subaligner

Release 0.3.4

Xi Bai

Aug 25, 2023

CONTENTS

Given an out-of-sync subtitle file along with a piece of audiovisual content carrying speeches described by it, Subaligner provides a one-stop solution on automatic subtitle synchronisation and translation with pretrained deep neural networks , forced alignments and transformers. In essence, aligning subtitles is a dual-stage process with a Bidirectional Long Short-Term Memory network trained upfront. Subaligner helps subtitlers not only in preprocessing raw subtitle materials (outcome from stenographers or STT workflow, etc.) but also in gaining quality control over their work within subtitle post-production. This tool also tolerates errors that occurred in live subtitles which sometimes do not completely or correctly represent what people actually spoke in the companion audiovisual content.

Subligner has been shipped with a command-line interface which helps users to conduct various tasks around subtitle synchronisation and multilingual translation without writing any code. Application programming interfaces are also provided to developers wanting to perform those tasks programmatically. Moreover, with existing audiovisual and in-sync subtitle files at hand, advanced users can train their own synchronisers with a single command and zero setup. A handful of subtitle formats are supported and can be converted from one to another either during synchronisation and translation or on on-demand.

Even without any subtitles available beforehand, Subaligner provides transcription by utilising SOTA Large Language Models (LLMs). This pipeline, combined with translation, can generate near ready-to-use subtitles of increasingly higher quality in various languages and formats which cater to your preferences, thanks to those models continually advancing over time.

Subligner supports the following subtitle formats: SubRip, TTML, WebVTT, (Advanced) SubStation Alpha, MicroDVD, MPL2, TMP, EBU STL, SAMI, SCC and SBV. The source code can be found on GitHub: subaligner.

CHAPTER

ONE

INSTALLATION

Install necessary dependencies:

```
$ apt-get install ffmpeg
```

or:

```
$ brew install ffmpeg
```

§ You may also need to install HomeBrew.

Install Subaligner via PyPI (pre-emptive NumPy):

```
$ pip install -U pip && pip install -U setuptools
$ pip install subaligner
```

Install dependencies for enabling translation:

```
$ pip install 'subaligner[llm]'
```

Pre-install additional dependencies before installing subaligner[stretch] or subaligner[dev]:

```
$ apt-get install espeak libespeak1 libespeak-dev espeak-data
```

or:

```
$ brew install espeak
```

Install dependencies for enabling forced alignment:

```
$ pip install 'subaligner[stretch]'
```

Install dependencies for setting up the development environment:

```
$ pip install 'subaligner[dev]'
```

Install all supported features:

```
$ pip install 'subaligner[harmony]'
```

Install Subaligner via pipx:

```
$ pipx install subaligner
$ pipx install 'subaligner[stretch]'
$ pipx install 'subaligner[dev]'
```

Install from GitHub via Pipenv:

```
$ pipenv install subaligner
$ pipenv install 'subaligner[stretch]'
$ pipenv install 'subaligner[dev]'
```

Container Support:

```
$ docker run -v `pwd`:`pwd` -w `pwd` -it baxtree/subaligner bash
```

Users may prefer using a containerised environment over installing everything locally. The following builds are available on dockerhub for several Linux distributions: CentOS 7 (latest and VERSION.el7), CentOS 8 (VERSION.el8), Ubuntu 18 (VERSION.u18), Ubuntu 20 (VERSION.u20), Debian 10 (VERSION.deb10), Fedora 31 (VERSION.fed31) and ArchLinux (VERSION.arch).

You can also download the latest release on GitHub and follow the steps down below to create a virtual environment and set up all the dependencies:

Install Subaligner from source:

```
$ git clone git@github.com:baxtree/subaligner.git && cd subaligner
$ pip install -U pip && pip install -U setuptools
$ python setup.py install
```

Subaligner CLI should be on your PATH now:

```
(.venv) $ subaligner --help
(.venv) $ subaligner_1pass --help # shortcut for "subaligner -m single"
(.venv) $ subaligner_2pass --help # shortcut for "subaligner -m dual"
(.venv) $ subaligner_batch --help
(.venv) $ subaligner_convert --help
(.venv) $ subaligner_train --help
(.venv) $ subaligner_tune --help
```

On Windows:

```
docker pull baxtree/subaligner
docker run -v "/d/media":/media -w "/media" -it baxtree/subaligner bash
```

Assuming that your media assets are stored under “d:\media”, open built-in command prompt, PowerShell, or Windows Terminal and run the above. Docker Desktop is the only option at present for Windows users.

CHAPTER

TWO

USAGE

Subaligner provides two ways of aligning subtitles: single-stage alignment and dual-stage alignment. The former way has lower latency and shifts all subtitle segments globally. The latter way has higher latency and shifts the segments individually with an option of stretching each segment. Multilingual translation on subtitles can be achieved together with the alignment in one go or separately (see in *Advanced Usage*).

With no subtitles in your hand beforehand, Subligner's transcribe mode utilises Large Language Models (LLMs) to transcribe audiovisual content and generates subtitles in various formats which suit your needs.

Make sure you have got the virtual environment activated upfront.

Single-stage alignment (high-level shift with lower latency):

```
(.venv) $ subaligner -m single -v video.mp4 -s subtitle.srt
(.venv) $ subaligner -m single -v https://example.org/video.mp4 -s https://example.
↪org/subtitle.srt -o subtitle_aligned.srt
```

Dual-stage alignment (low-level shift with higher latency):

```
(.venv) $ subaligner -m dual -v video.mp4 -s subtitle.srt
(.venv) $ subaligner -m dual -v https://example.org/video.mp4 -s https://example.org/
↪subtitle.srt -o subtitle_aligned.srt
```

Generate subtitles by transcribing audiovisual files:

```
(.venv) $ subaligner -m transcribe -v video.mp4 -ml eng -mr whisper -mf small -o␣
↪subtitle_aligned.srt
(.venv) $ subaligner -m transcribe -v video.mp4 -ml zho -mr whisper -mf medium -o␣
↪subtitle_aligned.srt
```

Alignment on segmented plain texts (double newlines as the delimiter):

```
(.venv) $ subaligner -m script -v test.mp4 -s subtitle.txt -o subtitle_aligned.srt
(.venv) $ subaligner -m script -v https://example.com/video.mp4 -s https://example.
↪com/subtitle.txt -o subtitle_aligned.srt
```

Alignment on multiple subtitles against the single media file:

```
(.venv) $ subaligner -m script -v test.mp4 -s subtitle_lang_1.txt -s subtitle_lang_2.
↪txt
(.venv) $ subaligner -m script -v test.mp4 -s subtitle_lang_1.txt subtitle_lang_2.txt
```

Alignment on embedded subtitles:

```
(.venv) $ subaligner -m single -v video.mkv -s embedded:stream_index=0 -o subtitle_
↪aligned.srt
(.venv) $ subaligner -m dual -v video.mkv -s embedded:stream_index=0 -o subtitle_
↪aligned.srt
```

Translative alignment with the ISO 639-3 language code pair (src,tgt):

```
(.venv) $ subaligner --languages
(.venv) $ subaligner -m single -v video.mp4 -s subtitle.srt -t src,tgt
(.venv) $ subaligner -m dual -v video.mp4 -s subtitle.srt -t src,tgt
(.venv) $ subaligner -m script -v test.mp4 -s subtitle.txt -o subtitle_aligned.srt -t␣
↪src,tgt
(.venv) $ subaligner -m dual -v video.mp4 -tr helsinki-nlp -o subtitle_aligned.srt -t␣
↪src,tgt
(.venv) $ subaligner -m dual -v video.mp4 -tr facebook-mbart -tf large -o subtitle_
↪aligned.srt -t src,tgt
(.venv) $ subaligner -m dual -v video.mp4 -tr whisper -tf small -o subtitle_aligned.
↪srt -t src,eng
```

Transcribe audiovisual files and generate translated subtitles:

```
(.venv) $ subaligner -m transcribe -v video.mp4 -ml src -mr whisper -mf small -tr␣
↪helsinki-nlp -o subtitle_aligned.srt -t src,tgt
```

Shift subtitle manually by offset in seconds:

```
(.venv) $ subaligner -m shift --subtitle_path subtitle.srt -os 5.5
(.venv) $ subaligner -m shift --subtitle_path subtitle.srt -os -5.5 -o subtitle_
↪shifted.srt
```

Run batch alignment against directories:

```
(.venv) $ subaligner_batch -m single -vd videos/ -sd subtitles/ -od aligned_subtitles/
(.venv) $ subaligner_batch -m dual -vd videos/ -sd subtitles/ -od aligned_subtitles/
(.venv) $ subaligner_batch -m dual -vd videos/ -sd subtitles/ -od aligned_subtitles/ -
↪of ttml
```

Run alignments with the docker image:

```
$ docker pull baxtree/subaligner
$ docker run -v `pwd`:`pwd` -w `pwd` -it baxtree/subaligner subaligner -m single -v␣
↪video.mp4 -s subtitle.srt
$ docker run -v `pwd`:`pwd` -w `pwd` -it baxtree/subaligner subaligner -m dual -v␣
↪video.mp4 -s subtitle.srt
$ docker run -it baxtree/subaligner subaligner -m single -v https://example.com/video.
↪mp4 -s https://example.com/subtitle.srt -o subtitle_aligned.srt
$ docker run -it baxtree/subaligner subaligner -m dual -v https://example.com/video.
↪mp4 -s https://example.com/subtitle.srt -o subtitle_aligned.srt
```

Run alignments with pipx:

```
$ pipx run subaligner -m single -v video.mp4 -s subtitle.srt
$ pipx run subaligner -m dual -v video.mp4 -s subtitle.srt
```

Run the module as a script:

```
$ python -m subaligner -m single -v video.mp4 -s subtitle.srt
$ python -m subaligner -m dual -v video.mp4 -s subtitle.srt
```

Currently the stretching is experimental and make sure subaligner[stretch] is installed before switching it on with *-so* or *–stretch_on* as shown below.

Switch on stretching when aligning subtitles:

```
(.venv) $ subaligner -m dual -v video.mp4 -s subtitle.srt -so
```

Save the aligned subtitle to a specific location:

```
(.venv) $ subaligner -m dual -v video.mp4 -s subtitle.srt -o /path/to/the/output/
↪subtitle.srt
```

On Windows:

```
docker run -v "/d/media":/media -w "/media" -it baxtree/subaligner COMMAND
```

The aforementioned commands can be run with Docker Desktop on Windows 10.

Re-configure FFmpeg/Libav path:

```
(.venv) $ export FFMPEG_PATH=/path/to/ffmpeg
(.venv) $ subaligner -m dual -v video.mp4 -s subtitle.srt
or
(.venv) $ FFMPEG_PATH=/path/to/ffmpeg subaligner -m dual -v video.mp4 -s subtitle.srt
or when using `Libav<https://libav.org/>`_
(.venv) $ FFMPEG_PATH=/path/to/avconv subaligner -m dual -v video.mp4 -s subtitle.srt
```

The lower case “ffmpeg_path” is also supported.

CHAPTER

THREE

ADVANCED USAGE

You can train a new model with your own audiovisual files and in-sync subtitle files using Subaligner CLI. Thereafter, the model can be imported and used for synchronising out-of-sync subtitles.

Start fresh training:

```
$ subaligner_train -vd av_directory -sd subtitle_directory -tod training_output_
↪directory
```

Make sure each subtitle file and its companion audiovisual file are sharing the same base filename, e.g., "awesome.mp4" and "awesome.srt" share the base filename "awesome". Then split them into two separate folders, e.g., av_directory and subtitle_directory, which are passed in with *-vd* or *–video_directory* and *-sd* or *–subtitle_directory* , respectively. You need to also specify an output directory with *-tod* or *–training_output_directory* and it will hold the results after training is finished and make sure it is writable to Subaligner, e.g., training_output_directory.

Resume training:

```
$ subaligner_train -vd av_directory -sd subtitle_directory -tod training_output_
↪directory -e 200 -r
```

Training over a large dataset is usually an expensive process and time consuming. You can stop the training and resume it with *-r* or *–resume* at another convenient time to enhance an existing model stored in the aforementioned training output directory. Note that the number of epochs you pass in with *-e* or *–epochs* needs to be greater than the number of epochs already completed in the past. If the number is forgotten, you can pass in *-dde* or *–display_done_epochs* to recall it.

Display completed epochs:

```
$ subaligner_train -dde -tod training_output_directory
```

Also note that on training resumption, av_directory and subtitle_directory will be ignored due to the reuse of feature embedding by default.

Reuse embeddings:

```
$ subaligner_train -utd -tod training_output_directory
```

Embeddings extracted from your media files can be reused with *-utd* or *–use_training_dump*. With that flag on, you can train a new model of another kind (instead of re-using the same model on training resumption) without going through the feature embedding process, which could take quite long to finish for a large dataset so as to be unnecessary if there is no change on it.

Ignore sound effects:

```
$ subaligner_train -vd av_directory -sd subtitle_directory -tod training_output_
↪directory --sound_effect_start_marker "(" --sound_effect_end_marker ")"
```

It is not uncommon that subtitles sometimes contain sound effects (e.g., "BARK", "(applause)" and "[MUSIC]", etc.). For limited training data sets and not sophisticated enough network architectures, the model usually cannot capture all the sound effects very well. To filter out sound effect subtitles and only preserve the vocal ones, you can pass in *-sesm* or *–sound_effect_start_marker* and/or *seem* or *–sound_effect_end_marker* with strings which will be used by subaligner for finding sound effects and ignoring them within the training process. For example, the above exemplary command will treat any strings starting with "(" and ending with ")" as sound effects.

Train with embedded subtitles:

```
$ subaligner_train -vd av_directory -ess embedded:stream_index=0,file_extension=srt -
↪tod training_output_directory
```

If your audiovisual files all contain embedded subtitles or teletexts of the same format and have been encoded in the same fashion, *-sd* or *–subtitle_directory* can be omitted and subtitles will be extracted based on the specified subtitle selector. For instance, "embedded:stream_index=0,file_extension=srt" can be passed in with *-ess* or *–embedded_subtitle_selector* and indicates that the embedded subtitle is located at stream 0 (the first subtitle track) and will be extracted as a SubRip file pre and post synchronisation, while "embedded:page_num=888,file_extension=srt" means the teletext is located on page 888. When *-sd* or *–subtitle_directory* is present, make sure the folder passed in is empty.

Run alignments after training:

```
$ subaligner -m single -v video.mp4 -s subtitle.srt -tod training_output_directory
$ subaligner -m dual -v video.mp4 -s subtitle.srt -tod training_output_directory
```

To apply your trained model to subtitle alignment, pass in the training_output_directory containing training results as shown above with *-tod* or *–training_output_directory*.

Hyperparameters:

```
-bs BATCH_SIZE, --batch_size BATCH_SIZE
                    Number of 32ms samples at each training step
-do DROPOUT, --dropout DROPOUT
    Dropout rate between 0 and 1 used at the end of each intermediate hidden layer
-e EPOCHS, --epochs EPOCHS
    Total training epochs
-p PATIENCE, --patience PATIENCE
    Number of epochs with no improvement after which training will be stopped
-fhs FRONT_HIDDEN_SIZE, --front_hidden_size FRONT_HIDDEN_SIZE
    Number of neurons in the front LSTM or Conv1D layer
-bhs BACK_HIDDEN_SIZE, --back_hidden_size BACK_HIDDEN_SIZE
    Comma-separated numbers of neurons in the back Dense layers
-lr LEARNING_RATE, --learning_rate LEARNING_RATE
    Learning rate of the optimiser
-nt {lstm,bi_lstm,conv_1d}, --network_type {lstm,bi_lstm,conv_1d}
    Network type
-vs VALIDATION_SPLIT, --validation_split VALIDATION_SPLIT
    Fraction between 0 and 1 of the training data to be used as validation data
-o {adadelta,adagrad,adam,adamax,ftrl,nadam,rmsprop,sgd}, --optimizer {adadelta,
↪adagrad,adam,adamax,ftrl,nadam,rmsprop,sgd}
    TensorFlow optimizer
```

You can pass in the above flags to manually change hyperparameters for each training cycle. Alternatively, you can let Subaligner tune hyperparameters automatically and the how-to is shown below.

Hyperparameters tuning:

```
$ subaligner_tune -vd av_directory -sd subtitle_directory -tod training_output_
↪directory
```

Subaligner has used the Tree-structured Parzen Estimator Approach (TPE) to automatically run trails on different settings of hyper-parameter values and recommend the best one. You can pass in the following flags to customise the configuration on tuning:

Optional custom flags:

```
-ept EPOCHS_PER_TRAIL, --epochs_per_trail EPOCHS_PER_TRAIL
    Number of training epochs for each trial
-t TRAILS, --trails TRAILS
    Number of tuning trials
-nt {lstm,bi_lstm,conv_1d}, --network_type {lstm,bi_lstm,conv_1d}
    Network type
-utd, --use_training_dump
    Use training dump instead of files in the video or subtitle directory
```

Convert the subtitle to another format:

```
$ subaligner_convert -i subtitle.srt -o subtitle.vtt
```

Convert the subtitle to another format and translate:

```
$ subaligner_convert --languages
$ subaligner_convert -i subtitle_en.srt -o subtitle_zh.vtt -t eng,zho
```

Translate the subtitle without changing the format:

```
$ subaligner_convert --languages
$ subaligner_convert -i subtitle_en.srt -o subtitle_es.srt -t eng,spa
```

For output subtitles like MicroDVD relying on the frame rate, its value needs to be passed in with *-fr* or *–frame_rate*.

On Windows:

```
docker run -v "/d/media":/media -w "/media" -it baxtree/subaligner COMMAND
```

The aforementioned commands can be run with Docker Desktop on Windows 10.

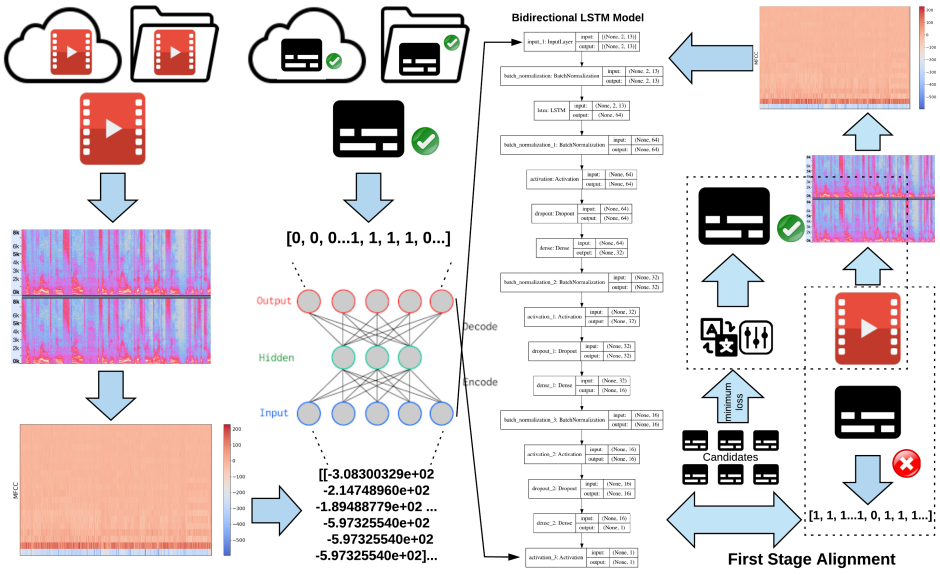
CHAPTER

FOUR

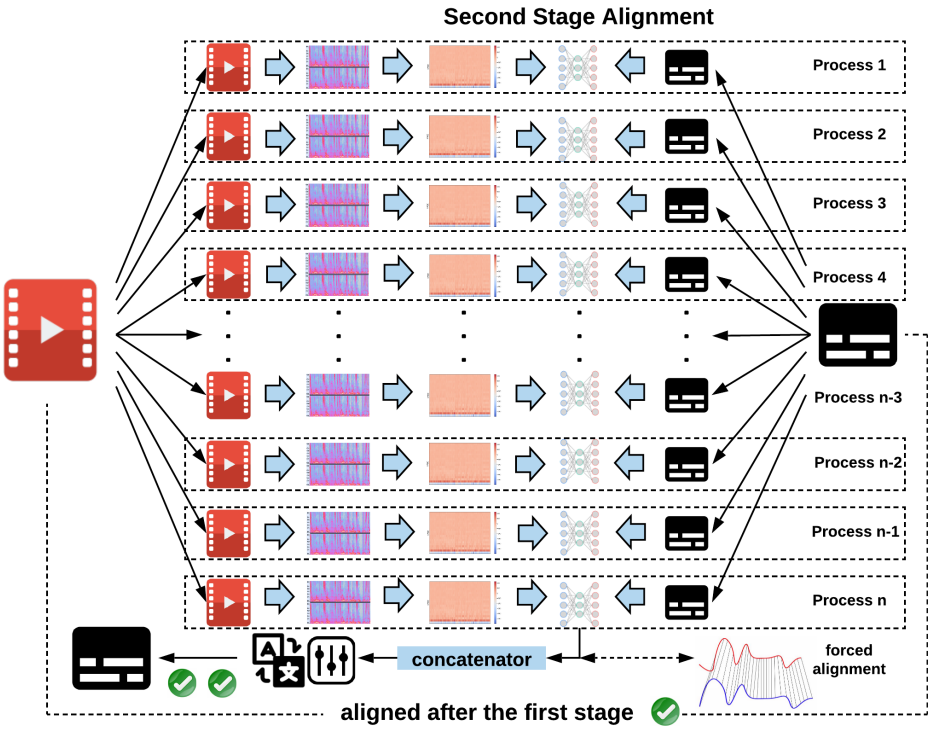
ANATOMY

Under the hood, a model has been trained with synchronised video and subtitle pairs and later used for predicating shifting offsets and directions under the guidance of a dual-stage aligning approach.

The following figure depicts the primary workflow of the first-stage subtitle alignment. It also includes upfront network training and later-on subtitle shifting. The data set used for training contains pairs of a video clip and a subtitle file with decent start and end time codes. Mel-Frequency Cepstral Coefficients are extracted in parallel and fed into a carefully designed DNN incorporating LSTM layers. For subtitle shifting, an out-of-sync subtitle file and a target video clip are fed in as input and the output can be either time offsets by which the subtitle should be shifted or a ready-to-playback subtitle file with calibrated start and end time. Notably, the shifting illustrated here leads to the completion of the first stage alignment (global shifting).



At the second stage, the target video clip and the globally shifted subtitle file will be broken down into evenly timed chunks respectively, each pair of which will be fed into the DNN and aligned in parallel (regional shifting) as shown in the following figure. And the concatenated subtitle chunks result in the final output.


Second Stage Alignment
Process 1
Process 2
Process 3
Process 4
Process n-3
Process n-2
Process n-1
Process n
concatenator
forced alignment
aligned after the first stage

CHAPTER

FIVE

TEST

Run unit tests:

```
$ make test
```

Run test coverage:

```
$ make coverage
```

Run tests with target Python versions:

```
$ make test-all
```

Run integration tests:

```
$ make test-int
```

CHAPTER

SIX

ACKNOWLEDGEMENT

This tool wouldn't be possible without the following packages:

```
- `librosa <https://librosa.github.io/librosa/>`_
- `tensorflow <https://www.tensorflow.org/>`_
- `scikit-learn <https://scikit-learn.org>`_
- `pycaption <https://pycaption.readthedocs.io>`_
- `pysrt <https://github.com/byroot/pysrt>`_
- `pysubs2 <https://github.com/tkarabela/pysubs2>`_
- `aeneas <https://www.readbeyond.it/aeneas/>`_
- `transformers <https://huggingface.co/transformers/>`_
- `openai-whisper <https://github.com/openai/whisper>`_
```

Thanks to Alan Robinson and Nigel Megitt for their invaluable feedback.

CHAPTER

SEVEN

LICENSE

CHAPTER

EIGHT

API REFERENCE

8.1 subaligner

8.1.1 subaligner package

Subpackages

Submodules

subaligner.embedder module

class subaligner.embedder.**FeatureEmbedder**(*n_mfcc: int = 13*, *frequency: int = 16000*, *hop_len: int = 512*, *step_sample: float = 0.04*, *len_sample: float = 0.075*)

Bases: object

Audio and subtitle feature embedding.

duration_to_position(*seconds: float*) → int

Return the cell position from a time in seconds.

Parameters **{float} -- The duration in seconds.** (*seconds*) –

Returns int – The cell position.

extract_data_and_label_from_audio(*audio_file_path: str*, *subtitle_file_path: Optional[str]*, *subtitles: Optional[pysrt.SubRipFile] = None*, *sound_effect_start_marker: Optional[str] = None*, *sound_effect_end_marker: Optional[str] = None*) → Tuple[numpy.ndarray, numpy.ndarray]

Generate a train dataset from an audio file and its subtitles

Parameters

- **{string} -- The path to the audio file.** (*audio_file_path*) –
- **{string} -- The path to the subtitle file.** (*subtitle_file_path*) –

Keyword Arguments

- **{pysrt.SubRipFile} -- The SubRipFile object (default** (*subtitles*) – {None}).
- **sound_effect_start_marker** – {string} – A string indicating the start of the ignored sound effect (default: {None}).

- **sound_effect_end_marker** – {string} – A string indicating the end of the ignored sound effect (default: {None}).

Returns tuple – The training data and the training lables.

property frequency
Get the sample rate.

Returns int – The sample rate.

get_len_mfcc() → float
Get the number of samples to get LEN_SAMPLE: LEN_SAMPLE/(HOP_LEN/FREQUENCY).

Returns float – The number of samples.

get_step_mfcc() → float
Get the number of samples to get STEP_SAMPLE: STEP_SAMPLE/(HOP_LEN/FREQUENCY).

Returns float – The number of samples.

property hop_len
Get the number of samples per frame.

Returns int – The number of samples per frame.

property len_sample
Get the length in seconds for the input samples.

Returns float – The length in seconds for the input samples.

property n_mfcc
Get the number of MFCC components.

Returns int – The number of MFCC components.

position_to_duration(*position: int*) → float
Return the time in seconds from a cell position.

Parameters **{int} -- The cell position.** (*position*) –

Returns float – The number of seconds.

position_to_time_str(*position: int*) → str
Return the time string from a cell position.

Parameters **{int} -- The cell position.** (*position*) –

Returns 23:20,150).

Return type string – The time string (e.g., 01

property step_sample
The space (in seconds) between the begining of each sample.

Returns float – The space (in seconds) between the begining of each sample.

time_to_position(*pysrt_time: pysrt.SubRipTime*) → int
Return a cell position from timestamp.

Parameters **{pysrt.SubRipTime} -- SubRipTime or coercible.** (*pysrt_time*) –

Returns int – The cell position.

classmethod time_to_sec(*pysrt_time: pysrt.SubRipTime*) → float
Convert timestamp to seconds.

Parameters **{pysrt.SubRipTime} -- SubRipTime or coercible.** (*pysrt_time*) –

Returns float – The number of seconds.

subaligner.exception module

exception subaligner.exception.**NoFrameRateException**
Bases: Exception

An exception raised due to frame rate not found.

exception subaligner.exception.**TerminalException**
Bases: Exception

An exception raised due to unrecoverable failures.

exception subaligner.exception.**TranscriptionException**
Bases: Exception

An exception raised due to transcription failures.

exception subaligner.exception.**TranslationException**
Bases: Exception

An exception raised due to translation failures.

exception subaligner.exception.**UnsupportedFormatException**
Bases: Exception

An exception raised due to unsupported formats.

subaligner.hparam_tuner module

class subaligner.hparam_tuner.**HyperParameterTuner**(*av_file_paths: List[str], subtitle_file_paths: List[str], training_dump_dir: str, num_of_trials: int = 5, tuning_epochs: int = 5, network_type: str = 'lstm', **kwargs*)
Bases: object

Hyperparameter tuning using the Tree of Parzen Estimators algorithm

SEARCH_SPACE = {'back_hidden_size': hyperopt.hp.choice, 'batch_size': hyperopt.pyll.

property hyperparameters

tune_hyperparameters() → None
Tune the hyperparameters

subaligner.hyperparameters module

class subaligner.hyperparameters.**Hyperparameters**

Bases: object

The configuration on hyperparameters used for training

OPTIMIZERS = ['adadelta', 'adagrad', 'adam', 'adamax', 'ftrl', 'nadam', 'rmsprop', 'sg

property back_hidden_size

property batch_size

clone() → *subaligner.hyperparameters.Hyperparameters*

Make a cloned hyperparameters object

Returns Hyperparameters – The cloned Hyperparameters object.

property dropout

property epochs

property es_min_delta

property es_mode

property es_patience

classmethod from_file(*file_path: str*) → *subaligner.hyperparameters.Hyperparameters*

Deserialise a file content into a Hyperparameters object

Parameters **{string} -- The path to the file containing hyperparameters.** (*file_path*) –

Returns Hyperparameters – The deserialised Hyperparameters object.

classmethod from_json(*json_str: str*) → *subaligner.hyperparameters.Hyperparameters*

Deserialise JSON string into a Hyperparameters object

Parameters **{string} -- Hyperparameters in JSON.** (*json_str*) –

Returns Hyperparameters – The deserialised Hyperparameters object.

property front_hidden_size

property learning_rate

property loss

property metrics

property monitor

property network_type

property optimizer

to_file(*file_path: str*) → None

Serialise hyperparameters into JSON and save the content to a file

Parameters **{string} -- The path to the file containing saved hyperparameters.** (*file_path*) –

to_json() → str

Serialise hyperparameters into JSON string

Returns string – The serialised hyperparameters in JSON

property validation_split

subaligner.llm module

class subaligner.llm.**FacebookMbartFlavour**(*value*)
Bases: enum.Enum

An enumeration.

LARGE = 'large'

class subaligner.llm.**HelsinkiNLPFlavour**(*value*)
Bases: enum.Enum

An enumeration.

OPUS_MT = 'Helsinki-NLP/opus-mt-{}-{}'

OPUS_MT_TC_BIG = 'Helsinki-NLP/opus-mt-tc-big-{}-{}'

OPUS_TATOEBA = 'Helsinki-NLP/opus-tatoeba-{}-{}'

class subaligner.llm.**TranscriptionRecipe**(*value*)
Bases: enum.Enum

An enumeration.

WHISPER = 'whisper'

class subaligner.llm.**TranslationRecipe**(*value*)
Bases: enum.Enum

An enumeration.

FACEBOOK_MBART = 'facebook-mbart'

HELSINKI_NLP = 'helsinki-nlp'

WHISPER = 'whisper'

class subaligner.llm.**WhisperFlavour**(*value*)
Bases: enum.Enum

An enumeration.

BASE = 'base'

BASE_EN = 'base.en'

LARGE = 'large'

LARGE_V1 = 'large-v1'

LARGE_V2 = 'large-v2'

MEDIUM = 'medium'

MEDIUM_EN = 'medium.en'

SMALL = 'small'

TINY = 'tiny'

TINY_EN = 'tiny.en'

subaligner.logger module

class subaligner.logger.**Logger**(**args*, ***kwargs*)
Bases: object

Common logging.

QUIET = True

VERBOSE = False

get_logger(*name: str*) → logging.Logger

subaligner.media_helper module

class subaligner.media_helper.**MediaHelper**
Bases: object

Utility for processing media assets including audio, video and subtitle files.

AUDIO_FILE_EXTENSION = ['.wav', '.aac']

FFMPEG_BIN = 'ffmpeg'

extract_audio(*video_file_path*, *decompress: bool = False*, *freq: int = 16000*) → str
Extract audio track from the video file and save it to a WAV file.

Parameters **{string} -- The input video file path.** (*video_file_path*) –

Keyword Arguments

- **{bool} -- Extract WAV if True otherwise extract AAC (default** (*decompress*) – {False}).
- **{int} -- The audio sample frequency (default** (*freq*) – {16000}).

Returns string – The file path of the extracted audio.

extract_audio_from_start_to_end(*audio_file_path: str*, *start: str*, *end: Optional[str] = None*) → Tuple[str, Optional[float]]
Extract audio based on the start time and the end time and save it to a temporary file.

Parameters

- **{string} -- The path of the audio file.** (*audio_file_path*) –
- **{string} -- The start time (e.g.** (*start*) – 00:00,750).
- **00** – 00:00,750).

Keyword Arguments **{string} -- The end time (e.g., 00** (*end*) – 00:10,230) (default: {None}).

Returns tuple – The file path to the extracted audio and its duration.

get_audio_segment_starts_and_ends(*subs: List[pysrt.SubRipItem]*) → Tuple[List[str], List[str], List[pysrt.SubRipFile]]
Group subtitle cues into larger segments in terms of silence gaps.

Parameters **{list} -- A list of SupRip cues.** (*subs*) –

Returns tuple – A list of start times, a list of end times and a list of grouped SubRip files.

get_duration_in_seconds(*start: Optional[str]*, *end: Optional[str]*) → Optional[float]
Get the duration in seconds between a start time and an end time.

Parameters

- **{string} -- The start time (e.g.** (*start*) – 00:00,750).
- **00** – 00:00,750).
- **{string} -- The end time (e.g.** (*end*) – 00:10,230).
- **00** – 00:10,230).

Returns float – The duration in seconds.

get_frame_rate(*file_path: str*) → float
Extract the video frame rate. Will return 25 when input is audio

Parameters **{string} -- The input audiovisual file path.** (*file_path*) –

Returns float – The frame rate

refragment_with_min_duration(*subs: List[pysrt.SubRipItem]*, *minimum_segment_duration: float*) → List[pysrt.SubRipItem]
Re-fragment a list of subtitle cues into new cues each of spans a minimum duration

Parameters

- **{list} -- A list of SupRip cues.** (*subs*) –
- **{float} -- The minimum duration in seconds for each output subtitle cue.** (*minimum_segment_duration*) –

Returns list – A list of new SupRip cues after fragmentation.

subaligner.media_helper.**clear_temp**(*_)

subaligner.network module

class subaligner.network.**Network**(*secret: Optional[object]*, *input_shape: Tuple*, *hyperparameters: subaligner.hyperparameters.Hyperparameters*, *model_path: Optional[str] = None*, *backend: str = 'tensorflow'*)

Bases: object

Network factory creates DNNs. Not thread safe since the session of keras_backend is global. Only factory methods are allowed when generating DNN objects.

BI_LSTM = 'bi_lstm'

CONV_1D = 'conv_1d'

LSTM = 'lstm'

TYPES = ['lstm', 'bi_lstm', 'conv_1d']

fit_and_get_history(*train_data: numpy.ndarray*, *labels: numpy.ndarray*, *model_filepath: str*, *weights_filepath: str*, *logs_dir: str*, *training_log: str*, *resume: bool*) → Tuple[List[float], List[float]]
Fit the training data to the network and save the network model as a HDF file.

Parameters

- **{numpy.array} -- The Numpy array of training data.** (*train_data*) –
- **{numpy.array} -- The Numpy array of training labels.** (*labels*) –
- **{string} -- The model file path.** (*model_filepath*) –
- **{string} -- The weights file path.** (*weights_filepath*) –
- **{string} -- The TensorBoard log file directory.** (*logs_dir*) –
- **{string} -- The path to the log file of epoch results.** (*training_log*) –
- **{bool} -- True to continue with previous training result or False to start a new one (default** (*resume*) – {False}).

Returns tuple – A tuple contains validation losses and validation accuracies.

fit_with_generator(*train_data_raw: numpy.ndarray*, *labels_raw: numpy.ndarray*, *model_filepath: str*, *weights_filepath: str*, *logs_dir: str*, *training_log: str*, *resume: bool*) → Tuple[List[float], List[float]]

Fit the training data to the network and save the network model as a HDF file.

Parameters

- **{list} -- The HDF5 raw training data.** (*train_data_raw*) –
- **{list} -- The HDF5 raw training labels.** (*labels_raw*) –
- **{string} -- The model file path.** (*model_filepath*) –
- **{string} -- The weights file path.** (*weights_filepath*) –
- **{string} -- The TensorBoard log file directory.** (*logs_dir*) –
- **{string} -- The path to the log file of epoch results.** (*training_log*) –
- **{bool} -- True to continue with previous training result or False to start a new one (default** (*resume*) – {False}).

Returns tuple – A tuple contains validation losses and validation accuracies.

classmethod get_from_model(*model_path: str*, *hyperparameters: subaligner.hyperparameters.Hyperparameters*) → *subaligner.network.Network*

Load model into a network object.

Parameters

- **{string} -- The path to the model file.** (*model_path*) –
- **{Hyperparameters} -- A configuration for hyperparameters used for training.** (*hyperparameters*) –

classmethod get_network(*input_shape: Tuple*, *hyperparameters: subaligner.hyperparameters.Hyperparameters*) → *subaligner.network.Network*

Factory method for creating a network.

Parameters

- **{tuple} -- A shape tuple** (*input_shape*) –

- **{Hyperparameters} -- A configuration for hyperparameters used for training.** (*hyperparameters*) –

Returns Network – A constructed network object.

get_predictions(*input_data: numpy.ndarray*, *weights_filepath: str*) → numpy.ndarray
Get a Numpy array of predictions.

Parameters

- **{numpy.ndarray} -- The input data** (*input_data*) –
- **a Numpy array.** (*as*) –
- **{string} -- The weights file path.** (*weights_filepath*) –

Returns numpy.ndarray – The Numpy array of predictions.

property input_shape
Get the input shape of the network.

Returns tuple – The input shape of the network.

property layers
Get the layers of the network.

Returns list – The statck of layers contained by the network

static load_model_and_weights(*model_filepath: str*, *weights_filepath: str*, *hyperparameters:* subaligner.hyperparameters.Hyperparameters) → *subaligner.network.Network*
Load weights to the Network model.

Parameters

- **{string} -- The model file path.** (*model_filepath*) –
- **{string} -- The weights file path.** (*weights_filepath*) –
- **{Hyperparameters} -- A configuration for hyperparameters used for training.** (*hyperparameters*) –

Returns Network – Reconstructed network object.

property n_type
Get the type of the network.

Returns string – The type of the network.

static reset() → None

classmethod save_model_and_weights(*model_filepath: str*, *weights_filepath: str*, *combined_filepath: str*) → None
Combine model and weights and save to a file

Parameters

- **{string} -- The path to the model file.** (*model_filepath*) –
- **{string} -- The path to the weights file.** (*weights_filepath*) –

classmethod simple_fit(*input_shape: Tuple*, *train_data: numpy.ndarray*, *labels: numpy.ndarray*, *hyperparameters:* subaligner.hyperparameters.Hyperparameters) → Tuple[List[float], List[float]]
Fit the training data to the network and save the network model as a HDF file.

Parameters

- **{tuple} -- A shape tuple** (*input_shape*) –
- **{numpy.array} -- The Numpy array of training data.** (*train_data*) –
- **{numpy.array} -- The Numpy array of training labels.** (*labels*) –
- **{Hyperparameters} -- A configuration for hyperparameters used for training.** (*hyperparameters*) –

Returns tuple – A tuple contains validation losses and validation accuracies.

classmethod simple_fit_with_generator(*input_shape: Tuple*, *train_data_raw: numpy.ndarray*, *labels_raw: numpy.ndarray*, *hyperparameters:* subaligner.hyperparameters.Hyperparameters) → Tuple[List[float], List[float]]

Fit the training data to the network and save the network model as a HDF file.

Parameters

- **{tuple} -- A shape tuple** (*input_shape*) –
- **{list} -- The HDF5 raw training data.** (*train_data_raw*) –
- **{list} -- The HDF5 raw training labels.** (*labels_raw*) –
- **{Hyperparameters} -- A configuration for hyperparameters used for training.** (*hyperparameters*) –

Returns tuple – A tuple contains validation losses and validation accuracies.

property summary

Print out the summary of the network.

subaligner.predictor module

class subaligner.predictor.**Predictor**(**args*, ***kwargs*)

Bases: object

Predictor for working out the time to shift subtitles

get_log_loss(*voice_probabilities: numpy.ndarray*, *subs: List[pysrt.SubRipItem]*) → float

Returns a single loss value on voice prediction

Parameters

- **{list} -- A list of probabilities of audio chunks being speech.** (*voice_probabilities*) –
- **{list} -- A list of subtitle segments.** (*subs*) –
- **Returns** – float – The loss value.

get_min_log_loss_and_index(*voice_probabilities: numpy.ndarray*, *subs: pysrt.SubRipFile*) → Tuple[float, int]

Returns the minimum loss value and its shift position after going through all possible shifts. :param voice_probabilities {list} – A list of probabilities of audio chunks being speech.: :param subs {list} – A list of subtitle segments.:

Returns tuple – The minimum loss value and its position.

predict_dual_pass(*video_file_path: str*, *subtitle_file_path: str*, *weights_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/subaligner/checkouts/development/subaligner/mod*, *stretch: bool = False*, *stretch_in_lang: str = 'eng'*, *exit_segfail: bool = False*) → Tuple[List[pysrt.SubRipItem], List[pysrt.SubRipItem], Union[numpy.ndarray, List[float]], Optional[float]]

Predict time to shift with single pass

Arguments: video_file_path {string} – The input video file path. subtitle_file_path {string} – The path to the subtitle file. weights_dir {string} – The the model weights directory. stretch {bool} – True to stretch the subtitle segments (default: {False}) stretch_in_lang {str} – The language used for stretching subtitles (default: {"eng"}). exit_segfail {bool} – True to exit on any segment alignment failures (default: {False})

Returns: tuple – The shifted subtitles, the globally shifted subtitles and the voice probabilities of the original audio.

predict_plain_text(*video_file_path: str*, *subtitle_file_path: str*, *stretch_in_lang: str = 'eng'*) → Tuple

Predict time to shift with plain texts

Arguments: video_file_path {string} – The input video file path. subtitle_file_path {string} – The path to the subtitle file. stretch_in_lang {str} – The language used for stretching subtitles (default: {"eng"}).

Returns: tuple – The shifted subtitles, the audio file path (None) and the voice probabilities of the original audio (None).

predict_single_pass(*video_file_path: str*, *subtitle_file_path: str*, *weights_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/subaligner/checkouts/development/subaligner/h*) → Tuple[List[pysrt.SubRipItem], str, Union[numpy.ndarray, List[float]], Optional[float]]

Predict time to shift with single pass

Parameters

- **{string} -- The input video file path.** (*video_file_path*) –
- **{string} -- The path to the subtitle file.** (*subtitle_file_path*) –
- **{string} -- The the model weights directory.** (*weights_dir*) –

Returns tuple – The shifted subtitles, the audio file path and the voice probabilities of the original audio.

subaligner.singleton module

class subaligner.singleton.**Singleton**

Bases: type

A metaclass that creates a Singleton base class when called.

subaligner.subtitle module

class subaligner.subtitle.**Subtitle**(*secret: object*, *subtitle_file_path: str*, *subtitle_format: str*)

Bases: object

Load a subtitle file into internal data structure

ADVANCED_SSA_EXTENTIONS = ['.ass']

MICRODVD_EXTENSIONS = ['.sub']

MPL2_EXTENSIONS = ['.txt']

SAMI_EXTENSIONS = ['.smi', '.sami']

SBV_EXTENSIONS = ['.sbv']

SCC_EXTENSIONS = ['.scc']

SSA_EXTENTIONS = ['.ssa']

STL_EXTENSIONS = ['.stl']

SUBRIP_EXTENTIONS = ['.srt']

TMP_EXTENSIONS = ['.tmp']

TTML_EXTENSIONS = ['.xml', '.ttml', '.dfxp']

TT_NS = {'tt': 'http://www.w3.org/ns/ttml'}

WEBVTT_EXTENSIONS = ['.vtt']

YT_TRANSCRIPT_EXTENSIONS = ['.ytt']

static export_subtitle(*source_file_path: str*, *subs: List[pysrt.SubRipItem]*, *target_file_path: str*, *frame_rate: float = 25.0*, *encoding: Optional[str] = None*) → None

Export subtitle in the format determined by the file extension.

Parameters

- **{string} -- The path to the original subtitle file.** (*source_file_path*) –
- **{list} -- A list of SubRipItems.** (*subs*) –
- **{string} -- The path to the exported subtitle file.** (*target_file_path*) –
- **{float} -- The frame rate for frame-based subtitle formats {default** (*frame_rate*) – 25.0}.
- **{str} -- The encoding of the exported subtitle file {default** (*encoding*) – None}.

static extract_text(*subtitle_file_path: str*, *delimiter: str = ' '*) → str

Extract plain texts from a subtitle file.

Parameters **{string} -- The path to the subtitle file.** (*subtitle_file_path*) –

Returns {string} – The plain text of subtitle.

classmethod load(*subtitle_file_path: str*) → *subaligner.subtitle.Subtitle*

Load a SubRip or TTML subtitle file based on the file extension.

Parameters **{string} -- The path to the subtitle file.** (*subtitle_file_path*) –

Returns Subtitle – Subtitle object.

classmethod load_ass(*subtitle_file_path: str*) → *subaligner.subtitle.Subtitle*
Load a Advanced SubStation Alpha v4.0+ subtitle file.

Parameters **{string} -- The path to the subtitle file.** (*subtitle_file_path*) –

Returns Subtitle – Subtitle object.

classmethod load_microdvd(*subtitle_file_path: str*) → *subaligner.subtitle.Subtitle*
Load a MicroDVD subtitle file.

Parameters **{string} -- The path to the subtitle file.** (*subtitle_file_path*) –

Returns Subtitle – Subtitle object.

classmethod load_mpl2(*subtitle_file_path: str*) → *subaligner.subtitle.Subtitle*
Load a MPL2 subtitle file.

Parameters **{string} -- The path to the subtitle file.** (*subtitle_file_path*) –

Returns Subtitle – Subtitle object.

classmethod load_sami(*subtitle_file_path: str*) → *subaligner.subtitle.Subtitle*
Load a SAMI subtitle file.

Parameters **{string} -- The path to the subtitle file.** (*subtitle_file_path*) –

Returns Subtitle – Subtitle object.

classmethod load_sbv(*subtitle_file_path: str*) → *subaligner.subtitle.Subtitle*
Load a SubViewer subtitle file.

Parameters **{string} -- The path to the subtitle file.** (*subtitle_file_path*) –

Returns Subtitle – Subtitle object.

classmethod load_scc(*subtitle_file_path: str*) → *subaligner.subtitle.Subtitle*
Load a Scenarist Closed Caption subtitle file.

Parameters **{string} -- The path to the subtitle file.** (*subtitle_file_path*) –

Returns Subtitle – Subtitle object.

classmethod load_ssa(*subtitle_file_path: str*) → *subaligner.subtitle.Subtitle*
Load a SubStation Alpha v4.0 subtitle file.

Parameters **{string} -- The path to the subtitle file.** (*subtitle_file_path*) –

Returns Subtitle – Subtitle object.

classmethod load_stl(*subtitle_file_path: str*) → *subaligner.subtitle.Subtitle*
Load an EBU STL subtitle file.

Parameters **{string} -- The path to the subtitle file.** (*subtitle_file_path*) –

Returns Subtitle – Subtitle object.

classmethod load_subrip(*subtitle_file_path: str*) → *subaligner.subtitle.Subtitle*
Load a SubRip subtitle file.

Parameters **{string} -- The path to the subtitle file.** (*subtitle_file_path*) –

Returns Subtitle – Subtitle object.

classmethod load_subrip_str(*subrip_raw: str*) → *subaligner.subtitle.Subtitle*
Load a SubRip subtitle string.

Parameters **{string} -- The string representation of the SubRip content.** (*subrip_str*) –

Returns Subtitle – Subtitle object.

classmethod load_tmp(*subtitle_file_path: str*) → *subaligner.subtitle.Subtitle*
Load a TMP subtitle file.

Parameters **{string} -- The path to the subtitle file.** (*subtitle_file_path*) –

Returns Subtitle – Subtitle object.

classmethod load_ttml(*subtitle_file_path: str*) → *subaligner.subtitle.Subtitle*
Load a TTML subtitle file.

Parameters **{string} -- The path to the subtitle file.** (*subtitle_file_path*) –

Returns Subtitle – Subtitle object.

classmethod load_webvtt(*subtitle_file_path: str*) → *subaligner.subtitle.Subtitle*
Load a WebVTT subtitle file.

Parameters **{string} -- The path to the subtitle file.** (*subtitle_file_path*) –

Returns Subtitle – Subtitle object.

classmethod load_ytt(*subtitle_file_path: str*) → *subaligner.subtitle.Subtitle*
Load a YouTube transcript subtitle file.

Parameters **{string} -- The path to the subtitle file.** (*subtitle_file_path*) –

Returns Subtitle – Subtitle object.

static remove_sound_effects_by_affixes(*subs: List[pysrt.SubRipItem]*, *se_prefix: str*, *se_suffix: Optional[str] = None*) → List[pysrt.SubRipItem]
Remove subtitles of sound effects based on prefix or prefix and suffix

Parameters

- **{list} -- A list of SubRipItems.** (*subs*) –
- **{string} -- A prefix indicating the start of the sound effect.** (*se_prefix*) –
- **{string} -- A suffix indicating the end of the sound effect (default** (*se_suffix*) – {None}).

Returns {list} – A list of SubRipItems.

static remove_sound_effects_by_case(*subs: List[pysrt.SubRipItem]*, *se_uppercase: bool = True*) → List[pysrt.SubRipItem]

Remove subtitles of sound effects based on case

Parameters

- **{list} -- A list of SubRipItems.** (*subs*) –
- **{bool} -- True when the sound effect is in uppercase or False when in lowercase (default** (*se_uppercase*) – {True}).

Returns {list} – A list of SubRipItems.

static save_subs_as_target_format(*subs: List[pysrt.SubRipItem]*, *source_file_path: str*, *target_file_path: str*, *frame_rate: Optional[float] = None*, *encoding: Optional[str] = None*) → None

Save SubRipItems with the format determined by the target file extension.

Parameters

- **{list} -- A list of SubRipItems.** (*subs*) –
- **{string} -- The path to the original subtitle file.** (*source_file_path*) –
- **{string} -- The path to the output subtitle file.** (*target_file_path*) –
- **{float} -- The frame rate used by conversion to formats such as MicroDVD** (*frame_rate*) –
- **{str} -- The encoding of the exported output file {default** (*encoding*) – None}.

classmethod shift_subtitle(*subtitle_file_path: str*, *seconds: float*, *shifted_subtitle_file_path: Optional[str] = None*, *suffix: str = '_shifted'*) → Optional[str]

Shift subtitle cues based on the input seconds.

Parameters

- **{string} -- The path to the subtitle file.** (*subtitle_file_path*) –
- **{float} -- The number of seconds by which the cues are shifted.** (*seconds*) –

Keyword Arguments

- **{string} -- The path to the shifted subtitle file (default** (*shifted_subtitle_file_path*) – {None}).
- **{string} -- The suffix used as part of the aligned subtitle file name.** (*suffix*) –

Returns string – The path to the shifted subtitle file.

property subs

static subtitle_extensions() → set

Get the file extensions of the supported subtitles.

Returns {set} – The subtitle extensions.

property subtitle_file_path

subaligner.trainer module

class subaligner.trainer.**Trainer**(*feature_embedder:* subaligner.embedder.FeatureEmbedder)
Bases: object

Network trainer.

EMBEDDING_TIMEOUT = 300

static get_done_epochs(*training_log: str*) → int
Get the number of finished epochs.

Parameters **{string} -- The path to the training log file.** (*training_log*) –

pre_train(*av_file_paths: List[str], subtitle_file_paths: List[str], training_dump_dir: str, hyperparameters:* subaligner.hyperparameters.Hyperparameters, *sound_effect_start_marker: Optional[str] = None, sound_effect_end_marker: Optional[str] = None*) → Tuple[List[float], List[float]]
Trigger the training process.

Parameters

- **{list} -- A list of paths to the input audio/video files.** (*av_file_paths*) –
- **{list} -- A list of paths to the subtitle files.** (*subtitle_file_paths*) –
- **{string} -- The directory of the training data dump file.** (*training_dump_dir*) –
- **{Hyperparameters} -- A configuration for hyperparameters used for training.** (*hyperparameters*) –
- **sound_effect_start_marker** – {string} – A string indicating the start of the ignored sound effect (default: {"("}).
- **sound_effect_end_marker** – {string} – A string indicating the end of the ignored sound effect (default: {")"}).

train(*av_file_paths: List[str], subtitle_file_paths: List[str], model_dir: str, weights_dir: str, config_dir: str, logs_dir: str, training_dump_dir: str, hyperparameters:* subaligner.hyperparameters.Hyperparameters, *training_log: str = 'training.log', resume: bool = False, sound_effect_start_marker: Optional[str] = None, sound_effect_end_marker: Optional[str] = None*) → None
Trigger the training process.

Parameters

- **{list} -- A list of paths to the input audio/video files.** (*av_file_paths*) –
- **{list} -- A list of paths to the subtitle files.** (*subtitle_file_paths*) –
- **{string} -- The directory of the model file.** (*model_dir*) –
- **{string} -- The directory of the weights file.** (*weights_dir*) –
- **{string} -- The directory of the hyperparameter file where hyperparameters will be saved.** (*config_dir*) –
- **{string} -- The directory of the log file.** (*logs_dir*) –

- **{string} -- The directory of the training data dump file.** (*training_dump_dir*) –
- **{Hyperparameters} -- A configuration for hyperparameters used for training.** (*hyperparameters*) –
- **{string} -- The path to the log file of epoch results (default** (*training_log*) – {"training.log"}).
- **{bool} -- True to continue with previous training result or False to start a new one (default** (*resume*) – {False}).
- **sound_effect_start_marker** – {string} – A string indicating the start of the ignored sound effect (default: {"("}).
- **sound_effect_end_marker** – {string} – A string indicating the end of the ignored sound effect (default: {")"}).

subaligner.transcriber module

subaligner.translator module

subaligner.utils module

class subaligner.utils.**Utils**

Bases: object

Utility functions

FFMPEG_BIN = 'ffmpeg'

static ass2srt(*ass_file_path: str*, *srt_file_path: Optional[str] = None*) → None

Convert Advanced SubStation Alpha v4.0+ subtitles to SubRip subtitles.

Parameters

- **{string} -- The path to the ASS file.** (*ass_file_path*) –
- **{string} -- The path to the SubRip file.** (*srt_file_path*) –

static contains_embedded_subtitles(*video_file_path: str*, *timeout_secs: int = 30*) → bool

Detect if the input video contains embedded subtitles.

Parameters

- **{string} -- The path to the video file.** (*video_file_path*) –
- **{int} -- The timeout in seconds on extraction {default** (*timeout_secs*) – 30}.

Returns bool – True if the video contains embedded subtitles or False otherwise.

static detect_encoding(*subtitle_file_path: str*) → str

Detect the encoding of the subtitle file.

Parameters **{string} -- The path to the subtitle file.** (*subtitle_file_path*) –

Returns string – The string represent the encoding

static double_quoted(*s: str*) → str

static download_file(*remote_file_url: str*, *local_file_path: str*) → None

static extract_matroska_subtitle(*mkv_file_path: str*, *stream_index: int*, *output_file_path: str*, *timeout_secs: int = 30*) → None

Extract subtitles from Matroska files and convert them into the output format.

Parameters

- **{string} -- The path to the Matroska file.** (*mkv_file_path*) –
- **{int} -- The index of the subtitle stream** (*stream_index*) –
- **{string} -- The path to the output file.** (*output_file_path*) –
- **{int} -- The timeout in seconds on extraction {default** (*timeout_secs*) – 30}.

static extract_teletext_as_subtitle(*ts_file_path: str*, *page_num: int*, *output_file_path: str*, *timeout_secs: int = 30*) → None

Extract DVB Teletext from MPEG transport stream files and convert them into the output format.

Parameters

- **{string} -- The path to the Transport Stream file.** (*ts_file_path*) –
- **{int} -- The page number for the Teletext** (*page_num*) –
- **{string} -- The path to the output file.** (*output_file_path*) –
- **{int} -- The timeout in seconds on extraction {default** (*timeout_secs*) – 30}.

static format_timestamp(*seconds: float*) → str

static get_file_root_and_extension(*file_path: str*) → Tuple[str, str]

Get the root path and the extension of the input file path.

Returns tuple – the root path and the extension of the input file path.

static get_iso_639_alpha_2(*language_code: str*) → str

Find the alpha 2 language code based on an alpha 3 one.

Parameters **{string} -- An alpha 3 language code derived from ISO 639-3.** (*language_code*) –

Returns string – The alpha 2 language code if exists otherwise the alpha 3 one.

Raises **ValueError -- Thrown when the input language code cannot be recognised.** –

static get_language_table() → List[str]

Get all known language codes and their human-readable versions.

Returns list – A list of all known language codes and their human-readable versions.

static get_misc_language_codes() → List[str]

Get all known language codes.

Returns list – A list of all known language codes.

static get_stretch_language_codes() → List[str]

Get language codes used by stretch.

Returns list – A list of language codes derived from ISO 639-3.

static microdvd2srt(*microdvd_file_path: str*, *srt_file_path: Optional[str] = None*) → None

Convert MicroDVD subtitles to SubRip subtitles.

Parameters

- **{string} -- The path to the MPL2 file.** (*microdvd_file_path*) –
- **{string} -- The path to the SubRip file.** (*srt_file_path*) –

static mpl22srt(*mpl2_file_path: str*, *srt_file_path: Optional[str] = None*) → None
Convert MPL2 subtitles to SubRip subtitles.

Parameters

- **{string} -- The path to the MPL2 file.** (*mpl2_file_path*) –
- **{string} -- The path to the SubRip file.** (*srt_file_path*) –

static remove_trailing_newlines(*source_file_path: str*, *encoding: Optional[str]*, *target_file_path: Optional[str] = None*) → None

static sami2srt(*sami_file_path: str*, *srt_file_path: Optional[str] = None*) → None
Convert SAMI subtitles to SubRip subtitles.

Parameters

- **{string} -- The path to the SAMI file.** (*sami_file_path*) –
- **{string} -- The path to the SubRip file.** (*srt_file_path*) –

static sbv2srt(*sbv_file_path: str*, *srt_file_path: Optional[str] = None*) → None
Convert SubViewer subtitles to SubRip subtitles.

Parameters

- **{string} -- The path to the SubViewer file.** (*sbv_file_path*) –
- **{string} -- The path to the SubRip file.** (*srt_file_path*) –

static scc2srt(*scc_file_path: str*, *srt_file_path: Optional[str] = None*) → None
Convert SCC subtitles to SubRip subtitles.

Parameters

- **{string} -- The path to the Scenarist Closed Captions file.** (*scc_file_path*) –
- **{string} -- The path to the SubRip file.** (*srt_file_path*) –

static srt2ass(*srt_file_path: str*, *ass_file_path: Optional[str] = None*) → None
Convert SubRip subtitles to Advanced SubStation Alpha v4.0+ subtitles.

Parameters

- **{string} -- The path to the SubRip file.** (*srt_file_path*) –
- **{string} -- The path to the ASS file.** (*ass_file_path*) –

static srt2microdvd(*srt_file_path: str*, *microdvd_file_path: Optional[str] = None*, *frame_rate: Optional[float] = 25.0*)
Convert SubRip subtitles to MicroDVD subtitles.

Parameters

- **{string} -- The path to the SubRip file.** (*srt_file_path*) –
- **{string} -- The path to the MicroDVD file.** (*microdvd_file_path*) –
- **{float} -- The frame rate for frame-based MicroDVD.** (*frame_rate*) –

static srt2mpl2(*srt_file_path: str*, *mpl2_file_path: Optional[str] = None*) → None
Convert SubRip subtitles to MPL2 subtitles.

Parameters

- **{string} -- The path to the SubRip file.** (*srt_file_path*) –
- **{string} -- The path to the MPL2 file.** (*mpl2_file_path*) –

static srt2sami(*srt_file_path: str*, *sami_file_path: Optional[str] = None*) → None
Convert SubRip subtitles to SAMI subtitles.

Parameters

- **{string} -- The path to the SubRip file.** (*srt_file_path*) –
- **{string} -- The path to the SAMI file.** (*sami_file_path*) –

static srt2sbv(*srt_file_path: str*, *sbv_file_path: Optional[str] = None*) → None
Convert SubRip subtitles to SubViewer subtitles.

Parameters

- **{string} -- The path to the SubRip file.** (*srt_file_path*) –
- **{string} -- The path to the SubViewer file.** (*sbv_file_path*) –

static srt2scc(*srt_file_path: str*, *scc_file_path: Optional[str] = None*) → None
Convert SubRip subtitles to SCC subtitles.

Parameters

- **{string} -- The path to the SubRip file.** (*srt_file_path*) –
- **{string} -- The path to the Scenarist Closed Captions file.** (*scc_file_path*) –

static srt2ssa(*srt_file_path: str*, *ssa_file_path: Optional[str] = None*) → None
Convert SubRip subtitles to SubStation Alpha v4.0 subtitles.

Parameters

- **{string} -- The path to the SubRip file.** (*srt_file_path*) –
- **{string} -- The path to the SSA file.** (*ssa_file_path*) –

static srt2tmp(*srt_file_path: str*, *tmp_file_path: Optional[str] = None*) → None
Convert SubRip subtitles to TMP subtitles.

Parameters

- **{string} -- The path to the SubRip file.** (*srt_file_path*) –
- **{string} -- The path to the TMP file.** (*tmp_file_path*) –

static srt2ttml(*srt_file_path: str*, *ttml_file_path: Optional[str] = None*) → None
Convert SubRip subtitles to TTML subtitles.

Parameters

- **{string} -- The path to the SubRip file.** (*srt_file_path*) –
- **{string} -- The path to the TTML file.** (*ttml_file_path*) –

static srt2vtt(*srt_file_path: str*, *vtt_file_path: Optional[str] = None*, *timeout_secs: int = 30*) → None
Convert SubRip subtitles to WebVTT subtitles.

Parameters

- **{string} -- The path to the SubRip file.** (*srt_file_path*) –
- **{string} -- The path to the WebVTT file.** (*vtt_file_path*) –
- **{int} -- The timeout in seconds on conversion {default** (*timeout_secs*) – 30}.

static srt2ytt(*srt_file_path: str*, *transcript_file_path: Optional[str] = None*) → None
Convert SubRip subtitles to YouTube transcript subtitles.

Parameters

- **{string} -- The path to the SubRip file.** (*srt_file_path*) –
- **{string} -- The path to the YouTube transcript file.** (*transcript_file_path*) –

static ssa2srt(*ssa_file_path: str*, *srt_file_path: Optional[str] = None*) → None
Convert SubStation Alpha v4.0 subtitles to SubRip subtitles.

Parameters

- **{string} -- The path to the SSA file.** (*ssa_file_path*) –
- **{string} -- The path to the SubRip file.** (*srt_file_path*) –

static stl2srt(*stl_file_path: str*, *srt_file_path: Optional[str] = None*) → None
Convert EBU-STL subtitles to SubRip subtitles.

Parameters

- **{string} -- The path to the EBU-STL file.** (*stl_file_path*) –
- **{string} -- The path to the SubRip file.** (*srt_file_path*) –

static suppress_lib_logs() → None

static tmp2srt(*tmp_file_path: str*, *srt_file_path: Optional[str] = None*) → None
Convert TMP subtitles to SubRip subtitles.

Parameters

- **{string} -- The path to the TMP file.** (*mpl2_file_path*) –
- **{string} -- The path to the SubRip file.** (*tmp_file_path*) –

static ttml2srt(*ttml_file_path: str*, *srt_file_path: Optional[str] = None*) → None
Convert TTML subtitles to SubRip subtitles.

Parameters

- **{string} -- The path to the TTML file.** (*ttml_file_path*) –
- **{string} -- The path to the SubRip file.** (*srt_file_path*) –

static vtt2srt(*vtt_file_path: str*, *srt_file_path: Optional[str] = None*, *timeout_secs: int = 30*) → None
Convert WebVTT subtitles to SubRip subtitles.

Parameters

- **{string} -- The path to the WebVTT file.** (*vtt_file_path*) –
- **{string} -- The path to the SubRip file.** (*srt_file_path*) –
- **{int} -- The timeout in seconds on conversion {default** (*timeout_secs*) – 30}.

static ytt2srt(*transcript_file_path: str*, *srt_file_path: Optional[str] = None*) → None
Convert YouTube transcript subtitles to SubRip subtitles.

Parameters

- **{string} -- The path to the YouTube transcript file.** (*transcript_file_path*) –
- **{string} -- The path to the SubRip file.** (*srt_file_path*) –

Module contents

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

S

INDEX

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

Y