ChildProject

Release 0.0.1

Lucas Gautheron

Jan 17, 2022

GETTING-STARTED

CHAPTER

ONE

INTRODUCTION

Day-long (audio-)recordings of children are increasingly common, but there is no scientific standard formatting that can benefit the organization and analyses of such data. ChildProject provides standardizing specifications and tools for the storage and management of day-long recordings of children and their associated meta-data and annotations.

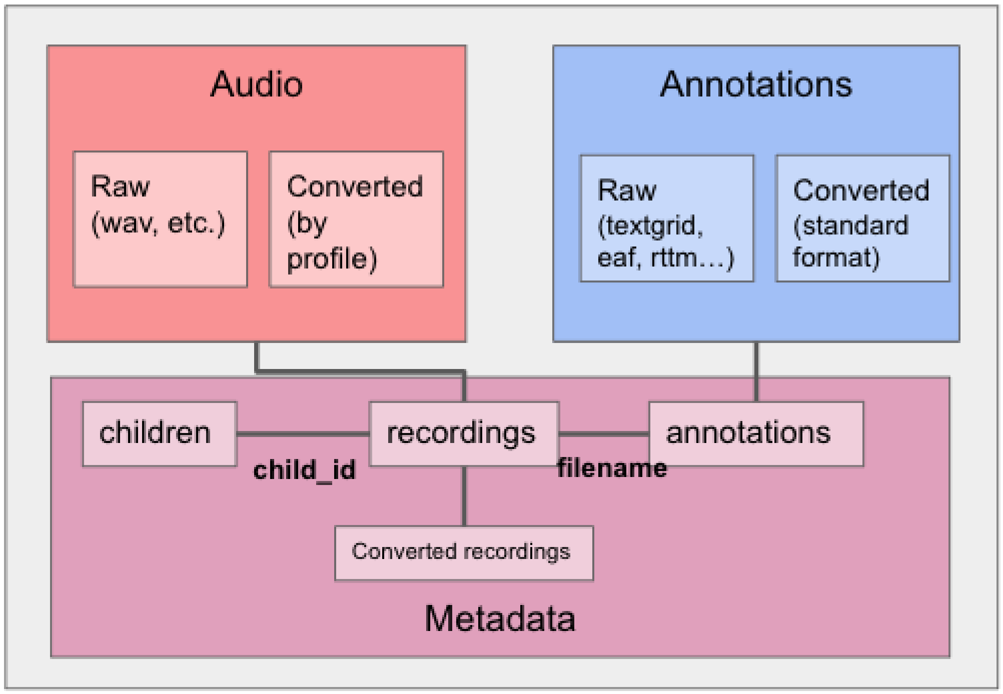


Fig. 1: File organization structure

We assume that the data include three very different types:

1. Audio, of which we distinguish the raw audio extracted from the hardware; and a version that has been converted into a standardized format. These audios are the long-form ones. At the time being, we do not foresee including clips extracted from these long-form audios, and assume that any such process will generate some form of annotation that can then be re-cast temporally to the long-form audio.
2. Annotations, of which we again distinguish raw and standardized versions. At present, we can import from Praat's textgrid, ELAN's eaf, and VTC's rttm format.
3. Metadata corresponding to the children, recordings, and annotations, which will therefore also describe the converted recordings.

1.1 Available tools

Day-long audiorecordings are often collected using a LENA recorder, and analyzed with the LENA software. However, open source alternatives to the LENA commercial environment are emerging, some of which are shown in the following figure.

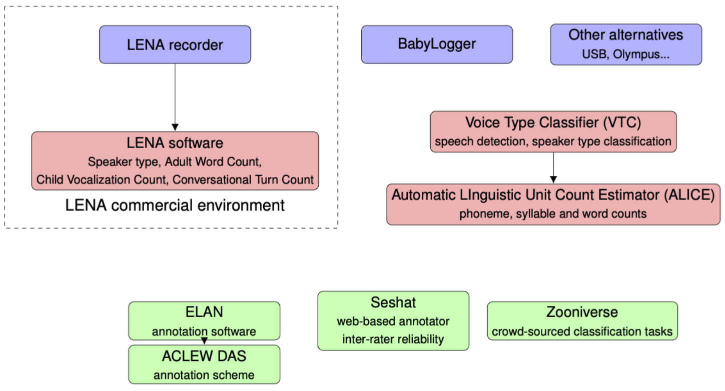

LENA recorder
LENA software
Speaker type, Adult Word Count,
Child Vocalization Count, Conversational Turn Count
LENA commercial environment
BabyLogger
Other alternatives
USB, Olympus...
Voice Type Classifier (VTC)
speech detection, speaker type classification
Automatic LInguistic Unit Count Estimator (ALICE)
phoneme, syllable and word counts
ELAN
annotation software
ACLEW DAS
annotation scheme
Seshat
web-based annotator
inter-rater reliability
Zooniverse
crowd-sourced classification tasks

Fig. 2: Overview of some tools in the day-long recordings environment

For instance, alternative hardware includes the babylogger and any other light-weight recording device with enough battery and storage to record over several hours.

Alternative automated analysis options include the Voice Type Classifier, which segments the audio into different talker types (key child, female adult, etc) and ALICE, an automated linguistic unit counter.

As for manual annotation options, ELAN can be used, for instance employing the ACLEW DAS annotation scheme; CHAT annotations are also supported.

Assignment of annotation to individuals and evaluation can be done using Seshat. Finally, Zooniverse can be used to crowd-source certain aspects of the classification with the help of citizen scientists.

In this context, we provide tools and a procedure to:

- Validate datasets (making sure that metadata, recordings and annotations are in the right place and format)
- Convert your raw recordings into the desired format
- Import annotations (from the LENA, ELAN, Praat, VTC/ALICE/VCM rttms, CHAT files) into a standardized format
- Generate reliability metrics by comparing annotators (confusion matrices, agreement coefficients, pyannote metrics)
- Extract metrics from the annotations (e.g. average vocalization rates, durations, etc.)
- Sample segments of the recordings to annotate from a set of sampling algorithms
- **Add clips to an annotation pipeline in Zooniverse, and retrieve the** ensuing annotations

(And more!)

1.2 Citing this work

If you are using this project for your research, please cite our introductory paper:

```
@article{gautheron_rochat_cristia_2021,
   title={Managing, storing, and sharing long-form recordings and their annotations},
   url={https://psyarxiv.com/w8trm},
   DOI={10.31234/osf.io/w8trm},
   publisher={PsyArXiv},
   author={Gautheron, Lucas and Rochat, Nicolas and Cristia, Alejandrina},
   year={2021},
   month={May}
}
```

1.3 Community

- You can ask for help, suggest ideas about the package or share code that relies on it with others on GitHub discussions.
- Bugs should be reported on GitHub too.

CHAPTER TWO

INSTALLATION

The following instructions will let you install two python packages:

- **ChildProject**, the package that is documented here.
- **DataLad**, a python software for the management and delivery of scientific data. Although ChildProject may work without it, a number of datasets of daylong recordings of children require it.

Note: The default installation procedure requires anaconda. If you are not sure you have conda installed, please do `conda --version`. If you don't, please refer to the instructions here.

2.1 Linux users

```
# download the conda environment
wget https://raw.githubusercontent.com/LAAC-LSCP/ChildProject/master/env_linux.yml -O␣
↪env.yml

# create the conda environment
conda env create -f env.yml

# activate the environment (this should be done systematically to use our package)
conda activate childproject
```

2.2 MacOS users

```
# download the conda environment
curl https://raw.githubusercontent.com/LAAC-LSCP/ChildProject/master/env_macos.yml -o␣
↪env.yml

# create the conda environment
conda env create -f env.yml

# activate the environment (this should be done systematically to use our package)
conda activate childproject

# install git-annex from brew
brew install git-annex
```

Note: The ChildProject package alone can also be installed directly through pip, without using conda. However, this means git-annex, ffmpeg, and other dependencies that are not installable through pip will have to be installed by hand.

The following command will install the python package alone via pip and pypi:

```
pip install ChildProject
```

2.3 Check the setup

You can now make sure the packages have been successfully installed:

```
$ child-project --help
usage: child-project [-h]
                     {validate,import-annotations,merge-annotations,intersect-
↪annotations,remove-annotations,rename-annotations,overview,explain,compute-durations,
↪process,sampler,zooniverse,eaf-builder,anonymize,metrics}
                     ...

positional arguments:
  {validate,import-annotations,merge-annotations,intersect-annotations,remove-
↪annotations,rename-annotations,overview,explain,compute-durations,process,sampler,
↪zooniverse,eaf-builder,anonymize,metrics}

optional arguments:
  -h, --help            show this help message and exit
```

```
$ # optional software, for getting and sharing data
datalad --version
datalad 0.15.4
```

Warning: ChildProject is only officially supported on Linux and Mac for python >= 3.7. We perform automated, continuous testing on these environments to look for potential issues at any step of the development.

We expect the package to work on Windows, although we do not perform automated tests on this OS at the moment.

Note: We recommend that you regularly keep DataLad and our package up to date. To force-upgrade this package, do `pip uninstall ChildProject` and then `pip install ChildProject --upgrade`.

You may also want to install the development version from GitHub in order to receive more recent updates before their release:

```
pip install git+https://github.com/LAAC-LSCP/ChildProject.git --force-reinstall
```

Since some updates may break compatibility with previous versions, we advise you to read the Change Log before upgrading.

DataLad can also be upgraded with `pip install datalad --upgrade` (see DataLad's documentation for more details).

2.4 Troubleshooting

If you are having trouble installing ChildProject, please look for similar issues on our GithHub (in Issues or Discussions).

If this issue is related to a dependency of the package, we recommend that you ask the developers of the depdendency directly as you may get more accurate advice.

If this issue is related to DataLad, please create an issue on DataLad's GitHub.

2.5 Frequently Asked Questions

I don't have anaconda and I can't install it. What should I do?

You should try to install the package inside of a python environment instead, e.g.:

```
python3 -m venv ~/ChildProjectVenv
source ~/ChildProjectVenv/bin/activate
pip3 install ChildProject
```

You will still need git-annex in order to use DataLad. It can be installed with brew for Mac users (*brew install git-annex*) or through apt for Linux users (*apt install git-annex*). Most likely, you will lack permissions to do so if you failed to install anaconda. In this case, pleaser refer to your system administrator.

``brew install git-annex`` does not work!

Please try `brew install --build-from-source git-annex`.

If this does not work better for you, make sure that your version of Mac OS is 10.14 or later. We are aware of issues with Mac OS 10.13 (High Sierra) and earlier.

If your issues persistent, please report it to [DataLad](https://github.com/datalad/datalad).

CHAPTER

THREE

DATASETS STRUCTURE

ChildProject assumes your data is structured in a specific way. This structure is necessary to check, for instance, that there are no unreferenced files, and no referenced files that are actually missing. The data curator therefore needs to organize their data in a specific way (respecting the dataset tree, with all specified metadata files, and all specified columns within the metadata files) before their data can be imported.

To be imported, datasets must pass the the validation routine (see *Data validation*). with no error. We also recommend you pay attention to the warnings, and try to sort as many of those out as possible before submission.

An example of dataset structured according to ChildProject's format can be found here.

3.1 Dataset tree

All datasets should have this structure before import (so you need to organize your files into this structure):

```
project
│
│
└───metadata
│   │   children.csv
│   │   recordings.csv
│   │   annotations.csv
|
└───recordings
│   └───raw
│   │   │   recording1.wav
│
└───annotations
│   └───vtc
│   │   └───raw
│   │   │   │   child1.rttm
│   └───annotator1
│   │   └───raw
│   │   │   │   child1_3600.TextGrid
│
└───docs (*)
│   │   children.csv
│   │   recordings.csv
└───extra
    │   notes.txt
```

The children and recordings notebooks should be CSV dataframes formatted according to the standards detailed right below.

(*) The `docs` folder is optional.

3.2 Metadata

3.2.1 Children notebook

The children metadata dataframe needs to be saved at `metadata/children.csv`. It should be formatted as instructed below; you can add more fields beyond those that are standardized, but make sure to document them.

Table 1: Children metadata

Name	Description	Required?	Format
experiment	one word to capture the unique ID of the data collection effort; for instance Tsimane_2018, solis-intervention-pre	**required**	
child_id	unique child ID – unique within the experiment (Id could be repeated across experiments to refer to different children)	**required**	
child_dob	child’s date of birth	**required**	`%Y-%m-%d`
location_id	Unique location ID – only specify here if children never change locations in this culture; otherwise, specify in the recordings metadata	optional	
child_sex	f= female, m=male	optional	m, M, f, F
language	language the child is exposed to if child is monolingual; small caps, indicate dialect by name or location if available; eg “france french”; “paris french”	optional	
languages	list languages child is exposed to separating them with ; and indicating the percentage if one is available; eg: “french 35%; english 65%”	optional	
mat_ed	maternal years of education	optional	
fat_ed	paternal years of education	optional	
car_ed	years of education of main caregiver (if not mother or father)	optional	
monoling	whether the child is monolingual (Y) or not (N)	optional	Y, N
monoling_criterion	how monoling was decided; eg “we asked families	optional	

3.2.2 Recordings notebook

The recordings metadata dataframe needs to be saved at `metadata/recordings.csv`. It should be formatted as instructed below; you can add more fields beyond those that are standardized, but make sure to document them.

Table 2: Recordings metadata

Name	Description	Required?	Format
experiment	one word to capture the unique ID of the data collection effort; for instance Tsimane_2018, solis-intervention-pre	**required**	
child_id	unique child ID – unique within the experiment (Id could be repeated across experiments to refer to different children)	**required**	
date_iso	date in which recording was started in ISO (eg 2020-09-17)	**required**	`%Y-%m-%d`
start_time	local time in which recording was started in format 24-hour (H)H:MM; if minutes are unknown, use 00. Set as 'NA' if unknown.	**required**	`%H:%M`
recording_device_type	lena, usb, olympus, babylogger (lowercase)	**required**	lena, usb, olympus, babylogger
recording_filename	the path to the file from the root of "recordings"). It MUST be unique (two recordings cannot point towards the same file).	**required**	True
duration	duration of the audio, in milliseconds	optional	`([0-9]+)`
session_id	identifier of the recording session.	optional	
session_offset	offset (in milliseconds) of the recording with respect to other recordings that are part of the same session. Each recording session is identified by their *session_id*.	optional	`[0-9]+`
recording_device_id	unique ID of the recording device	optional	
experimenter	who collected the data (could be anonymized ID)	optional	
location_id		optional	

3.2.3 Splitting the metadata across several files

Sometimes, access to parts of the metadata should be limited to a list of authorized users. This can be achieved by moving confidential information out of the main notebook to a separate CSV file to be only delivered to authorized users. These additional files should be placed according to the table below:

Table 3: Additional metadata

data	main notebook	location of additional notebooks
children	`metadata/children.csv`	`metadata/children/`
recordings	`metadata/recordings.csv`	`metadata/recordings/`

There can be as many additional notebooks as necessary, and recursion is permitted.

This is also useful if your metadata includes many columns and you'd like to spread it across several dataframes. This can also be used to deliver survey data in a separate file.

Note: In case two or more notebooks contain the same column, the files whose names come first in alphabetical order will prevail while loading the dataset with our package. For instance, if `child_dob` is specified in both `metadata/recordings/0_private.csv` and `metadata/recordings/1_public.csv`, the values in the former file will prevail if it is available. This is useful when anonymized values for a certain parameter still need to be shared, but should be replaced with the true values for those who have access to the full dataset.

Warning: For recursive metadata, two dataframes cannot share the same basename. For instance, if one dataframe is located at *metadata/children/dates-of-birth.csv*, an error will be thrown if another dataframe exists at `metadata/children/private/dates-of-birth.csv`.

3.3 Annotations

Upon importation, annotations are converted to standardized CSV dataframes (using built-in or custom ingestors) and registered into an index. The index of annotations stores the list of each interval that has been annotated for each annotator. This allows a number of functionalities such as the quick computation of the intersection of the portions of audio covered by a given set of annotators.

3.3.1 Annotations format

The package provides functions to convert any annotation into the following CSV format, with one row per segment (e.g. per vocalization event):

Table 4: Annotations format

Name	Description	Required?	Format
raw_filename	raw annotation path relative, relative to *annotations/<set>/raw*	**required**	

continues on next page

Table 4 – continued from previous page

Name	Description	Required?	Format
segment_onset	segment onset timestamp in milliseconds (since the start of the recording)	**required**	`([0-9]+)`
segment_offset	segment end time in milliseconds (since the start of the recording)	**required**	`([0-9]+)`
speaker_id	identity of speaker in the annotation	optional	
speaker_type	class of speaker (FEM = female adult, MAL = male adult, CHI = key child, OCH = other child)	optional	FEM, MAL, CHI, OCH, NA
ling_type	1 if the vocalization contains at least a vowel (ie canonical or non-canonical), 0 if crying or laughing	optional	1, 0, NA
vcm_type	vocal maturity defined as: C (canonical), N (non-canonical), Y (crying) L (laughing), J (junk)	optional	C, N, Y, L, J, NA
lex_type	W if meaningful, 0 otherwise	optional	W, 0, NA
mwu_type	M if multiword, 1 if single word – only filled if lex_type==W	optional	M, 1, NA
addressee	T if target-child-directed, C if other-child-directed, A if adult-directed, U if uncertain or other. Multiple values should be sorted and separated by commas	optional	T, C, A, U, NA
transcription	orthographic transcription of the speach	optional	
phonemes	amount of phonemes	optional	`(\d+(\.\d+)?)`
syllables	amount of syllables	optional	`(\d+(\.\d+)?)`

continues on next page

Table 4 – continued from previous page

Name	Description	Required?	Format
words	amount of words	optional	`(\d+(\.\d+)?)`
lena_block_type	whether regarded as part as a pause or a conversation by LENA	optional	pause, CM, CIC, CIOCX, CIOCAX, AMF, AICF, AIOCF, AIOCCXF, AMM, AICM, AIOCM, AIOCCXM, XM, XIOCC, XIOCA, XIC, XIOCAC
lena_block_number	number of the LENA pause/conversation the segment belongs to	optional	`(\d+(\.\d+)?)`
lena_conv_status	LENA conversation status	optional	BC, RC, EC
lena_response_count	LENA turn count within block	optional	`(\d+(\.\d+)?)`
lena_conv_floor_type	(FI): Floor Initiation, (FH): Floor Holding	optional	FI, FH
lena_conv_turn_type	LENA turn type	optional	TIFI, TIMI, TIFR, TIMR, TIFE, TIME, NT

continues on next page

Table 4 – continued from previous page

Name	Description	Required?	Format
lena_speaker	LENA speaker type	optional	TVF, FAN, OLN, SIL, NOF, CXF, OLF, CHF, MAF, TVN, NON, CXN, CHN, MAN, FAF
utterances_count	utterances count	optional	`(\d+(\.\d+)?)`
utterances_length	utterances length	optional	`([0-9]+)`
non_speech_length	non-speech length	optional	`([0-9]+)`
average_db	average dB level	optional	`(\-?)(\d+(\.\d+)?)`
peak_db	peak dB level	optional	`(\-?)(\d+(\.\d+)?)`
child_cry_vfx_len	childCryVfxLen	optional	`([0-9]+)`
utterances	LENA utterances details (json)	optional	
cries	cries (json)	optional	
vfxs	Vfx (json)	optional	

Custom columns may be used, although they should be documented somewhere in your dataset.

3.3.2 Annotations index

Warning: The index is maintained through the package functions only; it should never be updated by hand.

Annotations are indexed in one unique dataframe located at `/metadata/annotations.csv`, with the following format :

Table 5: Annotations metadata

Name	Description	Required?	Format
set	name of the annotation set (e.g. VTC, annotator1, etc.)	**required**	
recording_filename	recording filename as specified in the recordings index	**required**	
time_seek	shift between the timestamps in the raw input annotations and the actual corresponding timestamps in the recordings (in milliseconds)	**required**	`(\-?)([0-9]+)`
range_onset	covered range onset timestamp in milliseconds (since the start of the recording)	**required**	`[0-9]+`
range_offset	covered range offset timestamp in milliseconds (since the start of the recording)	**required**	`[0-9]+`
raw_filename	annotation input filename location, relative to *annotations/<set>/raw*	**required**	True
annotation_filename	output formatted annotation location, relative to `annotations/<set>/converted (automatic column, don't specify)	optional	True
imported_at	importation date (automatic column, don't specify)	optional	`%Y-%m-%d %H:%M:%S`
package_version	version of the package used when the importation was performed	optional	`[0-9]+\.[0-9]+\.[0-9]+`
error	error message in case the annotation could not be imported	optional	

Below is shown an example of an index file (some uninformative columns were hidden for clarity). In this case, one recording has been fully annotated using the Voice Type Classifier (vtc), and partially annotated by two humans (LM and SP). These humans have both annotated the same seven 15 second clips.

set	recording_filename	time_seek	range_onset	range_offset	raw_filename	format	annotation_filename
vtc	A730/A730_001105.wav	0	0	42764250	A730/A730_001105_vtc.rttm	vtc_rttm	A730/A730_001105_0_42764250.csv
eaf_2021/SP	A730/A730_001105.wav	0	2910000	2925000	A730_001105.eaf	eaf	A730/A730_001105_2910000_2925000.csv
eaf_2021/SP	A730/A730_001105.wav	0	4680000	4695000	A730_001105.eaf	eaf	A730/A730_001105_4680000_4695000.csv
eaf_2021/SP	A730/A730_001105.wav	0	4695000	4710000	A730_001105.eaf	eaf	A730/A730_001105_4695000_4710000.csv
eaf_2021/SP	A730/A730_001105.wav	0	14055000	14070000	A730_001105.eaf	eaf	A730/A730_001105_14055000_14070000.csv
eaf_2021/SP	A730/A730_001105.wav	0	15030000	15045000	A730_001105.eaf	eaf	A730/A730_001105_15030000_15045000.csv
eaf_2021/SP	A730/A730_001105.wav	0	36465000	36480000	A730_001105.eaf	eaf	A730/A730_001105_36465000_36480000.csv
eaf_2021/SP	A730/A730_001105.wav	0	39450000	39465000	A730_001105.eaf	eaf	A730/A730_001105_39450000_39465000.csv
eaf_2021/LM	A730/A730_001105.wav	0	2910000	2925000	A730_001105.eaf	eaf	A730/A730_001105_2910000_2925000.csv
eaf_2021/LM	A730/A730_001105.wav	0	4680000	4695000	A730_001105.eaf	eaf	A730/A730_001105_4680000_4695000.csv
eaf_2021/LM	A730/A730_001105.wav	0	4695000	4710000	A730_001105.eaf	eaf	A730/A730_001105_4695000_4710000.csv
eaf_2021/LM	A730/A730_001105.wav	0	14055000	14070000	A730_001105.eaf	eaf	A730/A730_001105_14055000_14070000.csv
eaf_2021/LM	A730/A730_001105.wav	0	15030000	15045000	A730_001105.eaf	eaf	A730/A730_001105_15030000_15045000.csv
eaf_2021/LM	A730/A730_001105.wav	0	36465000	36480000	A730_001105.eaf	eaf	A730/A730_001105_36465000_36480000.csv
eaf_2021/LM	A730/A730_001105.wav	0	39450000	39465000	A730_001105.eaf	eaf	A730/A730_001105_39450000_39465000.csv

3.3.3 Annotation importation input format

The annotations importation script (*Bulk importation*) and python method (`ChildProject.annotations.AnnotationManager.import_annotations()`) take a dataframe of the following format as an input:

Table 6: Input annotations

Name	Description	Required?	Format
set	name of the annotation set (e.g. VTC, annotator1, etc.)	**required**	
recording_filename	recording filename as specified in the recordings index	**required**	
time_seek	shift between the timestamps in the raw input annotations and the actual corresponding timestamps in the recordings (in milliseconds)	**required**	`(\-?)([0-9]+)`
range_onset	covered range onset timestamp in milliseconds (since the start of the recording)	**required**	`[0-9]+`
range_offset	covered range offset timestamp in milliseconds (since the start of the recording)	**required**	`[0-9]+`
raw_filename	annotation input filename location, relative to *annotations/<set>/raw*	**required**	True
format	input annotation format	optional	csv, vtc_rttm, vcm_rttm, alice, its, TextGrid, eaf, cha, NA
filter	source file to filter in (for rttm and alice only)	optional	

Note: In order to avoid rounding errors, all timestamps are integers, expressed in milliseconds.

3.4 Documentation

An important aspect of a dataset is its documentation. Documentation includes:

- authorship, references, contact information
- a description of the corpus (population, collection process, etc.)
- instructions to re-use the data
- description of the data itself (e.g. a definition of each metadata field)

We currently do not provide a format for *all* these annotations. It is up to you to decide how to provide users with each of these information.

However, we suggest several options below.

3.4.1 Metadata and annotations

The ChildProject package supports a machine-readable format to describe the contents of the metadata and the annotations.

This format consists in CSV dataframe structured according to the following table:

Table 7: Machine-readable documentation

Name	Description	Required?	Format
variable	name of the variable	**required**	
description	a definition of this field	**required**	
values	a summary of authorized values	optional	
scope	which group of users has access to it	optional	
annotation_set	for annotations: which set(s) contain this variable	optional	

3.4.2 Authorship

We recommend DataCite's .yaml format (see here)

CHAPTER

FOUR

GETTING SOME DATA

You can either have some data of your own that you would like to use the package on, or you may know of some datasets that are already in this format that you'd like to reuse.

It may be easier to start with an extant dataset. Here is the list that we know exists. Please note that the large majority of these data are NOT public, and thus if you cannot retrieve them, this means you need to get in touch with the data managers.

4.1 Public data sets

We have prepared a public dataset for testing purposes which is based on the VanDam Public Daylong HomeBank Corpus; VanDam, Mark (2018). VanDam Public Daylong HomeBank Corpus. doi:10.21415/T5388S.

4.2 From the LAAC team

Table 1: List of LAAC datasets

Name	Authors	Location	Recordings	Duration (h)
Namibia	Gandhi	https://github.com/LAAC-LSCP/namibia-data	113	1449
Solomon	Sarah	https://github.com/LAAC-LSCP/solomon-data	388	5954
Tsimane 2017		https://github.com/LAAC-LSCP/tsimane2017-data	41	556
png 2019		https://github.com/LAAC-LSCP/png2019-data	51	760
Vanuatu		unavailable	53	289

4.3 EL1000

The EL1000 dataset contains several corpora accessible upon request.

4.4 Other private datasets

We know of no other private datasets at present, but we hope one day to be able to use datalad's search feature

CHAPTER

FIVE

EXAMPLE PROJECTS

This is an (ever-growing) list of projets that are known to rely on ChildProject.

Table 1: Known projects

Project	Description
Vandam-Daylong	Copy of Vandam-Daylong in ChildProject format
EL1000	Dataset containing 16 corpora, derived metrics and reliabiliy estimations
Reproducible paper example	Example of a reproducible paper using DataLad and ChildProject
Zooniverse campaign template	A template with instructions for launching a campaign of annotations on the crowd-sourcing platform Zooniverse

Let us know about your own projects that you would like to share!

CHAPTER SIX

HOW TO REUSE GIN DATASETS

Our datasets are managed with datalad. Datalad allows the versioning and distribution of large datasets. Datalad relies on another tool called git-annex, which itself is an extension of git providing support for large file versioning with a high flexibility.

We host the data on GIN. GIN's interface is really similar to GitHub, but unlike the latter, GIN can handle our large files.

6.1 Installing datalad

The DataLad handbook provides extensive instructions for the installation of DataLad in their handbook.

If you have admin rights and you are working on Linux or Mac, the following should work:

1. Install git-annex using `apt install git-annex` (linux) or `brew install git-annex` (mac). Git-annex is available by default on Oberon.
2. Install datalad with pip : `pip3 install datalad`

Note: If you are having permission issues, consider using python virtual environments or conda (see DataLad's handbook). Otherwise, refer to your system administrator.

6.2 Setup your GIN account

Most repositories are private, and thus require authentication. We recommend that you always use SSH authentication and we will only provide instructions for this case.

Before anything, you will need to create an account on GIN, and to link your SSH public key to your GIN account.

1. Create an account on GIN
2. Copy your SSH public key (usually located in `~/.ssh/id_rsa.pub`)
3. Go to GIN > Settings > SSH Keys
4. Click on the blue button 'Add a key' and paste your public key where requested.

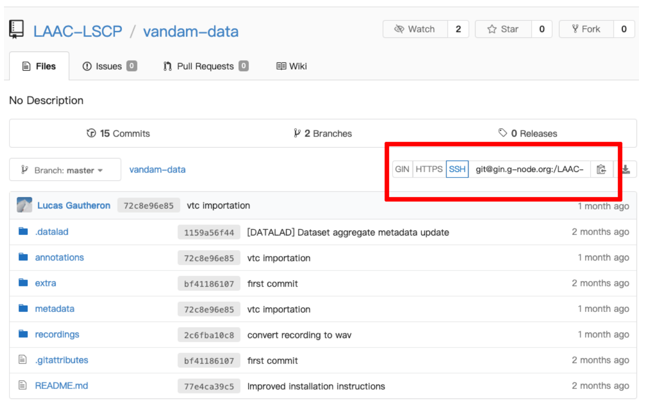
Note: Remember to communicate your username to the data administrator before you try to access the data in order for him to grant you permissions.

Note: You can configure as many keys as necessary. This is useful when you need to access GIN from different locations with different SSH keys (e.g. from your lab cluster, or from your own laptop).

Note: You may consider enabling the Keychain (append `~/.ssh/config` with `UseKeychain yes`) if you are prompted for your SSH passphrase everytime.

6.2.1 Installing a dataset

Installing a dataset can be done with the *datalad install* command. The input is the SSH location of the dataset. It can be found on the page of the repository on GIN:



Lucas Gautheron	72c8e96e85	vtc importation	1 month ago
.datalad	1159a56f44	[DATALAD] Dataset aggregate metadata update	2 months ago
annotations	72c8e96e85	vtc importation	1 month ago
extra	bf41186107	first commit	2 months ago
metadata	72c8e96e85	vtc importation	1 month ago
recordings	2c6fba10c8	convert recording to wav	1 month ago
.gitattributes	bf41186107	first commit	2 months ago
README.md	77e4ca39c5	Improved installation instructions	2 months ago

Fig. 1: A GIN dataset.

For instance, the VanDam public dataset (available on GIN) can be installed with the following command:

```
datalad install git@gin.g-node.org:/LAAC-LSCP/vandam-data.git
cd vandam-data
```

Datasets that contain subdatasets can be installed recursively using the -r switch. This is the case of the EL1000 dataset:

```
datalad install git@gin.g-node.org:/EL1000/EL1000.git
cd EL1000
```

Warning: Some datasets may require additional configuration steps. Pay attention to the README before you start using a dataset.

That's it ! Your dataset is ready to go. By default, large files do not get downloaded automatically. See the next section for help with downloading those files.

6.3 Downloading large files

Files can be retrieved using `datalad get [path]`. For instance, `datalad get recordings` will download all recordings.

Note: Technically speaking, the large files in your repository are symbolic links pointing to their actual location, somewhere under *.git*. You can ignore that and read/copy the content of these files as if they where actual files.

Warning: If you want to *edit* the content of a large file, you will need to unlock it beforehand, using `datalad unlock`, e.g.: `datalad unlock annotations/vtc/converted`.

6.4 Updating a dataset

A dataset can be updated from the sources using `git pull` together with `datalad update`.

6.5 Contributing

6.5.1 Pushing changes to a dataset

You can save local changes to a dataset with `datalad save [path] -m "commit message"`. For instance :

```
datalad save annotations/vtc/raw -m "adding vtc rttms"
```

`datalad save` is analoguous to a combination of `git add` and `git commit`.

These changes still have to be pushed, which can be done with :

```
datalad push
```

CHAPTER

SEVEN

CONVERTING A DATASET

This tutorial will guide you through the steps for the conversion of an existing dataset. We will use the VanDam-Daylong dataset from HomeBank as an example.

7.1 Set-up datalad and child-project

Make sure you have followed the *Installation* instructions before proceeding.

7.2 Create a dataset

The first step is to create a new dataset named vandam-data :

```
datalad create vandam-data
cd vandam-data
```

So far, the dataset contains nothing but hidden files:

```
$ ls -A
.datalad    .git        .gitattributes
```

Now, we would like to get the data from https://homebank.talkbank.org/access/Public/VanDam-Daylong.html, convert it to our standards, and then publish it.

7.3 Gather and sort the files

The first step is to create the directories:

```
mkdir metadata # Create the metadata subfolder
mkdir -p recordings/raw # Create the subfolders for raw recordings
mkdir annotations # Create the subfolder for annotations
mkdir extra # Create the subfolder for extra data (that are neither metadata, recordings␣
↪or annotations)
touch extra/.gitignore # Make sure the directory is present even though it's empty
```

Then, download the original data-set from HomeBank.

The audio first:

```
curl https://media.talkbank.org/homebank/Public/VanDam-Daylong/BN32/BN32_010007.mp3 -o␣
↪recordings/raw/BN32_010007.mp3
```

Now let's get the annotations.

```
curl https://homebank.talkbank.org/data/Public/VanDam-Daylong.zip -o VanDam-Daylong.zip
unzip VanDam-Daylong.zip
rm VanDam-Daylong.zip # Remove the zip archive
```

Let's explore the contents of VanDam-Daylong:

```
$ find . -not -path '*/\.*' -type f -print
./recordings/raw/BN32_010007.mp3
./VanDam-Daylong/BN32/0its/e20100728_143446_003489.its
./VanDam-Daylong/BN32/BN32_010007.cha
./VanDam-Daylong/0metadata.cdc
```

- `0metadata.cdc1` looks like some metadata file, so we will move it to `metadata/` :

```
mv VanDam-Daylong/0metadata.cdc metadata/
```

- `BN32_010007.cha` contains some transcriptions. Let's create a set of annotations `cha` and move it there :

```
mkdir -p annotations/cha/raw
mv VanDam-Daylong/BN32/BN32_010007.cha annotations/cha/raw
```

- `e20100728_143446_003489.its` contains diarization and other information such as word counts. Let's create another set of annotations for it. And for the sake of consistency, we'll rename it `BN32_010007.its`.

```
mkdir -p annotations/its/raw
mv VanDam-Daylong/BN32/0its/e20100728_143446_003489.its annotations/its/raw/BN32_010007.
↪its
```

Now we've got all the files. Let's try to run the validation on the dataset:

```
$ child-project validate .

Traceback (most recent call last):
  File "/Users/acristia/anaconda3/bin/child-project", line 8, in <module>
    sys.exit(main())
  File "/Users/acristia/anaconda3/lib/python3.7/site-packages/ChildProject/cmdline.py",␣
↪line 241, in main
    args.func(args)
  File "/Users/acristia/anaconda3/lib/python3.7/site-packages/ChildProject/cmdline.py",␣
↪line 39, in validate
    errors, warnings = project.validate(args.ignore_files)
  File "/Users/acristia/anaconda3/lib/python3.7/site-packages/ChildProject/projects.py",␣
↪line 102, in validate
    self.read()
  File "/Users/acristia/anaconda3/lib/python3.7/site-packages/ChildProject/projects.py",␣
↪line 86, in read
    self.children = self.ct.read(lookup_extensions = ['.csv', '.xls', '.xlsx'])
  File "/Users/acristia/anaconda3/lib/python3.7/site-packages/ChildProject/tables.py",␣
↪line 65, in read
```

(continues on next page)

(continued from previous page)

```
    raise Exception("could not find table '{}'".format(self.path))
Exception: could not find table './metadata/children'
```

This is expected. The validation should fail, because the metadata is missing. We need to store the metadata about the children and the recordings in a way that meets the specifications (see *Metadata*).

7.4 Create the metadata

We need two metadata files:

- **`metadata/recordings.csv`, which links each recording to their associate metadata** (recording date and time, recording device, etc.)
- `metadata/children.csv`, which stores the information about the participants.

Let's start with the recordings metadata. `metadata/recordings.csv` should at least have the following columns: experiment, child_id, date_iso, start_time, recording_device_type, recording_filename. The .its file contains (`annotations/its/raw/BN32_010007.its`) precious information about when the recording started:

```
<Recording num="1" startClockTime="2010-07-24T11:58:16Z" endClockTime="2010-07-
↪25T01:59:20Z" startTime="PT0.00S" endTime="PT50464.24S">
```

The 'Z' suffix in the clock times indicate they correspond to the UTC timezone. However, the metadata should contain local times only. The difference between the two is 5 hours, according to the following line in the .its file:

```
<TransferTime LocalTime="2010-07-28T14:34:46" TimeZone="CST" UTCTime="2010-07-28T19:34:46
↪" />
```

Therefore, the recording started on 2010-07-24, at 06:58 (local time).

In order to reflect that information, the recordings CSV metadata should look like this (we have decided that the only child of the dataset should have ID '1'):

Table 1: Recordings metadata

experiment	child_id	date_iso	start_time	recording_device_type	recording_filename
vandam-daylong	1	2010-07-24	11:58	lena	BN32_010007.mp3

We have prepared it for you. Download `recordings.csv` `here`, and save it in the `metadata` subfolder of your dataset. You can check its content by issuing the following command:

```
$ cat metadata/recordings.csv
experiment,child_id,date_iso,start_time,recording_device_type,recording_filename
vandam-daylong,1,2010-07-24,11:58,lena,BN32_010007.mp3
```

Now, let us proceed to the children metadata. The only fields that are required are: experiment, child_id and child_dob. The .its file also contains some information about the child:

```
<ChildInfo algorithmAge="P12M" gender="F" />
```

She was a 12 month old girl at the time of the recording. We can thus assign her a calculated date of birth: 2009-07-24. We will set `dob_criterion` to "extrapolated" to keep track of the fact that the date of birth was calculated from the approximate age at recording. We will also set `dob_accuracy` to 'month' for that child.

In other words, the children metadata CSV file should look like this:

Table 2: Children metadata

experiment	child_id	child_dob	dob_criterion	dob_accuracy
vandam-daylong	1	2009-07-24	extrapolated	month

We have prepared it for you. Download `children.csv` here, and save it in the `metadata` subfolder of your dataset. You can check its content by issuing the following command:

```
$ cat metadata/children.csv
experiment,child_id,child_dob,dob_criterion,dob_accuracy
vandam-daylong,1,2009-07-24,extrapolated,month
```

We can now make sure that they are no errors by running the validation command again:

```
child-project validate .
```

No error occurs.

Note: The metadata can be enriched with many more columns. See *Metadata* for standard columns. You can add as many extra, custom columns as you need.

7.5 Save the changes locally

A DataLad dataset is essentially a git repository, with the large files being handled by git-annex. Some of the files (usually the small, text files such as metadata and scripts) ought to be versionned with git, and the larger files or binary files should be stored in the *annex*.

The rules to decide what files should be stored which way can be set in the `.gitattributes` file. You should fill it will the following content:

```
* annex.backend=MD5E
**/.git* annex.largefiles=nothing
scripts/* annex.largefiles=nothing
metadata/* annex.largefiles=nothing
recordings/converted/* annex.largefiles=((mimeencoding=binary))
```

These rules will version all the files under `scripts/` and `metadata/`, as well as the text files inside of `recordings/converted/`. By default, the other files will be put in the annex.

The changes can now be saved. This can be done with datalad save. `datalad save` is equivalent to a combination of `git add` and `git commit` in one go. It decides, based on the rules in `.gitattributes`, whether to store files with git or git-annex.

```
datalad save . -m "first commit"
```

However, so far, your changes remain local, and your dataset still needs to be published into a *sibling* to be shared with others.

7.6 Processing

You can do some processing on the dataset. For instance, you can compute the duration of the recording, and update the metadata with this information. This is easily done with:

```
child-project compute-durations .
```

Now metadata/recordings.csv became:

```
$ cat metadata/recordings.csv
experiment,child_id,date_iso,start_time,recording_device_type,recording_filename,duration
vandam-daylong,1,2010-07-24,11:58,lena,BN32_010007.mp3,50464512
```

You can also convert and index the its annotation:

```
child-project import-annotations . --set its \
  --recording_filename BN32_010007.mp3 \
  --time_seek 0 \
  --range_onset 0 \
  --range_offset 50464512 \
  --raw_filename BN32_010007.its \
  --format its
```

And save the changes again:

```
datalad save . -m "its"
```

7.7 Publish the dataset

7.7.1 Where to publish my dataset ?

DataLad allows you to publish your datasets on large number of storage providers, including Amazon S3, Dropbox, Google Cloud Storage, Microsoft Azure Blob Storage, etc., each having their own advantages and limitations. It is also possible to publish to several platforms, as we do with our own datasets.

The table below summarises the features of a few storage supports. The solutions described here are by no mean exhaustive, but they are easy to generalize.

- Platforms that support Git store the .git files and will allow you to clone the datasets from them with `datalad install`
- Platforms that support Large Files will allow you to store and distribute the large or binary files that are stored with git-annex instead of the regular git files (such as scripts and metadata)

It is necessary to use a platform or a combination of platforms that supports both. We recommend the use of GIN, although you should always push your data to another platform as backup.

Provider	Git	Large Files	Authentication	Permissions	Cost	Quota
GIN	Yes	Yes	HTTPS/SSH	ACL	Free below ~10 TB	None
SSH server	Yes	Yes	SSH	Unix	-	None
GitHub	Yes	No	HTTPS/SSH	ACL	Free	~1 GB
GitLab	Yes	No	HTTPS/SSH	ACL	Free	~1 GB
Amazon S3	No	Yes	API	IAM	~4$/TB/month	None
Nextcloud	No	Yes	WebDav	ACL	-	None
OSF.io	Yes	Yes*	Token	ACL	Free	5 GB

Note: DataLad uses git-annex, which naturally handles encryption. This is particularly useful when using third-party providers such as Amazon S3.

7.7.2 Publish to GIN

Note: Before anything, you will need to create an account on GIN, and to link your SSH public key to your GIN account.

1. Create a new repository from GIN's web interface.
2. Copy the SSH url of your repository to your clipboard, e.g.: `git@gin.g-node.org:/<username>/<reponame>.git`
3. Add a datalad sibling pointing to this repository:

```
datalad siblings add \
   --name gin \
   --url git@gin.g-node.org:/<username>/<reponame>.git
```

4. Push the data to GIN:

```
datalad push --to gin
```

7.7.3 Publish to a SSH server

If you have access to a SSH server with enough storage capacity, you can use it to store and share the dataset. This is done with the datalad create-sibling command:

```
datalad create-sibling [-h] [-s [NAME]] [--target-dir PATH] [--target-url URL] [--target-
↪pushurl URL] [--dataset DATASET] [-r] [-R LEVELS] [--existing MODE] [--shared
↪{false|true|umask|group|all|world|everybody|0xxx}] [--group GROUP] [--ui
↪{false|true|html_filename}] [--as-common-datasrc NAME] [--publish-by-default REFSPEC]␣
↪[--publish-depends SIBLINGNAME] [--annex-wanted EXPR] [--annex-group EXPR] [--annex-
↪groupwanted EXPR] [--inherit] [--since SINCE] [SSHURL]
```

For instance, you can create it (this is only to be done once) by issuing:

```
datalad create-sibling -s cluster --annex-wanted 'include=*' <ssh-server>:/remote/path/
↪to/the/dataset
```

cluster is the name of the sibling, and <ssh-server>:/remote/path/to/the/dataset is the SSH url of its destination. --annex-wanted 'include=*' implies that all large files will be published to this sibling by default.

Once the sibling has been created, the changes can be published:

```
datalad push --to cluster
```

That's it! People can now get your data from:

```
datalad install <ssh-server>:/remote/path/to/the/dataset
```

If --annex-wanted had not been set to 'include=*', the large files (i.e. annexed files) would not be published unless you asked for it explicitly with the --data flag:

```
datalad push --to cluster --data anything
```

7.7.4 Publish to GitHub

You first need to create the repository, which can be done in a straightforward way from the command line with datalad create-sibling-github:

```
datalad create-sibling-github [-h] [--dataset DATASET] [-r] [-R LEVELS] [-s NAME] [--
↪existing MODE] [--github-login NAME] [--github-organization NAME] [--access-protocol
↪{https|ssh}] [--publish-depends SIBLINGNAME] [--private] [--dryrun] REPONAME
```

For instance:

```
datalad create-sibling-github -s origin --access-protocol ssh vandam-daylong-demo
```

origin will be the local name of the sibling, and vandam-daylong-demo the name of the GitHub repository. Once the sibling has been created, you can publish the changes with datalad push:

```
datalad push --to origin
```

You should get a repository identical to this one.

Users can now install your dataset from GitHub:

```
datalad install https://github.com/LAAC-LSCP/vandam-daylong-demo.git
```

PS: we recommend that you do git push --set-upstream origin to set upstream to the GitHub sibling. Users who install your dataset from GitHub will not need to do this.

GitHub + SSH mirror to store the large files

Now, let's assume you have already created a SSH sibling as well for your dataset, and that it is named cluster. You can make sure that all changes to github are published to cluster as well, by setting the publish-depends property of the github sibling:

```
datalad siblings configure -s origin --publish-depends cluster
```

Now, datalad push --to origin will publish the changes to both cluster and github.

However, when the users install your dataset from GitHub, they will not have access to the cluster sibling unless you make it available to them, which can be done this way :

```
git annex
git annex initremote cluster type=git location=ssh://cluster.com/path/to/the/repository␣
↪autoenable=true
git annex enableremote cluster
git remote add origin git@github.com:LAAC-LSCP/vandam-daylong-demo.git
```

7.7.5 Publish on S3

Like other *git annex special remotes*, Amazon S3 will not support the git files, only the large files. It could be used together win GitHub as the primary host for your large files, or as a backup.

*For the sake of simplicity, we will not use encryption here, but git annex implements several*encryption schemes*which are easy to use.*

First, store your AWS credentiels into your environment variables, like this:

```
export AWS_ACCESS_KEY_ID="08TJMT99S3511WOZEP91"
export AWS_SECRET_ACCESS_KEY="s3kr1t"
```

You are now readyto create the s3 sibling. This is done directly through git-annex this time:

```
git annex initremote s3 chunk=100MiB type=S3 encryption=none datacenter=eu-west-3␣
↪embedcreds=no signature=v4
```

You can now publish the data with:

```
datalad push --to s3 --data anything
```

(Optional) You can set the S3 sibling to require that all large files should be stored on it:

```
datalad siblings configure -s s3 --annex-wanted 'include=*'
```

This will let DataLad publish all the large files automatically without setting `--data`:

```
datalad push --to s3
```

Let's assume your users will install the dataset from a GitHub repository. You should publish the information about the newly created S3 sibling on GitHub, which can be done with (provided you have set up your GitHub repository as described in the previous section):

```
datalad push --to github
```

Now, users will be able to get the data by issuing the following commands:

```
datalad install git@github.com:<your-username>/vandam-daylong-demo.git
git annex enableremote s3
datalad get *
```

With this configuration, they will need to setup their AWS credentials as you did. But it is possible to configure the sibling so that the credentials are encrypted and stored in the repository, so all users with authorized private keys will be able to get the data without this step.

7.7.6 Publish on OSF

DataLad has an extension to publish data on the Open Science Framework.

This extension supports the following modes:

Table 3: datalad create-sibling-osf modes

Mode	datalad install	large files	history	older files	human-readable project
`annex`	Yes	Yes	Yes	Yes	No
`export`	Yes	Yes	Yes	No	Yes
`gitonly`	Yes	No	Yes	No	No
`export-only`	No	Yes	No	Yes	Yes

The first step is to install the extension:

```
pip install datalad-osf --upgrade
```

We decide to use the `export` mode - but you can decide which best suits your needs from the table above. We can now create the sibling:

```
datalad create-sibling-osf --title "VanDam Demo" \
  --mode export \
  -s osf \
  --category data \
  --tag reproducibility \
  --public
```

You will be prompted your credentials in the process, which will require access tokens to be created from your osf.io account.

And finally we can push the data. This is done in two steps:

1. publishing the .git files so people can clone the dataset directly from OSF

```
datalad push --to osf
```

2. exporting a human-readable snapshot of the files to OSF

```
git-annex export HEAD --to osf-storage
```

CHAPTER

EIGHT

BASIC TOOLS

8.1 Data validation

This is typically done (repeatedly!) in the process of importing your data into our format for the first time, but you should also do this whenever you make a change to the dataset.

Looks for errors and inconsistency in the metadata, or for missing audios. The validation will pass if formatting instructions are met (see *Datasets structure*).

```
$ child-project validate /path/to/dataset --help
usage: child-project validate [-h] [--ignore-recordings] [--profile PROFILE]
                              [--annotations ANNOTATIONS [ANNOTATIONS ...]]
                              [--threads THREADS]
                              source

validate the consistency of the dataset returning detailed errors and warnings

positional arguments:
  source                project path

optional arguments:
  -h, --help            show this help message and exit
  --ignore-recordings   ignore missing audio files
  --profile PROFILE     which recording profile to validate
  --annotations ANNOTATIONS [ANNOTATIONS ...]
                        path to or name of each annotation set(s) to check
                        (e.g. 'vtc' or '/path/to/dataset/annotations/vtc')
  --threads THREADS     amount of threads to run on (only applies to
                        --annotations)
```

Example:

```
# validate the metadata and raw recordings
child-project validate /path/to/dataset

# validate the metadata only
child-project validate /path/to/dataset --ignore-recordings

# validate the metadata and the recordings of the 'standard' profile
# (in recordings/converted/standard)
child-project validate /path/to/dataset --profile standard
```

(continues on next page)

(continued from previous page)

```
# validate the metadata and all annotations within /path/to/dataset/annotations
child-project validate /path/to/dataset --ignore-recordings --annotations /path/to/
↪dataset/annotations/*

# validate the metadata and annotations from the 'textgrid' set
child-project validate /path/to/dataset --ignore-recordings --annotations /path/to/
↪dataset/annotations/textgrid/*
```

8.2 Dataset overview

An overview of the contents of a dataset can be obtained with the `child-project overview` command.

```
$ child-project overview --help
usage: child-project overview [-h] source

prints an overview of the contents of a given dataset

positional arguments:
  source      source data path

optional arguments:
  -h, --help  show this help message and exit
```

Example:

```
$ child-project overview .

recordings:
lena: 288.00 hours, 0/18 files locally available
olympus: 49.57 hours, 0/3 files locally available
usb: 223.42 hours, 0/20 files locally available

annotations:
alice: 560.99 hours, 0/40 files locally available
alice_vtc: 560.99 hours, 0/40 files locally available
eaf/nk: 1.47 hours, 0/88 files locally available
lena: 272.00 hours, 0/17 files locally available
textgrid/mm: 8.75 hours, 0/525 files locally available
vtc: 560.99 hours, 40/40 files locally available
```

8.3 Compute recordings duration

Compute recordings duration and store in into a column named ‘duration’ in the metadata.

```
$ child-project compute-durations /path/to/dataset --help
usage: child-project compute-durations [-h] [--profile PROFILE] [--force]
                                        source

creates a 'duration' column into metadata/recordings

positional arguments:
  source             source data path

optional arguments:
  -h, --help         show this help message and exit
  --profile PROFILE  which audio profile to use
  --force            overwrite if column exists
```

CHAPTER NINE

MANAGING ANNOTATIONS

Warning: You should never run two of the following commands in parallel. All of them need to be run sequentially, otherwise the index may get corrupted.

If you need to parallelize the processing to speed it up, you can use the `--threads` option, which is built-in in all of our tools that might require it.

9.1 Importation

9.1.1 Single annotation importation

Annotations can be imported one by one or in bulk. Annotation importation does the following :

1. Convert all input annotations from their original format (e.g. rttm, eaf, textgrid..) into the CSV format defined at format-input-annotations and stores them into `annotations/`.
2. Registers them to the annotation index at `metadata/annotations.csv`

Use `child-project import-annotations` to import a single annotation.

```
$ child-project import-annotations /path/to/dataset --help
usage: child-project import-annotations [-h] [--annotations ANNOTATIONS]
                                        [--threads THREADS] [--set SET]
                                        [--recording_filename RECORDING_FILENAME]
                                        [--time_seek TIME_SEEK]
                                        [--range_onset RANGE_ONSET]
                                        [--range_offset RANGE_OFFSET]
                                        [--raw_filename RAW_FILENAME]
                                        [--format {csv,vtc_rttm,vcm_rttm,alice,its,
↪TextGrid,eaf,cha,NA}]
                                        [--filter FILTER]
                                        source

convert and import a set of annotations

positional arguments:
  source                project path

optional arguments:
  -h, --help            show this help message and exit
```

(continues on next page)

(continued from previous page)

```
  --annotations ANNOTATIONS
                        path to input annotations dataframe (csv) [only for
                        bulk importation]
  --threads THREADS     amount of threads to run on
  --set SET             name of the annotation set (e.g. VTC, annotator1,
                        etc.)
  --recording_filename RECORDING_FILENAME
                        recording filename as specified in the recordings
                        index
  --time_seek TIME_SEEK
                        shift between the timestamps in the raw input
                        annotations and the actual corresponding timestamps in
                        the recordings (in milliseconds)
  --range_onset RANGE_ONSET
                        covered range onset timestamp in milliseconds (since
                        the start of the recording)
  --range_offset RANGE_OFFSET
                        covered range offset timestamp in milliseconds (since
                        the start of the recording)
  --raw_filename RAW_FILENAME
                        annotation input filename location, relative to
                        `annotations/<set>/raw`
  --format {csv,vtc_rttm,vcm_rttm,alice,its,TextGrid,eaf,cha,NA}
                        input annotation format
  --filter FILTER       source file to filter in (for rttm and alice only)
```

Example:

```
child-project import-annotations /path/to/dataset \
   --set eaf \
   --recording_filename sound.wav \
   --time_seek 0 \
   --raw_filename example.eaf \
   --range_onset 0 \
   --range_offset 300 \
   --format eaf
```

Find more information about the allowed values for each parameter, see format-input-annotations.

9.1.2 Bulk importation

Use this to do bulk importation of many annotation files.

```
child-project import-annotations /path/to/dataset --annotations /path/to/dataframe.csv
```

The input dataframe `/path/to/dataframe.csv` must have one entry per annotation to import, according to the format specified at format-input-annotations.

9.2 Rename a set of annotations

Rename a set of annotations. This will move the annotations themselves, and update the index (`metadata/annotations.csv`) accordingly.

```
$ child-project rename-annotations /path/to/dataset --help
usage: child-project rename-annotations [-h] --set SET --new-set NEW_SET
                                        [--recursive] [--ignore-errors]
                                        source

rename a set of annotations by moving the files and updating the index
accordingly

positional arguments:
  source              project path

optional arguments:
  -h, --help          show this help message and exit
  --set SET           set to rename
  --new-set NEW_SET   new name for the set
  --recursive         enable recursive mode
  --ignore-errors     proceed despite errors
```

Example:

```
child-project rename-annotations /path/to/dataset --set vtc --new-set vtc_1
```

9.3 Remove a set of annotations

This will deleted converted annotations associated to a given set and remove them from the index.

```
$ child-project remove-annotations /path/to/dataset --help
usage: child-project remove-annotations [-h] --set SET [--recursive] source

remove converted annotations of a given set and their entries in the index

positional arguments:
  source       project path

optional arguments:
  -h, --help   show this help message and exit
  --set SET    set to remove
  --recursive  enable recursive mode
```

```
child-project remove-annotations /path/to/dataset --set vtc
```

9.4 ITS annotations anonymization

LENA .its files might contain information that can help recover the identity of the participants, which may be undesired. This command anonymizes .its files, based on a routine by HomeBank.

```
$ child-project anonymize /path/to/dataset --help
usage: child-project anonymize [-h] --input-set INPUT_SET --output-set
                               OUTPUT_SET
                               [--replacements-json-dict REPLACEMENTS_JSON_DICT]
                               path

Anonymize a set of its annotations (`input_set`) and saves it as `output_set`.

positional arguments:
  path                  project path

optional arguments:
  -h, --help            show this help message and exit
  --input-set INPUT_SET
                        input annotation set
  --output-set OUTPUT_SET
                        output annotation set
  --replacements-json-dict REPLACEMENTS_JSON_DICT
                        path to the replacements configuration (json dict)
```

```
child-project anonymize /path/to/dataset --input-set lena --output-set lena/anonymous
```

9.5 Merge annotation sets

Some processing tools use pre-existing annotations as an input, and label the original segments with more information. This is typically the case of ALICE, which labels segments generated by the VTC. In this case, one might want to merge the ALICE and VTC annotations altogether. This can be done with `child-project merge-annotations`.

```
$ child-project merge-annotations /path/to/dataset --help
usage: child-project merge-annotations [-h] --left-set LEFT_SET --right-set
                                       RIGHT_SET --left-columns LEFT_COLUMNS
                                       --right-columns RIGHT_COLUMNS
                                       --output-set OUTPUT_SET
                                       [--threads THREADS]
                                       source

merge segments sharing identical onset and offset from two sets of annotations

positional arguments:
  source                project path

optional arguments:
  -h, --help            show this help message and exit
  --left-set LEFT_SET   left set
  --right-set RIGHT_SET
```

(continues on next page)

(continued from previous page)

```
                        right set
  --left-columns LEFT_COLUMNS
                        comma-separated columns to merge from the left set
  --right-columns RIGHT_COLUMNS
                        comma-separated columns to merge from the right set
  --output-set OUTPUT_SET
                        name of the output set
  --threads THREADS     amount of threads to run on (default: 1)
```

```
child-project merge-annotations /path/to/dataset \
--left-set vtc \
--right-set alice \
--left-columns speaker_id,ling_type,speaker_type,vcm_type,lex_type,mwu_type,addresseee,
↪transcription \
--right-columns phonemes,syllables,words \
--output-set alice_vtc
```

9.6 Intersect annotations

In order to combine annotations from different annotators, or to compare them, it is necessary to calculate which portions of the audio have been annotated by all of them. This can be done from the command-line interface:

```
$ child-project intersect-annotations /path/to/dataset --help
usage: child-project intersect-annotations [-h] --destination DESTINATION
                                           --sets SETS [SETS ...]
                                           [--annotations ANNOTATIONS]
                                           source

calculate the intersection of the annotations belonging to the given sets

positional arguments:
  source                project path

optional arguments:
  -h, --help            show this help message and exit
  --destination DESTINATION
                        output CSV dataframe destination
  --sets SETS [SETS ...]
                        annotation sets to intersect
  --annotations ANNOTATIONS
                        path a custom input CSV dataframe of annotations to
                        intersect. By default, the whole index of the project
                        will be used.
```

Example:

```
child-project intersect-annotations /path/to/dataset \
--sets its textgrid/annotator1 textgrid/annotator2 textgrid/annotator3 \
--destination intersection.csv
```

The output dataframe has the same format as the annotations index (see *Annotations index*).

CHAPTER

TEN

METRICS EXTRACTION

10.1 Overview

This package allows to extract metrics that are commonly used from annotations produced by the LENA or other pipelines.

```
$ child-project metrics --help
usage: child-project metrics [-h] [--recordings RECORDINGS]
                             [--by {recording_filename,session_id,child_id}]
                             [-f FROM_TIME] [-t TO_TIME]
                             path destination {lena,aclew,period} ...

positional arguments:
  path                  path to the dataset
  destination           segments destination
  {lena,aclew,period}   pipeline
    lena                LENA metrics
    aclew               LENA metrics
    period              LENA metrics

optional arguments:
  -h, --help            show this help message and exit
  --recordings RECORDINGS
                        path to a CSV dataframe containing the list of
                        recordings to sample from (by default, all recordings
                        will be sampled). The CSV should have one column named
                        recording_filename.
  --by {recording_filename,session_id,child_id}
                        units to sample from (default behavior is to sample by
                        recording)
  -f FROM_TIME, --from-time FROM_TIME
                        time range start in HH:MM format (optional)
  -t TO_TIME, --to-time TO_TIME
                        time range end in HH:MM format (optional)
```

The list of supported metrics is shown below:

Variable	Description	pipelines
voc_fem/mal/och_ph	number of vocalizations by different talker types per hour	ACLEW,LENA,Period
voc_dur_fem/mal/och_ph	total duration of vocalizations by different talker types in seconds per hour	ACLEW,LENA,Period
avg_voc_dur_fem/mal/och	average vocalization length (conceptually akin to MLU) by different talker types	ACLEW,LENA,Period
wc_adu_ph	adult word count (collapsing across males and females)	ACLEW,LENA
wc_fem/mal_ph	adult word count by different talker types	ACLEW,LENA
sc_adu_ph	adult syllable count (collapsing across males and females)	ACLEW
sc_fem/mal_ph	adult syllable count by different talker types	ACLEW
pc_adu_ph	adult phoneme count (collapsing across males and females)	ACLEW
pc_fem/mal_ph	adult phoneme count by different talker types	ACLEW
freq_n	frequency of child voc out of all vocs based on number of vocalizations	ACLEW,LENA
freq_dur	frequency of child voc out of all vocs based on duration of vocalizations	ACLEW,LENA
cry_voc_chi_ph	number of child vocalizations that are crying	ACLEW,LENA
can_voc_chi_ph	number of child vocs that are canonical	ACLEW
non_can_vpc_chi_ph	number of child vocs that are non-canonical	ACLEW
sp_voc_chi_ph	number of child vocs that are speech-like (can+noncan for ACLEW)	ACLEW,LENA
cry_voc_dur_chi_ph	total duration of child vocalizations that are crying	ACLEW,LENA
can_voc_dur_chi_ph	total duration of child vocs that are canonical	ACLEW
non_can_voc_dur_chi_ph	total duration of child vocs that are non-canonical	ACLEW
sp_voc_dur_chi_ph	total duration of child vocs that are speech-like (can+noncan for ACLEW)	ACLEW,LENA
avg_cry_voc_dur_chi	average duration of child vocalizations that are crying	ACLEW,LENA
avg_cran_voc_dur_chi	average duration of child vocs that are canonical	ACLEW
avg_non_can_voc_dur_chi	average duration of child vocs that are non-canonical	ACLEW
avg_sp_voc_dur_chi	average duration of child vocs that are speech-like (can+noncan for ACLEW)	ACLEW,LENA
lp_n	linguistic proportion = (speech)/(cry+speech) based on number of vocalizations	ACLEW,LENA
cp_n	canonical proportion = canonical /(can+noncan) based on number of vocalizations	ACLEW
lp_dur	linguistic proportion = (speech)/(cry+speech) based on duration of vocalizations	ACLEW,LENA
cp_dur	canonical proportion = canonical /(can+noncan) based on duration of vocalizations	ACLEW

Note: Average rates are expressed in counts/hour (for events) or in seconds/hour (for durations).

10.2 LENA Metrics

```
$ child-project metrics /path/to/dataset output.csv lena --help
usage: child-project metrics path destination lena [-h]
                                                   [--types {TVF,FAN,OLN,SIL,NOF,CXF,OLF,
↪CHF,MAF,TVN,NON,CXN,CHN,MAN,FAF} [{TVF,FAN,OLN,SIL,NOF,CXF,OLF,CHF,MAF,TVN,NON,CXN,CHN,
↪MAN,FAF} ...]]
                                                   [--threads THREADS]
                                                   set

positional arguments:
  set                   name of the LENA its annotations set

optional arguments:
  -h, --help            show this help message and exit
```

(continues on next page)

(continued from previous page)

```
  --types {TVF,FAN,OLN,SIL,NOF,CXF,OLF,CHF,MAF,TVN,NON,CXN,CHN,MAN,FAF} [{TVF,FAN,OLN,
↪SIL,NOF,CXF,OLF,CHF,MAF,TVN,NON,CXN,CHN,MAN,FAF} ...]
                        list of LENA vocalizaton types to include
  --threads THREADS     amount of threads to run on
```

10.3 ACLEW Metrics

```
$ child-project metrics /path/to/dataset output.csv aclew --help
usage: child-project metrics path destination aclew [-h] [--vtc VTC]
                                                    [--alice ALICE]
                                                    [--vcm VCM]
                                                    [--threads THREADS]

optional arguments:
  -h, --help         show this help message and exit
  --vtc VTC          vtc set
  --alice ALICE      alice set
  --vcm VCM          vcm set
  --threads THREADS  amount of threads to run on
```

10.4 Period-aggregated metrics

The Period Metrics pipeline aggregates vocalizations for each time-of-the-day-unit based on a period specified by the user. For instance, if the period is set to `15Min` (i.e. 15 minutes), vocalization rates will be reported for each recording and time-unit (e.g. 09:00 to 09:15, 09:15 to 09:30, etc.).

The output dataframe has $r \times p$ rows, where r is the amount of recordings (or children if the `-by` option is set to `child_id`), and p is the amount of time-bins per day (i.e. $24 \times 4 = 96$ for a 15-minute period).

The output dataframe includes a `period` column that contains the onset of each time-unit in HH:MM:SS format. The `duration` columns contains the total amount of annotations covering each time-bin, in milliseconds.

If `--by` is set to e.g. `child_id`, then the values for each time-bin will be the average rates across all the recordings of every child.

```
$ child-project metrics /path/to/dataset output.csv period --help
usage: child-project metrics path destination period [-h] --set SET --period
                                                     PERIOD
                                                     [--period-origin PERIOD_ORIGIN]
                                                     [--threads THREADS]

optional arguments:
  -h, --help            show this help message and exit
  --set SET             annotations set
  --period PERIOD       time units to aggregate (optional); equivalent to
                        ``pandas.Grouper``'s freq argument.
  --period-origin PERIOD_ORIGIN
                        time origin of each time period; equivalent to
```

(continues on next page)

(continued from previous page)

```
                        ``pandas.Grouper``'s origin argument.
  --threads THREADS     amount of threads to run on
```

..note:

```
Average rates are expressed in seconds/hour regardless of the period.
```

CHAPTER

ELEVEN

AUDIO PROCESSORS

11.1 Overview

The package provides several tools for processing the recordings.

```
$ child-project process --help
usage: child-project process [-h] [--threads THREADS]
                             [--input-profile INPUT_PROFILE]
                             path name {basic,vetting,channel-mapping} ...

positional arguments:
  path                  path to the dataset
  name                  name of the export profile
  {basic,vetting,channel-mapping}
                        processor
    basic               basic audio conversion
    vetting             vetting
    channel-mapping     channel mapping

optional arguments:
  -h, --help            show this help message and exit
  --threads THREADS     amount of threads running conversions in parallel (0 =
                        uses all available cores)
  --input-profile INPUT_PROFILE
                        profile of input recordings (process raw recordings by
                        default)
```

11.2 Basic audio conversion

Converts all recordings in a dataset to a given encoding. Converted audios are stored into `recordings/converted/<profile-name>`.

```
$ child-project process /path/to/dataset test basic --help
usage: child-project process path name basic [-h] --format FORMAT --codec
                                             CODEC --sampling SAMPLING
                                             [--split SPLIT] [--skip-existing]
                                             [--recordings RECORDINGS [RECORDINGS ...]]

optional arguments:
```

(continues on next page)

(continued from previous page)

```
  -h, --help            show this help message and exit
  --format FORMAT       audio format (e.g. wav)
  --codec CODEC         audio codec (e.g. pcm_s16le)
  --sampling SAMPLING   sampling frequency (e.g. 16000)
  --split SPLIT         split duration (e.g. 15:00:00)
  --skip-existing
  --recordings RECORDINGS [RECORDINGS ...]
                        list of recordings to process, separated by commas;
                        only values of 'recording_filename' present in the
                        metadata are supported.
```

Example:

```
child-project process /path/to/dataset 16kHz basic --format=wav --sampling=16000 --
↪codec=pcm_s16le
```

Processing can be restricted to a white-list of recordings only using the `--recordings` option:

```
child-project process /path/to/dataset 16kHz basic --format=wav --sampling=16000 --
↪codec=pcm_s16le --recordings audio1.wav audio2.wav
```

Values provided to this option should be existing `recording_filename` values in `metadata/recordings.csv`.

The `--skip-existing` switch can be used to skip previously processed files.

11.2.1 Multi-core audio conversion with slurm on a cluster

If you have access to a cluster with slurm, you can use a command like the one below to batch-convert your recordings. Please note that you may need to change some details depending on your cluster (eg cpus per task). If needed, refer to the slurm user guide

```
sbatch --mem=64G --time=5:00:00 --cpus-per-task=4 --ntasks=1 -o namibia.txt child-
↪project process --threads 4 /path/to/dataset 16kHz basic --split=15:00:00 --format=wav␣
↪--sampling=16000 --codec=pcm_s16le
```

11.3 Vetting

The vetting pipeline mutes segments of the recordings provided by the user while preserving the duration of the audio files. This technique can be used to remove speech that might contain confidential information before releasing the audio.

The input needs to be a CSV dataframe with the following columns: `recording_filename`, `segment_onset`, `segment_onset`. The timestamps need to be expressed in milliseconds.

```
$ child-project process /path/to/dataset test vetting --help
usage: child-project process path name vetting [-h] --segments-path
                                               SEGMENTS_PATH
                                               [--recordings RECORDINGS [RECORDINGS ...]]

optional arguments:
  -h, --help            show this help message and exit
```

(continues on next page)

(continued from previous page)

```
  --segments-path SEGMENTS_PATH
                        path to the CSV dataframe containing the segments to
                        be vetted
  --recordings RECORDINGS [RECORDINGS ...]
                        list of recordings to process, separated by commas;
                        only values of 'recording_filename' present in the
                        metadata are supported.
```

11.4 Channel mapping

The channel mapping pipeline is meant to be used with multi-channel audio recordings, such as those produced by the BabyLogger. It allows to filter or to combine channels from the original recordings at your convenience.

```
$ child-project process /path/to/dataset test channel-mapping --help
usage: child-project process path name channel-mapping [-h] --channels
                                                       CHANNELS [CHANNELS ...]
                                                       [--recordings RECORDINGS␣
↪[RECORDINGS ...]]

optional arguments:
  -h, --help            show this help message and exit
  --channels CHANNELS [CHANNELS ...]
                        lists of weigths for each channel
  --recordings RECORDINGS [RECORDINGS ...]
                        list of recordings to process, separated by commas;
                        only values of 'recording_filename' present in the
                        metadata are supported.
```

In mathematical terms, assuming the input recordings have n channels with signals $s_j(t)$; If the output recordings should have m channels, the user defines a matrix of weights w_{ij} with m rows and n columns, such as the signal of each output channel $s'_i(t)$ is:

$$s'_i(t) = \sum_{j=1}^{n} w_{ij} s_j(t)$$

The weights matrix is defined through the `--channels` parameters.

The weights for each output channel are separated by blanks. For a given output channel, the weights of each input channels should be separated by commas.

For instance, if one would like to use the following weight matrix (which transforms 4-channel recordings into 2-channel audio):

$$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0.5 & 0.5 & 0 & 0 \end{pmatrix}$$

Then the correct values for the –channels parameters should be:

```
--channels 0,0,1,1 0.5,0.5,0,0
```

To make things clear, we provide a couple of examples below.

11.4.1 Muting all channels except for the first

Let's assume that the original recordings have 4 channels. The following command will extract the first channel from the recordings:

```
child-project process /path/to/dataset channel1 channel-mapping --channels 1,0,0,0
```

11.4.2 Invert a stereo signal

Let's assume that the original recordings are stereo signals, i.e. they have two channels. The command below will flip the two channels:

```
child-project process /path/to/dataset channel1 channel-mapping --channels 0,1 --
↪channels 1,0
```

CHAPTER

TWELVE

SAMPLERS

12.1 Overview

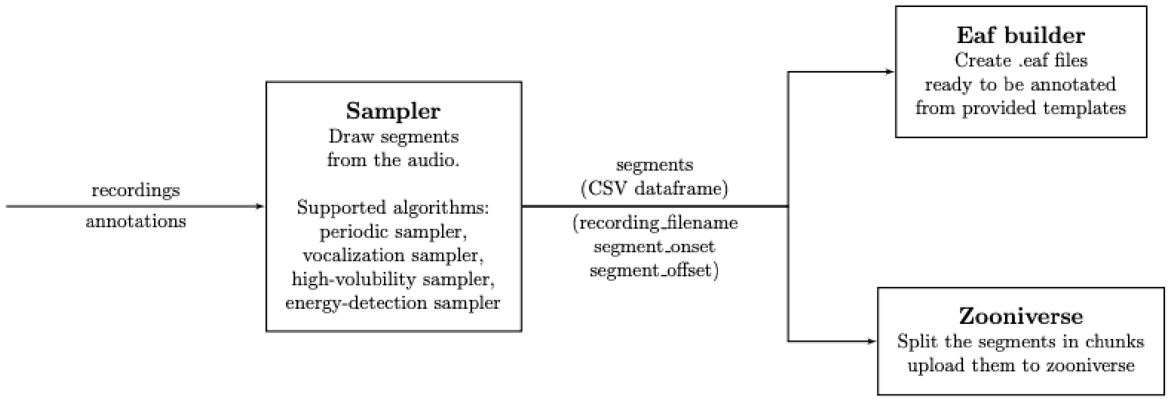

recordings
annotations
Sampler
Draw segments from the audio.
Supported algorithms: periodic sampler, vocalization sampler, high-volubility sampler, energy-detection sampler
segments (CSV dataframe)
(recording_filename segment_onset segment_offset)
Eaf builder
Create .eaf files ready to be annotated from provided templates
Zooniverse
Split the segments in chunks upload them to zooniverse

Fig. 1: Sampling audio segments to be annotated with ChildProject.

A sampler draws segments from the recordings, according to the algorithm and the parameters defined by the user. The sampler will produce two files into the *destination* folder :

- `segments_YYYYMMDD_HHMMSS.csv`, a CSV dataframe of all sampled segments, with three columns: `recording_filename`, `segment_onset` and `segment_offset`.
- `parameters_YYYYMMDD_HHMMSS.yml`, a Yaml file with all the parameters that were used to generate the samples.

If the folder *destination* does not exist, it is automatically created in the process.

Several samplers are implemented in our package, which are listed below.

The samples can then feed downstream pipelines such as the *Zooniverse* pipeline or the *ELAN builder*.

```
$ child-project sampler --help
usage: child-project sampler [-h] [--recordings RECORDINGS]
                             [--exclude EXCLUDE]
                             path destination
```

(continues on next page)

(continued from previous page)

```
                             {custom,periodic,random-vocalizations,energy-detection,high-
↪volubility,conversations}
                             ...

positional arguments:
  path                  path to the dataset
  destination           segments destination
  {custom,periodic,random-vocalizations,energy-detection,high-volubility,conversations}
                        sampler
    custom              custom sampling
    periodic            periodic sampling
    random-vocalizations
                        random sampling
    energy-detection    energy based activity detection
    high-volubility     high-volubility targeted sampling
    conversations       convesation sampler

optional arguments:
  -h, --help            show this help message and exit
  --recordings RECORDINGS
                        path to a CSV dataframe containing the list of
                        recordings to sample from (by default, all recordings
                        will be sampled). The CSV should have one column named
                        recording_filename.
  --exclude EXCLUDE     path to a CSV dataframe containing the list of
                        segments to exclude. The columns should be:
                        recording_filename, segment_onset and segment_offset.
```

All samplers have a few parameters in common:

- `--recordings`, which sets the white-list of recordings to sample from
- `--exclude`, which defines the portions of audio to exclude from the samples *after* sampling.

12.2 Periodic sampler

Draw segments from the recordings, periodically

```
$ child-project sampler /path/to/dataset /path/to/destination periodic --help
usage: child-project sampler path destination periodic [-h] --length LENGTH
                                                       --period PERIOD
                                                       [--offset OFFSET]
                                                       [--profile PROFILE]

optional arguments:
  -h, --help         show this help message and exit
  --length LENGTH    length of each segment, in milliseconds
  --period PERIOD    spacing between two consecutive segments, in milliseconds
  --offset OFFSET    offset of the first segment, in milliseconds
  --profile PROFILE  name of the profile of recordings to use to estimate
                     duration (uses raw recordings if empty)
```

12.3 Vocalization sampler

Draw segments from the recordings, targetting vocalizations from specific speaker-type(s).

```
$ child-project sampler /path/to/dataset /path/to/destination random-vocalizations --help
usage: child-project sampler path destination random-vocalizations
       [-h] [--annotation-set ANNOTATION_SET]
       [--target-speaker-type {CHI,OCH,FEM,MAL} [{CHI,OCH,FEM,MAL} ...]]
       --sample-size SAMPLE_SIZE [--threads THREADS]
       [--by {recording_filename,session_id,child_id}]

optional arguments:
  -h, --help            show this help message and exit
  --annotation-set ANNOTATION_SET
                        annotation set
  --target-speaker-type {CHI,OCH,FEM,MAL} [{CHI,OCH,FEM,MAL} ...]
                        speaker type to get chunks from
  --sample-size SAMPLE_SIZE
                        how many samples per unit (recording, session, or
                        child)
  --threads THREADS     amount of threads to run on
  --by {recording_filename,session_id,child_id}
                        units to sample from (default behavior is to sample by
                        recording)
```

12.4 Energy-based sampler

Draw segments from the recordings, targetting windows with energies above some threshold.

This algorithm proceeds by segmenting the recordings into windows; the energy of the signal is computed for each window (users have the option to apply a band-pass filter to calculate the energy in some frequency band).

Then, the algorithm samples as many windows as requested by the user from the windows that have energies above some threshold. The energy threshold is defined in term of energy quantile. By default, it is set to 0.8, i.e, only the windows with the 20% highest energies are sampled from.

The sampling is performed unit by unit, where the unit is set through the `--by` option and can be any either `recording_filename` (to sample an equal amount of windows from each recording), `session_id` (to equally from each observing day), or `child_id` (to sample equally from each child).

```
$ child-project sampler /path/to/dataset /path/to/destination energy-detection --help
usage: child-project sampler path destination energy-detection
       [-h] --windows-length WINDOWS_LENGTH --windows-spacing WINDOWS_SPACING
       --windows-count WINDOWS_COUNT [--windows-offset WINDOWS_OFFSET]
       [--threshold THRESHOLD] [--low-freq LOW_FREQ] [--high-freq HIGH_FREQ]
       [--threads THREADS] [--profile PROFILE]
       [--by {recording_filename,session_id,child_id}]

optional arguments:
  -h, --help            show this help message and exit
  --windows-length WINDOWS_LENGTH
                        length of each window (in milliseconds)
```

(continues on next page)

(continued from previous page)

```
  --windows-spacing WINDOWS_SPACING
                        spacing between the start of two consecutive windows
                        (in milliseconds)
  --windows-count WINDOWS_COUNT
                        how many windows to sample from each unit (recording,
                        session, or child)
  --windows-offset WINDOWS_OFFSET
                        start of the first window (in milliseconds)
  --threshold THRESHOLD
                        lowest energy quantile to sample from. default is 0.8
                        (i.e., sample from the 20% windows with the highest
                        energy).
  --low-freq LOW_FREQ   remove all frequencies below low-freq before
                        calculating each window's energy. (in Hz)
  --high-freq HIGH_FREQ
                        remove all frequencies above high-freq before
                        calculating each window's energy. (in Hz)
  --threads THREADS     amount of threads to run on
  --profile PROFILE     name of the profile of recordings to use (uses raw
                        recordings if empty)
  --by {recording_filename,session_id,child_id}
                        units to sample from (default behavior is to sample by
                        recording)
```

12.5 High-Volubility sampler

Return the top `windows_count` windows (of length `windows_length`) with the highest volubility from each recording, as calculated from the metric `metric`.

`metrics` can be any of three values: words, turns, and vocs.

- The **words** metric sums the amount of words within each window. For LENA annotations, it is equivalent to **awc**.
- The **turns** metric (aka ctc) sums conversational turns within each window. It relies on **lena_conv_turn_type** for LENA annotations. For other annotations, turns are estimated as adult/child speech switches in close temporal proximity.
- The **vocs** metric sums utterances (for LENA annotations) or vocalizations (for other annotations) within each window. If `metric="vocs"` and `speakers=['CHI']`, it is equivalent to the usual cvc metric (child vocalization counts).

```
$ child-project sampler /path/to/dataset /path/to/destination high-volubility --help
usage: child-project sampler path destination high-volubility
       [-h] --annotation-set ANNOTATION_SET --metric {turns,vocs,words}
       --windows-length WINDOWS_LENGTH --windows-count WINDOWS_COUNT
       [--speakers {CHI,FEM,MAL,OCH} [{CHI,FEM,MAL,OCH} ...]]
       [--threads THREADS] [--by {recording_filename,session_id,child_id}]

optional arguments:
  -h, --help            show this help message and exit
  --annotation-set ANNOTATION_SET
```

(continues on next page)

(continued from previous page)

```
                        annotation set
  --metric {turns,vocs,words}
                        which metric should be used to evaluate volubility
  --windows-length WINDOWS_LENGTH
                        window length (milliseconds)
  --windows-count WINDOWS_COUNT
                        how many windows to be sampled from each unit
                        (recording, session, or child)
  --speakers {CHI,FEM,MAL,OCH} [{CHI,FEM,MAL,OCH} ...]
                        speakers to include
  --threads THREADS     amount of threads to run on
  --by {recording_filename,session_id,child_id}
                        units to sample from (default behavior is to sample by
                        recording)
```

12.6 Conversation sampler

The conversation sampler returns the conversational blocks with the highest amount of turns (between adults and the key child). The first step is the detection of conversational blocks. Two consecutive vocalizations are considered part of the same conversational block if they are not separated by an interval longer than a certain duration, which by default is set to 1000 milliseconds.

Then, the amount of conversational turns (by default, between the key child and female/male adults) is calculated for each conversational block. The sampler returns, for each unit, the desired amount of conversations with the higher amount of turns.

This sampler, unlike the High-Volubility sampler, returns portions of audio with variable durations. Fixed duration can still be achieved by clipping or splitting each conversational block.

```
$ child-project sampler /path/to/dataset /path/to/destination conversations --help
usage: child-project sampler path destination conversations
       [-h] --annotation-set ANNOTATION_SET --count COUNT
       [--interval INTERVAL]
       [--speakers {CHI,FEM,MAL,OCH} [{CHI,FEM,MAL,OCH} ...]]
       [--threads THREADS] [--by {recording_filename,session_id,child_id}]

optional arguments:
  -h, --help            show this help message and exit
  --annotation-set ANNOTATION_SET
                        annotation set
  --count COUNT         how many conversations to be sampled from each unit
                        (recording, session, or child)
  --interval INTERVAL   maximum time-interval between two consecutive
                        vocalizations (in milliseconds) to consider them to be
                        part of the same conversational block. default is 1000
  --speakers {CHI,FEM,MAL,OCH} [{CHI,FEM,MAL,OCH} ...]
                        speakers to include
  --threads THREADS     amount of threads to run on
  --by {recording_filename,session_id,child_id}
                        units to sample from (default behavior is to sample by
                        recording)
```

Note: This sampler ignores LENA’s conversational turn types.

CHAPTER

THIRTEEN

ELAN BUILDER

13.1 Introduction

The ELAN .eaf builder exports eaf files ready to be annotated with the ELAN software, based on either the ACLEW DAS templates or custom templates.

The original code was written by Sarah Mac Ewan.

13.2 Usage

```
$ child-project eaf-builder --help
usage: child-project eaf-builder [-h] [--destination DESTINATION] --segments
                                 SEGMENTS --eaf-type
                                 {random,periodic,high-volubility,energy-detection}
                                 --template TEMPLATE
                                 [--context-onset CONTEXT_ONSET]
                                 [--context-offset CONTEXT_OFFSET]

generate .eaf templates based on intervals to code.

optional arguments:
  -h, --help            show this help message and exit
  --destination DESTINATION
                        eaf destination
  --segments SEGMENTS   path to the input segments dataframe
  --eaf-type {random,periodic,high-volubility,energy-detection}
                        eaf-type
  --template TEMPLATE   Which ACLEW templates (basic, native or non-native);
                        otherwise, the path to the etf et pfsx templates,
                        without the extension.
  --context-onset CONTEXT_ONSET
                        context onset and segment offset difference in
                        milliseconds, 0 for no introductory context
  --context-offset CONTEXT_OFFSET
                        context offset and segment offset difference in
                        milliseconds, 0 for no outro context
```

13.3 More resources

- Introduction: The ACLEW DAS template
- Tutorials: Using the ACLEW DAS template
- Bergelson, E., Warlaumont, A., Cristia, A., Soderstrom, M. & Vandam, M. (2017). A New Workflow for Semi-automatized Annotations: Tests with Long-Form Naturalistic Recordings of Children's Language Environments. In *Proceedings of Interspeech*. doi:DOI: 10.21437/Interspeech.2017-1418

CHAPTER

FOURTEEN

ZOONIVERSE

14.1 Introduction

We are providing here a pipeline to create, upload and analyse long format recordings using the Zooniverse citizen science platform.

We have an open project aimed at adding vocal maturity labels to segments LENA labeled as being key child in Zooniverse (https://www.zooniverse.org/projects/chiarasemenzin/maturity-of-baby-sounds).

If you would like your data labeled with this project, here is what you'd need to do.

1. Get in touch with us, so we know you are interested!
2. Have someone trustworthy & with some coding skills (henceforth, the RA) create a database using the formatting instructions (see *Datasets structure*).
3. Have the RA create an account on Zooniverse (top right of zooniverse.org) for them and yourself, & provide us with both handles. The RA should first update the team section to add you (have ready a picture and a blurb). The RA can also add your institution's logo if you'd like. Both of these are done in the lab section.
4. The RA will then follow the instructions in the present README to create subjects and push up your data – see below.
5. We also ask the RA to pitch in and help answer questions in the forum, at least one comment a day.
6. You can visit the stats section to look at how many annotations are being done.

You can also use this code and your own knowledge to set up a new project of your own. We provide a tutorial for creating a campaign of classification using Zooniverse and ChildProject.

14.2 Overview

```
$ child-project zooniverse --help
usage: child-project zooniverse [-h]
                                {extract-chunks,upload-chunks,retrieve-classifications}
                                ...

positional arguments:
  {extract-chunks,upload-chunks,retrieve-classifications}
                        action
    extract-chunks      extract chunks to <destination>, and exports the
                        metadata inside of this directory
    upload-chunks       upload chunks and updates chunk state
```

(continues on next page)

(continued from previous page)

```
    retrieve-classifications
                        retrieve classifications and save them as
                        <destination>

optional arguments:
  -h, --help            show this help message and exit
```

14.3 Chunk extraction

The `extract-chunks` pipeline creates wav and mp3 files for each chunk of audio to be classified on Zooniverse. It also saves a record of all these chunks into a CSV dataframe. This record can then be provided to the `upload-chunks` command, in order to upload the chunks to zooniverse.

..note:

```
``extract-chunks`` will require the list of segments to classify, which are provided as␣
↪a CSV dataframe with three columns:
``recording_filename``, ``segment_onset``, and ``segment_offset``. The path to this␣
↪dataframe has to be specified with the
``--segments`` parameter.

The list of segments can be generated with any of the samplers we provide (see␣
↪:ref:`samplers`), but custom lists
may also be provided.
```

Optionally, the segments provided to the pipeline can be split into chunks of the desired duration. By setting this duration to sufficently low values (e.g. 500 milliseconds), one can ensure that no meaningful information could be recovered while listening to the audio on Zooniverse. This is useful when the segments of audio provided to the pipeline may contain confidential information.

```
$ child-project zooniverse extract-chunks /path/to/dataset --help
usage: child-project zooniverse extract-chunks [-h] --keyword KEYWORD
                                               [--chunks-length CHUNKS_LENGTH]
                                               [--chunks-min-amount CHUNKS_MIN_AMOUNT]
                                               --segments SEGMENTS
                                               --destination DESTINATION
                                               [--profile PROFILE]
                                               [--threads THREADS]
                                               path

positional arguments:
  path                  path to the dataset

optional arguments:
  -h, --help            show this help message and exit
  --keyword KEYWORD     export keyword
  --chunks-length CHUNKS_LENGTH
                        chunk length (in milliseconds). if <= 0, the segments
                        will not be split into chunks (default value: 0)
  --chunks-min-amount CHUNKS_MIN_AMOUNT
                        minimum amount of chunks to extract from a segment
```

(continues on next page)

(continued from previous page)

```
                        (default value: 1)
  --segments SEGMENTS   path to the input segments dataframe
  --destination DESTINATION
                        destination
  --profile PROFILE     Recording profile to extract the audio clips from. If
                        not specified, raw recordings will be used
  --threads THREADS     how many threads to run on
```

If it does not exist, DESTINATION is created. Audio chunks are saved in wav and mp3 in `DESTINATION/chunks`. Metadata is stored in a CSV file into `DESTINATION/`.

The output dataframe will contain the following columns:

index	self-generated integer index
recording_filename	recording from which the chunk as extracted
onset	onset timestamp of the chunk within the recording
offset	offset timestamp of the chunk within the recording
segment_onset	onset timestamp of the segment from which the chunk was extracted
segment_offset	offset timestamp of the segment from which the chunk was extracted
wav	name of the wav file
mp3	name of the mp3 file
date_extracted	date at which the chunk was extracted
uploaded	boolean flag set to True if the chunk was uploaded to Zooniverse, False otherwise
project_id	zooniverse project ID
subject_set	name of the Zooniverse subject set
zooniverse_id	subject's Zooniverse ID
keyword	custom keyword provided by the user to label the chunks

14.4 Chunk upload

Once the chunks have been extracted, the next step is to upload them to Zooniverse. Note that due to quotas, it is recommended to upload only a few at time (e.g. 1000 per day).

You will need to provide the numerical id of your Zooniverse project, as well as your Zooniverse credentials.

`child-project zooniverse upload-chunks` uploads as many batches of audio chunks as specified to Zooniverse, and updates the chunks metadata accordingly, by setting the *zooniverse_id* field and *uploaded* to *True*.

```
$ child-project zooniverse upload-chunks /path/to/dataset --help
usage: child-project zooniverse upload-chunks [-h] --chunks CHUNKS
                                              --project-id PROJECT_ID
                                              --set-name SET_NAME
                                              [--amount AMOUNT]
                                              [--zooniverse-login ZOONIVERSE_LOGIN]
                                              [--zooniverse-pwd ZOONIVERSE_PWD]
                                              [--ignore-errors]


optional arguments:
  -h, --help            show this help message and exit
  --chunks CHUNKS       path to the chunk CSV dataframe
  --project-id PROJECT_ID
```

(continues on next page)

(continued from previous page)

```
                        zooniverse project id
  --set-name SET_NAME   subject set display name
  --amount AMOUNT       amount of chunks to upload
  --zooniverse-login ZOONIVERSE_LOGIN
                        zooniverse login. If not specified, the program
                        attempts to get it from the environment variable
                        ZOONIVERSE_LOGIN instead
  --zooniverse-pwd ZOONIVERSE_PWD
                        zooniverse password. If not specified, the program
                        attempts to get it from the environment variable
                        ZOONIVERSE_PWD instead
  --ignore-errors       keep uploading even when a subject fails to upload for
                        some reason
```

14.5 Classifications retrieval

```
$ child-project zooniverse retrieve-classifications /path/to/dataset --help
usage: child-project zooniverse retrieve-classifications [-h] --destination
                                                         DESTINATION
                                                         --project-id
                                                         PROJECT_ID
                                                         [--zooniverse-login ZOONIVERSE_
↪LOGIN]
                                                         [--zooniverse-pwd ZOONIVERSE_
↪PWD]
                                                         --chunks CHUNKS
                                                         [CHUNKS ...]

optional arguments:
  -h, --help            show this help message and exit
  --destination DESTINATION
                        output CSV dataframe destination
  --project-id PROJECT_ID
                        zooniverse project id
  --zooniverse-login ZOONIVERSE_LOGIN
                        zooniverse login. If not specified, the program
                        attempts to get it from the environment variable
                        ZOONIVERSE_LOGIN instead
  --zooniverse-pwd ZOONIVERSE_PWD
                        zooniverse password. If not specified, the program
                        attempts to get it from the environment variable
                        ZOONIVERSE_PWD instead
  --chunks CHUNKS [CHUNKS ...]
                        list of chunks
```

Retrieve classifications and save them as `DESTINATION`. The optional `--chunks` parameter can be used to match the classifications with the chunks metadata. Only the classifications that match the metadata will be saved.

Warning: Retrieving chunks may take a long time for large projects.

CHAPTER

FIFTEEN

CHEATSHEET

15.1 DataLad cheatsheet

15.1.1 Installing a dataset

```
datalad install [-h] [-s SOURCE] [-d DATASET] [-g] [-D DESCRIPTION] [-r] [-R LEVELS] [--
↪reckless [auto|ephemeral|shared-...]] [-J NJOBS] [PATH [PATH ...]]
```

Example:

```
datalad install -r git@github.com:LAAC-LSCP/datasets.git
```

Note: some datasets might have additional installation instructions!

More: datalad install

15.1.2 Getting data

```
datalad get [-h] [-s LABEL] [-d PATH] [-r] [-R LEVELS] [-n] [-D DESCRIPTION] [--reckless␣
↪[auto|ephemeral|shared-...]] [-J NJOBS] [PATH [PATH ...]]
```

Example:

```
datalad get annotations/vtc
```

More: datalad get

15.1.3 Getting updates

```
datalad update --merge
```

More: datalad update

15.1.4 Saving changes

```
datalad save [-h] [-m MESSAGE] [-d DATASET] [-t ID] [-r] [-R LEVELS] [-u] [-F MESSAGE_
↪FILE] [--to-git] [-J NJOBS] [PATH [PATH ...]]
```

Example:

```
datalad save metadata/children.csv -m "correcting children metadata"
```

`datalad save` is analoguous to doing `git add`+`git commit`. It will decide automatically whether to store the files in git or in the annex.

Note: datalad save records the changes locally. They still have to be pulished - just like with git commit !

More: datalad save

15.1.5 Publishing changes

```
datalad push [-h] [-d DATASET] [--to SIBLING] [--since SINCE] [--data
↪{anything|nothing|auto|auto-if-wanted}] [-f {all|gitpush|checkdatapresent}] [-r] [-R␣
↪LEVELS] [-J NJOBS] [PATH [PATH ...]]
```

Example:

```
datalad push
```

More: datalad push

15.2 ChildProject cheatsheet

CHAPTER

SIXTEEN

ANNOTATIONS

Annotations can be managed through both the command-line interface and the python API. This section documents the principle features of the API for the management of annotations.

Note: In order to reproduce the following examples, you will need to install the public VanDam corpus and its annotations using datalad:

```
datalad install git@gin.g-node.org:/LAAC-LSCP/vandam-data.git
datalad get vandam-data/annotations
```

16.1 Reading annotations

Annotations are managed with *ChildProject.annotations.AnnotationManager* class. The first step is create an instance of it based on the target project.

The *read()* method reads the index of annotations from `metadata/annotations.csv` and stores into its `annotations` attribute:

```
>>> from ChildProject.projects import ChildProject
>>> from ChildProject.annotations import AnnotationManager
>>> project = ChildProject('vandam-data')
>>> am = AnnotationManager(project)
>>> am.read()
([], ["vandam-data/metadata/annotations.csv: 'chat' is not a permitted value for column
↪'format' on line 4, should be any of [csv,vtc_rttm,vcm_rttm,alice,its,TextGrid,eaf,cha,
↪NA]"])
>>> am.annotations
           set recording_filename  time_seek  range_onset  range_offset             raw_
↪filename    format  filter               annotation_filename          imported_at ␣
↪error package_version
2          its    BN32_010007.mp3          0            0      50464512           BN32_
↪010007.its       its      NaN              BN32_010007_0_0.csv  2021-03-06 22:55:06  ␣
↪ NaN           0.0.1
3          vtc    BN32_010007.mp3          0            0      50464512          BN32_
↪010007.rttm  vtc_rttm     NaN              BN32_010007_0_0.csv  2021-05-12 19:28:25 ␣
↪  NaN           0.0.1
4          cha    BN32_010007.mp3          0            0      50464512           BN32_
↪010007.cha       chat     NaN              BN32_010007_0_0.csv  2021-05-12 19:39:05  ␣
↪ NaN           0.0.1
```

(continues on next page)

(continued from previous page)

```
5           eaf    BN32_010007.mp3          0      4138389       4199976          BN32_
↪010007.eaf       eaf     NaN    BN32_010007_4138389_4199976.csv  2021-07-14 17:39:50  ␣
↪ NaN          0.0.1
6           eaf    BN32_010007.mp3          0      4438842       4499995          BN32_
↪010007.eaf       eaf     NaN    BN32_010007_4438842_4499995.csv  2021-07-14 17:39:50  ␣
↪ NaN          0.0.1
7           eaf    BN32_010007.mp3          0     13199449      13256801          BN32_
↪010007.eaf       eaf     NaN  BN32_010007_13199449_13256801.csv  2021-07-14 17:39:50  ␣
↪ NaN          0.0.1
8           eaf    BN32_010007.mp3          0     37496002      37558424          BN32_
↪010007.eaf       eaf     NaN  BN32_010007_37496002_37558424.csv  2021-07-14 17:39:50  ␣
↪ NaN          0.0.1
9           eaf    BN32_010007.mp3          0     37616206      37679577          BN32_
↪010007.eaf       eaf     NaN  BN32_010007_37616206_37679577.csv  2021-07-14 17:39:50  ␣
↪ NaN          0.0.1
10  cha/aligned    BN32_010007.mp3          0            0      47725356  BN32_010007-
↪aligned.csv       csv      NaN         BN32_010007_0_47725356.csv  2021-07-15 16:15:48 ␣
↪  NaN          0.0.1
```

As seen in this example, `annotations` only contains the index of annotations, not their contents. To retrieve the actual annotations, use *get_segments()*:

```
>>> selection = am.annotations[am.annotations['set'].isin(['cha', 'vtc'])]
>>> segments = am.get_segments(selection)
>>> segments

    segment_onset  segment_offset speaker_type      raw_filename  set  annotation_
↪filename participant  ... range_onset range_offset    format filter          imported_
↪at error package_version
0               9992           10839       SPEECH  BN32_010007.rttm  vtc  BN32_010007_0_
↪0.csv         NaN  ...           0     50464512  vtc_rttm    NaN  2021-05-12 19:28:25 ␣
↪ NaN          0.0.1
1              10004           10814          CHI  BN32_010007.rttm  vtc  BN32_010007_0_
↪0.csv         NaN  ...           0     50464512  vtc_rttm    NaN  2021-05-12 19:28:25 ␣
↪ NaN          0.0.1
2              11298           11953       SPEECH  BN32_010007.rttm  vtc  BN32_010007_0_
↪0.csv         NaN  ...           0     50464512  vtc_rttm    NaN  2021-05-12 19:28:25 ␣
↪ NaN          0.0.1
3              11345           11828          CHI  BN32_010007.rttm  vtc  BN32_010007_0_
↪0.csv         NaN  ...           0     50464512  vtc_rttm    NaN  2021-05-12 19:28:25 ␣
↪ NaN          0.0.1
4              12113           12749          FEM  BN32_010007.rttm  vtc  BN32_010007_0_
↪0.csv         NaN  ...           0     50464512  vtc_rttm    NaN  2021-05-12 19:28:25 ␣
↪ NaN          0.0.1
             ...             ...          ...               ...  ...              ...␣
↪         ...  ...         ...          ...       ...     ...                  ...   ... ␣
↪           ...
31875       49705416        49952432          CHI   BN32_010007.cha  cha  BN32_010007_0_
↪0.csv         CHI  ...           0     50464512      chat    NaN  2021-05-12 19:39:05 ␣
↪ NaN          0.0.1
31876       49952432        50057166          CHI   BN32_010007.cha  cha  BN32_010007_0_
↪0.csv         CHI  ...           0     50464512      chat    NaN  2021-05-12 19:39:05 ␣
↪ NaN          0.0.1
```

(continues on next page)

(continued from previous page)

```
31877       50057166        50173260           CHI   BN32_010007.cha  cha  BN32_010007_0_
↪0.csv         CHI  ...            0      50464512      chat    NaN  2021-05-12 19:39:05 ␣
↪ NaN           0.0.1
31878       50173260        50330885           CHI   BN32_010007.cha  cha  BN32_010007_0_
↪0.csv         CHI  ...            0      50464512      chat    NaN  2021-05-12 19:39:05 ␣
↪ NaN           0.0.1
31879       50330885        50397134           CHI   BN32_010007.cha  cha  BN32_010007_0_
↪0.csv         CHI  ...            0      50464512      chat    NaN  2021-05-12 19:39:05 ␣
↪ NaN           0.0.1

[31880 rows x 22 columns]
```

Warning: Trying to load all annotations at once may quickly lead to out-of-memory errors, especially with automated annotators convering thousands of hours of audio. Memory issues can be alleviated by processing the data sequentially, e.g. by treating one recording after another.

16.2 Importing annotations

Although importing annotations can be done using the command-line tool, sometimes it is more efficient to do it directly with the python API; it can even become necessary when custom converters (the functions that transform any kind of annotations into the CSV format used by the package) need to be used.

Two examples are given below (one using built-in converters, one using a custom converter). In order to reproduce them, please make a copy of the original annotations:

```
mkdir vandam-data/annotations/playground
cp -r vandam-data/annotations/its vandam-data/annotations/playground
```

16.2.1 Built-in formats

The following code imports only the annotations from the LENA that correspond to the second hour of the audio. The package natively supports LENA's .its annotations.

Annotations are imported using *import_annotations()*. This first input argument of this method must be a pandas dataframe of all the annotations that need to be imported. This dataframe should be structured according to the format defined at format-input-annotations.

```
>>> import pandas as pd
>>> input = pd.DataFrame([{
...     'set': 'playground/its',
...     'recording_filename': 'BN32_010007.mp3',
...     'time_seek': 0,
...     'range_onset': 3600*1000,
...     'range_offset': 7200*1000,
...     'raw_filename': 'BN32_010007.its',
...     'format': 'its'
... }])
>>> am.import_annotations(input, threads = 1)
```

(continues on next page)

(continued from previous page)

```
            set recording_filename  time_seek  range_onset  range_offset     raw_
↪filename format        annotation_filename          imported_at package_version
0  playground/its    BN32_010007.mp3          0      3600000       7200000  BN32_010007.
↪its    its  BN32_010007_0_3600000.csv  2021-05-12 20:37:43           0.0.1
```

After reloading the index of annotations, the newly inserted entry now appears:

```
>>> am.read()
([], [])
>>> am.annotations
            set recording_filename  time_seek  range_onset  range_offset      raw_
↪filename    format  filter        annotation_filename          imported_at  error␣
↪package_version
2             its    BN32_010007.mp3          0            0      50464512   BN32_010007.
↪its       its     NaN        BN32_010007_0_0.csv  2021-03-06 22:55:06    NaN          ␣
↪0.0.1
3             vtc    BN32_010007.mp3          0            0      50464512  BN32_010007.
↪rttm  vtc_rttm      NaN         BN32_010007_0_0.csv  2021-05-12 19:28:25    NaN          ␣
↪ 0.0.1
4             cha    BN32_010007.mp3          0            0      50464512   BN32_010007.
↪cha      chat      NaN        BN32_010007_0_0.csv  2021-05-12 19:39:05    NaN          ␣
↪0.0.1
5  playground/its    BN32_010007.mp3          0      3600000       7200000   BN32_010007.
↪its       its     NaN  BN32_010007_0_3600000.csv  2021-05-12 20:37:43    NaN          ␣
↪0.0.1
```

Built-in converters include: LENA's its, VTC's and VCM's rttms, ALICE, ACLEW DAS eaf files. To import annotations under other formats, custom converters are needed.

16.2.2 Custom converter

A converter is a function that takes a filename for only input, and return a dataframe complying with the specifications defined in *Annotations index*.

The output dataframe _must_ contain at least a `segment_onset` and a `segment_offset` columns expressing the onset and offset of each segment in milliseconds as integers.

You are free to add as many extra columns as you want. It is however preferable to follow the standards listed in *Annotations index* when possible.

In our case, we'll write a very simple converter to extract only the segments onset and offset from rttm files:

```
>>> def convert_rttm(filename: str):
...     df = pd.read_csv(filename, sep = " ", names = ['type', 'file', 'chnl', 'tbeg',
↪'tdur', 'ortho', 'stype', 'name', 'conf', 'unk'])
...     df['segment_onset'] = df['tbeg'].mul(1000).round().astype(int)
...     df['segment_offset'] = (df['tbeg']+df['tdur']).mul(1000).round().astype(int)
...     df.drop(['type', 'file', 'chnl', 'tbeg', 'tdur', 'ortho', 'stype', 'name', 'conf
↪', 'unk'], axis = 1, inplace = True)
...     return df
...
>>>
```

The converter can now be used with `import_annotations()`:

```
>>> input = pd.DataFrame([{
...     'set': 'playground/vtc',
...     'recording_filename': 'BN32_010007.mp3',
...     'time_seek': 0,
...     'range_onset': 3600*1000,
...     'range_offset': 7200*1000,
...     'raw_filename': 'BN32_010007.rttm',
...     'format': 'custom_rttm'
... }])
>>> am.import_annotations(input, threads = 1, import_function = convert_rttm)
              set recording_filename  time_seek  range_onset  range_offset      raw_
↪filename       format        annotation_filename          imported_at package_version
0  playground/vtc    BN32_010007.mp3          0      3600000       7200000  BN32_010007.
↪rttm  custom_rttm  BN32_010007_0_3600000.csv  2021-05-13 17:25:20           0.0.1
```

The contents of the output CSV file can be checked:

```
>>> rttm = pd.read_csv('vandam-data/annotations/playground/vtc/converted/BN32_010007_0_
↪3600000.csv')
>>> rttm
    segment_onset  segment_offset      raw_filename
0           3600401         3601370  BN32_010007.rttm
1           3600403         3601464  BN32_010007.rttm
2           3601503         3602843  BN32_010007.rttm
3           3601527         3602833  BN32_010007.rttm
4           3604075         3605570  BN32_010007.rttm
...             ...             ...               ...
1622        7010992         7011243  BN32_010007.rttm
1623        7011495         7011615  BN32_010007.rttm
1624        7033826         7034142  BN32_010007.rttm
1625        7036539         7037008  BN32_010007.rttm
1626        7036556         7036996  BN32_010007.rttm

[1627 rows x 3 columns]
```

Warning: Do not import the same file twice, as duplicates in the index might cause issues. Make sure to remove an annotation from an index beforehand if you need to import it again. This can be done with `remove_set()` to remove a set of annotations from the index while preserving raw annotations.

16.3 Validating annotations

The contents of annotations can be searched for errors using the `validate()` function.

..code-block:: python

```
>>> errors, warnings = am.validate()
validating BN32_010007_0_0.csv...
validating BN32_010007_0_0.csv...
validating BN32_010007_0_0.csv...
validating BN32_010007_0_3600000.csv...
```

(continues on next page)

(continued from previous page)

```
validating BN32_010007_0_3600000.csv...
>>> errors
[]
>>> warnings
[]
```

`errors` and `warnings` are empty, indicating that there are no errors.

To gather the errors and warnings raised why validating the index of annotations, use *read()*:

..code-block:: python

```
>>> errors, warnings = am.read()
>>> errors
[]
>>> warnings
[]
```

16.4 Time-of-the-day

For a number of purposes, it may be convenient to retrieve the timestamp of each vocalization, or to filter out annotations outside some specific time-range.

Both tasks can be performed through the python API of the package.

16.4.1 Annotations within a specific time-range

A given set of annotations may be clipped within a given time-range using *get_within_time_range()*. For instance, annotations of audio between 9am and 12am may be retrieved from the following code:

```
>>> morning = am.get_within_time_range(am.annotations, '09:00', '12:00')
>>> morning
        set recording_filename  time_seek  range_onset  range_offset            raw_
↪filename  ...           imported_at  error package_version          start_time  range_
↪onset_time range_offset_time
0          its    BN32_010007.mp3          0    7320000.0    18120000.0            BN32_
↪010007.its  ...  2021-03-06 22:55:06    NaN           0.0.1 1900-01-01 06:58:00       ␣
↪     09:00             12:00
1          vtc    BN32_010007.mp3          0    7320000.0    18120000.0           BN32_
↪010007.rttm  ...  2021-05-12 19:28:25    NaN           0.0.1 1900-01-01 06:58:00      ␣
↪      09:00             12:00
2          cha    BN32_010007.mp3          0    7320000.0    18120000.0            BN32_
↪010007.cha  ...  2021-05-12 19:39:05    NaN           0.0.1 1900-01-01 06:58:00       ␣
↪     09:00             12:00
3          eaf    BN32_010007.mp3          0   13199449.0    13256801.0            BN32_
↪010007.eaf  ...  2021-07-14 17:39:50    NaN           0.0.1 1900-01-01 06:58:00       ␣
↪     10:37      10:38:56.352
4  cha/aligned    BN32_010007.mp3          0    7320000.0    18120000.0  BN32_010007-
↪aligned.csv  ...  2021-07-15 16:15:48    NaN           0.0.1 1900-01-01 06:58:00      ␣
↪      09:00             12:00
```

(continues on next page)

(continued from previous page)

```
[5 rows x 15 columns]
```

The onset and offset timestamps for each segments can be calculated with `get_segments_timestamps()`:

```
>>> segments = am.get_segments(morning)
>>> segments = am.get_segments_timestamps(segments)
>>> segments[['speaker_type', 'onset_time', 'offset_time']]
    speaker_type                onset_time               offset_time
0              CHI 2010-07-24 09:00:00.000 2010-07-24 09:20:39.793
1              CHI 2010-07-24 09:20:39.793 2010-07-24 09:21:43.496
2              CHI 2010-07-24 09:21:43.496 2010-07-24 09:23:45.168
3              CHI 2010-07-24 09:23:45.168 2010-07-24 09:24:12.371
4              CHI 2010-07-24 09:24:12.371 2010-07-24 09:27:27.019
...            ...                     ...                     ...
11801          CHI 2010-07-24 11:56:50.584 2010-07-24 11:56:51.011
11802          FEM 2010-07-24 11:57:15.749 2010-07-24 11:57:15.992
11803          MAL 2010-07-24 11:57:24.637 2010-07-24 11:57:25.010
11804       SPEECH 2010-07-24 11:57:35.237 2010-07-24 11:57:35.666
11805          CHI 2010-07-24 11:57:35.314 2010-07-24 11:57:35.511

[11806 rows x 3 columns]
```

16.5 Module reference

CHAPTER

SEVENTEEN

RELIABILITY METRICS

ChildProject implements several metrics for evaluating annotations and their reliability. This section demonstrates how to use the python API for these purposes.

Note: In order to reproduce the following examples, you will need to install the public VanDam corpus and its annotations using datalad:

```
datalad install git@gin.g-node.org:/LAAC-LSCP/vandam-data.git
datalad get vandam-data/annotations
```

17.1 Comparing two annotators

The performance of automated annotations is usually assessed by comparing them to a ground truth provided by experts. The ChildProject package provides several tools for such comparisons.

17.1.1 Confusion matrix

Confusion matrices are widely used to assess the performance of classification algorithms; they give an accurate visual description of the behavior of a classifier, preserving most relevant information while still being easy to interpret.

We show how to compute confusion matrices with the ChildProject package, using data from the VanDam public corpus. In this example, we will compare annotations from the LENA and the Voice Type Classifier.

The first step is to get all annotations common to the LENA and the VTC. This can be done with the *intersection()* static method of *AnnotationManager*:

```
>>> from ChildProject.projects import ChildProject
>>> from ChildProject.annotations import AnnotationManager
>>> from ChildProject.metrics import segments_to_grid, conf_matrix
>>> speakers = ['CHI', 'OCH', 'FEM', 'MAL']
>>> project = ChildProject('vandam-data')
>>> am = AnnotationManager(project)
>>> am.read()
([], ["vandam-data/metadata/annotations.csv: 'chat' is not a permitted value for column
↪'format' on line 4, should be any of [TextGrid,eaf,vtc_rttm,vcm_rttm,alice,its]",
↪"vandam-data/metadata/annotations.csv: 'custom_rttm' is not a permitted value for␣
↪column 'format' on line 6, should be any of [TextGrid,eaf,vtc_rttm,vcm_rttm,alice,its]
↪"])
```

(continues on next page)

(continued from previous page)

```
>>> intersection = AnnotationManager.intersection(am.annotations, ['vtc', 'its'])
>>> intersection
set recording_filename  time_seek  range_onset  range_offset      raw_filename    format␣
↪ filter  annotation_filename          imported_at  error package_version
2  its    BN32_010007.mp3          0            0      50464512   BN32_010007.its      ␣
↪its    NaN  BN32_010007_0_0.csv  2021-03-06 22:55:06    NaN          0.0.1
3  vtc    BN32_010007.mp3          0            0      50464512  BN32_010007.rttm  vtc_
↪rttm     NaN  BN32_010007_0_0.csv  2021-05-12 19:28:25    NaN          0.0.1
```

The next step is to retrieve the contents of the annotations that correspond to the intersection of the two sets. This is done with *get_collapsed_segments()*. This method from *AnnotationManager* does the following:

1. Read the contents of all annotations provided into one pandas dataframe.
2. Align them annotator by annotator, allowing cross-comparisons or combination
3. In case these annotations come from non-consecutive portions of audio, or from distinct audio files, they are aligned end-to-end into one virtual timeline.

In the case of the VanDam corpus, there is only one audio file, and it has been entirely annotated by all annotators. But the following will work even for sparse annotations covering several recordings.

```
>>> segments = am.get_collapsed_segments(intersection)
>>> segments = segments[segments['speaker_type'].isin(speakers)]
>>> segments
    segment_onset  segment_offset  speaker_id  ling_type speaker_type  vcm_type  lex_
↪type  ...          imported_at  error  package_version  abs_range_onset  abs_range_
↪offset    duration  position
1            9730.0         10540.0         NaN        NaN          OCH       NaN       ␣
↪NaN  ...  2021-03-06 22:55:06    NaN            0.0.1                0              ␣
↪50464512  50464512.0       0.0
15          35820.0         36930.0         NaN        NaN          OCH       NaN       ␣
↪NaN  ...  2021-03-06 22:55:06    NaN            0.0.1                0              ␣
↪50464512  50464512.0       0.0
21          67020.0         67620.0         NaN        NaN          OCH       NaN       ␣
↪NaN  ...  2021-03-06 22:55:06    NaN            0.0.1                0              ␣
↪50464512  50464512.0       0.0
25          71640.0         72240.0         NaN        NaN          FEM       NaN       ␣
↪NaN  ...  2021-03-06 22:55:06    NaN            0.0.1                0              ␣
↪50464512  50464512.0       0.0
29          87370.0         88170.0         NaN        NaN          OCH       NaN       ␣
↪NaN  ...  2021-03-06 22:55:06    NaN            0.0.1                0              ␣
↪50464512  50464512.0       0.0
...             ...             ...         ...        ...          ...       ...       ␣
↪...  ...                  ...    ...              ...              ...            ..
↪.          ...       ...
22342     50122992.0      50123518.0         NaN        NaN          FEM       NaN       ␣
↪NaN  ...  2021-05-12 19:28:25    NaN            0.0.1                0              ␣
↪50464512  50464512.0       0.0
22344     50152103.0      50153510.0         NaN        NaN          FEM       NaN       ␣
↪NaN  ...  2021-05-12 19:28:25    NaN            0.0.1                0              ␣
↪50464512  50464512.0       0.0
22348     50233080.0      50234492.0         NaN        NaN          FEM       NaN       ␣
↪NaN  ...  2021-05-12 19:28:25    NaN            0.0.1                0              ␣
↪50464512  50464512.0       0.0
```

(continues on next page)

(continued from previous page)

```
22350     50325867.0      50325989.0          NaN         NaN          CHI        NaN      ␣
↪NaN  ...  2021-05-12 19:28:25    NaN          0.0.1                0            ␣
↪50464512  50464512.0       0.0
22352     50356380.0      50357011.0          NaN         NaN          FEM        NaN      ␣
↪NaN  ...  2021-05-12 19:28:25    NaN          0.0.1                0            ␣
↪50464512  50464512.0       0.0

[20887 rows x 44 columns]
```

For an efficient computation of the confusion matrix, the timeline is then split into chunks of a given length (in our case, we will set the time steps to 100 milliseconds). This is done with *ChildProject.metrics.segments_to_grid()*, which transforms a dataframe of segments into a matrix of the indicator functions of each classification category at each time unit.

```
>>> vtc = segments_to_grid(segments[segments['set'] == 'vtc'], 0, segments['segment_
↪offset'].max(), 100, 'speaker_type', speakers)
/Users/acristia/anaconda3/lib/python3.7/site-packages/pandas/core/indexing.py:1676:␣
↪SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
self._setitem_single_column(ilocs[0], value, pi)
/Users/acristia/anaconda3/lib/python3.7/site-packages/pandas/core/indexing.py:1597:␣
↪SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
self.obj[key] = value
>>> its = segments_to_grid(segments[segments['set'] == 'its'], 0, segments['segment_
↪offset'].max(), 100, 'speaker_type', speakers)
>>> vtc.shape
(503571, 5)
>>> vtc
array([[0, 0, 0, 0, 1],
    [0, 0, 0, 0, 1],
    [0, 0, 0, 0, 1],
    ...,
    [0, 0, 1, 0, 0],
    [0, 0, 1, 0, 0],
    [0, 0, 0, 0, 1]])
```

Note that this matrix has 5 columns, even though there are only 4 categories (CHI, OCH, FEM and MAL). This is because *segments_to_grid()* appends the matrix with a 'none' column, which is set to 1 when all classes are inactive. It can be turned off by setting *none = False*. It is also possible to append an 'overlap' column by setting *overlap=True*; this column is set to 1 when at least 2 classes are active.

We can now compute the confusion matrix:

```
>>> confusion_counts = conf_matrix(vtc, its)
>>> confusion_counts
array([[ 20503,   7285,   4296,   1191,  21062],
[  1435,   3354,    704,    136,   4105],
[  2700,   1414,  18442,   4649,  19080],
[   323,    229,   4600,  17654,  12415],
[  3053,   2158,   3674,   2464, 365000]])
```

This means that 20503 of the 100 ms chunks were labelled as containing CHI speech by both the VTC and the LENA; 7285 chunks have been labelled as containing CHI speech by the VTC while being labelled as OCH by the LENA.

It is sometimes more useful to normalize confusion matrices:

```
>>> import numpy as np
>>> normalized = confusion_counts/(np.sum(vtc, axis = 0)[:,None])
>>> rel
array([[0.37733036, 0.13407071, 0.07906215, 0.02191877, 0.38761801],
[0.14742141, 0.34456544, 0.07232381, 0.01397165, 0.42171769],
[0.05833423, 0.03054985, 0.39844442, 0.10044291, 0.41222858],
[0.00917067, 0.0065018 , 0.1306039 , 0.50123506, 0.35248857],
[0.00811215, 0.00573404, 0.00976222, 0.00654711, 0.96984448]])
```

The top-left cell now reads as: 37,8% of the 100 ms chunks labelled as CHI by the VTC are also labelled as CHI by the LENA.

17.1.2 Using pyannote.metrics

Confusion matrices are still dimensional data (with $n \times n$ components for n labels), which renders performance comparisons of several annotators difficult: it is hard to tell which one of two classifiers is the closest to the ground truth using confusion matrices.

As a result, in Machine Learning, many scalar measures are used in order to assess the overall performance of a classifier. These include recall, precision, accuracy, etc.

The pyannote-metrics package implements many of the metrics that are typically used in speech processing. ChildProject interfaces well with pyannote-metrics. Below, we show how to use both packages to compute recall and precision.

The first step is to convert the dataframe of segments into one `pyannote.core.Annotation()` object per annotator:

```
>>> from ChildProject.metrics import segments_to_annotation
>>> ref = segments_to_annotation(segments[segments['set'] == 'vtc'], 'speaker_type')
>>> hyp = segments_to_annotation(segments[segments['set'] == 'its'], 'speaker_type')
```

Now, any pyannote metric can be instantianted and used with these annotations:

```
>>> from pyannote.metrics.detection import DetectionPrecisionRecallFMeasure
>>> metric = DetectionPrecisionRecallFMeasure()
>>> detail = metric.compute_components(ref, hyp)
>>> precision, recall, f = metric.compute_metrics(detail)
>>> print(f'{precision:.2f}/{recall:.2f}/{f:.2f}')
0.87/0.60/0.71
```

17.2 Reliability evaluations

17.3 Module reference

`ChildProject.metrics.`**`conf_matrix`**`(`*rows_grid*, *columns_grid*`)`

compute the confusion matrix (as counts) from grids of active classes.

See `ChildProject.metrics.segments_to_grid()` for a description of grids.

Parameters

- **`rows_grid`** (*`numpy.array`*) – the grid corresponding to the rows of the confusion matrix.
- **`columns_grid`** (*`numpy.array`*) – the grid corresponding to the columns of the confusion matrix.
- **`categories`** (*`list of strings`*) – the labels corresponding to each class

Returns a square numpy array of counts

Return type numpy.array

`ChildProject.metrics.`**`gamma`**`(`*segments: pandas.core.frame.DataFrame*, *column: str*, *alpha: float = 1*, *beta: float = 1*, *precision_level: float = 0.05*`)` → float

Compute Mathet et al. gamma agreement on *segments*.

The gamma measure evaluates the reliability of both the segmentation and the categorization simultaneously; a extensive description of the method and its parameters can be found in Mathet et al., 2015 (doi:10.1162/COLI_a_00227)

This function uses the pyagreement-agreement package by Titeux et al.

Parameters

- **`segments`** (*`pd.DataFrame`*) – input segments dataframe (see *Annotations format* for the dataframe format)
- **`column`** (*`str`*) – name of the categorical column of the segments to consider, e.g. 'speaker_type'
- **`alpha`** (*`float, optional`*) – gamma agreement time alignment weight, defaults to 1
- **`beta`** (*`float, optional`*) – gamma agreement categorical weight, defaults to 1
- **`precision_level`** (*`float, optional`*) – level of precision (see pygamma-agreement's documentation), defaults to 0.05

Returns gamma agreement

Return type float

`ChildProject.metrics.`**`grid_to_vector`**`(`*grid*, *categories*`)`

Transform a grid of active classes into a vector of labels. In case several classes are active at time i, the label is set to 'overlap'.

See `ChildProject.metrics.segments_to_grid()` for a description of grids.

Parameters

- **`grid`** (*`numpy.array`*) – a NumPy array of shape `(n, len(categories))`
- **`categories`** (*`list`*) – the list of categories

Returns the vector of labels of length `n` (e.g. `np.array([none FEM FEM FEM overlap overlap CHI])`)

Return type numpy.array

`ChildProject.metrics.`**`segments_to_annotation`**(*segments: pandas.core.frame.DataFrame*, *column: str*)

Transform a dataframe of annotation segments into a pyannote.core.Annotation object

Parameters

- **`segments`** (*`pd.DataFrame`*) – a dataframe of input segments. It should at least have the following columns: `segment_onset`, `segment_offset` and `column`.
- **`column`** (*`str`*) – the name of the column in `segments` that should be used for the values of the annotations (e.g. speaker_type).

Returns the pyannote.core.Annotation object.

Return type pyannote.core.Annotation

`ChildProject.metrics.`**`segments_to_grid`**(*segments: pandas.core.frame.DataFrame*, *range_onset: int*, *range_offset: int*, *timescale: int*, *column: str*, *categories: list*, *none=True*, *overlap=False*) → float

Transform a dataframe of annotation segments into a 2d matrix representing the indicator function of each of the `categories` across time.

Each row of the matrix corresponds to a unit of time of length `timescale` (in milliseconds), ranging from `range_onset` to `range_offset`; each column corresponds to one of the `categories` provided, plus two special columns (overlap and none).

The value of the cell `ij` of the output matrix is set to 1 if the class `j` is active at time `i`, 0 otherwise.

If *overlap* is True, an additional column is appended to the grid, which set to 1 if more than two classes are active at time `i`.

If *none* is set to True, an additional column is appended to the grid, which is set to one if none of the classes are active at time `i`.

The shape of the output matrix is therefore `((range_offset-range_onset)/timescale, len(categories) + n)`, where `n = 2` if both *overlap* and *none* are True, 1 if one of them is True, and 0 otherwise.

The fraction of time a class `j` is active can therefore be calculated as `np.mean(grid, axis = 0)[j]`

Parameters

- **`segments`** (*`pd.DataFrame`*) – a dataframe of input segments. It should at least have the following columns: `segment_onset`, `segment_offset` and `column`.
- **`range_onset`** (*`int`*) – timestamp of the beginning of the range to consider (in milliseconds)
- **`range_offset`** (*`int`*) – timestamp of the end of the range to consider (in milliseconds)
- **`timescale`** (*`int`*) – length of each time unit (in milliseconds)
- **`column`** (*`str`*) – the name of the column in `segments` that should be used for the values of the annotations (e.g. speaker_type).
- **`categories`** (*`list`*) – the list of categories
- **`none`** (*`bool`*) – append a ‘none’ column, default True
- **`overlap`** (*`bool`*) – append an overlap column, default False

Returns the output grid

Return type numpy.array

`ChildProject.metrics.`**`vectors_to_annotation_task`**`(`**args*, *drop: List[str] = []*`)`

transform vectors of labels into a nltk AnnotationTask object.

Parameters

- **`*args`** – vector of labels for each annotator; add one argument per annotator.
- **`drop`** (*`List[str]`*) – list of labels that should be ignored

Returns the AnnotationTask object

Return type nltk.metrics.agreement.AnnotationTask

CHAPTER EIGHTEEN

EXAMPLES OF PYTHON SCRIPTS

We provide examples of python scripts using our package on GitHub.

You can test these scripts by running them on the VanDam demonstration dataset.

CHAPTER

NINETEEN

API REFERENCE

19.1 ChildProject package

19.1.1 Subpackages

ChildProject.pipelines package

Submodules

ChildProject.pipelines.anonymize module

class `ChildProject.pipelines.anonymize.`**`AnonymizationPipeline`**

Bases: *`ChildProject.pipelines.pipeline.Pipeline`*

Anonymize a set of its annotations (*input_set*) and saves it as *output_set*.

```
DEFAULT_REPLACEMENTS = {'Bar': {'startClockTime': [{'replace_value':
'1000-01-01'}, {'only_time': 'true'}]}, 'BarSummary': {'leftBoundaryClockTime':
[{'replace_value': '1000-01-01'}, {'only_time': 'true'}],
'rightBoundaryClockTime': [{'replace_value': '1000-01-01'}, {'only_time':
'true'}]}, 'Child': {'DOB': '1000-01-01', 'EnrollDate': '1000-01-01', 'id':
'A999'}, 'ChildInfo': {'dob': '1000-01-01'}, 'FiveMinuteSection':
{'endClockTime': [{'replace_value': '1000-01-01'}, {'only_time': 'true'}],
'startClockTime': [{'replace_value': '1000-01-01'}, {'only_time': 'true'}]},
'ITS': {'fileName': 'new_filename_1001', 'timeCreated': [{'replace_value':
'1000-01-01'}, {'only_time': 'true'}]}, 'Item': {'timeStamp': [{'replace_value':
'1000-01-01'}, {'only_time': 'true'}]}, 'PrimaryChild': {'DOB': '1000-01-01'},
'ProcessingJob': {'logfile':
'exec10001010T100010Z_job00000001-10001010_101010_100100.upl.log'}, 'Recording':
{'endClockTime': [{'replace_value': '1000-01-01'}, {'only_time': 'true'}],
'startClockTime': [{'replace_value': '1000-01-01'}, {'only_time': 'true'}]},
'ResourceSnapshot': {'timegmt': [{'replace_value': '1000-01-01'}, {'only_time':
'true'}], 'timelocal': [{'replace_value': '1000-01-01'}, {'only_time': 'true'}]},
'TransferTime': {'LocalTime': [{'replace_value': '1000-01-01'}, {'only_time':
'true'}], 'UTCTime': [{'replace_value': '1000-01-01'}, {'only_time': 'true'}]}}
```

`run`(*path: str*, *input_set: str*, *output_set: str*, *replacements_json_dict: str = ''*, ***kwargs*)

Anonymize a set of its annotations (*input_set*) and saves it as *output_set*.

`static setup_parser`(*parser*)

ChildProject.pipelines.eafbuilder module

`class ChildProject.pipelines.eafbuilder.`**`EafBuilderPipeline`**

Bases: *`ChildProject.pipelines.pipeline.Pipeline`*

`run`(*destination: str*, *segments: str*, *eaf_type: str*, *template: str*, *context_onset: int = 0*, *context_offset: int = 0*, ***kwargs*)

generate .eaf templates based on intervals to code.

Parameters

- **`path`** (*`str`*) – project path
- **`destination`** (*`str`*) – eaf destination
- **`segments`** (*`str`*) – path to the input segments dataframe
- **`eaf_type`** (*`str`*) – eaf-type [random, periodic]
- **`template`** (*`str`*) – name of the template to use (basic, native, or non-native)
- **`context_onset`** (*`int`*) – context onset and segment offset difference in milliseconds, 0 for no introductory context
- **`context_offset`** (*`int`*) – context offset and segment offset difference in milliseconds, 0 for no outro context

`static setup_parser`(*parser*)

`ChildProject.pipelines.eafbuilder.`**`create_eaf`**(*etf_path: str*, *id: str*, *output_dir: str*, *recording_filename: str*, *timestamps_list: list*, *eaf_type: str*, *contxt_on: int*, *contxt_off: int*, *template: str*)

ChildProject.pipelines.metrics module

`class ChildProject.pipelines.metrics.`**`AclewMetrics`**(*project:* ChildProject.projects.ChildProject, *vtc: str = 'vtc'*, *alice: str = 'alice'*, *vcm: str = 'vcm'*, *recordings: Optional[Union[str, List[str], pandas.core.frame.DataFrame]] = None*, *from_time: Optional[str] = None*, *to_time: Optional[str] = None*, *by: str = 'recording_filename'*, *threads: int = 1*)

Bases: *`ChildProject.pipelines.metrics.Metrics`*

ACLEW metrics extractor. Extracts a number of metrics from the ACLEW pipeline annotations, which includes:

- The Voice Type Classifier by Lavechin et al. (arXiv:2005.12656)
- The Automatic LInguistic Unit Count Estimator (ALICE) by Räsänen et al. (doi:10.3758/s13428-020-01460-x)
- The VoCalisation Maturity model (VCMNet) by Al Futaisi et al. (doi:10.1145/3340555.3353751)

Parameters

- **`project`** (`ChildProject.projects.ChildProject`) – ChildProject instance of the target dataset.
- **`vtc`** (*`str`*) – name of the set associated to the VTC annotations
- **`alice`** (*`str`*) – name of the set associated to the ALICE annotations

- **vcm** (*str*) – name of the set associated to the VCM annotations
- **recordings** (*Union[str, List[str], pd.DataFrame], optional*) – recordings to sample from; if None, all recordings will be sampled, defaults to None
- **from_time** (*str, optional*) – If specified (in HH:MM format), ignore annotations outside of the given time-range, defaults to None
- **to_time** (*str, optional*) – If specified (in HH:MM format), ignore annotations outside of the given time-range, defaults to None
- **by** (*str, optional*) – units to sample from, defaults to 'recording_filename'
- **threads** (*int, optional*) – amount of threads to run on, defaults to 1

`SUBCOMMAND = 'aclew'`

`static add_parser`(*subparsers*, *subcommand*)

`extract()`

`class ChildProject.pipelines.metrics.LenaMetrics`(*project: ChildProject.projects.ChildProject*, *set: str*, *types: list = []*, *recordings: Optional[Union[str, List[str], pandas.core.frame.DataFrame]] = None*, *from_time: Optional[str] = None*, *to_time: Optional[str] = None*, *by: str = 'recording_filename'*, *threads: int = 1*)

Bases: *ChildProject.pipelines.metrics.Metrics*

LENA metrics extractor. Extracts a number of metrics from the LENA .its annotations.

Parameters

- **project** (ChildProject.projects.ChildProject) – ChildProject instance of the target dataset.
- **set** (*str*) – name of the set associated to the .its annotations
- **types** (*list*) – list of LENA vocalization/noise types (e.g. OLN, TVN)
- **recordings** (*Union[str, List[str], pd.DataFrame], optional*) – recordings to sample from; if None, all recordings will be sampled, defaults to None
- **from_time** (*str, optional*) – If specified (in HH:MM format), ignore annotations outside of the given time-range, defaults to None
- **to_time** (*str, optional*) – If specified (in HH:MM format), ignore annotations outside of the given time-range, defaults to None
- **by** (*str, optional*) – units to sample from, defaults to 'recording_filename'
- **threads** (*int, optional*) – amount of threads to run on, defaults to 1

`SUBCOMMAND = 'lena'`

`static add_parser`(*subparsers*, *subcommand*)

`extract()`

`class ChildProject.pipelines.metrics.Metrics`(*project: ChildProject.projects.ChildProject*, *by: str = 'recording_filename'*, *recordings: Optional[Union[str, List[str], pandas.core.frame.DataFrame]] = None*, *from_time: Optional[str] = None*, *to_time: Optional[str] = None*)

Bases: `abc.ABC`

abstract extract()

retrieve_segments(*sets: List[str]*, *unit: str*)

class ChildProject.pipelines.metrics.**MetricsPipeline**

Bases: *ChildProject.pipelines.pipeline.Pipeline*

run(*path*, *destination*, *pipeline*, *func=None*, ***kwargs*)

static setup_parser(*parser*)

class ChildProject.pipelines.metrics.**PeriodMetrics**(*project: ChildProject.projects.ChildProject*, *set: str*, *period: str*, *period_origin: Optional[str] = None*, *recordings: Optional[Union[str, List[str], pandas.core.frame.DataFrame]] = None*, *from_time: Optional[str] = None*, *to_time: Optional[str] = None*, *by: str = 'recording_filename'*, *threads: int = 1*)

Bases: *ChildProject.pipelines.metrics.Metrics*

Time-aggregated metrics extractor.

Aggregates vocalizations for each time-of-the-day-unit based on a period specified by the user. For instance, if the period is set to `15Min` (i.e. 15 minutes), vocalization rates will be reported for each recording and time-unit (e.g. 09:00 to 09:15, 09:15 to 09:30, etc.).

The output dataframe has `rp` rows, where `r` is the amount of recordings (or children if the `--by` option is set to `child_id`), `p` is the amount of time-bins per day (i.e. 24 x 4 = 96 for a 15-minute period).

The output dataframe includes a `period` column that contains the onset of each time-unit in HH:MM:SS format. The `duration` columns contains the total amount of annotations covering each time-bin, in milliseconds.

If `--by` is set to e.g. `child_id`, then the values for each time-bin will be the average rates across all the recordings of every child.

Parameters

- **project** (`ChildProject.projects.ChildProject`) – ChildProject instance of the target dataset
- **set** (*str*) – name of the set of annotations to derive the metrics from
- **period** (*str*) – Time-period. Values should be formatted as pandas offset aliases. For instance, *15Min* corresponds to a 15 minute period; *2H* corresponds to a 2 hour period.
- **period_origin** (*str, optional*) – NotImplemented, defaults to None
- **recordings** (*Union[str, List[str], pd.DataFrame], optional*) – white-list of recordings to process, defaults to None
- **from_time** (*str, optional*) – If specified (in HH:MM format), ignore annotations outside of the given time-range, defaults to None
- **to_time** (*str, optional*) – If specified (in HH:MM format), ignore annotations outside of the given time-range, defaults to None
- **by** (*str, optional*) – units to sample from, defaults to 'recording_filename'
- **threads** (*int, optional*) – amount of threads to run on, defaults to 1

SUBCOMMAND = 'period'

static add_parser(*subparsers*, *subcommand*)

extract()

ChildProject.pipelines.pipeline module

`class ChildProject.pipelines.pipeline.`**`Pipeline`**

Bases: `abc.ABC`

`check_setup``()`

`static` **`recordings_from_list`**`(`*recordings*`)`

`abstract` **`run`**`(`***kwargs*`)`

`setup``()`

`static` **`setup_pipeline`**`(`*parser*`)`

ChildProject.pipelines.processors module

`class ChildProject.pipelines.processors.`**`AudioProcessingPipeline`**

Bases: *`ChildProject.pipelines.pipeline.Pipeline`*

`run``(`*path: str*, *name: str*, *processor: str*, *threads: int = 1*, *func=None*, ***kwargs*`)`

`static` **`setup_parser`**`(`*parser*`)`

`class ChildProject.pipelines.processors.`**`AudioProcessor`**`(`*project: ChildProject.projects.ChildProject*, *name: str*, *input_profile: Optional[str] = None*, *threads: int = 1*, *recordings: Optional[Union[str, List[str], pandas.core.frame.DataFrame]] = None*`)`

Bases: `abc.ABC`

`static` **`add_parser`**`(`*parsers*`)`

`export_metadata``()`

`output_directory``()`

`process``(`*parameters*`)`

`abstract` **`process_recording`**`(`*recording*`)`

`read_metadata``()`

`class ChildProject.pipelines.processors.`**`BasicProcessor`**`(`*project: ChildProject.projects.ChildProject*, *name: str*, *format: str*, *codec: str*, *sampling: int*, *split: Optional[str] = None*, *threads: int = 1*, *recordings: Optional[Union[str, List[str], pandas.core.frame.DataFrame]] = None*, *skip_existing: bool = False*, *input_profile: Optional[str] = None*`)`

Bases: *`ChildProject.pipelines.processors.AudioProcessor`*

`SUBCOMMAND = 'basic'`

`static` **`add_parser`**`(`*subparsers*, *subcommand*`)`

`process_recording``(`*recording*`)`

class ChildProject.pipelines.processors.**ChannelMapper**(*project: ChildProject.projects.ChildProject, name: str, channels: list, threads: int = 1, recordings: Optional[Union[str, List[str], pandas.core.frame.DataFrame]] = None, input_profile: Optional[str] = None*)

Bases: `ChildProject.pipelines.processors.AudioProcessor`

SUBCOMMAND = 'channel-mapping'

static add_parser(*subparsers, subcommand*)

process_recording(*recording*)

class ChildProject.pipelines.processors.**VettingProcessor**(*project: ChildProject.projects.ChildProject, name: str, segments_path: str, threads: int = 1, recordings: Optional[Union[str, List[str], pandas.core.frame.DataFrame]] = None, input_profile: Optional[str] = None*)

Bases: `ChildProject.pipelines.processors.AudioProcessor`

SUBCOMMAND = 'vetting'

static add_parser(*subparsers, subcommand*)

process_recording(*recording*)

ChildProject.pipelines.samplers module

class ChildProject.pipelines.samplers.**ConversationSampler**(*project: ChildProject.projects.ChildProject, annotation_set: str, count: int, interval: int = 1000, speakers: List[str] = ['FEM', 'MAL', 'CHI'], threads: int = 1, by: str = 'recording_filename', recordings: Optional[Union[str, List[str], pandas.core.frame.DataFrame]] = None, exclude: Optional[Union[str, pandas.core.frame.DataFrame]] = None*)

Bases: `ChildProject.pipelines.samplers.Sampler`

Conversation sampler.

Parameters

- **project** (`ChildProject.projects.ChildProject`) – ChildProject instance
- **annotation_set** (`str`) – set of annotation to derive conversations from
- **count** (`int`) – amount of conversations to sample
- **interval** (`int, optional`) – maximum time-interval between two consecutive vocalizations (in milliseconds) to consider them part of the same conversational block, defaults to 1000
- **speakers** (`List[str], optional`) – list of speakers to target, defaults to ["FEM", "MAL", "CHI"]
- **threads** (`int, optional`) – threads to run on, defaults to 1

- **by** (*str, optional*) – units to sample from, defaults to "recording_filename"
- **recordings** (*Union[str, List[str], pd.DataFrame], optional*) – whitelist of recordings, defaults to None
- **exclude** (*Union[str, pd.DataFrame], optional*) – portions to exclude, defaults to None

SUBCOMMAND = 'conversations'

static add_parser(*subparsers*, *subcommand*)

class ChildProject.pipelines.samplers.**CustomSampler**(*project: ChildProject.projects.ChildProject*, *segments_path: str*, *recordings: Optional[Union[str, List[str], pandas.core.frame.DataFrame]] = None*, *exclude: Optional[Union[str, pandas.core.frame.DataFrame]] = None*)

Bases: `ChildProject.pipelines.samplers.Sampler`

SUBCOMMAND = 'custom'

static add_parser(*subparsers*, *subcommand*)

class ChildProject.pipelines.samplers.**EnergyDetectionSampler**(*project: ChildProject.projects.ChildProject*, *windows_length: int*, *windows_spacing: int*, *windows_count: int*, *windows_offset: int = 0*, *threshold: float = 0.8*, *low_freq: int = 0*, *high_freq: int = 100000*, *threads: int = 1*, *profile: str = ''*, *by: str = 'recording_filename'*, *recordings: Optional[Union[str, List[str], pandas.core.frame.DataFrame]] = None*, *exclude: Optional[Union[str, pandas.core.frame.DataFrame]] = None*)

Bases: `ChildProject.pipelines.samplers.Sampler`

Sample windows within each recording, targetting those that have a signal energy higher than some threshold.

Parameters

- **project** (`ChildProject.projects.ChildProject`) – ChildProject instance of the target dataset.
- **windows_length** (*int*) – Length of each window, in milliseconds.
- **windows_spacing** (*int*) – Spacing between the start of each window, in milliseconds.
- **windows_count** (*int*) – How many windows to retain per recording.
- **windows_offset** (*float, optional*) – start of the first window, in milliseconds, defaults to 0
- **threshold** (*float, optional*) – lowest energy quantile to sample from, defaults to 0.8
- **low_freq** (*int, optional*) – if > 0, frequencies below will be filtered before calculating the energy, defaults to 0

- **high_freq** (*int, optional*) – if < 100000, frequencies above will be filtered before calculating the energy, defaults to 100000
- **by** (*str, optional*) – units to sample from, defaults to 'recording_filename'
- **recordings** (*Union[str, List[str], pd.DataFrame], optional*) – recordings to sample from; if None, all recordings will be sampled, defaults to None
- **threads** (*int, optional*) – amount of threads to run on, defaults to 1

SUBCOMMAND = 'energy-detection'

static **add_parser**(*subparsers*, *subcommand*)

compute_energy_loudness(*chunk*, *sampling_frequency: int*)

get_recording_windows(*recording*)

class ChildProject.pipelines.samplers.**HighVolubilitySampler**(*project: ChildProject.projects.ChildProject*, *annotation_set: str*, *metric: str*, *windows_length: int*, *windows_count: int*, *speakers: List[str] = ['FEM', 'MAL', 'CHI']*, *threads: int = 1*, *by: str = 'recording_filename'*, *recordings: Optional[Union[str, List[str], pandas.core.frame.DataFrame]] = None*, *exclude: Optional[Union[str, pandas.core.frame.DataFrame]] = None*)

Bases: *ChildProject.pipelines.samplers.Sampler*

Return the top `windows_count` windows (of length `windows_length`) with the highest volubility from each recording, as calculated from the metric `metric`.

`metrics` can be any of three values: words, turns, and vocs.

- The **words** metric sums the amount of words within each window. For LENA annotations, it is equivalent to **awc**.
- The **turns** metric (aka ctc) sums conversational turns within each window. It relies on **lena_conv_turn_type** for LENA annotations. For other annotations, turns are estimated as adult/child speech switches in close temporal proximity.
- The **vocs** metric sums vocalizations within each window. If `metric="vocs"` and `speakers=['CHI']`, it is equivalent to the usual cvc metric (child vocalization counts).

Parameters

- **project** (`ChildProject.projects.ChildProject`) – ChildProject instance of the target dataset.
- **annotation_set** (*str*) – set of annotations to calculate volubility from.
- **metric** (*str*) – the metric to evaluate high-volubility. should be any of 'words', 'turns', 'vocs'.
- **windows_length** (*int*) – length of the windows, in milliseconds
- **windows_count** (*int*) – amount of top regions to extract per recording
- **by** (*str, optional*) – units to sample from, defaults to 'recording_filename'

- **recordings** (*Union[str, List[str], pd.DataFrame], optional*) – recordings to sample from; if None, all recordings will be sampled, defaults to None
- **threads** (*int*) – amount of threads to run the sampler on

SUBCOMMAND = 'high-volubility'

static add_parser(*subparsers*, *subcommand*)

class ChildProject.pipelines.samplers.**PeriodicSampler**(*project: ChildProject.projects.ChildProject*, *length: int*, *period: int*, *offset: int = 0*, *profile: Optional[str] = None*, *recordings: Optional[Union[str, List[str], pandas.core.frame.DataFrame]] = None*, *exclude: Optional[Union[str, pandas.core.frame.DataFrame]] = None*)

Bases: `ChildProject.pipelines.samplers.Sampler`

Periodic sampling of a recording.

Parameters

- **project** (`ChildProject.projects.ChildProject`) – ChildProject instance of the target dataset.
- **length** (*int*) – length of each segment, in milliseconds
- **period** (*int*) – spacing between two consecutive segments, in milliseconds
- **offset** (*int*) – offset of the first segment, in milliseconds, defaults to 0
- **recordings** (*Union[str, List[str], pd.DataFrame], optional*) – recordings to sample from; if None, all recordings will be sampled, defaults to None

SUBCOMMAND = 'periodic'

static add_parser(*subparsers*, *subcommand*)

class ChildProject.pipelines.samplers.**RandomVocalizationSampler**(*project: ChildProject.projects.ChildProject*, *annotation_set: str*, *target_speaker_type: list*, *sample_size: int*, *threads: int = 1*, *by: str = 'recording_filename'*, *recordings: Optional[Union[str, List[str], pandas.core.frame.DataFrame]] = None*, *exclude: Optional[Union[str, pandas.core.frame.DataFrame]] = None*)

Bases: `ChildProject.pipelines.samplers.Sampler`

Sample vocalizations based on some input annotation set.

Parameters

- **project** (`ChildProject.projects.ChildProject`) – ChildProject instance of the target dataset.
- **annotation_set** (*str*) – Set of annotations to get vocalizations from.
- **target_speaker_type** (*list*) – List of speaker types to sample vocalizations from.

- **sample_size** (*int*) – Amount of vocalizations to sample, per recording.
- **by** (*str, optional*) – units to sample from, defaults to 'recording_filename'
- **recordings** (*Union[str, List[str], pd.DataFrame], optional*) – recordings to sample from; if None, all recordings will be sampled, defaults to None
- **threads** (*int, optional*) – amount of threads to run on, defaults to 1

`SUBCOMMAND = 'random-vocalizations'`

`static add_parser`(*subparsers*, *subcommand*)

`class ChildProject.pipelines.samplers.Sampler`(*project: ChildProject.projects.ChildProject*, *recordings: Optional[Union[str, List[str], pandas.core.frame.DataFrame]] = None*, *exclude: Optional[Union[str, pandas.core.frame.DataFrame]] = None*)

Bases: `abc.ABC`

`abstract static add_parser`(*parsers*)

`assert_valid`()

`export_audio`(*destination*, *profile=None*, ***kwargs*)

`remove_excluded`()

`retrieve_segments`(*recording_filename=None*)

`sample`()

`class ChildProject.pipelines.samplers.SamplerPipeline`

Bases: `ChildProject.pipelines.pipeline.Pipeline`

`run`(*path*, *destination*, *sampler*, *func=None*, ***kwargs*)

`static setup_parser`(*parser*)

ChildProject.pipelines.zooniverse module

`class ChildProject.pipelines.zooniverse.Chunk`(*recording_filename*, *onset*, *offset*, *segment_onset*, *segment_offset*)

Bases: `object`

`getbasename`(*extension*)

`class ChildProject.pipelines.zooniverse.ZooniversePipeline`

Bases: `ChildProject.pipelines.pipeline.Pipeline`

`extract_chunks`(*path: str*, *destination: str*, *keyword: str*, *segments: str*, *chunks_length: int = - 1*, *chunks_min_amount: int = 1*, *profile: str = ''*, *threads: int = 1*, ***kwargs*)

extract-audio chunks based on a list of segments and prepare them for upload to zooniverse.

Parameters

- **path** (*str*) – dataset path
- **destination** (*str*) – path to the folder where to store the metadata and audio chunks
- **segments** (*str*) – path to the input segments csv dataframe, defaults to None
- **keyword** (*str*) – keyword to insert in the output metadata
- **chunks_length** (*int, optional*) – length of the chunks, in milliseconds, defaults to -1

- **chunks_min_amount** (*int, optional*) – minimum amount of chunk per segment, defaults to 1
- **profile** (*str*) – recording profile to extract from. If undefined, raw recordings will be used.
- **threads** (*int, optional*) – amount of threads to run-on, defaults to 0

`get_credentials`(*login: str = ''*, *pwd: str = ''*)
returns input credentials if provided or attempts to read them from the environment variables.

Parameters

- **login** (*str, optional*) – input login, defaults to ‘’
- **pwd** (*str, optional*) – input password, defaults to ‘’

Returns (login, pwd)

Return type (str, str)

`retrieve_classifications`(*destination: str*, *project_id: int*, *zooniverse_login: str = ''*, *zooniverse_pwd: str = ''*, *chunks: List[str] = []*, ***kwargs*)
Retrieve classifications from Zooniverse as a CSV dataframe. They will be matched with the original chunks metadata if the path one or more chunk metadata files is provided.

Parameters

- **destination** (*str*) – output CSV dataframe destination
- **project_id** (*int*) – zooniverse project id
- **zooniverse_login** (*str, optional*) – zooniverse login. If not specified, the program attempts to get it from the environment variable `ZOONIVERSE_LOGIN` instead, defaults to ‘’
- **zooniverse_pwd** (*str, optional*) – zooniverse password. If not specified, the program attempts to get it from the environment variable `ZOONIVERSE_PWD` instead, defaults to ‘’
- **chunks** (*List[str], optional*) – the list of chunk metadata files to match the classifications to. If provided, only the classifications that have a match will be returned.

`run`(*action*, ***kwargs*)

`static setup_parser`(*parser*)

`upload_chunks`(*chunks: str*, *project_id: int*, *set_name: str*, *zooniverse_login=''*, *zooniverse_pwd=''*, *amount: int = 1000*, *ignore_errors: bool = False*, ***kwargs*)
Uploads `amount` audio chunks from the CSV dataframe *chunks* to a zooniverse project.

Parameters

- **chunks** (*[type]*) – path to the chunk CSV dataframe
- **project_id** (*int*) – zooniverse project id
- **set_name** (*str*) – name of the subject set
- **zooniverse_login** (*str, optional*) – zooniverse login. If not specified, the program attempts to get it from the environment variable `ZOONIVERSE_LOGIN` instead, defaults to ‘’
- **zooniverse_pwd** (*str, optional*) – zooniverse password. If not specified, the program attempts to get it from the environment variable `ZOONIVERSE_PWD` instead, defaults to ‘’
- **amount** (*int, optional*) – amount of chunks to upload, defaults to 0

`ChildProject.pipelines.zooniverse.`**`pad_interval`**(*onset: int*, *offset: int*, *chunks_length: int*, *chunks_min_amount: int = 1*) → Tuple[int, int]

Module contents

ChildProject.templates package

Module contents

19.1.2 Submodules

19.1.3 ChildProject.annotations module

class `ChildProject.annotations.`**`AnnotationManager`**(*project:* ChildProject.projects.ChildProject)

Bases: `object`

INDEX_COLUMNS = [IndexColumn(name = set), IndexColumn(name = recording_filename), IndexColumn(name = time_seek), IndexColumn(name = range_onset), IndexColumn(name = range_offset), IndexColumn(name = raw_filename), IndexColumn(name = format), IndexColumn(name = filter), IndexColumn(name = annotation_filename), IndexColumn(name = imported_at), IndexColumn(name = package_version), IndexColumn(name = error)]

SEGMENTS_COLUMNS = [IndexColumn(name = raw_filename), IndexColumn(name = segment_onset), IndexColumn(name = segment_offset), IndexColumn(name = speaker_id), IndexColumn(name = speaker_type), IndexColumn(name = ling_type), IndexColumn(name = vcm_type), IndexColumn(name = lex_type), IndexColumn(name = mwu_type), IndexColumn(name = addressee), IndexColumn(name = transcription), IndexColumn(name = phonemes), IndexColumn(name = syllables), IndexColumn(name = words), IndexColumn(name = lena_block_type), IndexColumn(name = lena_block_number), IndexColumn(name = lena_conv_status), IndexColumn(name = lena_response_count), IndexColumn(name = lena_conv_floor_type), IndexColumn(name = lena_conv_turn_type), IndexColumn(name = lena_speaker), IndexColumn(name = utterances_count), IndexColumn(name = utterances_length), IndexColumn(name = non_speech_length), IndexColumn(name = average_db), IndexColumn(name = peak_db), IndexColumn(name = child_cry_vfx_len), IndexColumn(name = utterances), IndexColumn(name = cries), IndexColumn(name = vfxs)]

static clip_segments(*segments: pandas.core.frame.DataFrame*, *start: int*, *stop: int*) → pandas.core.frame.DataFrame

Clip all segments onsets and offsets within `start` and `stop`. Segments outside of the range [`start`,``stop``] will be removed.

Parameters

- **segments** (*pd.DataFrame*) – Dataframe of the segments to clip
- **start** (*int*) – range start (in milliseconds)
- **stop** (*int*) – range end (in milliseconds)

Returns Dataframe of the clipped segments

Return type pd.DataFrame

`get_collapsed_segments`(*annotations: pandas.core.frame.DataFrame*) → pandas.core.frame.DataFrame
get all segments associated to the annotations referenced in `annotations`, and collapses into one virtual timeline.

Parameters **`annotations`** (*`pd.DataFrame`*) – dataframe of annotations, according to *Annotations index*

Returns dataframe of all the segments merged (as specified in *Annotations format*), merged with `annotations`

Return type pd.DataFrame

`get_segments`(*annotations: pandas.core.frame.DataFrame*) → pandas.core.frame.DataFrame
get all segments associated to the annotations referenced in `annotations`.

Parameters **`annotations`** (*`pd.DataFrame`*) – dataframe of annotations, according to *Annotations index*

Returns dataframe of all the segments merged (as specified in *Annotations format*), merged with `annotations`.

Return type pd.DataFrame

`get_segments_timestamps`(*segments: pandas.core.frame.DataFrame*, *ignore_date: bool = False*, *onset: str = 'segment_onset'*, *offset: str = 'segment_offset'*) → pandas.core.frame.DataFrame
Calculate the onset and offset clock-time of each segment

Parameters

- **`segments`** (*`pd.DataFrame`*) – DataFrame of segments (as returned by `get_segments()`).
- **`ignore_date`** (*`bool, optional`*) – leave date information and use time data only, defaults to False
- **`onset`** (*`str, optional`*) – column storing the onset timestamp in milliseconds, defaults to “segment_onset”
- **`offset`** (*`str, optional`*) – column storing the offset timestamp in milliseconds, defaults to “segment_offset”

Returns Returns the input dataframe with two new columns `onset_time` and `offset_time`.

`onset_time` is a datetime object corresponding to the onset of the segment. `offset_time` is a datetime object corresponding to the offset of the segment. In case either `start_time` or `date_iso` is not specified for the corresponding recording, both values will be set to NaT. :rtype: pd.DataFrame

`get_subsets`(*annotation_set: str*, *recursive: bool = False*) → List[str]
Retrieve the list of subsets belonging to a given set of annotations.

Parameters

- **`annotation_set`** (*`str`*) – input set
- **`recursive`** (*`bool, optional`*) – If True, get subsets recursively, defaults to False

Returns the list of subsets names

Return type list

`get_within_ranges`(*ranges: pandas.core.frame.DataFrame*, *sets: Optional[Union[Set, List]] = None*, *missing_data: str = 'ignore'*)
Retrieve and clip annotations that cover specific portions of recordings (`ranges`).

The desired ranges are defined by an input dataframe with three columns: `recording_filename`, `range_onset`, and `range_offset`. The function returns a dataframe of annotations under the same format as the index of annotations (*Annotations index*).

This output get can then be provided to `get_segments()` in order to retrieve segments of annotations that match the desired range.

For instance, the code belows will prints all the segments of annotations corresponding to the first hour of each recording:

```
>>> from ChildProject.projects import ChildProject
>>> from ChildProject.annotations import AnnotationManager
>>> project = ChildProject('.')
>>> am = AnnotationManager(project)
>>> am.read()
>>> ranges = project.recordings
>>> ranges['range_onset'] = 0
>>> ranges['range_offset'] = 60*60*1000
>>> matches = am.get_within_ranges(ranges)
>>> am.get_segments(matches)
```

Parameters

- **ranges** (*pd.DataFrame*) – pandas dataframe with one row per range to be considered and three columns: `recording_filename`, `range_onset`, `range_offset`.
- **sets** (*Union[Set, List]*) – optional list of annotation sets to retrieve. If None, all annotations from all sets will be retrieved.
- **missing_data** (*str, defaults to ignore*) – how to handle missing annotations ("ignore", "warn" or "raise")

Return type pd.DataFrame

get_within_time_range(*annotations: pandas.core.frame.DataFrame*, *start_time: str*, *end_time: str*, *errors='raise'*)

Clip all input annotations within a given HH:MM clock-time range. Those that do not intersect the input time range at all are filtered out.

Parameters

- **annotations** (*pd.DataFrame*) – DataFrame of input annotations to filter. The only columns that are required are: `recording_filename`, `range_onset`, and `range_offset`.
- **start** (*str*) – onset HH:MM clocktime
- **end** (*str*) – offset HH:MM clocktime
- **errors** (*str*) – how to deal with invalid start_time values for the recordings. Takes the same values as `pandas.to_datetime`.

Returns a DataFrame of annotations;

For each row, `range_onset` and `range_offset` are clipped within the desired clock-time range. The clock-time corresponding to the onset and offset of each annotation is stored in two newly created columns named `range_onset_time` and `range_offset_time`. If the input annotation exceeds 24 hours, one row per matching interval is returned. :rtype: pd.DataFrame

import_annotations(*input: pandas.core.frame.DataFrame*, *threads: int = - 1*, *import_function: Optional[Callable[[str], pandas.core.frame.DataFrame]] = None*) → pandas.core.frame.DataFrame

Import and convert annotations.

Parameters

- **input** (*pd.DataFrame*) – dataframe of all annotations to import, as described in format-input-annotations.
- **threads** (*int, optional*) – If > 1, conversions will be run on `threads` threads, defaults to -1
- **import_function** (*Callable[[str], pd.DataFrame], optional*) – If specified, the custom `import_function` function will be used to convert all `input` annotations, defaults to None

Returns dataframe of imported annotations, as in *Annotations index*.

Return type pd.DataFrame

static intersection(*annotations: pandas.core.frame.DataFrame*, *sets: Optional[list] = None*) → pandas.core.frame.DataFrame

Compute the intersection of all annotations for all sets and recordings, based on their `recording_filename`, `range_onset` and `range_offset` attributes. (Only these columns are required, but more can be passed and they will be preserved).

Parameters **annotations** (*pd.DataFrame*) – dataframe of annotations, according to *Annotations index*

Returns dataframe of annotations, according to *Annotations index*

Return type pd.DataFrame

merge_annotations(*left_columns*, *right_columns*, *columns*, *output_set*, *input*)

merge_sets(*left_set: str*, *right_set: str*, *left_columns: List[str]*, *right_columns: List[str]*, *output_set: str*, *columns: dict = {}*, *threads=- 1*)

Merge columns from `left_set` and `right_set` annotations, for all matching segments, into a new set of annotations named `output_set`.

Parameters

- **left_set** (*str*) – Left set of annotations.
- **right_set** (*str*) – Right set of annotations.
- **left_columns** (*List*) – Columns which values will be based on the left set.
- **right_columns** (*List*) – Columns which values will be based on the right set.
- **output_set** (*str*) – Name of the output annotations set.

Returns [description]

Return type [type]

read() → Tuple[List[str], List[str]]

Read the index of annotations from `metadata/annotations.csv` and store it into self.annotations.

Returns a tuple containing the list of errors and the list of warnings generated while reading the index

Return type Tuple[List[str],List[str]]

remove_set(*annotation_set: str*, *recursive: bool = False*)
Remove a set of annotations, deleting every converted file and removing them from the index. This preserves raw annotations.

Parameters

- **annotation_set** (*str*) – set of annotations to remove
- **recursive** (*bool, optional*) – remove subsets as well, defaults to False

rename_set(*annotation_set: str*, *new_set: str*, *recursive: bool = False*, *ignore_errors: bool = False*)
Rename a set of annotations, moving all related files and updating the index accordingly.

Parameters

- **annotation_set** (*str*) – name of the set to rename
- **new_set** (*str*) – new set name
- **recursive** (*bool, optional*) – rename subsets as well, defaults to False
- **ignore_errors** (*bool, optional*) – If True, keep going even if unindexed files are detected, defaults to False

set_from_path(*path: str*) → str

validate(*annotations: Optional[pandas.core.frame.DataFrame] = None*, *threads: int = 0*) → Tuple[List[str], List[str]]
check all indexed annotations for errors

Parameters

- **annotations** (*pd.DataFrame, optional*) – annotations to validate, defaults to None. If None, the whole index will be scanned.
- **threads** (*int, optional*) – how many threads to run the tests with, defaults to 0. If <= 0, all available CPU cores will be used.

Returns a tuple containg the list of errors and the list of warnings detected

Return type Tuple[List[str], List[str]]

validate_annotation(*annotation: dict*) → Tuple[List[str], List[str]]

write()
Update the annotations index, while enforcing its good shape.

19.1.4 ChildProject.cmdline module

ChildProject.cmdline.**arg**(**name_or_flags*, ***kwargs*)

ChildProject.cmdline.**get_doc_summary**(*doc*)

ChildProject.cmdline.**main**()

ChildProject.cmdline.**perform_validation**(*project: ChildProject.projects.ChildProject*, *require_success: bool = True*, ***args*)

ChildProject.cmdline.**register_pipeline**(*subcommand*, *cls*)

`ChildProject.cmdline.`**`subcommand`**`(`*args=[], parent=_SubParsersAction(option_strings=[], dest='==SUPPRESS==', nargs='A...', const=None, default=None, type=None, choices={'validate': ArgumentParser(prog='__main__.py validate', usage=None, description='validate the consistency of the dataset returning detailed errors and warnings', formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error', add_help=True), 'import-annotations': ArgumentParser(prog='__main__.py import-annotations', usage=None, description='convert and import a set of annotations', formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error', add_help=True), 'merge-annotations': ArgumentParser(prog='__main__.py merge-annotations', usage=None, description='merge segments sharing identical onset and offset from two sets of annotations', formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error', add_help=True), 'intersect-annotations': ArgumentParser(prog='__main__.py intersect-annotations', usage=None, description='calculate the intersection of the annotations belonging to the given sets', formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error', add_help=True), 'remove-annotations': ArgumentParser(prog='__main__.py remove-annotations', usage=None, description='remove converted annotations of a given set and their entries in the index', formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error', add_help=True), 'rename-annotations': ArgumentParser(prog='__main__.py rename-annotations', usage=None, description='rename a set of annotations by moving the files and updating the index accordingly', formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error', add_help=True), 'overview': ArgumentParser(prog='__main__.py overview', usage=None, description='prints an overview of the contents of a given dataset', formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error', add_help=True), 'explain': ArgumentParser(prog='__main__.py explain', usage=None, description='prints information about a certain metadata variable', formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error', add_help=True), 'compute-durations': ArgumentParser(prog='__main__.py compute-durations', usage=None, description="creates a 'duration' column into metadata/recordings", formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error', add_help=True)}, help=None, metavar=None)*`)`

19.1.5 ChildProject.converters module

class `ChildProject.converters.`**`AliceConverter`**

Bases: `ChildProject.converters.AnnotationConverter`

`FORMAT = 'alice'`

static convert(*filename: str*, *source_file: str = ''*) → pandas.core.frame.DataFrame

class `ChildProject.converters.`**`AnnotationConverter`**

Bases: `object`

SPEAKER_ID_TO_TYPE = {'C1': 'OCH', 'C2': 'OCH', 'CHI': 'CHI', 'CHI*': 'CHI', 'EE1': 'NA', 'EE2': 'NA', 'FA0': 'FEM', 'FA1': 'FEM', 'FA2': 'FEM', 'FA3': 'FEM', 'FA4': 'FEM', 'FA5': 'FEM', 'FA6': 'FEM', 'FA7': 'FEM', 'FA8': 'FEM', 'FAE': 'NA', 'FC1': 'OCH', 'FC2': 'OCH', 'FC3': 'OCH', 'FCE': 'NA', 'MA0': 'MAL', 'MA1': 'MAL', 'MA2': 'MAL', 'MA3': 'MAL', 'MA4': 'MAL', 'MA5': 'MAL', 'MAE': 'NA', 'MC1': 'OCH', 'MC2': 'OCH', 'MC3': 'OCH', 'MC4': 'OCH', 'MC5': 'OCH', 'MCE': 'NA', 'MI1': 'OCH', 'MOT*': 'FEM', 'OC0': 'OCH', 'UA1': 'NA', 'UA2': 'NA', 'UA3': 'NA', 'UA4': 'NA', 'UA5': 'NA', 'UA6': 'NA', 'UC1': 'OCH', 'UC2': 'OCH', 'UC3': 'OCH', 'UC4': 'OCH', 'UC5': 'OCH', 'UC6': 'OCH'}

THREAD_SAFE = True

class ChildProject.converters.**ChatConverter**
Bases: *ChildProject.converters.AnnotationConverter*

ADDRESSEE_TABLE = {'CHI': 'T', 'FEM': 'A', 'MAL': 'A', 'OCH': 'C'}

FORMAT = 'cha'

SPEAKER_ROLE_TO_TYPE = {'Adult': 'NA', 'Attorney': 'NA', 'Audience': 'NA', 'Boy': 'OCH', 'Brother': 'OCH', 'Caretaker': 'NA', 'Child': 'OCH', 'Doctor': 'NA', 'Environment': 'NA', 'Father': 'MAL', 'Female': 'FEM', 'Friend': 'OCH', 'Girl': 'OCH', 'Grandfather': 'MAL', 'Grandmother': 'FEM', 'Group': 'NA', 'Guest': 'NA', 'Host': 'NA', 'Investigator': 'NA', 'Justice': 'NA', 'LENA': 'NA', 'Leader': 'NA', 'Male': 'MAL', 'Media': 'NA', 'Member': 'NA', 'Mother': 'FEM', 'Narrator': 'NA', 'Nurse': 'NA', 'Other': 'NA', 'Participant': 'CHI', 'Partner': 'NA', 'PlayRole': 'NA', 'Playmate': 'OCH', 'Relative': 'NA', 'Sibling': 'OCH', 'Sister': 'OCH', 'Speaker': 'NA', 'Student': 'NA', 'Target_Adult': 'NA', 'Target_Child': 'CHI', 'Teacher': 'NA', 'Teenager': 'NA', 'Text': 'NA', 'Uncertain': 'NA', 'Unidentified': 'NA', 'Visitor': 'NA'}

THREAD_SAFE = False

static **convert**(*filename: str*, *filter=None*) → pandas.core.frame.DataFrame

static **role_to_addressee**(*role*)

class ChildProject.converters.**CsvConverter**
Bases: *ChildProject.converters.AnnotationConverter*

FORMAT = 'csv'

static **convert**(*filename: str*, *filter=''*) → pandas.core.frame.DataFrame

class ChildProject.converters.**EafConverter**
Bases: *ChildProject.converters.AnnotationConverter*

FORMAT = 'eaf'

static **convert**(*filename: str*, *filter=None*) → pandas.core.frame.DataFrame

class ChildProject.converters.**ItsConverter**
Bases: *ChildProject.converters.AnnotationConverter*

FORMAT = 'its'

SPEAKER_TYPE_TRANSLATION = {'CHN': 'CHI', 'CXN': 'OCH', 'FAN': 'FEM', 'MAN': 'MAL'}

static **convert**(*filename: str*, *recording_num: Optional[int] = None*) → pandas.core.frame.DataFrame

class ChildProject.converters.**TextGridConverter**
Bases: *ChildProject.converters.AnnotationConverter*

FORMAT = 'TextGrid'

static convert(*filename: str*, *filter=None*) → pandas.core.frame.DataFrame

class ChildProject.converters.**VcmConverter**

Bases: *ChildProject.converters.AnnotationConverter*

FORMAT = 'vcm_rttm'

SPEAKER_TYPE_TRANSLATION = {'CHI': 'OCH', 'CNS': 'CHI', 'CRY': 'CHI', 'FEM': 'FEM', 'MAL': 'MAL', 'NCS': 'CHI'}

VCM_TRANSLATION = {'CNS': 'C', 'CRY': 'Y', 'NCS': 'N', 'OTH': 'J'}

static convert(*filename: str*, *source_file: str = ''*) → pandas.core.frame.DataFrame

class ChildProject.converters.**VtcConverter**

Bases: *ChildProject.converters.AnnotationConverter*

FORMAT = 'vtc_rttm'

SPEAKER_TYPE_TRANSLATION = {'CHI': 'OCH', 'FEM': 'FEM', 'KCHI': 'CHI', 'MAL': 'MAL'}

static convert(*filename: str*, *source_file: str = ''*) → pandas.core.frame.DataFrame

19.1.6 ChildProject.metrics module

ChildProject.metrics.**conf_matrix**(*rows_grid*, *columns_grid*)

compute the confusion matrix (as counts) from grids of active classes.

See *ChildProject.metrics.segments_to_grid()* for a description of grids.

Parameters

- **rows_grid** (*numpy.array*) – the grid corresponding to the rows of the confusion matrix.
- **columns_grid** (*numpy.array*) – the grid corresponding to the columns of the confusion matrix.
- **categories** (*list of strings*) – the labels corresponding to each class

Returns a square numpy array of counts

Return type numpy.array

ChildProject.metrics.**gamma**(*segments: pandas.core.frame.DataFrame*, *column: str*, *alpha: float = 1*, *beta: float = 1*, *precision_level: float = 0.05*) → float

Compute Mathet et al. gamma agreement on *segments*.

The gamma measure evaluates the reliability of both the segmentation and the categorization simultaneously; a extensive description of the method and its parameters can be found in Mathet et al., 2015 (doi:10.1162/COLI_a_00227)

This function uses the pyagreement-agreement package by Titeux et al.

Parameters

- **segments** (*pd.DataFrame*) – input segments dataframe (see *Annotations format* for the dataframe format)
- **column** (*str*) – name of the categorical column of the segments to consider, e.g. 'speaker_type'
- **alpha** (*float, optional*) – gamma agreement time alignment weight, defaults to 1
- **beta** (*float, optional*) – gamma agreement categorical weight, defaults to 1

- **`precision_level`** (*`float, optional`*) – level of precision (see pygamma-agreement's documentation), defaults to 0.05

Returns gamma agreement

Return type float

`ChildProject.metrics.`**`grid_to_vector`**(*grid*, *categories*)

Transform a grid of active classes into a vector of labels. In case several classes are active at time `i`, the label is set to 'overlap'.

See *`ChildProject.metrics.segments_to_grid()`* for a description of grids.

Parameters

- **`grid`** (*`numpy.array`*) – a NumPy array of shape `(n, len(categories))`
- **`categories`** (*`list`*) – the list of categories

Returns the vector of labels of length `n` (e.g. `np.array([none FEM FEM FEM overlap overlap CHI])`)

Return type numpy.array

`ChildProject.metrics.`**`pyannote_metric`**(*segments: pandas.core.frame.DataFrame*, *reference: str*, *hypothesis: str*, *metric*, *column: str*)

`ChildProject.metrics.`**`segments_to_annotation`**(*segments: pandas.core.frame.DataFrame*, *column: str*)

Transform a dataframe of annotation segments into a pyannote.core.Annotation object

Parameters

- **`segments`** (*`pd.DataFrame`*) – a dataframe of input segments. It should at least have the following columns: `segment_onset`, `segment_offset` and `column`.
- **`column`** (*`str`*) – the name of the column in `segments` that should be used for the values of the annotations (e.g. speaker_type).

Returns the pyannote.core.Annotation object.

Return type pyannote.core.Annotation

`ChildProject.metrics.`**`segments_to_grid`**(*segments: pandas.core.frame.DataFrame*, *range_onset: int*, *range_offset: int*, *timescale: int*, *column: str*, *categories: list*, *none=True*, *overlap=False*) → float

Transform a dataframe of annotation segments into a 2d matrix representing the indicator function of each of the `categories` across time.

Each row of the matrix corresponds to a unit of time of length `timescale` (in milliseconds), ranging from `range_onset` to `range_offset`; each column corresponds to one of the `categories` provided, plus two special columns (overlap and none).

The value of the cell `ij` of the output matrix is set to 1 if the class `j` is active at time `i`, 0 otherwise.

If *overlap* is True, an additional column is appended to the grid, which set to 1 if more than two classes are active at time `i`.

If *none* is set to True, an additional column is appended to the grid, which is set to one if none of the classes are active at time `i`.

The shape of the output matrix is therefore `((range_offset-range_onset)/timescale, len(categories) + n)`, where n = 2 if both *overlap* and *none* are True, 1 if one of them is True, and 0 otherwise.

The fraction of time a class `j` is active can therefore be calculated as `np.mean(grid, axis = 0)[j]`

Parameters

- **segments** (*pd.DataFrame*) – a dataframe of input segments. It should at least have the following columns: `segment_onset`, `segment_offset` and `column`.
- **range_onset** (*int*) – timestamp of the beginning of the range to consider (in milliseconds)
- **range_offset** (*int*) – timestamp of the end of the range to consider (in milliseconds)
- **timescale** (*int*) – length of each time unit (in milliseconds)
- **column** (*str*) – the name of the column in `segments` that should be used for the values of the annotations (e.g. speaker_type).
- **categories** (*list*) – the list of categories
- **none** (*bool*) – append a 'none' column, default True
- **overlap** (*bool*) – append an overlap column, default False

Returns the output grid

Return type numpy.array

`ChildProject.metrics.`**`vectors_to_annotation_task`**(**args*, *drop: List[str] = []*)

transform vectors of labels into a nltk AnnotationTask object.

Parameters

- ***args** – vector of labels for each annotator; add one argument per annotator.
- **drop** (*List[str]*) – list of labels that should be ignored

Returns the AnnotationTask object

Return type nltk.metrics.agreement.AnnotationTask

19.1.7 ChildProject.projects module

`class ChildProject.projects.`**`ChildProject`**(*path: str*, *enforce_dtypes: bool = False*, *ignore_discarded: bool = False*)

Bases: `object`

ChildProject instance This class is a representation of a ChildProject dataset

Constructor parameters:

Parameters

- **path** (*str*) – path to the root of the dataset.
- **enforce_dtypes** (*bool, optional*) – enforce dtypes on children/recordings dataframes, defaults to False
- **ignore_discarded** (*bool, optional*) – ignore entries such that discard=1, defaults to False

Attributes: :param path: path to the root of the dataset. :type path: str :param recordings: pandas dataframe representation of this dataset metadata/recordings.csv :type recordings: class:*pd.DataFrame* :param children: pandas dataframe representation of this dataset metadata/children.csv :type children: class:*pd.DataFrame*

`CHILDREN_COLUMNS = [IndexColumn(name = experiment), IndexColumn(name = child_id), IndexColumn(name = child_dob), IndexColumn(name = location_id), IndexColumn(name = child_sex), IndexColumn(name = language), IndexColumn(name = languages), IndexColumn(name = mat_ed), IndexColumn(name = fat_ed), IndexColumn(name = car_ed), IndexColumn(name = monoling), IndexColumn(name = monoling_criterion), IndexColumn(name = normative), IndexColumn(name = normative_criterion), IndexColumn(name = mother_id), IndexColumn(name = father_id), IndexColumn(name = order_of_birth), IndexColumn(name = n_of_siblings), IndexColumn(name = household_size), IndexColumn(name = dob_criterion), IndexColumn(name = dob_accuracy), IndexColumn(name = discard)]`

`CONVERTED_RECORDINGS = 'recordings/converted'`

`DOCUMENTATION_COLUMNS = [IndexColumn(name = variable), IndexColumn(name = description), IndexColumn(name = values), IndexColumn(name = scope), IndexColumn(name = annotation_set)]`

`PROJECT_FOLDERS = ['recordings', 'annotations', 'metadata', 'doc', 'scripts']`

`RAW_RECORDINGS = 'recordings/raw'`

`RECORDINGS_COLUMNS = [IndexColumn(name = experiment), IndexColumn(name = child_id), IndexColumn(name = date_iso), IndexColumn(name = start_time), IndexColumn(name = recording_device_type), IndexColumn(name = recording_filename), IndexColumn(name = duration), IndexColumn(name = session_id), IndexColumn(name = session_offset), IndexColumn(name = recording_device_id), IndexColumn(name = experimenter), IndexColumn(name = location_id), IndexColumn(name = its_filename), IndexColumn(name = upl_filename), IndexColumn(name = trs_filename), IndexColumn(name = lena_id), IndexColumn(name = lena_recording_num), IndexColumn(name = might_feature_gaps), IndexColumn(name = start_time_accuracy), IndexColumn(name = noisy_setting), IndexColumn(name = notes), IndexColumn(name = discard)]`

`REQUIRED_DIRECTORIES = ['recordings', 'extra']`

`accumulate_metadata`(*table: str*, *df: pandas.core.frame.DataFrame*, *columns: list*, *merge_column: str*, *verbose=False*) → pandas.core.frame.DataFrame

`compute_recordings_duration`(*profile: Optional[str] = None*) → pandas.core.frame.DataFrame
compute recordings duration

Parameters `profile` (`str, optional`) – name of the profile of recordings to compute the duration from. If None, raw recordings are used. defaults to None

Returns dataframe of the recordings, with an additional/updated duration columns.

Return type pd.DataFrame

`get_converted_recording_filename`(*profile: str*, *recording_filename: str*) → str
retrieve the converted filename of a recording under a given `profile`, from its original filename.

Parameters

- `profile` (*str*) – recording profile
- `recording_filename` (*str*) – original recording filename, as indexed in the metadata

Returns corresponding converted filename of the recording under this profile

Return type str

`get_recording_path`(*recording_filename: str*, *profile: Optional[str] = None*) → str
return the path to a recording

Parameters

- **recording_filename** (*str*) – recording filename, as in the metadata
- **profile** (*str, optional*) – name of the conversion profile, defaults to None

Returns path to the recording

Return type str

get_recordings_from_list(*recordings: list*, *profile: Optional[str] = None*) → pandas.core.frame.DataFrame

Recover recordings metadata from a list of recordings or path to recordings.

Parameters **recordings** (*list*) – list of recording names or paths

Returns matching recordings

Return type pd.DataFrame

read(*verbose=False*)

Read the metadata

read_documentation() → pandas.core.frame.DataFrame

recording_from_path(*path: str*, *profile: Optional[str] = None*) → str

validate(*ignore_recordings: bool = False*, *profile: Optional[str] = None*) → tuple

Validate a dataset, returning all errors and warnings.

Parameters **ignore_recordings** (*bool, optional*) – if True, no errors will be returned for missing recordings.

Returns A tuple containing the list of errors, and the list of warnings.

Return type a tuple of two lists

19.1.8 ChildProject.tables module

class ChildProject.tables.**IndexColumn**(*name=''*, *description=''*, *required=False*, *regex=None*, *filename=False*, *datetime=None*, *function=None*, *choices=None*, *dtype=None*, *unique=False*, *generated=False*)

Bases: `object`

class ChildProject.tables.**IndexTable**(*name*, *path=None*, *columns=[]*, *enforce_dtypes: bool = False*)

Bases: `object`

msg(*text*)

read()

validate()

exception ChildProject.tables.**MissingColumnsException**(*name: str*, *missing: Set*)

Bases: `Exception`

ChildProject.tables.**assert_columns_presence**(*name: str*, *df: pandas.core.frame.DataFrame*, *columns: Union[Set, List]*)

ChildProject.tables.**assert_dataframe**(*name: str*, *df: pandas.core.frame.DataFrame*, *not_empty: bool = False*)

ChildProject.tables.**is_boolean**(*x*)

19.1.9 ChildProject.utils module

class ChildProject.utils.**Segment**(*start*, *stop*)

Bases: object

length()

ChildProject.utils.**get_audio_duration**(*filename*)

ChildProject.utils.**intersect_ranges**(*xs*, *ys*)

ChildProject.utils.**path_is_parent**(*parent_path: str*, *child_path: str*)

19.1.10 Module contents

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX