audiomate Documentation

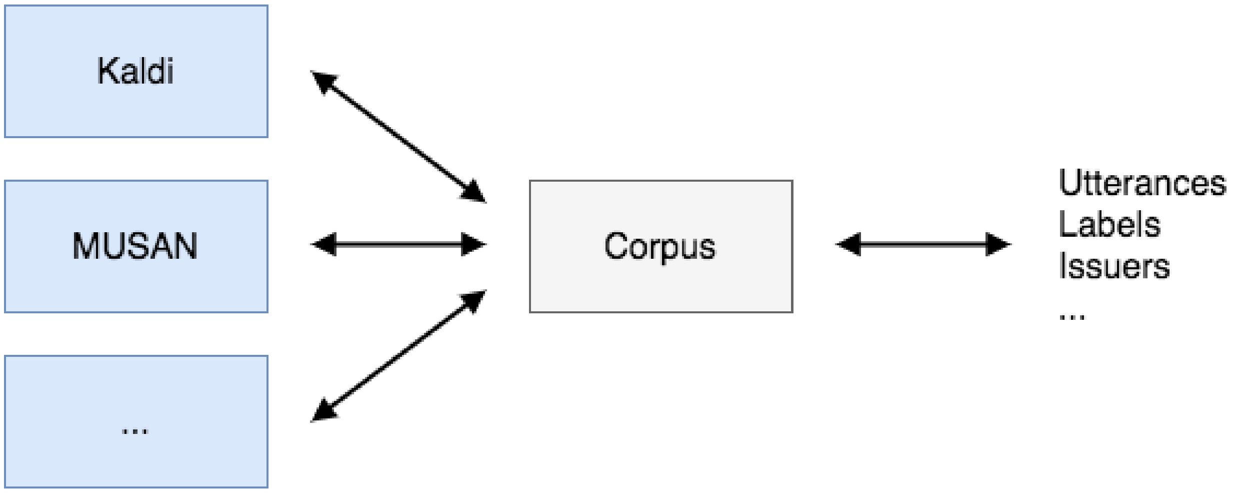
Release 6.0.0

buec

Aug 05, 2020

Notes

Audiomate is a library for easy access to audio datasets. It provides the datastructures for accessing/loading different datasets in a generic way. This should ease the use of audio datasets for example for machine learning tasks.



CHAPTER 1

Installation

Install the latest stable version:

```
pip install audiomate
```

Install the latest development version:

```
pip install git+https://github.com/ynop/audiomate.git
```

1.1 Dependencies

sox

For parts of the functionality (e.g. audio format conversion) sox is used. In order to use it, you have to install sox.

```
# macos
brew install sox

# with support for specific formats
brew install sox --with-lame --with-flac --with-libvorbis

# linux
apt-get install sox

# anaconda for macOS/windows/linux:
conda install -c conda-forge sox
```

CHAPTER 2

Changelog

2.1 Next Version

2.2 v6.0.0

Breaking Changes

- Drop support of Python 3.5 because a required dependency (llvmlite) does not support it anymore.

New Features

- Setup consistent way for logging. (*Logging*)
- Added downloader (`audiomate.corpus.io.CommonVoiceDownloader`) for the Common Voice Corpora.
- Add existence checks for reader (`audiomate.corpus.io.CorpusReader`) to see if folder exists.
- Add existence checks and a option for forcing redownload for downloader (`audiomate.corpus.io.CorpusDownloader`).

2.3 v5.2.0

New Features

- Added reader (`audiomate.corpus.io.LibriSpeechReader`) and downloader (`audiomate.corpus.io.LibriSpeechDownloader`) for the LibriSpeech Dataset.

2.4 v5.1.0

New Features

- Added Downloader for SWC Corpus ((`audiomate.corpus.io.SWCDownloader`).
- Updated SWC-Reader (`audiomate.corpus.io.SWCReader`) with an own implementation, so no manual preprocessing is needed anymore.
- Added conversion class (*`audiomate.corpus.conversion.WavAudioFileConverter`*) to convert all files (or files that do not meet the requirements) of a corpus.
- Added writer (`audiomate.corpus.io.NvidiaJasperWriter`) for NVIDIA Jasper.
- Create a consistent way to define invalid utterances of a dataset. Invalid utterance ids are defined in a json-file (e.g. `audiomate/corpus/io/data/tuda/invalid_utterances.json`). Those are loaded automatically in the base-reader and can be accessed in the concrete implementation.

2.5 v5.0.0

Breaking Changes

- Changed *`audiomate.corpus.validation.InvalidItemsResult`* to use it not only for Utterances, but also for Tracks for example.
- Refactoring and addition of splitting functions in the *`audiomate.corpus.subset.Splitter`*.

New Features

- Added *`audiomate.corpus.validation.TrackReadValidator`* to check for corrupt audio tracks/files.
- Added reader (`audiomate.corpus.io.FluentSpeechReader`) for the Fluent Speech Commands Dataset.
- Added functions to check for contained tracks and issuers (*`audiomate.corpus.CorpusView.contains_track()`*, *`audiomate.corpus.CorpusView.contains_issuer()`*).
- Multiple options for controlling the behavior of the `audiomate.corpus.io.KaldiWriter`.
- Added writer (`audiomate.corpus.io.Wav2LetterWriter`) for the wav2letter engine.
- Added module with functions to read/write sclite trn files (*`audiomate.formats.trn`*).

Fixes

- Improved performance of Tuda-Reader (`audiomate.corpus.io.TudaReader`).
- Added wrapper for the `` `audioread.audio_open` `` function (*`audiomate.utils.audioread`*) to cache available backends. This speeds up audioopen operations a lot.
- Performance improvements, especially for importing utterances, merging, subviews.

2.6 v4.0.1

Fixes

- Fix `audiomate.corpus.io.CommonVoiceReader` to use correct file-extension of the audio files.

2.7 v4.0.0

Breaking Changes

- For utterances and labels `-1` was used for representing that the end is the same as the end of the parent utterance/track. In order to prevent `-1` checks in different methods/places `float('inf')` is now used. This makes it easier to implement stuff like label overlapping.
- *`audiomate.annotations.LabelList`* is now backed by an interval-tree instead of a simple list. Therefore the labels have no fixed order anymore. The interval-tree provides functionality for operations like merging, splitting, finding overlaps with much lower code complexity.
- Removed module `audiomate.annotations.label_cleaning`, since those methods are available on *`audiomate.annotations.LabelList`* directly.

New Features

- Added reader (`audiomate.corpus.io.RouenReader`) and downloader (`audiomate.corpus.io.RouenDownloader`) for the LITIS Rouen Audio scene dataset.
- Added downloader (`audiomate.corpus.io.AEDDownloader`) for the Acoustic Event Dataset.
- [#69] Method to get labels within range: *`audiomate.annotations.LabelList.labels_in_range()`*.
- [#68] Add convenience method to create Label-List with list of label values: *`audiomate.annotations.LabelList.with_label_values()`*.
- [#61] Added function to split utterances of a corpus into multiple utterances with a maximal duration: *`audiomate.corpus.CorpusView.split_utterances_to_max_time()`*.
- Add functions to check for overlap between labels: *`audiomate.annotations.Label.do_overlap()`* and *`audiomate.annotations.Label.overlap_duration()`*.
- Add function to merge equal labels that overlap within a label-list: `audiomate.annotations.LabelList.merge_overlapping_labels()`.
- Added reader (`audiomate.corpus.io.AudioMNISTReader`) and downloader (`audiomate.corpus.io.AudioMNISTDownloader`) for the AudioMNIST dataset.

Fixes

- [#76][#77][#78] Multiple fixes on KaldiWriter

2.8 v3.0.0

Breaking Changes

- Moved label-encoding to its own module (*`audiomate.encoding`*). It now provides the processing of full corpora and store it in containers.
- Moved *`audiomate.feeding.PartitioningFeatureIterator`* to the *`audiomate.feeding`* module.
- Added *`audiomate.containers.AudioContainer`* to store audio tracks in a single file. All container classes are now in a separate module *`audiomate.containers`*.
- Corpus now contains Tracks not Files anymore. This makes it possible to different kinds of audio sources. Audio from a file is now included using *`audiomate.tracks.FileTrack`*. New is the *`audiomate.tracks.ContainerTrack`*, which reads data stored in a container.

- The audiomate.corpus.io.DefaultReader and the audiomate.corpus.io.DefaultWriter now load and store tracks, that are stored in a container.
- All functionality regarding labels was moved to its own module *audiomate.annotations*.
- The class *audiomate.tracks.Utterance* was moved to the tracks module.

New Features

- Introducing the *audiomate.feeding* module. It provides different tools for accessing container data. Via a *audiomate.feeding.Dataset* data can be accessed by indices. With a *audiomate.feeding.DataIterator* one can easily iterate over data, such as frames.
- Added processing steps for computing Onset-Strength (*audiomate.processing.pipeline.OnsetStrength*)) and Tempogram (*audiomate.processing.pipeline.Tempogram*)).
- Introduced *audiomate.corpus.validation* module, that is used to validate a corpus.
- Added reader (audiomate.corpus.io.SWCReader) for the SWC corpus. But it only works for the prepared corpus.
- Added function (audiomate.corpus.utils.label_cleaning.merge_consecutive_labels_with_same_va for merging consecutive labels with the same value
- Added downloader (audiomate.corpus.io.GtzanDownloader) for the GTZAN Music/Speech.
- Added audiomate.corpus.assets.Label.tokenized() to get a list of tokens from a label. It basically splits the value and trims whitespace.
- Added methods on *audiomate.corpus.CorpusView*, audiomate.corpus.assets.Utterance and audiomate.corpus.assets.LabelList to get a set of occurring tokens.
- Added *audiomate.encoding.TokenOrdinalEncoder* to encode labels of an utterance by mapping every token of the label to a number.
- Create container base class (audiomate.corpus.assets.Container), that can be used to store arbitrary data per utterance. The audiomate.corpus.assets.FeatureContainer is now an extension of the container, that provides functionality especially for features.
- Added functions to split utterances and label-lists into multiple parts. (audiomate.corpus.assets.Utterance.split(), audiomate.corpus.assets.LabelList.split())
- Added *audiomate.processing.pipeline.AddContext* to add context to frames, using previous and subsequent frames.
- Added reader (audiomate.corpus.io.MailabsReader) and downloader (audiomate.corpus.io.MailabsDownloader) for the M-AILABS Speech Dataset.

Fixes

- [#58] Keep track of number of samples per frame and between frames. Now the correct values will be stored in a Feature-Container, if the processor implements it correctly.
- [#72] Fix bug, when reading samples from utterance, using a specific duration, while the utterance end is not defined.

2.9 v2.0.0

Breaking Changes

- Update various readers to use the correct label-list identifiers as defined in *Data Mapping*.

New Features

- Added downloader (`audiomate.corpus.io.TatoebaDownloader`) and reader (`audiomate.corpus.io.TatoebaReader`) for the Tatoeba platform.
- Added downloader (`audiomate.corpus.io.CommonVoiceDownloader`) and reader (`audiomate.corpus.io.CommonVoiceReader`) for the Common Voice Corpus.
- Added processing steps *`audiomate.processing.pipeline.AvgPool`* and *`audiomate.processing.pipeline.VarPool`* for computing average and variance over a given number of sequential frames.
- Added downloader (`audiomate.corpus.io.MusanDownloader`) for the Musan Corpus.
- Added constants for common label-list identifiers/keys in *`audiomate.corpus`*.

2.10 v1.0.0

Breaking Changes

- The (pre)processing module has moved to *`audiomate.processing`*. It now supports online processing in chunks. For this purpose a pipeline step can require context. The pipeline automatically buffers data, until enough frames are ready.

New Features

- Added downloader (`audiomate.corpus.io.FreeSpokenDigitDownloader`) and reader (`audiomate.corpus.io.FreeSpokenDigitReader`) for the Free-Spoken-Digit-Dataset.

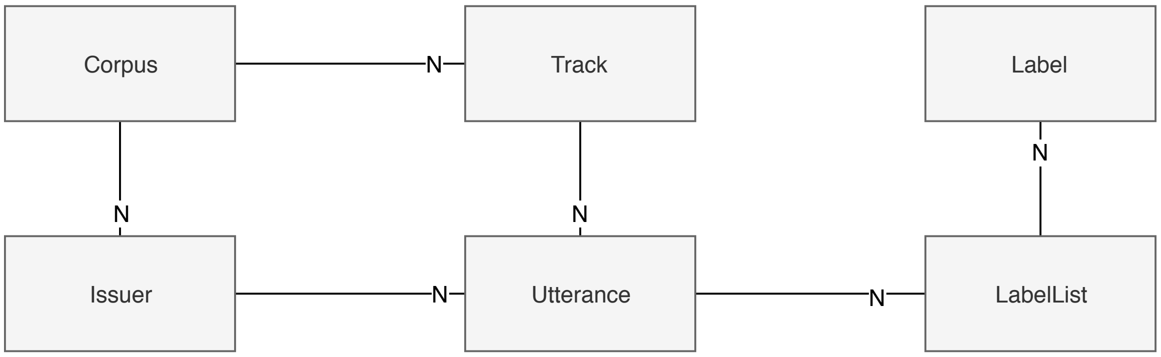
2.11 v0.1.0

Initial release

CHAPTER 3

Corpus Structure

To represent any corpus/dataset in a generic way, a structure is needed that can represent the data of any audio dataset as good as possible. The basic structure consists of the following components.



3.1 Corpus

The Corpus is the main object that represents a dataset/corpus.

3.2 Track

A track is an abstract representation of an audio signal. There are currently two implementations. One that reads the audio signal from a file and one that read the audio signal from a HDF5 container.

3.3 Utterance

An utterance represents a segment of a track. It is used to divide a track into independent segments. A track can have one or more utterances. The utterances are basically the samples in terms of machine learning.

3.4 Issuer

The issuer is defined as the person/thing/… who generate/produced the utterance (e.g. The speaker who read a given utterance).

An issuer can be further distinguished into different types. The current implementation provides classes for speaker (for spoken audio content) and for artists (for musical content).

3.5 LabelList

The label-list is a container for holding all labels of a given type for one utterance. For example there is a label-list containing the textual transcription of recorded speech. Another possible type of label-list could hold all labels classifying the audio type (music, speech, noise) of every part of a radio broadcast recording.

3.6 Label

The label is defining any kind of annotation for a part of or the whole utterance.

3.7 FeatureContainer

A feature-container is a container holding the feature matrices of a given type (e.g. mfcc) for all utterances. A corpus can contain multiple feature-containers.

CHAPTER 4

Corpus Formats

4.1 Default Format

This describes, how a corpus with the default format is saved on disk. Every corpus is a folder with a bunch of files.

files.txt

This file contains a list of every audio file in the corpus. Every file is identified by a unique id. Every line in the file contains the mapping from file-id to the file-path for a single file. The filepath is the path to the audio file relative to the corpus folder.

```
<recording-id> <wav-file-path>
```

Example:

```
2014-03-17-09-45-16_Kinect-Beam train/2014-03-17-09-45-16_Kinect-Beam.wav
2014-03-17-09-45-16_Realtek train/2014-03-17-09-45-16_Realtek.wav
2014-03-17-09-45-16_Yamaha train/2014-03-17-09-45-16_Yamaha.wav
2014-03-17-10-26-07_Realtek train/2014-03-17-10-26-07_Realtek.wav
```

utterances.txt

This file contains all utterances in the corpus. An utterance is a part of a file (A file can contain one or more utterances). Every line in this file defines a single utterance, which consists of utterance-id, file-id, start and end. Start and end are measured in seconds within the file. If end is -1 it is considered to be the end of the file (If the utterance is the full length of the file, start and end are 0/-1).

```
<utterance-id> <recording-id> <start> <end>
```

Example:

```
1_hello 2014-03-17-09-45-16_Kinect-Beam
1_hello_sam 2014-03-17-09-45-16_Realtek 0 -1
2_this_is 2014-03-17-09-45-16_Yamaha 0 5
3_goto 2014-03-17-09-45-16_Yamaha 5 -1
```

utt_issuers.txt

This file contains the mapping from utterance to issuers, which gives the information who/what is the origin of a given utterance (e.g. the speaker). Every line contains one mapping from utterance-id to issuer-id.

```
<utterance-id> <issuer-id>
```

Example:

```
1_hello marc
1_hello_sam marc
2_this_is sam
3_goto jenny
```

issuers.json

This file contains additional information about the issuers. Depending on the type of the issuer (Issuer, Speaker, Artist which are defined via *type* parameter) different parameters can be set.

Issuer:

- info: An arbitrary info dictionary.

Speaker:

- gender (MALE/FEMALE): The gender of a speaker
- age_group (child, youth, adult, senior): The age group of a speaker
- native_language (language code ISO 639-3): The language natively spoken

Artist:

- name: The name of the artist/band/group

```
{
  "speaker-1": {
    "info": {},
    "type": "speaker",
    "gender": "MALE",
    "age_group": "ADULT",
    "native_language": "deu"
  },
  "speaker-2": {
    "info": {},
    "type": "artist",
    "name": "Ohooo"
  },
  "speaker-3": {
    "info": {
      "region": "zh"
    }
  }
}
```

labels_[x].txt

There can be multiple label-lists in a corpus (e.g. text-transcription, raw-text-transcription - with punctuation, audio classification type, . . .). Every label-list is saved in a separate file with the prefix *labels_*. A single file contains labels of a specific type for all utterances. A label-list of an utterance can contain one or more labels (e.g. in a text segmentation every word could be a label). A label optionally can have a start and end time (in seconds within the utterance). For labels without start/end defined 0/-1 is set. Every line in the file defines one label. The labels are stored in order per utterance (e.g. 1. word, 2. word, 3. word, . . .). Optionally addtional meta-information can be stored per label. This has to be a json string in square brackets.

```
<utterance-id> <start> <end> <label-value> [<label-meta>]
```

Example:

```
1_hello 0 -1 hi
1_hello 0 -1 this
1_hello 0 -1 is
1_hello_sam 0 -1 hello
1_hello_sam 0 -1 sam
2_this_is 0 -1 this
2_this_is 0 -1 is [{"prio": 3}]
2_this_is 0 -1 me [{"stress": true}]
3_goto 0 -1 go
3_goto 0 -1 to
3_goto 0 -1 the
3_goto 0 -1 mall
```

features.txt

Contains a list of stored features. A corpus can have different feature containers. Every container contains the features of all utterances of a given type (e.g. MFCC features). A feature container is a h5py file which contains a dataset per utterance. Every line contains one container of features.

```
<feature-name> <relative-path>
```

Example:

```
mfcc mfcc_features
fbank fbank_features
```

audio.txt

Contains a list of tracks that are stored in audio-containers. Every entry consists of a track-id, the relative path to the container and a key that identifies the track in the audio-container.

```
<track-id> <audio-container-path> <audio-container-key>
```

Example:

```
track-1 ../audio.hdf5 track-1
track-2 ../audio.hdf5 track-2x
```

4.2 Broadcast Format

The broadcast format is basically the same as *Default Format*, except it uses another format to store labels. This format is meant for data where not many utterances are given, but with a lot of labels. So instead to have all labels per label-list in one file, a label-file per utterance is used.

labels.txt

This files defines where to find the effective label files. It stores the label-file path per utterance. Additionaly a label-list-id can be given, if there are multiple label-lists per utterance.

```
<utt-id> <label-file-path> <label-list-idx>
```

Example:

```
utt-1 files/a/labels.txt
utt-2 files/b/music.txt music
utt-2 files/b/jingles.txt jingles
utt-3 files/c/trailers.txt
```

[label-file].txt

The label files reference by the *labels.txt* are in the following format. It contains the start and end in seconds. The values are **Tab-separated**. Optionally additional meta-information can be stored per label. This has to be a json string in square brackets with a space separated after the label-value.

```
<start> <end>   <value> [<label-meta>]
```

Example:

```
0   40  hallo
40.5    100 velo
102.4   109.2   auto [{"lang": "de", "type": 2}]
```

A corpus format defines how a corpus is saved on disk. For the use with this library some formats were specifically developed:

- Default `audiomate.corpus.io.DefaultReader` / `audiomate.corpus.io.DefaultWriter`
- Broadcast `audiomate.corpus.io.BroadcastReader`

Furthermore there exist downloaders, readers and writers for other formats or specific datasets. For a list of available downloaders, readers and writers check *Implementations*.

CHAPTER 5

Add Dataset/Format

In this section it is described how to a downloader, reader or writer for a new dataset or another corpus format. The implementation is pretty straight-forward. For examples checkout some of the existing implementations at `audiomate.corpus.io`.

Important:

- Use the same name (`type()` method) for downloader/reader/writer.
- Import your components in `audiomate.corpus.io.__init__`. So all components are available from the io module.
- Checkout *Data Mapping* on what and how to add info/data when reading a corpus.

5.1 Corpus Downloader

If we aim to load some specific dataset/corpus, a downloader can be implemented, if it is possible to automate the whole download process. First we create a new class that inherits from `audiomate.corpus.io.CorpusDownloader`. There we have to implement two methods. The `type` method just has to return a string with the name of the new dataset/format. The `_download` method will do the heavy work of download all the files to the path `target_path`.

```
from audiomate.corpus.io import base


class MyDownloader(base.CorpusDownloader):

    @classmethod
    def type(cls):
        return 'MyDataset'

```

(continues on next page)

(continued from previous page)

```
    def _download(self, target_path):
        # Download the data to target_path
```

In the module `audiomate.corpus.io.downloader`, common base classes for downloaders are implemented. This is useful since for a lot of corpora the way of downloading is similar.

- `audiomate.corpus.io.ArchiveDownloader`: For corpora based on a single archive.

5.2 Corpus Reader

The reader is the one component that is mostly used. Either for a specific dataset/corpus or a custom format, a reader is most likely to be required. First we create a new class that inherits from `audiomate.corpus.io.CorpusReader`. There we have to implement three methods. The `type` method just has to return a string with the name of the new dataset/format. The `_check_for_missing_files` method can be used to check if the given path is a valid input. For example if the format/dataset requires some specific meta-files it can be check here if they are available. Finally in the `_load` method the actual loading is done and the loaded corpus is returned.

```
from audiomate.corpus.io import base


class MyReader(base.CorpusReader):

    @classmethod
    def type(cls):
        return 'MyDataset'

    def _check_for_missing_files(self, path):
        # Check the path for missing files that are required to read with this reader.
        # Return a list of missing files
        return []

    def _load(self, path):
        # Create a new corpus
        corpus = audiomate.Corpus(path=path)

        # Create files ...
        corpus.new_file(file_path, file_idx)

        # Issuers ...
        issuer = assets.Speaker(issuer_idx)
        corpus.import_issuers(issuer)

        # Utterances with labels ...
        utterance = corpus.new_utterance(file_idx, file_idx, issuer_idx)
        utterance.set_label_list(annotations.LabelList(idx='transcription', labels=[
            annotations.Label(str(digit))
        ]))

        return corpus
```

For some datasets there are files/utterances that are not valid. (This can be due to a corrupt file, invalid transcription, ...) For this case a json-file `audiomate/corpus/io/data/[reader-type]/invalid_utterances.json` can be created that contains a list with ids of invalid utterances. The ids correspond to id of the utterance, if it would be loaded anyway.

5.2.1 Testing

For testing a reader the `tests.corpus.io.reader_test.CorpusReaderTest` can be used. It provides base test methods for checking the correctness/existence of the basic components (tracks, utterances, labels, . . .).

```
from tests.corpus.io import reader_test as rt

class TestMyReader(rt.CorpusReaderTest):

   #
   # Define via EXPECTED_* variables, what components are expected to be loaded
   #
   EXPECTED_NUMBER_OF_TRACKS = 3
   EXPECTED_TRACKS = [
      rt.ExpFileTrack('file-id', '/path/to/file'),
   ]

   #
   # Override the load method, that loads the sample-corpus.
   #
   def load(self):
      return MyReader().load('/path/to/sample/corpus')
```

For testing any custom functionality specific test-methods can be added as well.

5.3 Corpus Writer

A writer is only useful for custom formats. For a specific dataset a writer is most likely not needed. First we create a new class that inherits from `audiomate.corpus.io.CorpusWriter`. There we have to implement two methods. The `type` method just has to return a string with the name of the new dataset/format. The `_save` method does the serialization of the given corpus to the given path.

```
from audiomate.corpus.io import base


class DefaultWriter(base.CorpusWriter):

    @classmethod
    def type(cls):
        return 'MyDataset'

    def _save(self, corpus, path):
        # Do the serialization
```

CHAPTER 6

Data Mapping

Since we want to have a consistent abstraction of different formats and datasets, it is important that all data and information is mapped correctly into the python classes.

6.1 Issuer

The issuer holds information about the source of the audio content. Depending on the audio content different attributes are important. Therefore different types of issuers can be used.

Speech For audio content that mainly contains spoken content the `audiomate.issuers.Speaker` has to be used. This is most common for datasets regarding speech recognition/synthesis etc.

Music For audio content that contains music, the `audiomate.issuers.Artist` has to be used.

6.2 Labels

In the corpus data structures an utterance can have multiple label-lists. In order to access a label-list a key is used.

```
utterance = ...
label_list = utterance.label_lists['word-transcription']
```

The used key should be consistent for all datasets. Therefore the identifiers/keys should be selected from below if possible. For these predefined keys, constants are defined in `audiomate.corpus`.

6.2.1 general

domain A high-level category for a given audio excerpt. Should be one of the following values:

- speech
- music

- noise

6.2.2 speech

word-transcript Non-aligned transcription of speech.

word-transcript-raw Non-aligned transcription of speech. Used for unprocessed transcriptions (e.g. containing punctuation, . . .).

word-transcript-aligned Aligned transcription of speech. The begin and end of the words is defined. Every word is a single label in the label-list.

phone-transcript Non-aligned transcription of phones.

phone-transcript-aligned Aligned transcription of phones. Begin and end of phones is defined.

6.2.3 music

genre The genre of the music.

6.2.4 noise

sound-class Labels defining any sound-event, acoustic-scene, environmental noise, . . . e.g. siren, dog_bark, train, car, snoring . . .

This list isn't complete. Please open an issue for any additional domains/classes that maybe needed.

CHAPTER 7

Indirectly Supported Corpora

Some corpora are hard to integrate directly, e.g. due to necessary preprocessing steps.

CHAPTER 8

Logging

Logging in audiomate is done using the standard Python logging facilities.

8.1 Enable Logging

By default, only messages of severity `Warning` or higher are printed to `sys.stderr`. Audiomate provides detailed information about progress of long-running tasks with messages of severity `Info`. To enable logging of messages of lower severity, configure Python's logging system as follows:

```
import logging

logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)-15s  %(name)s  %(message)s'
)
```

For further information check the python logging documentation.

8.2 Create log messages in audiomate

Logging in audiomate is done with a single logger. The logger is available in `audiomate.logutil`.

```
from audiomate import logutil

logger = logutil.getLogger()

def some_functionality():
    logger.debug('message')
```

Since audiomate has a lot of long-running tasks, a special function for logging the progress of a loop can be used. It basically is a wrapper around an iterable to check and log the progress. In order to keep the logs as small as possible, progress is logged in steps of 5 minutes.

```
from audiomate import logutil

logger = logutil.getLogger()

for utterance in logger.progress(
        corpus.utterances.values(),
        total=corpus.num_utterances,
        description='Process utterances'):

    # Do something with the utterance,
    # that takes up some time.
```

CHAPTER 9

audiomate.tracks

This module contains the different implementations of a track. A track is an abstract representation of an audio signal.

A concrete implementation provides the functionalty for reading the audio samples from a specific source.

9.1 Track

class audiomate.tracks.**Track**(*idx*)

Track is the abstract base class for an audio track.

Parameters **idx** (*str*) – A identifier to uniquely identify a track.

duration

Return the duration in seconds.

num_channels

Return the number of channels.

num_samples

Return the total number of samples.

read_frames(*frame_size*, *hop_size*, *offset=0*, *duration=None*, *buffer_size=5760000*)

Generator that reads and returns the samples of the track in frames.

Parameters

- **frame_size** (*int*) – The number of samples per frame.
- **hop_size** (*int*) – The number of samples between two frames.
- **offset** (*float*) – The time in seconds, from where to start reading the samples (rel. to the track start).
- **duration** (*float*) – The length of the samples to read in seconds.

Returns A generator yielding a tuple for every frame. The first item is the frame, the second the sampling-rate and the third a boolean indicating if it is the last frame.

Return type Generator

read_samples (*sr=None*, *offset=0*, *duration=None*)
Return the samples of the track.

Parameters

- **sr** (*int*) – If None, uses the native sampling-rate, otherwise resamples to the given sampling rate.
- **offset** (*float*) – The time in seconds, from where to start reading the samples (rel. to the track start).
- **duration** (*float*) – The length of the samples to read in seconds.

Returns A numpy array containing the samples as a floating point (numpy.float32) time series.

Return type np.ndarray

sampling_rate
Return the sampling rate.

9.2 FileTrack

class audiomate.tracks.**FileTrack** (*idx*, *path*)
A track that is stored in a file.

Parameters

- **idx** (*str*) – A identifier to uniquely identify a track.
- **path** (*str*) – The path to the file.

duration
Return the duration in seconds.

num_channels
Return the number of channels.

num_samples
Return the total number of samples.

read_frames (*frame_size*, *hop_size*, *offset=0*, *duration=None*, *buffer_size=5760000*)
Generator that reads and returns the samples of the track in frames.

Parameters

- **frame_size** (*int*) – The number of samples per frame.
- **hop_size** (*int*) – The number of samples between two frames.
- **offset** (*float*) – The time in seconds, from where to start reading the samples (rel. to the track start).
- **duration** (*float*) – The length of the samples to read in seconds.

Returns A generator yielding a tuple for every frame. The first item is the frame, the second the sampling-rate and the third a boolean indicating if it is the last frame.

Return type Generator

read_samples (*sr=None*, *offset=0*, *duration=None*)
Return the samples from the file. Uses librosa for loading (see http://librosa.github.io/librosa/generated/librosa.core.load.html).

Parameters

- **sr** (*int*) – If None, uses the sampling rate given by the file, otherwise resamples to the given sampling rate.
- **offset** (*float*) – The time in seconds, from where to start reading the samples (rel. to the file start).
- **duration** (*float*) – The length of the samples to read in seconds.

Returns A numpy array containing the samples as a floating point (numpy.float32) time series.

Return type np.ndarray

sampling_rate
Return the sampling rate.

9.3 ContainerTrack

class audiomate.tracks.**ContainerTrack**(*idx*, *container*, *key=None*)
A track that is stored in a *audiomate.containers.AudioContainer*.

Parameters

- **idx** (*str*) – A identifier to uniquely identify a track.
- **container** (AudioContainer) – The audio container with the samples.
- **key** (*str*) – The key of the samples in the container. If None, it is assumed it's the same as idx.

duration
Return the duration in seconds.

num_channels
Return the number of channels.

num_samples
Return the total number of samples.

read_frames(*frame_size*, *hop_size*, *offset=0*, *duration=None*, *buffer_size=None*)
Generator that reads and returns the samples of the track in frames.

Parameters

- **frame_size** (*int*) – The number of samples per frame.
- **hop_size** (*int*) – The number of samples between two frames.
- **offset** (*float*) – The time in seconds, from where to start reading the samples (rel. to the track start).
- **duration** (*float*) – The length of the samples to read in seconds.

Returns A generator yielding a tuple for every frame. The first item is the frame, the second the sampling-rate and the third a boolean indicating if it is the last frame.

Return type Generator

read_samples(*sr=None*, *offset=0*, *duration=None*)
Return the samples from the track in the container. Uses librosa for resampling, if needed.

Parameters

- **sr** (*int*) – If None, uses the sampling rate given by the file, otherwise resamples to the given sampling rate.
- **offset** (*float*) – The time in seconds, from where to start reading the samples (rel. to the file start).
- **duration** (*float*) – The length of the samples to read in seconds.

Returns A numpy array containing the samples as a floating point (numpy.float32) time series.

Return type np.ndarray

sampling_rate
Return the sampling rate.

9.4 Utterance

class audiomate.tracks.**Utterance**(*idx*, *track*, *issuer=None*, *start=0*, *end=inf*, *label_lists=None*)
An utterance defines a sample of audio. It is part of a track or can span over the whole track.

Parameters

- **idx** (*str*) – A unique identifier for the utterance within a dataset.
- **track** (Track) – The track this utterance is belonging to.
- **issuer** (Issuer) – The issuer this utterance was created from.
- **start** (*float*) – The start of the utterance within the audio track in seconds. (default 0)
- **end** (*float*) – The end of the utterance within the audio track in seconds. inf indicates that the utterance ends at the end of the track. (default inf)
- **label_lists** (LabelList*, list*) – A single or multiple label-lists.

Variables label_lists (*dict*) – A dictionary containing label-lists with the label-list-idx as key.

all_label_values(*label_list_ids=None*)
Return a set of all label-values occurring in this utterance.

Parameters label_list_ids (*list*) – If not None, only label-values from label-lists with an id contained in this list are considered.

Returns A set of distinct label-values.

Return type set

all_tokens(*delimiter=' '*, *label_list_ids=None*)
Return a list of all tokens occurring in one of the labels in the label-lists.

Parameters

- **delimiter** (*str*) – The delimiter used to split labels into tokens (see *audiomate.annotations.Label.tokenized()*).
- **label_list_ids** (*list*) – If not None, only labels from label-lists with an idx contained in this list are considered.

Returns A set of distinct tokens.

Return type set

duration
Return the absolute duration in seconds.

end_abs
Return the absolute end of the utterance relative to the signal.

label_count (*label_list_ids=None*)
Return a dictionary containing the number of times, every label-value in this utterance is occurring.

Parameters **label_list_ids** (*list*) – If not None, only labels from label-lists with an id contained in this list are considered.

Returns

A dictionary containing the number of occurrences with the label-value as key.

Return type dict

label_total_duration (*label_list_ids=None*)
Return a dictionary containing the number of seconds, every label-value is occurring in this utterance.

Parameters **label_list_ids** (*list*) – If not None, only labels from label-lists with an id contained in this list are considered.

Returns

A dictionary containing the number of seconds with the label-value as key.

Return type dict

num_samples (*sr=None*)
Return the number of samples.

Parameters **sr** (*int*) – Calculate the number of samples with the given sampling-rate. If None use the native sampling-rate.

Returns Number of samples

Return type int

read_samples (*sr=None*, *offset=0*, *duration=None*)
Read the samples of the utterance.

Parameters

- **sr** (*int*) – If None uses the sampling rate given by the track, otherwise resamples to the given sampling rate.
- **offset** (*float*) – Offset in seconds to read samples from.
- **duration** (*float*) – If not `None` read only this number of seconds in maximum.

Returns

A numpy array containing the samples as a floating point (numpy.float32) time series.

Return type np.ndarray

sampling_rate
Return the sampling rate.

set_label_list (*label_lists*)
Set the given label-list for this utterance. If the label-list-idx is not set, `default` is used. If there is already a label-list with the given idx, it will be overriden.

Parameters **label_list** (*LabelList, list*) – A single or multi. label-lists to add.

split (*cutting_points*, *track_relative=False*, *overlap=0.0*)

Split the utterance into x parts (sub-utterances) and return them as new utterances. x is defined by cutting_points (`x = len(cutting_points) + 1`).

By default cutting-points are relative to the start of the utterance. For example if an utterance starts at 50s, a cutting-point of 10.0 will split the utterance at 60s relative to the track.

Parameters

- **cutting_points** (*list*) – List of floats defining the times in seconds where to split the utterance.
- **track_relative** (*bool*) – If `True`, cutting-points are relative to the start of the track. Otherwise they are relative to the start of the utterance.
- **overlap** (*float*) – Amount of overlap in seconds. This amount is subtracted from a start-cutting-point, and added to a end-cutting-point.

Returns List of *Utterance*'s.

Return type list

Example

```
>>> utt = Utterance('utt-1', 'file-x', start=0.0, end=30.0)
>>> sub_utts = utt.split([10.0, 20.0])
>>> len(sub_utts)
3
>>> sub_utts[0].start
0.0
>>> sub_utts[0].end
10.0
```

CHAPTER 10

audiomate.annotations

This module contains classes to describe the content of an audio-segment.

10.1 Label

class `audiomate.annotations.`**`Label`**(*value*, *start=0*, *end=inf*, *meta=None*)

Represents a label that describes some part of an utterance.

Parameters

- **`value`** (*`str`*) – The text of the label.
- **`start`** (*`float`*) – Start of the label within the utterance in seconds. (default: 0)
- **`end`** (*`float`*) – End of the label within the utterance in seconds. (default: inf) (inf defines the end of the utterance)
- **`meta`** (*`dict`*) – A dictionary containing additional information for the label.

Variables **`label_list`** (`LabelList`) – The label-list this label is belonging to.

`do_overlap`(*other_label*, *adjacent=True*)

Determine whether `other_label` overlaps with this label. If `adjacent==True`, adjacent labels are also considered as overlapping.

Parameters

- **`other_label`** (`Label`) – Another label.
- **`adjacent`** (*`bool`*) – If `True`, adjacent labels are considered as overlapping.

Returns `True` if the two labels overlap, `False` otherwise.

Return type bool

`duration`

Return the duration of the label in seconds.

end_abs
Return the absolute end of the label in seconds relative to the signal. If the label isn't linked to any utterance via label-list, it is assumed `self.end` is relative to the start of the signal, hence `self.end` == `self.end_abs`.

length
Return the length of the label (Number of characters).

overlap_duration(*other_label*)
Return the duration of the overlapping part between this label and `other_label`.

Parameters **other_label** (`Label`) – Another label to check.

Returns The duration of overlap in seconds.

Return type float

Example

```
>>> label_a = Label('a', 3.4, 5.6)
>>> label_b = Label('b', 4.8, 6.2)
>>> label_a.overlap_duration(label_b)
0.8
```

read_samples(*sr=None*)
Read the samples of the utterance.

Parameters **sr** (*int*) – If None uses the sampling rate given by the track, otherwise resamples to the given sampling rate.

Returns A numpy array containing the samples as a floating point (numpy.float32) time series.

Return type np.ndarray

start_abs
Return the absolute start of the label in seconds relative to the signal. If the label isn't linked to any utterance via label-list, it is assumed `self.start` is relative to the start of the signal, hence `self.start` == `self.start_abs`.

tokenized(*delimiter=' '*)
Return a list with tokens from the value of the label. Tokens are extracted by splitting the string using `delimiter` and then trimming any whitespace before and after splitted strings.

Parameters **delimiter** (*str*) – The delimiter used to split into tokens. (default: space)

Returns A list of tokens in the order they occur in the label.

Return type list

Examples

```
>>> label = Label('as is oh')
>>> label.tokenized()
['as', 'is', 'oh']
```

Using a different delimiter (whitespace is trimmed anyway):

```
>>> label = Label('oh hi, as, is  ')
>>> label.tokenized(delimiter=',')
['oh hi', 'as', 'is']
```

10.2 LabelList

class audiomate.annotations.**LabelList**(*idx='default'*, *labels=None*)

Represents a list of labels which describe an utterance. An utterance can have multiple label-lists.

Parameters

- **idx** (*str*) – An unique identifier for the label-list within a corpus for one utterance.
- **labels** (*list*) – The list containing the *audiomate.annotations.Label*.

Variables

- **utterance** (Utterance) – The utterance this label-list is belonging to.
- **label_tree** (*IntervalTree*) – The interval-tree storing the labels.

Example

```
>>> label_list = LabelList(idx='transcription', labels=[
>>>     Label('this', 0, 2),
>>>     Label('is', 2, 4),
>>>     Label('timmy', 4, 8)
>>> ])
```

add(*label*)

Add a label to the end of the list.

Parameters **label** (Label) – The label to add.

addl(*value*, *start=0.0*, *end=inf*)

Shortcut for `add(Label(value, start, end))`.

all_tokens(*delimiter=' '*)

Return a list of all tokens occurring in the label-list.

Parameters **delimiter** (*str*) – The delimiter used to split labels into tokens. See *audiomate.annotations.Label.tokenized()*

Returns A set of distinct tokens.

Return type set

apply(*fn*)

Apply the given function *fn* to every label in this label list. *fn* is a function of one argument that receives the current label which can then be edited in place.

Parameters **fn** (*func*) – Function to apply to every label

Example

```
>>> ll = LabelList(labels=[
...     Label('a_label', 1.0, 2.0),
...     Label('another_label', 2.0, 3.0)
... ])
>>> def shift_labels(label):
...     label.start += 1.0
...     label.end += 1.0
...
>>> ll.apply(shift_labels)
>>> ll.labels
[Label(a_label, 2.0, 3.0), Label(another_label, 3.0, 4.0)]
```

classmethod create_single(*value*, *idx='default'*)
Create a label-list with a single label containing the given value.

end
Return end of the lastly ending label (upper bound).

join(*delimiter=' '*, *overlap_threshold=0.1*)
Return a string with all labels concatenated together. The order of the labels is defined by the start of the label. If the overlapping between two labels is greater than `overlap_threshold`, an Exception is thrown.

Parameters

- **delimiter** (*str*) – A string to join two consecutive labels.
- **overlap_threshold** (*float*) – Maximum overlap between two consecutive labels.

Returns A string with all labels concatenated together.

Return type str

Example

```
>>> ll = LabelList(idx='some', labels=[
>>>     Label('a', start=0, end=4),
>>>     Label('b', start=3.95, end=6.0),
>>>     Label('c', start=7.0, end=10.2),
>>>     Label('d', start=10.3, end=14.0)
>>> ])
>>> ll.join(' - ')
'a - b - c - d'
```

label_count()
Return for each label the number of occurrences within the list.

Returns A dictionary containing for every label-value (key) the number of occurrences (value).

Return type dict

Example

```
>>> ll = LabelList(labels=[
>>>     Label('a', 3.2, 4.5),
>>>     Label('b', 5.1, 8.9),
>>>     Label('a', 7.2, 10.5),
```

(continues on next page)

(continued from previous page)

```
>>>     Label('b', 10.5, 14),
>>>     Label('a', 15, 18)
>>> ])
>>> ll.label_count()
{'a': 3 'b': 2}
```

label_total_duration()
Return for each distinct label value the total duration of all occurrences.

Returns

A dictionary containing for every label-value (key) the total duration in seconds (value).

Return type dict

Example

```
>>> ll = LabelList(labels=[
>>>     Label('a', 3, 5),
>>>     Label('b', 5, 8),
>>>     Label('a', 8, 10),
>>>     Label('b', 10, 14),
>>>     Label('a', 15, 18.5)
>>> ])
>>> ll.label_total_duration()
{'a': 7.5 'b': 7.0}
```

label_values()
Return a list of all occuring label values.

Returns Lexicographically sorted list (str) of label values.

Return type list

Example

```
>>> ll = LabelList(labels=[
>>>     Label('a', 3.2, 4.5),
>>>     Label('b', 5.1, 8.9),
>>>     Label('c', 7.2, 10.5),
>>>     Label('d', 10.5, 14),
>>>     Label('d', 15, 18)
>>> ])
>>> ll.label_values()
['a', 'b', 'c', 'd']
```

labels
Return list of labels.

labels_in_range(*start*, *end*, *fully_included=False*)
Return a list of labels, that are within the given range. Also labels that only overlap are included.

Parameters

- **start** (`float`) – Start-time in seconds.
- **end** (`float`) – End-time in seconds.

- **fully_included** (*bool*) – If `True`, only labels fully included in the range are returned. Otherwise also overlapping ones are returned. (default `False`)

Returns List of labels in the range.

Return type list

Example

```
>>> ll = LabelList(labels=[
>>>     Label('a', 3.2, 4.5),
>>>     Label('b', 5.1, 8.9),
>>>     Label('c', 7.2, 10.5),
>>>     Label('d', 10.5, 14)
>>>])
>>> ll.labels_in_range(6.2, 10.1)
[Label('b', 5.1, 8.9), Label('c', 7.2, 10.5)]
```

merge_overlaps (*threshold=0.0*)

Merge overlapping labels with the same value. Two labels are considered overlapping, if `l2.start - l1.end < threshold`.

Parameters **threshold** (*float*) – Maximal distance between two labels to be considered as overlapping. (default: 0.0)

Example

```
>>> ll = LabelList(labels=[
...     Label('a_label', 1.0, 2.0),
...     Label('a_label', 1.5, 2.7),
...     Label('b_label', 1.0, 2.0),
... ])
>>> ll.merge_overlapping_labels()
>>> ll.labels
[
    Label('a_label', 1.0, 2.7),
    Label('b_label', 1.0, 2.0),
]
```

ranges (*yield_ranges_without_labels=False, include_labels=None*)

Generate all ranges of the label-list. A range is defined as a part of the label-list for which the same labels are defined.

Parameters

- **yield_ranges_without_labels** (*bool*) – If True also yields ranges for which no labels are defined.
- **include_labels** (*list*) – If not empty, only the label values in the list will be considered.

Returns A generator which yields one range (tuple start/end/list-of-labels) at a time.

Return type generator

Example

```
>>> ll = LabelList(labels=[
>>>     Label('a', 3.2, 4.5),
>>>     Label('b', 5.1, 8.9),
>>>     Label('c', 7.2, 10.5),
>>>     Label('d', 10.5, 14)
>>>])
>>> ranges = ll.ranges()
>>> next(ranges)
(3.2, 4.5, [ < audiomate.annotations.Label at 0x1090527c8 > ])
>>> next(ranges)
(4.5, 5.1, [])
>>> next(ranges)
(5.1, 7.2, [ < audiomate.annotations.label.Label at 0x1090484c8 > ])
```

separated()

Create a separate Label-List for every distinct label-value.

Returns

A dictionary with distinct label-values as keys. Every value is a LabelList containing only labels with the same value.

Return type dict

Example

```
>>> ll = LabelList(idx='some', labels=[
>>>     Label('a', start=0, end=4),
>>>     Label('b', start=3.95, end=6.0),
>>>     Label('a', start=7.0, end=10.2),
>>>     Label('b', start=10.3, end=14.0)
>>> ])
>>> s = ll.separate()
>>> s['a'].labels
[Label('a', start=0, end=4), Label('a', start=7.0, end=10.2)]
>>> s['b'].labels
[Label('b', start=3.95, end=6.0), Label('b', start=10.3, end=14.0)]
```

split(*cutting_points*, *shift_times=False*, *overlap=0.0*)

Split the label-list into x parts and return them as new label-lists. x is defined by the number of cutting-points (`x == len(cutting_points) + 1`).

The result is a list of label-lists corresponding to each part. Label-list 0 contains labels between `0` and `cutting_points[0]`. Label-list 1 contains labels between `cutting_points[0]` and `cutting_points[1]`. And so on.

Parameters

- **cutting_points** (*list*) – List of floats defining the points in seconds, where the label-list is splitted.
- **shift_times** (*bool*) – If True, start and end-time are shifted in splitted label-lists. So the start is relative to the cutting point and not to the beginning of the original label-list.
- **overlap** (*float*) – Amount of overlap in seconds. This amount is subtracted from a start-cutting-point, and added to a end-cutting-point.

Returns A list of of: class: *audiomate.annotations.LabelList*.

Return type list

Example

```
>>> ll = LabelList(labels=[
>>>     Label('a', 0, 5),
>>>     Label('b', 5, 10),
>>>     Label('c', 11, 15),
>>>])
>>>
>>> res = ll.split([4.1, 8.9, 12.0])
>>> len(res)
4
>>> res[0].labels
[Label('a', 0.0, 4.1)]
>>> res[1].labels
[
    Label('a', 4.1, 5.0),
    Label('b', 5.0, 8.9)
]
>>> res[2].labels
[
    Label('b', 8.9, 10.0),
    Label('c', 11.0, 12.0)
]
>>> res[3].labels
[Label('c', 12.0, 15.0)]
```

If `shift_times = True`, the times are adjusted to be relative to the cutting-points for every label-list but the first.

```
>>> ll = LabelList(labels=[
>>>     Label('a', 0, 5),
>>>     Label('b', 5, 10),
>>>])
>>>
>>> res = ll.split([4.6])
>>> len(res)
4
>>> res[0].labels
[Label('a', 0.0, 4.6)]
>>> res[1].labels
[
    Label('a', 0.0, 0.4),
    Label('b', 0.4, 5.4)
]
```

start
Return start of the earliest starting label (lower bound).

tokenized(*delimiter=' '*, *overlap_threshold=0.1*)
Return a ordered list of tokens based on all labels. Joins all token from all labels (`label.tokenized()``). If the overlapping between two labels is greater than `overlap_threshold`, an Exception is thrown.

Parameters

- **delimiter** (*str*) – The delimiter used to split labels into tokens. (default: space)
- **overlap_threshold** (*float*) – Maximum overlap between two consecutive labels.

Returns

A list containing tokens of all labels ordered according to the label order.

Return type str

Example

```
>>> ll = LabelList(idx='some', labels=[
>>>     Label('a d q', start=0, end=4),
>>>     Label('b', start=3.95, end=6.0),
>>>     Label('c a', start=7.0, end=10.2),
>>>     Label('f g', start=10.3, end=14.0)
>>> ])
>>> ll.tokenized(delimiter=' ', overlap_threshold=0.1)
['a', 'd', 'q', 'b', 'c', 'a', 'f', 'g']
```

total_length

Return the cumulative length of all labels (Number of characters).

update (*labels*)

Add a list of labels to the end of the list.

Parameters labels (*list*) – Labels to add.

classmethod with_label_values (*values*, *idx='default'*)

Create a new label-list containing labels with the given values. All labels will have default start/end values of 0 and `inf`.

Parameters

- **values** (*list*) – List of values(str) that should be created and appended to the label-list.
- **idx** (*str*) – The idx of the label-list.

Returns New label-list.

Return type (*LabelList*)

Example

```
>>> ll = LabelList.with_label_values(['a', 'x', 'z'], idx='letters')
>>> ll.idx
'letters'
>>> ll.labels
[
    Label('a', 0, inf),
    Label('x', 0, inf),
    Label('z', 0, inf),
]
```

10.3 Relabeling

exception audiomate.annotations.relabeling.**UnmappedLabelsException**(*message*)

audiomate.annotations.relabeling.**find_missing_projections**(*label_list*, *projections*)

Finds all combinations of labels in *label_list* that are not covered by an entry in the dictionary of *projections*. Returns a list containing tuples of uncovered label combinations or en empty list if there are none. All uncovered label combinations are naturally sorted.

Each entry in the dictionary of projections represents a single projection that maps a combination of labels (key) to a single new label (value). The combination of labels to be mapped is a tuple of naturally sorted labels that apply to one or more segments simultaneously. By defining a special wildcard projection using *('**',)* is is not required to specify a projection for every single combination of labels.

Parameters

- **label_list** (audiomate.annotations.LabelList) – The label list to relabel
- **projections** (*dict*) – A dictionary that maps tuples of label combinations to string labels.

Returns List of combinations of labels that are not covered by any projection

Return type List

Example

```
>>> ll = annotations.LabelList(labels=[
...     annotations.Label('b', 3.2, 4.5),
...     annotations.Label('a', 4.0, 4.9),
...     annotations.Label('c', 4.2, 5.1)
... ])
>>> find_missing_projections(ll, {('b',): 'new_label'})
[('a', 'b'), ('a', 'b', 'c'), ('a', 'c'), ('c',)]
```

audiomate.annotations.relabeling.**load_projections**(*projections_file*)

Loads projections defined in the given *projections_file*.

The *projections_file* is expected to be in the following format:

```
old_label_1 | new_label_1
old_label_1 old_label_2 | new_label_2
old_label_3 |
```

You can define one projection per line. Each projection starts with a list of one or multiple old labels (separated by a single whitespace) that are separated from the new label by a pipe (|). In the code above, the segment labeled with *old_label_1* will be labeled with *new_label_1* after applying the projection. Segments that are labeled with *old_label_1* **and** *old_label_2* concurrently are relabeled to *new_label_2*. All segments labeled with *old_label_3* are dropped. Combinations of multiple labels are automatically sorted in natural order.

Parameters **projections_file** (*str*) – Path to the file with projections

Returns Dictionary where the keys are tuples of labels to project to the key's value

Return type dict

Example

```
>>> load_projections('/path/to/projections.txt')
{('b',): 'foo', ('a', 'b'): 'a_b', ('a',): 'bar'}
```

`audiomate.annotations.relabeling.`**`relabel`**(*label_list*, *projections*)

Relabel an entire *`LabelList`* using user-defined projections. Labels can be renamed, removed or overlapping labels can be flattened to a single label per segment.

Each entry in the dictionary of projections represents a single projection that maps a combination of labels (key) to a single new label (value). The combination of labels to be mapped is a tuple of naturally sorted labels that apply to one or more segments simultaneously. By defining a special wildcard projection using *('**',)* is is not required to specify a projection for every single combination of labels.

This method raises a `UnmappedLabelsException` if a projection for one or more combinations of labels is not defined.

Parameters

- **`label_list`** (`audiomate.annotations.LabelList`) – The label list to relabel
- **`projections`** (*`dict`*) – A dictionary that maps tuples of label combinations to string labels.

Returns New label list with remapped labels

Return type *audiomate.annotations.LabelList*

Raises *`UnmappedLabelsException`* – If a projection for one or more combinations of labels is not defined.

Example

```
>>> projections = {
...     ('a',): 'a',
...     ('b',): 'b',
...     ('c',): 'c',
...     ('a', 'b',): 'a_b',
...     ('a', 'b', 'c',): 'a_b_c',
...     ('**',): 'b_c',
... }
>>> label_list = annotations.LabelList(labels=[
...     annotations.Label('a', 3.2, 4.5),
...     annotations.Label('b', 4.0, 4.9),
...     annotations.Label('c', 4.2, 5.1)
... ])
>>> ll = relabel(label_list, projections)
>>> [l.value for l in ll]
['a', 'a_b', 'a_b_c', 'b_c', 'c']
```

10.4 Exceptions

exception `audiomate.annotations.relabeling.`**`UnmappedLabelsException`**(*message*)

CHAPTER 11

audiomate.issuers

This module contains classes to represent issuers fo audio samples. An issuer represents the person/machine/thing, that is creator of an audio segment.

11.1 Issuer

class audiomate.issuers.**Issuer**(*idx*, *info=None*)

The issuer represents a person, object or something that produced an utterance. Technically the issuer can be used to group utterances that came from the same source.

Parameters

- **idx** (*str*) – An unique identifier for this issuer within a dataset.
- **info** (*dict*) – Any additional info for this issuer as dict.

Variables **Issuer.utterances** (*list*) – List of utterances that this issuer owns.

11.2 Speaker

class audiomate.issuers.**Speaker**(*idx*, *gender=<Gender.UNKNOWN: 'unknown'>*, *age_group=<AgeGroup.UNKNOWN: 'unknown'>*, *native_language=None*, *info=None*)

The speaker is the person who spoke in a utterance.

Parameters

- **idx** (*str*) – An unique identifier for this speaker within a dataset.
- **info** (*dict*) – Any additional info for this speaker as dict.
- **age_group** (*AgeGroup*) – The age-group of the speaker (child, adult, . . .)
- **native_language** (*str*) – The native language of the speaker. (ISO 639-3)

Variables **`Issuer.utterances`** (*`list`*) – List of utterances that this issuer owns.

11.3 Artist

class `audiomate.issuers.`**`Artist`**(*idx*, *name*, *info=None*)

The artist is the person/group who have produced a musical segment in a utterance.

Parameters

- **`idx`** (*`str`*) – An unique identifier for this speaker within a dataset.
- **`name`** (*`str`*) – The name of the artist/band/. . .
- **`info`** (*`dict`*) – Any additional info for this speaker as dict.

Variables **`Issuer.utterances`** (*`list`*) – List of utterances that this issuer owns.

CHAPTER 12

audiomate.containers

This module contains the different implementations of containers. A container is normally used to store data of a specific type for all instances of a corpus (e.g. mfcc-features of all utterances).

All container implementations are based on `audiomate.containers.Container`, which provides the basic functionality to access a HDF5-file using h5py.

12.1 Container

class `audiomate.containers.`**`Container`**(*path*, *mode='a'*)

A container is a wrapper around a HDF5 file. In a container is used to store array-like data. Every array is associated with some idx/key. Every array (a dataset in h5py-terms) may have additional attributes.

Parameters

- **path** (*str*) – Path where the HDF5 file is stored. If the file doesn't exist, one is created.
- **mode** (*str*) – Either 'r' for read-only, 'w' for truncate and write or 'a' for append. (default: 'a').

Example

```
>>> ct = Container('/path/to/hdf5file')
>>> with ct:
>>>     ct.set('utt-1', np.array([1,2,3,4]))
>>>     data = ct.get('utt-1')
array([1, 2, 3, 4])
```

append(*key*, *data*)

Append the given data to the data that already exists in the container for the given key. Only data with equal dimensions (except the first) are allowed, since they are concatenated/stacked along the first dimension.

Parameters

- **key** (*str*) – Key to store data for.
- **data** (*numpy.ndarray*) – Array-like data. Has to have the same dimension as the existing data after the first dimension.

Note: The container has to be opened in advance. For appending to existing data the HDF5-Dataset has to be chunked, so it is not allowed to first add data via `set`.

close()
Close the container file if its open.

get(*key*, *mem_map=True*)
Read and return the data stored for the given key.

Parameters

- **key** (*str*) – The key to read the data from.
- **mem_map** (*bool*) – If `True` returns the data as memory-mapped array, otherwise a copy is returned.

Note: The container has to be opened in advance.

Returns The stored data.

Return type numpy.ndarray

is_open()
Return `True`, if container is already open. `False` otherwise.

keys()
Return a list of keys for which an array is stored in the container.

Returns List of identifiers available in the container.

Return type list

Note: The container has to be opened in advance.

open(*mode=None*)
Open the container file.

Parameters **mode** (*str*) – Either 'r' for read-only, 'w' for truncate and write or 'a' for append. (default: 'a'). If `None`, uses `self.mode`.

open_if_needed(*mode=None*)
Convenience context-manager for the use with `with`. Opens the container if not already done. Only closes the container if it was opened within this context.

Parameters **mode** (*str*) – Either 'r' for read-only, 'w' for truncate and write or 'a' for append. (default: 'a'). If `None`, uses `self.mode`.

raise_error_if_not_open()
Check if container is opened, raise error if not.

remove(*key*)
Remove the data stored for the given key.

Parameters **key** (*str*) – Key of the data to remove.

Note: The container has to be opened in advance.

set (*key*, *data*)
Set the given data to the container with the given key. Any existing data for the given key is discarded/overwritten.

Parameters

- **key** (*str*) – A key to store the data for.
- **data** (*numpy.ndarray*) – Array-like data.

Note: The container has to be opened in advance.

12.2 FeatureContainer

class audiomate.containers.**FeatureContainer** (*path*, *mode='a'*)
The FeatureContainer is a container for storing features extracted from audio data. Features are array-like data, where every feature represents the properties of a given segment of audio.

Parameters

- **path** (*str*) – Path to where the HDF5 file is stored. If the file doesn't exist, one is created.
- **mode** (*str*) – Either 'r' for read-only, 'w' for truncate and write or 'a' for append. (default: 'a').

Example

```
>>> fc = FeatureContainer('/path/to/hdf5file')
>>> with fc:
>>>     fc.set('utt-1', np.array([1,2,3,4]))
>>>     data = fc.get('utt-1')
array([1, 2, 3, 4])
```

append (*key*, *data*)
Append the given data to the data that already exists in the container for the given key. Only data with equal dimensions (except the first) are allowed, since they are concatenated/stacked along the first dimension.

Parameters

- **key** (*str*) – Key to store data for.
- **data** (*numpy.ndarray*) – Array-like data. Has to have the same dimension as the existing data after the first dimension.

Note: The container has to be opened in advance. For appending to existing data the HDF5-Dataset has to be chunked, so it is not allowed to first add data via `set`.

close ()
Close the container file if its open.

frame_size
The number of samples used per frame.

get (*key*, *mem_map=True*)
Read and return the data stored for the given key.

Parameters

- **key** (*str*) – The key to read the data from.
- **mem_map** (*bool*) – If `True` returns the data as memory-mapped array, otherwise a copy is returned.

Note: The container has to be opened in advance.

Returns The stored data.

Return type numpy.ndarray

hop_size
The number of samples between two frames.

is_open ()
Return `True`, if container is already open. `False` otherwise.

keys ()
Return a list of keys for which an array is stored in the container.

Returns List of identifiers available in the container.

Return type list

Note: The container has to be opened in advance.

open (*mode=None*)
Open the container file.

Parameters **mode** (*str*) – Either 'r' for read-only, 'w' for truncate and write or 'a' for append. (default: 'a'). If `None`, uses `self.mode`.

open_if_needed (*mode=None*)
Convenience context-manager for the use with `with`. Opens the container if not already done. Only closes the container if it was opened within this context.

Parameters **mode** (*str*) – Either 'r' for read-only, 'w' for truncate and write or 'a' for append. (default: 'a'). If `None`, uses `self.mode`.

raise_error_if_not_open ()
Check if container is opened, raise error if not.

remove (*key*)
Remove the data stored for the given key.

Parameters **key** (*str*) – Key of the data to remove.

Note: The container has to be opened in advance.

sampling_rate
The sampling-rate of the signal these frames are based on.

set (*key*, *data*)
Set the given data to the container with the given key. Any existing data for the given key is discarded/overwritten.

Parameters

- **key** (*str*) – A key to store the data for.
- **data** (*numpy.ndarray*) – Array-like data.

Note: The container has to be opened in advance.

stats ()
Return statistics calculated overall features in the container.

Note: The feature container has to be opened in advance.

Returns Statistics overall data points of all features.

Return type DataStats

stats_per_key ()
Return statistics calculated for each key in the container.

Note: The feature container has to be opened in advance.

Returns A dictionary containing a DataStats object for each key.

Return type dict

12.3 AudioContainer

class audiomate.containers.**AudioContainer** (*path*, *mode='a'*)
Container to store raw audio samples.

Notes

The samples are stored as 16-Bit Integers. But all methods expect or return the samples as 32-Bit Floats, in the range of -1.0 to 1.0.

append (*key*, *samples*, *sampling_rate*)
Append the given samples to the data that already exists in the container for the given key.

Parameters

- **key** (*str*) – A key to store the data for.
- **samples** (*numpy.ndarray*) – 1-D array of audio samples (int-16).
- **sampling_rate** (*int*) – The sampling-rate of the audio samples.

Note: The container has to be opened in advance. For appending to existing data the HDF5-Dataset has to be chunked, so it is not allowed to first add data via `set`.

`close`()
Close the container file if its open.

`get`(*key*, *mem_map=True*)
Return the samples for the given key and the sampling-rate.

Parameters

- **`key`** (*`str`*) – The key to read the data from.
- **`mem_map`** (*`bool`*) – If `True` returns the data as memory-mapped array, otherwise a copy is returned.

Note: The container has to be opened in advance.

Returns

A tuple containing the samples as numpy array with `np.float32` [-1.0,1.0] and the sampling-rate.

Return type tuple

`is_open`()
Return `True`, if container is already open. `False` otherwise.

`keys`()
Return a list of keys for which an array is stored in the container.

Returns List of identifiers available in the container.

Return type list

Note: The container has to be opened in advance.

`open`(*mode=None*)
Open the container file.

Parameters **`mode`** (*`str`*) – Either 'r' for read-only, 'w' for truncate and write or 'a' for append. (default: 'a'). If `None`, uses `self.mode`.

`open_if_needed`(*mode=None*)
Convenience context-manager for the use with `with`. Opens the container if not already done. Only closes the container if it was opened within this context.

Parameters **`mode`** (*`str`*) – Either 'r' for read-only, 'w' for truncate and write or 'a' for append. (default: 'a'). If `None`, uses `self.mode`.

`raise_error_if_not_open`()
Check if container is opened, raise error if not.

`remove`(*key*)
Remove the data stored for the given key.

Parameters **`key`** (*`str`*) – Key of the data to remove.

Note: The container has to be opened in advance.

set (*key*, *samples*, *sampling_rate*)

Set the samples and sampling-rate for the given key. Existing data will be overwritten. The samples have to have `np.float32` datatype and values in the range of -1.0 and 1.0.

Parameters

- **key** (*str*) – A key to store the data for.
- **samples** (*numpy.ndarray*) – 1-D array of audio samples (np.float32).
- **sampling_rate** (*int*) – The sampling-rate of the audio samples.

Note: The container has to be opened in advance.

CHAPTER 13

audiomate.corpus

This module contains all parts needed for using a corpus. Aside the main corpus class `audiomate.Corpus`, there are different loaders in the `audiomate.corpus.io`.

13.1 CorpusView

class `audiomate.corpus.`**`CorpusView`**

This class defines the basic interface of a corpus. It is not meant to be instantiated directly. It only describes the methods for accessing data of the corpus.

Notes

All paths to files should be held as absolute paths in memory.

`all_label_values`(*label_list_ids=None*)

Return a set of all label-values occurring in this corpus.

Parameters **`label_list_ids`** (*`list`*) – If not `None`, only labels from label-lists with an id contained in this list are considered.

Returns A set of distinct label-values.

Return type `set`

`all_tokens`(*delimiter=' ', label_list_ids=None*)

Return a list of all tokens occurring in one of the labels in the corpus.

Parameters

- **`delimiter`** (*`str`*) – The delimiter used to split labels into tokens. (see *`audiomate.annotations.Label.tokenized()`*)
- **`label_list_ids`** (*`list`*) – If not `None`, only labels from label-lists with an idx contained in this list are considered.

Returns A set of distinct tokens.

Return type `set`

`contains_issuer`(*issuer*)
Return `True` if the given issuer is in the corpus already, `False` otherwise.

`contains_track`(*track*)
Return `True` if the given track is in the corpus already, `False` otherwise.

`feature_containers`
Return the feature-containers in the corpus.

Returns

A dictionary containing `audiomate.container.FeatureContainer` objects with the feature-idx as key.

Return type dict

`issuers`
Return the issuers in the corpus.

Returns

A dictionary containing ***`audiomate.issuers.Issuer`*** objects with the issuer-idx as key.

Return type dict

`label_count`(*label_list_ids=None*)
Return a dictionary containing the number of times, every label-value in this corpus is occurring.

Parameters **`label_list_ids`** (*`list`*) – If not `None`, only labels from label-lists with an id contained in this list are considered.

Returns

A dictionary containing the number of occurrences with the label-value as key.

Return type dict

`label_durations`(*label_list_ids=None*)
Return a dictionary containing the total duration, every label-value in this corpus is occurring.

Parameters **`label_list_ids`** (*`list`*) – If not None, only labels from label-lists with an id contained in this list are considered.

Returns

A dictionary containing the total duration with the label-value as key.

Return type dict

`name`
Return the name of the dataset (Equals basename of the path, if not None).

`num_feature_containers`
Return the number of feature-containers in the corpus.

`num_issuers`
Return the number of issuers in the corpus.

`num_subviews`
Return the number of subviews in the corpus.

num_tracks
Return number of tracks.

num_utterances
Return number of utterances.

split_utterances_to_max_time(*max_time=60.0*, *overlap=0.0*)
Create a new corpus, where all the utterances are of given maximal duration. Utterance longer than `max_time` are split up into multiple utterances.

> **Warning:** Subviews and FeatureContainers are not added to the newly create corpus.

Parameters

- **max_time** (*float*) – Maximal duration for target utterances in seconds.
- **overlap** (*float*) – Amount of overlap in seconds. The overlap is measured from the center of the splitting. (The actual overlap of two segments is 2 * overlap)

Returns A new corpus instance.

Return type *Corpus*

stats()
Return statistics calculated overall samples of all utterances in the corpus.

Returns

A DataStats object containing statistics overall samples in the corpus.

Return type DataStats

stats_per_utterance()
Return statistics calculated for all samples of each utterance in the corpus.

Returns A dictionary containing a DataStats object for each utt.

Return type dict

subviews
Return the subviews of the corpus.

Returns

A dictionary containing `audiomate.corpus.Subview` objects with the subview-idx as key.

Return type dict

total_duration
Return the total amount of audio summed over all utterances in the corpus in seconds.

tracks
Return the tracks in the corpus.

Returns

A dictionary containing `audiomate.track.Track` objects with the track-idx as key.

Return type dict

utterances
Return the utterances in the corpus.

Returns

A dictionary containing `audiomate.corpus.assets.Utterance` objects with the utterance-idx as key.

Return type dict

13.2 Corpus

class `audiomate.corpus.`**`Corpus`**(*path=None*)

The Corpus class represents a single corpus. It extends *`audiomate.corpus.CorpusView`* with the functionality for loading and saving. Furthermore it provides the functionality for adding/modifying assets of the corpus like tracks and utterances.

Parameters **`path`** (*`str`*) – Path where the corpus is stored. (Optional)

`feature_containers`

Return the feature-containers in the corpus.

Returns

A dictionary containing `audiomate.container.FeatureContainer` objects with the feature-idx as key.

Return type dict

classmethod from_corpus(*corpus*)

Create a new modifiable corpus from any other CorpusView. This for example can be used to create a independent modifiable corpus from a subview.

Parameters **`corpus`** (`CorpusView`) – The corpus to create a copy from.

Returns A new corpus with the same data as the given one.

Return type *Corpus*

`import_issuers`(*new_issuers*)

Add the given issuers/issuer to the corpus. If any of the given issuer-ids already exists, a suffix is appended so it is unique.

Parameters **`issuers`** (*`list`*) – Either a list of or a single *`audiomate.issuers.Issuer`*.

Returns

A dictionary containing idx mappings (old-issuer-idx/issuer-instance). If a issuer is imported, whose id already exists this mapping can be used to check the new id.

Return type dict

`import_subview`(*idx*, *subview*)

Add the given subview to the corpus.

Parameters

- **`idx`** (*`str`*) – An idx that is unique in the corpus for identifying the subview. If already a subview exists with the given id it will be overridden.
- **`subview`** (`Subview`) – The subview to add.

`import_tracks`(*import_tracks*)

Add the given tracks/track to the corpus. If any of the given track-ids already exists, a suffix is appended so it is unique.

Parameters **`import_tracks`** (*`list`*) – Either a list of or a single *`audiomate.tracks.Track`*.

Returns

A dictionary containing track-idx mappings (old-track-idx/track-instance). If a track is imported, whose idx already exists this mapping can be used to check the new id.

Return type dict

`import_utterances`(*utterances*)

Add the given utterances/utterance to the corpus. If any of the given utterance-ids already exists, a suffix is appended so it is unique.

Parameters **`utterances`** (*`list`*) – Either a list of or a single *`audiomate.tracks.Utterance`*.

Returns

A dictionary containing idx mappings (old-utterance-idx/utterance-instance). If a utterance is imported, whose id already exists this mapping can be used to check the new id.

Return type dict

`issuers`

Return the issuers in the corpus.

Returns

A dictionary containing ***`audiomate.issuers.Issuer`*** objects with the issuer-idx as key.

Return type dict

`classmethod load`(*path*, *reader=None*, ***kwargs*)

Loads the corpus from the given path, using the given reader. If no reader is given the `audiomate.corpus.io.DefaultReader` is used.

Parameters

- **`path`** (*`str`*) – Path to load the corpus from.
- **`reader`** (*`str, CorpusReader`*) – The reader or the name of the reader to use.

Returns The loaded corpus.

Return type *Corpus*

`classmethod merge_corpora`(*corpora*)

Merge a list of corpora into one.

Parameters **`corpora`** (*`Iterable`*) – An iterable of *`audiomate.corpus.CorpusView`*.

Returns

A corpus with the data from all given corpora merged into one.

Return type *Corpus*

`merge_corpus`(*corpus*)

Merge the given corpus into this corpus. All assets (tracks, utterances, issuers, . . .) are copied into this corpus. If any ids (utt-idx, track-idx, issuer-idx, subview-idx, . . .) are occurring in both corpora, the ids from the merging corpus are suffixed by a number (starting from 1 until no other is matching).

Parameters **`corpus`** (`CorpusView`) – The corpus to merge.

name
Return the name of the dataset (Equals basename of the path, if not None).

new_feature_container (*idx*, *path=None*)
Add a new feature container with the given data.

Parameters

- **idx** (*str*) – An unique identifier within the dataset.
- **path** (*str*) – The path to store the feature file. If `None` a default path is used.

Returns The newly added feature-container.

Return type *FeatureContainer*

new_file (*path*, *track_idx*, *copy_file=False*)
Adds a new audio file to the corpus with the given data.

Parameters

- **path** (*str*) – Path of the file to add.
- **track_idx** (*str*) – The id to associate the file-track with.
- **copy_file** (*bool*) – If `True` the file is copied to the data set folder, otherwise the given path is used directly.

Returns The newly added file.

Return type *FileTrack*

new_issuer (*issuer_idx*, *info=None*)
Add a new issuer to the dataset with the given data.

Parameters

- **issuer_idx** (*str*) – The id to associate the issuer with. If `None` or already exists, one is generated.
- **info** (*dict, list*) – Additional info of the issuer.

Returns The newly added issuer.

Return type *Issuer*

new_utterance (*utterance_idx*, *track_idx*, *issuer_idx=None*, *start=0*, *end=inf*)
Add a new utterance to the corpus with the given data.

Parameters

- **track_idx** (*str*) – The track id the utterance is in.
- **utterance_idx** (*str*) – The id to associate with the utterance. If None or already exists, one is generated.
- **issuer_idx** (*str*) – The issuer id to associate with the utterance.
- **start** (*float*) – Start of the utterance within the track [seconds].
- **end** (*float*) – End of the utterance within the track [seconds]. `inf` equals the end of the track.

Returns The newly added utterance.

Return type *Utterance*

`relocate_audio_to_single_container`(*target_path*)
Copies every track to a single container. Afterwards all tracks in the container are linked against this single container.

`relocate_audio_to_wav_files`(*target_path*)
Copies every track to its own wav file in the given folder. Every track will be stored at `target_path/track_id.wav`.

`save`(*writer=None*)
If self.path is defined, it tries to save the corpus at the given path.

`save_at`(*path*, *writer=None*)
Save this corpus at the given path. If the path differs from the current path set, the path gets updated.

Parameters

- **`path`** (*`str`*) – Path to save the data set to.
- **`writer`** (*`str, CorpusWriter`*) – The writer or the name of the reader to use.

`subviews`
Return the subviews of the corpus.

Returns

A dictionary containing `audiomate.corpus.Subview` objects with the subview-idx as key.

Return type dict

`tracks`
Return the tracks in the corpus.

Returns

A dictionary containing `audiomate.track.Track` objects with the track-idx as key.

Return type dict

`utterances`
Return the utterances in the corpus.

Returns

A dictionary containing `audiomate.corpus.assets.Utterance` objects with the utterance-idx as key.

Return type dict

CHAPTER 14

audiomate.corpus.io

14.1 Base Classes

14.2 Implementations

Table 1: Support for Reading and Writing by Format

Format	Download	Read	Write
Acoustic Event Dataset	x	x	
AudioMNIST	x	x	
Broadcast		x	
Common Voice	x	x	
Default		x	x
ESC-50	x	x	
Free-Spoken-Digit-Dataset	x	x	
Folder		x	
Fluent Speech Commands Dataset		x	
Google Speech Commands		x	
GTZAN	x	x	
Kaldi		x	x
LibriSpeech	x	x	
Mozilla DeepSpeech			x
MUSAN	x	x	
M-AILABS Speech Dataset	x	x	
LITIS Rouen Audio scene dataset	x	x	
Spoken Wikipedia Corpora	x	x	
Tatoeba	x	x	
TIMIT		x	
TUDA German Distant Speech	x	x	
Urbansound8k		x	
VoxForge	x	x	
Wav2Letter			x

14.2.1 Acoustic Event Dataset

14.2.2 AudioMNIST

14.2.3 Broadcast

14.2.4 Common-Voice

14.2.5 Default

14.2.6 ESC-50

14.2.7 Folder

14.2.8 Free-Spoken-Digit-Dataset

14.2.9 Fluent Speech Commands Dataset

14.2.10 Google Speech Commands

14.2.11 GTZAN

14.2.12 Kaldi

14.2.13 LibriSpeech

14.2.14 Mozilla DeepSpeech

14.2.15 MUSAN

14.2.16 M-AILABS Speech Dataset

14.2.17 NVIDIA Jasper

14.2.18 LITIS Rouen Audio scene dataset

14.2.19 SWC - Spoken Wikipedia Corpora

14.2.20 Tatoeba

14.2.21 TIMIT DARPA Acoustic-Phonetic Continuous Speech Corpus

14.2.22 TUDA German Distant Speech

14.2.23 Urbansound8k

14.2.24 VoxForge

14.2.25 Wav2Letter

CHAPTER 15

audiomate.corpus.subset

This module contains functionality for creating any kind of subsets from a corpus. A subset of a corpus is represented with a *Subview*. The data contained in a subview is defined by one or more *FilterCriterion*.

For creating subviews there are additional classes. *Splitter* can be used to divide a corpus into subsets according to given proportions. *SubsetGenerator* can be used to create subset with given settings.

15.1 Subview

class audiomate.corpus.subset.**Subview**(*corpus*, *filter_criteria*)

A subview is a readonly layer representing some subset of a corpus. The assets the subview contains are defined by filter criteria. Only if an utterance passes all filter criteria it is contained in the subview.

Parameters

- **corpus** (CorpusView) – The corpus this subview is based on.
- **filter_criteria** (*list*, FilterCriterion) – List of *FilterCriterion*

Example:

```
>>> filter = subview.MatchingUtteranceIdxFilter(utterance_idxs=(['utt-1', 'utt-3
↪']))
>>> corpus = audiomate.corpus.load('path/to/corpus')
>>> corpus.num_utterances
14
>>> subset = subview.Subview(self.corpus, filter_criteria=[filter])
>>> subset.num_utterances
2
```

all_label_values(*label_list_ids=None*)

Return a set of all label-values occurring in this corpus.

Parameters label_list_ids (*list*) – If not None, only labels from label-lists with an id contained in this list are considered.

Returns A set of distinct label-values.

Return type `set`

`all_tokens`(*delimiter=' '*, *label_list_ids=None*)
Return a list of all tokens occurring in one of the labels in the corpus.

Parameters

- **`delimiter`** (*`str`*) – The delimiter used to split labels into tokens. (see *`audiomate.annotations.Label.tokenized()`*)
- **`label_list_ids`** (*`list`*) – If not `None`, only labels from label-lists with an idx contained in this list are considered.

Returns A set of distinct tokens.

Return type `set`

`contains_issuer`(*issuer*)
Return `True` if the given issuer is in the corpus already, `False` otherwise.

`contains_track`(*track*)
Return `True` if the given track is in the corpus already, `False` otherwise.

`feature_containers`
Return the feature-containers in the corpus.

Returns

A dictionary containing `audiomate.container.FeatureContainer` objects with the feature-idx as key.

Return type dict

`issuers`
Return the issuers in the corpus.

Returns

A dictionary containing ***`audiomate.issuers.Issuer`*** objects with the issuer-idx as key.

Return type dict

`label_count`(*label_list_ids=None*)
Return a dictionary containing the number of times, every label-value in this corpus is occurring.

Parameters **`label_list_ids`** (*`list`*) – If not `None`, only labels from label-lists with an id contained in this list are considered.

Returns

A dictionary containing the number of occurrences with the label-value as key.

Return type dict

`label_durations`(*label_list_ids=None*)
Return a dictionary containing the total duration, every label-value in this corpus is occurring.

Parameters **`label_list_ids`** (*`list`*) – If not None, only labels from label-lists with an id contained in this list are considered.

Returns

A dictionary containing the total duration with the label-value as key.

Return type dict

`name`
Return the name of the dataset (Equals basename of the path, if not None).

`num_feature_containers`
Return the number of feature-containers in the corpus.

`num_issuers`
Return the number of issuers in the corpus.

`num_subviews`
Return the number of subviews in the corpus.

`num_tracks`
Return number of tracks.

`num_utterances`
Return number of utterances.

`classmethod parse`(*representation*, *corpus=None*)
Creates a subview from a string representation (created with `self.serialize`).

Parameters **`representation`** (*`str`*) – The representation.

Returns The created subview.

Return type *Subview*

`serialize`()
Return a string representing the subview with all of its filter criteria.

Returns String with subview definition.

Return type str

`split_utterances_to_max_time`(*max_time=60.0*, *overlap=0.0*)
Create a new corpus, where all the utterances are of given maximal duration. Utterance longer than `max_time` are split up into multiple utterances.

Warning: Subviews and FeatureContainers are not added to the newly create corpus.

Parameters

- **`max_time`** (*`float`*) – Maximal duration for target utterances in seconds.
- **`overlap`** (*`float`*) – Amount of overlap in seconds. The overlap is measured from the center of the splitting. (The actual overlap of two segments is 2 * overlap)

Returns A new corpus instance.

Return type *Corpus*

`stats`()
Return statistics calculated overall samples of all utterances in the corpus.

Returns

A DataStats object containing statistics overall samples in the corpus.

Return type DataStats

stats_per_utterance()
Return statistics calculated for all samples of each utterance in the corpus.

Returns A dictionary containing a DataStats object for each utt.

Return type dict

subviews
Return the subviews of the corpus.

Returns

A dictionary containing `audiomate.corpus.Subview` objects with the subview-idx as key.

Return type dict

total_duration
Return the total amount of audio summed over all utterances in the corpus in seconds.

tracks
Return the tracks in the corpus.

Returns

A dictionary containing `audiomate.track.Track` objects with the track-idx as key.

Return type dict

utterances
Return the utterances in the corpus.

Returns

A dictionary containing `audiomate.corpus.assets.Utterance` objects with the utterance-idx as key.

Return type dict

15.2 Filter

class audiomate.corpus.subset.**FilterCriterion**
A filter criterion decides wheter a given utterance contained in a given corpus matches the filter.

match(*utterance*, *corpus*)
Check if the utterance matches the filter.

Parameters

- **utterance** (`Utterance`) – The utterance to match.
- **corpus** (`CorpusView`) – The corpus that contains the utterance.

Returns True if the filter matches the utterance, False otherwise.

Return type bool

classmethod name()
Returns a name identifying this type of filter criterion.

classmethod parse(*representation*)
Create a filter criterion based on a string representation (created with `serialize`).

Parameters **representation** (*str*) – The string representation.

Returns The filter criterion from that representation.

Return type *FilterCriterion*

serialize()
Serialize this filter criterion to write to a file. The output needs to be a single line without line breaks.

Returns A string representing this filter criterion.

Return type str

15.2.1 MatchingUtteranceIdxFilter

class audiomate.corpus.subset.**MatchingUtteranceIdxFilter**(*utterance_idxs*, *inverse=False*)
A filter criterion that matches utterances based on utterance-ids.

Parameters

- **utterance_idxs** (set) – A list of utterance-ids. Only utterances in the list will pass the filter
- **inverse** (*bool*) – If True only utterance not in the list pass the filter.

15.2.2 MatchingLabelFilter

class audiomate.corpus.subset.**MatchingLabelFilter**(*labels*, *label_list_ids=None*)
A filter criterion that only accepts utterances which only have the given labels.

Parameters

- **labels** (set) – A set of labels which are accepted.
- **label_list_ids** (set) – Only check label-lists with these ids. If None, checks all label-lists.

15.3 Splitter

class audiomate.corpus.subset.**Splitter**(*corpus*, *random_seed=None*)
A splitter provides methods for splitting a corpus into different subsets. It provides different approaches for splitting the corpus. (Methods indicated by split_by_) These methods mostly take some proportions parameter, which defines how big (in relation) the subsets should be. The subsets are returned as audiomate.corpus.Subview.

Parameters

- **corpus** (Corpus) – The corpus that should be splitted.
- **random_seed** (*int*) – Seed to use for random number generation.

split(*proportions*, *separate_issuers=False*)
Split the corpus based on the number of utterances. The utterances are distributed to *len(proportions)* subsets, according to the ratios *proportions[subset]*.

Parameters

- **proportions** (*dict*) – A dictionary containing the relative size of the target subsets. The key is an identifier for the subset.

- **separate_issuers** (*bool*) – If True it makes sure that all utterances of an issuer are in the same subset.

Returns

A dictionary containing the subsets with the identifier from the input as key.

Return type (dict)

Example:

```
>>> spl = Splitter(corpus)
>>> corpus.num_utterances
100
>>> subsets = spl.split(proportions={
>>>     "train" : 0.6,
>>>     "dev" : 0.2,
>>>     "test" : 0.2
>>> })
>>> print(subsets)
{'dev': <audiomate.corpus.subview.Subview at 0x104ce7400>,
'test': <audiomate.corpus.subview.Subview at 0x104ce74e0>,
'train': <audiomate.corpus.subview.Subview at 0x104ce7438>}
>>> subsets['train'].num_utterances
60
>>> subsets['dev'].num_utterances
20
>>> subsets['test'].num_utterances
20
```

split_by_audio_duration(*proportions*, *separate_issuers=False*)

Split the corpus based on the the total duration of audio. The utterances are distributed to *len(proportions)* subsets. Utterances are split up in a way that each subset contains audio with a duration proportional to the given proportions.

Parameters

- **proportions** (*dict*) – A dictionary containing the relative size of the target subsets. The key is an identifier for the subset.
- **separate_issuers** (*bool*) – If True it makes sure that all utterances of an issuer are in the same subset.

Returns

A dictionary containing the subsets with the identifier from the input as key.

Return type (dict)

Example:

```
>>> spl = Splitter(corpus)
>>> corpus.num_utterances
100
>>> subsets = spl.split_by_audio_duration(proportions={
>>>     "train" : 0.6,
>>>     "dev" : 0.2,
>>>     "test" : 0.2
>>> })
>>> print(subsets)
{'dev': <audiomate.corpus.subview.Subview at 0x104ce7400>,
```

(continues on next page)

(continued from previous page)

```
'test': <audiomate.corpus.subview.Subview at 0x104ce74e0>,
'train': <audiomate.corpus.subview.Subview at 0x104ce7438>}
>>> subsets['train'].num_utterances
55
>>> subsets['dev'].num_utterances
35
>>> subsets['test'].num_utterances
10
```

split_by_label_duration(*proportions*, *separate_issuers=False*)

Split the corpus based on the total duration of labels (end - start). The utterances are distributed to *len(proportions)* subsets. Utterances are split up in a way that each subset contains labels with a duration proportional to the given proportions.

Parameters

- **proportions** (*dict*) – A dictionary containing the relative size of the target subsets. The key is an identifier for the subset.
- **separate_issuers** (*bool*) – If True it makes sure that all utterances of an issuer are in the same subset.

Returns

A dictionary containing the subsets with the identifier from the input as key.

Return type (dict)

Example:

```
>>> spl = Splitter(corpus)
>>> corpus.num_utterances
100
>>> subsets = spl.split_by_label_duration(proportions={
>>>     "train" : 0.6,
>>>     "dev" : 0.2,
>>>     "test" : 0.2
>>> })
>>> print(subsets)
{'dev': <audiomate.corpus.subview.Subview at 0x104ce7400>,
'test': <audiomate.corpus.subview.Subview at 0x104ce74e0>,
'train': <audiomate.corpus.subview.Subview at 0x104ce7438>}
>>> subsets['train'].num_utterances
55
>>> subsets['dev'].num_utterances
35
>>> subsets['test'].num_utterances
10
```

split_by_label_length(*proportions*, *label_list_idx=None*, *separate_issuers=False*)

Split the corpus based on the the total length of the label-list. The utterances are distributed to *len(proportions)* subsets. Utterances are split up in a way that each subset contains labels summed up to a length proportional to the given proportions. Length is defined as the number of characters.

Parameters

- **proportions** (*dict*) – A dictionary containing the relative size of the target subsets. The key is an identifier for the subset.

- **label_list_idx** (*str*) – The idx of the label-list to use for compute the length. If *None* all label-lists are used.
- **separate_issuers** (*bool*) – If True it makes sure that all utterances of an issuer are in the same subset.

Returns

A dictionary containing the subsets with the identifier from the input as key.

Return type (dict)

Example:

```
>>> spl = Splitter(corpus)
>>> corpus.num_utterances
100
>>> subsets = spl.split_by_label_length(proportions={
>>>     "train" : 0.6,
>>>     "dev" : 0.2,
>>>     "test" : 0.2
>>> })
>>> print(subsets)
{'dev': <audiomate.corpus.subview.Subview at 0x104ce7400>,
'test': <audiomate.corpus.subview.Subview at 0x104ce74e0>,
'train': <audiomate.corpus.subview.Subview at 0x104ce7438>}
>>> subsets['train'].num_utterances
55
>>> subsets['dev'].num_utterances
35
>>> subsets['test'].num_utterances
10
```

split_by_label_occurence (*proportions*, *separate_issuers=False*)

Split the corpus based on the total number of occcurences of labels. The utterances are distributed to *len(proportions)* subsets. Utterances are split up in a way that each subset contains labels-occurences proportional to the given proportions.

Parameters

- **proportions** (*dict*) – A dictionary containing the relative size of the target subsets. The key is an identifier for the subset.
- **separate_issuers** (*bool*) – If True it makes sure that all utterances of an issuer are in the same subset.

Returns

A dictionary containing the subsets with the identifier from the input as key.

Return type (dict)

Example:

```
>>> spl = Splitter(corpus)
>>> corpus.num_utterances
100
>>> subsets = spl.split_by_label_occurence(proportions={
>>>     "train" : 0.6,
>>>     "dev" : 0.2,
>>>     "test" : 0.2
>>> })
```

(continues on next page)

(continued from previous page)

```
>>> print(subsets)
{'dev': <audiomate.corpus.subview.Subview at 0x104ce7400>,
'test': <audiomate.corpus.subview.Subview at 0x104ce74e0>,
'train': <audiomate.corpus.subview.Subview at 0x104ce7438>}
>>> subsets['train'].num_utterances
55
>>> subsets['dev'].num_utterances
35
>>> subsets['test'].num_utterances
10
```

15.4 SubsetGenerator

class audiomate.corpus.subset.**SubsetGenerator**(*corpus*, *random_seed=None*)

This class is used to generate subsets of a corpus.

Parameters

- **corpus** (Corpus) – The corpus to create subsets from.
- **random_seed** (*int*) – Seed to use for random number generation.

maximal_balanced_subset(*by_duration=False*, *label_list_ids=None*)

Create a subset of the corpus as big as possible, so that the labels are balanced approximately. The label with the shortest duration (or with the fewest utterance if by_duration=False) is taken as reference. All other labels are selected so they match the shortest one as far as possible.

Parameters

- **by_duration** (*bool*) – If True the size measure is the duration of all utterances in a subset/corpus.
- **label_list_ids** (*list*) – List of label-list ids. If none is given, all label-lists are considered for balancing. Otherwise only the ones that are in the list are considered.

Returns The subview representing the subset.

Return type *Subview*

random_subset(*relative_size*, *balance_labels=False*, *label_list_ids=None*)

Create a subview of random utterances with a approximate size relative to the full corpus. By default x random utterances are selected with x equal to `relative_size * corpus.num_utterances`.

Parameters

- **relative_size** (*float*) – A value between 0 and 1. (0.5 will create a subset with approximately 50% of the full corpus size)
- **balance_labels** (*bool*) – If True, the labels of the selected utterances are balanced as far as possible. So the count/duration of every label within the subset is equal.
- **label_list_ids** (*list*) – List of label-list ids. If none is given, all label-lists are considered for balancing. Otherwise only the ones that are in the list are considered.

Returns The subview representing the subset.

Return type *Subview*

random_subset_by_duration(*relative_duration*, *balance_labels=False*, *label_list_ids=None*)
Create a subview of random utterances with a approximate duration relative to the full corpus. Random utterances are selected so that the sum of all utterance durations equals to the relative duration of the full corpus.

Parameters

- **relative_duration** (*float*) – A value between 0 and 1. (e.g. 0.5 will create a subset with approximately 50% of the full corpus duration)
- **balance_labels** (*bool*) – If True, the labels of the selected utterances are balanced as far as possible. So the count/duration of every label within the subset is equal.
- **label_list_ids** (*list*) – List of label-list ids. If none is given, all label-lists are considered for balancing. Otherwise only the ones that are in the list are considered.

Returns The subview representing the subset.

Return type *Subview*

random_subsets(*relative_sizes*, *by_duration=False*, *balance_labels=False*, *label_list_ids=None*)
Create a bunch of subsets with the given sizes relative to the size or duration of the full corpus. Basically the same as calling `random_subset` or `random_subset_by_duration` multiple times with different values. But this method makes sure that every subset contains only utterances, that are also contained in the next bigger subset.

Parameters

- **relative_sizes** (*list*) – A list of numbers between 0 and 1 indicating the sizes of the desired subsets, relative to the full corpus.
- **by_duration** (*bool*) – If True the size measure is the duration of all utterances in a subset/corpus.
- **balance_labels** (*bool*) – If True the labels contained in a subset are chosen to be balanced as far as possible.
- **label_list_ids** (*list*) – List of label-list ids. If none is given, all label-lists are considered for balancing. Otherwise only the ones that are in the list are considered.

Returns A dictionary containing all subsets with the relative size as key.

Return type dict

15.5 Utils

`audiomate.corpus.subset.utils.`**absolute_proportions**(*proportions*, *count*)
Split a given integer into n parts according to len(proportions) so they sum up to count and match the given proportions.

Parameters **proportions** (*dict*) – Dict of proportions, with a identifier as key.

Returns Dictionary with absolute proportions and same identifiers as key.

Return type dict

Example:

```
>>> absolute_proportions({'train': 0.5, 'test': 0.5}, 100)
{'train': 50, 'test': 50}
```

audiomate.corpus.subset.utils.**get_identifiers_splitted_by_weights**(*identifiers*, *proportions*, *seed=None*)

Divide the given identifiers based on the given proportions. But instead of randomly split the identifiers it is based on category weights. Every identifier has a weight for any number of categories. The target is, to split the identifiers in a way, so the sum of category k within part x is proportional to the sum of category x over all parts according to the given proportions. This is done by greedily insert the identifiers step by step in a part which has free space (weight). If there are no fitting parts anymore, the one with the least weight exceed is used. This function is deterministic, given the same seed. First the identifiers are sorted before shuffled using the given seed.

Parameters

- **identifiers** (*dict*) – A dictionary containing the weights for each identifier (key). Per item a dictionary of weights per category is given.
- **proportions** (*dict*) – Dict of proportions, with a identifier as key.
- **seed** (*int*) – Seed to use for random operations.

Returns Dictionary containing a list of identifiers per part with the same key as the proportions dict.

Return type dict

Example:

```
>>> identifiers = {
>>>     'a': {'music': 2, 'speech': 1},
>>>     'b': {'music': 5, 'speech': 2},
>>>     'c': {'music': 2, 'speech': 4},
>>>     'd': {'music': 1, 'speech': 4},
>>>     'e': {'music': 3, 'speech': 4}
>>> }
>>> proportions = {
>>>     "train" : 0.6,
>>>     "dev" : 0.2,
>>>     "test" : 0.2
>>> }
>>> get_identifiers_splitted_by_weights(identifiers, proportions)
{
    'train': ['a', 'b', 'd'],
    'dev': ['c'],
    'test': ['e']
}
```

audiomate.corpus.subset.utils.**select_balanced_subset**(*items*, *select_count*, *categories*, *select_count_values=None*, *seed=None*)

Select items so the summed category weights are balanced. Each item has a dictionary containing the category weights. Items are selected until `select_count` is reached. The value that is added to `select_count` for an item can be defined in the dictionary `select_count_values`. If this is not defined it is assumed to be 1, which means *select_count* items are selected.

Parameters

- **items** (*dict*) – Dictionary containing items with category weights.
- **select_count** (*float*) – Value to reach for selected items.
- **categories** (*list*) – List of all categories.

- **select_count_values** (*dict*) – The select_count values to be used. For example an utterance with multiple labels: The category weights (label-lengths) are used for balance, but the utterance-duration is used for reaching the select_count.

Returns List of item ids, containing `number_of_items` (or `len(items)` if smaller).

Return type list

Example

```
>>> items = {
>>>     'utt-1' : {'m': 1, 's': 0, 'n': 0},
>>>     'utt-2' : {'m': 0, 's': 2, 'n': 1},
>>>     ...
>>> }
>>> select_balanced_subset(items, 5)
>>> ['utt-1', 'utt-3', 'utt-9', 'utt-33', 'utt-34']
```

audiomate.corpus.subset.utils.**split_identifiers**(*identifiers*, *proportions*, *seed=None*)

Split the given identifiers by the given proportions. This function is deterministic, given the same seed. First the identifiers are sorted before shuffled using the given seed.

Parameters

- **identifiers** (*list*) – List of identifiers (str).
- **proportions** (*dict*) – A dictionary containing the proportions with the identifier from the
- **as key.** (*input*) –
- **seed** (*int*) – Seed to use for random operations.

Returns Dictionary containing a list of identifiers per part with the same key as the proportions dict.

Return type dict

Example:

```
>>> split_identifiers(
>>>     identifiers=['a', 'b', 'c', 'd'],
>>>     proportions={'melvin' : 0.5, 'timmy' : 0.5}
>>> )
{'melvin' : ['a', 'c'], 'timmy' : ['b', 'd']}
```

CHAPTER 16

audiomate.corpus.validation

This module contains functions for validating a corpus on different properties. e.g. if the length of the utterance is to short for its corresponding transcription.

`audiomate.corpus.validation.Validator` is the base class for performing validations. It can be extended to implement validators for specific tests/validations. Thre result of every validator has to be a `audiomate.corpus.validation.ValidationResult` or a subclass of it.

16.1 Base

class audiomate.corpus.validation.**Validator**

A validator is a class that tests a specific behaviour/state of a corpus.

name()

Return a name, identifying the task.

validate(*corpus_to_validate*)

Perform the validation on the given corpus.

Parameters **corpus** (`Corpus`) – The corpus to test/validate.

Returns

The result containing at least the pass/fail indication.

Return type *ValidationResult*

class audiomate.corpus.validation.**ValidationResult**(*passed*, *name='Validation'*, *info=None*)

Representation of the result of a validation. The basic result just indicates a pass or fail. Depending on the validator it can be extended to hold more information (e.g. utterance-ids which triggered the task to fail).

Parameters

- **passed** (*bool*) – A boolean indicating, if the validation has passed (`True`) or failed (`False`).

- **name** (*str*) – The name of the validator, that produced the result.
- **info** (*dict*) – Dictionary containing key/value string-pairs with detailed information of the validation. For example id of the label-list that was validated.

get_report()
Return a string containing a report of the result. This can used to print or save to a text file.

Returns String containing infos about the result

Return type str

class audiomate.corpus.validation.**InvalidItemsResult**(*passed*, *invalid_items*, *name='Validation'*, *item_name='Utterances'*, *info=None*)
A generic result class for validators that return a list of items (utterances, tracks) that were classified invalid. Besides the id of the item, a reason may be appended.

Parameters

- **passed** (*bool*) – A boolean indicating, if the validation has passed (`True`) or failed (`False`).
- **invalid_items** (*dict*) – A dictionary containing item-ids, that are invalid. The values are reasons why they are invalid.
- **name** (*str*) – The name of the validator, that produced the result.
- **info** (*dict*) – Dictionary containing key/value string-pairs with detailed information of the validation. For example id of the label-list that was validated.

get_report()
Return a string containing a report of the result. This can used to print or save to a text file.

Returns String containing infos about the result

Return type str

16.2 Combination

class audiomate.corpus.validation.**CombinedValidator**(*validators=None*)
The CombinedValidator is used to execute multiple validators at once.

Parameters **validators** (*list*) – A list of validators that are executed.

name()
Return a name, identifying the task.

validate(*corpus_to_validate*)
Perform validation on the given corpus.

Parameters **corpus** (`Corpus`) – The corpus to test/validate.

class audiomate.corpus.validation.**CombinedValidationResult**(*passed*, *results=None*, *info=None*)
Result of running multiple validation-tasks with the validator.

Parameters

- **passed** (*bool*) – A boolean, indicating if all tasks have passed (True) or at least one failed (False).

- **results** (*dict*) – A dictionary containing the results of all validators, with the task name as key.
- **info** (*dict*) – Dictionary containing key/value string-pairs with detailed information of the validation. For example id of the label-list that was validated.

get_report()

Return a string containing a report of the result. This can used to print or save to a text file.

Returns String containing infos about the result

Return type str

16.3 Label-List

class audiomate.corpus.validation.**UtteranceTranscriptionRatioValidator**(*max_characters_per_second label_list_idx='word-transcript', num_threads=1*)

Checks if the ratio between utterance-duration and transcription-length is below a given ratio. This is used to find utterances where the speech transcription is to long for a given utterance, meaning too much characters per second.

Parameters

- **max_characters_per_second** (*int*) – If char/sec of an utterance is higher than this it is returned.
- **label_list_idx** (*str*) – The label-list to use for validation.
- **num_threads** (*int*) – Number of threads to use.

name()

Return a name, identifying the task.

validate(*corpus_to_validate*)

Perform the validation on the given corpus.

Parameters **corpus_to_validate** (Corpus) – The corpus to test/validate.

Returns Validation result.

Return type *InvalidItemsResult*

class audiomate.corpus.validation.**LabelCountValidator**(*min_number_of_labels=1, label_list_idx='word-transcript'*)

Checks if every utterance contains a label-list with the given id and has at least *min_number_of_labels*.

Parameters

- **min_number_of_labels** (*int*) – Minimum number of expected labels.
- **label_list_idx** (*str*) – The label-list to use for validation.

name()

Return a name, identifying the task.

validate(*corpus_to_validate*)

Perform the validation on the given corpus.

Parameters **corpus_to_validate** (Corpus) – The corpus to test/validate.

Returns Validation result.

Return type *InvalidItemsResult*

class `audiomate.corpus.validation.`**`LabelCoverageValidator`**(*label_list_idx*, *threshold=0.01*)

Check if every portion of the utterance is covered with at least one label. The validator returns segments (start, end) of an utterance, where no label is defined within the given label-list.

Parameters

- **`label_list_idx`** (*`str`*) – The idx of the label-list to check.
- **`threshold`** (*`float`*) – A threshold for the length of a segment to be considered as uncovered.

`name`()

Return a name, identifying the task.

`validate`(*corpus_to_validate*)

Perform the validation on the given corpus.

Parameters **`corpus_to_validate`** (`Corpus`) – The corpus to test/validate.

Returns Validation result.

Return type *LabelCoverageValidationResult*

`validate_utterance`(*utterance*)

Validate the given utterance and return a list of uncovered segments (start, end).

class `audiomate.corpus.validation.`**`LabelCoverageValidationResult`**(*passed*, *uncovered_segments*, *name*, *info=None*)

Result of a the *`LabelCoverageValidator`*.

Parameters

- **`passed`** (*`bool`*) – A boolean indicating, if the validation has passed (`True`) or failed (`False`).
- **`uncovered_segments`** (*`dict`*) – A dictionary containing a list of uncovered segments for every utterance.
- **`name`** (*`str`*) – The name of the validator, that produced the result.
- **`info`** (*`dict`*) – Dictionary containing key/value string-pairs with detailed information of the validation. For example id of the label-list that was validated.

`get_report`()

Return a string containing a report of the result. This can used to print or save to a text file.

Returns String containing infos about the result

Return type str

class `audiomate.corpus.validation.`**`LabelOverflowValidator`**(*label_list_idx*, *threshold=0.01*)

Check if all labels are within the boundaries of an utterance. Finds all segments of labels that lie outside of an utterance.

Parameters

- **`label_list_idx`** (*`str`*) – The idx of the label-list to check.
- **`threshold`** (*`float`*) – A threshold for a time distance to be considered for an overflow.

name()
Return a name, identifying the task.

validate(*corpus_to_validate*)
Perform the validation on the given corpus.

Parameters **corpus_to_validate** (Corpus) – The corpus to test/validate.

Returns Validation result.

Return type *LabelOverflowValidationResult*

validate_utterance(*utterance*)
Validate the given utterance and return a list of segments (start, end, label-value), that are outside of the utterance.

class audiomate.corpus.validation.**LabelOverflowValidationResult**(*passed*, *overflow_segments*, *name*, *info=None*)
Result of a the *LabelOverflowValidator*.

Parameters

- **passed** (*bool*) – A boolean indicating, if the validation has passed (True) or failed (False).
- **overflow_segments** (*dict*) – A dictionary containing a list of overflowing segments for every utterance.
- **name** (*str*) – The name of the validator, that produced the result.
- **info** (*dict*) – Dictionary containing key/value string-pairs with detailed information of the validation. For example id of the label-list that was validated.

get_report()
Return a string containing a report of the result. This can used to print or save to a text file.

Returns String containing infos about the result

Return type str

16.4 Track

class audiomate.corpus.validation.**TrackReadValidator**(*num_workers=1*)
Check if the track can be opened and read. By reading the first few samples.

name()
Return a name, identifying the task.

validate(*corpus_to_validate*)
Perform the validation on the given corpus.

Parameters **corpus** (Corpus) – The corpus to test/validate.

Returns Validation result.

Return type *InvalidItemsResult*

CHAPTER 17

audiomate.corpus.conversion

This module contains classes to convert the data of a corpus. It is for example used to convert all audio data to wav files.

17.1 Audio File Conversion

class `audiomate.corpus.conversion.`**`AudioFileConverter`**(*sampling_rate=16000*, *separate_file_per_utterance=False*, *force_conversion=False*)

Base class for converters that convert all audio to a specific format. A converter creates a new instance of a corpus, so that all audio files meet given requirements.

Parameters

- **`sampling_rate`** (*`int`*) – Target sampling rate to convert audio to.
- **`separate_file_per_utterance`** (*`bool`*) – If `True`, every utterance in the resulting corpus is in a separate file. If `False`, the file/utt structure will be preserved.
- **`force_conversion`** (*`bool`*) – If `True`, all utterances will be converted whether or not it already matches the target format. If `False`, only utterances not matching the target format will be converted. Others are reference to the original files.

`convert`(*corpus*, *target_audio_path*)

Convert the given corpus.

Parameters

- **`corpus`** (`Corpus`) – The input corpus.
- **`target_audio_path`** (*`str`*) – The path where the audio files of the converted corpus should be saved.

Returns The newly created corpus.

Return type *Corpus*

class audiomate.corpus.conversion.**WavAudioFileConverter**(*num_workers=4*, *sampling_rate=16000*, *separate_file_per_utterance=False*, *force_conversion=False*)

Class that creates a new instance of a corpus, so that all audio files meet given requirements.

convert(*corpus*, *target_audio_path*)

Convert the given corpus.

Parameters

- **corpus** (Corpus) – The input corpus.
- **target_audio_path** (*str*) – The path where the audio files of the converted corpus should be saved.

Returns The newly created corpus.

Return type *Corpus*

CHAPTER 18

audiomate.processing

The processing module provides tools for processing audio data in a batch-wise manner. The idea is to setup a predefined tool that can process all the audio from a corpus.

The basic component is the *audiomate.processing.Processor*. It provides the functionality to reduce any input component like a corpus, feature-container, utterance, file to the abstraction of frames. A concrete implementation then only has to provide the proper method to process these frames.

Often in audio processing the same components are used in combination with others. For this purpose a pipeline can be built that processes the frames in multiple steps. The *audiomate.processing.pipeline* provides the *audiomate.processing.pipeline.Computation* and *audiomate.processing.pipeline.Reduction* classes. These abstract classes can be extended to create processing components of a pipeline. The different components are then be coupled to create custom pipelines.

18.1 Processor

class `audiomate.processing.`**`Processor`**

The processor base class provides the functionality to process audio data on different levels (Corpus, Utterance, Track). For every level there is an offline and an online method. In the offline mode the data is processed in one step (e.g. the whole track/utterance at once). This means the `process_frames` method is called with all the frames of the track/utterance. In online mode the data is processed in chunks, so the `process_frames` method is called multiple times per track/utterance with different chunks.

To implement a concrete processor the `process_frames` method has to be implemented. This method is called in online and offline mode. So it is up to the user to determine if a processor can be called in either online or offline mode, maybe both. This differs between use cases.

If the implementation of a processor does change the frame or hop-size, it is expected to provide a transform via the `frame_transform` method. Frame-size and hop-size are measured in samples regarding the original audio signal (or simply its sampling rate).

`frame_transform`(*frame_size*, *hop_size*)

If the processor changes the number of samples that build up a frame or the number of samples between two consecutive frames (hop-size), this function needs transform the original frame- and/or hop-size.

This is used to store the frame-size and hop-size in a feature-container. In the end one can calculate start and end time of a frame with this information.

By default it is assumed that the processor doesn't change the frame-size and the hop-size.

Parameters

- **frame_size** (*int*) – The original frame-size.
- **hop_size** (*int*) – The original hop-size.

Returns The (frame-size, hop-size) after processing.

Return type tuple

process_corpus (*corpus*, *output_path*, *frame_size=400*, *hop_size=160*, *sr=None*)
Process all utterances of the given corpus and save the processed features in a feature-container. The utterances are processed in **offline** mode so the full utterance in one go.

Parameters

- **corpus** (Corpus) – The corpus to process the utterances from.
- **output_path** (*str*) – A path to save the feature-container to.
- **frame_size** (*int*) – The number of samples per frame.
- **hop_size** (*int*) – The number of samples between two frames.
- **sr** (*int*) – Use the given sampling rate. If None uses the native sampling rate from the underlying data.

Returns The feature-container containing the processed features.

Return type *FeatureContainer*

process_corpus_online (*corpus*, *output_path*, *frame_size=400*, *hop_size=160*, *chunk_size=1*, *buffer_size=5760000*)
Process all utterances of the given corpus and save the processed features in a feature-container. The utterances are processed in **online** mode, so chunk by chunk.

Parameters

- **corpus** (Corpus) – The corpus to process the utterances from.
- **output_path** (*str*) – A path to save the feature-container to.
- **frame_size** (*int*) – The number of samples per frame.
- **hop_size** (*int*) – The number of samples between two frames.
- **chunk_size** (*int*) – Number of frames to process per chunk.
- **buffer_size** (*int*) – Number of samples to load into memory at once. The exact number of loaded samples depends on the block-size of the audioread library. So it can be of block-size higher, where the block-size is typically 1024 or 4096.

Returns The feature-container containing the processed features.

Return type *FeatureContainer*

process_features (*corpus*, *input_features*, *output_path*)
Process all features of the given corpus and save the processed features in a feature-container. The features are processed in **offline** mode, all features of an utterance at once.

Parameters

- **corpus** (Corpus) – The corpus to process the utterances from.

- **input_features** (`FeatureContainer`) – The feature-container to process the frames from.
- **output_path** (*str*) – A path to save the feature-container to.

Returns The feature-container containing the processed features.

Return type *FeatureContainer*

process_features_online (*corpus*, *input_features*, *output_path*, *chunk_size=1*)
Process all features of the given corpus and save the processed features in a feature-container. The features are processed in **online** mode, chunk by chunk.

Parameters

- **corpus** (`Corpus`) – The corpus to process the utterances from.
- **input_features** (`FeatureContainer`) – The feature-container to process the frames from.
- **output_path** (*str*) – A path to save the feature-container to.
- **chunk_size** (*int*) – Number of frames to process per chunk.

Returns The feature-container containing the processed features.

Return type *FeatureContainer*

process_frames (*data*, *sampling_rate*, *offset=0*, *last=False*, *utterance=None*, *corpus=None*)
Process the given chunk of frames. Depending on online or offline mode, the given chunk is either the full data or just part of it.

Parameters

- **data** (*np.ndarray*) – nD Array of frames (num-frames x frame-dimensions).
- **sampling_rate** (*int*) – The sampling rate of the underlying signal.
- **offset** (*int*) – The index of the first frame in the chunk. In offline mode always 0. (Relative to the first frame of the utterance/sequence)
- **last** (*bool*) – True indicates that this is the last frame of the sequence/utterance. In offline mode always True.
- **utterance** (`Utterance`) – The utterance the frame is from, if available.
- **corpus** (`Corpus`) – The corpus the frame is from, if available.

Returns The processed frames.

Return type np.ndarray

process_track (*track*, *frame_size=400*, *hop_size=160*, *sr=None*, *start=0*, *end=inf*, *utterance=None*, *corpus=None*)
Process the track in **offline** mode, in one go.

Parameters

- **track** (`Track`) – The track to process.
- **frame_size** (*int*) – The number of samples per frame.
- **hop_size** (*int*) – The number of samples between two frames.
- **sr** (*int*) – Use the given sampling rate. If `None`, uses the native sampling rate from the underlying data.
- **start** (*float*) – The point within the track in seconds, to start processing from.

- **end** (*float*) – The point within the track in seconds, to end processing.
- **utterance** (Utterance) – The utterance that is associated with this track, if available.
- **corpus** (Corpus) – The corpus this track is part of, if available.

Returns The processed features.

Return type np.ndarray

process_track_online(*track*, *frame_size=400*, *hop_size=160*, *start=0*, *end=inf*, *utterance=None*, *corpus=None*, *chunk_size=1*, *buffer_size=5760000*)

Process the track in **online** mode, chunk by chunk. The processed chunks are yielded one after another.

Parameters

- **track** (Track) – The track to process.
- **frame_size** (*int*) – The number of samples per frame.
- **hop_size** (*int*) – The number of samples between two frames.
- **start** (*float*) – The point within the track in seconds to start processing from.
- **end** (*float*) – The point within the trac in seconds to end processing.
- **utterance** (Utterance) – The utterance that is associated with this track, if available.
- **corpus** (Corpus) – The corpus this track is part of, if available.
- **chunk_size** (*int*) – Number of frames to process per chunk.
- **buffer_size** (*int*) – Number of samples to load into memory at once. The exact number of loaded samples depends on the type of track. It can be of block-size higher, where the block-size is typically 1024 or 4096.

Returns A generator that yield processed chunks.

Return type Generator

process_utterance(*utterance*, *frame_size=400*, *hop_size=160*, *sr=None*, *corpus=None*)

Process the utterance in **offline** mode, in one go.

Parameters

- **utterance** (Utterance) – The utterance to process.
- **frame_size** (*int*) – The number of samples per frame.
- **hop_size** (*int*) – The number of samples between two frames.
- **sr** (*int*) – Use the given sampling rate. If None uses the native sampling rate from the underlying data.
- **corpus** (Corpus) – The corpus this utterance is part of, if available.

Returns The processed features.

Return type np.ndarray

process_utterance_online(*utterance*, *frame_size=400*, *hop_size=160*, *chunk_size=1*, *buffer_size=5760000*, *corpus=None*)

Process the utterance in **online** mode, chunk by chunk. The processed chunks are yielded one after another.

Parameters

- **utterance** (Utterance) – The utterance to process.
- **frame_size** (*int*) – The number of samples per frame.

- **hop_size** (*int*) – The number of samples between two frames.
- **chunk_size** (*int*) – Number of frames to process per chunk.
- **buffer_size** (*int*) – Number of samples to load into memory at once. The exact number of loaded samples depends on the block-size of the audioread library. So it can be of block-size higher, where the block-size is typically 1024 or 4096.
- **corpus** (Corpus) – The corpus this utterance is part of, if available.

Returns A generator that yield processed chunks.

Return type Generator

18.2 Pipeline

This module contains classes for creating frame processing pipelines.

A pipeline consists of one of two types of steps. A computation step takes data from a previous step or the input and processes it. A reduction step is used to merge outputs of multiple previous steps. It takes outputs of all incoming steps and outputs a single data block.

The steps are managed as a directed graph, which is built by passing the parent steps to the `__init__` method of a step. Every step that is created has his own graph, but inherits all nodes and edges of the graphs of his parent steps.

Every pipeline represents a processor and implements the `process_frames` method.

class `audiomate.processing.pipeline.`**Chunk**(*data*, *offset*, *is_last*, *left_context=0*, *right_context=0*)

Represents a chunk of data. It is used to pass data between different steps of a pipeline.

Parameters

- **data** (*np.ndarray or list*) – A single array of frames or a list of separate chunks of frames of equal size.
- **offset** (*int*) – The index of the first frame in the chunk within the sequence.
- **is_last** (*bool*) – Whether this is the last chunk of the sequence.
- **left_context** (*int*) – Number of frames that act as context at the begin of the chunk (left).
- **right_context** (*int*) – Number of frames that act as context at the end of the chunk (right).

class `audiomate.processing.pipeline.`**Step**(*name=None*, *min_frames=1*, *left_context=0*, *right_context=0*)

This class is the base class for a step in a processing pipeline.

It handles the procedure of executing the pipeline. It makes sure the steps are computed in the correct order. It also provides the correct inputs to every step.

Every step has to provide a `compute` method which is the actual processing.

If the implementation of a step does change the frame or hop-size, it is expected to provide a transform via the `frame_transform_step` method. Frame-size and hop-size are measured in samples regarding the original audio signal (or simply its sampling rate).

Parameters **name** (*str, optional*) – A name for identifying the step.

compute(*chunk*, *sampling_rate*, *corpus=None*, *utterance=None*)

Do the computation of the step. If the step uses context, the result has to be returned without context.

Parameters

- **chunk** (Chunk) – The chunk containing data and info about context, offset, …
- **sampling_rate** (*int*) – The sampling rate of the underlying signal.
- **corpus** (Corpus) – The corpus the data is from, if available.
- **utterance** (Utterance) – The utterance the data is from, if available.

Returns The array of processed frames, without context.

Return type np.ndarray

frame_transform(*frame_size*, *hop_size*)

If the processor changes the number of samples that build up a frame or the number of samples between two consecutive frames (hop-size), this function needs transform the original frame- and/or hop-size.

This is used to store the frame-size and hop-size in a feature-container. In the end one can calculate start and end time of a frame with this information.

By default it is assumed that the processor doesn't change the frame-size and the hop-size.

Parameters

- **frame_size** (*int*) – The original frame-size.
- **hop_size** (*int*) – The original hop-size.

Returns The (frame-size, hop-size) after processing.

Return type tuple

frame_transform_step(*frame_size*, *hop_size*)

If the processor changes the number of samples that build up a frame or the number of samples between two consecutive frames (hop-size), this function needs transform the original frame- and/or hop-size.

This is used to store the frame-size and hop-size in a feature-container. In the end one can calculate start and end time of a frame with this information.

By default it is assumed that the processor doesn't change the frame-size and the hop-size.

Note: This function is simply for this step, whereas `frame_transform()` computes the transformation for the whole pipeline.

Parameters

- **frame_size** (*int*) – The original frame-size.
- **hop_size** (*int*) – The original hop-size.

Returns The (frame-size, hop-size) after processing.

Return type tuple

process_frames(*data*, *sampling_rate*, *offset=0*, *last=False*, *utterance=None*, *corpus=None*)

Execute the processing of this step and all dependent parent steps.

class audiomate.processing.pipeline.**Computation**(*parent=None*, *name=None*, *min_frames=1*, *left_context=0*, *right_context=0*)

Base class for a computation step. To implement a computation step for pipeline the `compute` method has to be implemented. This method gets the frames from its parent step including context frames if defined. It has to return the same number of frames but without context frames.

Parameters

- **`parent`** (`Step`, *`optional`*) – The parent step this step depends on.
- **`name`** (*`str`*, *`optional`*) – A name for identifying the step.

class `audiomate.processing.pipeline.`**`Reduction`**(*parents*, *name=None*, *min_frames=1*, *left_context=0*, *right_context=0*)

Base class for a reduction step. It gets the frames of all its parent steps as a list. It has to return a single chunk of frames.

Parameters

- **`parents`** (*`list`*) – List of parent steps this step depends on.
- **`name`** (*`str`*, *`optional`*) – A name for identifying the step.

18.3 Implementations

Some processing pipeline steps are already implemented.

Table 1: Implementations of processing pipeline steps.

Name	Description
MeanVarianceNorm	Normalizes features with given mean and variance.
MelSpectrogram	Exctracts MelSpectrogram features.
MFCC	Extracts MFCC features.
PowerToDb	Convert power spectrum to Db.
Delta	Compute delta features.
AddContext	Add previous and subsequent frames to the current frame.
Stack	Reduce multiple features into one by stacking them on top of each other.
AvgPool	Compute the average (per dimension) over a given number of sequential frames.
VarPool	Compute the variance (per dimension) over a given number of sequential frames.
OnsetStrength	Compute onset strengths.
Tempogram	Compute tempogram features.

class `audiomate.processing.pipeline.`**`MeanVarianceNorm`**(*mean*, *variance*, *parent=None*, *name=None*)

Pre-processing step to normalize mean and variance.

frame = (frame - mean) / sqrt(variance)

Parameters

- **`mean`** (*`float`*) – The mean to use for normalization.
- **`variance`** (*`float`*) – The variance to use for normalization.s

`compute`(*chunk*, *sampling_rate*, *corpus=None*, *utterance=None*)

Do the computation of the step. If the step uses context, the result has to be returned without context.

Parameters

- **`chunk`** (`Chunk`) – The chunk containing data and info about context, offset, . . .
- **`sampling_rate`** (*`int`*) – The sampling rate of the underlying signal.
- **`corpus`** (`Corpus`) – The corpus the data is from, if available.
- **`utterance`** (`Utterance`) – The utterance the data is from, if available.

Returns The array of processed frames, without context.

Return type np.ndarray

class `audiomate.processing.pipeline.`**`MelSpectrogram`**(*n_mels=128*, *parent=None*, *name=None*)

Computation step that extracts mel-spectrogram features from the given frames.

Based on http://librosa.github.io/librosa/generated/librosa.feature.melspectrogram.html

Parameters **`n_mels`** (*`int`*) – Number of mel bands to generate.

`compute`(*chunk*, *sampling_rate*, *corpus=None*, *utterance=None*)

Do the computation of the step. If the step uses context, the result has to be returned without context.

Parameters

- **`chunk`** (`Chunk`) – The chunk containing data and info about context, offset, . . .
- **`sampling_rate`** (*`int`*) – The sampling rate of the underlying signal.
- **`corpus`** (`Corpus`) – The corpus the data is from, if available.
- **`utterance`** (`Utterance`) – The utterance the data is from, if available.

Returns The array of processed frames, without context.

Return type np.ndarray

class `audiomate.processing.pipeline.`**`MFCC`**(*n_mfcc=13*, *n_mels=128*, *parent=None*, *name=None*)

Computation step that extracts mfcc features from the given frames.

Based on http://librosa.github.io/librosa/generated/librosa.feature.mfcc.html

Parameters

- **`n_mels`** (*`int`*) – Number of mel bands to generate.
- **`n_mfcc`** (*`int`*) – number of MFCCs to return.

`compute`(*chunk*, *sampling_rate*, *corpus=None*, *utterance=None*)

Do the computation of the step. If the step uses context, the result has to be returned without context.

Parameters

- **`chunk`** (`Chunk`) – The chunk containing data and info about context, offset, . . .
- **`sampling_rate`** (*`int`*) – The sampling rate of the underlying signal.
- **`corpus`** (`Corpus`) – The corpus the data is from, if available.
- **`utterance`** (`Utterance`) – The utterance the data is from, if available.

Returns The array of processed frames, without context.

Return type np.ndarray

class `audiomate.processing.pipeline.`**`PowerToDb`**(*ref=1.0*, *amin=1e-10*, *top_db=80.0*, *parent=None*, *name=None*)

Convert a power spectrogram (amplitude squared) to decibel (dB) units.

See http://librosa.github.io/librosa/generated/librosa.core.power_to_db.html

Note: The output can differ depending on offline or online processing, since it depends on statistics over all values. And in online mode it only considers values from a single chunk, while in offline mode all values of the whole sequence are considered.

compute (*chunk*, *sampling_rate*, *corpus=None*, *utterance=None*)
Do the computation of the step. If the step uses context, the result has to be returned without context.

Parameters

- **chunk** (`Chunk`) – The chunk containing data and info about context, offset, . . .
- **sampling_rate** (*int*) – The sampling rate of the underlying signal.
- **corpus** (`Corpus`) – The corpus the data is from, if available.
- **utterance** (`Utterance`) – The utterance the data is from, if available.

Returns The array of processed frames, without context.

Return type np.ndarray

class audiomate.processing.pipeline.**Delta** (*width=9*, *order=1*, *axis=0*, *mode='interp'*, *parent=None*, *name=None*)
Compute delta features.

See http://librosa.github.io/librosa/generated/librosa.feature.delta.html

compute (*chunk*, *sampling_rate*, *corpus=None*, *utterance=None*)
Do the computation of the step. If the step uses context, the result has to be returned without context.

Parameters

- **chunk** (`Chunk`) – The chunk containing data and info about context, offset, . . .
- **sampling_rate** (*int*) – The sampling rate of the underlying signal.
- **corpus** (`Corpus`) – The corpus the data is from, if available.
- **utterance** (`Utterance`) – The utterance the data is from, if available.

Returns The array of processed frames, without context.

Return type np.ndarray

class audiomate.processing.pipeline.**AddContext** (*left_frames*, *right_frames*, *parent=None*, *name=None*)
For every frame add context frames from left or/and right. For frames at the beginning and end of a sequence, where no context is available, zeros are used.

Parameters

- **left_frames** (*int*) – Number of previous frames to prepend to a frame.
- **right_frames** (*int*) – Number of subsequent frames to append to a frame.

Example

```
>>> input = np.array([
>>>     [1,2,3],
>>>     [4,5,6],
>>>     [7,8,9]
>>> ])
>>> chunk = Chunk(input, offset=0, is_last=True)
>>> AddContext(left_frames=1, right_frames=1).compute(chunk, 16000)
array([[0, 0, 0, 1, 2, 3, 4, 5, 6],
       [1, 2, 3, 4, 5, 6, 7, 8, 9],
       [4, 5, 6, 7, 8, 9, 0, 0, 0]])
```

compute(*chunk*, *sampling_rate*, *corpus=None*, *utterance=None*)
Do the computation of the step. If the step uses context, the result has to be returned without context.

Parameters

- **chunk** (Chunk) – The chunk containing data and info about context, offset, ...
- **sampling_rate** (*int*) – The sampling rate of the underlying signal.
- **corpus** (Corpus) – The corpus the data is from, if available.
- **utterance** (Utterance) – The utterance the data is from, if available.

Returns The array of processed frames, without context.

Return type np.ndarray

class audiomate.processing.pipeline.**Stack**(*parents*, *name=None*, *min_frames=1*, *left_context=0*, *right_context=0*)
Stack the features from multiple inputs. All input matrices have to be of the same length (same number of frames).

compute(*chunk*, *sampling_rate*, *corpus=None*, *utterance=None*)
Do the computation of the step. If the step uses context, the result has to be returned without context.

Parameters

- **chunk** (Chunk) – The chunk containing data and info about context, offset, ...
- **sampling_rate** (*int*) – The sampling rate of the underlying signal.
- **corpus** (Corpus) – The corpus the data is from, if available.
- **utterance** (Utterance) – The utterance the data is from, if available.

Returns The array of processed frames, without context.

Return type np.ndarray

class audiomate.processing.pipeline.**AvgPool**(*size*, *parent=None*, *name=None*)
Average a given number of sequential frames into a single frame. If at the end of a stream just the remaining frames are used, no matter how many there are left.

Parameters **size** (*float*) – The maximum number of frames to pool by taking the mean.

compute(*chunk*, *sampling_rate*, *corpus=None*, *utterance=None*)
Do the computation of the step. If the step uses context, the result has to be returned without context.

Parameters

- **chunk** (Chunk) – The chunk containing data and info about context, offset, ...
- **sampling_rate** (*int*) – The sampling rate of the underlying signal.
- **corpus** (Corpus) – The corpus the data is from, if available.
- **utterance** (Utterance) – The utterance the data is from, if available.

Returns The array of processed frames, without context.

Return type np.ndarray

frame_transform_step(*frame_size*, *hop_size*)
If the processor changes the number of samples that build up a frame or the number of samples between two consecutive frames (hop-size), this function needs transform the original frame- and/or hop-size.

This is used to store the frame-size and hop-size in a feature-container. In the end one can calculate start and end time of a frame with this information.

By default it is assumed that the processor doesn't change the frame-size and the hop-size.

Note: This function is simply for this step, whereas `frame_transform()` computes the transformation for the whole pipeline.

Parameters

- **`frame_size`** (*`int`*) – The original frame-size.
- **`hop_size`** (*`int`*) – The original hop-size.

Returns The (frame-size, hop-size) after processing.

Return type tuple

class `audiomate.processing.pipeline.`**`VarPool`**(*size*, *parent=None*, *name=None*)

Variance over a given number of sequential frames to form a single frame. If at the end of a stream just the remaining frames are used, no matter how many there are left.

Parameters **`size`** (*`float`*) – The maximum number of frames to pool by taking the mean.

`compute`(*chunk*, *sampling_rate*, *corpus=None*, *utterance=None*)

Do the computation of the step. If the step uses context, the result has to be returned without context.

Parameters

- **`chunk`** (`Chunk`) – The chunk containing data and info about context, offset, . . .
- **`sampling_rate`** (*`int`*) – The sampling rate of the underlying signal.
- **`corpus`** (`Corpus`) – The corpus the data is from, if available.
- **`utterance`** (`Utterance`) – The utterance the data is from, if available.

Returns The array of processed frames, without context.

Return type np.ndarray

`frame_transform_step`(*frame_size*, *hop_size*)

If the processor changes the number of samples that build up a frame or the number of samples between two consecutive frames (hop-size), this function needs transform the original frame- and/or hop-size.

This is used to store the frame-size and hop-size in a feature-container. In the end one can calculate start and end time of a frame with this information.

By default it is assumed that the processor doesn't change the frame-size and the hop-size.

Note: This function is simply for this step, whereas `frame_transform()` computes the transformation for the whole pipeline.

Parameters

- **`frame_size`** (*`int`*) – The original frame-size.
- **`hop_size`** (*`int`*) – The original hop-size.

Returns The (frame-size, hop-size) after processing.

Return type tuple

class audiomate.processing.pipeline.**OnsetStrength**(*n_mels=128*, *parent=None*, *name=None*)

Compute a spectral flux onset strength envelope.

Based on http://librosa.github.io/librosa/generated/librosa.onset.onset_strength.html

Parameters **n_mels** (*int*) – Number of mel bands to generate.

compute(*chunk*, *sampling_rate*, *corpus=None*, *utterance=None*)

Do the computation of the step. If the step uses context, the result has to be returned without context.

Parameters

- **chunk** (Chunk) – The chunk containing data and info about context, offset, . . .
- **sampling_rate** (*int*) – The sampling rate of the underlying signal.
- **corpus** (Corpus) – The corpus the data is from, if available.
- **utterance** (Utterance) – The utterance the data is from, if available.

Returns The array of processed frames, without context.

Return type np.ndarray

class audiomate.processing.pipeline.**Tempogram**(*n_mels=128*, *win_length=384*, *parent=None*, *name=None*)

Computation step to compute tempogram

Based on http://librosa.github.io/librosa/generated/librosa.feature.tempogram.html

Parameters

- **n_mels** (*int*) – Number of mel bands to generate.
- **win_length** (*int*) – Length of the onset autocorrelation window (in frames/onset measurements). The default settings (384) corresponds to 384 * hop_length / sr ~= 8.9s.

compute(*chunk*, *sampling_rate*, *corpus=None*, *utterance=None*)

Do the computation of the step. If the step uses context, the result has to be returned without context.

Parameters

- **chunk** (Chunk) – The chunk containing data and info about context, offset, . . .
- **sampling_rate** (*int*) – The sampling rate of the underlying signal.
- **corpus** (Corpus) – The corpus the data is from, if available.
- **utterance** (Utterance) – The utterance the data is from, if available.

Returns The array of processed frames, without context.

Return type np.ndarray

CHAPTER 19

audiomate.encoding

The encoding module provides functionality to encode labels to use for example for training a DNN.

19.1 Encoder

class `audiomate.encoding.`**`Encoder`**

Base class for an encoder. The goal of an encoder is to extract encoded targets for an utterance. The base class provides functionality to perform encoding for a full corpus. A concrete encoder just has to provide the method to encode a single utterance via `encode_utterance`.

For example for training a frame-classifier, an encoder extracts one-hot encoded vectors from a label-list.

`encode_corpus`(*corpus*, *output_path*)

Encode all utterances of the given corpus and store them in a `audiomate.container.Container`.

Parameters

- **`corpus`** (`Corpus`) – The corpus to process.
- **`output_path`** (*`str`*) – The path to store the container with the encoded data.

Returns The container with the encoded data.

Return type *Container*

`encode_utterance`(*utterance*, *corpus=None*)

Encode the given utterance.

Parameters

- **`utterance`** (`Utterance`) – The utterance to encode.
- **`corpus`** (`Corpus`) – The corpus the utterance is from.

Returns Encoded data.

Return type np.ndarray

19.2 Frame-Based

class audiomate.encoding.**FrameHotEncoder**(*labels*, *label_list_idx*, *frame_settings*, *sr=None*)

The FrameHotEncoder is used to encode the labels per frame. It creates a matrix with dimension num-frames x len(labels). The vector (2nd dim) has an entry for every label in the passed labels-list. If the sequence contains a given label within a frame it is set to 1.

Parameters

- **labels** (*list*) – List of labels (str) which should be included in the vector representation.
- **label_list_idx** (*str*) – The name of the label-list to use for encoding. Only labels of this label-list are considered.
- **frame_settings** (FrameSettings) – Frame settings to use.
- **sr** (*int*) – The sampling rate used, if None it is assumed the native sampling rate from the file is used.

Example

```
>>> from audiomate import annotations
>>> from audiomate.utils import units import
>>>
>>> ll = annotations.LabelList(idx='test', labels=[
>>>     annotations.Label('music', 0, 2),
>>>     annotations.Label('speech', 2, 5),
>>>     annotations.Label('noise', 4, 6),
>>>     annotations.Label('music', 6, 8)
>>> ])
>>> utt.set_label_list(ll)
>>>
>>> labels = ['speech', 'music', 'noise']
>>> fs = units.FrameSettings(16000, 16000)
>>> encoder = FrameHotEncoder(labels, 'test', frame_settings=fs, sr=16000)
>>> encoder.encode_utterance(utt)
array([
    [0, 1, 0],
    [0, 1, 0],
    [1, 0, 0],
    [1, 0, 0],
    [1, 0, 1],
    [0, 0, 1],
    [0, 1, 0],
    [0, 1, 0]
])
```

encode_utterance(*utterance*, *corpus=None*)

Encode the given utterance.

Parameters

- **utterance** (Utterance) – The utterance to encode.
- **corpus** (Corpus) – The corpus the utterance is from.

Returns Encoded data.

Return type np.ndarray

class audiomate.encoding.**FrameOrdinalEncoder**(*labels*, *label_list_idx*, *frame_settings*, *sr=None*)

The FrameOrdinalEncoder is used to encode the labels per frame. It creates a vector with length num-frames. For every frame sets the index of the label that is present for that frame. If multiple labels are present the longest within the frame. If multiple labels have the same length the smaller index is selected, hence the passed *labels* list acts as a priority.

Parameters

- **labels** (*list*) – List of labels (str) which should be included in the vector representation.
- **label_list_idx** (*str*) – The name of the label-list to use for encoding. Only labels of this label-list are considered.
- **frame_settings** (FrameSettings) – Frame settings to use.
- **sr** (*int*) – The sampling rate used, if None it is assumed the native sampling rate from the file is used.

Example

```
>>> from audiomate import annotations
>>> from audiomate.utils import units import
>>>
>>> ll = annotations.LabelList(idx='test', labels=[
>>>     annotations.Label('music', 0, 2),
>>>     annotations.Label('speech', 2, 5),
>>>     annotations.Label('noise', 4, 6),
>>>     annotations.Label('music', 6, 8)
>>> ])
>>> utt.set_label_list(ll)
>>>
>>> labels = ['speech', 'music', 'noise']
>>> fs = units.FrameSettings(16000, 16000)
>>> encoder = FrameOrdinalEncoder(labels, 'test', frame_settings=fs)
>>> encoder.encode_utterance(utt)
array([1,1,0,0,0,2,1,1])
```

encode_utterance(*utterance*, *corpus=None*)

Encode the given utterance.

Parameters

- **utterance** (Utterance) – The utterance to encode.
- **corpus** (Corpus) – The corpus the utterance is from.

Returns Encoded data.

Return type np.ndarray

19.3 Utterance-Based

class audiomate.encoding.**TokenOrdinalEncoder**(*label_list_idx*, *tokens*, *token_delimiter=' '*)

Class to encode labels of a given label-list. Every token of the labels is mapped to a number. For the full utterance a sequence/array of numbers are computed, which correspond to tokens.

Tokens are extracted from labels by splitting using a delimiter (by default space). See *audiomate.annotations.Label.tokenized()*. Hence a token can be word, phone, . . . , depending on the label and the delimiter.

Parameters

- **label_list_idx** (*str*) – The name of the label-list to use for encoding. Only labels of this label-list are considered.
- **tokens** (*list*) – List of tokens that defines the mapping. First label will get the 0 in the encoding and so on.
- **token_delimiter** (*str*) – Delimiter to split labels into tokens.

Example

```
>>> ll = LabelList(idx='words', labels=[Label('down the  road')])
>>> utt = Utterance('utt-1', 'file-x', label_lists=ll)
>>>
>>> tokens = ['up', 'down', 'road', 'stree', 'the']
>>> encoder = TokenOrdinalEncoder('words', tokens, token_delimiter=' ')
>>> encoder.encode_utterance(utt)
np.array([1, 4, 2])
```

encode_utterance(*utterance*, *corpus=None*)

Encode the given utterance.

Parameters

- **utterance** (Utterance) – The utterance to encode.
- **corpus** (Corpus) – The corpus the utterance is from.

Returns Encoded data.

Return type np.ndarray

CHAPTER 20

audiomate.feeding

The *audiomate.feeding* module provides tools for a simple access to data stored in different `audiomate.corpus.assets.Container`.

20.1 Datasets

class `audiomate.feeding.`**`Dataset`**(*corpus_or_utt_ids*, *feature_containers*)

An abstract class representing a dataset. A dataset provides indexable access to data. An implementation of a concrete dataset should override the methods `__len__` and `__getitem`.

A sample returned from a dataset is a tuple containing the data for this sample from every container. The data from different containers is ordered in the way the containers were passed to the Dataset.

Parameters

- **`corpus_or_utt_ids`** (`Corpus`, *`list`*) – Either a corpus or a list of utterances. This defines which utterances are considered for iterating.
- **`containers`** (*`list`*, `Container`) – A single container or a list of containers.

class `audiomate.feeding.`**`FrameDataset`**(*corpus_or_utt_ids*, *container*)

A dataset wrapping frames of a corpus. A single sample represents a single frame.

Parameters

- **`corpus_or_utt_ids`** (`Corpus`, *`list`*) – Either a corpus or a list of utterances. This defines which utterances are considered for iterating.
- **`container`** (*`list`*, `Container`) – A single container or a list of containers.

Note: For a frame dataset it is expected that every container contains exactly one value/vector for every frame. So the first dimension of every array in every container have to match.

Example

```
>>> corpus = audiomate.Corpus.load('/path/to/corpus')
>>> container_inputs = containers.FeatureContainer('/path/to/features.hdf5')
>>> container_outputs = containers.Container('/path/to/targets.hdf5')
>>>
>>> ds = FrameDataset(corpus, [container_inputs, container_outputs])
>>> len(ds) # Number of frames in the dataset
2938
>>> ds[293] # Frame (inputs, outputs) with index 293
(
    array([0.58843831, 0.18128443, 0.19718328, 0.25284105]),
    array([0.0, 1.0])
)
```

get_utt_regions()

Return the regions of all utterances, assuming all utterances are concatenated. It is assumed that the utterances are sorted in ascending order for concatenation.

A region is defined by offset (in chunks), length (num-chunks) and a list of references to the utterance datasets in the containers.

Returns List of with a tuple for every utterances containing the region info.

Return type list

partitioned_iterator(*partition_size*, *shuffle=True*, *seed=None*)

Return a partitioning *audiomate.feeding.FrameIterator* for the dataset.

Parameters

- **partition_size** (*str*) – Size of the partitions in bytes. The units `k` (kibibytes), `m` (mebibytes) and `g` (gibibytes) are supported, i.e. a `partition_size` of `1g` equates 2^{30} bytes.
- **shuffle** (*bool*) – Indicates whether the data should be returned in random order (`True`) or not (`False`).
- **seed** (*int*) – Seed to be used for the random number generator.

Returns A partition iterator over the dataset.

Return type *FrameIterator*

class audiomate.feeding.**MultiFrameDataset**(*corpus_or_utt_ids*, *container*, *frames_per_chunk*, *return_length=False*, *pad=False*)

A dataset wrapping chunks of frames of a corpus. A single sample represents a chunk of frames.

A chunk doesn't overlap an utterances boundaries. So if the utterance length is not divisible by the chunk length, the last chunk of an utterance may be smaller than the chunk size.

Parameters

- **corpus_or_utt_ids** (*Corpus, list*) – Either a corpus or a list of utterances. This defines which utterances are considered for iterating.
- **container** (*list, Container*) – A single container or a list of containers.
- **frames_per_chunk** (*int*) – Number of subsequent frames in a single sample.

- **return_length** (*bool*) – If True, the length of the chunk is returned as well. (default `False`) The length is appended to tuple as the last element. (e.g. [container1-data, container2-data, length])
- **pad** (*bool*) – If True, samples that are shorter are padded with zeros to match `frames_per_chunk`. If padding is enabled, the lengths are always returned `return_length = True`.

Note: For a multi-frame dataset it is expected that every container contains exactly one value/vector for every frame. So the first dimension of every array in every container have to match.

Examples

```
>>> corpus = audiomate.Corpus.load('/path/to/corpus')
>>> container_inputs = containers.FeatureContainer('/path/to/features.hdf5')
>>> container_outputs = containers.Container('/path/to/targets.hdf5')
>>>
>>> ds = MultiFrameDataset(corpus, [container_inputs, container_outputs], 5)
>>> len(ds) # Number of chunks in the dataset
355
>>> ds[20] # Chunk (inputs, outputs) with index 20
(
    array([[0.72991909, 0.20258683, 0.30574747, 0.53783217],
           [0.38875413, 0.83611128, 0.49054591, 0.15710017],
           [0.35153358, 0.40051009, 0.93647765, 0.29589257],
           [0.97465772, 0.80160451, 0.81871436, 0.4892925 ],
           [0.59310933, 0.8565602 , 0.95468696, 0.07933512]]),
    array([[0.0, 1.0], [0.0, 1.0],[0.0, 1.0],[0.0, 1.0], [0.0, 1.0]])
)
```

If the length should be returned, pass `True` to `return_length` (Except for chunks at the of utterances the length will be equal to `frames_per_chunk`.)

```
>>> corpus = audiomate.Corpus.load('/path/to/corpus')
>>> container_inputs = containers.FeatureContainer('/path/to/features.hdf5')
>>> container_outputs = containers.Container('/path/to/targets.hdf5')
>>>
>>> ds = MultiFrameDataset(corpus, [container_inputs, container_outputs], 5)
>>> len(ds) # Number of chunks in the dataset
355
>>> ds[20] # Chunk (inputs, outputs) with index 20
(
    array([[0.72991909, 0.20258683, 0.30574747, 0.53783217],
           [0.38875413, 0.83611128, 0.49054591, 0.15710017],
           [0.35153358, 0.40051009, 0.93647765, 0.29589257],
           [0.97465772, 0.80160451, 0.81871436, 0.4892925 ],
           [0.59310933, 0.8565602 , 0.95468696, 0.07933512]]),
    array([[0.0, 1.0], [0.0, 1.0],[0.0, 1.0],[0.0, 1.0], [0.0, 1.0]]),
    5
)
```

get_utt_regions()
Return the regions of all utterances, assuming all utterances are concatenated. It is assumed that the utterances are sorted in ascending order for concatenation.

A region is defined by offset (in chunks), length (num-chunks) and a list of references to the utterance datasets in the containers.

Returns List of with a tuple for every utterances containing the region info.

Return type list

`partitioned_iterator`(*partition_size*, *shuffle=True*, *seed=None*)

Return a partitioning *`audiomate.feeding.MultiFrameIterator`* for the dataset.

Parameters

- **`partition_size`** (*`str`*) – Size of the partitions in bytes. The units `k` (kibibytes), `m` (mebibytes) and `g` (gibibytes) are supported, i.e. a `partition_size` of `1g` equates 2^{30} bytes.
- **`shuffle`** (*`bool`*) – Indicates whether the data should be returned in random order (`True`) or not (`False`).
- **`seed`** (*`int`*) – Seed to be used for the random number generator.

Returns A partition iterator over the dataset.

Return type *MultiFrameIterator*

20.2 Iterator

`class` `audiomate.feeding.`**`DataIterator`**(*corpus_or_utt_ids*, *feature_containers*, *shuffle=True*, *seed=None*)

An abstract class representing a data-iterator. A data-iterator provides sequential access to data. An implementation of a concrete data-iterator should override the methods `__iter__` and `__next__`.

A sample returned from a data-iterator is a tuple containing the data for this sample from every container. The data from different containers is ordered in the way the containers were passed to the DataIterator.

Parameters

- **`corpus_or_utt_ids`** (`Corpus`, *`list`*) – Either a corpus or a list of utterances. This defines which utterances are considered for iterating.
- **`containers`** (*`list`*, `Container`) – A single container or a list of containers.
- **`shuffle`** (*`bool`*) – Indicates whether the data should be returned in random order (`True`) or not (`False`).
- **`seed`** (*`int`*) – Seed to be used for the random number generator.

`class` `audiomate.feeding.`**`FrameIterator`**(*corpus_or_utt_ids*, *container*, *partition_size*, *shuffle=True*, *seed=None*)

A data-iterator wrapping frames of a corpus. A single sample represents a single frame.

Parameters

- **`corpus_or_utt_ids`** (`Corpus`, *`list`*) – Either a corpus or a list of utterances. This defines which utterances are considered for iterating.
- **`container`** (*`list`*, `Container`) – A single container or a list of containers.
- **`partition_size`** (*`str`*) – Size of the partitions in bytes. The units `k` (kibibytes), `m` (mebibytes) and `g` (gibibytes) are supported, i.e. a `partition_size` of `1g` equates 2^{30} bytes.

- **shuffle** (*bool*) – Indicates whether the data should be returned in random order (`True`) or not (`False`).
- **seed** (*int*) – Seed to be used for the random number generator.

Note: For a FrameIterator it is expected that every container contains exactly one value/vector for every frame. So the first dimension of every array in every container have to match.

Example

```
>>> corpus = audiomate.Corpus.load('/path/to/corpus')
>>> container_inputs = containers.FeatureContainer('/path/to/features.hdf5')
>>> container_outputs = containers.Container('/path/to/targets.hdf5')
>>>
>>> ds = FrameIterator(corpus, [container_inputs, container_outputs], '1G',␣
↪shuffle=True, seed=23)
>>> next(ds) # Next Frame (inputs, outputs)
(
    array([0.58843831, 0.18128443, 0.19718328, 0.25284105]),
    array([0.0, 1.0])
)
```

class audiomate.feeding.**MultiFrameIterator**(*corpus_or_utt_ids*, *container*, *partition_size*, *frames_per_chunk*, *return_length=False*, *pad=False*, *shuffle=True*, *seed=None*)

A data-iterator wrapping chunks of subsequent frames of a corpus. A single sample represents a chunk of frames.

Parameters

- **corpus_or_utt_ids** (`Corpus`, *list*) – Either a corpus or a list of utterances. This defines which utterances are considered for iterating.
- **container** (*list*, `Container`) – A single container or a list of containers.
- **partition_size** (*str*) – Size of the partitions in bytes. The units `k` (kibibytes), `m` (mebibytes) and `g` (gibibytes) are supported, i.e. a `partition_size` of `1g` equates 2^{30} bytes.
- **frames_per_chunk** (*int*) – Number of subsequent frames in a single sample.
- **return_length** (*bool*) – If True, the length of the chunk is returned as well. (default `False`) The length is appended to tuple as the last element. (e.g. [container1-data, container2-data, length])
- **pad** (*bool*) – If True, samples that are shorter are padded with zeros to match `frames_per_chunk`. If padding is enabled, the lengths are always returned `return_length = True`.
- **shuffle** (*bool*) – Indicates whether the data should be returned in random order (`True`) or not (`False`).
- **seed** (*int*) – Seed to be used for the random number generator.

Note: For a MultiFrameIterator it is expected that every container contains exactly one value/vector for every frame. So the first dimension (outermost) of every array in every container have to match.

Example

```
>>> corpus = audiomate.Corpus.load('/path/to/corpus')
>>> container_inputs = containers.FeatureContainer('/path/to/features.hdf5')
>>> container_outputs = containers.Container('/path/to/targets.hdf5')
>>>
>>> ds = MultiFrameIterator(corpus, [container_inputs, container_outputs], '1G',␣
↪5, shuffle=True, seed=23)
>>> next(ds) # Next Chunk (inputs, outputs)
(
    array([[0.72991909, 0.20258683, 0.30574747, 0.53783217],
           [0.38875413, 0.83611128, 0.49054591, 0.15710017],
           [0.35153358, 0.40051009, 0.93647765, 0.29589257],
           [0.97465772, 0.80160451, 0.81871436, 0.4892925 ],
           [0.59310933, 0.8565602 , 0.95468696, 0.07933512]])
    array([[0.0, 1.0], [0.0, 1.0],[0.0, 1.0],[0.0, 1.0], [0.0, 1.0]])
)
```

20.3 Partitioning

class `audiomate.feeding.`**`PartitioningContainerLoader`**(*corpus_or_utt_ids*, *feature_containers*, *partition_size*, *shuffle=True*, *seed=None*)

Load data from one or more containers in partitions. It computes a scheme to load the data of as many utterances as possible in one partition.

A scheme is initially computed on creation of the loader. To compute a new one the `reload()` method can be used. This only has an effect if `shuffle == True`, otherwise the utterances are defined always loaded in the same order.

With a given scheme, data of a partition can be retrieved via `load_partition_data()`. It loads all data of the partition with the given index into memory.

Parameters

- **corpus_or_utt_ids** (`Corpus, list`) – Either a corpus or a list of utterances. This defines which utterances are considered for loading.
- **containers** (*`container.Container, list`*) – Either a single or a list of Container objects. From the given containers data is loaded.
- **partition_size** (*`str`*) – Size of the partitions in bytes. The units `k` (kibibytes), `m` (mebibytes) and `g` (gibibytes) are supported, i.e. a `partition_size` of `1g` equates 2^{30} bytes.
- **shuffle** (*`bool`*) – Indicates whether the utterances should be returned in random order (`True`) or not (`False`).
- **seed** (*`int`*) – Seed to be used for the random number generator.

Example

```
>>> corpus = audiomate.Corpus.load('/path/to/corpus')
>>> container_inputs = containers.FeatureContainer('/path/to/feat.hdf5')
>>> container_outputs = containers.Container('/path/to/targets.hdf5')
```

(continues on next page)

(continued from previous page)

```
>>>
>>> lo = PartitioningContainerLoader(
>>>     corpus,
>>>     [container_inputs, container_outputs],
>>>     '1G',
>>>     shuffle=True,
>>>     seed=23
>>> )
>>> len(lo.partitions) # Number of parititions
5
>>> lo.partitions[0].utt_ids # Utterances in the partition with index 0
['utt-1', 'utt-2', ...]
>>> p0 = lo.load_partition_data(0) # Load partition 0 into memory
>>> p0.info.utt_ids[0] # First utterance in the partition
'utt-1'
>>> p0.utt_data[0] # Data of the first utterance
(
    array([[0.58843831, 0.18128443, 0.19718328, 0.25284105], ...]),
    array([[0.0, 1.0], ...])
)
```

load_partition_data(*index*)
Load and return the partition with the given index.

Parameters **index** (*int*) – The index of partition, that refers to the index in `self.partitions`.

Returns

A PartitionData object containing the data for the partition with the given index.

Return type *PartitionData*

reload()
Create a new partition scheme. A scheme defines which utterances are in which partition. The scheme only changes after every call if `self.shuffle == True`.

Returns

List of PartitionInfo objects, defining the new partitions (same as `self.partitions`).

Return type list

class audiomate.feeding.**PartitionInfo**
Class for holding the info of a partition.

Variables

- **utt_ids** (*list*) – A list of utterance-ids in the partition.
- **utt_lengths** (*list*) – List with lengths of the utterances (Outermost dimension in the dataset of the container). Since there are maybe multiple containers, every item is a tuple of lengths. They correspond to the length of the utterance in every container, in the order of the containers passed to the ParitioningContainerLoader.
- **size** (*int*) – The number of bytes the partition will allocate, when loaded.

total_lengths()
Return the total length of all utterances for every container.

class audiomate.feeding.**PartitionData**(*info*)

Class for holding the loaded data of a partition.

Parameters **info** (PartitionInfo) – The info about the partition.

Variables **utt_data** (*list*) – A list holding the data-objects for every utterance in the order of `info.utt_ids`. The entries are also lists or tuples containing the array for every container.

class audiomate.feeding.**PartitioningFeatureIterator**(*hdf5file*, *partition_size*, *shuffle=True*, *seed=None*, *includes=None*, *excludes=None*)

Iterates over all features in the given HDF5 file.

Before iterating over the features, the iterator slices the file into one or more partitions and loads the data into memory. This leads to significant speed-ups even with moderate partition sizes, regardless of the type of disk (spinning or flash). Pseudo random access is supported with a negligible impact on performance and randomness: The data is randomly sampled (without replacement) within each partition and the partitions are loaded in random order, too.

The features are emitted as triplets in the form of `(utterance name, index of the feature within the utterance, feature)`.

When calculating the partition sizes only the size of the features itself is factored in, overhead of data storage is ignored. This overhead is usually negligible even with partition sizes of multiple gigabytes because the data is stored as numpy ndarrays in memory (one per utterance). The overhead of a single ndarray is 96 bytes regardless of its size. Nonetheless the partition size should be chosen to be lower than the total available memory.

Parameters

- **hdf5file** (*h5py.File*) – HDF5 file containing the features
- **partition_size** (*str*) – Size of the partitions in bytes. The units `k` (kibibytes), `m` (mebibytes) and `g` (gibibytes) are supported, i.e. a `partition_size` of `1g` equates 2^{30} bytes.
- **shuffle** (*bool*) – Indicates whether the features should be returned in random order (`True`) or not (`False`).
- **seed** (*int*) – Seed to be used for the random number generator.
- **includes** (*iterable*) – Iterable of names of data sets that should be included when iterating over the feature container. Mutually exclusive with `excludes`. If both are specified, only `includes` will be considered.
- **excludes** (*iterable*) – Iterable of names of data sets to skip when iterating over the feature container. Mutually exclusive with `includes`. If both are specified, only `includes` will be considered.

Example

```
>>> import h5py
>>> from audiomate.feeding import PartitioningFeatureIterator
>>> hdf5 = h5py.File('features.h5', 'r')
>>> iterator = PartitioningFeatureIterator(hdf5, '12g', shuffle=True)
>>> next(iterator)
('music-fma-0100', 227, array([
    -0.15004082, -0.30246958, -0.38708138, ...,
    -0.93471956, -0.94194776, -0.90878332 ], dtype=float32))
>>> next(iterator)
('music-fma-0081', 2196, array([
```

(continues on next page)

(continued from previous page)

```
    -0.00207647, -0.00101351, -0.00058832, ...,
    -0.00207647, -0.00292684, -0.00292684], dtype=float32))
>>> next(iterator)
('music-hd-0050', 1026, array([
    -0.57352495, -0.63049972, -0.63049972, ...,
    0.82490814, 0.84680521,  0.75517786], dtype=float32))
```

CHAPTER 21

audiomate.formats

This module contains code for working with different file formats.

21.1 Audacity Labels

`audiomate.formats.audacity.`**`read_label_file`**`(`*path*`)`

Read the labels from an audacity label file.

Parameters **`path`** (*`str`*) – Path to the label file.

Returns List of labels (start [sec], end [sec], label)

Return type list

Example:

```
>>> read_label_file('/path/to/label/file.txt')
[
    [0.0, 0.2, 'sie'],
    [0.2, 2.2, 'hallo']
]
```

`audiomate.formats.audacity.`**`read_label_list`**`(`*path*`)`

Reads labels from an Audacity label file and returns them wrapped in a *`audiomate.annotations.LabelList`*.

Parameters **`path`** (*`str`*) – Path to the Audacity label file

Returns Label list containing the labels

Return type *audiomate.annotations.LabelList*

`audiomate.formats.audacity.`**`write_label_file`**`(`*path*, *entries*`)`

Writes an audacity label file. Start and end times are in seconds.

Parameters

- **path** (*str*) – Path to write the file to.
- **entries** (*list*) – List with entries to write.

Example:

```
>>> data = [
>>>     [0.0, 0.2, 'sie'],
>>>     [0.2, 2.2, 'hallo']
>>> ]
>>>
>>> write_label_file('/some/path/to/file.txt', data)
```

audiomate.formats.audacity.**write_label_list**(*path*, *label_list*)
Writes the given *label_list* to an audacity label file.

Parameters

- **path** (*str*) – Path to write the file to.
- **label_list** (audiomate.annotations.LabelList) – Label list

21.2 CTM Files

audiomate.formats.ctm.**read_file**(*path*)
Reads a ctm file.

Parameters **path** (*str*) – Path to the file

Returns Dictionary with entries.

Return type (dict)

Example:

```
>>> read_file('/path/to/file.txt')
{
    'wave-ab': [
        ['1', 0.00, 0.07, 'HI', 1],
        ['1', 0.09, 0.08, 'AH', 1]
    ],
    'wave-xy': [
        ['1', 0.00, 0.07, 'HI', 1],
        ['1', 0.09, 0.08, 'AH', 1]
    ]
}
```

audiomate.formats.ctm.**write_file**(*path*, *entries*)
Writes a ctm file.

Parameters

- **path** (*str*) – Path to write the file to.
- **entries** (*list*) – List with entries to write. (entries -> wave-file, channel, start (seconds), duration (seconds), label)

Example:

```
>>> data = [
>>>     ["wave-ab", '1', 0.0, 0.82, "duda"],
>>>     ["wave-xy", '1', 0.82, 0.57, "Jacques"],
>>> ]
>>>
>>> write_file('/path/to/file.txt', data)
```

21.3 TRN Files

Functions for reading/writing sclite transcription files.

Description of the format: http://www1.icsi.berkeley.edu/Speech/docs/sctk-1.2/infmts.htm#trn_fmt_name_0

audiomate.formats.trn.**read**(*path*)
Read the labels from a transcription file.

Parameters **path** (*str*) – Path to the label file.

Returns Dictionary of transcriptions (utt-idx: transcription)

Return type dict

Example:

```
>>> read_label_file('/path/to/label/file.txt')
{
    'utt-1': 'sie',
    'utt-2': 'hallo'
}
```

audiomate.formats.trn.**write**(*path*, *entries*)
Writes an transcription file.

Parameters

- **path** (*str*) – Path to write the file to.
- **entries** (*dict*) – List with entries to write.

Example:

```
>>> data = {
>>>     'utt-1': 'sie',
>>>     'utt-2': 'hallo',
>>> }
>>>
>>> write_label_file('/some/path/to/file.txt', data)
```

CHAPTER 22

audiomate.utils

22.1 Audio

`audiomate.utils.audio.`**`process_buffer`**(*buffer*, *n_channels*)

Merge the read blocks and resample if necessary.

Parameters

- **`buffer`** (*`list`*) – A list of blocks of samples.
- **`n_channels`** (*`int`*) – The number of channels of the input data.

Returns The samples

Return type np.array

`audiomate.utils.audio.`**`read_blocks`**(*file_path*, *start=0.0*, *end=inf*, *buffer_size=5760000*)

Read an audio file block after block. The blocks are yielded one by one.

Parameters

- **`file_path`** (*`str`*) – Path to the file to read.
- **`start`** (*`float`*) – Start in seconds to read from.
- **`end`** (*`float`*) – End in seconds to read to. `inf` means to the end of the file.
- **`buffer_size`** (*`int`*) – Number of samples to load into memory at once and return as a single block. The exact number of loaded samples depends on the block-size of the audioread library. So it can be of x higher, where the x is typically 1024 or 4096.

Returns A generator yielding the samples for every block.

Return type Generator

`audiomate.utils.audio.`**`read_frames`**(*file_path*, *frame_size*, *hop_size*, *start=0.0*, *end=inf*, *buffer_size=5760000*)

Read an audio file frame by frame. The frames are yielded one after another.

Parameters

- **file_path** (*str*) – Path to the file to read.
- **frame_size** (*int*) – The number of samples per frame.
- **hop_size** (*int*) – The number of samples between two frames.
- **start** (*float*) – Start in seconds to read from.
- **end** (*float*) – End in seconds to read to. `inf` means to the end of the file.
- **buffer_size** (*int*) – Number of samples to load into memory at once and return as a single block. The exact number of loaded samples depends on the block-size of the audioread library. So it can be of x higher, where the x is typically 1024 or 4096.

Returns A generator yielding a tuple for every frame. The first item is the frame and the second a boolean indicating if it is the last frame.

Return type Generator

`audiomate.utils.audio.`**`write_wav`**(*path*, *samples*, *sr=16000*)

Write to given samples to a wav file. The samples are expected to be floating point numbers in the range of -1.0 to 1.0.

Parameters

- **path** (*str*) – The path to write the wav to.
- **samples** (*np.array*) – A float array .
- **sr** (*int*) – The sampling rate.

22.2 Audioread

Wrapping opening function of audioread library. This is used to cache the available backends. If backend evaluation is done on every call it is very inefficient.

`audiomate.utils.audioread.`**`audio_open`**(*path*)

Just calls `audioread.audio_open`, but with backends cached in a global variable. Brings better performance, since available backends are evaluated only once.

22.3 JSON File

This module contains functions for reading and writing json files.

`audiomate.utils.jsonfile.`**`read_json_file`**(*path*)

Reads and return the data from the json file at the given path.

Parameters **path** (*str*) – Path to read

Returns The read json as dict/list.

Return type dict,list

`audiomate.utils.jsonfile.`**`write_json_to_file`**(*path*, *data*)

Writes data as json to file.

Parameters

- **path** (*str*) – Path to write to
- **data** (*dict, list*) – Data

22.4 Naming

This module contains functions for working with names. For example to generate identifiers or find an identifier which not already exists in a given list.

`audiomate.utils.naming.`**`generate_name`**(*length=15*, *not_in=None*)

Generates a random string of lowercase letters with the given length.

Parameters

- **`length`** (*`int`*) – Length of the string to output.
- **`not_in`** (*`list`*) – Only return a string not in the given iterator.

Returns A new name thats not in the given list.

Return type str

`audiomate.utils.naming.`**`index_name_if_in_list`**(*name*, *name_list*, *suffix=''*, *prefix=''*)

Find a unique name by adding an index to the name so it is unique within the given list.

Parameters

- **`name`** (*`str`*) – Name
- **`name_list`** (*`iterable`*) – List of names that the new name must differ from.
- **`suffix`** (*`str`*) – The suffix to append after the index.
- **`prefix`** (*`str`*) – The prefix to append in front of the index.

Returns A unique name within the given list.

Return type str

22.5 Text

This module contains any functions for working with text/strings/punctuation.

`audiomate.utils.text.`**`remove_punctuation`**(*text*, *exceptions=None*)

Return a string with punctuation removed.

Parameters

- **`text`** (*`str`*) – The text to remove punctuation from.
- **`exceptions`** (*`list`*) – List of symbols to keep in the given text.

Returns The input text without the punctuation.

Return type str

`audiomate.utils.text.`**`starts_with_prefix_in_list`**(*text*, *prefixes*)

Return *True* if the given string starts with one of the prefixes in the given list, otherwise return *False*.

Parameters

- **`text`** (*`str`*) – Text to check for prefixes.
- **`prefixes`** (*`list`*) – List of prefixes to check for.

Returns

***True* if the given text starts with any of the given prefixes,** *False* otherwise.

Return type bool

22.6 Text File

The textfile module contains functions for reading and writing textfiles.

`audiomate.utils.textfile.`**`read_key_value_lines`**(*path*, *separator=' '*, *default_value="*)
Reads lines of a text file with two columns as key/value dictionary.

Parameters

- **`path`** (*`str`*) – Path to the file.
- **`separator`** (*`str`*) – Separator that is used to split key and value.
- **`default_value`** (*`str`*) – If no value is given this value is used.

Returns A dictionary with first column as key and second as value.

Return type dict

`audiomate.utils.textfile.`**`read_separated_lines`**(*path*, *separator=' '*, *max_columns=-1*, *keep_empty=False*)
Reads a text file where each line represents a record with some separated columns.

Parameters

- **`path`** (*`str`*) – Path to the file to read.
- **`separator`** (*`str`*) – Separator that is used to split the columns.
- **`max_columns`** (*`int`*) – Number of max columns (if the separator occurs within the last column).
- **`keep_empty`** (*`bool`*) – If True empty columns are returned as well.

Returns A list containing a list for each line read.

Return type list

`audiomate.utils.textfile.`**`read_separated_lines_generator`**(*path*, *separator=' '*, *max_columns=-1*, *ignore_lines_starting_with=None*, *keep_empty=False*)
Creates a generator through all lines of a file and returns the splitted line.

Parameters

- **`path`** (*`str`*) – Path to the file.
- **`separator`** (*`str`*) – Separator that is used to split the columns.
- **`max_columns`** (*`int`*) – Number of max columns (if the separator occurs within the last column).
- **`ignore_lines_starting_with`** (*`list`*) – Lines starting with a string in this list will be ignored.
- **`keep_empty`** (*`bool`*) – If True empty columns are returned as well.

`audiomate.utils.textfile.`**`read_separated_lines_with_first_key`**(*path: str*, *separator: str = ' '*, *max_columns: int = -1*, *keep_empty: bool = False*)

Reads the separated lines of a file and return a dictionary with the first column as keys, value is a list with the rest of the columns.

Parameters

- **`path`** (*`str`*) – Path to the file to read.
- **`separator`** (*`str`*) – Separator that is used to split the columns.
- **`max_columns`** (*`str`*) – Number of max columns (if the separator occurs within the last column).
- **`keep_empty`** (*`bool`*) – If True empty columns are returned as well.

Returns Dictionary with list of column values and first column value as key.

Return type dict

`audiomate.utils.textfile.`**`write_separated_lines`**(*path*, *values*, *separator=' '*, *sort_by_column=0*)

Writes list or dict to file line by line. Dict can have list as value then they written separated on the line.

Parameters

- **`path`** (*`str`*) – Path to write file to.
- **`values`** (*`dict, list`*) – A dictionary or a list to write to the file.
- **`separator`** (*`str`*) – Separator to use between columns.
- **`sort_by_column`** (*`int`*) – if >= 0, sorts the list by the given index, if its 0 or 1 and its a dictionary it sorts it by either the key (0) or value (1). By default 0, meaning sorted by the first column or the key.

22.7 Units

This module contains functions for handling different units. Especially it provides function to convert from one to another unit (e.g. seconds -> sample-indexn).

`class` `audiomate.utils.units.`**`FrameSettings`**(*frame_size*, *hop_size*)

This class provides functions for handling conversions/calculations between time, samples and frames.

By default the framing is done as follows:

- The first frame starts at sample 0
- The end of the last frame is higher than the last sample.
- The end of the last frame is smaller than the last sample + hop_size

Parameters

- **`frame_size`** (*`int`*) – Number of samples used per frame.
- **`hop_size`** (*`int`*) – Number of samples between two frames.

frame_to_sample(*frame_index*)
Return a tuple containing the indices of the sample which are the first sample and the end (exclusive) of the frame with the given index.

frame_to_seconds(*frame_index*, *sr*)
Return a tuple containing the start and end of the frame in seconds.

num_frames(*num_samples*)
Return the number of frames that will be used for a signal with the length of `num_samples`.

sample_to_frame_range(*sample_index*)
Return a tuple containing the indices of the first frame containing the sample with the given index and the last frame (exclusive, doesn't contain the sample anymore).

time_range_to_frame_range(*start*, *end*, *sr*)
Calculate the frames containing samples from the given time range in seconds.

Parameters

- **start** (*float*) – Start time in seconds.
- **end** (*float*) – End time in seconds.
- **sr** (*int*) – The sampling rate to use for time-to-sample conversion.

Returns A tuple containing the start and end (exclusive) frame indices.

Return type tuple

audiomate.utils.units.**parse_storage_size**(*storage_size*)
Parses an expression that represents an amount of storage/memory and returns the number of bytes it represents.

Parameters **storage_size** (*str*) – Size in bytes. The units `k` (kibibytes), `m` (mebibytes) and `g` (gibibytes) are supported, i.e. a `partition_size` of `1g` equates 2^{30} bytes.

Returns Number of bytes.

Return type int

audiomate.utils.units.**sample_to_seconds**(*sample*, *sampling_rate=16000*)
Convert a sample index to seconds based on the given sampling rate.

Parameters

- **sample** (*int*) – The index of the sample (0 based).
- **sampling_rate** (*int*) – The sampling rate to use for conversion.

Returns The time in seconds.

Return type float

Example::

```
>>> sample_to_seconds(20800, sampling_rate=16000)
1.3
```

audiomate.utils.units.**seconds_to_sample**(*seconds*, *sampling_rate=16000*)
Convert a value in seconds to a sample index based on the given sampling rate.

Parameters

- **seconds** (*float*) – The value in seconds.
- **sampling_rate** (*int*) – The sampling rate to use for conversion.

Returns The sample index (0-based).

Return type int

Example::

```
>>> seconds_to_sample(1.3, sampling_rate=16000)
20800
```

22.8 Misc

`audiomate.utils.misc.`**`length_of_overlap`**(*first_start*, *first_end*, *second_start*, *second_end*)

Find the length of the overlapping part of two segments.

Parameters

- **`first_start`** (*`float`*) – Start of the first segment.
- **`first_end`** (*`float`*) – End of the first segment.
- **`second_start`** (*`float`*) – Start of the second segment.
- **`second_end`** (*`float`*) – End of the second segment.

Returns The amount of overlap or 0 if they don’t overlap at all.

Return type float

CHAPTER 23

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index