soundata

Release 0.1.3

The Soundata development team

Feb 06, 2024

CONTENTS

Soundata is a Python library for loading and working with audio datasets in a standardized way, removing the need for writing custom loaders in every project, and improving reproducibility by providing tools validate data against a canonical version. It speeds up research pipelines by allowing you to quickly `download` a dataset, `load` it into memory in a `standardized` and `reproducible` way, `validate` that the dataset is complete and correct, and more.

Soundata is based and inspired on mirdata, and was created following these desig principles:

- **Easy to use:** Soundata is designed to be easy to use and to simplify the research pipeline considerably. Check out the examples in the *Getting started* page.
- **Easy to contribute to:** we welcome and encourage contributions, especially new datasets. You can contribute following the instructions in our *Contributing* page.
- **Increase reproducibility:** by providing a common framework for researchers to compare and validate their data, when mistakes are found in annotations or audio versions change, using Soundata the audio community can fix mistakes while still being able to compare methods moving forward.
- **Standardize usage of sound datasets:** we standardized common attributes of sound datasets such as `audio` or `tags` to simplify audio research pipelines, while preserving each dataset's idiosyncracies: if a dataset has 'non-standard' attributes, we include them as well.

CHAPTER

ONE

INSTALLATION

To install Soundata simply do:

```
pip install soundata
```

We recommend to do this inside a conda or virtual environment for reproducibility.

CHAPTER

TWO

EXAMPLE USAGE

```
import soundata

# learn wich datasets are available in soundata
print(soundata.list_datasets())

# choose a dataset and download it
dataset = soundata.initialize('urbansound8k', data_home='choose_where_data_live')
dataset.download()

# get annotations and audio for a random clip
example_clip = dataset.choice_clip()
tags = example_clip.tags
y, sr = example_clip.audio
```

CHAPTER

THREE

CONTRIBUTING

We welcome and encourage contributions to soundata, especially new dataset loaders. To contribute a new loader, please follow the steps indicated in the *Contributing* page and submit a Pull Request (PR) to the github repository. For any doubt or comment about your contribution, you can always submit an issue or open a discussion in the repository.

- Issue Tracker
- Source Code

CHAPTER

FOUR

CITING SOUNDATA

TBA

4.1 Getting started

4.1.1 Installation

To install Soundata simply do:

```
pip install soundata
```

Soundata is easily imported into your Python code by:

```
import soundata
```

4.1.2 Initializing a dataset

Print a list of all available dataset loaders by calling:

```
import soundata
print(soundata.list_datasets())
```

To use a loader, (for example, `urbansound8k`) you need to initialize it by calling:

```
import soundata
dataset = soundata.initialize('urbansound8k', data_home='choose_where_data_live')
```

You can indicate where the data would be stored and access by passing a path to `data_home`, as explained below. Now `us8k` is a `Dataset` object containing common methods, described in the following.

4.1.3 Downloading a dataset

All dataset loaders in soundata have a `download()` function that allows the user to download the *canonical* version of the dataset (when available). When initializing a dataset it is important to correctly set up the directory where the dataset is going to be stored and retrieved.

Downloading a dataset into the default folder:

In this first example, `data_home` is not specified. Thus, UrbanSound8K will be downloaded and retrieved from the default folder, `sound_datasets`, created in the user's root folder:

```
import soundata
dataset = soundata.initialize('urbansound8k')
dataset.download()  # Dataset is downloaded into "sound_datasets" folder inside user
↪'s root folder
```

Downloading a dataset into a specified folder:

In the next example `data_home` is specified, so UrbanSound8K will be downloaded and retrieved from the specified location:

```
dataset = soundata.initialize('urbansound8k', data_home='Users/johnsmith/Desktop')
dataset.download()  # Dataset is downloaded to John Smith's desktop
```

Partially downloading a dataset

The `download()` function allows to partially download a dataset. In other words, if applicable, the user can select which elements of the dataset they want to download. Each dataset has a `REMOTES` dictionary were all the available downloadable elements are listed.

`tau2019uas` has different elements as seen in the `REMOTES` dictionary. You can specify a subset of these elements to download by passing the `download()` function a list of the `REMOTES` keys that we are interested in via the `partial_download` variable.

Example REMOTES

```
REMOTES = {
"development.audio.1": download_utils.RemoteFileMetadata(
    filename="TAU-urban-acoustic-scenes-2019-development.audio.1.zip",
    url="https://zenodo.org/record/2589280/files/TAU-urban-acoustic-scenes-2019-
↪development.audio.1.zip?download=1",
    checksum="aca4ebfd9ed03d5f747d6ba8c24bc728",
),
"development.audio.2": download_utils.RemoteFileMetadata(
    filename="TAU-urban-acoustic-scenes-2019-development.audio.2.zip",
    url="https://zenodo.org/record/2589280/files/TAU-urban-acoustic-scenes-2019-
↪development.audio.2.zip?download=1",
    checksum="c4f170408ce77c8c70c532bf268d7be0",
),
"development.audio.3": download_utils.RemoteFileMetadata(
    filename="TAU-urban-acoustic-scenes-2019-development.audio.3.zip",
    url="https://zenodo.org/record/2589280/files/TAU-urban-acoustic-scenes-2019-
↪development.audio.3.zip?download=1",
    checksum="c7214a07211f10f3250290d05e72c37e",
),
....
```

A partial download example for `tau2019uas` dataset could be:

```
dataset = soundata.initialize('tau2019uas')
dataset.download(partial_download=['development.audio.1', 'development.audio.2'])
↪# download only two remotes
```

Downloading a multipart dataset

In some cases, datasets consist of multiple remote files that have to be extracted together locally to correctly recover the data. In those cases, remotes that need to be extracted together should be grouped in a list, so all the necessary files are downloaded at once (even in a partial download). An example of this is the *fsd50k* loader:

Example multipart REMOTES

```
REMOTES = {
    "FSD50K.dev_audio": [
        download_utils.RemoteFileMetadata(
            filename="FSD50K.dev_audio.zip",
            url="https://zenodo.org/record/4060432/files/FSD50K.dev_audio.zip?
↪download=1",
            checksum="c480d119b8f7a7e32fdb58f3ea4d6c5a",
        ),
        download_utils.RemoteFileMetadata(
            filename="FSD50K.dev_audio.z01",
            url="https://zenodo.org/record/4060432/files/FSD50K.dev_audio.z01?
↪download=1",
            checksum="faa7cf4cc076fc34a44a479a5ed862a3",
        ),
        download_utils.RemoteFileMetadata(
            filename="FSD50K.dev_audio.z02",
            url="https://zenodo.org/record/4060432/files/FSD50K.dev_audio.z02?
↪download=1",
            checksum="8f9b66153e68571164fb1315d00bc7bc",
        ),
        download_utils.RemoteFileMetadata(
            filename="FSD50K.dev_audio.z03",
            url="https://zenodo.org/record/4060432/files/FSD50K.dev_audio.z03?
↪download=1",
            checksum="1196ef47d267a993d30fa98af54b7159",
        ),
        download_utils.RemoteFileMetadata(
            filename="FSD50K.dev_audio.z04",
            url="https://zenodo.org/record/4060432/files/FSD50K.dev_audio.z04?
↪download=1",
            checksum="d088ac4e11ba53daf9f7574c11cccac9",
        ),
        download_utils.RemoteFileMetadata(
            filename="FSD50K.dev_audio.z05",
            url="https://zenodo.org/record/4060432/files/FSD50K.dev_audio.z05?
↪download=1",
            checksum="81356521aa159accd3c35de22da28c7f",
        ),
    ],
    ...
```

4.1.4 Validating a dataset

Using the `validate()` method you can ensure that the files in our local copy of a dataset are identical to the *canonical* version of the dataset. The function computes the md5 checksum of every downloaded file to ensure it was downloaded correctly and isn't corrupted.

For big datasets: In future `soundata` versions, a random validation will be included. This improvement will reduce validation time for very big datasets.

4.1.5 Accessing annotations

You can choose a random clip from a dataset with the `choice_clip()` method.

Example Index

```
dataset = soundata.initialize('urbansed')
random_clip = dataset.choice_clip()
print(random_clip)
>>> Clip(
      audio_path="/Users/theuser/sound_datasets/urbansed/audio/test/soundscape_test_
↪bimodal73.wav",
      clip_id="soundscape_test_bimodal73",
      jams_path="/Users/mf3734/sound_datasets/urbansed/annotations/test/soundscape_test_
↪bimodal73.jams",
      txt_path="/Users/mf3734/sound_datasets/urbansed/annotations/test/soundscape_test_
↪bimodal73.txt",
      audio: The clips audio
                * np.ndarray - audio signal
                * float - sample rate,
      events: The audio events
                * annotations.Events - audio event object,
      split: The data splits (e.g. train)
                * str - split,
   )
```

You can also access specific clips by id. The available clip ids can be acessed by doing `dataset.clip_ids`. In the next example we take the first clip id, and then we retrieve its `tags` annotation.

```
dataset = soundata.initialize('urbansound8k')
ids = dataset.clip_ids  # the list of urbansound8k's clip ids
clips = dataset.load_clips()  # Load all clips in the dataset
example_clip = clips[ids[0]]  # Get the first clip

# Accessing the clip's tags annotation
example_tags = example_clip.tags
print(example_tags)
>>>> Tags(confidence, labels, labels_unit)
print(example_tags.labels)
>>>> ['children_playing']
```

You can also load a single clip without loading all clips in the dataset:

```
ids = dataset.clip_ids  # the list of urbansound8k's clip ids
example_clip = dataset.clip(ids[0])  # load this particular clip
example_tags = example_clip.tags  # Get the tags for the first clip
```

4.1.6 Accessing data remotely

Annotations can also be accessed through `load_*()` methods which may be useful, for instance, when your data aren't available locally. If you specify the annotation's path, you can use the module's loading functions directly. Let's see an example.

Accessing annotations remotely example

```
# Load list of clip ids of the dataset
ids = dataset.clip_ids

# Load a single clip, specifying the remote location
example_clip = dataset.clip(ids[0], data_home='remote/data/path')
audio_path = example_clip.audio_path

print(audio_path)
>>> remote/data/path/audio/fold1/135776-2-0-49.wav
print(os.path.exists(audio_path))
>>> False

# Write code here to download the remote 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 your local 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

# Get path to where your data live
temp_path = my_downloader(audio_path)

# Accessing the clip audio
example_audio = dataset.load_audio(temp_path)
```

4.1.7 Annotation classes

soundata defines annotation-specific data classes such as *Tags* or *Events*. These data classes are meant to standarize the format for all loaders, so you can use the same code with different datasets. The list and descriptions of available annotation classes can be found in *Annotations*.

Note: These classes are standarized to the point that the data allow for it. In some cases where the dataset has its own idiosyncrasies, the classes may be extended e.g. adding a customize, uncommon attribute.

4.1.8 Iterating over datasets and annotations

In general, most datasets are a collection of clips, and in most cases each clip has an audio file along with annotations.

With the `load_clips()` method, all clips are loaded as a dictionary with the clip id as keys and clip objects as values. The clip objects include their respective audio and annotations, which are lazy-loaded on access to keep things speedy and memory efficient.

```
dataset = soundata.initialize('urbansound8k')
for key, clip in dataset.load_clips().items():
    print(key, clip.audio_path)
>>>> soundscape_train_bimodal0 /Users/mf3734/sound_datasets/urbansed/audio/train/
↪soundscape_train_bimodal0.wav
     .....
```

Alternatively, you can loop over the `clip_ids` list to directly access each clip in the dataset.

```
dataset = soundata.initialize('urbansound8k')
for clip_id in dataset.clip_ids:
    print(clip_id, dataset.clip(clip_id).audio_path)
>>>> soundscape_train_bimodal0 /Users/mf3734/sound_datasets/urbansed/audio/train/
↪soundscape_train_bimodal0.wav
     .....
```

4.1.9 Including soundata in your pipeline

If you wanted to use `urbansound8k` to evaluate the performance of an urban sound classifier, (in our case, `random_classifier`), and then split the scores based on the metadata, you could do the following:

---

soundata usage example

```
import sed_eval
import soundata
import numpy as np
from dcase_util.containers import MetaDataContainer, ProbabilityContainer

def random_classifier(classes):
    return [np.random.random(1)[0] for c in classes]

# Evaluate on the full dataset
dataset = soundata.initialize('urbansound8k')
```

(continues on next page)

(continued from previous page)

```
scores = {}
data = dataset.load_clips()

classes = np.unique([c for _, clip_data in data.items() for c in clip_data.tags.labels])
fold = 2  # Choose a fold to evaluate

ref_tags, est_tags, est_tag_probs = [], [], []
for id, clip in data.items():
    if clip.fold == 2:
        ref_tags.append({'filename': id, 'tags': clip.tags.labels[0]})  # Urbansound8k␣
↪has one label per clip
        probs = random_classifier(classes)
        for c, p in zip(classes, probs):
            est_tag_probs.append({'filename': id, 'label': c, 'probability': p},)
            if p > 0.5:  # Detection threshold of 0.5
                est_tags.append({'filename': id, 'tags': [c]})

tag_evaluator = sed_eval.audio_tag.AudioTaggingMetrics(tags=MetaDataContainer(ref_tags).
↪unique_tags)
tag_evaluator.evaluate(
    reference_tag_list=MetaDataContainer(ref_tags),
    estimated_tag_list=MetaDataContainer(est_tags),
    estimated_tag_probabilities=ProbabilityContainer(est_tag_probs))
```

This is the result of the example above:

Example result

```
print(tag_evaluator)
>>> Audio tagging metrics
=========================================
  Tags                                 : 10
  Evaluated units                      : 888

  Overall metrics (micro-average)
  =======================================
  F-measure
    F-measure (F1)                     : 9.57 %
    Precision                          : 9.57 %
    Recall                             : 9.57 %
  Equal error rate
    Equal error rate (EER)             : 51.01 %

  Class-wise average metrics (macro-average)
  =======================================
  F-measure
    F-measure (F1)                     : 6.47 %
    Precision                          : 7.54 %
    Recall                             : 9.33 %
  Equal error rate
    Equal error rate (EER)             : 50.95 %
```

(continues on next page)

(continued from previous page)

```
  Class-wise metrics
  ======================================
    Tag                | Nref       Nsys       | F-score     Pre          Rec          | EER
    ------------------ | ---------  ---------  | ---------   ---------    ---------    | -----
↪----
    air_conditioner    | 100        419        | 19.3%       11.9         50.0         | 49.0%
    car_horn           | 42         227        | 4.5%        2.6          14.3         | 54.8%
    children_playing   | 100        126        | 9.7%        8.7          11.0         | 54.0%
    dog_bark           | 100        58         | 13.9%       19.0         11.0         | 47.1%
    drilling           | 100        31         | 9.2%        19.4         6.0          | 52.4%
    engine_idling      | 100        16         | 1.7%        6.2          1.0          | 50.0%
    gun_shot           | 35         7          | 0.0%        0.0          0.0          | 48.1%
    jackhammer         | 120        1          | 0.0%        0.0          0.0          | 52.5%
    siren              | 91         3          | 0.0%        0.0          0.0          | 51.6%
    street_music       | 100        0          | nan%        nan          0.0          | 50.0%
```

4.1.10 Using soundata with tensorflow

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

soundata with tf.data.Dataset example

```
import soundata
import numpy as np
import tensorflow as tf

def data_generator(dataset_name):
    # using the default data_home
    dataset = soundata.initialize(dataset_name)
    ids = dataset.clip_ids()
    for clip_id in ids:
        clip = dataset.clip(clip_id)
        audio_signal, sample_rate = clip.audio
        yield {
            "audio": audio_signal.astype(np.float32),
            "sample_rate": sample_rate,
            "label": clip.tags.labels[0],
            "metadata": {"clip_id": clip.clip_id, "fold": clip.fold}
        }

dataset = tf.data.Dataset.from_generator(
    data_generator('urbansound8k'),
    {
        "audio": tf.float32,
        "sample_rate": tf.float32,
        "label": tf.string,
        "metadata": {'clip_id': tf.string, 'fold': tf.string}
    }
)
```

In future `soundata` versions, generators for Tensorflow and PyTorch will be included out-of-the-box.

4.2 Contributing

We encourage contributions to soundata, especially new dataset loaders. To contribute a new loader, follow the steps indicated below and create a Pull Request (PR) to the github repository. For any doubt or comment about your contribution, you can always submit an issue or open a discussion in the repository.

- Issue Tracker
- Source Code

4.2.1 Quick link to contributing templates

If you're familiar with Soundata's API already, you can find the template files for contributing here, and the loader checklist for submiting your PR here.

4.2.2 Installing soundata for development purposes

To install `soundata` for development purposes:

- First run:

```
git clone https://github.com/soundata/soundata.git
```

- Then, after opening source data library you have to install the dependencies for updating the documentation and running tests:

```
pip install .
pip install ."[tests]"
pip install ."[docs]"
```

We recommend using miniconda or pyenv to manage your Python versions and install all `soundata` requirements. You will want to install the latest supported Python versions (see README.md). Once `conda` or `pyenv` and the Python versions are configured, install `pytest`. Make sure you've installed all the necessary pytest plugins needed (e.g. *pytest-cov*) to automatically test your code successfully.

Before running the tests, make sure to have formatted `soundata/` and `tests/` with `black`.

```
black soundata/ tests/
```

Also, make sure that they pass flake8 and mypy tests specified in lint-python.yml github action workflow.

```
flake8 soundata --count --select=E9,F63,F7,F82 --show-source --statistics
python -m mypy soundata --ignore-missing-imports --allow-subclassing-any
```

Finally, run:

```
pytest -vv --cov-report term-missing --cov-report=xml --cov=soundata tests/ --local
```

All tests should pass!

Note: Soundata assumes that your system has the zip library installed for unzipping files.

4.2.3 Writing a new dataset loader

The steps to add a new dataset loader to `soundata` are:

1. *Create an index*
2. *Create a module*
3. *Add tests*
4. *Submit your loader*

Before starting, if your dataset **is not fully downloadable** you should:

1. Contact the soundata team by opening an issue or PR so we can discuss how to proceed with the closed dataset.
2. Show that the version used to create the checksum is the "canonical" one, either by getting the version from the dataset creator, or by verifying equivalence with several other copies of the dataset.

To reduce friction, we will make commits on top of contributors PRs by default unless the `please-do-not-edit` flag is used.

1. Create an index

Soundata's structure relies on `indexes`. Indexes are dictionaries that contain information about the structure of the dataset which is necessary for the loading and validating functionalities of Soundata. In particular, indexes contain information about the files included in the dataset, their location and checksums, see some example indexes below. To create an index, the necessary steps are:

1. Create a script in `scripts/`, called `make_<datasetname>_index.py`, which generates an index file.
2. Then run the script on the canonical version of the dataset and save the index in `soundata/datasets/indexes/` as `<datasetname>_index.json`.

The function `make_<datasetname>_index.py` should automate the generation of an index by computing the MD5 checksums for given files in a dataset located at data_path. Users can adapt this function to create an index for their dataset by adding their file paths and using the md5 function to generate checksums for their files.

Here's an example of an index to use as a guide:

Example Make Index Script

```
import argparse
import glob
import json
import os
from soundata.validate import md5

DATASET_INDEX_PATH = "../soundata/datasets/indexes/dataset_index.json"


def make_dataset_index(dataset_data_path):
```

(continues on next page)

(continued from previous page)

```
    annotation_dir = os.path.join(dataset_data_path, "annotation")
    annotation_files = glob.glob(os.path.join(annotation_dir, "*.lab"))
    clip_ids = sorted([os.path.basename(f).split(".")[0] for f in annotation_files])

    # top-key level metadata
    metadata_checksum = md5(os.path.join(dataset_data_path, "id_mapping.txt"))
    index_metadata = {"metadata": {"id_mapping": ("id_mapping.txt", metadata_checksum)}}

    # top-key level clips
    index_clips = {}
    for clip_id in clip_ids:
        audio_checksum = md5(
            os.path.join(dataset_data_path, "Wavfile/{}.wav".format(clip_id))
        )
        annotation_checksum = md5(
            os.path.join(dataset_data_path, "annotation/{}.lab".format(clip_id))
        )

        index_clips[clip_id] = {
            "audio": ("Wavfile/{}.wav".format(clip_id), audio_checksum),
            "annotation": ("annotation/{}.lab".format(clip_id), annotation_checksum),
        }

    # top-key level version
    dataset_index = {"version": None}

    # combine all in dataset index
    dataset_index.update(index_metadata)
    dataset_index.update({"clips": index_clips})

    with open(DATASET_INDEX_PATH, "w") as fhandle:
        json.dump(dataset_index, fhandle, indent=2)


def main(args):
    make_dataset_index(args.dataset_data_path)


if __name__ == "__main__":
    PARSER = argparse.ArgumentParser(description="Make dataset index file.")
    PARSER.add_argument(
        "dataset_data_path", type=str, help="Path to dataset data folder."
    )

    main(PARSER.parse_args())
```

More examples of scripts used to create dataset indexes can be found in the scripts folder.

Example index with clips

Most sound datasets are organized as a collection of clips and annotations. In such case, the index should make use of the `clips` top-level key. Under this `clips` top-level key, you should store a dictionary where the keys are the unique clip ids of the dataset, and the values are dictionaries of files associated with a clip id, along with their checksums. These files can be for instance audio files or annotations related to the clip id. File paths are relative to the top level directory of a dataset.

Index Examples - Clips

If the version *1.0* of a given dataset has the structure:

```
> Example_Dataset/
    > audio/
        clip1.wav
        clip2.wav
        clip3.wav
    > annotations/
        clip1.csv
        clip2.csv
        clip3.csv
    > metadata/
        metadata_file.csv
```

The top level directory is `Example_Dataset` and the relative path for `clip1.wav` would be `audio/clip1.wav`. Any unavailable fields are indicated with *null*. A possible index file for this example would be:

```
{   "version": "1.0",
    "clips":
        "clip1": {
            "audio": [
                "audio/clip1.wav",  // the relative path for clip1's audio file
                "912ec803b2ce49e4a541068d495ab570"  // clip1.wav's md5 checksum
            ],
            "annotation": [
                "annotations/clip1.csv",  // the relative path for clip1's annotation
                "2cf33591c3b28b382668952e236cccd5"  // clip1.csv's md5 checksum
            ]
        },
        "clip2": {
            "audio": [
                "audio/clip2.wav",
                "65d671ec9787b32cfb7e33188be32ff7"
            ],
            "annotation": [
                "annotations/Clip2.csv",
                "e1964798cfe86e914af895f8d0291812"
            ]
        },
        "clip3": {
            "audio": [
                "audio/clip3.wav",
                "60edeb51dc4041c47c031c4bfb456b76"
```

(continues on next page)

(continued from previous page)

```
            ],
            "annotation": [
                "annotations/clip3.csv",
                "06cb006cc7b61de6be6361ff904654b3"
            ]
        },
    }
"metadata": {
        "metadata_file": [
            "metadata/metadata_file.csv",
            "7a41b280c7b74e2ddac5184708f9525b"
        ]
}
}
```

Note: In this example there is a (purposeful) mismatch between the name of the audio file `clip2.wav` and its corresponding annotation file, `Clip2.csv`, compared with the other pairs. This mismatch should be included in the index. This type of slight difference in filenames happens often in publicly available datasets, making pairing audio and annotation files more difficult. We use a fixed, version-controlled index to account for this kind of mismatch, rather than relying on string parsing on load.

2. Create a module

Once the index is created you can create the loader. For that, we suggest you use the following template and adjust it for your dataset. To quickstart a new module:

1. Copy the example below and save it to `soundata/datasets/<your_dataset_name>.py`
2. Find & Replace `Example` with the <your_dataset_name>.
3. Remove any lines beginning with # – which are there as guidelines.

Example Module

```
"""Example Dataset Loader

.. admonition:: Dataset Info
    :class: dropdown

    Please include the following information at the top level docstring for the dataset's␣
↪module `dataset.py`:

    1. Describe annotations included in the dataset
    2. Indicate the size of the datasets (e.g. number files and duration, hours)
    3. Mention the origin of the dataset (e.g. creator, institution)
    4. Indicate any relevant papers related to the dataset
    5. Include a description about how the data can be accessed and the license it uses␣
↪(if applicable)
    6. Indicate the dataset version
```

(continues on next page)

(continued from previous page)

```
"""
import os
import csv
import json


import librosa
import numpy as np
# -- import whatever you need here and remove
# -- example imports you won't use

from soundata import download_utils, jams_utils, core, annotations, io

# -- Add any relevant citations here
BIBTEX = """
@article{article-minimal,
  author = "L[eslie] B. Lamport",
  title = "The Gnats and Gnus Document Preparation System",
  journal = "G-Animal's Journal",
  year = "1986"
}
"""

# -- REMOTES is a dictionary containing all files that need to be downloaded.
# -- The keys should be descriptive (e.g. 'annotations', 'audio').
# -- When having data that can be partially downloaded, remember to set up
# -- correctly destination_dir to download the files following the correct structure.
REMOTES = {
    'remote_data': download_utils.RemoteFileMetadata(
        filename='a_zip_file.zip',
        url='http://website/hosting/the/zipfile.zip',
        checksum='00000000000000000000000000000000',  # -- the md5 checksum
        destination_dir='path/to/unzip' # -- relative path for where to unzip the data,␣
↪or None
    ),
}

# -- Include any information that should be printed when downloading
# -- remove this variable if you don't need to print anything during download
DOWNLOAD_INFO = """
Include any information you want to be printed when dataset.download() is called.
These can be instructions for how to download the dataset (e.g. request access on␣
↪zenodo),
caveats about the download, etc
"""

# -- Include the dataset's license information
LICENSE_INFO = """
The dataset's license information goes here.
"""
```

(continues on next page)

(continued from previous page)

```
class Clip(core.Clip):
    """Example Clip class
    # -- YOU CAN AUTOMATICALLY GENERATE THIS DOCSTRING BY CALLING THE SCRIPT:
    # -- `scripts/print_track_docstring.py my_dataset`
    # -- note that you'll first need to have a test clip (see "Adding tests to your␣
↪dataset" below)

    Args:
        clip_id (str): clip id of the clip

    Attributes:
        clip_id (str): clip id
        # -- Add any of the dataset specific attributes here

    """
    def __init__(self, clip_id, data_home, dataset_name, index, metadata):

        # -- this sets the following attributes:
        # -- * clip_id
        # -- * _dataset_name
        # -- * _data_home
        # -- * _clip_paths
        # -- * _clip_metadata
        super().__init__(
            clip_id,
            data_home,
            dataset_name=dataset_name,
            index=index,
            metadata=metadata,
        )

        # -- add any dataset specific attributes here
        self.audio_path = self.get_path("audio")
        self.annotation_path = self.get_path("annotation")

    # -- `annotation` will behave like an attribute, but it will only be loaded
    # -- and saved when someone accesses it. Useful when loading slightly
    # -- bigger files or for bigger datasets. By default, we make any time
    # -- series data loaded from a file a cached property
    @core.cached_property
    def annotation(self):
        """output type: description of output"""
        return load_annotation(self.annotation_path)

    # -- `audio` will behave like an attribute, but it will only be loaded
    # -- when someone accesses it and it won't be stored. By default, we make
    # -- any memory heavy information (like audio), properties
    @property
    def audio(self):
        """(np.ndarray, float): DESCRIPTION audio signal, sample rate"""
        return load_audio(self.audio_path)
```

(continues on next page)

(continued from previous page)

```
    # -- we use the to_jams function to convert all the annotations in the JAMS format.
    # -- The converter takes as input all the annotations in the proper format (e.g. tags)
    # -- and returns a jams object with the annotations.
    def to_jams(self):
        """Jams: the clip's data in jams format"""
        return jams_utils.jams_converter(
            audio_path=self.audio_path,
            annotation_data=[(self.annotation, None)],
            metadata=self._metadata,
        )
        # -- see the documentation for `jams_utils.jams_converter for all fields

@io.coerce_to_bytes_io
def load_audio(fhandle):
    """Load a Example audio file.

    Args:
        fhandle (str or file-like): path or file-like object pointing to an audio file

    Returns:
        * np.ndarray - the audio signal
        * float - The sample rate of the audio file

    """
    # -- for example, the code below. This should be dataset specific!
    # -- By default we load to mono
    # -- change this if it doesn't make sense for your dataset.
    return librosa.load(fhandle, sr=None, mono=True)


# -- Write any necessary loader functions for loading the dataset's data
@io.coerce_to_string_io
def load_annotation(fhandle):

    # -- if there are some file paths for this annotation type in this dataset's
    # -- index that are None/null, uncomment the lines below.
    # if annotation_path is None:
    #     return None

    reader = csv.reader(fhandle, delimiter=' ')
    intervals = []
    annotation = []
    for line in reader:
        intervals.append([float(line[0]), float(line[1])])
        annotation.append(line[2])

    annotation_data = annotations.EventData(
        np.array(intervals), np.array(annotation)
    )
    return annotation_data
```

(continues on next page)

(continued from previous page)

```
# -- use this decorator so the docs are complete (i.e. they are inherited from the␣
↪parent class)
@core.docstring_inherit(core.Dataset)
class Dataset(core.Dataset):
    """The Example dataset
    """

    def __init__(self, data_home=None):
        super().__init__(
            data_home,
            name='dataset_name',
            clip_class=Clip,
            bibtex=BIBTEX,
            remotes=REMOTES,
            download_info=DOWNLOAD_INFO,
            license_info=LICENSE_INFO,
        )

    # -- Copy any loader functions you wrote that should be part of the Dataset class
    # -- use this core.copy_docs decorator to copy the docs from the parent class
    # -- load_ function
    @core.copy_docs(load_audio)
    def load_audio(self, *args, **kwargs):
        return load_audio(*args, **kwargs)

    @core.copy_docs(load_annotation)
    def load_annotation(self, *args, **kwargs):
        return load_annotation(*args, **kwargs)

    # -- if your dataset has a top-level metadata file, write a loader for it here
    # -- you do not have to include this function if there is no metadata
    @core.cached_property
    def _metadata(self):

        # load metadata however makes sense for your dataset
        metadata_path = os.path.join(self.data_home, 'example_metadata.json')
        with open(metadata_path, 'r') as fhandle:
            metadata = json.load(fhandle)

        return metadata

    # -- if your dataset needs to overwrite the default download logic, do it here.
    # -- this function is usually not necessary unless you need very custom download␣
↪logic
    def download(
        self, partial_download=None, force_overwrite=False, cleanup=False
    ):
        """Download the dataset

        Args:
            partial_download (list or None):
```

(continues on next page)

(continued from previous page)

```
            A list of keys of remotes to partially download.
            If None, all data is downloaded
        force_overwrite (bool):
            If True, existing files are overwritten by the downloaded files.
        cleanup (bool):
            Whether to delete any zip/tar files after extracting.

    Raises:
        ValueError: if invalid keys are passed to partial_download
        IOError: if a downloaded file's checksum is different from expected

    """
    # see download_utils.downloader for basic usage - if you only need to call␣
↪downloader
    # once, you do not need this function at all.
    # only write a custom function if you need it!
```

You may find these examples useful as references:

- A simple, fully downloadable dataset
- A dataset which uses dataset-level metadata
- A dataset which does not use dataset-level metadata

For many more examples, see the datasets folder.

3. Add tests

To finish your contribution, please include tests that check the integrity of your loader. For this, follow these steps:

1. Make a toy version of the dataset in the tests folder `tests/resources/sound_datasets/my_dataset/`, so you can test against little data. For example:
    - Include all audio and annotation files for one clip of the dataset
    - For each audio/annotation file, reduce the audio length to 1-2 seconds and remove all but a few of the annotations.
    - If the dataset has a metadata file, reduce the length to a few lines.
2. Test all of the dataset specific code, e.g. the public attributes of the Clip class, the load functions and any other custom functions you wrote. See the tests folder for reference. If your loader has a custom download function, add tests similar to this mirdata loader.
3. Locally run `pytest -s tests/test_full_dataset.py --local --dataset my_dataset` before submitting your loader to make sure everything is working.

Note: We have written automated tests for all loader's `cite`, `download`, `validate`, `load`, `clip_ids` functions, as well as some basic edge cases of the `Clip` class, so you don't need to write tests for these!

Example Test File

```
import numpy as np

from soundata import annotations
from soundata.datasets import example  # the name of your loader here
from tests.test_utils import run_clip_tests

TEST_DATA_HOME = "tests/resources/sound_datasets/example"

def test_clip():
    default_clipid = "some_id"
    dataset = example.Dataset(TEST_DATA_HOME)
    clip = dataset.clip(default_clipid)

    expected_attributes = {
        "clip_id": "some_id",
        "audio_path": "tests/resources/sound_datasets/example/audio/some_id.wav",
        "annotation_path": "tests/resources/sound_datasets/example/annotation/some_id.pv
↪",
    }

    # List here all the properties of your loader
    expected_property_types = {"tags": annotations.Tags,
                               "some_other_annotation": "some_annotation_type"}

    run_clip_tests(clip, expected_attributes, expected_property_types)

# Test all the load functions, for instance, the load audio one
def test_load_audio():
    dataset = example.Dataset(TEST_DATA_HOME)
    clip = dataset.clip("some_id")
    audio_path = clip.audio_path
    audio, sr = example.load_audio(audio_path)
    assert sr == 44100
    assert type(audio) is np.ndarray
    assert len(audio.shape) == 1  # check audio is loaded e.g. as mono
    assert audio.shape[0] == 44100  # Check audio duration in samples is as expected


def test_to_jams():

    default_clipid = "some_id"
    data_home = "tests/resources/sound_datasets/dataset"
    dataset = example.Dataset(data_home)
    clip = dataset.clip(default_clipid)
    jam = clip.to_jams()

    annotations = jam.search(namespace="annotation")[0]["data"]
    assert [annotation.time for annotation in annotations] == [0.027, 0.232]
    assert [annotation.duration for annotation in annotations] == [
        0.20500000000000002,
        0.736,
    ]
    # ... etc
```

(continues on next page)

(continued from previous page)

```
# Test each of the load functions (e.g. Tags, etc)
def test_load_annotation():
    # load a file which exists
    annotation_path = "tests/resources/sound_datasets/dataset/annotation/some_id.pv"
    annotation_data = example.load_annotation(annotation_path)

    # check types
    assert type(annotation_data) == "some_annotation_type"
    assert type(annotation_data.times) is np.ndarray
    # ... etc

    # check values
    assert np.array_equal(annotation_data.times, np.array([0.016, 0.048]))
    # ... etc


def test_metadata():
    data_home = "tests/resources/sound_datasets/dataset"
    dataset = example.Dataset(data_home)
    metadata = dataset._metadata
    assert metadata["some_id"] == "something"
```

Running your tests locally

Before creating a PR you should run the tests. But before that, make sure to have formatted `soundata/` and `tests/` with `black`.

```
black soundata/ tests/
```

Also, make sure that they pass flake8 and mypy tests specified in lint-python.yml github action workflow.

```
flake8 soundata --count --select=E9,F63,F7,F82 --show-source --statistics
python -m mypy soundata --ignore-missing-imports --allow-subclassing-any
```

Finally, run all the tests locally like this:

```
pytest -vv --cov-report term-missing --cov-report=xml --cov=soundata tests/ --local
```

The `--local` flag skips tests that are built to run only on the remote testing environment.

To run one specific test file:

```
pytest tests/test_urbansed.py
```

Finally, there is one local test you should run, which we can't easily run in our testing environment.

```
pytest -s tests/test_full_dataset.py --local --dataset dataset
```

Where `dataset` is the name of the module of the dataset you added. The `-s` tells pytest not to skip print statments, which is useful here for seeing the download progress bar when testing the download function.

This tests that your dataset downloads, validates, and loads properly for every clip. This test takes a long time for some datasets, but it's important to ensure the integrity of the library.

The `--skip-download` flag can be added to `pytest` command to run the tests skipping the download. This will skip the downloading step. Note that this is just for convenience during debugging - the tests should eventually all pass without this flag.

Working with big datasets

In the development of large datasets, it is advisable to create an index as small as possible to optimize the implementation process of the dataset loader and pass the tests.

Working with remote indexes

For the end-user there is no difference between the remote and local indexes. However, indexes can get large when there are a lot of clips in the dataset. In these cases, storing and accessing an index remotely can be convenient. Large indexes can be added to REMOTES, and will be downloaded with the rest of the dataset. For example:

```
"index": download_utils.RemoteFileMetadata(
    filename="remote_index.json.zip",
    url="https://zenodo.org/record/.../remote_index.json.zip?download=1",
    checksum="810f1c003f53cbe58002ba96e6d4d138",
)
```

Unlike local indexes, the remote indexes will live in the `data_home` directory. When creating the `Dataset` object, specify the `custom_index_path` to where the index will be downloaded (as a relative path to `data_home`).

Reducing the testing space usage

We are trying to keep the test resources folder size as small as possible, because it can get really heavy as new loaders are added. We kindly ask the contributors to **reduce the size of the testing data** if possible (e.g. trimming the audio clips, keeping just two rows for csv files).

4. Submit your loader

Before you submit your loader make sure to:

1. Add your module to `docs/source/soundata.rst` following an alphabetical order.
2. Add your module to `docs/source/table.rst` following an alphabetical order as follows:

```
* - Dataset
  - Downloadable?
  - Annotations
  - Clips
  - Hours
  - Usecase
  - License
```

An example of this for the `UrbanSound8k` dataset:

```
* - UrbanSound8K
  - - audio:
    - annotations:
  - :ref:`tags`
  - 8732
  - 8.75
  - Urban sound classification
  - .. image:: https://licensebuttons.net/l/by-nc/4.0/80x15.png
       :target: https://creativecommons.org/licenses/by-nc/4.0
```

You can find license badges images and links here.

Pull Request template

When starting your PR please use the new_loader.md template, it will simplify the reviewing process and also help you make a complete PR. You can do that by adding `&template=new_loader.md` at the end of the url when you are creating the PR :

`...soundata/soundata/compare?expand=1` will become `...soundata/soundata/compare?expand=1&template=new_loader.md`.

Docs

Staged docs for every new PR are built and accessible at `soundata.github.io/preview/PR-<#PR_ID>` in which `<#PR_ID>` is the pull request ID. To quickly troubleshoot any issues, you can build the docs locally by nagivating to the `docs` folder, and running `make html` (note, you must have `sphinx` installed). Then open the generated `_build/source/index.html` file in your web browser to view.

Troubleshooting

If github shows a red `X` next to your latest commit, it means one of our checks is not passing. This could mean:

1. running `black` has failed – this means that your code is not formatted according to `black`'s code-style. To fix this, simply run the following from inside the top level folder of the repository:

```
black soundata/ tests/
```

2. Your code does not pass `flake8` test.

```
flake8 soundata --count --select=E9,F63,F7,F82 --show-source --statistics
```

3. Your code does not pass `mypy` test.

```
python -m mypy soundata --ignore-missing-imports --allow-subclassing-any
```

4. the test coverage is too low – this means that there are too many new lines of code introduced that are not tested.
5. the docs build has failed – this means that one of the changes you made to the documentation has caused the build to fail. Check the formatting in your changes and make sure they are consistent.
6. the tests have failed – this means at least one of the tests is failing. Run the tests locally to make sure they are passing. If they are passing locally but failing in the check, open an *issue* and we can help debug.

4.2.4 Documentation

This documentation is in rst format. It is built using Sphinx and hosted on readthedocs. The API documentation is built using autodoc, which autogenerates documentation from the code's docstrings. We use the napoleon plugin for building docs in Google docstring style. See the next section for docstring conventions.

soundata uses Google's Docstring formatting style. Here are some common examples.

Note: The small formatting details in these examples are important. Differences in new lines, indentation, and spacing make a difference in how the documentation is rendered. For example writing `Returns:` will render correctly, but `Returns` or `Returns :` will not.

Functions:

```
def add_to_list(list_of_numbers, scalar):
    """Add a scalar to every element of a list.
    You can write a continuation of the function description here on the next line.

    You can optionally write more about the function here. If you want to add an example
    of how this function can be used, you can do it like below.

    Example:
        .. code-block:: python

        foo = add_to_list([1, 2, 3], 2)

    Args:
        list_of_numbers (list): A short description that fits on one line.
        scalar (float):
            Description of the second parameter. If there is a lot to say you can
            overflow to a second line.

    Returns:
        list: Description of the return. The type here is not in parentheses

    """
    return [x + scalar for x in list_of_numbers]
```

Functions with more than one return value:

```
def multiple_returns():
    """This function has no arguments, but more than one return value. Autodoc with␣
↪napoleon doesn't handle this well,
    and we use this formatting as a workaround.

    Returns:
        * int - the first return value
        * bool - the second return value

    """
    return 42, True
```

One-line docstrings

```
def some_function():
    """
    One line docstrings must be on their own separate line, or autodoc does not build␣
↪them properly
    """
    ...
```

Objects

```
"""Description of the class
overflowing to a second line if it's long

Some more details here

Args:
    foo (str): First argument to the __init__ method
    bar (int): Second argument to the __init__ method

Attributes:
    foobar (str): First clip attribute
    barfoo (bool): Second clip attribute

Cached Properties:
    foofoo (list): Cached properties are special soundata attributes
    barbar (None): They are lazy loaded properties.
    barf (bool): Document them with this special header.

"""
```

4.2.5 Conventions

Loading from files

We use the following libraries for loading data from files:

Format	library
audio (wav, mp3, . . .)	librosa
json	json
csv	csv
jams	jams

Clip Attributes

Custom clip attributes should be global, clip-level data. For some datasets, there is a separate, dataset-level metadata file with clip-level metadata, e.g. as a csv. When a single file is needed for more than one clip, we recommend using writing a `_metadata` cached property (which returns a dictionary, either keyed by clip_id or freeform) in the Dataset class (see the dataset module example code above). When this is specified, it will populate a clip's hidden `_clip_metadata` field, which can be accessed from the clip class.

For example, if `_metadata` returns a dictionary of the form:

```
{
    'clip1': {
        'microphone-type': 'Awesome',
        'recording-date': '27.10.2021'
    },
    'clip2': {
        'microphone-type': 'Less_awesome',
        'recording-date': '27.10.2021'
    }
}
```

the `_clip metadata` for `clip_id=clip2` will be:

```
{
    'microphone-type': 'Less_awesome',
    'recording-date': '27.10.2021'
}
```

Load methods vs Clip properties

Clip properties and cached properties should be simple, and directly call a `load_*` method. Like this example from `urbansed`:

```
@property
def split(self):
    """The data splits (e.g. train)

    Returns
        * str - split

    """
    return self._clip_metadata.get("split")

@core.cached_property
def events(self) -> Optional[annotations.Events]:
    """The audio events

    Returns
        * annotations.Events - audio event object

    """
    return load_events(self.txt_path)
```

There should be no additional logic in a clip property/cached property, and instead all logic should be done in the load method. We separate these because the clip properties are only usable when data is available locally - when data is remote, the load methods are used instead.

Missing Data

Clip properties that are available for some clips and not for others should be set to `None` when whey are not available. Like this example in the `tau2019aus` loader:

```
@property
def tags(self):
    scene_label = self._clip_metadata.get("scene_label")
    if scene_label is None:
        return None
    else:
        return annotations.Tags([scene_label], "open", np.array([1.0]))
```

The index should only contain key-values for files that exist.

4.2.6 Custom Decorators

cached_property

This is used primarily for Clip classes.

This decorator causes an Object's function to behave like an attribute (aka, like the `@property` decorator), but caches the value in memory after it is first accessed. This is used for data which is relatively large and loaded from files.

docstring_inherit

This decorator is used for children of the Dataset class, and copies the Attributes from the parent class to the docstring of the child. This gives us clear and complete docs without a lot of copy-paste.

copy_docs

This decorator is used mainly for a dataset's `load_` functions, which are attached to a loader's Dataset class. The attached function is identical, and this decorator simply copies the docstring from another function.

coerce_to_bytes_io/coerce_to_string_io

These are two decorators used to simplify the loading of various *Clip* members in addition to giving users the ability to use file streams instead of paths in case the data is in a remote location e.g. GCS. The decorators modify the function to:

- Return *None* if *None* if passed in.
- Open a file if a string path is passed in either *'w'* mode for *string_io* or *wb* for *bytes_io* and pass the file handle to the decorated function.
- Pass the file handle to the decorated function if a file-like object is passed.

This cannot be used if the function to be decorated takes multiple arguments. *coerce_to_bytes_io* should not be used if trying to load an mp3 with librosa as libsndfile does not support *mp3* yet and *audioread* expects a path.

4.3 Supported Datasets and Annotations

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. To access comprehensive details and API documentation for each dataset, please consult the section *dataset loaders* within the documentation.

"Downloadable" possible values:

- Freely downloadable
- Youtube Links only
- Not available

Tasks Codes (More information at the bottom of the page):

SEL Sound Event Localization
SED Sound Event Detection
SEC Sound Event Classification
ASC Acoustic Scene Classification
AC Audio Captioning

Please note that you can click on each tag to access more information related to that specific usecase.

Dataset	Downloadable?	Annotations	Clips	Hours	Tasks	Soundscapes	License
3D-MARCo	audio: annotations:	*Tags*	26	0.3	*SEL*	*MUSIC*	CC BY-NC
DCASE23-Task2	audio: annotations:	*Tags*	174	21	*SEC*	*MACHINE*	CC BY
DCASE23-Task4B	audio: annotations:	*Events*	49	3.16	*SED*	*ENVIRONMENT BIOACOUSTIC*	CC BY-NC
DCASE23-Task6A	audio: annotations:	*Tags*	6974	43.2	*AC*		CC BY
DCASE23-Task6B	audio: annotations:	*Tags*	6974	43.2	*AC*		CC BY
DCASE-Bioacoustic	audio: annotations:	*Events*	174	21	*SED*	*BIOACOUSTIC*	CC BY
DCASE-BirdVox20k	audio: annotations:	*Tags*	20,000	55.5	*SEC*	*BIOACOUSTIC*	CC BY
EigenScape (HOA 25 ch)	audio: annotations:	*Tags*	64	10.7	*ASC*		CC BY
EigenScape Raw (32 ch)	audio: annotations:	*Tags*	64	10.7	*ASC*		CC BY
ESC-50	audio: annotations:	*Tags*	2000	2.8	*SEC*	*ENVIRONMENT*	CC BY-NC
Freefield1010	audio: annotations:	*Tags*	7690	21.3	*SEC*	*BIOACOUSTIC*	CC BY
FSD50K	audio: annotations:	*Tags*	51197	108.3	*SEC*	*ENVIRONMENT MUSIC*	CC BY

4.3.1 Annotation Types

The table above provides annotation types as a guide for choosing appropriate datasets. 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. To see how these annotation types are implemented in Soundata see *Annotations*.

Tags

One or more `string labels` with corresponding `confidence values`. Tags do not have start or end times, and span the full duration of the clip. Tags are used to represent annotations for:

- Acoustic Scene Classification (ASC)
- Sound Event Classification (SEC)
- Sound Event Detection (SED) - weak labels

When every Tags annotation in a dataset contains exactly one label, it is typically a `multi-class` problem. When Tags annotations contain varying numbers of labels (including 0), it is typically a `multi-label` problem.

Events

Sound events with a `start time`, `end time`, `label`, and `confidence`. Events are used to represent annotations for:

- Sound Event Detection (SED) - strong labels

Spatial Events

Spatial events represent annotations used for various applications, including spatial event detection and tracking. Similar to Sound Events, Spatial Events include essential attributes such as `start time`, `end time`, `label`, and `confidence` to characterize and annotate spatial phenomena. This can be extended to include additional attributes specific to the application, such as geographical coordinates (latitude, longitude), altitude, direction (azimuth and elevation), and distance from reference points. Spatial events are used to represent annotations for:

- Sound Event Detection (SED) + Sound Event Localization (SEL)

4.3.2 Usecases

Tasks

Sound Event Localization (SEL)
: SEL involves determining the spatial location from where a sound originates within an environment. It goes beyond detection and classification to include the position in space relative to the listener or recording device.

Sound Event Detection (SED)
: SED is concerned with identifying the presence and duration of sound events within an audio stream. It uses both weak labels (Tags) for presence and strong labels (Events) for temporal localization of sound events.

Sound Event Classification (SEC)

SEC categorizes sounds into predefined classes and involves analyzing audio to assign a category based on the type of sound event it contains, using Tags for the entire clip's duration.

Acoustic Scene Classification (ASC)

ASC classifies an entire audio stream into a scene category, characterizing the recording's environment. Tags are used to indicate the single acoustic scene represented in the clip.

Audio Captioning (AC)

AC involves generating a textual description of the sound events and context within an audio clip. It is similar to image captioning but for audio content.

Soundscapes

URBAN

Urban environments are characterized by a blend of sounds from traffic, human activity, construction, and sometimes nature. Recordings in urban areas are often used to study noise pollution, city planning, or to create soundscapes for multimedia productions.

ENVIRONMENT

The spectrum of environmental sounds includes all the background noises found in various habitats. These auditory elements can be as diverse as the whisper of foliage in woodlands, the gentle flow of water in brooks, or the fierce gusts of wind sweeping through arid landscapes.

MACHINE

Machine sounds refer to the audio signatures of mechanical devices, such as engines, factory machinery, household appliances, and office equipment. These sounds are crucial for monitoring equipment performance, diagnosing faults, and designing sound-aware applications.

BIOACOUSTIC

Bioacoustic sounds are produced by biological organisms, like the vocalizations of animals and birds. Studying these sounds can provide insights into animal behavior, biodiversity, and ecosystem health.

MUSIC

Music sounds encompass the vast array of musical compositions, instruments, and the human voice as used in singing. These sounds are central to the entertainment industry, cultural studies, and music therapy.

4.4 Initialize a dataset

soundata.**initialize**(*dataset_name*, *data_home=None*)

Load a soundata dataset by name

Example

```
urbansound8k = soundata.initialize('urbansound8k')  # get the urbansound8k dataset
urbansound8k.download()  # download orchset
urbansound8k.validate()  # validate orchset
clip = urbansound8k.choice_clip()  # load a random clip
print(clip)  # see what data a clip contains
urbansound8k.clip_ids()  # load all clip ids
```

Parameters

- **dataset_name** (*str*) – the dataset's name see soundata.DATASETS for a complete list of possibilities
- **data_home** (*str or None*) – path where the data lives. If None uses the default location.

Returns

Dataset – a soundata.core.Dataset object

soundata.**list_datasets**()

Get a list of all soundata dataset names

Returns

list – list of dataset names as strings

4.5 Dataset Loaders

4.5.1 3D-MARCo

3D-MARCo Dataset Loader

Dataset Info

3D-MARCo: database of 3D sound recordings of musical performances and room impulse responses

Created By:

Hyunkook Lee, Dale Johnson, Bogdan Bacila.

Centre for Audio and Psychoacoustic Engineering, University of Huddersfield.

Version 1.0.1

Description:

3D-MARCo is an open-access database of 3D sound recordings of musical performances and room impulse responses. The recordings were made in the St. Paul's concert hall in Huddersfield, UK A total of 71 microphone capsules were used simultaneously. The main microphone arrays included in the database comprise PCMA-3D, OCT-3D, 2L-Cube, Decca Cubioid, First-order Ambisonics (FOA), Higher-order Ambisonics (HOA) and

Hamasaki Square with height. In addition, ORTF, side/height, Voice of God and floor channels as well as a dummy head and spot microphones are included. The sound sources recorded are string quartet, piano trio, piano solo, organ, a cappella group, various single sources and room impulse responses of a virtual ensemble with 13 source positions captured by all of the microphones. 3D-MARCo would be useful for spatial audio research, recording education, critical ear training, etc.

Audio Files Included:

- **For each musical performance sound source (Acappella, Organ, Piano Solo 1, Piano solo 2, Quartet, Trio), there are 65 wav files that correspond to:**
  - 64 individual capsules (24-bit / 96kHz resolution)
  - one 32-channel EigenMike file in A-format (24-bit / 48kHz resolution).
- The piano recordings contain two more channels (left and right) that correspond to spot microphones placed just outside the piano pointing toward the hammers.
- The quartet recordings contain four more channels corresponding to spot microphones placed above the instruments (violin 1, violin 2, cello, viola) pointing toward the F hole.
- The trio recordins contain four more channels corresponding to spot microphones, two placed above the string instruments (violin, cello) pointing toward the F hole, and two placed just outside the piano pointing toward the hammers.
- The single sources were recorded at 7 different azimuth angles. For each angle there are also 65 wav files.
- The impulse responses were recorded at 13 different azimuth angles. For each angle there are 66 wav files. The extra one is the EigenMike 4th-order B-format ambisonics (ACN SN3D; 24-bit / 48kHz resolution).

Annotations Included:

- No event labels associated with this dataset
- No predefined training, validation, or testing splits.
- Angular orientation for "impulse responses" and "single sources" (follows the ITU-R convention where positive angles in the left-hand side and negative angles in the right-hand side, e.g. +30° for Front Left and -30° for Front Right).

Please Acknowledge 3D-MARCo in Academic Research:

- If you use this dataset please cite its original publication:
- Lee H, Johnson D. An open-access database of 3D microphone array recordings. InAudio Engineering Society Convention 147 2019 Oct 8. Audio Engineering Society.

License:

- CC-BY NC 3.0 license (free to share and adapt the material, but not permitted to use it for commercial purposes)

`class` soundata.datasets.marco.**Clip**(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

3D-MARCo Clip class

Parameters
clip_id (*str*) – id of the clip

Variables

- **source_label** (*str*) – label of the source being recorded
- **source_angle** (*str*) – angle of the source being recorded
- **audio_path** (*str*) – path to the audio file

- **clip_id** (*str*) – clip id
- **microphone_info** (*list*) – list of strings with all relevant microphone metadata

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

to_jams()

Get the clip's data in jams format

Returns
jams.JAMS – the clip's data in jams format

class soundata.datasets.marco.**Dataset**(*data_home=None*)

The 3D-MARCo dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

cite()

Print the reference

`clip_ids`

Return clip ids

Returns
list – A list of clip ids

`clipgroup_ids`

Return clip ids

Returns
list – A list of clip ids

`property default_path`

Get the default path for the dataset

Returns
str – Local path to the dataset

`download`(*partial_download=None*, *force_overwrite=False*, *cleanup=False*)

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- `ValueError` – if invalid keys are passed to partial_download
- `IOError` – if a downloaded file's checksum is different from expected

`explore_dataset`(*clip_id=None*)

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters
clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

`license`()

Print the license

`load_audio`(**args*, ***kwargs*)

Load a 3D-MARCo audio file. :Parameters: * **fhandle** (*str or file-like*) – file-like object or path to audio file

- **sr** (*int or None*) – sample rate for loaded audio, 48000 by default, which re-samples all files except the EigenMike ones, resulting in constant sampling rate between all clips in the dataset.

Returns

- np.ndarray - the audio signal
- float - The sample rate of the audio file

`load_clipgroups()`

Load all clipgroups in the dataset

Returns
dict – {*clipgroup_id*: clipgroup data}

Raises
NotImplementedError – If the dataset does not support Clipgroups

`load_clips()`

Load all clips in the dataset

Returns
dict – {*clip_id*: clip data}

Raises
NotImplementedError – If the dataset does not support Clips

`validate(verbose=True)`

Validate if the stored dataset is a valid version

Parameters
verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

`soundata.datasets.marco.load_audio(fhandle: BinaryIO, sr=48000)` → Tuple[numpy.ndarray, float]

Load a 3D-MARCo audio file. :Parameters: * **fhandle** (*str or file-like*) – file-like object or path to audio file

- **sr** (*int or None*) – sample rate for loaded audio, 48000 by default, which re-samples all files except the EigenMike ones, resulting in constant sampling rate between all clips in the dataset.

Returns

- np.ndarray - the audio signal
- float - The sample rate of the audio file

4.5.2 DCASE23-Task2

DCASE23_Task2 Dataset Loader

Dataset Info

Created By

Noboru Harada, Daisuke Niizumi, Yasunori Ohishi, Daiki Takeuchi, and Masahiro Yasuda (Hitachi, Ltd. and NTT Corporation).

Version
1.0

Description
The DCASE 2023 Task 2 "First-Shot Unsupervised Anomalous Sound Detection for Machine Condition Monitoring" dataset provides the operating sounds of seven real/toy machines: ToyCar, ToyTrain, Fan, Gearbox,

Bearing, Slide rail, and Valve. Each recording is a single-channel, 10-second audio that includes both a machine's operating sound and environmental noise. The dataset contains training clips containing normal sounds in the source and target domain and test clips of both normal and anomalous sounds.

Audio Files Included

10,000 ten-second audio recordings for each machine type in WAV format. The *raw* directory contains recordings as WAV files, with the source/target domain and attributes provided in the file name.

Meta-data Files Included

Attribute csv files accompany the audio files for easy access to attributes that cause domain shifts. Each file lists the file names, domain shift parameters, and the value or type of these parameters.

Please Acknowledge DCASE 2023 Task 2 in Academic Research

When the DCASE 2023 Task 2 dataset is used for academic research, we would highly appreciate it if scientific publications of works partly based on this dataset cite the following publications:

Conditions of Use

The DCASE 2023 Task 2 dataset was created jointly by Hitachi, Ltd. and NTT Corporation. It is available under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0) license.

Feedback

For any issues or feedback regarding the dataset, please reach out to: | * Kota Dohi: kota.dohi.gr@hitachi.com | * Keisuke Imoto: keisuke.imoto@ieee.org | * Noboru Harada: noboru@ieee.org | * Daisuke Niizumi: daisuke.niizumi.dt@hco.ntt.co.jp | * Yohei Kawaguchi: yohei.kawaguchi.xk@hitachi.com.

`class soundata.datasets.dcase23_task2.Clip(`*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*`)`

DCASE23_Task2 Clip class :Parameters: **clip_id** (*str*) – ID of the clip

Variables

- **audio** (*np.ndarray, float*) – Array representation of the audio clip
- **audio_path** (*str*) – Path to the audio file
- **file_name** (*str*) – Name of the clip file, useful for cross-referencing
- **d1p** (*str*) – First domain shift parameter specifying the attribute causing the domain shift
- **d1v** (*str*) – First domain shift value or type associated with the domain shift parameter

`property audio: Optional[Tuple[numpy.ndarray, float]]`

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

`property d1p`

The clip's first domain shift parameter (d1p).

Returns

- str - first domain shift parameter of the clip

`property d1v`

The clip's first domain shift value (d1v).

Returns

- str - first domain shift value of the clip

property file_name

The clip's file name.

Used for cross-referencing with attribute CSV files for additional metadata.

Returns

- str - name of the clip file

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

to_jams()

Get the clip's data in jams format

Returns
jams.JAMS – the clip's data in jams format

class soundata.datasets.dcase23_task2.Dataset(*data_home=None*)

The DCASE23_Task2 dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

cite()

Print the reference

clip_ids

Return clip ids

Returns
list – A list of clip ids

`clipgroup_ids`

Return clip ids

Returns
list – A list of clip ids

`property default_path`

Get the default path for the dataset

Returns
str – Local path to the dataset

`download`(*partial_download=None*, *force_overwrite=False*, *cleanup=False*)

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- `ValueError` – if invalid keys are passed to partial_download
- `IOError` – if a downloaded file's checksum is different from expected

`explore_dataset`(*clip_id=None*)

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters
clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

`license`()

Print the license

`load_audio`(**args*, ***kwargs*)

Load a DCASE23_Task2 audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, 44100 Hz by default. If different from file's sample rate it will be resampled on load. Use None to load the file using its original sample rate (sample rate varies from file to file).

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

`load_clipgroups`()

Load all clipgroups in the dataset

Returns
dict – {*clipgroup_id*: clipgroup data}

Raises
NotImplementedError – If the dataset does not support Clipgroups

`load_clips()`

Load all clips in the dataset

Returns
dict – {*clip_id*: clip data}

Raises
NotImplementedError – If the dataset does not support Clips

`validate(verbose=True)`

Validate if the stored dataset is a valid version

Parameters
verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

`soundata.datasets.dcase23_task2.load_audio(fhandle: BinaryIO, sr=44100)` → Tuple[numpy.ndarray, float]

Load a DCASE23_Task2 audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, 44100 Hz by default. If different from file's sample rate it will be resampled on load. Use None to load the file using its original sample rate (sample rate varies from file to file).

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

4.5.3 DCASE23-Task4B

DCASE23 Task 4B Dataset Loader

Dataset Info

Created By:

Annamaria Mesaros, Tuomas Heittola, and Tuomas Virtanen.
Tampere University of Technology.

Version 1.0

Description:
MAESTRO real development contains 49 real-life audio files from 5 different acoustic scenes, each of them from 3 to 5 minutes long. The other 26 files are kept for evaluation purposes on the DCASE task 4 B. The distribution of files per scene is the following: cafe restaurant 10 files, city center 10 files, residential_area 11 files, metro station 9 files and grocery store 9 files. The total duration of the development dataset is 97 minutes and 4 seconds.

The audio files contain sounds from the following classes:

- announcement
- birds singing
- breakes squeaking
- car
- cash register
- children voices
- coffee machine
- cutlery/dishes
- door opens/closes
- footsteps
- furniture dragging

The real life-recordings used in this study include a subset of the TUT Sound Events 2016 and a subset of TUT Sound Events 2017.

Please Acknowledge TUT Acoustic Scenes Strong Label Dataset in Academic Research:

If you use this dataset, please cite the following paper:
A. Mesaros, T. Heittola, and T. Virtanen, "TUT database for acoustic scene classification and sound event detection," in 2016 24th European Signal Processing Conference (EUSIPCO), 2016, pp. 1128-1132.

License:

Feedback:
For questions or feedback, please contact irene.martinmorato@tuni.fi.

`class` soundata.datasets.dcase23_task4b.**Clip**(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

DCASE23_Task4B Clip class

Parameters
clip_id (*str*) – id of the clip

Variables

- **audio** (*np.ndarray, float*) – path to the audio file
- **audio_path** (*str*) – path to the audio file
- **annotations_path** (*str*) – path to the annotations file
- **clip_id** (*str*) – clip id
- **events** (soundata.annotations.Events) – sound events with start time, end time, label and confidence
- **split** (*str*) – subset the clip belongs to: development or evaluation

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

events

The clip's events.

Returns

- annotations.Events - sound events with start time, end time, label and confidence

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

property split

The clip's split.

Returns
** str - subset the clip belongs to* – development or evaluation

to_jams()

Get the clip's data in jams format

Returns
jams.JAMS – the clip's data in jams format

class soundata.datasets.dcase23_task4b.**Dataset**(*data_home=None*)

The DCASE23_Task4B dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset

- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

`choice_clip()`

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

`choice_clipgroup()`

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

`cite()`

Print the reference

`clip_ids`

Return clip ids

Returns
list – A list of clip ids

`clipgroup_ids`

Return clip ids

Returns
list – A list of clip ids

`property default_path`

Get the default path for the dataset

Returns
str – Local path to the dataset

`download(partial_download=None, force_overwrite=False, cleanup=False)`

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- **ValueError** – if invalid keys are passed to partial_download
- **IOError** – if a downloaded file's checksum is different from expected

`explore_dataset(clip_id=None)`

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters
clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

`license()`

Print the license

`load_audio(*args, **kwargs)`

Load a DCASE23_Task4B audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate of 44100 without resampling.

Returns

- np.ndarray - the stereo audio signal
- float - The sample rate of the audio file

`load_clipgroups()`

Load all clipgroups in the dataset

Returns
dict – {*clipgroup_id*: clipgroup data}

Raises
`NotImplementedError` – If the dataset does not support Clipgroups

`load_clips()`

Load all clips in the dataset

Returns
dict – {*clip_id*: clip data}

Raises
`NotImplementedError` – If the dataset does not support Clips

`load_events(*args, **kwargs)`

Load a DCASE23_Task4B annotation file :Parameters: * **fhandle** (*str or file-like*) – File-like object or path to the sound

- **events annotation file**

Returns
Events – sound events annotation data

`validate(verbose=True)`

Validate if the stored dataset is a valid version

Parameters
verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

`soundata.datasets.dcase23_task4b.load_audio(fhandle: BinaryIO, sr=None)` → Tuple[numpy.ndarray, float]

Load a DCASE23_Task4B audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate of 44100 without resampling.

Returns

- np.ndarray - the stereo audio signal
- float - The sample rate of the audio file

`soundata.datasets.dcase23_task4b.load_events`(*fhandle: TextIO*) → *Events*

Load a DCASE23_Task4B annotation file :Parameters: * **fhandle** (*str or file-like*) – File-like object or path to the sound

- **events annotation file**

Returns

Events – sound events annotation data

4.5.4 DCASE23-Task6a

DCASE 2023 Task-6A Dataset Loader

Dataset Info

DCASE 2023 Task-6A

Clotho (c) by K. Drossos, S. Lipping, and T. Virtanen.

Clotho is licensed under the terms set by Tampere University and Creative Commons licenses for the audio files as per their origin from the Freesound platform.

You should have received a copy of the license along with this work.

https://github.com/audio-captioning/clotho-dataset

Paper: "Clotho: an Audio Captioning Dataset," ICASSP 2020

Created By:

K. Drossos, S. Lipping, and T. Virtanen.

Tampere University, Finland

Version 2.1.0

Fixes for corrupted files and illegal characters.

More details on version changes are available in the dataset repository.

Description

Clotho is an audio captioning dataset, consisting of 6974 audio samples, each accompanied by five captions, totaling 34,870 captions.

- Audio samples are 15 to 30 seconds in duration.
- Captions are 8 to 20 words long.
- Dataset splits: development, validation, and evaluation.
- Detailed description and usage guidelines in the ICASSP 2020 paper and dataset repository.

Audio Files Included

- Development split: 3840 audio files (including 947 new files in version 2)
- Validation split: 1046 new audio files
- Evaluation split: No changes from version 1
- File format: Single channel (mono), various bitrates and sample rates, WAV format.

Caption Files Included

- Clotho captions in CSV format for each dataset split.
- Captions follow consistent word usage, no named entities or speech transcription.
- Unique vocabulary across splits to prevent data leakage.

Metadata Files Included

- Accompanying metadata for each audio file, including file name, keywords, original URL, excerpt samples, uploader, and license link.

Conditions of Use

Dataset created by K. Drossos, S. Lipping, and T. Virtanen.
Audio files under various Creative Commons licenses as per Freesound platform terms.
Captions under Tampere University license, primarily non-commercial with attribution.
Full details in the LICENSE file included with the dataset.

Acknowledgment in Academic Research

When using Clotho for academic research, please cite: K. Drossos, S. Lipping, and T. Virtanen, "Clotho: an Audio Captioning Dataset," ICASSP 2020.

Feedback and Contributions

Feedback and contributions are welcome.
Please contact the creators through the GitHub repository.

`class` soundata.datasets.dcase23_task6a.**Clip**(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

DCASE'23 Task 6A Clip class

Parameters
clip_id (*str*) – id of the clip

Variables

- **audio** (*np.ndarray, float*) – Audio signal and sample rate.
- **file_name** (*str*) – Name of the file.
- **keywords** (*str*) – Associated keywords.
- **sound_id** (*str*) – Unique identifier for the sound.
- **sound_link** (*str*) – Link to the sound.
- **start_end_samples** (*tuple*) – Start and end samples in the audio file.
- **manufacturer** (*str*) – Manufacturer of the recording equipment.
- **license** (*str*) – License of the clip.

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

property file_name

The name of the audio file.

Returns

- str - Name of the file.

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

property keywords

Keywords associated with the clip.

Returns

- str - Keywords for the clip.

property license

License of the clip.

Returns

- str - License information.

property manufacturer

Manufacturer of the recording equipment.

Returns

- str - Manufacturer name.

property sound_id

Unique identifier for the sound.

Returns

- str - Sound ID.

property sound_link

Link to the sound.

Returns

- str - URL of the sound.

property start_end_samples

Start and end samples in the audio file.

Returns

- tuple - Start and end samples.

to_jams()

Get the clip's data in jams format

Returns
jams.JAMS – the clip's data in jams format

class soundata.datasets.dcase23_task6a.**Dataset**(*data_home=None*)

The DCASE'23 Task 6A dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

cite()

Print the reference

clip_ids

Return clip ids

Returns
list – A list of clip ids

clipgroup_ids

Return clip ids

Returns
list – A list of clip ids

property default_path

Get the default path for the dataset

Returns
str – Local path to the dataset

download(*partial_download=None*, *force_overwrite=False*, *cleanup=False*)

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- `ValueError` – if invalid keys are passed to partial_download
- `IOError` – if a downloaded file's checksum is different from expected

`explore_dataset`(*clip_id=None*)

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters

clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

`license`()

Print the license

`load_audio`(**args*, ***kwargs*)

Load a DCASE'23 Task 6A audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate of 44100 without resampling.

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

`load_clipgroups`()

Load all clipgroups in the dataset

Returns

dict – {*clipgroup_id*: clipgroup data}

Raises

`NotImplementedError` – If the dataset does not support Clipgroups

`load_clips`()

Load all clips in the dataset

Returns

dict – {*clip_id*: clip data}

Raises

`NotImplementedError` – If the dataset does not support Clips

`validate`(*verbose=True*)

Validate if the stored dataset is a valid version

Parameters

verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

soundata.datasets.dcase23_task6a.**load_audio**(*fhandle: BinaryIO*, *sr=None*) → Tuple[numpy.ndarray, float]

Load a DCASE'23 Task 6A audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate of 44100 without resampling.

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

4.5.5 DCASE23-Task6b

DCASE 2023 Task-6B Dataset Loader

Dataset Info

DCASE 2023 Task-6B

Clotho (c) by K. Drossos, S. Lipping, and T. Virtanen.
Clotho is licensed under the terms set by Tampere University and Creative Commons licenses for the audio files as per their origin from the Freesound platform.
You should have received a copy of the license along with this work.
https://github.com/audio-captioning/clotho-dataset
Paper: "Clotho: an Audio Captioning Dataset," ICASSP 2020

Created By:

K. Drossos, S. Lipping, and T. Virtanen.
Tampere University, Finland

Version 2.1.0

Fixes for corrupted files and illegal characters.
More details on version changes are available in the dataset repository.

Description

Clotho is an audio captioning dataset, consisting of 6974 audio samples, each accompanied by five captions, totaling 34,870 captions.

- Audio samples are 15 to 30 seconds in duration.
- Captions are 8 to 20 words long.
- Dataset splits: development, validation, and evaluation.
- Detailed description and usage guidelines in the ICASSP 2020 paper and dataset repository.

Audio Files Included

- Development split: 3840 audio files (including 947 new files in version 2)
- Validation split: 1046 new audio files
- Evaluation split: No changes from version 1
- File format: Single channel (mono), various bitrates and sample rates, WAV format.

Caption Files Included

- Clotho captions in CSV format for each dataset split.
- Captions follow consistent word usage, no named entities or speech transcription.
- Unique vocabulary across splits to prevent data leakage.

Metadata Files Included

- Accompanying metadata for each audio file, including file name, keywords, original URL, excerpt samples, uploader, and license link.

Conditions of Use

Dataset created by K. Drossos, S. Lipping, and T. Virtanen.
Audio files under various Creative Commons licenses as per Freesound platform terms.
Captions under Tampere University license, primarily non-commercial with attribution.
Full details in the LICENSE file included with the dataset.

Acknowledgment in Academic Research

When using Clotho for academic research, please cite: K. Drossos, S. Lipping, and T. Virtanen, "Clotho: an Audio Captioning Dataset," ICASSP 2020.

Feedback and Contributions

Feedback and contributions are welcome.
Please contact the creators through the GitHub repository.

`class` soundata.datasets.dcase23_task6b.**Clip**(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

DCASE'23 Task 6B Clip class

Parameters
clip_id (*str*) – id of the clip

Variables

- **audio** (*np.ndarray, float*) – Audio signal and sample rate.
- **file_name** (*str*) – Name of the file.
- **keywords** (*str*) – Associated keywords.
- **sound_id** (*str*) – Unique identifier for the sound.
- **sound_link** (*str*) – Link to the sound.
- **start_end_samples** (*tuple*) – Start and end samples in the audio file.
- **manufacturer** (*str*) – Manufacturer of the recording equipment.
- **license** (*str*) – License of the clip.

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

property file_name

The name of the audio file.

Returns

- str - Name of the file.

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

property keywords

Keywords associated with the clip.

Returns

- str - Keywords for the clip.

property license

License of the clip.

Returns

- str - License information.

property manufacturer

Manufacturer of the recording equipment.

Returns

- str - Manufacturer name.

property sound_id

Unique identifier for the sound.

Returns

- str - Sound ID.

property sound_link

Link to the sound.

Returns

- str - URL of the sound.

property start_end_samples

Start and end samples in the audio file.

Returns

- tuple - Start and end samples.

to_jams()

Get the clip's data in jams format

Returns
jams.JAMS – the clip's data in jams format

class soundata.datasets.dcase23_task6b.**Dataset**(*data_home=None*)

The DCASE'23 Task 6B dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

cite()

Print the reference

clip_ids

Return clip ids

Returns
list – A list of clip ids

clipgroup_ids

Return clip ids

Returns
list – A list of clip ids

property default_path

Get the default path for the dataset

Returns
str – Local path to the dataset

download(*partial_download=None*, *force_overwrite=False*, *cleanup=False*)

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- **ValueError** – if invalid keys are passed to partial_download
- **IOError** – if a downloaded file's checksum is different from expected

explore_dataset(*clip_id=None*)

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters

clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

license()

Print the license

load_audio(**args*, ***kwargs*)

Load a DCASE'23 Task 6B audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate of 44100 without resampling.

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

load_clipgroups()

Load all clipgroups in the dataset

Returns

dict – {*clipgroup_id*: clipgroup data}

Raises

NotImplementedError – If the dataset does not support Clipgroups

load_clips()

Load all clips in the dataset

Returns

dict – {*clip_id*: clip data}

Raises

NotImplementedError – If the dataset does not support Clips

validate(*verbose=True*)

Validate if the stored dataset is a valid version

Parameters

verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

soundata.datasets.dcase23_task6b.**load_audio**(*fhandle: BinaryIO*, *sr=None*) → Tuple[numpy.ndarray, float]

Load a DCASE'23 Task 6B audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate of 44100 without resampling.

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

4.5.6 DCASE-bioacoustic

DCASE-BIOACOUSTIC Dataset Loader

Dataset Info

DCASE-BIOACOUSTIC

Development set:

The development set for task 5 of DCASE 2022 "Few-shot Bioacoustic Event Detection" consists of 192 audio files acquired from different bioacoustic sources. The dataset is split into training and validation sets.

Multi-class annotations are provided for the training set with positive (POS), negative (NEG) and unkwown (UNK) values for each class. UNK indicates uncertainty about a class.

Single-class (class of interest) annotations are provided for the validation set, with events marked as positive (POS) or unkwown (UNK) provided for the class of interest.

this version (3):

- fixes issues with annotations from HB set

Folder Structure:

Development_Set.zip

|_Development_Set/

|__Training_Set/

|___JD/

|____*.wav

|____*.csv

|___HT/

|____*.wav

|____*.csv

|___BV/

|____*.wav

|____*.csv

|___MT/

|____*.wav

|____*.csv

|___WMW/

|____*.wav

|____*.csv

|__Validation_Set/

|___HB/

|____*.wav

|____*.csv

|___PB/

|____*.wav

|____*.csv

|___ME/

|____*.wav

|____*.csv

Development_Set_Annotations.zip has the same structure but contains only the *.csv files

Annotation structure

Each line of the annotation csv represents an event in the audio file. The column descriptions are as follows:

Audiofilename, Starttime, Endtime, CLASS_1, CLASS_2, . . . CLASS_N

Audiofilename, Starttime, Endtime, Q

Classes

DCASE2022_task5_training_set_classes.csv and DCASE2022_task5_validation_set_classes.csv provide a table with class code correspondence to class name for all classes in the Development set.

dataset, class_code, class_name

dataset, recording, class_code, class_name

Evaluation set

The evaluation set for task 5 of DCASE 2022 "Few-shot Bioacoustic Event Detection" consists of 46 audio files acquired from different bioacoustic sources.

The first 5 annotations are provided for each file, with events marked as positive (POS) for the class of interest.

This dataset is to be used for evaluation purposes during the task and the rest of the annotations will be released after the end of the DCASE 2022 challenge (July 1st).

Folder Structure

Evaluation_Set.zip

|___DC/

|____*.wav

|____*.csv

|___CT/

|____*.wav

|____*.csv

|___CHE/

|____*.wav

|____*.csv

|___MGE/

|____*.wav

|____*.csv

|___MS/

|____*.wav

|____*.csv

|___QU/

|____*.wav

|____*.csv

Evaluation_Set_5shots.zip has the same structure but contains only the *.wav files.

Evaluation_Set_5shots_annotations_only.zip has the same structure but contains only the *.csv files

The subfolders denote different recording sources and there may or may not be overlap between classes of interest from different wav files.

Annotation structure

Each line of the annotation csv represents an event in the audio file. The column descriptions are as follows: [Audiofilename, Starttime, Endtime, Q]

Open Access:

This dataset is available under a Creative Commons Attribution 4.0 International (CC BY 4.0) license.

Contact info:

Please send any feedback or questions to:

Ines Nolasco - i.dealmeidanolasco@qmul.ac.uk

---

class soundata.datasets.dcase_bioacoustic.**Clip**(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

DCASE bioacoustic Clip class

Parameters

clip_id (*str*) – id of the clip

Variables

- **audio** (*np.ndarray, float*) – path to the audio file
- **audio_path** (*str*) – path to the audio file
- **csv_path** (*str*) – path to the csv file

- **clip_id** (*str*) – clip id
- **split** (*str*) – subset the clip belongs to (for experiments): train, validate, or test

Other Parameters

- **events_classes** (*list*) – list of classes annotated for the file
- **events** (*soundata.annotations.Events*) – sound events with start time, end time, labels (list for all classes) and confidence
- **POSevents** (*soundata.annotations.Events*) – sound events for the positive class with start time, end time, label and confidence

POSevents

The audio events for POS (positive) class

Returns

- annotations.Events - audio event object

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

events

The audio events

Returns

- annotations.Events - audio event object

events_classes

The audio events

Returns

- list - list of the annotated events

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

property split

The data splits (e.g. train)

Returns

- str - split

property subdataset

The (sub)dataset

Returns

- str - subdataset

to_jams()

Get the clip's data in jams format

Returns
jams.JAMS – the clip's data in jams format

class soundata.datasets.dcase_bioacoustic.Dataset(*data_home=None*)

The DCASE bioacoustic dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

cite()

Print the reference

clip_ids

Return clip ids

Returns
list – A list of clip ids

clipgroup_ids

Return clip ids

Returns
list – A list of clip ids

property default_path

Get the default path for the dataset

Returns
str – Local path to the dataset

download(*partial_download=None*, *force_overwrite=False*, *cleanup=False*)

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- **ValueError** – if invalid keys are passed to partial_download
- **IOError** – if a downloaded file's checksum is different from expected

explore_dataset(*clip_id=None*)

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters

clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

license()

Print the license

load_audio(**args*, ***kwargs*)

Load a DCASE bioacoustic audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate without resampling.

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

load_clipgroups()

Load all clipgroups in the dataset

Returns

dict – {*clipgroup_id*: clipgroup data}

Raises

NotImplementedError – If the dataset does not support Clipgroups

load_clips()

Load all clips in the dataset

Returns

dict – {*clip_id*: clip data}

Raises

NotImplementedError – If the dataset does not support Clips

validate(*verbose=True*)

Validate if the stored dataset is a valid version

Parameters

verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

soundata.datasets.dcase_bioacoustic.**load_POSevents**(*fhandle: TextIO*) → *Events*

Load an DCASE bioacoustic sound events annotation file, just for POS labels

Parameters
fhandle (*str or file-like*) – File-like object or path to the sound events annotation file

Raises
IOError – if csv_path doesn't exist

Returns
Events – sound events annotation data

soundata.datasets.dcase_bioacoustic.**load_audio**(*fhandle: BinaryIO*, *sr=None*) → Tuple[numpy.ndarray, float]

Load a DCASE bioacoustic audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate without resampling.

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

soundata.datasets.dcase_bioacoustic.**load_events**(*fhandle: TextIO*) → *Events*

Load an DCASE bioacoustic sound events annotation file

Parameters
fhandle (*str or file-like*) – File-like object or path to the sound events annotation file

Raises
IOError – if csv_path doesn't exist

Returns
Events – sound events annotation data

soundata.datasets.dcase_bioacoustic.**load_events_classes**(*fhandle: TextIO*) → list

Load an DCASE bioacoustic sound events annotation file

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to the sound events annotation file
- **positive** (*bool*) – False get all labels, True get just POS labels

Raises
IOError – if csv_path doesn't exist

Returns
class_ids – list of events classes

4.5.7 DCASE-birdVox20k

BirdVox20k Dataset Loader

Dataset Info

Created By

Vincent Lostanlen*^#, Justin Salamon^#, Andrew Farnsworth*, Steve Kelling*, and Juan Pablo Bello^#
* Cornell Lab of Ornithology (CLO)
^ Center for Urban Science and Progress, New York University
# Music and Audio Research Lab, New York University

Version 1.0

Description

The BirdVox-DCASE-20k dataset contains 20,000 ten-second audio recordings. These recordings come from ROBIN autonomous recording units, placed near Ithaca, NY, USA during the fall 2015. They were captured on the night of September 23rd, 2015, by six different sensors, originally numbered 1, 2, 3, 5, 7, and 10. Out of these 20,000 recording, 10,017 (50.09%) contain at least one bird vocalization (either song, call, or chatter). The dataset is a derivative work of the BirdVox-full-night dataset [1], containing almost as much data but formatted into ten-second excerpts rather than ten-hour full night recordings. In addition, the BirdVox-DCASE-20k dataset is provided as a development set in the context of the "Bird Audio Detection" challenge, organized by DCASE (Detection and Classification of Acoustic Scenes and Events) and the IEEE Signal Processing Society. The dataset can be used, among other things, for the development and evaluation of bioacoustic classification models.

Audio Files Included

20,000 ten-second audio recordings (see description above) in WAV format. The wav folder contains the recordings as WAV files, sampled at 44,1 kHz, with a single channel (mono). The original sample rate was 24 kHz.

Meta-data Files Included

A table containing a binary label "hasbird" associated to every recording in BirdVox-DCASE-20k is available on the website of the DCASE "Bird Audio Detection" challenge: http://machine-listening.eecs.qmul.ac.uk/bird-audio-detection-challenge/ These labels were automatically derived from the annotations of avian flight call events in the BirdVox-full-night dataset.

Please Acknowledge UrbanSound8K in Academic Research

When BirdVox-70k is used for academic research, we would highly appreciate it if scientific publications of works partly based on this dataset cite the following publication:

The creation of this dataset was supported by NSF grants 1125098 (BIRDCAST) and 1633259 (BIRDVOX), a Google Faculty Award, the Leon Levy Foundation, and two anonymous donors.

Conditions of Use

Dataset created by Vincent Lostanlen, Justin Salamon, Andrew Farnsworth, Steve Kelling, and Juan Pablo Bello.

Feedback

Please help us improve BirdVox-DCASE-20k by sending your feedback to: | * Vincent Lostanlen: vincent.lostanlen@gmail.com for feedback regarding data pre-processing, | * Andrew Farnsworth:

af27@cornell.edu for feedback regarding data collection and ornithology, or | * Dan Stowell: dan.stowell@qmul.ac.uk for feedback regarding the DCASE "Bird Audio Detection" challenge.

In case of a problem, please include as many details as possible.

---

class soundata.datasets.dcase_birdVox20k.**Clip**(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

BirdVox20k Clip class

Parameters
clip_id (*str*) – id of the clip

Variables

- **audio** (*np.ndarray, float*) – path to the audio file
- **audio_path** (*str*) – path to the audio file
- **itemid** (*str*) – clip id
- **datasetid** (*str*) – the dataset to which the clip belongs to
- **hasbird** (*str*) – indication of whether the clips contains bird sounds (0/1)

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

property dataset_id

The clip's dataset ID.

Returns

- str - ID of the dataset from where this clip is extracted

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

property has_bird

The flag to tell whether the clip has bird sound or not.

Returns

- str - 1/0 depending on whether the clip contains bird sound

property item_id

The clip's item ID.

Returns

- str - ID of the clip

to_jams()

Get the clip's data in jams format

Returns
jams.JAMS – the clip's data in jams format

class soundata.datasets.dcase_birdVox20k.Dataset(*data_home=None*)

The BirdVox20k dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

cite()

Print the reference

clip_ids

Return clip ids

Returns
list – A list of clip ids

clipgroup_ids

Return clip ids

Returns
list – A list of clip ids

property default_path

Get the default path for the dataset

Returns
str – Local path to the dataset

download(*partial_download=None*, *force_overwrite=False*, *cleanup=False*)

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- **ValueError** – if invalid keys are passed to partial_download
- **IOError** – if a downloaded file's checksum is different from expected

explore_dataset(*clip_id=None*)

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters

clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

license()

Print the license

load_audio(**args*, ***kwargs*)

Load a BirdVox20k audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, 44100 Hz by default. If different from file's sample rate it will be resampled on load. Use None to load the file using its original sample rate (sample rate varies from file to file).

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

load_clipgroups()

Load all clipgroups in the dataset

Returns

dict – {*clipgroup_id*: clipgroup data}

Raises

NotImplementedError – If the dataset does not support Clipgroups

load_clips()

Load all clips in the dataset

Returns

dict – {*clip_id*: clip data}

Raises

NotImplementedError – If the dataset does not support Clips

validate(*verbose=True*)

Validate if the stored dataset is a valid version

Parameters

verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

soundata.datasets.dcase_birdVox20k.**load_audio**(*fhandle: BinaryIO*, *sr=44100*) → Tuple[numpy.ndarray, float]

Load a BirdVox20k audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, 44100 Hz by default. If different from file's sample rate it will be resampled on load. Use None to load the file using its original sample rate (sample rate varies from file to file).

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

4.5.8 EigenScape

EigenScape Dataset Loader

Dataset Info

EigenScape: a database of spatial acoustic scene recordings

Created By:

Marc Ciufo Green, Damian Murphy.
Audio Lab, Department of Electronic Engineering, University of York.

Version 2.0

Description:
EigenScape is a database of acoustic scenes recorded spatially using the mh Acoustics EigenMike. All scenes were recorded in 4th-order Ambisonics The database contains recordings of eight different location classes: Beach, Busy Street, Park, Pedestrian Zone, Quiet Street, Shopping Centre, Train Station, Woodland. The recordings were made in May 2017 at sites across the North of England.

Audio Files Included:

- 8 different examples of each location class were recorded over a duration of 10 minutes
- 64 recordings in total.
- ACN channel ordering with SN3D normalisation at 24-bit / 48 kHz resolution.

Annotations Included:

- No event labels associated with this dataset
- The metadata file gives more tempogeographic detail on each recording
- the EigenScape [recording map](http://bit.ly/EigenSMap) shows the locations and classes of all the recordings.
- No predefined training, validation, or testing splits.

Please Acknowledge EigenScape in Academic Research:

- **If you use this dataset please cite its original publication:**
  - Green MC, Murphy D. EigenScape: A database of spatial acoustic scene recordings. Applied Sciences. 2017 Nov;7(11):1204.

License:

- Creative Commons Attribution 4.0 International

***Important:**

- Use with caution. This loader "Engineers" a solution to obtain the correct files after Park6 and Park8 got mixed-up at the *eigenscape* and *eigenscape_raw* remotes. See the REMOTES and index if you want to understand how this engineered solution works. Also see the discussion about this engineered solution with the dataset author https://github.com/micarraylib/micarraylib/issues/8#issuecomment-1105357329

`class` soundata.datasets.eigenscape.**Clip**(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

Eigenscape Clip class

Parameters
clip_id (*str*) – id of the clip

Variables

- **tags** (*soundata.annotation.Tags*) – tag (scene label) of the clip + confidence.
- **audio_path** (*str*) – path to the audio file
- **clip_id** (*str*) – clip id
- **location** (*str*) – city were the audio signal was recorded
- **time** (*str*) – time when the audio signal was recorded
- **date** (*str*) – date when the audio signal was recorded
- **information** (*additional*) – notes included by the dataset authors with other details relevant to the specific clip

`property additional_information`

The clip's additional information.

Returns

- str - notes included by the dataset authors with other details relevant to the specific clip

`property audio: Optional[Tuple[numpy.ndarray, float]]`

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

`property date`

The clip's date.

Returns

- str - date when the audio signal was recorded

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

property location

The clip's location.

Returns

- str - Tags annotation object

property tags

The clip's tags

Returns

- annotations.Tags - Tags (scene label) of the clip + confidence.

property time

The clip's time.

Returns

- str - time when the audio signal was recorded

to_jams()

Get the clip's data in jams format

Returns
jams.JAMS – the clip's data in jams format

class soundata.datasets.eigenscape.**Dataset**(*data_home=None*)

The EigenScape dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

`cite()`

Print the reference

`clip_ids`

Return clip ids

Returns
list – A list of clip ids

`clipgroup_ids`

Return clip ids

Returns
list – A list of clip ids

`property default_path`

Get the default path for the dataset

Returns
str – Local path to the dataset

`download(partial_download=None, force_overwrite=False, cleanup=False)`

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- `ValueError` – if invalid keys are passed to partial_download
- `IOError` – if a downloaded file's checksum is different from expected

`explore_dataset(clip_id=None)`

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters
clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

`license()`

Print the license

`load_audio(*args, **kwargs)`

Load an EigenScape audio file. :Parameters: * **fhandle** (*str or file-like*) – file-like object or path to audio file

- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sampling rate of 48000 without resampling.

Returns

- np.ndarray - the audio signal

- float - The sample rate of the audio file

`load_clipgroups()`

Load all clipgroups in the dataset

Returns
dict – {*clipgroup_id*: clipgroup data}

Raises
NotImplementedError – If the dataset does not support Clipgroups

`load_clips()`

Load all clips in the dataset

Returns
dict – {*clip_id*: clip data}

Raises
NotImplementedError – If the dataset does not support Clips

`validate(verbose=True)`

Validate if the stored dataset is a valid version

Parameters
verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

`soundata.datasets.eigenscape.load_audio(fhandle: BinaryIO, sr=None)` → Tuple[numpy.ndarray, float]

Load an EigenScape audio file. :Parameters: * **fhandle** (*str or file-like*) – file-like object or path to audio file

- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sampling rate of 48000 without resampling.

Returns

- np.ndarray - the audio signal
- float - The sample rate of the audio file

4.5.9 EigenScape Raw

EigenScape Dataset Loader

Dataset Info

EigenScape: a database of spatial acoustic scene recordings

Created By:

Marc Ciufo Green, Damian Murphy.
Audio Lab, Department of Electronic Engineering, University of York.

Version raw

Description:

EigenScape is a database of acoustic scenes recorded spatially using the mh Acoustics EigenMike. All scenes in this format are in Raw format (A-format) with 32 channels The database contains recordings of eight different location classes: Beach, Busy Street, Park, Pedestrian Zone, Quiet Street, Shopping Centre, Train Station, Woodland. The recordings were made in May 2017 at sites across the North of England.

Audio Files Included:

- 8 different examples of each location class were recorded over a duration of 10 minutes
- 64 recordings in total.
- EigenMike channel ordering (32 total) with calibration and PGA level (captured with firewire interface and EigenStudio). 24-bit / 48 kHz resolution.

Annotations Included:

- No event labels associated with this dataset
- The metadata file gives more tempogeographic detail on each recording
- the EigenScape recording map shows the locations and classes of all the recordings.
- No predefined training, validation, or testing splits.

Please Acknowledge EigenScape in Academic Research:

- **If you use this dataset please cite its original publication:**
  - Green MC, Murphy D. EigenScape: A database of spatial acoustic scene recordings. Applied Sciences. 2017 Nov;7(11):1204.

License:

- Creative Commons Attribution 4.0 International

***Important:**

- Use with caution. This loader "Engineers" a solution to obtain the correct files after Park6 and Park8 got mixed-up at the *eigenscape* and *eigenscape_raw* remotes. See the REMOTES and index if you want to understand how this engineered solution works. Also see the discussion about this engineered solution with the dataset author https://github.com/micarraylib/micarraylib/issues/8#issuecomment-1105357329

`class soundata.datasets.eigenscape_raw.Clip(clip_id, data_home, dataset_name, index, metadata)`

Eigenscape Raw Clip class

Parameters

clip_id (*str*) – id of the clip

Variables

- **audio_path** (*str*) – path to the audio file
- **information** (*additional*) – notes included by the dataset authors with other details relevant to the specific clip
- **clip_id** (*str*) – clip id
- **date** (*str*) – date when the audio signal was recorded
- **location** (*str*) – city were the audio signal was recorded
- **tags** (*soundata.annotation.Tags*) – tag (scene label) of the clip + confidence.
- **time** (*str*) – time when the audio signal was recorded

property additional_information

The clip's additional information.

Returns

- str - notes included by the dataset authors with other details relevant to the specific clip

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

property date

The clip's date.

Returns

- str - date when the audio signal was recorded

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

property location

The clip's location.

Returns

- str - Tags annotation object

property tags

The clip's tags

Returns

- annotations.Tags - Tags (scene label) of the clip + confidence.

property time

00-23:59).

Returns

- str - time when the audio signal was recorded

Type
The clip's time (00

to_jams()

Get the clip's data in jams format

Returns
jams.JAMS – the clip's data in jams format

class soundata.datasets.eigenscape_raw.**Dataset**(*data_home=None*)

The EigenScape Raw dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns

Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns

Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

cite()

Print the reference

clip_ids

Return clip ids

Returns

list – A list of clip ids

clipgroup_ids

Return clip ids

Returns

list – A list of clip ids

property **default_path**

Get the default path for the dataset

Returns

str – Local path to the dataset

download(*partial_download=None*, *force_overwrite=False*, *cleanup=False*)

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- **ValueError** – if invalid keys are passed to partial_download
- **IOError** – if a downloaded file's checksum is different from expected

explore_dataset(*clip_id=None*)

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters
clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

license()

Print the license

load_audio(**args*, ***kwargs*)

Load an EigenScape Raw audio file. :Parameters: * **fhandle** (*str or file-like*) – file-like object or path to audio file

- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sampling rate of 48000 without resampling.

Returns

- np.ndarray - the audio signal
- float - The sample rate of the audio file

load_clipgroups()

Load all clipgroups in the dataset

Returns
dict – {*clipgroup_id*: clipgroup data}

Raises
NotImplementedError – If the dataset does not support Clipgroups

load_clips()

Load all clips in the dataset

Returns
dict – {*clip_id*: clip data}

Raises
NotImplementedError – If the dataset does not support Clips

validate(*verbose=True*)

Validate if the stored dataset is a valid version

Parameters
verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

soundata.datasets.eigenscape_raw.**load_audio**(*fhandle: BinaryIO*, *sr=None*) → Tuple[numpy.ndarray, float]

Load an EigenScape Raw audio file. :Parameters: * **fhandle** (*str or file-like*) – file-like object or path to audio file

- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sampling rate of 48000 without resampling.

Returns

- np.ndarray - the audio signal
- float - The sample rate of the audio file

4.5.10 ESC-50

ESC-50 Dataset Loader

Dataset Info

ESC-50: Dataset for Environmental Sound Classification

The ESC-50 dataset is a labeled collection of 2000 environmental audio recordings suitable for benchmarking methods of environmental sound classification. The total duration of the dataset is 2.8 hours (2000 x 5 seconds).

The dataset consists of 5-second-long recordings organized into 50 semantical classes (with 40 examples per class) loosely arranged into 5 major categories:

Animals Natural soundscapes & water sounds Human, non-speech sounds Interior/domestic sounds Exterior/urban noises Dog Rain Crying baby Door knock Helicopter Rooster Sea waves Sneezing Mouse click Chainsaw Pig Crackling fire Clapping Keyboard typing Siren Cow Crickets Breathing Door, wood creaks Car horn Frog Chirping birds Coughing Can opening Engine Cat Water drops Footsteps Washing machine Train Hen Wind Laughing Vacuum cleaner Church bells Insects (flying)Pouring water Brushing teeth Clock alarm Airplane Sheep Toilet flush Snoring Clock tick Fireworks Crow Thunderstorm Drinking, sipping Glass breaking Hand saw

Clips in this dataset have been manually extracted from public field recordings gathered by the Freesound.org project. The dataset has been prearranged into 5 folds for comparable cross-validation, making sure that fragments from the same original source file are contained in a single fold.

A more thorough description of the dataset is available in the original paper with some supplementary materials on GitHub:

https://github.com/karolpiczak/ESC-50

Repository content audio/*.wav

2000 audio recordings in WAV format (5 seconds, 44.1 kHz, mono) with the following naming convention:

{FOLD}-{CLIP_ID}-{TAKE}-{TARGET}.wav

{FOLD} - index of the cross-validation fold, {CLIP_ID} - ID of the original Freesound clip, {TAKE} - letter disambiguating between different fragments from the same Freesound clip, {TARGET} - class in numeric format [0, 49]. meta/esc50.csv

CSV file with the following structure:

filename fold target category esc10 src_file take

The esc10 column indicates if a given file belongs to the ESC-10 subset (10 selected classes, CC BY license).

https://github.com/karolpiczak/ESC-50/blob/master/meta/esc50-human.xlsx

Additional data pertaining to the crowdsourcing experiment (human classification accuracy).

---

class soundata.datasets.esc50.Clip(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

ESC-50 Clip class

Parameters

clip_id (*str*) – id of the clip

Variables

- **audio** (*np.ndarray, float*) – path to the audio file
- **audio_path** (*str*) – path to the audio file
- **category** (*str*) – clip class in string format, i.e., label
- **clip_id** (*str*) – clip id
- **esc10** (*bool*) – True if the clip belongs to the ESC-10 subset (10 selected classes, CC BY license)
- **filename** (*str*) – clip filename
- **fold** (*int*) – index of the cross-validation fold the clip belongs to
- **src_file** (*str*) – freesound ID of the original file from which the clip was taken
- **tags** (soundata.annotations.Tags) – tag (label) of the clip + confidence. In ESC-50 every clip has one tag.
- **take** (*str*) – letter disambiguating between different fragments from the same Freesound clip (e.g., "A", "B", etc.)
- **target** (*int*) – clip class in numeric format

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

property category

The clip's category.

Returns

- str - clip class in string format, i.e., label

property esc10

The clip's esc10.

Returns

- bool - True if the clip belongs to the ESC-10 subset (10 selected classes, CC BY license)

property filename

The clip's filename

Returns

- str - clip filename

property fold

The clip's fold

Returns

- int - index of the cross-validation fold the clip belongs to

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

property src_file

The clip's source file.

Returns

- str - freesound ID of the original file from which the clip was taken

property tags

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

property take

The clip's take

Returns

- str - letter disambiguating between different fragments from the same Freesound clip (e.g., "A", "B", etc.)

property target

The clip's target.

Returns

- int - clip class in numeric format

to_jams()

Get the clip's data in jams format

Returns
jams.JAMS – the clip's data in jams format

class soundata.datasets.esc50.Dataset(*data_home=None*)

The ESC-50 dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded

- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

cite()

Print the reference

clip_ids

Return clip ids

Returns
list – A list of clip ids

clipgroup_ids

Return clip ids

Returns
list – A list of clip ids

property default_path

Get the default path for the dataset

Returns
str – Local path to the dataset

download(*partial_download=None*, *force_overwrite=False*, *cleanup=False*)

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- **ValueError** – if invalid keys are passed to partial_download
- **IOError** – if a downloaded file's checksum is different from expected

explore_dataset(*clip_id=None*)

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters
clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

`license()`

Print the license

`load_audio(*args, **kwargs)`

Load an ESC-50 audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which loads the file using its original sample rate of 44100.

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

`load_clipgroups()`

Load all clipgroups in the dataset

Returns
dict – {*clipgroup_id*: clipgroup data}

Raises
`NotImplementedError` – If the dataset does not support Clipgroups

`load_clips()`

Load all clips in the dataset

Returns
dict – {*clip_id*: clip data}

Raises
`NotImplementedError` – If the dataset does not support Clips

`validate(verbose=True)`

Validate if the stored dataset is a valid version

Parameters
verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

`soundata.datasets.esc50.load_audio(fhandle: BinaryIO, sr=None)` → Tuple[numpy.ndarray, float]

Load an ESC-50 audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which loads the file using its original sample rate of 44100.

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

4.5.11 Freefield1010

freefield1010 Dataset Loader

Dataset Info

freefield1010: A Dataset of Field Recording Excerpts for Bioacoustic Research

Created By:

Dan Stowell, Mark D. Plumbley.
Centre for Digital Music, Queen Mary University of London.

Version 1.0

Description:
The freefield1010 dataset is a collection of 7,690 field recording excerpts from various global locations, standardized for research purposes. These recordings cover a wide range of environments and locales. The dataset is part of the "Bird Audio Detection" challenge, a joint venture by DCASE (Detection and Classification of Acoustic Scenes and Events) and the IEEE Signal Processing Society. It's particularly useful for bioacoustic classification models, with annotations indicating the presence or absence of birds in the recordings.

Audio Files Included:

- The dataset consists of 7,690 audio clips, sourced from the field-recording tag in the Freesound audio archive.
- All sounds have been converted to standard CD-quality mono WAV format.
- Files are stored as 16-bit 44.1 kHz WAV files in the 'wav' folder.
- Amplitude of each excerpt has been normalized due to the varying levels in the Freesound archive.

Meta-data Files Included:

- A binary label "hasbird" is associated with every recording.
- The metadata is available on the DCASE "Bird Audio Detection" challenge website: http://machine-listening.eecs.qmul.ac.uk/bird-audio-detection-challenge/

Please Acknowledge freefield1010 in Academic Research:
When using the freefield1010 dataset for academic research, please cite the following paper:

D. Stowell, M. Plumbley. "An open dataset for research on audio field recording archives: Freefield1010.", Proc. Audio Engineering Society 53rd Conference on Semantic Audio (AES53), 2014.

Conditions of Use:

- The freefield1010 dataset is created by Dan Stowell and Mark D. Plumbley.
- It is available under the Creative Commons Attribution 4.0 International (CC BY 4.0) license: https://creativecommons.org/licenses/by/4.0/

`class soundata.datasets.freefield1010.Clip`(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

freefield1010 Clip class

Parameters
clip_id (*str*) – id of the clip

Variables

- **audio** (*np.ndarray, float*) – path to the audio file

- **audio_path** (*str*) – path to the audio file
- **itemid** (*str*) – clip id
- **datasetid** (*str*) – the dataset to which the clip belongs to
- **hasbird** (*str*) – indication of whether the clips contains bird sounds (0/1)

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

property dataset_id

The clip's dataset ID.

Returns

- str - ID of the dataset from where this clip is extracted

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

property has_bird

The flag to tell whether the clip has bird sound or not.

Returns

- str - 1/0 depending on whether the clip contains bird sound

property item_id

The clip's item ID.

Returns

- str - ID of the clip

to_jams()

Get the clip's data in jams format

Returns
jams.JAMS – the clip's data in jams format

class soundata.datasets.freefield1010.Dataset(*data_home=None*)

The freefield1010 dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded

- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

cite()

Print the reference

clip_ids

Return clip ids

Returns
list – A list of clip ids

clipgroup_ids

Return clip ids

Returns
list – A list of clip ids

property default_path

Get the default path for the dataset

Returns
str – Local path to the dataset

download(*partial_download=None*, *force_overwrite=False*, *cleanup=False*)

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- **ValueError** – if invalid keys are passed to partial_download
- **IOError** – if a downloaded file's checksum is different from expected

explore_dataset(*clip_id=None*)

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters
clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

`license()`

Print the license

`load_audio(*args, **kwargs)`

Load a freefield1010 audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, 44100 Hz by default. If different from file's sample rate it will be resampled on load. Use None to load the file using its original sample rate (sample rate varies from file to file).

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

`load_clipgroups()`

Load all clipgroups in the dataset

Returns
dict – {*clipgroup_id*: clipgroup data}

Raises
`NotImplementedError` – If the dataset does not support Clipgroups

`load_clips()`

Load all clips in the dataset

Returns
dict – {*clip_id*: clip data}

Raises
`NotImplementedError` – If the dataset does not support Clips

`validate(verbose=True)`

Validate if the stored dataset is a valid version

Parameters
verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

`soundata.datasets.freefield1010.load_audio(fhandle: BinaryIO, sr=44100)` → Tuple[numpy.ndarray, float]

Load a freefield1010 audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, 44100 Hz by default. If different from file's sample rate it will be resampled on load. Use None to load the file using its original sample rate (sample rate varies from file to file).

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

4.5.12 FSD50K

FSD50K Dataset Loader

Dataset Info

FSD50K: an Open Dataset of Human-Labeled Sound Events

Created By:

Eduardo Fonseca, Xavier Favory, Jordi Pons, Frederic Font, Xavier Serra.
Music Technology Group, Universitat Pompeu Fabra (Barcelona).

Version 1.0

Description:
FSD50K is an open dataset of human-labeled sound events containing 51,197 Freesound clips unequally distributed in 200 classes drawn from the AudioSet Ontology. FSD50K has been created at the Music Technology Group of Universitat Pompeu Fabra.

Audio Files Included:

- FSD50K contains 51,197 audio clips from Freesound, totalling 108.3 hours of multi-labeled audio.
- The audio content is composed mainly of sound events produced by physical sound sources and production mechanisms, including human sounds, sounds of things, animals, natural sounds, musical instruments and more. The vocabulary can be inspected in vocabulary.csv.
- Clips are of variable length from 0.3 to 30s, due to the diversity of the sound classes and the preferences of Freesound users when recording sounds.
- All clips are provided as uncompressed PCM 16 bit 44.1 kHz mono audio files.

Annotations Included:

- The dataset encompasses 200 sound classes (144 leaf nodes and 56 intermediate nodes) hierarchically organized with a subset of the AudioSet Ontology. Please refer to the included vocabulary.csv file for a complete list of considered classes.
- The acoustic material has been manually labeled by humans following a data labeling process using the Freesound Annotator platform.
- Ground truth labels are provided at the clip-level (i.e., weak labels).
- Note: All classes in FSD50K are represented in AudioSet, except Crash cymbal, Human group actions, Human voice, Respiratory sounds, and Domestic sounds, home sounds.
- Note: We use a slightly different format than AudioSet for the naming of class labels in order to avoid potential problems with spaces, commas, etc. Example: we use Accelerating_and_revving_and_vroom instead of the original Accelerating, revving, vroom. You can go back to the original AudioSet naming using the information provided in vocabulary.csv (class label and mid for the 200 classes of FSD50K) and the AudioSet Ontology specification.

Organization:
FSD50K is split in two subsets: the developement (dev) and the evaluation (eval) sets. Especifications of both subsets is detailed below:

- **Dev set:**
  - 40,966 audio clips totalling 80.4 hours of audio
  - Avg duration/clip: 7.1s
  - 114,271 smeared labels (i.e., labels propagated in the upwards direction to the root of the ontology)
  - Labels are correct but could be occasionally incomplete
  - A train/validation split is provided. If a different split is used, it should be specified for reproducibility and fair comparability of results
- **Eval set:**
  - 10,231 audio clips totalling 27.9 hours of audio
  - Avg duration/clip: 9.8s
  - 38,596 smeared labels
  - Eval set is labeled exhaustively (labels are correct and complete for the considered vocabulary)

Ground-truth Files Included:

FSD50K ground-truth is represented through the following file structure:

- **dev.csv:**
  Each row (i.e. audio clip) of dev.csv contains the following information:
  - **fname:**
    The file name without the .wav extension, e.g., the fname 64760 corresponds to the file 64760.wav in disk. This number is the Freesound id. We always use Freesound ids as filenames.
  - **labels:**
    The class labels (i.e., the ground truth). Note these class labels are smeared, i.e., the labels have been propagated in the upwards direction to the root of the ontology. More details about the label smearing process can be found in Appendix D of our paper.
  - **mids:**
    The Freebase identifiers corresponding to the class labels, as defined in the AudioSet Ontology specification.
  - **split:**
    Whether the clip belongs to train or val (see paper for details on the proposed split)
- **eval.csv:**
  Rows in eval.csv follow the same format as dev.csv, except that there is no split column.

Metadata Files Included:

To allow a variety of analysis and approaches with FSD50K, we provide the following metadata:

- **class_info_FSD50K.json:**
  Python dictionary where each entry corresponds to one sound class and contains: FAQs utilized during the annotation of the class, examples (representative audio clips), and verification_examples (audio clips presented to raters during annotation as a quality control mechanism). Audio clips are described by the Freesound id. Note: It may be that some of these examples are not included in the FSD50K release.
- **dev_clips_info_FSD50K.json:**
  Python dictionary where each entry corresponds to one dev clip and contains: title, description, tags, clip license, and the uploader name. All these metadata are provided by the uploader.
- **eval_clips_info_FSD50K.json:**
  Same as above, but with eval clips.

- **pp_pnp_ratings.json:**
  Python dictionary where each entry corresponds to one clip in the dataset and contains the PP/PNP ratings for the labels associated with the clip. More specifically, these ratings are gathered for the labels validated in the validation task. This file includes 59,485 labels for the 51,197 clips in FSD50K. Out of these labels:
  - 56,095 labels have inter-annotator agreement (PP twice, or PNP twice). Each of these combinations can be occasionally accompanied by other (non-positive) ratings.
  - 3390 labels feature other rating configurations such as i) only one PP rating and one PNP rating (and nothing else). This can be considered inter-annotator agreement at the "Present" level; ii) only one PP rating (and nothing else); iii) only one PNP rating (and nothing else).

  Ratings' legend: PP=1; PNP=0.5; U=0; NP=-1.

  Note: The PP/PNP ratings have been provided in the validation task. Subsequently, a subset of these clips corresponding to the eval set was exhaustively labeled in the refinement task, hence receiving additional labels in many cases. For these eval clips, you might want to check their labels in eval.csv in order to have more info about their audio content.

collection folder:

This folder contains metadata for what we call the sound collection format. This format consists of the raw annotations gathered, featuring all generated class labels without any restriction. We provide the collection format to make available some annotations that do not appear in the FSD50K ground truth release. This typically happens in the case of classes for which we gathered human-provided annotations, but that were discarded in the FSD50K release due to data scarcity (more specifically, they were merged with their parents). In other words, the main purpose of the collection format is to make available annotations for tiny classes. The format of these files in analogous to that of the files in FSD50K.ground_truth/. A couple of examples show the differences between collection and ground truth formats:

- clip: labels_in_collection - labels_in_ground_truth
  - 51690: Owl - Bird,Wild_Animal,Animal
  - 190579: Toothbrush,Electric_toothbrush - Domestic_sounds_and_home_sounds

In the first example, raters provided the label Owl. However, due to data scarcity, Owl labels were merged into their parent Bird. Then, labels Wild_Animal,Animal were added via label propagation (smearing). The second example shows one of the most extreme cases, where raters provided the labels Electric_toothbrush,Toothbrush, which both had few data. Hence, they were merged into Toothbrush's parent, which unfortunately is Domestic_sounds_and_home_sounds (a rather vague class containing a variety of children sound classes).

Note: Labels in the collection format are not smeared.

Note: While in FSD50K's ground truth the vocabulary encompasses 200 classes (common for dev and eval), since the collection format is composed of raw annotations, the vocabulary here is much larger (over 350 classes), and it is slightly different in dev and eval.

Please Acknowledge FSD50K in Academic Research:

If you use the FSD50K Dataset please cite the following paper:

- Eduardo Fonseca, Xavier Favory, Jordi Pons, Frederic Font, Xavier Serra. "FSD50K: an Open Dataset of Human-Labeled Sound Events", arXiv:2010.00475, 2020.

The authors would like to thank everyone who contributed to FSD50K with annotations, and especially Mercedes Collado, Ceren Can, Rachit Gupta, Javier Arredondo, Gary Avendano and Sara Fernandez for their commitment and perseverance. The authors would also like to thank Daniel P.W. Ellis and Manoj Plakal from Google Research for valuable discussions. This work is partially supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 688382 AudioCommons, and two Google Faculty Research Awards 2017 and 2018, and the Maria de Maeztu Units of Excellence Programme (MDM-2015-0502).

License:

All audio clips in FSD50K are released under Creative Commons (CC) licenses. Each clip has its own license as defined by the clip uploader in Freesound, some of them requiring attribution to their original authors and some forbidding further commercial reuse. For attribution purposes and to facilitate attribution of these files to third parties, we include a mapping from the audio clips to their corresponding licenses. The licenses are specified in the files dev_clips_info_FSD50K.json and eval_clips_info_FSD50K.json. These licenses are CC0, CC-BY, CC-BY-NC and CC Sampling+.

In addition, FSD50K as a whole is the result of a curation process and it has an additional license: FSD50K is released under CC-BY. This license is specified in the LICENSE-DATASET file downloaded with the FSD50K.doc zip file.

Usage of FSD50K for commercial purposes: If you'd like to use FSD50K for commercial purposes, please contact Eduardo Fonseca and Frederic Font at eduardo.fonseca@upf.edu and frederic.font@upf.edu.

Feedback:

For further questions, please contact eduardo.fonseca@upf.edu, or join the freesound-annotator Google Group.

---

class soundata.datasets.fsd50k.**Clip**(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

FSD50K Clip class

Parameters

clip_id (*str*) – id of the clip

Variables

- **audio** (*np.ndarray, float*) – path to the audio file
- **audio_path** (*str*) – path to the audio file
- **clip_id** (*str*) – clip id
- **description** (*str*) – description of the sound provided by the Freesound uploader
- **mids** (soundata.annotations.Tags) – tag (labels) encoded in Audioset formatting
- **pp_pnp_ratings** (*dict*) – PP/PNP ratings given to the main label of the clip
- **split** (*str*) – flag to identify if clip belongs to developement, evaluation or validation splits
- **tags** (soundata.annotations.Tags) – tag (label) of the clip + confidence
- **title** (*str*) – the title of the uploaded file in Freesound

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio.

Returns

- np.ndarray - audio signal
- float - sample rate

property description

The clip's description.

Returns

- str - description of the sound provided by the Freesound uploader

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

`property mids`

The clip's mids.

Returns

- annotations.Tags - tag (labels) encoded in Audioset formatting

`property pp_pnp_ratings`

The clip's PP/PNP ratings.

Returns

- dict - PP/PNP ratings given to the main label of the clip

`property split`

The clip's split.

Returns

- str - flag to identify if clip belongs to developement, evaluation or validation splits

`property tags`

The clip's tags.

Returns

- annotations.Tags - tag (label) of the clip + confidence

`property title`

The clip's title.

Returns

- str - the title of the uploaded file in Freesound

`to_jams()`

Get the clip's data in jams format

Returns
jams.JAMS – the clip's data in jams format

`class soundata.datasets.fsd50k.Dataset(data_home=None)`

The FSD50K dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

cite()

Print the reference

clip_ids

Return clip ids

Returns
list – A list of clip ids

clipgroup_ids

Return clip ids

Returns
list – A list of clip ids

property default_path

Get the default path for the dataset

Returns
str – Local path to the dataset

download(*partial_download=None*, *force_overwrite=False*, *cleanup=False*)

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- **ValueError** – if invalid keys are passed to partial_download
- **IOError** – if a downloaded file's checksum is different from expected

explore_dataset(*clip_id=None*)

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters
clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

license()

Print the license

`load_audio`(**args*, ***kwargs*)

Load a FSD50K audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, 44100 Hz by default. If different from file's sample rate it will be resampled on load. Use None to load the file using its original sample rate (sample rate varies from file to file).

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

`load_clipgroups`()

Load all clipgroups in the dataset

Returns
dict – {*clipgroup_id*: clipgroup data}

Raises
`NotImplementedError` – If the dataset does not support Clipgroups

`load_clips`()

Load all clips in the dataset

Returns
dict – {*clip_id*: clip data}

Raises
`NotImplementedError` – If the dataset does not support Clips

`load_fsd50k_vocabulary`(**args*, ***kwargs*)

Load vocabulary of FSD50K to relate FSD50K labels with AudioSet onthology

Parameters
data_path (*str*) – Path to the vocabulary file

Returns
** fsd50k_to_audioset (dict)* – vocabulary to convert FSD50K to AudioSet * audioset_to_fsd50k (dict): vocabulary to convert from AudioSet to FSD50K

`load_ground_truth`(**args*, ***kwargs*)

Load ground truth files of FSD50K

Parameters
data_path (*str*) – Path to the ground truth file

Returns
** ground_truth_dict (dict)* – ground truth dict of the clips in the input split * clip_ids (list): list of clip ids of the input split

`validate`(*verbose=True*)

Validate if the stored dataset is a valid version

Parameters
verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally

- list - files which have an invalid checksum

soundata.datasets.fsd50k.**load_audio**(*fhandle: BinaryIO*, *sr=None*) → Tuple[numpy.ndarray, float]

Load a FSD50K audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, 44100 Hz by default. If different from file's sample rate it will be resampled on load. Use None to load the file using its original sample rate (sample rate varies from file to file).

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

soundata.datasets.fsd50k.**load_fsd50k_vocabulary**(*data_path*)

Load vocabulary of FSD50K to relate FSD50K labels with AudioSet onthology

Parameters

data_path (*str*) – Path to the vocabulary file

Returns

** fsd50k_to_audioset (dict)* – vocabulary to convert FSD50K to AudioSet * audioset_to_fsd50k (dict): vocabulary to convert from AudioSet to FSD50K

soundata.datasets.fsd50k.**load_ground_truth**(*data_path*)

Load ground truth files of FSD50K

Parameters

data_path (*str*) – Path to the ground truth file

Returns

** ground_truth_dict (dict)* – ground truth dict of the clips in the input split * clip_ids (list): list of clip ids of the input split

4.5.13 FSDnoisy18K

FSDnoisy18K Dataset Loader

Dataset Info

Created By:

Eduardo Fonseca, Mercedes Collado, Manoj Plakal, Daniel P. W. Ellis, Frederic Font, Xavier Favory, Xavier Serra.

Music Technology Group, Universitat Pompeu Fabra (Barcelona).

Version 1.0

Description:

FSDnoisy18k is an audio dataset collected with the aim of fostering the investigation of label noise in sound event classification. It contains 42.5 hours of audio across 20 sound classes, including a small amount of manually-labeled data and a larger quantity of real-world noisy data.

What follows is a summary of the most basic aspects of FSDnoisy18k. For a complete description of FSDnoisy18k, make sure to check:

- The FSDnoisy18k companion site: http://www.eduardofonseca.net/FSDnoisy18k/
- The description provided in Section 2 of our ICASSP 2019 paper

FSDnoisy18k is an audio dataset collected with the aim of fostering the investigation of label noise in sound event classification. It contains 42.5 hours of audio across 20 sound classes, including a small amount of manually-labeled data and a larger quantity of real-world noisy data.

The source of audio content is Freesound—a sound sharing site created an maintained by the Music Technology Group hosting over 400,000 clips uploaded by its community of users, who additionally provide some basic metadata (e.g., tags, and title). The 20 classes of FSDnoisy18k are drawn from the AudioSet Ontology and are selected based on data availability as well as on their suitability to allow the study of label noise. The 20 classes are: "Acoustic guitar", "Bass guitar", "Clapping", "Coin (dropping)", "Crash cymbal", "Dishes, pots, and pans", "Engine", "Fart", "Fire", "Fireworks", "Glass", "Hi-hat", "Piano", "Rain", "Slam", "Squeak", "Tearing", "Walk, footsteps", "Wind", and "Writing". FSDnoisy18k was created with the Freesound Annotator, which is a platform for the collaborative creation of open audio datasets.

We defined a clean portion of the dataset consisting of correct and complete labels. The remaining portion is referred to as the noisy portion. Each clip in the dataset has a single ground truth label (singly-labeled data).

The clean portion of the data consists of audio clips whose labels are rated as present in the clip and predominant (almost all with full inter-annotator agreement), meaning that the label is correct and, in most cases, there is no additional acoustic material other than the labeled class. A few clips may contain some additional sound events, but they occur in the background and do not belong to any of the 20 target classes. This is more common for some classes that rarely occur alone, e.g., "Fire", "Glass", "Wind" or "Walk, footsteps".

The noisy portion of the data consists of audio clips that received no human validation. In this case, they are categorized on the basis of the user-provided tags in Freesound. Hence, the noisy portion features a certain amount of label noise.

Included files and statistics:

- FSDnoisy18k contains 18,532 audio clips (42.5h) unequally distributed in the 20 aforementioned classes drawn from the AudioSet Ontology.
- The audio clips are provided as uncompressed PCM 16 bit, 44.1 kHz, mono audio files.
- The audio clips are of variable length ranging from 300ms to 30s, and each clip has a single ground truth label (singly-labeled data).
- The dataset is split into a test set and a train set. The test set is drawn entirely from the clean portion, while the remainder of data forms the train set.
- The train set is composed of 17,585 clips (41.1h) unequally distributed among the 20 classes. It features a clean subset and a noisy subset. In terms of number of clips their proportion is 10%/90%, whereas in terms of duration the proportion is slightly more extreme (6%/94%). The per-class percentage of clean data within the train set is also imbalanced, ranging from 6.1% to 22.4%. The number of audio clips per class ranges from 51 to 170, and from 250 to 1000 in the clean and noisy subsets, respectively. Further, a noisy small subset is defined, which includes an amount of (noisy) data comparable (in terms of duration) to that of the clean subset.
- The test set is composed of 947 clips (1.4h) that belong to the clean portion of the data. Its class distribution is similar to that of the clean subset of the train set. The number of per-class audio clips in the test set ranges from 30 to 72. The test set enables a multi-class classification problem.
- FSDnoisy18k is an expandable dataset that features a per-class varying degree of types and amount of label noise. The dataset allows investigation of label noise as well as other approaches, from semi-supervised learning, e.g., self-training to learning with minimal supervision.

Additional code:

We've released the code for our ICASSP 2019 paper at https://github.com/edufonseca/icassp19. The framework

comprises all the basic stages: feature extraction, training, inference and evaluation. After loading the FSDnoisy18k dataset, log-mel energies are computed and a CNN baseline is trained and evaluated. The code also allows to test four noise-robust loss functions. Please check our paper for more details.

Label noise characteristics:

FSDnoisy18k features real label noise that is representative of audio data retrieved from the web, particularly from Freesound. The analysis of a per-class, random, 15% of the noisy portion of FSDnoisy18k revealed that roughly 40% of the analyzed labels are correct and complete, whereas 60% of the labels show some type of label noise. Please check the FSDnoisy18k companion site for a detailed characterization of the label noise in the dataset, including a taxonomy of label noise for singly-labeled data as well as a per-class description of the label noise.

Relevant links:

- Source code for our preprint: https://github.com/edufonseca/icassp19
- Freesound Annotator: https://annotator.freesound.org/
- Freesound: https://freesound.org
- Eduardo Fonseca's personal website: http://www.eduardofonseca.net/

Please Acknowledge FSDnoisy18K in Academic Research:

If you use the FSDnoisy18K Dataset please cite the following paper:

- Eduardo Fonseca, Manoj Plakal, Daniel P. W. Ellis, Frederic Font, Xavier Favory, and Xavier Serra, "Learning Sound Event Classifiers from Web Audio with Noisy Labels", arXiv preprint arXiv:1901.01189, 2019

This work is partially supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 688382 AudioCommons. Eduardo Fonseca is also sponsored by a Google Faculty Research Award 2017. We thank everyone who contributed to FSDnoisy18k with annotations.

License:

FSDnoisy18k has licenses at two different levels, as explained next. All sounds in Freesound are released under Creative Commons (CC) licenses, and each audio clip has its own license as defined by the audio clip uploader in Freesound. In particular, all Freesound clips included in FSDnoisy18k are released under either CC-BY or CC0. For attribution purposes and to facilitate attribution of these files to third parties, we include a relation of audio clips and their corresponding license in the LICENSE-INDIVIDUAL-CLIPS file downloaded with the dataset.

In addition, FSDnoisy18k as a whole is the result of a curation process and it has an additional license. FSDnoisy18k is released under CC-BY. This license is specified in the LICENSE-DATASET file downloaded with the dataset.

Feedback:

For further questions, please contact eduardo.fonseca@upf.edu, or join the freesound-annotator Google Group.

`class soundata.datasets.fsdnoisy18k.Clip(clip_id, data_home, dataset_name, index, metadata)`

FSDnoisy18K Clip class

Parameters

clip_id (*str*) – id of the clip

Variables

- **audio** (*np.ndarray, float*) – path to the audio file
- **aso_id** (*str*) – the id of the corresponding category as per the AudioSet Ontology
- **audio_path** (*str*) – path to the audio file
- **clip_id** (*str*) – clip id

- **manually_verified** (*int*) – flag to indicate whether the clip belongs to the clean portion (1), or to the noisy portion (0) of the train set
- **noisy_small** (*int*) – flag to indicate whether the clip belongs to the noisy_small portion (1) of the train set
- **split** (*str*) – flag to indicate whether the clip belongs the train or test split
- **tag** (soundata.annotations.Tags) – tag (label) of the clip + confidence

property aso_id

The clip's Audioset ontology ID.

Returns

- str - the id of the corresponding category as per the AudioSet Ontology

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

property manually_verified

The clip's manually annotated flag.

Returns

- int - flag to indicate whether the clip belongs to the clean portion (1), or to the noisy portion (0) of the train set

property noisy_small

The clip's noisy flag.

Returns

- int - flag to indicate whether the clip belongs to the noisy_small portion (1) of the train set

property split

The clip's split.

Returns

- str - flag to indicate whether the clip belongs the train or test split

property tags

The clip's tags.

Returns

- annotations.Tags - tag (label) of the clip + confidence

to_jams()

Get the clip's data in jams format

Returns
jams.JAMS – the clip's data in jams format

class soundata.datasets.fsdnoisy18k.Dataset(*data_home=None*)

The FSDnoisy18K dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

cite()

Print the reference

clip_ids

Return clip ids

Returns
list – A list of clip ids

clipgroup_ids

Return clip ids

Returns
list – A list of clip ids

property default_path

Get the default path for the dataset

Returns
str – Local path to the dataset

download(*partial_download=None*, *force_overwrite=False*, *cleanup=False*)

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- **ValueError** – if invalid keys are passed to partial_download
- **IOError** – if a downloaded file's checksum is different from expected

explore_dataset(*clip_id=None*)

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters

clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

license()

Print the license

load_audio(**args*, ***kwargs*)

Load a FSDnoisy18K audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, 44100 Hz by default. If different from file's sample rate it will be resampled on load. Use None to load the file using its original sample rate (sample rate varies from file to file).

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

load_clipgroups()

Load all clipgroups in the dataset

Returns

dict – {*clipgroup_id*: clipgroup data}

Raises

NotImplementedError – If the dataset does not support Clipgroups

load_clips()

Load all clips in the dataset

Returns

dict – {*clip_id*: clip data}

Raises

NotImplementedError – If the dataset does not support Clips

validate(*verbose=True*)

Validate if the stored dataset is a valid version

Parameters

verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

soundata.datasets.fsdnoisy18k.**load_audio**(*fhandle: BinaryIO*, *sr=None*) → Tuple[numpy.ndarray, float]

Load a FSDnoisy18K audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, 44100 Hz by default. If different from file's sample rate it will be resampled on load. Use None to load the file using its original sample rate (sample rate varies from file to file).

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

4.5.14 SINGA:PURA

SINGA:PURA Dataset Loader

Dataset Info

SINGA:PURA (SINGApore: Polyphonic URban Audio) v1.0a

Created by:

- Kenneth Ooi, Karn N. Watcharasupat, Santi Peksi, Furi Andi Karnapi, Zhen-Ting Ong, Danny Chua, Hui-Wen Leow, Li-Long Kwok, Xin-Lei Ng, Zhen-Ann Loh, Woon-Seng Gan
- Digital Signal Processing Laboratory, School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore.

Description:

The SINGA:PURA (SINGApore: Polyphonic URban Audio) dataset is a strongly-labelled polyphonic urban sound dataset with spatiotemporal context. The dataset contains 6547 strongly-labelled and 72406 unlabelled recordings from a wireless acoustic sensor network deployed in Singapore to identify and mitigate noise sources in Singapore. The strongly-labelled and unlabelled recordings are disjoint, so there are a total of 78953 unique recordings. The recordings are all 10 seconds in length, and may have 1 or 7 channels, depending on the recording device used to record them. Total duration for the labelled subset provided here is 18.2 hours.

For full details regarding the sensor units used, the recording conditions, and annotation methodology, please refer to our conference paper.

Annotations:

Our label taxonomy is derived from the taxonomy used in the SONYC-UST datasets, but has been adapted to fit the local (Singapore) context while retaining compatibility with the SONYC-UST ontonology. We chose this taxonomy to allow the SINGA:PURA dataset to be used in conjunction with the SONYC-UST datasets when training urban sound tagging models by simply omitting the labels that are absent in the SONYC-UST taxonomy from the recordings in the SINGA:PURA dataset.

Specifically, our label taxonomy consists of 14 coarse-grained classes and 40 fine-grained classes. Their organisation is as follows:

1. **Engine**
    1. Small engine

    2. Medium engine
    3. Large engine
2. **Machinery impact**
    1. Rock drill
    2. Jackhammer
    3. Hoe ram
    4. Pile driver
3. **Non-machinery impact**
    1. Glass breaking (*)
    2. Car crash (*)
    3. Explosion (*)
4. **Powered saw**
    1. Chainsaw
    2. Small/medium rotating saw
    3. Large rotating saw
5. **Alert signal**
    1. Car horn
    2. Car alarm
    3. Siren
    4. Reverse beeper
6. **Music**
    1. Stationary music
    2. Mobile music
7. **Human voice**
    1. Talking
    2. Shouting
    3. Large crowd
    4. Amplified speech
    5. Singing (*)
8. **Human movement (*)**
    1. Footsteps (*)
    2. Clapping (*)
9. **Animal (*)**
    1. Dog barking
    2. Bird chirping (*)
    3. Insect chirping (*)

10. **Water (*)**
    1. Hose pump (*)
11. **Weather (*)**
    1. Rain (*)
    2. Thunder (*)
    3. Wind (*)
12. **Brake (*)**
    1. Friction brake (*)
    2. Exhaust brake (*)
13. **Train (*)**
    1. Electric train (*)

X. **Others (*)**
    1. Screeching (*)
    2. Plastic crinkling (*)
    3. Cleaning (*)
    4. Gear (*)

Classes marked with an asterisk (*) are present in the SINGA:PURA taxonomy but not the SONYC taxonomy. The "Ice cream truck" class from the SONYC taxonomy has been excluded from the SINGA:PURA taxonomy because this class does not exist in the local context.

In addition, note that the label for the coarse-grained class "Others" in the soundata loader is "0", which is different from the label "X" that is used in the full version of the SINGA:PURA dataset.

This dataset is also accessible via:

- Zenodo (labelled subset only): https://zenodo.org/record/5645825
- DR-NTU (all): https://researchdata.ntu.edu.sg/dataset.xhtml?persistentId=doi:10.21979/N9/Y8UQ6F

Please Acknowledge SINGA:PURA in Academic Research:

If you use this dataset please cite its original publication:

K. Ooi, K. N. Watcharasupat, S. Peksi, F. A. Karnapi, Z.-T. Ong, D. Chua, H.-W. Leow, L.-L. Kwok, X.-L. Ng, Z.-A. Loh, W.-S. Gan, "A Strongly-Labelled Polyphonic Dataset of Urban Sounds with Spatiotemporal Context," in Proceedings of the 13th Asia Pacific Signal and Information Processing Association Annual Summit and Conference, 2021.

License:

`class soundata.datasets.singapura.Clip(`*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*`)`

Parameters

clip_id (*str*) – clip id of the clip

Variables

- **clip_id** (*str*) – clip id
- **audio** (*np.ndarray, float*) – audio data

- **audio_path** (*str*) – path to the audio file
- **events** (annotations.MultiAnnotator) – sound events with start time, end time, label and confidence
- **annotation_path** (*str*) – path to the annotation file
- **sensor_id** (*str*) – sensor_id of the device used to record the data
- **town** (*str*) – town in Singapore where the sensor is located
- **timestamp** (*np.datetime*) – timestamp of the recording
- **dotw** (*int*) – day of the week when the clip was recorded, starting from 0 for Sunday

property audio

The clip's audio

Returns

- np.ndarray - audio signal

property dotw: int

The clip's day of the week

Returns

- int - day of the week when the clip was recorded, starting from 0 for Sunday

events

The clip's event annotations

Returns

- annotations.MultiAnnotator - sound events with start time, end time, label and confidence

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

property sensor_id: str

The clip's sensor ID

Returns

- str - sensor_id of the device used to record the data

property timestamp: numpy.datetime64

The clip's timestamp

Returns

- np.datetime64 - timestamp of the clip

to_jams()

Jams: the clip's data in jams format

property town: str

The clip's location

Returns

- str - location of the sensor, one of {'East 1', 'East 2', 'West 1', 'West 2'}

class soundata.datasets.singapura.Dataset(*data_home=None*)

SINGA:PURA v1.0 dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns

Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns

Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

cite()

Print the reference

clip_ids

Return clip ids

Returns

list – A list of clip ids

clipgroup_ids

Return clip ids

Returns

list – A list of clip ids

property default_path

Get the default path for the dataset

Returns

str – Local path to the dataset

download(*partial_download=None*, *force_overwrite=False*, *cleanup=False*)

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- **ValueError** – if invalid keys are passed to partial_download
- **IOError** – if a downloaded file's checksum is different from expected

explore_dataset(*clip_id=None*)

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters

clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

license()

Print the license

load_annotation(**args*, ***kwargs*)

Load an annotation file.

Parameters

fhandle (*str or file-like*) – path or file-like object pointing to an annotation file

Returns

- annotations.MultiAnnotator - sound events with start time, end time, label and confidence

load_audio(**args*, ***kwargs*)

Load a Example audio file.

Parameters

fhandle (*str or file-like*) – path or file-like object pointing to an audio file

Returns

- np.ndarray - the audio signal at 44.1 kHz

load_clipgroups()

Load all clipgroups in the dataset

Returns

dict – {*clipgroup_id*: clipgroup data}

Raises

NotImplementedError – If the dataset does not support Clipgroups

load_clips()

Load all clips in the dataset

Returns

dict – {*clip_id*: clip data}

Raises

NotImplementedError – If the dataset does not support Clips

validate(*verbose=True*)

Validate if the stored dataset is a valid version

Parameters
verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

soundata.datasets.singapura.**load_annotation**(*fhandle: TextIO*) → *MultiAnnotator*

Load an annotation file.

Parameters
fhandle (*str or file-like*) – path or file-like object pointing to an annotation file

Returns

- annotations.MultiAnnotator - sound events with start time, end time, label and confidence

soundata.datasets.singapura.**load_audio**(*fhandle*)

Load a Example audio file.

Parameters
fhandle (*str or file-like*) – path or file-like object pointing to an audio file

Returns

- np.ndarray - the audio signal at 44.1 kHz

4.5.15 STARSS 2022

Sony-TAu Realistic Spatial Soundscapes (STARSS) 2022 Dataset Loader

---

Dataset Info

*Sony-TAu Realistic Spatial Soundscapes: sound scenes in various rooms and environments, together with temporal and spatial annotations of prominent events belonging to a set of target classes.

Created By:

Archontis Politis, Parthasaarathy Sudarsanam, Sharath Adavanne, Daniel Krause, Tuomas Virtanen
Audio Research Group, Tampere University (Finland).

Yuki Mitsufuji, Kazuki Shimada, Naoya Takahashi, Yuichiro Koyama, Shusuke Takahashi
SONY

Version 1.0.0

Description:
Contains multichannel recordings of sound scenes in various rooms and environments, together with temporal and spatial annotations of prominent events belonging to a set of target classes. The dataset is collected in two different countries, in Tampere, Finland by the Audio Researh Group (ARG) of Tampere University (TAU), and in Tokyo, Japan by SONY, using a similar setup and annotation procedure. The dataset is delivered in two 4-channel spatial recording formats, a microphone array one (MIC), and first-order Ambisonics one (FOA). These

recordings serve as the development dataset for the DCASE 2022 Sound Event Localization and Detection Task of the DCASE 2022 Challenge.

Contrary to the three previous datasets of synthetic spatial sound scenes of

- TAU Spatial Sound Events 2019 (development/evaluation),
- TAU-NIGENS Spatial Sound Events 2020, and
- TAU-NIGENS Spatial Sound Events 2021

associated with the previous iterations of the DCASE Challenge, the STARS22 dataset contains recordings of real sound scenes and hence it avoids some of the pitfalls of synthetic generation of scenes. Some such key properties are:

- annotations are based on a combination of human annotators for sound event activity and optical tracking for spatial positions,
- the annotated target event classes are determined by the composition of the real scenes,
- the density, polyphony, occurences and co-occurences of events and sound classes is not random, and it follows actions and interactions of participants in the real scenes.

The recordings were collected between September 2021 and January 2022. Collection of data from the TAU side has received funding from Google.

Audio Files Included:

- 70 recording clips of 30 sec ~ 5 min durations, with a total time of ~2hrs, contributed by SONY (development dataset).
- 51 recording clips of 1 min ~ 5 min durations, with a total time of ~3hrs, contributed by TAU (development dataset).
- 40 recordings contributed by SONY for the training split, captured in 2 rooms (dev-train-sony).
- 30 recordings contributed by SONY for the testing split, captured in 2 rooms (dev-test-sony).
- 27 recordings contributed by TAU for the training split, captured in 4 rooms (dev-train-tau).
- 24 recordings contributed by TAU for the testing split, captured in 3 rooms (dev-test-tau).
- A total of 11 unique rooms captured in the recordings, 4 from SONY and 7 from TAU (development set).
- Sampling rate 24kHz.
- Two 4-channel 3-dimensional recording formats: first-order Ambisonics (FOA) and tetrahedral microphone array (MIC).
- Recordings are taken in two different countries and two different sites.
- Each recording clip is part of a recording session happening in a unique room.
- Groups of participants, sound making props, and scene scenarios are unique for each session (with a few exceptions).
- 13 target classes are identified in the recordings and strongly annotated by humans.
- Spatial annotations for those active events are captured by an optical tracking system.
- Sound events out of the target classes are considered as interference and are not labeled.

Annotations Included:

- Each recording in the development set has labels of events and DoAs in a plain csv file with the same filename.
- Each row in the csv file has a frame number, active class index, source number index, azimuth, and elevation.

- Frame, class, and source enumeration begins at 0.
- Frames correspond to a temporal resolution of 100msec.
- Azimuth and elevation angles are given in degrees, rounded to the closest integer value, with azimuth and elevation being zero at the front, azimuth $\phi \in [-180°, 180°]$, and elevation $\theta \in [-90°, 90°]$. Note that the azimuth angle is increasing counter-clockwise ($\phi = 90°$ at the left).
- The source index is a unique integer for each source in the scene, and it is provided only as additional information. Note that each unique actor gets assigned one such identifier, but not individual events produced by the same actor; e.g. a clapping event and a laughter event produced by the same person have the same identifier. Independent sources that are not actors (e.g. a loudspeaker playing music in the room) get a 0 identifier. Note that source identifier information is only included in the development metadata and is not required to be provided by the participants in their results.
- Overlapping sound events are indicated with duplicate frame numbers, and can belong to a different or the same class.

Organization

- The development dataset is split in training and test sets.
- The training set consists of 67 recordings.
- The test set consists of 54 recordings.

Please Acknowledge Sony-TAu Realistic Spatial Soundscapes (STARSS) 2022 in Academic Research:

- If you use this dataset please cite the report on its creation, and the corresponding DCASE2022 task setup:
  - Politis, Adavanne, Mitsufuji, Yuki, Sudarsanam, Parthasaarathy, Shimada, Kazuki, Adavanne, Sharath, Koyama, Yuichiro, Krause, Daniel, Takahashi, Naoya, Takahashi, Shusuke, & Virtanen, Tuomas. (2022). STARSS22: Sony-TAu Realistic Spatial Soundscapes 2022 dataset (1.0.0) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.6387880

License:

- This dataset is licensed under the [MIT](https://opensource.org/licenses/MIT) license

---

class soundata.datasets.starss2022.**Clip**(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

STARSS 2022 Clip class :Parameters: **clip_id** (*str*) – id of the clip

Variables

- **audio_path** (*str*) – path to the audio file
- **csv_path** (*str*) – path to the csv file
- **format** (*str*) – whether the clip is in FOA or MIC format
- **set** (*str*) – the data subset the clip belongs to (development or evaluation)
- **split** (*str*) – the set slip the clip belongs to (training or test)
- **clip_id** (*str*) – clip id
- **spatial_events** (SpatialEvents) – sound events with time step, elevation, azimuth, distance, label, clip_number and confidence.

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio :returns:

- np.ndarray - audio signal
- float - sample rate

`get_path`(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

`spatial_events`

The clip's event annotations :returns:

- **SpatialEvents with attributes**
  - **intervals (list): list of size n np.ndarrays of shape (m, 2), with intervals** (as floats) in TIME_UNITS in the form [start_time, end_time]
  - intervals_unit (str): intervals unit, one of TIME_UNITS
  - time_step (int, float, or None): the time-step between events
  - **elevations (list): list of size n with np.ndarrays with dtype int,** indicating the elevation of the sound event per time_step.
  - elevations_unit (str): elevations unit, one of ELEVATIONS_UNITS
  - **azimuths (list): list of size n with np.ndarrays with dtype int,** indicating the azimuth of the sound event per time_step if moving
  - azimuths_unit (str): azimuths unit, one of AZIMUTHS_UNITS
  - **distances (list): list of size n with np.ndarrays with dtype int,** indicating the distance of the sound event per time_step if moving
  - distances_unit (str): distances unit, one of DISTANCES_UNITS
  - labels (list): list of event labels (as strings)
  - labels_unit (str): labels unit, one of LABELS_UNITS
  - clip_number_indices (list): list of clip number indices (as strings)
  - confidence (np.ndarray or None): array of confidence values

`to_jams`()

Get the clip's data in jams format :returns: *jams.JAMS* – the clip's data in jams format

`class` soundata.datasets.starss2022.`Dataset`(*data_home=None*)

The STARSS 2022 dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

cite()

Print the reference

clip_ids

Return clip ids

Returns
list – A list of clip ids

clipgroup_ids

Return clip ids

Returns
list – A list of clip ids

property default_path

Get the default path for the dataset

Returns
str – Local path to the dataset

download(*partial_download=None*, *force_overwrite=False*, *cleanup=False*)

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- **ValueError** – if invalid keys are passed to partial_download
- **IOError** – if a downloaded file's checksum is different from expected

explore_dataset(*clip_id=None*)

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters
clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

license()

Print the license

`load_audio`(**args*, ***kwargs*)

Load a STARSS 2022 audio file. :Parameters: * **fhandle** (*str or file-like*) – path or file-like object pointing to an audio file

- **sr** (*int or None*) – sample rate for loaded audio, 24000 Hz by default.
- **If different from file's sample rate it will be resampled on load.**
- **Use None to load the file using its original sample rate** (*24000*)

Returns

- np.ndarray - the audio signal
- float - The sample rate of the audio file

`load_clipgroups`()

Load all clipgroups in the dataset

Returns
dict – {*clipgroup_id*: clipgroup data}

Raises
`NotImplementedError` – If the dataset does not support Clipgroups

`load_clips`()

Load all clips in the dataset

Returns
dict – {*clip_id*: clip data}

Raises
`NotImplementedError` – If the dataset does not support Clips

`validate`(*verbose=True*)

Validate if the stored dataset is a valid version

Parameters
verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

`soundata.datasets.starss2022.load_audio`(*fhandle: BinaryIO*, *sr=24000*) → Tuple[numpy.ndarray, float]

Load a STARSS 2022 audio file. :Parameters: * **fhandle** (*str or file-like*) – path or file-like object pointing to an audio file

- **sr** (*int or None*) – sample rate for loaded audio, 24000 Hz by default.
- **If different from file's sample rate it will be resampled on load.**
- **Use None to load the file using its original sample rate** (*24000*)

Returns

- np.ndarray - the audio signal
- float - The sample rate of the audio file

soundata.datasets.starss2022.**load_spatialevents**(*fhandle: TextIO*, *dt=0.1*) → *SpatialEvents*

Load a STARSS 2022 annotation file :Parameters: * **fhandle** (*str or file-like*) – File-like object or path to the sound events annotation file

- **dt** (*float*) – time step

Raises
IOError – if fhandle doesn't exist

Returns
SpatialEvents – sound spatial events annotation data

4.5.16 TAU NIGENS SSE 2020

TAU NIGENS SSE 2020 Dataset Loader

Dataset Info

TAU NIGENS Spatial Sound Events: scene recordings with (moving) sound events of distinct categories

Created By:

Archontis Politis, Sharath Adavanne, Tuomas Virtanen.
Audio Research Group, Tampere University (Finland).

Version 1.2.0 *Description:*

Spatial sound-scene recordings, consisting of sound events of distinct categories in a variety of acoustical spaces, and from multiple source directions and distances. The spatialization of all sound events is based on filtering through real spatial room impulse responses (RIRs) of diverse acoustic environments. The sound events are spatialized as either stationary sound sources, or moving sound sources, in which case time-variant RIRs are used. Each scene recording is delivered in microphone array (MIC) and first-order Ambisonics (FOA) format.

Audio Files Included:

- 600 one-minute long sound scene recordings (development dataset).
- 200 one-minute long sound scene recordings (evaluation dataset).
- Sampling rate is 24 kHz (16-bit signed integer PCM).
- About 700 sound event samples spread over 14 classes (see here for more details).
- 8 provided cross-validation folds of 100 recordings each, with unique sound event samples and rooms in each of them.
- Two 4-channel 3-dimensional recording formats: first-order Ambisonics (FOA) and tetrahedral microphone array.
- Realistic spatialization and reverberation through RIRs collected in 15 different enclosures.
- From about 1500 to 3500 possible RIR positions across the different rooms.
- Both static reverberant and moving reverberant sound events.
- Up to two overlapping sound events allowed, temporally and spatially.
- Realistic spatial ambient noise collected from each room is added to the spatialized sound events, at varying signal-to-noise ratios (SNR) ranging from noiseless (30dB) to noisy (6dB).

Annotations Included:

- Each recording in the development set has labels of events and Directions of arrival in a plain csv file with the same filename.
- Each row in the csv file has a frame number, active class index, clip number index, azimuth, and elevation.
- Frame, class, and clip enumeration begins at 0.
- Frames correspond to a temporal resolution of 100msec.
- Azimuth and elevation angles are given in degrees, rounded to the closest integer value, with azimuth and elevation being zero at the front, azimuth $\phi \in [-180°, 180°]$, and elevation $\theta \in [-90°, 90°]$. Note that the azimuth angle is increasing counter-clockwise ($\phi = 90°$ at the left).
- The event number index is a unique integer for each event in the recording, enumerating them in the order of appearance. This event identifiers are useful to disentangle directions of co-occuring events through time in the metadata file.
- Overlapping sound events are indicated with duplicate frame numbers, and can belong to a different or the same class.

Please Acknowledge TAU-NIGENS SSE 2020 in Academic Research:

- **If you use this dataset please cite the report on its creation, and the corresponding DCASE2020 task setup:**
  - Politis., Archontis, Adavanne, Sharath, & Virtanen, Tuomas (2020). A Dataset of Reverberant Spatial Sound Scenes with Moving Sources for Sound Event Localization and Detection. In Proceedings of the Detection and Classification of Acoustic Scenes and Events 2020 Workshop (DCASE2020), Tokyo, Japan.

License:

- Creative Commons Attribution Non Commercial 4.0 International

---

class soundata.datasets.tau2020sse_nigens.**Clip**(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

TAU NIGENS SSE 2020 Clip class :Parameters: **clip_id** (*str*) – id of the clip

Variables

- **audio_path** (*str*) – path to the audio file
- **tags** (*soundata.annotation.Tags*) – tag
- **clip_id** (*str*) – clip id
- **spatial_events** (SpatialEvents) – sound events with time step, elevation, azimuth, distance, label, clip_number and confidence.

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio :returns:

- np.ndarray - audio signal
- float - sample rate

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters

key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

`spatial_events`

The clip's event annotations

Returns

- **SpatialEvents with attributes**
  - **intervals (list): list of size n np.ndarrays of shape (m, 2), with intervals** (as floats) in TIME_UNITS in the form [start_time, end_time]
  - intervals_unit (str): intervals unit, one of TIME_UNITS
  - time_step (int, float, or None): the time-step between events
  - **elevations (list): list of size n with np.ndarrays with dtype int,** indicating the elevation of the sound event per time_step.
  - elevations_unit (str): elevations unit, one of ELEVATIONS_UNITS
  - **azimuths (list): list of size n with np.ndarrays with dtype int,** indicating the azimuth of the sound event per time_step if moving
  - azimuths_unit (str): azimuths unit, one of AZIMUTHS_UNITS
  - **distances (list): list of size n with np.ndarrays with dtype int,** indicating the distance of the sound event per time_step if moving
  - distances_unit (str): distances unit, one of DISTANCES_UNITS
  - labels (list): list of event labels (as strings)
  - labels_unit (str): labels unit, one of LABELS_UNITS
  - clip_number_indices (list): list of clip number indices (as strings)
  - confidence (np.ndarray or None): array of confidence values

`to_jams()`

Get the clip's data in jams format :returns: *jams.JAMS* – the clip's data in jams format

`class soundata.datasets.tau2020sse_nigens.Dataset(`*data_home=None*`)`

The TAU NIGENS SSE 2020 dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

`choice_clip()`

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

`choice_clipgroup()`

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

`cite()`

Print the reference

`clip_ids`

Return clip ids

Returns
list – A list of clip ids

`clipgroup_ids`

Return clip ids

Returns
list – A list of clip ids

`property default_path`

Get the default path for the dataset

Returns
str – Local path to the dataset

`download(partial_download=None, force_overwrite=False, cleanup=False)`

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- `ValueError` – if invalid keys are passed to partial_download
- `IOError` – if a downloaded file's checksum is different from expected

`explore_dataset(clip_id=None)`

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters
clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

`license()`

Print the license

`load_audio(*args, **kwargs)`

Load a TAU NIGENS SSE 2020 audio file. :Parameters: * **fhandle** (*str or file-like*) – path or file-like object pointing to an audio file

- **sr** (*int or None*) – sample rate for loaded audio, 24000 Hz by default.
- **If different from file's sample rate it will be resampled on load.**

- **Use None to load the file using its original sample rate** (*24000*)

Returns

- np.ndarray - the audio signal
- float - The sample rate of the audio file

`load_clipgroups()`

Load all clipgroups in the dataset

Returns
dict – {*clipgroup_id*: clipgroup data}

Raises
`NotImplementedError` – If the dataset does not support Clipgroups

`load_clips()`

Load all clips in the dataset

Returns
dict – {*clip_id*: clip data}

Raises
`NotImplementedError` – If the dataset does not support Clips

`validate(verbose=True)`

Validate if the stored dataset is a valid version

Parameters
verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

`soundata.datasets.tau2020sse_nigens.load_audio(fhandle: BinaryIO, sr=24000)` → Tuple[numpy.ndarray, float]

Load a TAU NIGENS SSE 2020 audio file. :Parameters: * **fhandle** (*str or file-like*) – path or file-like object pointing to an audio file

- **sr** (*int or None*) – sample rate for loaded audio, 24000 Hz by default.
- **If different from file's sample rate it will be resampled on load.**
- **Use None to load the file using its original sample rate** (*24000*)

Returns

- np.ndarray - the audio signal
- float - The sample rate of the audio file

`soundata.datasets.tau2020sse_nigens.load_spatialevents(fhandle: TextIO, dt=0.1)` → *SpatialEvents*

Load an TAU NIGENS SSE 2020 annotation file :Parameters: * **fhandle** (*str or file-like*) – File-like object or path to
the sound events annotation file

- **dt** (*float*) – time step

Raises
`IOError` – if txt_path doesn't exist

Returns
SpatialEvents – sound spatial events annotation data

4.5.17 TAU NIGENS SSE 2021

TAU NIGENS SSE 2021 Dataset Loader

Dataset Info

TAU NIGENS Spatial Sound Events: scene recordings with (moving) sound events of distinct categories

Created By:

Archontis Politis, Sharath Adavanne, Tuomas Virtanen.
Audio Research Group, Tampere University (Finland).

Version 1.2.0

Description:

Spatial sound-scene recordings, consisting of sound events of distinct categories in a variety of acoustical spaces, and from multiple source directions and distances. The spatialization of all sound events is based on filtering through real spatial room impulse responses (RIRs) of diverse acoustic environments. The sound events are spatialized as either stationary sound sources, or moving sound sources, in which case time-variant RIRs are used.

Each scene recording is delivered in microphone array (MIC) and first-order Ambisonics (FOA) format.

Audio Files Included:

- 600 one-minute-long sound scene recordings with annotations (development dataset).
- 200 one-minute-long sound scene recordings without annotations (evaluation dataset).
- Sampling rate is 24 kHz (16-bit signed integer PCM).
- About 500 sound event samples distirbuted over 12 target classes.
- About 400 sound event samples used as interference events.
- 1st order HOA or tetrahedral microphone array formats.
- Realistic spatialization and reverberation through multichannel RIRs collected in 13 different enclosures.
- From 1184 to 6480 possible RIR positions across the different rooms.
- Both static reverberant and moving reverberant sound events.
- Three possible angular speeds for moving sources of approximately 10, 20, or 40deg/sec.
- Up to three overlapping sound events possible, temporally and spatially.
- Simultaneous directional interfering sound events with their own temporal activities, static or moving.
- Realistic spatial ambient noise collected from each room is added to the spatialized sound events, at varying signal-to-noise ratios (SNR) ranging from noiseless (30dB) to noisy (6dB) conditions.

Annotations Included:

- Each recording in the development set has labels of events and DoAs in a plain csv file with the same filename.
- Each row in the csv file has a frame number, active class index, event number index, azimuth, and elevation.

- Frame, class, and clip enumeration begins at 0.
- Frames correspond to a temporal resolution of 100msec.
- Azimuth and elevation angles are given in degrees, rounded to the closest integer value, with azimuth and elevation being zero at the front, azimuth $\phi \in [-180°, 180°]$, and elevation $\theta \in [-90°, 90°]$. Note that the azimuth angle is increasing counter-clockwise ($\phi = 90°$ at the left).
- The event number index is a unique integer for each event in the recording, enumerating them in the order of appearance. This event identifiers are useful to disentangle directions of co-occuring events through time in the metadata file. The interferers are considered unknown and no activity or direction labels of them are provided with the training datasets.
- Overlapping sound events are indicated with duplicate frame numbers, and can belong to a different or the same class.

Organization

- The development dataset is split in training, validation, and test sets.
- The training set consists of 400 recordings.
- The validation set consists of 100 recordings.
- The test set consists of 100 recordings.
- The evalutation dataset constists of 200 recordings.

Please Acknowledge TAU-NIGENS SSE 2021 in Academic Research:

- If you use this dataset please cite the report on its creation, and the corresponding DCASE2021 task setup:
  - Archontis Politis, Sharath Adavanne, Daniel Krause, Antoine Deleforge, Prerak Srivastava, and Tuomas Virtanen. A dataset of dynamic reverberant sound scenes with directional interferers for sound event localization and detection. arXiv preprint arXiv:2106.06999, 2021. URL: https://arxiv.org/abs/2106.06999, arXiv:2106.06999.

License:

- Creative Commons Attribution Non Commercial 4.0 International

---

class soundata.datasets.tau2021sse_nigens.**Clip**(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

TAU NIGENS SSE 2021 Clip class :Parameters: **clip_id** (*str*) – id of the clip

Variables

- **audio_path** (*str*) – path to the audio file
- **tags** (*soundata.annotation.Tags*) – tag
- **clip_id** (*str*) – clip id
- **spatial_events** (SpatialEvents) – sound events with time step, elevation, azimuth, distance, label, clip_number and confidence.

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio :returns:

- np.ndarray - audio signal
- float - sample rate

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

`spatial_events`

The clip's event annotations :returns:

- **SpatialEvents with attributes**
  - **intervals (list): list of size n np.ndarrays of shape (m, 2), with intervals** (as floats) in TIME_UNITS in the form [start_time, end_time]
  - intervals_unit (str): intervals unit, one of TIME_UNITS
  - time_step (int, float, or None): the time-step between events
  - **elevations (list): list of size n with np.ndarrays with dtype int,** indicating the elevation of the sound event per time_step.
  - elevations_unit (str): elevations unit, one of ELEVATIONS_UNITS
  - **azimuths (list): list of size n with np.ndarrays with dtype int,** indicating the azimuth of the sound event per time_step if moving
  - azimuths_unit (str): azimuths unit, one of AZIMUTHS_UNITS
  - **distances (list): list of size n with np.ndarrays with dtype int,** indicating the distance of the sound event per time_step if moving
  - distances_unit (str): distances unit, one of DISTANCES_UNITS
  - labels (list): list of event labels (as strings)
  - labels_unit (str): labels unit, one of LABELS_UNITS
  - clip_number_indices (list): list of clip number indices (as strings)
  - confidence (np.ndarray or None): array of confidence values

`to_jams()`

Get the clip's data in jams format :returns: *jams.JAMS* – the clip's data in jams format

`class soundata.datasets.tau2021sse_nigens.Dataset(data_home=None)`

The TAU NIGENS SSE 2021 dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

`choice_clip()`

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

`choice_clipgroup()`

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

`cite()`

Print the reference

`clip_ids`

Return clip ids

Returns
list – A list of clip ids

`clipgroup_ids`

Return clip ids

Returns
list – A list of clip ids

`property default_path`

Get the default path for the dataset

Returns
str – Local path to the dataset

`download(partial_download=None, force_overwrite=False, cleanup=False)`

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- `ValueError` – if invalid keys are passed to partial_download
- `IOError` – if a downloaded file's checksum is different from expected

`explore_dataset(clip_id=None)`

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters
clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

`license()`

Print the license

`load_audio(*args, **kwargs)`

Load a TAU NIGENS SSE 2021 audio file. :Parameters: * **fhandle** (*str or file-like*) – path or file-like object pointing to an audio file

- **sr** (*int or None*) – sample rate for loaded audio, 24000 Hz by default.
- **If different from file's sample rate it will be resampled on load.**
- **Use None to load the file using its original sample rate** (*24000*)

Returns

- np.ndarray - the audio signal
- float - The sample rate of the audio file

`load_clipgroups()`

Load all clipgroups in the dataset

Returns
dict – {*clipgroup_id*: clipgroup data}

Raises
`NotImplementedError` – If the dataset does not support Clipgroups

`load_clips()`

Load all clips in the dataset

Returns
dict – {*clip_id*: clip data}

Raises
`NotImplementedError` – If the dataset does not support Clips

`validate(verbose=True)`

Validate if the stored dataset is a valid version

Parameters
verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

`soundata.datasets.tau2021sse_nigens.load_audio(fhandle: BinaryIO, sr=24000)` → Tuple[numpy.ndarray, float]

Load a TAU NIGENS SSE 2021 audio file. :Parameters: * **fhandle** (*str or file-like*) – path or file-like object pointing to an audio file

- **sr** (*int or None*) – sample rate for loaded audio, 24000 Hz by default.
- **If different from file's sample rate it will be resampled on load.**
- **Use None to load the file using its original sample rate** (*24000*)

Returns

- np.ndarray - the audio signal
- float - The sample rate of the audio file

`soundata.datasets.tau2021sse_nigens.load_spatialevents(fhandle: TextIO, dt=0.1)` → *SpatialEvents*

Load an TAU NIGENS SSE 2021 annotation file :Parameters: * **fhandle** (*str or file-like*) – File-like object or path to
the sound events annotation file

- **dt** (*float*) – time step

Raises
IOError – if txt_path doesn't exist

Returns
SpatialEvents – sound spatial events annotation data

4.5.18 TAU Spatial Sound Events 2019

TAU SSE 2019 Dataset Loader

Dataset Info

TAU SSE 2019

Created By:

Sharath Adavanne; Archontis Politis; Tuomas Virtanen
Audio Research Group, Tampere University.

Version 2

Description:
Recordings with stationary point sources (events) from multiple sound classes. Up to two temporally overlaping sound events. Recordings of identical scenes are available in both 1st-order ambisonics and corresponding four-channel tetrahedral microphone format. Recordings can happen in one of five different rooms. The sound classes are the 11 different ones from the DCASE 2016 challenge task 2. Each class has 20 different examples.

Audio Files Included:

- 500 one-minute-long recordings (400 development and 100 evaluation; 48kHz sampling rate and 16-bit precision).

Annotations Included:

- **sound event category with:**
  - start time
  - end time
  - elevation
  - azimuth
  - distance
- **Moreover, the clip id indicates:**
  - data split number (4 in development and 1 in evaluation)
  - room number (IR: impulse response)
  - whether there are temporally-overlapping events

Please Acknowledge TAU SSE 2019 in Academic Research:

- **If you use this dataset please cite its original publication:**
  - Sharath Adavanne, Archontis Politis, and Tuomas Virtanen. A multi-room reverberant dataset for sound event localization and uetection. In Submitted to Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE2019). 2019. URL: https://arxiv.org/abs/1905.08546.

License:

`class soundata.datasets.tau2019sse.Clip(`*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*`)`

TAU SSE 2019 Clip class

Parameters

clip_id (*str*) – id of the clip

Variables

- **spatial_events** (SpatialEvents) – sound events with start time, end time, elevation, azimuth, distance, label and confidence.
- **audio_path** (*str*) – path to the audio file
- **set** (*str*) – subset the clip belongs to (development or evaluation)
- **format** (*str*) – whether the clip is in foa or mic format
- **clip_id** (*str*) – clip id

`property audio: Optional[Tuple[numpy.ndarray, float]]`

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

`get_path(`*key*`)`

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters

key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

`spatial_events`

The clip's spatial events

Returns

- **SpatialEvents class with attributes**
  - **intervals (np.ndarray): (n x 2) array of intervals** (as floats) in seconds in the form [start_time, end_time] with positive time stamps and end_time >= start_time.
  - elevations (np.ndarray): (n,) array of elevations
  - azimuths (np.ndarray): (n,) array of azimuths
  - distances (np.ndarray): (n,) array of distances
  - labels (list): list of event labels (as strings)
  - confidence (np.ndarray or None): array of confidence values, float in [0, 1]
  - labels_unit (str): labels unit, one of LABELS_UNITS
  - intervals_unit (str): intervals unit, one of TIME_UNITS

`to_jams()`

Get the clip's data in jams format

Returns
jams.JAMS – the clip's data in jams format

`class soundata.datasets.tau2019sse.Dataset(data_home=None)`

The TAU SSE 2019 dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

`choice_clip()`

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

`choice_clipgroup()`

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

`cite()`

Print the reference

`clip_ids`

Return clip ids

Returns
list – A list of clip ids

`clipgroup_ids`

Return clip ids

Returns
list – A list of clip ids

`property default_path`

Get the default path for the dataset

Returns
str – Local path to the dataset

`download(partial_download=None, force_overwrite=False, cleanup=False)`

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- `ValueError` – if invalid keys are passed to partial_download
- `IOError` – if a downloaded file's checksum is different from expected

`explore_dataset(clip_id=None)`

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters
clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

`license()`

Print the license

`load_audio(*args, **kwargs)`

Load a TAU SSE 2019 audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate of 48000 without resampling.

Returns

- np.ndarray - the multichannel audio signal

- float - The sample rate of the audio file

load_clipgroups()

Load all clipgroups in the dataset

Returns
dict – {*clipgroup_id*: clipgroup data}

Raises
NotImplementedError – If the dataset does not support Clipgroups

load_clips()

Load all clips in the dataset

Returns
dict – {*clip_id*: clip data}

Raises
NotImplementedError – If the dataset does not support Clips

validate(*verbose=True*)

Validate if the stored dataset is a valid version

Parameters
verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

class soundata.datasets.tau2019sse.**TAU2019_SpatialEvents**(*intervals*, *intervals_unit*, *elevations*, *elevations_unit*, *azimuths*, *azimuths_unit*, *distances*, *distances_unit*, *labels*, *labels_unit*, *confidence=None*)

TAU SSE 2019 Spatial Events

Variables

- **intervals** (*np.ndarray*) – (n x 2) array of intervals (as floats) in seconds in the form [start_time, end_time] with positive time stamps and end_time >= start_time.
- **elevations** (*np.ndarray*) – (n,) array of elevations
- **azimuths** (*np.ndarray*) – (n,) array of azimuths
- **distances** (*np.ndarray*) – (n,) array of distances
- **labels** (*list*) – list of event labels (as strings)
- **confidence** (*np.ndarray or None*) – array of confidence values, float in [0, 1]
- **labels_unit** (*str*) – labels unit, one of LABELS_UNITS
- **intervals_unit** (*str*) – intervals unit, one of TIME_UNITS

soundata.datasets.tau2019sse.**load_audio**(*fhandle: BinaryIO*, *sr=None*) → Tuple[numpy.ndarray, float]

Load a TAU SSE 2019 audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate of 48000 without resampling.

Returns

- np.ndarray - the multichannel audio signal
- float - The sample rate of the audio file

soundata.datasets.tau2019sse.**load_spatialevents**(*fhandle: TextIO*) → *TAU2019_SpatialEvents*

Load an TAU SSE 2019 annotation file :Parameters: **fhandle** (*str or file-like*) – File-like object or path to the sound events annotation file

Raises

IOError – if csv_path doesn't exist

Returns

Events – sound events annotation data

soundata.datasets.tau2019sse.**validate_locations**(*locations*)

Validate if TAU SSE 2019 locations are well-formed.

If locations is None, validation passes automatically

Parameters

locations (*np.ndarray*) – (n x 3) array

Raises

ValueError – if locations have an invalid shape or have cartesian coordinate values outside the expected ranges.

4.5.19 TAU Urban Acoustic Scenes 2019

TAU Urban Acoustic Scenes 2019 Loader

Dataset Info

TAU Urban Acoustic Scenes 2019, Development, Leaderboard and Evaluation datasets

Audio Research Group, Tampere University of Technology

Authors

- Toni Heittola
- Annamaria Mesaros
- Tuomas Virtanen

Recording and annotation

- Henri Laakso
- Ronal Bejarano Rodriguez
- Toni Heittola

Links

- Development dataset
- Leaderboard dataset
- Evaluation dataset

Dataset

TAU Urban Acoustic Scenes 2019 dataset consists of 10-seconds audio segments from 10 acoustic scenes:

- Airport - *airport*
- Indoor shopping mall - *shopping_mall*
- Metro station - *metro_station*
- Pedestrian street - *street_pedestrian*
- Public square - *public_square*
- Street with medium level of traffic - *street_traffic*
- Travelling by a tram - *tram*
- Travelling by a bus - *bus*
- Travelling by an underground metro - *metro*
- Urban park - *park*

A detailed description of the data recording and annotation procedure is available in:

```
Annamaria Mesaros, Toni Heittola, and Tuomas Virtanen.
"A multi-device dataset for urban acoustic scene classification",
In Proceedings of the Detection and Classification of Acoustic
Scenes and Events 2018 Workshop (DCASE2018), Surrey, UK, 2018.
```

Development dataset

Each acoustic scene has 1440 segments (240 minutes of audio). The dataset contains in total 40 hours of audio.

Evaluation dataset

The dataset contains in total 7200 segments (20 hours of audio).

Leaderboard dataset

The dataset contains in total 1200 segments (200 minutes of audio).

The dataset was collected by Tampere University of Technology between 05/2018 -11/2018. The data collection has received funding from the European Research Council under the ERC Grant Agreement 637422 EVERYSOUND.

Preparation of the dataset

The dataset was recorded in 12 large European cities: Amsterdam, Barcelona, Helsinki, Lisbon, London, Lyon, Madrid, Milan, Prague, Paris, Stockholm, and Vienna. For all acoustic scenes, audio was captured in multiple locations: different streets, different parks, different shopping malls. In each location, multiple 2-3 minute long audio recordings were captured in a few slightly different positions (2-4) within the selected location. Collected audio material was cut into segments of 10 seconds length.

The equipment used for recording consists of a binaural Soundman OKM II Klassik/studio A3 electret in-ear microphone and a Zoom F8 audio recorder using 48 kHz sampling rate and 24 bit resolution. During the recording, the microphones were worn by the recording person in the ears, and head movement was kept to minimum.

Post-processing of the recorded audio involves aspects related to privacy of recorded individuals, and possible errors in the recording process. The material was screened for content, and segments containing close microphone conversation were eliminated. Some interferences from mobile phones are audible, but are considered part of real-world recording process.

A subset of the dataset has been previously published as TUT Urban Acoustic Scenes 2018 Development dataset. Audio segment filenames are retained for the segments coming from this dataset.

Dataset statistics

The **development dataset** contains audio material from 10 cities, whereas the evaluation dataset (TAU Urban Acoustic Scenes 2019 evaluation) contains data from all 12 cities. The dataset is perfectly balanced at acoustic scene level, with very slight differences in the number of segments from each city.

Audio segments (Development dataset)

Scene class	Segments	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	1440	128	149	144	145	144	144	156	144	158	128
Bus	1440	144	144	144	144	144	144	144	144	144	144
Metro	1440	141	144	144	146	144	144	144	144	145	144
Metro station	1440	144	144	144	144	144	144	144	144	144	144
Park	1440	144	144	144	144	144	144	144	144	144	144
Public square	1440	144	144	144	144	144	144	144	144	144	144
Shopping mall	1440	144	144	144	144	144	144	144	144	144	144
Street, pedestrian	1440	145	145	144	145	144	144	144	144	145	140
Street, traffic	1440	144	144	144	144	144	144	144	144	144	144
Tram	1440	143	145	144	144	144	144	144	144	144	144
Total	**14400**	**1421**	**1447**	**1440**	**1444**	**1440**	**1440**	**1452**	**1440**	**1456**	**1420**

Audio segments (Recording locations)

Scene class	Locations	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	40	4	3	4	3	4	4	4	6	5	3
Bus	71	4	4	11	7	7	7	11	10	6	4
Metro	67	3	5	11	4	9	8	9	10	4	4
Metro station	57	5	6	4	12	5	4	9	4	4	4
Park	41	4	4	4	4	4	4	4	4	5	4
Public_square	43	4	4	4	4	5	4	4	6	4	4
Shopping mall	36	4	4	4	2	3	3	4	4	4	4
Street, pedestrian	46	7	4	4	4	4	5	5	5	4	4
Street, traffic	43	4	4	4	5	4	6	4	4	4	4
Tram	70	4	4	6	9	7	11	9	11	5	4
Total	**514**	**43**	**42**	**56**	**54**	**52**	**56**	**63**	**65**	**45**	**39**

Usage

The partitioning of the data was done based on the location of the original recordings. All segments recorded at the same location were included into a single subset - either **development dataset** or **evaluation dataset**. For each acoustic scene, 1440 segments were included in the development dataset provided here. Evaluation dataset is provided separately.

Training / test setup

A suggested training/test partitioning of the development set is provided in order to make results reported with this

dataset uniform. The partitioning is done such that the segments recorded at the same location are included into the same subset - either training or testing. The partitioning is done aiming for a 70/30 ratio between the number of segments in training and test subsets while taking into account recording locations, and selecting the closest available option. Audio segments coming from nine cities are used for training and all ten cities are used for testing (Milan is used only for testing). Since the dataset includes balanced amount of material from ten cities, this partitioning will leave a small subset of data from Milan unused in the training / test setup. This material can be used when using full dataset to train the system and testing it with evaluation dataset.

The setup is provided with the dataset in the directory *evaluation_setup*.

Statistics

Scene class	Train / Segments	Train / Locations	Test / Segments	Test / Locations	Unused / Segments	Unused / Locations
Airport	911	25	421	12	108	3
Bus	928	46	415	20	97	5
Metro	902	41	433	20	105	6
Metro station	897	37	435	17	108	3
Park	946	27	386	11	108	3
Public square	945	28	387	12	108	3
Shopping mall	896	24	441	10	103	2
Street, pedestrian	924	29	429	14	87	3
Street, traffic	942	27	402	12	96	4
Tram	894	41	436	21	110	8
Total	**9185**	**325**	**4185**	**149**	**1030**	**40**

License

License permits free academic usage. Any commercial use is strictly prohibited. For commercial use, contact dataset authors.

> Copyright (c) 2019 Tampere University and its licensors All rights reserved. Permission is hereby granted, without written agreement and without license or royalty fees, to use and copy the TAU Urban Acoustic Scenes 2019 (“Work”) described in this document and composed of audio and metadata. This grant is only for experimental and non-commercial purposes, provided that the copyright notice in its entirety appear in all copies of this Work, and the original source of this Work, (Audio Research Group at Tampere University of Technology), is acknowledged in any publication that reports research using this Work. Any commercial use of the Work or any part thereof is strictly prohibited. Commercial use include, but is not limited to: - selling or reproducing the Work - selling or distributing the results or content achieved by use of the Work - providing services by using the Work.
>
> IN NO EVENT SHALL TAMPERE UNIVERSITY OR ITS LICENSORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS WORK AND ITS DOCUMENTATION, EVEN IF TAMPERE UNIVERSITY OR ITS LICENSORS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
>
> TAMPERE UNIVERSITY AND ALL ITS LICENSORS SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE WORK PROVIDED HEREUNDER IS ON AN “AS IS” BASIS, AND THE TAMPERE UNIVERSITY HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

class soundata.datasets.tau2019uas.Clip(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

TAU Urban Acoustic Scenes 2019 Clip class

Parameters

clip_id (*str*) – id of the clip

Variables

- **audio** (*np.ndarray, float*) – path to the audio file
- **audio_path** (*str*) – path to the audio file
- **city** (*str*) – city were the audio signal was recorded
- **clip_id** (*str*) – clip id
- **identifier** (*str*) – identifier present in the metadata
- **split** (*str*) – subset the clip belongs to (for experiments): development (fold1, fold2, fold3, fold4), leaderboard or evaluation
- **tags** (soundata.annotations.Tags) – tag (scene label) of the clip + confidence.

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

property city

The clip's city.

Returns

- str - city were the audio signal was recorded

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters

key (*string*) – Index key of the audio or annotation type

Returns

str or None – joined path string or None

property identifier

The clip's identifier.

Returns

- str - identifier present in the metadata

property split

The clip's split.

Returns

** str - subset the clip belongs to (for experiments)* – development (fold1, fold2, fold3, fold4), leaderboard or evaluation

property tags

The clip's tags.

Returns

- annotations.Tags - tag (scene label) of the clip + confidence.

to_jams()

Get the clip's data in jams format

Returns

jams.JAMS – the clip's data in jams format

class soundata.datasets.tau2019uas.Dataset(*data_home=None*)

The TAU Urban Acoustic Scenes 2019 dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns

Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns

Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

cite()

Print the reference

clip_ids

Return clip ids

Returns

list – A list of clip ids

clipgroup_ids

Return clip ids

Returns

list – A list of clip ids

property default_path

Get the default path for the dataset

Returns

str – Local path to the dataset

`download`(*partial_download=None*, *force_overwrite=False*, *cleanup=False*)

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- `ValueError` – if invalid keys are passed to partial_download
- `IOError` – if a downloaded file's checksum is different from expected

`explore_dataset`(*clip_id=None*)

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters

clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

`license`()

Print the license

`load_audio`(**args*, ***kwargs*)

Load a TAU Urban Acoustic Scenes 2019 audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate of 44100 without resampling.

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

`load_clipgroups`()

Load all clipgroups in the dataset

Returns

dict – {*clipgroup_id*: clipgroup data}

Raises

`NotImplementedError` – If the dataset does not support Clipgroups

`load_clips`()

Load all clips in the dataset

Returns

dict – {*clip_id*: clip data}

Raises

`NotImplementedError` – If the dataset does not support Clips

validate(*verbose=True*)

Validate if the stored dataset is a valid version

Parameters

verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

soundata.datasets.tau2019uas.**load_audio**(*fhandle: BinaryIO*, *sr=None*) → Tuple[numpy.ndarray, float]

Load a TAU Urban Acoustic Scenes 2019 audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate of 44100 without resampling.

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

4.5.20 TAU Urban Acoustic Scenes 2020 Mobile

TAU Urban Acoustic Scenes 2020 Mobile Loader

Dataset Info

TAU Urban Acoustic Scenes 2020 Mobile, Development and Evaluation datasets

Audio Research Group, Tampere University of Technology

Authors

- Toni Heittola
- Annamaria Mesaros
- Tuomas Virtanen

Recording and annotation

- Henri Laakso
- Ronal Bejarano Rodriguez
- Toni Heittola

Links

- Development dataset
- Leaderboard dataset
- Evaluation dataset

Dataset

TAU Urban Acoustic Scenes 2020 Mobile development dataset consists of 10-seconds audio segments from 10 acoustic scenes:

- Airport - *airport*
- Indoor shopping mall - *shopping_mall*
- Metro station - *metro_station*
- Pedestrian street - *street_pedestrian*
- Public square - *public_square*
- Street with medium level of traffic - *street_traffic*
- Travelling by a tram - *tram*
- Travelling by a bus - *bus*
- Travelling by an underground metro - *metro*
- Urban park - *park*

A detailed description of the data recording and annotation procedure is available in:

```
Annamaria Mesaros, Toni Heittola, and Tuomas Virtanen.
"Acoustic scene classification in DCASE 2020 Challenge:
generalization across devices and low complexity solutions",
In Proceedings of the Detection and Classification of Acoustic
Scenes and Events 2020 Workshop (DCASE2020), Tokyo, Japan, 2020.
```

Recordings were made with three devices (A, B and C) that captured audio simultaneously and 6 simulated devices (S1-S6). Each acoustic scene has 1440 segments (240 minutes of audio) recorded with device A (main device) and 108 segments of parallel audio (18 minutes) each recorded with devices B,C, and S1-S6.

Development dataset

The dataset contains in total 64 hours of audio.

Evaluation dataset

The dataset contains in total 33 hours of audio.

The dataset was collected by Tampere University of Technology between 05/2018 -11/2018. The data collection has received funding from the European Research Council under the ERC Grant Agreement 637422 EVERYSOUND.

Preparation of the dataset

The dataset was recorded in 12 large European cities: Amsterdam, Barcelona, Helsinki, Lisbon, London, Lyon, Madrid, Milan, Prague, Paris, Stockholm, and Vienna. For all acoustic scenes, audio was captured in multiple locations: different streets, different parks, different shopping malls. In each location, multiple 2-3 minute long audio recordings were captured in a few slightly different positions (2-4) within the selected location. Collected audio material was cut into segments of 10 seconds length.

The main recording device (referred to as device A) consists of a binaural Soundman OKM IIKlassik/studio A3 electret in-ear microphone and a Zoom F8 audio recorder using 48 kHz sampling rate and 24 bit resolution. During the recording, the microphones were worn by the recording person in the ears, and head movement was kept to minimum.

Devices B and C are commonly available customer devices (e.g. smartphones, cameras) and were handled in typical ways (e.g. hand held). The audio recordings from these devices are of different quality than device A. All simultaneous recordings are time synchronized.

Post-processing of the recorded audio involves aspects related to privacy of recorded individuals, and possible errors in the recording process. The material was screened for content, and segments containing close microphone conversation were eliminated. Some interferences from mobile phones are audible, but are considered part of real-world recording process. In addition, data from device A was resampled and averaged into a single channel, to align with the properties of the data recorded with devices B and C.

Additionally, 11 mobile devices S1-S11 are simulated using the audio recorded with device A, impulse responses recorded with real devices, and additional dynamic range compression, in order to simulate realistic recordings. A recording from device A is processed through convolution with the selected Si impulse response, then processed with a selected set of parameters for dynamic range compression (device specific). The impulse responses are proprietary data and will not be published.

All provided audio data is single-channel, having a 44.1 KHz sampling rate, and 24 bit resolution.

A subset of the dataset has been previously published as TUT Urban Acoustic Scenes 2019 Development dataset. Audio segment filenames are retained for the segments coming from this dataset.

Dataset statistics

The development set contains data from 10 cities and 9 devices: 3 real devices (A, B, C) and 6 simulated devices (S1-S6). Data from devices B, C and S1-S6 consists of randomly selected segments from the simultaneous recordings, therefore all overlap with the data from device A, but not necessarily with each other. The total amount of audio in the development set is **64 hours**. The evaluation dataset (TAU Urban Acoustic Scenes 2020 Mobile evaluation) contains data from all 12 cities, and five new devices (not available in the development set): real device D and simulated devices S7-S11.

Device A

Audio segments

Scene class	Segments	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	1440	128	149	144	145	144	144	156	144	158	128
Bus	1440	144	144	144	144	144	144	144	144	144	144
Metro	1440	141	144	144	146	144	144	144	144	145	144
Metro station	1440	144	144	144	144	144	144	144	144	144	144
Park	1440	144	144	144	144	144	144	144	144	144	144
Public square	1440	144	144	144	144	144	144	144	144	144	144
Shopping mall	1440	144	144	144	144	144	144	144	144	144	144
Street, pedestrian	1440	145	145	144	145	144	144	144	144	145	140
Street, traffic	1440	144	144	144	144	144	144	144	144	144	144
Tram	1440	143	145	144	144	144	144	144	144	144	144
Total	**14400**	**1421**	**1447**	**1440**	**1444**	**1440**	**1440**	**1452**	**1440**	**1456**	**1420**

Recording locations

Scene class	Locations	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	40	4	3	4	3	4	4	4	6	5	3
Bus	71	4	4	11	7	7	7	11	10	6	4
Metro	67	3	5	11	4	9	8	9	10	4	4
Metro station	57	5	6	4	12	5	4	9	4	4	4
Park	41	4	4	4	4	4	4	4	4	5	4
Public_square	43	4	4	4	4	5	4	4	6	4	4
Shopping mall	36	4	4	4	2	3	3	4	4	4	4
Street, pedestrian	46	7	4	4	4	4	5	5	5	4	4
Street, traffic	43	4	4	4	5	4	6	4	4	4	4
Tram	70	4	4	6	9	7	11	9	11	5	4
Total	**514**	**43**	**42**	**56**	**54**	**52**	**56**	**63**	**65**	**45**	**39**

Device B

Audio segments

Scene class	Segments	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	107	11	12	12	11	11	10	10	10	10	10
Bus	108	12	12	12	11	11	10	10	10	10	10
Metro	108	12	12	12	11	11	10	10	10	10	10
Metro station	108	12	12	12	11	11	10	10	10	10	10
Park	108	12	12	12	11	11	10	10	10	10	10
Public square	107	11	12	12	11	11	10	10	10	10	10
Shopping mall	108	12	12	12	11	11	10	10	10	10	10
Street, pedestrian	108	12	12	12	11	11	10	10	10	10	10
Street, traffic	108	12	12	12	11	11	10	10	10	10	10
Tram	108	12	12	12	11	11	10	10	10	10	10
Total	**1078**	**118**	**120**	**120**	**110**	**110**	**100**	**100**	**100**	**100**	**100**

Recording locations

Scene class	Locations	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	36	3	3	4	3	3	4	4	5	4	3
Bus	57	4	4	9	7	6	5	8	7	3	4
Metro	47	3	4	6	4	6	5	6	6	4	4
Metro station	45	4	4	3	8	5	3	7	3	4	4
Park	37	4	4	4	4	4	3	4	3	3	4
Public_square	37	3	4	4	4	5	3	4	4	3	3
Shopping mall	34	4	4	4	2	3	3	4	4	3	3
Street, pedestrian	43	6	3	4	4	4	5	5	4	4	4
Street, traffic	41	4	4	4	4	4	6	4	4	4	4
Tram	50	4	4	5	6	5	5	7	7	3	4
Total	**427**	**39**	**37**	**47**	**46**	**44**	**42**	**53**	**47**	**35**	**37**

Device C

Audio segments

Scene class	Segments	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	107	11	12	12	11	11	10	10	10	10	10
Bus	108	12	12	12	11	11	10	10	10	10	10
Metro	108	12	12	12	11	11	10	10	10	10	10
Metro station	108	12	12	12	11	11	10	10	10	10	10
Park	108	12	12	12	11	11	10	10	10	10	10
Public square	108	12	12	12	11	11	10	10	10	10	10
Shopping mall	107	12	12	12	10	11	10	10	10	10	10
Street, pedestrian	108	12	12	12	11	11	10	10	10	10	10
Street, traffic	108	12	12	12	11	11	10	10	10	10	10
Tram	107	11	12	12	11	11	10	10	10	10	10
Total	**1077**	**118**	**120**	**120**	**109**	**110**	**100**	**100**	**100**	**100**	**100**

Recording locations

Scene class	Locations	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	38	4	3	4	3	3	4	4	5	5	3
Bus	50	4	4	7	6	5	4	7	7	3	3
Metro	54	3	3	6	4	9	6	7	8	4	4
Metro station	48	5	3	4	8	5	4	7	4	4	4
Park	39	4	4	4	4	4	4	4	4	3	4
Public_square	40	4	3	4	4	4	4	4	6	3	4
Shopping mall	35	4	4	4	2	3	3	4	4	3	4
Street, pedestrian	41	6	3	4	4	3	5	4	5	4	3
Street, traffic	40	4	3	4	4	4	6	4	4	4	3
Tram	51	4	4	5	6	4	8	6	7	3	4
Total	**436**	**42**	**34**	**46**	**45**	**44**	**48**	**51**	**54**	**36**	**36**

Device S1

Audio segments

Scene class	Segments	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	108	12	12	12	11	11	10	10	10	10	10
Bus	108	12	12	12	11	11	10	10	10	10	10
Metro	108	12	12	12	11	11	10	10	10	10	10
Metro station	108	12	12	12	11	11	10	10	10	10	10
Park	108	12	12	12	11	11	10	10	10	10	10
Public square	108	12	12	12	11	11	10	10	10	10	10
Shopping mall	108	12	12	12	11	11	10	10	10	10	10
Street, pedestrian	108	12	12	12	11	11	10	10	10	10	10
Street, traffic	108	12	12	12	11	11	10	10	10	10	10
Tram	108	12	12	12	11	11	10	10	10	10	10
Total	**1080**	**120**	**120**	**120**	**110**	**110**	**100**	**100**	**100**	**100**	**100**

Recording locations

Scene class	Locations	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	37	4	3	4	3	4	4	4	4	4	3
Bus	54	4	4	8	6	6	6	7	6	3	4
Metro	50	3	3	8	4	7	6	6	6	4	3
Metro station	48	5	4	4	9	5	4	5	4	4	4
Park	36	4	4	4	4	3	4	3	3	3	4
Public_square	37	4	4	4	4	4	4	3	3	3	4
Shopping mall	33	4	4	4	2	3	3	3	3	3	4
Street, pedestrian	40	6	3	4	4	3	5	2	5	4	4
Street, traffic	40	4	4	4	4	4	6	3	3	4	4
Tram	52	4	4	5	7	6	7	6	6	3	4
Total	**427**	**42**	**37**	**49**	**47**	**45**	**49**	**42**	**43**	**35**	**38**

Device S2

Audio segments

Scene class	Segments	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	108	12	12	12	11	11	10	10	10	10	10
Bus	108	12	12	12	11	11	10	10	10	10	10
Metro	108	12	12	12	11	11	10	10	10	10	10
Metro station	108	12	12	12	11	11	10	10	10	10	10
Park	108	12	12	12	11	11	10	10	10	10	10
Public square	108	12	12	12	11	11	10	10	10	10	10
Shopping mall	108	12	12	12	11	11	10	10	10	10	10
Street, pedestrian	108	12	12	12	11	11	10	10	10	10	10
Street, traffic	108	12	12	12	11	11	10	10	10	10	10
Tram	108	12	12	12	11	11	10	10	10	10	10
Total	**1080**	**120**	**120**	**120**	**110**	**110**	**100**	**100**	**100**	**100**	**100**

Recording locations

Scene class	Locations	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	36	3	3	4	3	4	4	4	4	4	3
Bus	58	4	4	9	6	6	7	9	6	3	4
Metro	55	3	3	10	4	8	8	5	7	4	3
Metro station	49	5	4	4	7	5	4	8	4	4	4
Park	38	4	4	4	4	4	4	4	4	2	4
Public_square	41	4	4	4	4	5	4	4	5	3	4
Shopping mall	34	4	4	3	2	3	3	4	4	3	4
Street, pedestrian	42	7	3	4	4	3	5	5	4	4	3
Street, traffic	42	4	4	4	5	4	6	4	4	4	3
Tram	51	4	4	5	7	6	7	7	4	3	4
Total	**446**	**42**	**37**	**51**	**46**	**48**	**52**	**54**	**46**	**34**	**36**

Device S3

Audio segments

Scene class	Segments	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	108	12	12	12	11	11	10	10	10	10	10
Bus	108	12	12	12	11	11	10	10	10	10	10
Metro	108	12	12	12	11	11	10	10	10	10	10
Metro station	108	12	12	12	11	11	10	10	10	10	10
Park	108	12	12	12	11	11	10	10	10	10	10
Public square	108	12	12	12	11	11	10	10	10	10	10
Shopping mall	108	12	12	12	11	11	10	10	10	10	10
Street, pedestrian	108	12	12	12	11	11	10	10	10	10	10
Street, traffic	108	12	12	12	11	11	10	10	10	10	10
Tram	108	12	12	12	11	11	10	10	10	10	10
Total	**1080**	**120**	**120**	**120**	**110**	**110**	**100**	**100**	**100**	**100**	**100**

Recording locations

Scene class	Locations	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	36	3	3	4	3	4	4	4	4	4	3
Bus	50	4	4	6	5	6	6	7	5	3	4
Metro	50	3	3	10	4	5	6	4	8	3	4
Metro station	44	4	4	4	6	5	4	7	3	4	3
Park	39	4	4	4	4	4	4	4	4	3	4
Public_square	39	4	4	3	4	5	4	4	4	3	4
Shopping mall	32	4	4	3	2	3	3	4	3	3	3
Street, pedestrian	39	6	3	3	4	4	4	5	3	4	3
Street, traffic	40	4	4	4	5	4	5	4	3	3	4
Tram	50	4	4	5	8	5	7	6	5	3	3
Total	**419**	**40**	**37**	**46**	**45**	**45**	**47**	**49**	**42**	**33**	**35**

Device S4

Audio segments

Scene class	Segments	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	108	12	12	12	11	11	10	10	10	10	10
Bus	108	12	12	12	11	11	10	10	10	10	10
Metro	108	12	12	12	11	11	10	10	10	10	10
Metro station	108	12	12	12	11	11	10	10	10	10	10
Park	108	12	12	12	11	11	10	10	10	10	10
Public square	108	12	12	12	11	11	10	10	10	10	10
Shopping mall	108	12	12	12	11	11	10	10	10	10	10
Street, pedestrian	108	12	12	12	11	11	10	10	10	10	10
Street, traffic	108	12	12	12	11	11	10	10	10	10	10
Tram	108	12	12	12	11	11	10	10	10	10	10
Total	**1080**	**120**	**120**	**120**	**110**	**110**	**100**	**100**	**100**	**100**	**100**

Recording locations

Scene class	Locations	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	36	3	3	4	3	4	4	4	4	4	3
Bus	53	4	4	9	5	6	5	6	7	3	4
Metro	50	3	2	8	4	7	6	7	6	4	3
Metro station	47	5	4	4	7	5	4	6	4	4	4
Park	38	4	3	4	4	4	4	4	4	3	4
Public_square	38	4	4	3	3	5	4	4	4	3	4
Shopping mall	35	4	4	4	2	3	3	4	4	3	4
Street, pedestrian	42	7	3	3	4	4	4	4	5	4	4
Street, traffic	41	4	4	4	4	4	5	4	4	4	4
Tram	51	4	4	6	6	7	5	7	5	3	4
Total	**431**	**42**	**35**	**49**	**42**	**49**	**44**	**50**	**47**	**35**	**38**

Device S5

Audio segments

Scene class	Segments	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	108	12	12	12	11	11	10	10	10	10	10
Bus	108	12	12	12	11	11	10	10	10	10	10
Metro	108	12	12	12	11	11	10	10	10	10	10
Metro station	108	12	12	12	11	11	10	10	10	10	10
Park	108	12	12	12	11	11	10	10	10	10	10
Public square	108	12	12	12	11	11	10	10	10	10	10
Shopping mall	108	12	12	12	11	11	10	10	10	10	10
Street, pedestrian	108	12	12	12	11	11	10	10	10	10	10
Street, traffic	108	12	12	12	11	11	10	10	10	10	10
Tram	108	12	12	12	11	11	10	10	10	10	10
Total	**1080**	**120**	**120**	**120**	**110**	**110**	**100**	**100**	**100**	**100**	**100**

Recording locations

Scene class	Locations	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	38	4	3	4	3	4	4	3	5	5	3
Bus	54	3	4	6	6	6	7	8	7	3	4
Metro	51	3	3	7	4	8	6	6	7	4	3
Metro station	45	5	3	3	7	4	4	7	4	4	4
Park	36	3	4	3	3	4	4	4	4	3	4
Public_square	39	3	4	3	4	4	4	4	6	3	4
Shopping mall	33	3	4	3	2	3	3	4	4	3	4
Street, pedestrian	42	6	3	4	4	4	4	5	5	4	3
Street, traffic	38	3	3	4	4	4	4	4	4	4	4
Tram	50	4	4	4	6	5	8	7	6	3	3
Total	**426**	**37**	**35**	**41**	**43**	**46**	**48**	**52**	**52**	**36**	**36**

Device S6

Audio segments

Scene class	Segments	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	108	12	12	12	11	11	10	10	10	10	10
Bus	108	12	12	12	11	11	10	10	10	10	10
Metro	108	12	12	12	11	11	10	10	10	10	10
Metro station	108	12	12	12	11	11	10	10	10	10	10
Park	108	12	12	12	11	11	10	10	10	10	10
Public square	108	12	12	12	11	11	10	10	10	10	10
Shopping mall	108	12	12	12	11	11	10	10	10	10	10
Street, pedestrian	108	12	12	12	11	11	10	10	10	10	10
Street, traffic	108	12	12	12	11	11	10	10	10	10	10
Tram	108	12	12	12	11	11	10	10	10	10	10
Total	**1080**	**120**	**120**	**120**	**110**	**110**	**100**	**100**	**100**	**100**	**100**

Recording locations

Scene class	Locations	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	36	4	3	4	3	4	3	3	5	4	3
Bus	55	3	4	9	7	6	5	9	6	2	4
Metro	51	3	2	7	4	7	6	7	8	3	4
Metro station	47	5	4	4	9	3	3	7	4	4	4
Park	37	3	4	4	4	4	3	4	4	3	4
Public_square	39	4	4	4	4	4	3	4	5	3	4
Shopping mall	33	3	4	4	2	3	2	4	4	3	4
Street, pedestrian	39	5	3	4	4	3	4	4	4	4	4
Street, traffic	39	3	4	3	4	4	5	4	4	4	4
Tram	56	4	4	6	7	6	7	6	9	3	4
Total	**432**	**37**	**35**	**49**	**48**	**44**	**41**	**52**	**53**	**33**	**39**

Usage

The partitioning of the data was done based on the location of the original recordings. All segments recorded at the same location were included into a single subset - either **development dataset** or **evaluation dataset**. For each acoustic scene, 1440 segments recorded with device A, 108 segments recorded with device B, C and S1-S6 were included in the development dataset provided here. Evaluation dataset is provided separately.

Training / test setup

A suggested training/test partitioning of the development set is provided in order to make results reported with this dataset uniform. The partitioning is done such that the segments recorded at the same location are included into the same subset - either training or testing. The partitioning is done aiming for a 70/30 ratio between the number of segments in training and test subsets while taking into account recording locations, and selecting the closest available option.

Data from devices A, B, C, S1, S2, S3 are available in both training and test sets. Audio segments coming from devices S4, S5, and S6 are used only for testing. Since the dataset includes balanced amount of material from devices (B, C, and S1-S6), this partitioning will leave a small subset of data from devices S4-S6 unused in the training / test setup. This material can be used when using full dataset to train the system and testing it with evaluation dataset.

The setup is provided with the dataset in the directory *evaluation_setup*.

Statistics

Scene class	Train / Segments	Train / Locations	Test / Segments	Test / Locations	Unused / Segments	Unused / Locations
Airport	1393	28	296	12	613	40
Bus	1400	51	297	19	607	66
Metro	1382	47	297	20	625	65
Metro station	1380	40	297	16	627	55
Park	1429	30	297	11	578	39
Public square	1427	31	297	12	579	42
Shopping mall	1373	26	297	10	633	35
Street, pedestrian	1386	32	297	14	621	45
Street, traffic	1413	31	297	12	594	43
Tram	1379	49	296	20	628	67
Total	**13962**	**365**	**2968**	**146**	**6105**	**497**

Number of segments in train / test setup

License

License permits free academic usage. Any commercial use is strictly prohibited. For commercial use, contact dataset authors.

> Copyright (c) 2020 Tampere University and its licensors All rights reserved. Permission is hereby granted, without written agreement and without license or royalty fees, to use and copy the TAU Urban Acoustic Scenes 2020 Mobile ("Work") described in this document and composed of audio and metadata. This grant is only for experimental and non-commercial purposes, provided that the copyright notice in its entirety appear in all copies of this Work, and the original source of this Work, (Audio Research Group at Tampere University of Technology), is acknowledged in any publication that reports research using this Work. Any commercial use of the Work or any part thereof is strictly prohibited. Commercial use include, but is not limited to: - selling or reproducing the Work - selling or distributing the results or content achieved by use of the Work - providing services by using the Work.
>
> IN NO EVENT SHALL TAMPERE UNIVERSITY OR ITS LICENSORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS WORK AND ITS DOCUMENTATION, EVEN IF TAMPERE UNIVERSITY OR ITS LICENSORS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
>
> TAMPERE UNIVERSITY AND ALL ITS LICENSORS SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE WORK PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE TAMPERE UNIVERSITY HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

`class` soundata.datasets.tau2020uas_mobile.**Clip**(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

TAU Urban Acoustic Scenes 2020 Mobile Clip class

Parameters

clip_id (*str*) – id of the clip

Variables

- **audio** (*np.ndarray, float*) – path to the audio file

- **audio_path** (*str*) – path to the audio file
- **city** (*str*) – city were the audio signal was recorded
- **clip_id** (*str*) – clip id
- **identifier** (*str*) – the clip identifier
- **source_label** (*str*) – source label
- **split** (*str*) – subset the clip belongs to (for experiments): development (fold1, fold2, fold3, fold4) or evaluation
- **tags** (soundata.annotations.Tags) – tag (label) of the clip + confidence

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

property city

The clip's city.

Returns

- str - city were the audio signal was recorded

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

property identifier

The clip's identifier.

Returns

- str - clip identifier

property source_label

The clip's source label.

Returns

- str - source label

property split

The clip's split.

Returns
** str - subset the clip belongs to (for experiments)* – development (fold1, fold2, fold3, fold4) or evaluation

property tags

The clip's tags.

Returns

- annotations.Tags - tag (label) of the clip + confidence

to_jams()

Get the clip's data in jams format

Returns
jams.JAMS – the clip's data in jams format

class soundata.datasets.tau2020uas_mobile.**Dataset**(*data_home=None*)

The TAU Urban Acoustic Scenes 2020 Mobile dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

cite()

Print the reference

clip_ids

Return clip ids

Returns
list – A list of clip ids

clipgroup_ids

Return clip ids

Returns
list – A list of clip ids

property default_path

Get the default path for the dataset

Returns
str – Local path to the dataset

download(*partial_download=None*, *force_overwrite=False*, *cleanup=False*)

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- `ValueError` – if invalid keys are passed to partial_download
- `IOError` – if a downloaded file's checksum is different from expected

`explore_dataset`(*clip_id=None*)

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters

clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

`license`()

Print the license

`load_audio`(**args*, ***kwargs*)

Load a TAU Urban Acoustic Scenes 2020 Mobile audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate of 44100 without resampling.

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

`load_clipgroups`()

Load all clipgroups in the dataset

Returns

dict – {*clipgroup_id*: clipgroup data}

Raises

`NotImplementedError` – If the dataset does not support Clipgroups

`load_clips`()

Load all clips in the dataset

Returns

dict – {*clip_id*: clip data}

Raises

`NotImplementedError` – If the dataset does not support Clips

`validate`(*verbose=True*)

Validate if the stored dataset is a valid version

Parameters

verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

soundata.datasets.tau2020uas_mobile.**load_audio**(*fhandle: BinaryIO*, *sr=None*) → Tuple[numpy.ndarray, float]

Load a TAU Urban Acoustic Scenes 2020 Mobile audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate of 44100 without resampling.

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

4.5.21 TAU Urban Acoustic Scenes 2022 Mobile

TAU Urban Acoustic Scenes 2022 Mobile Loader

Dataset Info

TAU Urban Acoustic Scenes 2022 Mobile, Development and Evaluation datasets

Audio Research Group, Tampere University of Technology

Authors

- Toni Heittola
- Annamaria Mesaros
- Tuomas Virtanen

Recording and annotation

- Henri Laakso
- Ronal Bejarano Rodriguez
- Toni Heittola

Links

- Development dataset
- Evaluation dataset

Dataset

TAU Urban Acoustic Scenes 2022 Mobile development dataset consists of 1-seconds audio segments from 10 acoustic scenes:

- Airport - *airport*
- Indoor shopping mall - *shopping_mall*
- Metro station - *metro_station*
- Pedestrian street - *street_pedestrian*
- Public square - *public_square*

- Street with medium level of traffic - *street_traffic*
- Travelling by a tram - *tram*
- Travelling by a bus - *bus*
- Travelling by an underground metro - *metro*
- Urban park - *park*

The dataset contains the same material than TAU Urban Acoustic Scenes 2020 Mobile development dataset, 10-second audio segments have been split into non-overlapping 1-second segments for 2022 version of the dataset.

A detailed description of the data recording and annotation procedure is available in:

```
Annamaria Mesaros, Toni Heittola, and Tuomas Virtanen.
"Acoustic scene classification in DCASE 2020 Challenge:
generalization across devices and low complexity solutions",
In Proceedings of the Detection and Classification of Acoustic
Scenes and Events 2020 Workshop (DCASE2020), Tokyo, Japan, 2020.
```

Recordings were made with three devices (A, B and C) that captured audio simultaneously and 6 simulated devices (S1-S6). Each acoustic scene has 1440 segments (240 minutes of audio) recorded with device A (main device) and 108 segments of parallel audio (18 minutes) each recorded with devices B,C, and S1-S6.

Development dataset

The dataset contains in total 64 hours of audio.

Evaluation dataset

The dataset contains in total 33 hours of audio.

The dataset was collected by Tampere University of Technology between 05/2018 -11/2018. The data collection has received funding from the European Research Council under the ERC Grant Agreement 637422 EVERYSOUND.

Preparation of the dataset

The dataset was recorded in 12 large European cities: Amsterdam, Barcelona, Helsinki, Lisbon, London, Lyon, Madrid, Milan, Prague, Paris, Stockholm, and Vienna. For all acoustic scenes, audio was captured in multiple locations: different streets, different parks, different shopping malls. In each location, multiple 2-3 minute long audio recordings were captured in a few slightly different positions (2-4) within the selected location. Collected audio material was cut into segments of 10 seconds length.

The main recording device (referred to as device A) consists of a binaural Soundman OKM IIKlassik/studio A3 electret in-ear microphone and a Zoom F8 audio recorder using 48 kHz sampling rate and 24 bit resolution. During the recording, the microphones were worn by the recording person in the ears, and head movement was kept to minimum.

Devices B and C are commonly available customer devices (e.g. smartphones, cameras) and were handled in typical ways (e.g. hand held). The audio recordings from these devices are of different quality than device A. All simultaneous recordings are time synchronized.

Post-processing of the recorded audio involves aspects related to privacy of recorded individuals, and possible errors in the recording process. The material was screened for content, and segments containing close microphone conversation were eliminated. Some interferences from mobile phones are audible, but are considered part of real-world recording process. In addition, data from device A was resampled and averaged into a single channel, to align with the properties of the data recorded with devices B and C.

Additionally, 11 mobile devices S1-S11 are simulated using the audio recorded with device A, impulse responses recorded with real devices, and additional dynamic range compression, in order to simulate realistic recordings. A recording from device A is processed through convolution with the selected Si impulse response, then processed with a selected set of parameters for dynamic range compression (device specific). The impulse responses are proprietary data and will not be published.

All provided audio data is single-channel, having a 44.1 KHz sampling rate, and 24 bit resolution.

A subset of the dataset has been previously published as TUT Urban Acoustic Scenes 2019 Development dataset. Audio segment filenames are retained for the segments coming from this dataset.

Dataset statistics

The development set contains data from 10 cities and 9 devices: 3 real devices (A, B, C) and 6 simulated devices (S1-S6). Data from devices B, C and S1-S6 consists of randomly selected segments from the simultaneous recordings, therefore all overlap with the data from device A, but not necessarily with each other. The total amount of audio in the development set is **64 hours**. The evaluation dataset (TAU Urban Acoustic Scenes 2022 Mobile evaluation) contains data from all 12 cities, and five new devices (not available in the development set): real device D and simulated devices S7-S11.

Device A

Audio segments

Scene class	Segments	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	14400	1280	1490	1440	1450	1440	1440	1560	1440	1580	1280
Bus	14400	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440
Metro	14400	1410	1440	1440	1460	1440	1440	1440	1440	1450	1440
Metro station	14400	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440
Park	14400	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440
Public square	14400	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440
Shopping mall	14400	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440
Street, pedestrian	14400	1450	1450	1440	1450	1440	1440	1440	1440	1450	1400
Street, traffic	14400	1440	1440	1440	1440	1440	1440	1440	1440	1440	1440
Tram	14400	1430	1450	1440	1440	1440	1440	1440	1440	1440	1440
Total	**144000**	**14210**	**14470**	**14400**	**14440**	**14400**	**14400**	**14520**	**14400**	**14560**	**14200**

Recording locations

Scene class	Locations	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	40	4	3	4	3	4	4	4	6	5	3
Bus	71	4	4	11	7	7	7	11	10	6	4
Metro	67	3	5	11	4	9	8	9	10	4	4
Metro station	57	5	6	4	12	5	4	9	4	4	4
Park	41	4	4	4	4	4	4	4	4	5	4
Public_square	43	4	4	4	4	5	4	4	6	4	4
Shopping mall	36	4	4	4	2	3	3	4	4	4	4
Street, pedestrian	46	7	4	4	4	4	5	5	5	4	4
Street, traffic	43	4	4	4	5	4	6	4	4	4	4
Tram	70	4	4	6	9	7	11	9	11	5	4
Total	**514**	**43**	**42**	**56**	**54**	**52**	**56**	**63**	**65**	**45**	**39**

Device B

Audio segments

Scene class	Segments	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	1070	110	120	120	110	110	100	100	100	100	100
Bus	1080	120	120	120	110	110	100	100	100	100	100
Metro	1080	120	120	120	110	110	100	100	100	100	100
Metro station	1080	120	120	120	110	110	100	100	100	100	100
Park	1080	120	120	120	110	110	100	100	100	100	100
Public square	1070	110	120	120	110	110	100	100	100	100	100
Shopping mall	1080	120	120	120	110	110	100	100	100	100	100
Street, pedestrian	1080	120	120	120	110	110	100	100	100	100	100
Street, traffic	1080	120	120	120	110	110	100	100	100	100	100
Tram	1080	120	120	120	110	110	100	100	100	100	100
Total	**10780**	**1180**	**1200**	**1200**	**1100**	**1100**	**1000**	**1000**	**1000**	**1000**	**1000**

Recording locations

Scene class	Locations	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	36	3	3	4	3	3	4	4	5	4	3
Bus	57	4	4	9	7	6	5	8	7	3	4
Metro	47	3	4	6	4	6	5	6	6	4	4
Metro station	45	4	4	3	8	5	3	7	3	4	4
Park	37	4	4	4	4	4	3	4	3	3	4
Public_square	37	3	4	4	4	5	3	4	4	3	3
Shopping mall	34	4	4	4	2	3	3	4	4	3	3
Street, pedestrian	43	6	3	4	4	4	5	5	4	4	4
Street, traffic	41	4	4	4	4	4	6	4	4	4	4
Tram	50	4	4	5	6	5	5	7	7	3	4
Total	**427**	**39**	**37**	**47**	**46**	**44**	**42**	**53**	**47**	**35**	**37**

Device C

Audio segments

Scene class	Segments	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	1070	110	120	120	110	110	100	100	100	100	100
Bus	1080	120	120	120	110	110	100	100	100	100	100
Metro	1080	120	120	120	110	110	100	100	100	100	100
Metro station	1080	120	120	120	110	110	100	100	100	100	100
Park	1080	120	120	120	110	110	100	100	100	100	100
Public square	1080	120	120	120	110	110	100	100	100	100	100
Shopping mall	1070	120	120	120	100	110	100	100	100	100	100
Street, pedestrian	1080	120	120	120	110	110	100	100	100	100	100
Street, traffic	1080	120	120	120	110	110	100	100	100	100	100
Tram	1070	110	120	120	110	110	100	100	100	100	100
Total	**10770**	**1180**	**1200**	**1200**	**1090**	**1100**	**1000**	**1000**	**1000**	**1000**	**1000**

Recording locations

Scene class	Locations	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	38	4	3	4	3	3	4	4	5	5	3
Bus	50	4	4	7	6	5	4	7	7	3	3
Metro	54	3	3	6	4	9	6	7	8	4	4
Metro station	48	5	3	4	8	5	4	7	4	4	4
Park	39	4	4	4	4	4	4	4	4	3	4
Public_square	40	4	3	4	4	4	4	4	6	3	4
Shopping mall	35	4	4	4	2	3	3	4	4	3	4
Street, pedestrian	41	6	3	4	4	3	5	4	5	4	3
Street, traffic	40	4	3	4	4	4	6	4	4	4	3
Tram	51	4	4	5	6	4	8	6	7	3	4
Total	**436**	**42**	**34**	**46**	**45**	**44**	**48**	**51**	**54**	**36**	**36**

Device S1

Audio segments

Scene class	Segments	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	1080	120	120	120	110	110	100	100	100	100	100
Bus	1080	120	120	120	110	110	100	100	100	100	100
Metro	1080	120	120	120	110	110	100	100	100	100	100
Metro station	1080	120	120	120	110	110	100	100	100	100	100
Park	1080	120	120	120	110	110	100	100	100	100	100
Public square	1080	120	120	120	110	110	100	100	100	100	100
Shopping mall	1080	120	120	120	110	110	100	100	100	100	100
Street, pedestrian	1080	120	120	120	110	110	100	100	100	100	100
Street, traffic	1080	120	120	120	110	110	100	100	100	100	100
Tram	1080	120	120	120	110	110	100	100	100	100	100
Total	**10800**	**1200**	**1200**	**1200**	**1100**	**1100**	**1000**	**1000**	**1000**	**1000**	**1000**

Recording locations

Scene class	Locations	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	37	4	3	4	3	4	4	4	4	4	3
Bus	54	4	4	8	6	6	6	7	6	3	4
Metro	50	3	3	8	4	7	6	6	6	4	3
Metro station	48	5	4	4	9	5	4	5	4	4	4
Park	36	4	4	4	4	3	4	3	3	3	4
Public_square	37	4	4	4	4	4	4	3	3	3	4
Shopping mall	33	4	4	4	2	3	3	3	3	3	4
Street, pedestrian	40	6	3	4	4	3	5	2	5	4	4
Street, traffic	40	4	4	4	4	4	6	3	3	4	4
Tram	52	4	4	5	7	6	7	6	6	3	4
Total	**427**	**42**	**37**	**49**	**47**	**45**	**49**	**42**	**43**	**35**	**38**

Device S2

Audio segments

Scene class	Segments	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	1080	120	120	120	110	110	100	100	100	100	100
Bus	1080	120	120	120	110	110	100	100	100	100	100
Metro	1080	120	120	120	110	110	100	100	100	100	100
Metro station	1080	120	120	120	110	110	100	100	100	100	100
Park	1080	120	120	120	110	110	100	100	100	100	100
Public square	1080	120	120	120	110	110	100	100	100	100	100
Shopping mall	1080	120	120	120	110	110	100	100	100	100	100
Street, pedestrian	1080	120	120	120	110	110	100	100	100	100	100
Street, traffic	1080	120	120	120	110	110	100	100	100	100	100
Tram	1080	120	120	120	110	110	100	100	100	100	100
Total	**10800**	**1200**	**1200**	**1200**	**1100**	**1100**	**1000**	**1000**	**1000**	**1000**	**1000**

Recording locations

Scene class	Locations	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	36	3	3	4	3	4	4	4	4	4	3
Bus	58	4	4	9	6	6	7	9	6	3	4
Metro	55	3	3	10	4	8	8	5	7	4	3
Metro station	49	5	4	4	7	5	4	8	4	4	4
Park	38	4	4	4	4	4	4	4	4	2	4
Public_square	41	4	4	4	4	5	4	4	5	3	4
Shopping mall	34	4	4	3	2	3	3	4	4	3	4
Street, pedestrian	42	7	3	4	4	3	5	5	4	4	3
Street, traffic	42	4	4	4	5	4	6	4	4	4	3
Tram	51	4	4	5	7	6	7	7	4	3	4
Total	**446**	**42**	**37**	**51**	**46**	**48**	**52**	**54**	**46**	**34**	**36**

Device S3

Audio segments

Scene class	Segments	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	1080	120	120	120	110	110	100	100	100	100	100
Bus	1080	120	120	120	110	110	100	100	100	100	100
Metro	1080	120	120	120	110	110	100	100	100	100	100
Metro station	1080	120	120	120	110	110	100	100	100	100	100
Park	1080	120	120	120	110	110	100	100	100	100	100
Public square	1080	120	120	120	110	110	100	100	100	100	100
Shopping mall	1080	120	120	120	110	110	100	100	100	100	100
Street, pedestrian	1080	120	120	120	110	110	100	100	100	100	100
Street, traffic	1080	120	120	120	110	110	100	100	100	100	100
Tram	1080	120	120	120	110	110	100	100	100	100	100
Total	**10800**	**1200**	**1200**	**1200**	**1100**	**1100**	**1000**	**1000**	**1000**	**1000**	**1000**

Recording locations

Scene class	Locations	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	36	3	3	4	3	4	4	4	4	4	3
Bus	50	4	4	6	5	6	6	7	5	3	4
Metro	50	3	3	10	4	5	6	4	8	3	4
Metro station	44	4	4	4	6	5	4	7	3	4	3
Park	39	4	4	4	4	4	4	4	4	3	4
Public_square	39	4	4	3	4	5	4	4	4	3	4
Shopping mall	32	4	4	3	2	3	3	4	3	3	3
Street, pedestrian	39	6	3	3	4	4	4	5	3	4	3
Street, traffic	40	4	4	4	5	4	5	4	3	3	4
Tram	50	4	4	5	8	5	7	6	5	3	3
Total	**419**	**40**	**37**	**46**	**45**	**45**	**47**	**49**	**42**	**33**	**35**

Device S4

Audio segments

Scene class	Segments	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	1080	120	120	120	110	110	100	100	100	100	100
Bus	1080	120	120	120	110	110	100	100	100	100	100
Metro	1080	120	120	120	110	110	100	100	100	100	100
Metro station	1080	120	120	120	110	110	100	100	100	100	100
Park	1080	120	120	120	110	110	100	100	100	100	100
Public square	1080	120	120	120	110	110	100	100	100	100	100
Shopping mall	1080	120	120	120	110	110	100	100	100	100	100
Street, pedestrian	1080	120	120	120	110	110	100	100	100	100	100
Street, traffic	1080	120	120	120	110	110	100	100	100	100	100
Tram	1080	120	120	120	110	110	100	100	100	100	100
Total	**10800**	**1200**	**1200**	**1200**	**1100**	**1100**	**1000**	**1000**	**1000**	**1000**	**1000**

Recording locations

Scene class	Locations	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	36	3	3	4	3	4	4	4	4	4	3
Bus	53	4	4	9	5	6	5	6	7	3	4
Metro	50	3	2	8	4	7	6	7	6	4	3
Metro station	47	5	4	4	7	5	4	6	4	4	4
Park	38	4	3	4	4	4	4	4	4	3	4
Public_square	38	4	4	3	3	5	4	4	4	3	4
Shopping mall	35	4	4	4	2	3	3	4	4	3	4
Street, pedestrian	42	7	3	3	4	4	4	4	5	4	4
Street, traffic	41	4	4	4	4	4	5	4	4	4	4
Tram	51	4	4	6	6	7	5	7	5	3	4
Total	**431**	**42**	**35**	**49**	**42**	**49**	**44**	**50**	**47**	**35**	**38**

Device S5

Audio segments

Scene class	Segments	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	1080	120	120	120	110	110	100	100	100	100	100
Bus	1080	120	120	120	110	110	100	100	100	100	100
Metro	1080	120	120	120	110	110	100	100	100	100	100
Metro station	1080	120	120	120	110	110	100	100	100	100	100
Park	1080	120	120	120	110	110	100	100	100	100	100
Public square	1080	120	120	120	110	110	100	100	100	100	100
Shopping mall	1080	120	120	120	110	110	100	100	100	100	100
Street, pedestrian	1080	120	120	120	110	110	100	100	100	100	100
Street, traffic	1080	120	120	120	110	110	100	100	100	100	100
Tram	1080	120	120	120	110	110	100	100	100	100	100
Total	**10800**	**1200**	**1200**	**1200**	**1100**	**1100**	**1000**	**1000**	**1000**	**1000**	**1000**

Recording locations

Scene class	Locations	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	38	4	3	4	3	4	4	3	5	5	3
Bus	54	3	4	6	6	6	7	8	7	3	4
Metro	51	3	3	7	4	8	6	6	7	4	3
Metro station	45	5	3	3	7	4	4	7	4	4	4
Park	36	3	4	3	3	4	4	4	4	3	4
Public_square	39	3	4	3	4	4	4	4	6	3	4
Shopping mall	33	3	4	3	2	3	3	4	4	3	4
Street, pedestrian	42	6	3	4	4	4	4	5	5	4	3
Street, traffic	38	3	3	4	4	4	4	4	4	4	4
Tram	50	4	4	4	6	5	8	7	6	3	3
Total	**426**	**37**	**35**	**41**	**43**	**46**	**48**	**52**	**52**	**36**	**36**

Device S6

Audio segments

Scene class	Segments	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	1080	120	120	120	110	110	100	100	100	100	100
Bus	1080	120	120	120	110	110	100	100	100	100	100
Metro	1080	120	120	120	110	110	100	100	100	100	100
Metro station	1080	120	120	120	110	110	100	100	100	100	100
Park	1080	120	120	120	110	110	100	100	100	100	100
Public square	1080	120	120	120	110	110	100	100	100	100	100
Shopping mall	1080	120	120	120	110	110	100	100	100	100	100
Street, pedestrian	1080	120	120	120	110	110	100	100	100	100	100
Street, traffic	1080	120	120	120	110	110	100	100	100	100	100
Tram	1080	120	120	120	110	110	100	100	100	100	100
Total	**10800**	**1200**	**1200**	**1200**	**1100**	**1100**	**1000**	**1000**	**1000**	**1000**	**1000**

Recording locations

Scene class	Locations	Barcelona	Helsinki	Lisbon	London	Lyon	Milan	Paris	Prague	Stockholm	Vienna
Airport	36	4	3	4	3	4	3	3	5	4	3
Bus	55	3	4	9	7	6	5	9	6	2	4
Metro	51	3	2	7	4	7	6	7	8	3	4
Metro station	47	5	4	4	9	3	3	7	4	4	4
Park	37	3	4	4	4	4	3	4	4	3	4
Public_square	39	4	4	4	4	4	3	4	5	3	4
Shopping mall	33	3	4	4	2	3	2	4	4	3	4
Street, pedestrian	39	5	3	4	4	3	4	4	4	4	4
Street, traffic	39	3	4	3	4	4	5	4	4	4	4
Tram	56	4	4	6	7	6	7	6	9	3	4
Total	**432**	**37**	**35**	**49**	**48**	**44**	**41**	**52**	**53**	**33**	**39**

Usage

The partitioning of the data was done based on the location of the original recordings. All segments recorded at the same location were included into a single subset - either **development dataset** or **evaluation dataset**. For each acoustic scene, 1440 segments recorded with device A, 108 segments recorded with device B, C and S1-S6 were included in the development dataset provided here. Evaluation dataset is provided separately.

Training / test setup

A suggested training/test partitioning of the development set is provided in order to make results reported with this dataset uniform. The partitioning is done such that the segments recorded at the same location are included into the same subset - either training or testing. The partitioning is done aiming for a 70/30 ratio between the number of segments in training and test subsets while taking into account recording locations, and selecting the closest available option.

Data from devices A, B, C, S1, S2, S3 are available in both training and test sets. Audio segments coming from devices S4, S5, and S6 are used only for testing. Since the dataset includes balanced amount of material from devices (B, C, and S1-S6), this partitioning will leave a small subset of data from devices S4-S6 unused in the training / test setup. This material can be used when using full dataset to train the system and testing it with evaluation dataset.

The setup is provided with the dataset in the directory *evaluation_setup*.

Statistics

Scene class	Train / Segments	Train / Locations	Test / Segments	Test / Locations	Unused / Segments	Unused / Locations
Airport	13930	28	2960	12	6130	40
Bus	14000	51	2970	19	6070	66
Metro	13820	47	2970	20	6250	65
Metro station	13800	40	2970	16	6270	55
Park	14290	30	2970	11	5780	39
Public square	14270	31	2970	12	5790	42
Shopping mall	13730	26	2970	10	6330	35
Street, pedestrian	13860	32	2970	14	6210	45
Street, traffic	14130	31	2970	12	5940	43
Tram	13790	49	2960	20	6280	67
Total	**139620**	**365**	**29680**	**146**	**610500**	**497**

Number of segments in train / test setup

License

License permits free academic usage. Any commercial use is strictly prohibited. For commercial use, contact dataset authors.

> Copyright (c) 2022 Tampere University and its licensors All rights reserved. Permission is hereby granted, without written agreement and without license or royalty fees, to use and copy the TAU Urban Acoustic Scenes 2022 Mobile ("Work") described in this document and composed of audio and metadata. This grant is only for experimental and non-commercial purposes, provided that the copyright notice in its entirety appear in all copies of this Work, and the original source of this Work, (Audio Research Group at Tampere University of Technology), is acknowledged in any publication that reports research using this Work. Any commercial use of the Work or any part thereof is strictly prohibited. Commercial use include, but is not limited to: - selling or reproducing the Work - selling or distributing the results or content achieved by use of the Work - providing services by using the Work.
>
> IN NO EVENT SHALL TAMPERE UNIVERSITY OR ITS LICENSORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS WORK AND ITS DOCUMENTATION, EVEN IF TAMPERE UNIVERSITY OR ITS LICENSORS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
>
> TAMPERE UNIVERSITY AND ALL ITS LICENSORS SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE WORK PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE TAMPERE UNIVERSITY HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

`class` soundata.datasets.tau2022uas_mobile.**Clip**(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

TAU Urban Acoustic Scenes 2022 Mobile Clip class

Parameters

clip_id (*str*) – id of the clip

Variables

- **audio** (*np.ndarray, float*) – path to the audio file

- **audio_path** (*str*) – path to the audio file
- **city** (*str*) – city were the audio signal was recorded
- **clip_id** (*str*) – clip id
- **identifier** (*str*) – the clip identifier
- **source_label** (*str*) – source label
- **split** (*str*) – subset the clip belongs to (for experiments): development (fold1, fold2, fold3, fold4) or evaluation
- **tags** (soundata.annotations.Tags) – tag (label) of the clip + confidence

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

property city

The clip's city.

Returns

- str - city were the audio signal was recorded

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

property identifier

The clip's identifier.

Returns

- str - clip identifier

property source_label

The clip's source label.

Returns

- str - source label

property split

The clip's split.

Returns
** str - subset the clip belongs to (for experiments)* – development (fold1, fold2, fold3, fold4) or evaluation

property tags

The clip's tags.

Returns

- annotations.Tags - tag (label) of the clip + confidence

to_jams()

Get the clip's data in jams format

Returns
jams.JAMS – the clip's data in jams format

class soundata.datasets.tau2022uas_mobile.**Dataset**(*data_home=None*)

The TAU Urban Acoustic Scenes 2022 Mobile dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

cite()

Print the reference

clip_ids

Return clip ids

Returns
list – A list of clip ids

clipgroup_ids

Return clip ids

Returns
list – A list of clip ids

property default_path

Get the default path for the dataset

Returns
str – Local path to the dataset

download(*partial_download=None*, *force_overwrite=False*, *cleanup=False*)

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- `ValueError` – if invalid keys are passed to partial_download
- `IOError` – if a downloaded file's checksum is different from expected

`explore_dataset`(*clip_id=None*)

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters

clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

`license`()

Print the license

`load_audio`(**args*, ***kwargs*)

Load a TAU Urban Acoustic Scenes 2022 Mobile audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate of 44100 without resampling.

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

`load_clipgroups`()

Load all clipgroups in the dataset

Returns

dict – {*clipgroup_id*: clipgroup data}

Raises

`NotImplementedError` – If the dataset does not support Clipgroups

`load_clips`()

Load all clips in the dataset

Returns

dict – {*clip_id*: clip data}

Raises

`NotImplementedError` – If the dataset does not support Clips

`validate`(*verbose=True*)

Validate if the stored dataset is a valid version

Parameters

verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

soundata.datasets.tau2022uas_mobile.**load_audio**(*fhandle: BinaryIO*, *sr=None*) → Tuple[numpy.ndarray, float]

Load a TAU Urban Acoustic Scenes 2022 Mobile audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate of 44100 without resampling.

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

4.5.22 TUT Sound events 2017

TUT Sound events 2017 Dataset Loader

Dataset Info

TUT Sound events 2017, Development and Evaluation datasets

Audio Research Group, Tampere University of Technology

Authors

- Toni Heittola
- Annamaria Mesaros
- Tuomas Virtanen

Recording and annotation

- Eemi Fagerlund
- Aku Hiltunen

Links

- Development dataset
- Evaluation dataset

Dataset

TUT Sound Events 2017 dataset consists of two subsets: development dataset and evaluation dataset. Partitioning of data into these subsets was done based on the amount of examples available for each sound event class, while also taking into account recording location. Because the event instances belonging to different classes are distributed unevenly within the recordings, the partitioning of individual classes can be controlled only to a certain extent, but so that the majority of events are in the development set.

A detailed description of the data recording and annotation procedure is available in:

```
Annamaria Mesaros, Toni Heittola, and Tuomas Virtanen.
"TUT database for acoustic scene classification and sound event
detection", In 24th European Signal Processing Conference 2016,
Budapest, Hungary, 2016.
```

TUT Sound events 2017, development and evaluation datasets consist of 24 and 8 audio recordings from a single acoustic scene respectively:

- Development: Street (outdoor), totaling 1:32:08
- Evaluation: Street (outdoor), totaling 29:09

The dataset was collected in Finland by Tampere University of Technology between 06/2015 - 01/2016. The data collection has received funding from the European Research Council under the ERC Grant Agreement 637422 EVERYSOUND.

Preparation of the dataset

The recordings were captured each in a different location (different streets). The equipment used for recording consists of a binaural Soundman OKM II Klassik/studio A3 electret in-ear microphone and a Roland Edirol R-09 wave recorder using 44.1 kHz sampling rate and 24 bit resolution.

For audio material recorded in private places, written consent was obtained from all people involved. Material recorded in public places (residential area) does not require such consent.

Individual sound events in each recording were annotated by a research assistant using freely chosen labels for sounds. The annotator was trained first on few example recordings. He was instructed to annotate all audible sound events, and choose event labels freely. This resulted in a large set of raw labels. Mapping of the raw labels was performed, merging sounds into classes described by their source before selecting target classes. Target sound event classes for the dataset were selected based on the frequency of the obtained labels, resulting in selection of most common sounds for the street acoustic scene, in sufficient numbers for learning acoustic models. Mapping of the raw labels was performed, merging sounds into classes described by their source, for example “car passing by”, “car engine running”, “car idling”, etc into “car”, sounds produced by buses and trucks into “large vehicle”, “children yelling” and “children talking” into “children”, etc.

Due to the high level of subjectivity inherent to the annotation process, a verification of the reference annotation was done using these mapped classes. Three persons (other than the annotator) listened to each audio segment annotated as belonging to one of these classes, marking agreement about the presence of the indicated sound within the segment. Agreement/disagreement did not take into account the sound event onset and offset, only the presence of the sound event within the annotated segment. Event instances that were confirmed by at least one person were kept, resulting in elimination of about 10% of the original event instances in the development set.

The original metadata file is available in the directory *non_verified*.

The ground truth is provided as a list of the sound events present in the recording, with annotated onset and offset for each sound instance. Annotations with only targeted sound events classes are in the directory *meta*.

The sound event instance counts for the dataset are shown below.

Development set

	Development dataset		Evaluation dataset
Event label	Verified set	Non-verified set	Verified set
brakes squeaking	52	59	23
car	304	304	106
children	44	58	15
large vehicle	61	61	24
people speaking	89	117	37
people walking	109	130	42
Total	**659**	**729**	**247**

Usage

Partitioning of data into **development dataset** and **evaluation dataset** was done based on the amount of examples available for each event class, while also taking into account recording location. Ideally the subsets should have the same amount of data for each class, or at least the same relative amount, such as a 70-30% split. Because the event instances belonging to different classes are distributed unevenly within the recordings, the partitioning of individual classes can be controlled only to a certain extent.

The split condition was relaxed so that 65-75% of instances of each class were selected into the development set.

Cross-validation setup

The setup is provided with the dataset in the directory *evaluation_setup*.

License

See file EULA.pdf

`class soundata.datasets.tut2017se.Clip(`*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*`)`

TUT Sound events 2017 Clip class

Parameters

clip_id (*str*) – id of the clip

Variables

- **audio** (*np.ndarray, float*) – path to the audio file
- **audio_path** (*str*) – path to the audio file
- **annotations_path** (*str*) – path to the annotations file
- **clip_id** (*str*) – clip id
- **events** (soundata.annotations.Events) – sound events with start time, end time, label and confidence
- **non_verified_annotations_path** (*str*) – path to the non-verified annotations file
- **non_verified_events** (soundata.annotations.Events) – non-verified sound events with start time, end time, label and confidence
- **split** (*str*) – subset the clip belongs to (for experiments): development (fold1, fold2, fold3, fold4) or evaluation

`property audio: Optional[Tuple[numpy.ndarray, float]]`

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

events

The clip's events.

Returns

- annotations.Events - sound events with start time, end time, label and confidence

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

non_verified_events

The clip's non verified events path.

Returns

- str - path to the non-verified annotations file

property split

The clip's split.

Returns
** str - subset the clip belongs to (for experiments)* – development (fold1, fold2, fold3, fold4) or evaluation

to_jams()

Get the clip's data in jams format

Returns
jams.JAMS – the clip's data in jams format

class soundata.datasets.tut2017se.**Dataset**(*data_home=None*)

The TUT Sound events 2017 dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

`cite()`

Print the reference

`clip_ids`

Return clip ids

Returns
list – A list of clip ids

`clipgroup_ids`

Return clip ids

Returns
list – A list of clip ids

`property default_path`

Get the default path for the dataset

Returns
str – Local path to the dataset

`download(partial_download=None, force_overwrite=False, cleanup=False)`

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- `ValueError` – if invalid keys are passed to partial_download
- `IOError` – if a downloaded file's checksum is different from expected

`explore_dataset(clip_id=None)`

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters
clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

`license()`

Print the license

`load_audio(*args, **kwargs)`

Load a TUT Sound events 2017 audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate of 44100 without resampling.

Returns

- np.ndarray - the stereo audio signal
- float - The sample rate of the audio file

`load_clipgroups()`

Load all clipgroups in the dataset

Returns
dict – {*clipgroup_id*: clipgroup data}

Raises
`NotImplementedError` – If the dataset does not support Clipgroups

`load_clips()`

Load all clips in the dataset

Returns
dict – {*clip_id*: clip data}

Raises
`NotImplementedError` – If the dataset does not support Clips

`load_events(*args, **kwargs)`

Load an TUT Sound events 2017 annotation file :Parameters: * **fhandle** (*str or file-like*) – File-like object or path to the sound

- **events annotation file**

Returns
Events – sound events annotation data

`validate(verbose=True)`

Validate if the stored dataset is a valid version

Parameters
verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

`soundata.datasets.tut2017se.load_audio(fhandle: BinaryIO, sr=None)` → Tuple[numpy.ndarray, float]

Load a TUT Sound events 2017 audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate of 44100 without resampling.

Returns

- np.ndarray - the stereo audio signal
- float - The sample rate of the audio file

`soundata.datasets.tut2017se.load_events(fhandle: TextIO)` → *Events*

Load an TUT Sound events 2017 annotation file :Parameters: * **fhandle** (*str or file-like*) – File-like object or path to the sound

- **events annotation file**

Returns
Events – sound events annotation data

4.5.23 URBAN-SED

URBAN-SED Dataset Loader

Dataset Info

URBAN-SED

URBAN-SED (c) by Justin Salamon, Duncan MacConnell, Mark Cartwright, Peter Li, and Juan Pablo Bello.
URBAN-SED is licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0).
You should have received a copy of the license along with this work. If not, see
<http://creativecommons.org/licenses/by/4.0/>.

Created By:

Justin Salamon*^, Duncan MacConnell*, Mark Cartwright*, Peter Li*, and Juan Pablo Bello*.
* Music and Audio Research Lab (MARL), New York University, USA
^ Center for Urban Science and Progress (CUSP), New York University, USA
http://urbansed.weebly.com
http://steinhardt.nyu.edu/marl/
http://cusp.nyu.edu/

Version 2.0.0

- Audio files generated with scaper v0.1.0 (identical to audio in URBAN-SED 1.0)
- Jams annotation files generated with scaper v0.1.0 and updated to comply with scaper v1.0.0 (namespace changed from "sound_event" to "scaper")
- NOTE: due to updates to the scaper library, regenerating the audio from the jams annotations using scaper >=1.0.0 will result in audio files that are highly similar, but not identical, to the audio files provided. This is because the provided audio files were generated with scaper v0.1.0 and have been purposely kept the same as in URBAN-SED v1.0 to ensure comparability to previously published results.

Description

URBAN-SED is a dataset of 10,000 soundscapes with sound event annotations generated using scaper (github.com/justinsalamon/scaper).

A detailed description of the dataset is provided in the following article:

A summary is provided here:

- The dataset includes 10,000 soundscapes, totals almost 30 hours and includes close to 50,000 annotated sound events
- Complete annotations are provided in JAMS format, and simplified annotations are provided as tab-separated text files
- Every soundscape is 10 seconds long and has a background of Brownian noise resembling the typical "hum" often heard in urban environments
- **Every soundscape contains between 1-9 sound events from the following classes:**

  - air_conditioner, car_horn, children_playing, dog_bark, drilling, engine_idling, gun_shot, jackhammer, siren and street_music

- The source material for the sound events are the clips from the UrbanSound8K dataset
- **URBAN-SED comes pre-sorted into three sets: train, validate and test:**
  - There are 6000 soundscapes in the training set, generated using clips from folds 1-6 in UrbanSound8K
  - There are 2000 soundscapes in the validation set, generated using clips from folds 7-8 in UrbanSound8K
  - There are 2000 soundscapes in the test set, generated using clips from folds 9-10 in UrbanSound8K
- Further details about how the soundscapes were generated including the distribution of sound event start times, durations, signal-to-noise ratios, pitch shifting, time stretching, and the range of sound event polyphony (overlap) can be found in Section 3 of the aforementioned scaper paper
- The scripts used to generated URBAN-SED using scaper can be found here: https://github.com/justinsalamon/scaper_waspaa2017/tree/master/notebooks

Audio Files Included

- 10,000 synthesized soundscapes in single channel (mono), 44100Hz, 16-bit, WAV format.
- The files are split into a training set (6000), validation set (2000) and test set (2000).

Annotation Files Included

The annotations list the sound events that occur in every soundscape. The annotations are "strong", meaning for every sound event the annotations include (at least) the start time, end time, and label of the sound event. Sound events come from the following 10 labels (categories):

- air_conditioner, car_horn, children_playing, dog_bark, drilling, engine_idling, gun_shot, jackhammer,

siren, street_music

There are two types of annotations: full annotations in JAMS format, and simplified annotations in tab-separated txt format.

JAMS Annotations

- The full annotations are distributed in JAMS format (https://github.com/marl/jams).
- There are 10,000 JAMS annotation files, each one corresponding to a single soundscape with the same filename (other than the extension)
- Each JAMS file contains a single annotation in the scaper namespace format - jams >=v0.3.2 is required in order to load the annotation into python with jams:

import jams jam = jams.load('soundscape_train_bimodal0.jams'). * The value of each observation (sound event) is a dictionary storing all scaper-related sound event parameters:

- label, source_file, source_time, event_time, event_duration, snr, role, pitch_shift, time_stretch.
- Note: the event_duration stored in the value dictionary represents the specified duration prior to any time

stretching. The actual event durtation in the soundscape is stored in the duration field of the JAMS observation.

- The observations (sound events) in the JAMS annotation include both foreground sound events and the background(s).
- The probabilistic scaper foreground and background event specifications are stored in the annotation's sandbox, allowing

a complete reconstruction of the soundscape audio from the JAMS annotation (assuming access to the original source material) using scaper.generate_from_jams('soundscape_train_bimodal0.jams'). * The annotation sandbox also includes additional metadata such as the total number of foreground sound events, the maximum polyphony (sound event overlap) of the soundscape and its gini coefficient (a measure of soundscape complexity).

Simplified Annotations

- The simplified annotations are distributed as tab-separated text files.
- There are 10,000 simplified annotation files, each one corresponding to a single soundscape with the same filename (other than the extension)
- Each simplified annotation has a 3-column format (no header): start_time, end_time, label.
- Background sounds are NOT included in the simplified annotations (only foreground sound events)
- No additional information is stored in the simplified events (see the JAMS annotations for more details).

Please Acknowledge URBAN-SED in Academic Research

When URBAN-SED is used for academic research, we would highly appreciate it if scientific publications of works partly based on the URBAN-SED dataset cite the following publication:

The creation of this dataset was supported by NSF award 1544753.

Conditions of Use

Dataset created by J. Salamon, D. MacConnell, M. Cartwright, P. Li, and J. P. Bello. Audio files contain excerpts of recordings uploaded to www.freesound.org. Please see FREESOUNDCREDITS.txt for an attribution list.

Feedback

Please help us improve URBAN-SED by sending your feedback to: justin.salamon@nyu.edu
In case of a problem report please include as many details as possible.

`class` soundata.datasets.urbansed.**Clip**(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

URBAN-SED Clip class

Parameters

clip_id (*str*) – id of the clip

Variables

- **audio** (*np.ndarray, float*) – path to the audio file
- **audio_path** (*str*) – path to the audio file
- **clip_id** (*str*) – clip id

- **events** (soundata.annotations.Events) – sound events with start time, end time, label and confidence
- **split** (*str*) – subset the clip belongs to (for experiments): train, validate, or test

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

events

The audio events

Returns

- annotations.Events - audio event object

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

property split

The data splits (e.g. train)

Returns

- str - split

to_jams()

Get the clip's data in jams format

Returns
jams.JAMS – the clip's data in jams format

class soundata.datasets.urbansed.**Dataset**(*data_home=None*)

The URBAN-SED dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

`choice_clipgroup()`

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

`cite()`

Print the reference

`clip_ids`

Return clip ids

Returns
list – A list of clip ids

`clipgroup_ids`

Return clip ids

Returns
list – A list of clip ids

`property default_path`

Get the default path for the dataset

Returns
str – Local path to the dataset

`download(partial_download=None, force_overwrite=False, cleanup=False)`

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- `ValueError` – if invalid keys are passed to partial_download
- `IOError` – if a downloaded file's checksum is different from expected

`explore_dataset(clip_id=None)`

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters
clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

`license()`

Print the license

`load_audio(*args, **kwargs)`

Load a UrbanSound8K audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate of 44100 without resampling.

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

`load_clipgroups()`

Load all clipgroups in the dataset

Returns
dict – {*clipgroup_id*: clipgroup data}

Raises
`NotImplementedError` – If the dataset does not support Clipgroups

`load_clips()`

Load all clips in the dataset

Returns
dict – {*clip_id*: clip data}

Raises
`NotImplementedError` – If the dataset does not support Clips

`validate(verbose=True)`

Validate if the stored dataset is a valid version

Parameters
verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

`soundata.datasets.urbansed.load_audio(fhandle: BinaryIO, sr=None)` → Tuple[numpy.ndarray, float]

Load a UrbanSound8K audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, None by default, which uses the file's original sample rate of 44100 without resampling.

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

`soundata.datasets.urbansed.load_events(fhandle: TextIO)` → *Events*

Load an URBAN-SED sound events annotation file :Parameters: **fhandle** (*str or file-like*) – File-like object or path to the sound events annotation file

Raises
`IOError` – if txt_path doesn't exist

Returns
Events – sound events annotation data

4.5.24 UrbanSound8K

UrbanSound8K Dataset Loader

Dataset Info

Created By:

Justin Salamon*^, Christopher Jacoby* and Juan Pablo Bello*
* Music and Audio Research Lab (MARL), New York University, USA
^ Center for Urban Science and Progress (CUSP), New York University, USA
https://urbansounddataset.weebly.com/
https://steinhardt.nyu.edu/marl
http://cusp.nyu.edu/

Version 1.0

Description:

This dataset contains 8732 labeled sound excerpts (<=4s) of urban sounds from 10 classes: air_conditioner, car_horn, children_playing, dog_bark, drilling, engine_idling, gun_shot, jackhammer, siren, and street_music. The classes are drawn from the urban sound taxonomy described in the following article, which also includes a detailed description of the dataset and how it was compiled:

All excerpts are taken from field recordings uploaded to www.freesound.org. The files are pre-sorted into ten folds (folders named fold1-fold10) to help in the reproduction of and comparison with the automatic classification results reported in the article above.

In addition to the sound excerpts, a CSV file containing metadata about each excerpt is also provided.

Audio Files Included:

8732 audio files of urban sounds (see description above) in WAV format. The sampling rate, bit depth, and number of channels are the same as those of the original file uploaded to Freesound (and hence may vary from file to file).

UrbanSound8k.csv

This file contains meta-data information about every audio file in the dataset. This includes:

- slice_file_name:

The name of the audio file. The name takes the following format: [fsID]-[classID]-[occurrenceID]-[sliceID].wav, where: [fsID] = the Freesound ID of the recording from which this excerpt (slice) is taken [classID] = a numeric identifier of the sound class (see description of classID below for further details) [occurrenceID] = a numeric identifier to distinguish different occurrences of the sound within the original recording [sliceID] = a numeric identifier to distinguish different slices taken from the same occurrence

- fsID:

The Freesound ID of the recording from which this excerpt (slice) is taken

- start

The start time of the slice in the original Freesound recording

- end:

The end time of slice in the original Freesound recording

- salience:

A (subjective) salience rating of the sound. 1 = foreground, 2 = background.

- fold:

The fold number (1-10) to which this file has been allocated.

- classID:

A numeric identifier of the sound class: 0 = air_conditioner 1 = car_horn 2 = children_playing 3 = dog_bark 4 = drilling 5 = engine_idling 6 = gun_shot 7 = jackhammer 8 = siren 9 = street_music

- class:

The class name: air_conditioner, car_horn, children_playing, dog_bark, drilling, engine_idling, gun_shot, jackhammer, siren, street_music.

Please Acknowledge EigenScape in Academic Research:

When UrbanSound8K is used for academic research, we would highly appreciate it if scientific publications of works partly based on the UrbanSound8K dataset cite the following publication:

The creation of this dataset was supported by a seed grant by NYU's Center for Urban Science and Progress (CUSP).

Conditions of Use

Dataset compiled by Justin Salamon, Christopher Jacoby and Juan Pablo Bello. All files are excerpts of recordings uploaded to www.freesound.org. Please see FREESOUNDCREDITS.txt for an attribution list.

The UrbanSound8K dataset is offered free of charge for non-commercial use only under the terms of the Creative Commons Attribution Noncommercial License (by-nc), version 3.0: http://creativecommons.org/licenses/by-nc/3.0/

The dataset and its contents are made available on an "as is" basis and without warranties of any kind, including without limitation satisfactory quality and conformity, merchantability, fitness for a particular purpose, accuracy or completeness, or absence of errors. Subject to any liability that may not be excluded or limited by law, NYU is not liable for, and expressly excludes, all liability for loss or damage however and whenever caused to anyone by any use of the UrbanSound8K dataset or any part of it.

Feedback

Please help us improve UrbanSound8K by sending your feedback to: justin.salamon@nyu.edu
In case of a problem report please include as many details as possible.

---

class soundata.datasets.urbansound8k.**Clip**(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

urbansound8k Clip class

Parameters

clip_id (*str*) – id of the clip

Variables

- **audio** (*np.ndarray, float*) – path to the audio file
- **audio_path** (*str*) – path to the audio file
- **class_id** (*int*) – integer representation of the class label (0-9). See Dataset Info in the documentation for mapping
- **class_label** (*str*) – string class name: air_conditioner, car_horn, children_playing, dog_bark, drilling, engine_idling, gun_shot, jackhammer, siren, street_music
- **clip_id** (*str*) – clip id
- **fold** (*int*) – fold number (1-10) to which this clip is allocated. Use these folds for cross validation

- **freesound_end_time** (*float*) – end time in seconds of the clip in the original freesound recording
- **freesound_id** (*str*) – ID of the freesound.org recording from which this clip was taken
- **freesound_start_time** (*float*) – start time in seconds of the clip in the original freesound recording
- **salience** (*int*) – annotator estimate of class sailence in the clip: 1 = foreground, 2 = background
- **slice_file_name** (*str*) – The name of the audio file. The name takes the following format: [fsID]-[classID]-[occurrenceID]-[sliceID].wav Please see the Dataset Info in the soundata documentation for further details
- **tags** (soundata.annotations.Tags) – tag (label) of the clip + confidence. In UrbanSound8K every clip has one tag

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

property class_id

The clip's class id.

Returns

- int - integer representation of the class label (0-9). See Dataset Info in the documentation for mapping

property class_label

The clip's class label.

Returns

** str - string class name* – air_conditioner, car_horn, children_playing, dog_bark, drilling, engine_idling, gun_shot, jackhammer, siren, street_music

property fold

The clip's fold.

Returns

- int - fold number (1-10) to which this clip is allocated. Use these folds for cross validation

property freesound_end_time

The clip's end time in Freesound.

Returns

- float - end time in seconds of the clip in the original freesound recording

property freesound_id

The clip's Freesound ID.

Returns

- str - ID of the freesound.org recording from which this clip was taken

property freesound_start_time

The clip's start time in Freesound.

Returns

- float - start time in seconds of the clip in the original freesound recording

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters

key (*string*) – Index key of the audio or annotation type

Returns

str or None – joined path string or None

property salience

The clip's salience.

Returns

** int - annotator estimate of class sailence in the clip* – 1 = foreground, 2 = background

property slice_file_name

The clip's slice filename.

Returns

** str - The name of the audio file. The name takes the following format* – [fsID]-[classID]-[occurrenceID]-[sliceID].wav

property tags

The clip's tags.

Returns

- annotations.Tags - tag (label) of the clip + confidence. In UrbanSound8K every clip has one tag

to_jams()

Get the clip's data in jams format

Returns

jams.JAMS – the clip's data in jams format

class soundata.datasets.urbansound8k.Dataset(*data_home=None*)

The urbansound8k dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

cite()

Print the reference

clip_ids

Return clip ids

Returns
list – A list of clip ids

clipgroup_ids

Return clip ids

Returns
list – A list of clip ids

property default_path

Get the default path for the dataset

Returns
str – Local path to the dataset

download(*partial_download=None*, *force_overwrite=False*, *cleanup=False*)

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- **ValueError** – if invalid keys are passed to partial_download
- **IOError** – if a downloaded file's checksum is different from expected

explore_dataset(*clip_id=None*)

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters
clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

license()

Print the license

`load_audio`(**args*, ***kwargs*)

Load a UrbanSound8K audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, 44100 Hz by default. If different from file's sample rate it will be resampled on load. Use None to load the file using its original sample rate (sample rate varies from file to file).

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

`load_clipgroups`()

Load all clipgroups in the dataset

Returns
dict – {*clipgroup_id*: clipgroup data}

Raises
`NotImplementedError` – If the dataset does not support Clipgroups

`load_clips`()

Load all clips in the dataset

Returns
dict – {*clip_id*: clip data}

Raises
`NotImplementedError` – If the dataset does not support Clips

`validate`(*verbose=True*)

Validate if the stored dataset is a valid version

Parameters
verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

`soundata.datasets.urbansound8k.load_audio`(*fhandle: BinaryIO*, *sr=44100*) → Tuple[numpy.ndarray, float]

Load a UrbanSound8K audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, 44100 Hz by default. If different from file's sample rate it will be resampled on load. Use None to load the file using its original sample rate (sample rate varies from file to file).

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

4.5.25 Warblrb10k

Warblrb10k Dataset Loader

Dataset Info

Created By

Dan Stowell*#, Mike Wood†, Yannis Stylianou‡, and Hervé Glotin§

* Machine Listening Lab, Centre for Digital Music, Queen Mary University of London

† Ecosystems and Environment Research Centre, School of Environment and Life Sciences, University of Salford

‡ Computer Science Department, University of Crete

§ LSIS UMR CNRS, University of Toulon, Institut Universitaire de France

Version 1.0

Description

The Warblr dataset consists of 10,000 ten-second audio files, collected via the Warblr app from users across the UK in 2015-2016. Using a classification method by Stowell and Plumbley (2014a), this app aims to identify bird species from user-submitted recordings. The dataset, inclusive of various human and environmental noises, is broadly distributed over different times and seasons but has biases towards mornings, weekends, and populated areas. Despite having initial automated bird species estimates, the recordings underwent manual annotation due to precision inadequacies for establishing ground-truth data. The dataset proves instrumental for research and development in bird species detection amidst variable noise conditions.

Audio Files Included

10,000 ten-second audio recordings in WAV format, amassed through the Warblr app during 2015-2016 from users throughout the UK.

Meta-data Files Included

A table containing a binary label "hasbird" associated to every recording in Warblr is available on the website of the DCASE "Bird Audio Detection" challenge: http://machine-listening.eecs.qmul.ac.uk/bird-audio-detection-challenge/

Please Acknowledge Warblr in Academic Research

When the Warblr dataset is employed for academic research, we sincerely request that scientific publications of works partially based on this dataset cite the following publication:

- Stowell, Dan and Wood, Michael and Pamuła, Hanna and Stylianou, Yannis and Glotin, Hervé. "Automatic acoustic detection of birds through deep learning: The first Bird Audio Detection challenge", Methods in Ecology and Evolution, 2018.

The creation and curating of this dataset were possible through the participation and contributions of the general public using the Warblr app, enabling a comprehensive collection of bird sound recordings from various regions within the UK during 2015-2016.

Conditions of Use

Dataset created by [Creators/Researchers involved].

class soundata.datasets.warblrb10k.Clip(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

warblrb10k Clip class

Parameters

clip_id (*str*) – id of the clip

Variables

- **audio** (*np.ndarray, float*) – path to the audio file
- **audio_path** (*str*) – path to the audio file
- **item_id** (*str*) – clip id
- **has_bird** (*str*) – indication of whether the clips contains bird sounds (0/1)

property audio: Optional[Tuple[numpy.ndarray, float]]

The clip's audio

Returns

- np.ndarray - audio signal
- float - sample rate

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters

key (*string*) – Index key of the audio or annotation type

Returns

str or None – joined path string or None

property has_bird

The flag to tell whether the clip has bird sound or not.

Returns

- str - 1/0 depending on whether the clip contains bird sound

property item_id

The clip's item ID.

Returns

- str - ID of the clip

to_jams()

Get the clip's data in jams format

Returns

jams.JAMS – the clip's data in jams format

class soundata.datasets.warblrb10k.Dataset(*data_home=None*)

The Warblrb10k dataset

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded

- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

choice_clip()

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

cite()

Print the reference

clip_ids

Return clip ids

Returns
list – A list of clip ids

clipgroup_ids

Return clip ids

Returns
list – A list of clip ids

property default_path

Get the default path for the dataset

Returns
str – Local path to the dataset

download(*partial_download=None*, *force_overwrite=False*, *cleanup=False*)

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- **ValueError** – if invalid keys are passed to partial_download
- **IOError** – if a downloaded file's checksum is different from expected

explore_dataset(*clip_id=None*)

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters
clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

`license()`

Print the license

`load_audio(*args, **kwargs)`

Load a Warblrb10k audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, 44100 Hz by default. If different from file's sample rate it will be resampled on load. Use None to load the file using its original sample rate (sample rate varies from file to file).

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

`load_clipgroups()`

Load all clipgroups in the dataset

Returns
dict – {*clipgroup_id*: clipgroup data}

Raises
`NotImplementedError` – If the dataset does not support Clipgroups

`load_clips()`

Load all clips in the dataset

Returns
dict – {*clip_id*: clip data}

Raises
`NotImplementedError` – If the dataset does not support Clips

`validate(verbose=True)`

Validate if the stored dataset is a valid version

Parameters
verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

`soundata.datasets.warblrb10k.load_audio(fhandle: BinaryIO, sr=44100)` → Tuple[numpy.ndarray, float]

Load a Warblrb10k audio file.

Parameters

- **fhandle** (*str or file-like*) – File-like object or path to audio file
- **sr** (*int or None*) – sample rate for loaded audio, 44100 Hz by default. If different from file's sample rate it will be resampled on load. Use None to load the file using its original sample rate (sample rate varies from file to file).

Returns

- np.ndarray - the mono audio signal
- float - The sample rate of the audio file

4.6 Core

Core soundata classes

class soundata.core.**Clip**(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

Clip base class

See the docs for each dataset loader's Clip class for details

__init__(*clip_id*, *data_home*, *dataset_name*, *index*, *metadata*)

Clip init method. Sets boilerplate attributes, including:

- `clip_id`
- `_dataset_name`
- `_data_home`
- `_clip_paths`
- `_clip_metadata`

Parameters

- **clip_id** (*str*) – clip id
- **data_home** (*str*) – path where soundata will look for the dataset
- **dataset_name** (*str*) – the identifier of the dataset
- **index** (*dict*) – the dataset's file index
- **metadata** (*function or None*) – a function returning a dictionary of metadata or None

get_path(*key*)

Get absolute path to clip audio and annotations. Returns None if the path in the index is None

Parameters
key (*string*) – Index key of the audio or annotation type

Returns
str or None – joined path string or None

class soundata.core.**ClipGroup**(*clipgroup_id*, *data_home*, *dataset_name*, *index*, *clip_class*, *metadata*)

ClipGroup class.

A clipgroup class is a collection of clip objects and their associated audio that can be mixed together. A clipgroup is itself a Clip, and can have its own associated audio (such as a mastered mix), its own metadata and its own annotations.

__init__(*clipgroup_id*, *data_home*, *dataset_name*, *index*, *clip_class*, *metadata*)

Clipgroup init method. Sets boilerplate attributes, including:

- `clipgroup_id`

- _dataset_name
- _data_home
- _clipgroup_paths
- _clipgroup_metadata

Parameters

- **clipgroup_id** (*str*) – clipgroup id
- **data_home** (*str*) – path where soundata will look for the dataset
- **dataset_name** (*str*) – the identifier of the dataset
- **index** (*dict*) – the dataset's file index
- **metadata** (*function or None*) – a function returning a dictionary of metadata or None

property clip_audio_property

The clip's audio property.

Returns:

get_mix()

Create a linear mixture given a subset of clips.

Parameters

clip_keys (*list*) – list of clip keys to mix together

Returns

np.ndarray – mixture audio with shape (n_samples, n_channels)

get_path(*key*)

Get absolute path to clipgroup audio and annotations. Returns None if the path in the index is None

Parameters

key (*string*) – Index key of the audio or annotation type

Returns

str or None – joined path string or None

get_random_target(*n_clips=None*, *min_weight=0.3*, *max_weight=1.0*)

Get a random target by combining a random selection of clips with random weights

Parameters

- **n_clips** (*int or None*) – number of clips to randomly mix. If None, uses all clips
- **min_weight** (*float*) – minimum possible weight when mixing
- **max_weight** (*float*) – maximum possible weight when mixing

Returns

- np.ndarray - mixture audio with shape (n_samples, n_channels)
- list - list of keys of included clips
- list - list of weights used to mix clips

get_target(*clip_keys*, *weights=None*, *average=True*, *enforce_length=True*)

Get target which is a linear mixture of clips

Parameters

- **clip_keys** (*list*) – list of clip keys to mix together
- **weights** (*list or None*) – list of positive scalars to be used in the average
- **average** (*bool*) – if True, computes a weighted average of the clips if False, computes a weighted sum of the clips
- **enforce_length** (*bool*) – If True, raises ValueError if the clips are not the same length. If False, pads audio with zeros to match the length of the longest clip

Returns

np.ndarray – target audio with shape (n_channels, n_samples)

Raises

ValueError – if sample rates of the clips are not equal if enforce_length=True and lengths are not equal

class soundata.core.**Dataset**(*data_home=None*, *name=None*, *clip_class=None*, *clipgroup_class=None*, *bibtex=None*, *remotes=None*, *download_info=None*, *license_info=None*, *custom_index_path=None*)

soundata Dataset class

Variables

- **data_home** (*str*) – path where soundata will look for the dataset
- **name** (*str*) – the identifier of the dataset
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **readme** (*str*) – information about the dataset
- **clip** (*function*) – a function mapping a clip_id to a soundata.core.Clip
- **clipgroup** (*function*) – a function mapping a clipgroup_id to a soundata.core.Clipgroup

__init__(*data_home=None*, *name=None*, *clip_class=None*, *clipgroup_class=None*, *bibtex=None*, *remotes=None*, *download_info=None*, *license_info=None*, *custom_index_path=None*)

Dataset init method

Parameters

- **data_home** (*str or None*) – path where soundata will look for the dataset
- **name** (*str or None*) – the identifier of the dataset
- **clip_class** (*soundata.core.Clip or None*) – a Clip class
- **clipgroup_class** (*soundata.core.Clipgroup or None*) – a Clipgroup class
- **bibtex** (*str or None*) – dataset citation/s in bibtex format
- **remotes** (*dict or None*) – data to be downloaded
- **download_info** (*str or None*) – download instructions or caveats
- **license_info** (*str or None*) – license of the dataset
- **custom_index_path** (*str or None*) – overwrites the default index path for remote indexes

choice_clip()

Choose a random clip

Returns
Clip – a Clip object instantiated by a random clip_id

choice_clipgroup()

Choose a random clipgroup

Returns
Clipgroup – a Clipgroup object instantiated by a random clipgroup_id

cite()

Print the reference

clip_ids

Return clip ids

Returns
list – A list of clip ids

clipgroup_ids

Return clip ids

Returns
list – A list of clip ids

property default_path

Get the default path for the dataset

Returns
str – Local path to the dataset

download(*partial_download=None*, *force_overwrite=False*, *cleanup=False*)

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

Parameters

- **partial_download** (*list or None*) – A list of keys of remotes to partially download. If None, all data is downloaded
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete any zip/tar files after extracting.

Raises

- **ValueError** – if invalid keys are passed to partial_download
- **IOError** – if a downloaded file's checksum is different from expected

explore_dataset(*clip_id=None*)

Explore the dataset for a given clip_id or a random clip if clip_id is None.

Parameters
clip_id (*str or None*) – The identifier of the clip to explore. If None, a random clip will be chosen.

license()

Print the license

load_clipgroups()

Load all clipgroups in the dataset

Returns
dict – {*clipgroup_id*: clipgroup data}

Raises
NotImplementedError – If the dataset does not support Clipgroups

load_clips()

Load all clips in the dataset

Returns
dict – {*clip_id*: clip data}

Raises
NotImplementedError – If the dataset does not support Clips

validate(*verbose=True*)

Validate if the stored dataset is a valid version

Parameters
verbose (*bool*) – If False, don't print output

Returns

- list - files in the index but are missing locally
- list - files which have an invalid checksum

class soundata.core.**cached_property**(*func*)

Cached propery decorator

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

soundata.core.**copy_docs**(*original*)

Decorator function to copy docs from one function to another

soundata.core.**docstring_inherit**(*parent*)

Decorator function to inherit docstrings from the parent class.

Adds documented Attributes from the parent to the child docs.

4.7 Annotations

soundata annotation data types

soundata.annotations.**AZIMUTH_UNITS = {'degrees': 'values in the interval [-360, 360]', 'radians': 'values in the interval [-2*pi, 2*pi]'}**

Azimuth units

class soundata.annotations.**Annotation**

Annotation base class

soundata.annotations.**DISTANCE_UNITS = {'centimeters': 'centimeters', 'meters': 'meters', 'millimeters': 'millimeters'}**

Distance units

soundata.annotations.**ELEVATIONS_UNITS = {'degrees': 'degrees'}**

position units

class soundata.annotations.**Events**(*intervals*, *intervals_unit*, *labels*, *labels_unit*, *confidence=None*, *azimuth=None*, *azimuth_unit=None*, *elevation=None*, *elevation_unit=None*, *distance=None*, *distance_unit=None*, *cartesian_coord=None*, *cartesian_coord_unit=None*)

Events class

Variables

- **intervals** (*np.ndarray*) – (n x 2) array of intervals (as floats) in seconds in the form [start_time, end_time] with positive time stamps and end_time >= start_time.
- **labels** (*list*) – list of event labels (as strings)
- **confidence** (*np.ndarray or None*) – array of confidence values, float in [0, 1]
- **labels_unit** (*str*) – labels unit, one of LABELS_UNITS
- **intervals_unit** (*str*) – intervals unit, one of TIME_UNITS
- **azimuth** (*np.ndarray or None*) – list of size n with np.ndarrays with dtype float, indicating the azimuth of the sound event. Values between -360 and 360 for degrees and between -2*pi, 2*pi for radians or None.
- **azimuth_unit** (*str*) – azimuth unit, one of AZIMUTH_UNITS
- **elevation** (*np.ndarray or None*) – list of size n with np.ndarrays with dtype float, indicating the elevation of the sound event. Values between -90 and 90 or None.
- **elevation_unit** (*str*) – elevation unit, one of AZIMUTH_UNITS
- **distance** (*np.ndarray or None*) – list of size n with np.ndarrays with dtype float, indicating the distance of the sound event. Values must be positive or None.
- **distance_unit** (*str*) – distance unit, one of DISTANCE_UNITS
- **cartesian_coord** (*np.ndarray or None*) –
- **cartesian_coord_unit** (*str*) – cartesian_coord unit, one of DISTANCE_UNITS

soundata.annotations.**LABEL_UNITS = {'open': 'no strict schema or units'}**

Label units

class soundata.annotations.**MultiAnnotator**(*annotators*, *annotations*)

Multiple annotator class. This class should be used for datasets with multiple annotators (e.g. multiple annotators per clip).

Variables

- **annotators** (*list*) – list with annotator ids
- **annotations** (*list*) – list of annotations (e.g. [annotations.Tags, annotations.Tags]

class soundata.annotations.**SpatialEvents**(*intervals*, *intervals_unit*, *elevations*, *elevations_unit*, *azimuths*, *azimuths_unit*, *distances*, *distances_unit*, *labels*, *labels_unit*, *clip_number_index=None*, *time_step=None*, *confidence=None*)

SpatialEvents class :ivar intervals: list of size n np.ndarrays of shape (m, 2), with intervals (as floats) in TIME_UNITS in the form [start_time, end_time] with positive time stamps and end_time >= start_time. n is the number of sound events. m is the number of sounding instances for each sound event.

Variables

- **intervals_unit** (*str*) – intervals unit, one of TIME_UNITS
- **time_step** (*int, float, or None*) – the time-step between events over time in intervals_unit
- **elevations** (*list*) – list of size n with np.ndarrays with dtype int, indicating the elevation of the sound event per time_step if moving or a single value if static. Values between -90 and 90
- **elevations_unit** (*str*) – elevations unit, one of ELEVATIONS_UNITS
- **azimuths** (*list*) – list of size n with np.ndarrays with dtype int, indicating the azimuth of the sound event per time_step if moving or a single value if static. Values between -180 and 180
- **azimuths_unit** (*str*) – azimuths unit, one of AZIMUTHS_UNITS
- **distances** (*list*) – list of size n with np.ndarrays with dtype int, indicating the distance of the sound event per time_step if moving or a single value if static. Values must be positive or None
- **distances_unit** (*str*) – distances unit, one of DISTANCES_UNITS
- **labels** (*list*) – list of event labels (as strings)
- **labels_unit** (*str*) – labels unit, one of LABELS_UNITS
- **clip_number_indices** (*list*) – list of clip number indices (as strings)
- **confidence** (*np.ndarray or None*) – array of confidence values, float in [0, 1]

soundata.annotations.**TIME_UNITS = {'milliseconds': 'milliseconds', 'seconds': 'seconds'}**

Time units

class soundata.annotations.**Tags**(*labels*, *labels_unit*, *confidence=None*)

Tags class

Variables

- **labels** (*list*) – list of string tags
- **confidence** (*np.ndarray or None*) – array of confidence values, float in [0, 1]
- **labels_unit** (*str*) – labels unit, one of LABELS_UNITS

soundata.annotations.**validate_array_like**(*array_like*, *expected_type*, *expected_dtype*, *check_child=False*, *none_allowed=False*)

Validate that array-like object is well formed If array_like is None, validation passes automatically. :Parameters: * **array_like** (*array-like*) – object to validate

- **expected_type** (*type*) – expected type, either list or np.ndarray
- **expected_dtype** (*type*) – expected dtype
- **check_child** (*bool*) – if True, checks if all elements of array are children of expected_dtype
- **none_allowed** (*bool*) – if True, allows array to be None

Raises

- **TypeError** – if type/dtype does not match expected_type/expected_dtype
- **ValueError** – if array

soundata.annotations.**validate_confidence**(*confidence*)

Validate if confidence is well-formed.

If confidence is None, validation passes automatically

Parameters
confidence (*np.ndarray*) – an array of confidence values

Raises
ValueError – if confidence are not between 0 and 1

soundata.annotations.**validate_intervals**(*intervals*)

Validate if intervals are well-formed.

If intervals is None, validation passes automatically
Parameters
intervals (*np.ndarray*) – (n x 2) array

Raises

- **ValueError** – if intervals have an invalid shape, have negative values
- **or if end times are smaller than start times.** –

soundata.annotations.**validate_lengths_equal**(*array_list*)

Validate that arrays in list are equal in length

Some arrays may be None, and the validation for these are skipped.
Parameters
array_list (*list*) – list of array-like objects

Raises
ValueError – if arrays are not equal in length

soundata.annotations.**validate_locations**(*locations*)

Validate if locations are well-formed. If locations is None, validation passes automatically :Parameters: **locations** (*np.ndarray*) – (n x 3) array
Raises
ValueError – if locations have an invalid shape or have cartesian coordinate values outside the expected ranges.

soundata.annotations.**validate_time_steps**(*time_step*, *locations*, *interval*)

Validate if timesteps are well-formed. If locations is None, validation passes automatically :Parameters: * **time_step** (*float*) – spacing between location steps
- **locations** (*np.ndarray*) – (n x 3) array
- **interval** (*np.ndarray*) – (n x 2) expected start and end time for the locations

Raises
ValueError – if the number of locations does not match the number of time_steps that fit in the interval

soundata.annotations.**validate_times**(*times*)

Validate if times are well-formed.

If times is None, validation passes automatically
Parameters
times (*np.ndarray*) – an array of time stamps

Raises
ValueError – if times have negative values or are non-increasing

soundata.annotations.**validate_unit**(*unit*, *unit_values*, *allow_none=False*)

Validate that the given unit is one of the allowed unit values. :Parameters: * **unit** (*str*) – the unit name
- **unit_values** (*dict*) – dictionary of possible unit values
- **allow_none** (*bool*) – if true, allows unit=None to pass validation

Raises

ValueError – If the given unit is not one of the allowed unit values

4.8 Advanced

4.8.1 soundata.validate

Utility functions for soundata

soundata.validate.**log_message**(*message*, *verbose=True*)

Helper function to log message

Parameters

- **message** (*str*) – message to log
- **verbose** (*bool*) – if false, the message is not logged

soundata.validate.**md5**(*file_path*)

Get md5 hash of a file.

Parameters

file_path (*str*) – File path

Returns

str – md5 hash of data in file_path

soundata.validate.**validate**(*local_path*, *checksum*)

Validate that a file exists and has the correct checksum

Parameters

- **local_path** (*str*) – file path
- **checksum** (*str*) – md5 checksum

Returns

- bool - True if file exists
- bool - True if checksum matches

soundata.validate.**validate_files**(*file_dict*, *data_home*, *verbose*)

Validate files

Parameters

- **file_dict** (*dict*) – dictionary of file information
- **data_home** (*str*) – path where the data lives
- **verbose** (*bool*) – if True, show progress

Returns

- dict - missing files
- dict - files with invalid checksums

soundata.validate.**validate_index**(*dataset_index*, *data_home*, *verbose=True*)

Validate files in a dataset's index

Parameters

- **dataset_index** (*list*) – dataset indices

- **data_home** (*str*) – Local home path that the dataset is being stored
- **verbose** (*bool*) – if true, prints validation status while running

Returns

- dict - file paths that are in the index but missing locally
- dict - file paths with differing checksums

soundata.validate.**validate_metadata**(*file_dict*, *data_home*, *verbose*)

Validate files

Parameters

- **file_dict** (*dict*) – dictionary of file information
- **data_home** (*str*) – path where the data lives
- **verbose** (*bool*) – if True, show progress

Returns

- dict - missing files
- dict - files with invalid checksums

soundata.validate.**validator**(*dataset_index*, *data_home*, *verbose=True*)

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
- **verbose** (*bool*) – if True (default), prints missing and invalid files to stdout. Otherwise, this function is equivalent to validate_index.

Returns

missing_files (list) –

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.

4.8.2 soundata.download_utils

utilities for downloading from the web.

class soundata.download_utils.**DownloadProgressBar**(*_, **__)

Wrap *tqdm* to show download progress

class soundata.download_utils.**RemoteFileMetadata**(*filename*, *url*, *checksum*, *destination_dir=None*, *unpack_directories=None*)

The metadata for a remote file

Variables

- **filename** (*str*) – the remote file's basename
- **url** (*str*) – the remote file's url

- **checksum** (*str*) – the remote file's md5 checksum
- **destination_dir** (*str or None*) – the relative path for where to save the file
- **unpack_directories** (*list or None*) – list of relative directories. For each directory the contents will be moved to destination_dir (or data_home if not provided)

soundata.download_utils.**download_7z_file**(*tar_remote*, *save_dir*, *force_overwrite*, *cleanup*)

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

soundata.download_utils.**download_from_remote**(*remote*, *save_dir*, *force_overwrite*)

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

str – Full path of the created file.

soundata.download_utils.**download_multipart_zip**(*zip_remotes*, *save_dir*, *force_overwrite*, *cleanup*)

Download and unzip a multipart zip file.

Parameters

- **zip_remotes** (*list*) – A list of RemoteFileMetadata Objects 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

soundata.download_utils.**download_tar_file**(*tar_remote*, *save_dir*, *force_overwrite*, *cleanup*)

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

soundata.download_utils.**download_zip_file**(*zip_remote*, *save_dir*, *force_overwrite*, *cleanup*)

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

soundata.download_utils.**downloader**(*save_dir*, *remotes=None*, *partial_download=None*, *info_message=None*, *force_overwrite=False*, *cleanup=False*)

Download data to *save_dir* and optionally log 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 an element of the dictionary is a list of RemoteFileMetadata,

  it is handled as a multipart zip file

  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 log when this function is called. If None, no string is logged.
- **force_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

soundata.download_utils.**extractall_unicode**(*zfile*, *out_dir*)

Extract all files inside a zip archive to a output directory.

In comparison to the zipfile, it checks for correct file name encoding

Parameters

- **zfile** (*obj*) – Zip file object created with zipfile.ZipFile
- **out_dir** (*str*) – Output folder

soundata.download_utils.**move_directory_contents**(*source_dir*, *target_dir*)

Move the contents of source_dir into target_dir, and delete source_dir

Parameters

- **source_dir** (*str*) – path to source directory
- **target_dir** (*str*) – path to target directory

soundata.download_utils.**un7z**(*sevenz_path*, *cleanup*)

Unzip a 7z file inside its current directory.

Parameters

- **sevenz_path** (*str*) – Path to the 7z file
- **cleanup** (*bool*) – If True, remove 7z file after extraction

soundata.download_utils.**untar**(*tar_path*, *cleanup*)

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

soundata.download_utils.**unzip**(*zip_path*, *cleanup*)

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

4.8.3 soundata.jams_utils

Utilities for converting soundata Annotation classes to jams format.

soundata.jams_utils.**events_to_jams**(*events*, *annotator=None*, *description=None*)

Convert events annotations into jams format.

Parameters

- **events** (*annotations.Events*) – events data object
- **annotator** (*str*) – annotator id
- **description** (*str*) – annotation description

Returns

jams.Annotation – jams annotation object.

soundata.jams_utils.**jams_converter**(*audio_path=None*, *spectrogram_path=None*, *metadata=None*, *tags=None*, *events=None*)

Convert annotations from a clip to JAMS format.

Parameters

- **audio_path** (*str or None*) – 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.
- **spectrogram_path** (*str or None*) – A path to the corresponding spectrum file, or None.
- **tags** (*annotations.Tags or annotations.MultiAnnotator or None*) – An instance of annotations.Tags/annotations.MultiAnnotator describing the audio tags.
- **events** (*annotations.Events or annotations.MultiAnnotator or None*) – An instance of annotations.Events/annotations.MultiAnnotator describing the sound events.

Returns

jams.JAMS – A JAMS object containing the annotations.

soundata.jams_utils.**multiannotator_to_jams**(*multiannot: MultiAnnotator*, *converter: Callable[[...], Annotation]*, ***kwargs*) → List[jams.Annotation]

Convert tags annotations into jams format.

Parameters

- **tags** (*annotations.MultiAnnotator*) – MultiAnnotator object

- **converter** (*Callable[..., annotations.Annotation]*) – a function that takes an annotation object, its annotator, (and other optional arguments), and return a jams annotation object

Returns

List[jams.Annotation] – List of jams annotation objects.

soundata.jams_utils.**tags_to_jams**(*tags*, *annotator=None*, *duration=0*, *namespace='tag_open'*, *description=None*)

Convert tags annotations into jams format.

Parameters

- **tags** (*annotations.Tags*) – tags annotation object
- **annotator** (*str*) – annotator id
- **namespace** (*str*) – the jams-compatible tag namespace
- **description** (*str*) – annotation description

Returns

jams.Annotation – jams annotation object.

4.9 Changelog

The list of all changes in Soundata releases can be found here.

4.10 FAQ

How do I add a new loader?

Take a look at our *Contributing* docs!

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.

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

Sorry, we do not host or distribute datasets.

What is the canonical version of a loader?

The `canonical` version of a loader is the source version of a dataset, i.e. the version that you get directly from the creators of the dataset or similar oficial source.

How do I request a new dataset?

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

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 soundata 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.

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.

Does soundata provide data loaders for pytorch/Tensorflow?

Not yet, but we plan to include this functionality soon. For now, check the examples of Tensorflow generators in *Getting started*

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.

Why the name, soundata?

soundata = sound + data, and the library was built for working with audio data.

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

Please do not. 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. If you are in doubt, open an issue and discuss it with the community.

Does soundata support data which lives off-disk?

Yes. While the simple useage of soundata assumes that data lives on-disk, it can be used for off-disk data as well. See *Accessing data remotely* for details.

PYTHON MODULE INDEX

S

INDEX

Symbols

A

C

D

E

F

G

H

I

J

K

L

M

N

P

R

S

T

U

V