

`mirdata.maestro.load_midi(midi_path)`
Load a MAESTRO midi file.

Parameters **midi_path** (*str*) – path to midi file

Returns pretty_midi object

Return type midi_data (obj)

`mirdata.maestro.load_notes(midi_path, midi=None)`
Load note data from the midi file.

Parameters

- **midi_path** (*str*) – path to midi file
- **midi** (*pretty_midi.PrettyMIDI*) – pre-loaded midi object or None if None, the midi object is loaded using midi_path

Returns note_data (NoteData)

`mirdata.maestro.track_ids()`
Return track ids

Returns A list of track ids

Return type (list)

`mirdata.maestro.validate(data_home=None, silence=False)`
Validate if the stored dataset is a valid version

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns

List of file paths that are in the dataset index but missing locally

invalid_checksums (list): List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

Return type missing_files (list)

mirdata.medleydb_melody

MedleyDB melody Dataset Loader

MedleyDB is a dataset of annotated, royalty-free multitrack recordings. MedleyDB was curated primarily to support research on melody extraction, addressing important shortcomings of existing collections. For each song we provide melody f0 annotations as well as instrument activations for evaluating automatic instrument recognition.

For more details, please visit: https://medleydb.weebly.com

class `mirdata.medleydb_melody.Track(track_id, data_home=None)`
medleydb_melody Track class

Parameters

- **track_id** (*str*) – track id of the track
- **data_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir_datasets*

artist
artist

Type str

audio_path
path to the audio file

Type str

genre
genre

Type str

is_excerpt
True if the track is an excerpt

Type bool

is_instrumental
True of the track does not contain vocals

Type bool

melody1_path
path to the melody1 annotation file

Type str

melody2_path
path to the melody2 annotation file

Type str

melody3_path
path to the melody3 annotation file

Type str

n_sources
Number of instruments in the track

Type int

title
title

Type str

track_id
track id

Type str

audio
audio signal, sample rate

Type (np.ndarray, float)

melody1
The pitch of the single most predominant source (often the voice)

Type F0Data

melody2
The pitch of the predominant source for each point in time

Type F0Data

melody3
The pitch of any melodic source. Allows for more than one f0 value at a time.

Type *MultipitchData*

to_jams()
Jams: the track's data in jams format

mirdata.medleydb_melody.**cite**()
Print the reference

mirdata.medleydb_melody.**download**(*data_home=None*)
MedleyDB is not available for downloading directly. This function prints a helper message to download MedleyDB through zenodo.org.

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

mirdata.medleydb_melody.**load**(*data_home=None*)
Load MedleyDB melody dataset

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns {*track_id*: track data}

Return type (dict)

mirdata.medleydb_melody.**load_audio**(*audio_path*)
Load a MedleyDB audio file.

Parameters **audio_path** (*str*) – path to audio file

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

mirdata.medleydb_melody.**track_ids**()
Return track ids

Returns A list of track ids

Return type (list)

mirdata.medleydb_melody.**validate**(*data_home=None*, *silence=False*)
Validate if the stored dataset is a valid version

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns

List of file paths that are in the dataset index but missing locally

invalid_checksums (list): List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

Return type missing_files (list)

mirdata.medleydb_pitch

MedleyDB pitch Dataset Loader

MedleyDB is a dataset of annotated, royalty-free multitrack recordings. MedleyDB was curated primarily to support research on melody extraction, addressing important shortcomings of existing collections. For each song we provide melody f0 annotations as well as instrument activations for evaluating automatic instrument recognition.

For more details, please visit: https://medleydb.weebly.com

class `mirdata.medleydb_pitch.`**`Track`**(*track_id*, *data_home=None*)

medleydb_pitch Track class

Parameters

- **`track_id`** (*str*) – track id of the track
- **`data_home`** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir_datasets*

`artist`

artist

Type str

`audio_path`

path to the audio file

Type str

`genre`

genre

Type str

`instrument`

instrument of the track

Type str

`pitch_path`

path to the pitch annotation file

Type str

`title`

title

Type str

`track_id`

track id

Type str

`audio`

audio signal, sample rate

Type (np.ndarray, float)

`pitch`

The human-annotated pitch

Type F0Data

to_jams()
Jams: the track's data in jams format

mirdata.medleydb_pitch.**cite**()
Print the reference

mirdata.medleydb_pitch.**download**(*data_home=None*)
MedleyDB is not available for downloading directly. This function prints a helper message to download MedleyDB through zenodo.org.

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

mirdata.medleydb_pitch.**load**(*data_home=None*)
Load MedleyDB pitch dataset

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns {*track_id*: track data}

Return type (dict)

mirdata.medleydb_pitch.**load_audio**(*audio_path*)
Load a MedleyDB audio file.

Parameters **audio_path** (*str*) – path to audio file

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

mirdata.medleydb_pitch.**track_ids**()
Return track ids

Returns A list of track ids

Return type (list)

mirdata.medleydb_pitch.**validate**(*data_home=None*, *silence=False*)
Validate if the stored dataset is a valid version

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns

List of file paths that are in the dataset index but missing locally

invalid_checksums (list): List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

Return type missing_files (list)

mirdata.medley_solos_db

Medley-solos-DB Dataset Loader.

Medley-solos-DB is a cross-collection dataset for automatic musical instrument recognition in solo recordings. It consists of a training set of 3-second audio clips, which are extracted from the MedleyDB dataset (Bittner et al., ISMIR 2014) as well as a test set of 3-second clips, which are extracted from the solosDB dataset (Essid et al., IEEE TASLP 2009). Each of these clips contains a single instrument among a taxonomy of eight:

0. clarinet,

1. distorted electric guitar,
2. female singer,
3. flute,
4. piano,
5. tenor saxophone,
6. trumpet, and
7. violin.

The Medley-solos-DB dataset is the dataset that is used in the benchmarks of musical instrument recognition in the publications of Lostanlen and Cella (ISMIR 2016) and Andén et al. (IEEE TSP 2019).

class `mirdata.medley_solos_db.`**`Track`**(*track_id*, *data_home=None*)

medley_solos_db Track class

Parameters

- **`track_id`** (*str*) – track id of the track
- **`data_home`** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir_datasets*

`audio_path`

path to the track's audio file

Type str

`instrument`

instrument encoded by its English name

Type str

`instrument_id`

instrument encoded as an integer

Type int

`song_id`

song encoded as an integer

Type int

`subset`

either equal to 'train', 'validation', or 'test'

Type str

`track_id`

track id

Type str

`audio`

audio signal, sample rate

Type (np.ndarray, float)

`to_jams`()

Jams: the track's data in jams format

`mirdata.medley_solos_db.`**`cite`**()

Print the reference

`mirdata.medley_solos_db.download(data_home=None, partial_download=None, force_overwrite=False, cleanup=True)`

Download Medley-solos-DB.

Parameters

- **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*
- **force_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **partial_download** (*list*) –

  List indicating what to partially download. The list can include any of:

  - *'annotations'* the annotation files
  - *'audio'* the audio files

  If *None*, all data is downloaded.
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

`mirdata.medley_solos_db.load(data_home=None)`

Load Medley-solos-DB :param data_home: Local path where Medley-solos-DB is stored.

If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns {*track_id*: track data}

Return type (dict)

`mirdata.medley_solos_db.load_audio(audio_path)`

Load a Medley Solos DB audio file.

Parameters **audio_path** (*str*) – path to audio file

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

`mirdata.medley_solos_db.track_ids()`

Return track ids

Returns A list of track ids

Return type (list)

`mirdata.medley_solos_db.validate(data_home=None, silence=False)`

Validate if the stored dataset is a valid version

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns

List of file paths that are in the dataset index but missing locally

invalid_checksums (list): List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

Return type missing_files (list)

mirdata.orchset

ORCHSET Dataset Loader

Orchset is intended to be used as a dataset for the development and evaluation of melody extraction algorithms. This collection contains 64 audio excerpts focused on symphonic music with their corresponding annotation of the melody.

For more details, please visit: https://zenodo.org/record/1289786#.XREpzaeZPx6

class `mirdata.orchset.`**`Track`**(*track_id*, *data_home=None*)

orchset Track class

Parameters

- **track_id** (*str*) – track id of the track
- **data_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir_datasets*

alternating_melody

True if the melody alternates between instruments

Type bool

audio_path_mono

path to the mono audio file

Type str

audio_path_stereo

path to the stereo audio file

Type str

composer

the work's composer

Type str

contains_brass

True if the track contains any brass instrument

Type bool

contains_strings

True if the track contains any string instrument

Type bool

contains_winds

True if the track contains any wind instrument

Type bool

excerpt

True if the track is an excerpt

Type str

melody_path

path to the melody annotation file

Type str

only_brass

True if the track contains brass instruments only

Type bool

only_strings
True if the track contains string instruments only

Type bool

only_winds
True if the track contains wind instruments only

Type bool

predominant_melodic_instruments
List of instruments which play the melody

Type list

track_id
track id

Type str

work
The musical work

Type str

audio_mono
mono audio signal, sample rate

Type (np.ndarray, float)

audio_stereo
stereo audio signal, sample rate

Type (np.ndarray, float)

melody
melody annotation

Type F0Data

to_jams()
Jams: the track's data in jams format

mirdata.orchset.**cite**()
Print the reference

mirdata.orchset.**download**(*data_home=None*, *force_overwrite=False*, *cleanup=True*)
Download ORCHSET Dataset.

Parameters

- **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*
- **force_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

mirdata.orchset.**load**(*data_home=None*)
Load ORCHSET dataset

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns {*track_id*: track data}

Return type (dict)

`mirdata.orchset.load_audio_mono(audio_path)`
Load a Orchset audio file.

Parameters **audio_path** (*str*) – path to audio file

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

`mirdata.orchset.load_audio_stereo(audio_path)`
Load a Orchset audio file.

Parameters **audio_path** (*str*) – path to audio file

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

`mirdata.orchset.track_ids()`
Return track ids

Returns A list of track ids

Return type (list)

`mirdata.orchset.validate(data_home=None, silence=False)`
Validate if the stored dataset is a valid version

Parameters

- **dataset_path** (*str*) – ORCHSET dataset local path
- **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns

List of file paths that are in the dataset index but missing locally

invalid_checksums (list): List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

Return type missing_files (list)

mirdata.rwc_classical

RWC Classical Dataset Loader

The Classical Music Database consists of 50 pieces:

- Symphonies: 4 pieces
- Concerti: 2 pieces
- Orchestral music: 4 pieces
- Chamber music: 10 pieces
- Solo performances: 24 pieces
- Vocal performances: 6 pieces

For more details, please visit: https://staff.aist.go.jp/m.goto/RWC-MDB/rwc-mdb-c.html

class `mirdata.rwc_classical.`**`Track`**(*track_id*, *data_home=None*)

rwc_classical Track class

Parameters

- **track_id** (*str*) – track id of the track
- **data_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir_datasets*

artist

the track's artist

Type str

audio_path

path of the audio file

Type str

beats_path

path of the beat annotation file

Type str

category

One of 'Symphony', 'Concerto', 'Orchestral', 'Solo', 'Chamber', 'Vocal', or blank.

Type str

composer

Composer of this Track.

Type str

duration

Duration of the track in seconds

Type float

piece_number

Piece number of this Track, [1-50]

Type str

sections_path

path of the section annotation file

Type str

suffix

string within M01-M06

Type str

title

Title of The track.

Type str

track_id

track id

Type str

track_number

CD track number of this Track

Type str

audio
audio signal, sample rate

Type (np.ndarray, float)

beats
human labeled beat annotations

Type BeatData

sections
human labeled section annotations

Type SectionData

to_jams()
Jams: the track's data in jams format

mirdata.rwc_classical.**download**(*data_home=None*, *partial_download=None*, *force_overwrite=False*, *cleanup=True*)
Download the RWC Classical (annotations and metadata). The audio files are not provided due to copyright issues.

Parameters

- **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*
- **force_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **partial_download** (*list*) – List indicating what to partially download. The list can include any of * *'annotations_beat'* the beat annotation files * *'annotations_sections'* the sections annotation files * *'metadata'* the metadata files If *None*, all data is downloaded.
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

mirdata.rwc_classical.**load**(*data_home=None*)
Load RWC-Classical dataset

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns {*track_id*: track data}

Return type (dict)

mirdata.rwc_classical.**load_audio**(*audio_path*)
Load a RWC audio file.

Parameters **audio_path** (*str*) – path to audio file

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

mirdata.rwc_classical.**track_ids**()
Return track ids

Returns A list of track ids

Return type (list)

mirdata.rwc_classical.**validate**(*data_home=None*, *silence=False*)
Validate if the stored dataset is a valid version

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns

List of file paths that are in the dataset index but missing locally

invalid_checksums (list): List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

Return type missing_files (list)

mirdata.rwc_jazz

RWC Jazz Dataset Loader.

The Jazz Music Database consists of 50 pieces:

- Instrumentation variations: 35 pieces (5 pieces × 7 instrumentations).

The instrumentation-variation pieces were recorded to obtain different versions of the same piece; i.e., different arrangements performed by different player instrumentations. Five standard-style jazz pieces were originally composed and then performed in modern-jazz style using the following seven instrumentations: 1. Piano solo 2. Guitar solo 3. Duo: Vibraphone + Piano, Flute + Piano, and Piano + Bass 4. Piano trio: Piano + Bass + Drums 5. Piano trio + Trumpet or Tenor saxophone 6. Octet: Piano trio + Guitar + Alto saxophone + Baritone saxophone + Tenor saxophone × 2 7. Piano trio + Vibraphone or Flute

- Style variations: 9 pieces

The style-variation pieces were recorded to represent various styles of jazz. They include four well-known public-domain pieces and consist of 1. Vocal jazz: 2 pieces (including "Aura Lee") 2. Big band jazz: 2 pieces (including "The Entertainer") 3. Modal jazz: 2 pieces 4. Funky jazz: 2 pieces (including "Silent Night") 5. Free jazz: 1 piece (including "Joyful, Joyful, We Adore Thee") Fusion (crossover): 6 pieces The fusion pieces were recorded to obtain music that combines elements of jazz with other styles such as popular, rock, and latin. They include music with an eighth-note feel, music with a sixteenth-note feel, and Latin jazz music.

For more details, please visit: https://staff.aist.go.jp/m.goto/RWC-MDB/rwc-mdb-j.html

class mirdata.rwc_jazz.**Track**(*track_id*, *data_home=None*)

rwc_jazz Track class

Parameters

- **track_id** (*str*) – track id of the track
- **data_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir_datasets*

artist

Artist name

Type str

audio_path

path of the audio file

Type str

beats_path

path of the beat annotation file

Type str

duration
Duration of the track in seconds

Type float

instruments
list of used instruments.

Type str

piece_number
Piece number of this Track, [1-50]

Type str

sections_path
path of the section annotation file

Type str

suffix
M01-M04

Type str

title
Title of The track.

Type str

track_id
track id

Type str

track_number
CD track number of this Track

Type str

variation
TODO

Type str

audio
audio signal, sample rate

Type (np.ndarray, float)

beats
human-labeled beat data

Type BeatData

sections
human-labeled section data

Type SectionData

to_jams()
Jams: the track's data in jams format

mirdata.rwc_jazz.**download**(*data_home=None*, *partial_download=None*, *force_overwrite=False*, *cleanup=True*)
Download the RWC Jazz (annotations and metadata). The audio files are not provided due to copyright issues.

Parameters

- **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*
- **force_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **partial_download** (*list*) – List indicating what to partially download. The list can include any of: * *'annotations_beat'* the beat annotation files * *'annotations_sections'* the sections annotation files * *'metadata'* the metadata files If *None*, all data is downloaded.
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

`mirdata.rwc_jazz.load(data_home=None)`

Load RWC-Jazz dataset

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns {*track_id*: track data}

Return type (dict)

`mirdata.rwc_jazz.track_ids()`

Return track ids

Returns A list of track ids

Return type (list)

`mirdata.rwc_jazz.validate(data_home=None, silence=False)`

Validate if the stored dataset is a valid version

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns

List of file paths that are in the dataset index but missing locally

invalid_checksums (list): List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

Return type missing_files (list)

mirdata.rwc_popular

RWC Popular Dataset Loader

The Popular Music Database consists of 100 songs — 20 songs with English lyrics performed in the style of popular music typical of songs on the American hit charts in the 1980s, and 80 songs with Japanese lyrics performed in the style of modern Japanese popular music typical of songs on the Japanese hit charts in the 1990s.

For more details, please visit: https://staff.aist.go.jp/m.goto/RWC-MDB/rwc-mdb-p.html

class `mirdata.rwc_popular.Track(track_id, data_home=None)`

rwc_popular Track class

Parameters

- **track_id** (*str*) – track id of the track
- **data_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir_datasets*

artist
artist

Type str

audio_path
path of the audio file

Type str

beats_path
path of the beat annotation file

Type str

chords_path
path of the chord annotation file

Type str

drum_information
If the drum is 'Drum sequences', 'Live drums', or 'Drum loops'

Type str

duration
Duration of the track in seconds

Type float

instruments
List of used instruments

Type str

piece_number
Piece number, [1-50]

Type str

sections_path
path of the section annotation file

Type str

singer_information
TODO

Type str

suffix
M01-M04

Type str

tempo
Tempo of the track in BPM

Type str

title
title

Type str

track_id
track id

Type str

track_number
CD track number

Type str

voca_inst_path
path of the vocal/instrumental annotation file

Type str

audio
audio signal, sample rate

Type (np.ndarray, float)

beats
human-labeled beat annotation

Type BeatData

chords
human-labeled chord annotation

Type ChordData

sections
human-labeled section annotation

Type SectionData

to_jams()
Jams: the track's data in jams format

vocal_instrument_activity
human-labeled vocal/instrument activity

Type EventData

mirdata.rwc_popular.**download**(*data_home=None*, *partial_download=None*, *force_overwrite=False*, *cleanup=True*)
Download the RWC Popular (annotations and metadata). The audio files are not provided due to copyright issues.

Parameters

- **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*
- **force_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **partial_download** (*list*) – List indicating what to partially download. The list can include any of: * *'annotations_beat'* the beat annotation files * *'annotations_sections'* the sections annotation files * *'annotations_chords'* the chords annotation files * *'annotations_vocal_act'* the vocal activity annotation files * *'metadata'* the metadata files If *None*, all data is downloaded.
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

mirdata.rwc_popular.**load**(*data_home=None*)
Load RWC-Genre dataset

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns {*track_id*: track data}

Return type (dict)

`mirdata.rwc_popular.track_ids()`
Return track ids

Returns A list of track ids

Return type (list)

`mirdata.rwc_popular.validate(data_home=None, silence=False)`
Validate if the stored dataset is a valid version

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns

List of file paths that are in the dataset index but missing locally

invalid_checksums (list): List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

Return type missing_files (list)

mirdata.salami

SALAMI Dataset Loader

The SALAMI dataset contains Structural Annotations of a Large Amount of Music Information: the public portion contains over 2200 annotations of over 1300 unique tracks.

NB: mirdata relies on the **corrected** version of the 2.0 annotations: Details can be found at https://github.com/bmcfee/salami-data-public/tree/hierarchy-corrections and https://github.com/DDMAL/salami-data-public/pull/15.

For more details, please visit: https://github.com/DDMAL/salami-data-public

class `mirdata.salami.Track(track_id, data_home=None)`
salami Track class

Parameters

- **track_id** (*str*) – track id of the track
- **data_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir_datasets*

annotator_1_id
number that identifies annotator 1

Type str

annotator_1_time
time that the annotator 1 took to complete the annotation

Type str

annotator_2_id
number that identifies annotator 1

Type str

annotator_2_time
time that the annotator 1 took to complete the annotation

Type str

artist
song artist

Type str

audio_path
path to the audio file

Type str

broad_genre
broad genre of the song

Type str

duration
duration of song in seconds

Type float

genre
genre of the song

Type str

sections_annotator1_lowercase_path
path to annotations in hierarchy level 1 from annotator 1

Type str

sections_annotator1_uppercase_path
path to annotations in hierarchy level 0 from annotator 1

Type str

sections_annotator2_lowercase_path
path to annotations in hierarchy level 1 from annotator 2

Type str

sections_annotator2_uppercase_path
path to annotations in hierarchy level 0 from annotator 2

Type str

source
dataset or source of song

Type str

title
title of the song

Type str

audio
audio signal, sample rate

Type (np.ndarray, float)

sections_annotator_1_lowercase
annotations in hierarchy level 1 from annotator 1

Type SectionData

sections_annotator_1_uppercase
annotations in hierarchy level 0 from annotator 1

Type SectionData

sections_annotator_2_lowercase
annotations in hierarchy level 1 from annotator 2

Type SectionData

sections_annotator_2_uppercase
annotations in hierarchy level 0 from annotator 2

Type SectionData

to_jams()
Jams: the track's data in jams format

mirdata.salami.**cite**()
Print the reference

mirdata.salami.**download**(*data_home=None*, *force_overwrite=False*, *cleanup=True*)
Download SALAMI Dataset (annotations). The audio files are not provided.

Parameters

- **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*
- **force_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

mirdata.salami.**load**(*data_home=None*)
Load SALAMI dataset

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns {*track_id*: track data}

Return type (dict)

mirdata.salami.**load_audio**(*audio_path*)
Load a Salami audio file.

Parameters **audio_path** (*str*) – path to audio file

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

mirdata.salami.**track_ids**()
Return track ids

Returns A list of track ids

Return type (list)

mirdata.salami.**validate**(*data_home=None*, *silence=False*)
Validate if the stored dataset is a valid version

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns

List of file paths that are in the dataset index but missing locally

invalid_checksums (list): List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

Return type missing_files (list)

mirdata.tinysol

TinySOL Dataset Loader.

TinySOL is a dataset of 2913 samples, each containing a single musical note from one of 14 different instruments:

> Bass Tuba French Horn Trombone Trumpet in C Accordion Contrabass Violin Viola Violoncello Bassoon Clarinet in B-flat Flute Oboe Alto Saxophone

These sounds were originally recorded at Ircam in Paris (France) between 1996 and 1999, as part of a larger project named Studio On Line (SOL). Although SOL contains many combinations of mutes and extended playing techniques, TinySOL purely consists of sounds played in the so-called "ordinary" style, and in absence of mute.

TinySOL can be used for education and research purposes. In particular, it can be employed as a dataset for training and/or evaluating music information retrieval (MIR) systems, for tasks such as instrument recognition or fundamental frequency estimation. For this purpose, we provide an official 5-fold split of TinySOL as a metadata attribute. This split has been carefully balanced in terms of instrumentation, pitch range, and dynamics. For the sake of research reproducibility, we encourage users of TinySOL to adopt this split and report their results in terms of average performance across folds.

We encourage TinySOL users to subscribe to the Ircam Forum so that they can have access to larger versions of SOL.

For more details, please visit: https://www.orch-idea.org/

class `mirdata.tinysol.`**`Track`**(*track_id*, *data_home=None*)

tinysol Track class

Parameters

- **`track_id`** (*`str`*) – track id of the track
- **`data_home`** (*`str`*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir_datasets*

`audio_path`

path of the audio file

Type str

`dynamics`

dynamics abbreviation. Ex: pp, mf, ff, etc.

Type str

`dynamics_id`

pp=0, p=1, mf=2, f=3, ff=4

Type int

`family`

instrument family encoded by its English name

Type str

`instance_id`

instance ID. Either equal to 0, 1, 2, or 3.

Type int

instrument_abbr
instrument abbreviation

Type str

instrument_full
instrument encoded by its English name

Type str

is_resampled
True if this sample was pitch-shifted from a neighbor; False if it was genuinely recorded.

Type bool

pitch
string containing English pitch class and octave number

Type str

pitch_id
MIDI note index, where middle C ("C4") corresponds to 60

Type int

string_id
string ID. By musical convention, the first string is the highest. On wind instruments, this is replaced by *None*.

Type NoneType

technique_abbr
playing technique abbreviation

Type str

technique_full
playing technique encoded by its English name

Type str

track_id
track id

Type str

audio
audio signal, sample rate

Type (np.ndarray, float)

to_jams()
Jams: the track's data in jams format

mirdata.tinysol.**cite**()
Print the reference

mirdata.tinysol.**download**(*data_home=None*, *partial_download=None*, *force_overwrite=False*, *cleanup=True*)
Download TinySOL.

Parameters

- **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

- **force_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **partial_download** (*list*) – List indicating what to partially download. The list can include any of: * *'annotations'* the annotation files * *'audio'* the audio files If *None*, all data is downloaded.
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

`mirdata.tinysol.load(data_home=None)`

Load TinySOL :param data_home: Local path where TinySOL is stored.

If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns {*track_id*: track data}

Return type (dict)

`mirdata.tinysol.load_audio(audio_path)`

Load a TinySOL audio file.

Parameters **audio_path** (*str*) – path to audio file

Returns the mono audio signal sr (float): The sample rate of the audio file

Return type y (np.ndarray)

`mirdata.tinysol.track_ids()`

Return track ids

Returns A list of track ids

Return type (list)

`mirdata.tinysol.validate(data_home=None, silence=False)`

Validate if the stored dataset is a valid version

Parameters **data_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir_datasets*

Returns

List of file paths that are in the dataset index but missing locally

invalid_checksums (list): List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

Return type missing_files (list)

2.3.2 Utilities

mirdata.track

track object utility functions

mirdata.utils

Utility functions for mirdata

`mirdata.utils.MIR_DATASETS_DIR`

home folder for MIR datasets

Type str

mirdata.utils.**NoteData**
intervals, *notes*, *confidence*

Type namedtuple

mirdata.utils.**F0Data**
times, *frequencies*, *confidence*

Type namedtuple

mirdata.utils.**LyricData**
start_times, *end_times*, *lyrics*, *pronounciations*

Type namedtuple

mirdata.utils.**SectionData**
start_times, *end_times*, *sections*

Type namedtuple

mirdata.utils.**BeatData**
beat_times, *beat_positions*

Type namedtuple

mirdata.utils.**ChordData**
start_times, *end_times*, *chords*

Type namedtuple

mirdata.utils.**KeyData**
start_times, '*end_times*, *keys*

Type namedtuple

mirdata.utils.**EventData**
start_times, *end_times*, *event*

Type namedtuple

mirdata.utils.**TempoData**
time, *duration*, *value*, *confidence*

Type namedtuple

class mirdata.utils.**BeatData**(*beat_times*, *beat_positions*)

beat_positions
Alias for field number 1

beat_times
Alias for field number 0

class mirdata.utils.**ChordData**(*intervals*, *labels*)

intervals
Alias for field number 0

labels
Alias for field number 1

class mirdata.utils.**EventData**(*start_times*, *end_times*, *event*)

end_times
Alias for field number 1

event
Alias for field number 2

start_times
Alias for field number 0

class mirdata.utils.**F0Data**(*times*, *frequencies*, *confidence*)

confidence
Alias for field number 2

frequencies
Alias for field number 1

times
Alias for field number 0

class mirdata.utils.**KeyData**(*start_times*, *end_times*, *keys*)

end_times
Alias for field number 1

keys
Alias for field number 2

start_times
Alias for field number 0

class mirdata.utils.**LyricData**(*start_times*, *end_times*, *lyrics*, *pronunciations*)

end_times
Alias for field number 1

lyrics
Alias for field number 2

pronunciations
Alias for field number 3

start_times
Alias for field number 0

class mirdata.utils.**MultipitchData**(*times*, *frequency_list*, *confidence_list*)

confidence_list
Alias for field number 2

frequency_list
Alias for field number 1

times
Alias for field number 0

class mirdata.utils.**NoteData**(*intervals*, *notes*, *confidence*)

confidence
Alias for field number 2

intervals
Alias for field number 0

notes
Alias for field number 1

class mirdata.utils.**SectionData**(*intervals*, *labels*)

intervals
Alias for field number 0

labels
Alias for field number 1

class mirdata.utils.**TempoData**(*time*, *duration*, *value*, *confidence*)

confidence
Alias for field number 3

duration
Alias for field number 1

time
Alias for field number 0

value
Alias for field number 2

class mirdata.utils.**cached_property**(*func*)
A property that is only computed once per instance and then replaces itself with an ordinary attribute. Deleting the attribute resets the property. Source: https://github.com/bottlepy/bottle/commit/fa7733e075da0d790d809aa3d2f53071897e6f76

mirdata.utils.**check_index**(*dataset_index*, *data_home*)
check index to find out missing files and files with invalid checksum

Parameters

- **dataset_index** (*list*) – dataset indices
- **data_home** (*str*) – Local home path that the dataset is being stored

Returns

List of file paths that are in the dataset index but missing locally

invalid_checksums (list): List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

Return type missing_files (list)

mirdata.utils.**get_default_dataset_path**(*dataset_name*)
Get the default path for a dataset given it's name

Parameters **dataset_name** (*str or None*) – The name of the dataset folder, e.g. 'Orchset'

Returns Local path to the dataset

Return type save_path (str)

`mirdata.utils.`**`log_message`**(*message*, *silence=False*)
Helper function to log message

Parameters

- **`message`** (*`str`*) – message to log
- **`silence`** (*`bool`*) – if true, the message is not logged

`mirdata.utils.`**`md5`**(*file_path*)
Get md5 hash of a file.

Parameters **`file_path`** (*`str`*) – File path

Returns md5 hash of data in file_path

Return type md5_hash (str)

`mirdata.utils.`**`none_path_join`**(*partial_path_list*)
Join a list of partial paths. If any part of the path is None, returns None.

Parameters **`partial_path_list`** (*`list`*) – List of partial paths

Returns joined path string or None

Return type path or None (str or None)

`mirdata.utils.`**`validator`**(*dataset_index*, *data_home*, *silence=False*)
Checks the existence and validity of files stored locally with respect to the paths and file checksums stored in the reference index. Logs invalid checksums and missing files.

Parameters

- **`dataset_index`** (*`list`*) – dataset indices
- **`data_home`** (*`str`*) – Local home path that the dataset is being stored
- **`silence`** (*`bool`*) – if False (default), prints missing and invalid files
- **`stdout. Otherwise, this function is equivalent to check_index.`** (*`to`*) –

Returns

List of file paths that are in the dataset index but missing locally.

invalid_checksums (list): List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum.

Return type missing_files (list)

mirdata.download_utils

functions for downloading from the web

`mirdata.download_utils.`**`RemoteFileMetadata`**
It specifies the metadata of the remote file to download. The metadata consists of *filename*, *url*, *checksum*, and *destination_dir*.

Type namedtuple

class mirdata.download_utils.**DownloadProgressBar**(*iterable=None, desc=None, total=None, leave=True, file=None, ncols=None, mininterval=0.1, maxinterval=10.0, miniters=None, ascii=None, disable=False, unit='it', unit_scale=False, dynamic_ncols=False, smoothing=0.3, bar_format=None, initial=0, position=None, postfix=None, unit_divisor=1000, write_bytes=None, lock_args=None, nrows=None, colour=None, gui=False, **kwargs*)

Wrap *tqdm* to show download progress

class mirdata.download_utils.**RemoteFileMetadata**(*filename, url, checksum, destination_dir*)

checksum
Alias for field number 2

destination_dir
Alias for field number 3

filename
Alias for field number 0

url
Alias for field number 1

mirdata.download_utils.**download_from_remote**(*remote, save_dir, force_overwrite=False*)

Download a remote dataset into path Fetch a dataset pointed by remote's url, save into path using remote's filename and ensure its integrity based on the MD5 Checksum of the downloaded file.

Adapted from scikit-learn's sklearn.datasets.base._fetch_remote.

Parameters

- **remote** (*RemoteFileMetadata*) – Named tuple containing remote dataset meta information: url, filename and checksum
- **save_dir** (*str*) – Directory to save the file to. Usually *data_home*
- **force_overwrite** (*bool*) – If True, overwrite existing file with the downloaded file. If False, does not overwrite, but checks that checksum is consistent.

Returns Full path of the created file.

Return type file_path (str)

mirdata.download_utils.**download_tar_file**(*tar_remote, save_dir, force_overwrite, cleanup=True*)

Download and untar a tar file.

Parameters

- **tar_remote** (*RemoteFileMetadata*) – Object containing download information
- **save_dir** (*str*) – Path to save downloaded file
- **force_overwrite** (*bool*) – If True, overwrites existing files
- **cleanup** (*bool*) – If True, remove tarfile after untarring. Default=False

`mirdata.download_utils.download_zip_file(zip_remote, save_dir, force_overwrite, cleanup=True)`

Download and unzip a zip file.

Parameters

- **zip_remote** (*RemoteFileMetadata*) – Object containing download information
- **save_dir** (*str*) – Path to save downloaded file
- **force_overwrite** (*bool*) – If True, overwrites existing files
- **cleanup** (*bool*) – If True, remove zipfile after unziping. Default=False

`mirdata.download_utils.downloader(save_dir, remotes=None, partial_download=None, info_message=None, force_overwrite=False, cleanup=True)`

Download data to *save_dir* and optionally print a message.

Parameters

- **save_dir** (*str*) – The directory to download the data
- **remotes** (*dict or None*) – A dictionary of RemoteFileMetadata tuples of data in zip format. If None, there is no data to download
- **partial_download** (*list or None*) – A list of keys to partially download the remote objects of the download dict. If None, all data is downloaded
- **info_message** (*str or None*) – A string of info to print when this function is called. If None, no string is printed.
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

`mirdata.download_utils.untar(tar_path, cleanup=True)`

Untar a tar file inside it's current directory.

Parameters

- **tar_path** (*str*) – Path to tar file
- **cleanup** (*bool*) – If True, remove tarfile after untarring. Default=False

`mirdata.download_utils.unzip(zip_path, cleanup=True)`

Unzip a zip file inside it's current directory.

Parameters

- **zip_path** (*str*) – Path to zip file
- **cleanup** (*bool*) – If True, remove zipfile after unzipping. Default=False

mirdata.jams_utils

functions for converting mirdata annotations to jams format

`mirdata.jams_utils.beats_to_jams(beats)`

Convert beats annotations into jams format.

Parameters **beats** (*tuple*) – A tuple in the format (BeatData, str), where str describes the annotation and BeatData is the beats mirdata annotation format.

Returns **jannot_beat**

Return type JAM beat annotation object.

`mirdata.jams_utils.`**`chords_to_jams`**(*chords*)

Convert chords annotations into jams format.

Parameters **chords** (*tuple*) – A tuple in the format (ChordData, str), where str describes the annotation and ChordData is the chords mirdata annotation format.

Returns **jannot_chord**

Return type JAM chord annotation object.

`mirdata.jams_utils.`**`events_to_jams`**(*events*)

Convert events annotations into jams format.

Parameters **events** (*tuple*) – A tuple in the format (EventData, str), where str describes the annotation and EventData is the events mirdata annotation format.

Returns **jannot_events**

Return type JAM tag_open annotation object.

`mirdata.jams_utils.`**`f0s_to_jams`**(*f0s*)

Convert f0 annotations into jams format.

Parameters **f0s** (*tuple*) – A tuple in the format (F0Data, str), where str describes the annotation and F0Data is the f0 mirdata annotation format.

Returns **jannot_f0**

Return type JAM pitch_contour annotation object.

`mirdata.jams_utils.`**`jams_converter`**(*audio_path=None*, *beat_data=None*, *chord_data=None*, *note_data=None*, *f0_data=None*, *section_data=None*, *multi_section_data=None*, *tempo_data=None*, *event_data=None*, *key_data=None*, *lyrics_data=None*, *tags_gtzan_data=None*, *metadata=None*)

Convert annotations from a track to JAMS format.

Parameters

- **(str or None)** (*audio_path*) – A path to the corresponding audio file, or None. If provided, the audio file will be read to compute the duration. If None, 'duration' must be a field in the metadata dictionary, or the resulting jam object will not validate.
- **(list or None)** (*tags_gtzan_data*) – A list of tuples of (BeatData, str), where str describes the annotation (e.g. 'beats_1').
- **(list or None)** – A list of tuples of (ChordData, str), where str describes the annotation.
- **(list or None)** – A list of tuples of (NoteData, str), where str describes the annotation.
- **(list or None)** – A list of tuples of (F0Data, str), where str describes the annotation.
- **(list or None)** – A list of tuples of (SectionData, str), where str describes the annotation.
- **(list or None)** – A list of tuples. Tuples in multi_section_data should contain another list of tuples, indicating annotations in the different levels e.g. ([(segments0, level0), '(segments1, level1)], annotator) and a str indicating the annotator
- **(list or None)** – A list of tuples of (float, str), where float gives the tempo in bpm and str describes the annotation.

- **(list or None)** – A list of tuples of (EventData, str), where str describes the annotation.
- **(list or None)** – A list of tuples of (KeyData, str), where str describes the annotation.
- **(list or None)** – A list of tuples of (LyricData, str), where str describes the annotation.
- **(list or None)** – A list of tuples of (str, str), where the first srt is the tag and the second is a descriptor of the annotation.
- **(dict or None)** (*metadata*) – A dictionary containing the track metadata.

Returns **jam** – A JAM object with all the annotations.

Return type JAM object

`mirdata.jams_utils.`**`keys_to_jams`**(*keys*)

Convert keys annotations into jams format.

Parameters **keys** (*tuple*) – A tuple in the format (KeyData, str), where str describes the annotation and KeyData is the keys mirdata annotation format.

Returns **jannot_key**

Return type JAM key_mode annotation object.

`mirdata.jams_utils.`**`lyrics_to_jams`**(*lyrics*)

Convert lyrics annotations into jams format.

Parameters **lyrics** (*tuple*) – A tuple in the format (LyricData, str), where str describes the annotation and LyricData is the lyric mirdata annotation format.

Returns **jannot_lyric**

Return type JAM lyric annotation object.

`mirdata.jams_utils.`**`multi_sections_to_jams`**(*multi_sections*)

Convert hierarchical annotations into jams format.

Parameters **multi_segment** (*list*) – A list of tuples in the format ([(segments0, level0), (segments1, level1)], annotator), where segments are SectionData mirdata format, level indicates the hierarchy (e.g. 0, 1) and annotator describes the annotator. This format is customize for Salami dataset annotations.

Returns **jannot_multi**

Return type JAM multi_segment annotation object.

`mirdata.jams_utils.`**`notes_to_jams`**(*notes*)

Convert notes annotations into jams format using note_to_midi from librosa.

Parameters **notes** (*tuple*) – A tuple in the format (NoteData, str), where str describes the annotation and NoteData is the notes mirdata annotation format.

Returns **jannot_notes**

Return type JAM note_midi annotation object.

`mirdata.jams_utils.`**`sections_to_jams`**(*sections*)

Convert sections annotations into jams format.

Parameters **sections** (*tuple*) – A tuple in the format (SectionData, str), where str describes the annotation and SectionData is the sections mirdata annotation format.

Returns **jannot_seg**

Return type JAM segment_open annotation object.

`mirdata.jams_utils.tag_gtzan_to_jams(tags)`

Convert tag-gtzan annotations into jams format.

Parameters **tags** (*tuple*) – A tuple in the format (str, str), where the first str is the tag and the second describes the annotation.

Returns **jannot_tag_gtzan**

Return type JAM tag_gtzan annotation object.

`mirdata.jams_utils.tempos_to_jams(tempos)`

Convert tempo annotations into jams format.

Parameters **tempo** (*tuple*) – A tuple in the format (float, str), where str describes the annotation and float is the tempo in beats per minute.

Returns **jannot_tempo**

Return type JAM tempo annotation object.

2.4 FAQ

2.4.1 How do I add a new loader?

Take a look at our instructions!

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

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.

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

No, we do not host or distribute datasets.

2.4.4 How do I request a new dataset?

Open an issue and tag it with the "New Loader" label.

2.4.5 What do I do if my data fails validation?

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.

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

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.

2.4.7 Does mirdata provide data loaders for pytorch/Tensorflow?

For now, no. Music datasets are very widely varied in their annotation types and supported tasks. To make a data loader, there would need to be "standard" ways to encode the desired inputs/outputs - unofortunately this is not universal for most datasets and usages. Still, this library provides the necessary first step for building data loaders and it is easy to build data loaders on top of this. For a simple example, see our examples page.

2.4.8 Why didn't you release a version of this library in MATLAB/C/Java/R?

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.

2.4.9 A download link is broken for a loader's `.download()` function. What do I do?

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

2.4.10 Why the name, mirdata?

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

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

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.

2.4.12 Does mirdata support data which lives off-disk?

Yes. While the simple useage of mirdata assumes that data lives on-disk, it can be used for off-disk data as well. See the "local vs remote" example in the examples page for details.

CHAPTER 3

Contribute

- Issue Tracker
- Source Code

Python Module Index

m

Index

A

B

C

D

E

F

G

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Y