spych Documentation

Release 0.1.0

M.Buechi

Apr 07, 2018

NOTES

Spych is a set of python tools for tasks related to automatic speech recognition. The most used functions are accessible via command line interface.

CHAPTER 1

Installation

1.1 Requirements

Python 3 is needed for spych.

1.2 Source

Install latest development version via:

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

CHAPTER 2

Data Formats

2.1 Spych dataset

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

files.txt

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

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

Example:

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

utterances.txt

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

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

Example:

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

speakers.json (optional)

Contains any additional information about the speakers. Of course any other information can be stored. For every speaker a key/value pairs can be defined. Basically any information can be saved, but there are some predefined values:

- gender : The gender of the speaker.
- synthesized : (True/False) If it is synthesized speech.
- synthesizer_voice : If it is synthesized, what voice was used.
- synthesizer_effects : If it is synthesized, what effect were applied.
- synthesizer_tool : If it is synthesized, what tool was used for synthesis.

Example:

```
{
    "marc": {
        "gender": "m",
        "synthesized": true,
        "synthesizer_tool": "MaryTTS"
    },
    "sam": {
        "gender": "m"
    },
    "jenny": {
        "gender": "m"
    }
}
```

utt2spk.txt

This file contains the mapping from utterance to speaker, which gives the information which speaker has spoken a given utterance. Every line contains one mapping from utterance-id to speaker-id.

```
<utterance-id> <speaker-id>
```

Example:

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

segmentation_[x].txt

There can be multiple segmentations in a dataset (e.g. text-transcription, raw-text-transcription - with punctuation). Every segmentation is saved in a separate file with the prefix *segmentation_*. A single file contains segmentations of a specific type of all utterances. A segmentation of an utterance can contain one or more tokens (e.g. in a text segmentation every word would be a token). A token optionally can have a start and end time (in seconds within the utterance). For tokens without start/end defined -1 is set. Every line in the file defines one token. The tokens are stored in order per utterance (e.g. 1. word, 2. word, 3. word, . . .).

```
<utterance-id> <start> <end> <token-value>
```

Example:

```
1_hello -1 -1 hi
1_hello -1 -1 this
```

```
1_hello -1 -1 is
1_hello_sam -1 -1 hello
1_hello_sam -1 -1 sam
2_this_is -1 -1 this
2_this_is -1 -1 is
2_this_is -1 -1 me
3_goto -1 -1 go
3_goto -1 -1 to
3_goto -1 -1 the
3_goto -1 -1 mall
```

subview_[x].txt

A subview defines a kind of a filter for dataset, which builds a subset. A dataset can contain multiple subviews. Every subview can has multiple filter criteria:

- filtered-utterances : A list of utterance-ids which are used in the subview.
- filtered-speakers : A list of speaker-ids which are used in the subview.
- utterance_idx_patterns : A list of reg-expressions to match used utterances.
- speaker_idx_patterns : A list of reg-expressions to match used speakers.
- utterance_idx_not_patterns : A list of reg-expressions to match not used utterances.
- speaker_idx_not_patterns : A list of reg-expressions to match not used speakers.

The subview just contains the items, that match all criteria. Every line in the file contains one filter criterion.

```
<criterion-name> <criterion-value>
```

Example:

```
filtered_speaker_ids marc sam
filtered_utt_ids 1_hello 2_this
```

features.txt

Contains a list of stored features. A dataset can have different feature containers. Every container contains the features of all utterances of a given type (e.g. MFCC features). A feature container is a folder on the filesystem, which contains one numpy file per utterance with it's feature matrix. Every line contains one container of features.

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

Example:

```
mfcc mfcc_features
fbank fbank_features
```

2.2 Spych dataset legacy

This is the old version of the spych dataset format. A dataset is a folder consisting of the following files.

wavs.txt

The mapping between recording id to audio file path.

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

Example:

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

utterances.txt

Defines the utterances within a recording. Every line defines one utterance. If start and end are not given it's assumed the recording consist of a single utterance. And end value of -1 indicates the end of the recording. Start and end are measured in seconds.

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

Example:

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

transcriptions.txt (optional)

List of transcriptions for utterances.

<utterance-id> <transcription>

Example:

```
1_hello hi this is
1_hello_sam hello sam
2_this_is this is me
3_goto go to the mall
```

transcriptions_raw.txt (optional)

List of raw transcriptions for utterances. These may contain punctuation, numbers that aren't written out, abbreviations …

<utterance-id> <raw-transcription>

Example:

```
1_hello hi, this is?
1_hello_sam hello sam!!
2_this_is this is me.
3_goto go to the mall
```

utt2spk.txt (optional)

Defines the speakers of the utterances.

<utterance-id> <speaker-id>

Example:

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

speaker_info.json (optional)

Contains any additional information about the speakers. Currently gender is the only defined key. Of course any other information can be stored.

Example:

```
{
    "marc": {"gender": "m"},
    "sam": {"gender": "m"},
    "jenny": {"gender": "f"}
}
```

CHAPTER 3

Command Line Tutorial

3.1 Dataset

For the following examples it is assumed we are in a directory containing a dataset in the spych format:

```
$ ls
audio_files
features_fbank
features_mfcc
features_mfcc_cmn_deltas
features_mfcc_cmvn_deltas
features_spectrum
features.txt
files.txt
segmentation_raw_text.txt
segmentation_text.txt
speakers.json
utt2spk.txt
utterances.txt
```

3.1.1 Display info

Display information for a dataset (num. speakers, num. utterances, ...):

```
spych dataset info .
```

Output:

```
Dataset .:

Path                        /some/path/data/audio/real_test
Num. Utterances             1080
Num. Files                  1080
```

```
Num. Speakers                16

Segmentations:               raw_text, text
Features:                    mfcc, spectrum, mfcc_cmvn_deltas, fbank, mfcc_cmn_deltas
Subviews:
```

For detailed info (feature statistics) append the **–detailed** flag.

CHAPTER 4

Dataset Tutorial

4.1 Load / Save

To load a dataset in python from a directory, the *spych.data.dataset.Dataset.load()* can be used. By default spych expects the dataset in the spych format. If the dataset has another format, the loader argument has to be provided.

```
# create a loader
kaldi_loader = loader.KaldiDatasetLoader()

# load the dataset
ds = dataset.Dataset.load('/path/to/the/dataset/folder', loader=kaldi_loader)
```

For saving a dataset the same principle is used. You can either use *spych.data.dataset.Dataset.save_at()* or *spych.data.dataset.Dataset.save()*. When using the latter the dataset object needs to have a path defined, where to save it. Every dataset is associated with a loader (by default the spych loader), which it will use for saving, if not defined otherwise (as in the following example).

```
# create a loader
kaldi_loader = loader.KaldiDatasetLoader()

# save the dataset
ds.save_at('/path/to/the/dataset/folder', loader=kaldi_loader)
```

The load and save functions also can be used to copy or convert a dataset. Following an example that loads a dataset from the spych format and stores it as kaldi format in another place.

```
# load it (spych is default, so no loader has to be defined)
ds = dataset.Dataset.load('/path/to/the/spych/dataset/folder')

# save it in kaldi format (loader can also be passed with its name)
ds.save_at('/path/to/save/as/kaldi/format', loader='kaldi')
```

Now by default the audio files and feature files are not copied to the new path. There will be just a relative path to the old folder. If you also want to copy the files:

```
ds.save_at('/path/to/save/as/kaldi/format', loader='kaldi', copy_files=True)
```

4.2 Generate batches

For training deep neural networks the data needs to be split up in batches. For this purpose the *spych.data.dataset.BatchGenerator* was implemented. It basically generates batches of a specified size with randomly selected utterances. One special thing is that you can specify multiple datasets. If you do so it always first checks for common utterance ids over all given datasets and just uses these to create the batches. It can be used to in different ways.

One way is to generate batches that contain a list of utterances.

```
# create the batch generator
batcher = dataset.BatchGenerator(dataset_a)

# get generator over batches with the desired batch size (number of utterances per␣
↪batch)
gen = batcher.generate_utterance_batches(512)

# now you can iterate over the batches
for batch in gen:

    # batch is now a list of utterance-ids
    print(len(batch)) # --> 512 (not the last batch)
```

Another way is to generate batches that contains the concatenated features of given number of utterances.

```
# create the batch generator
batcher = dataset.BatchGenerator([dataset_a, dataset_b])

# get generator over batches with the desired batch size (number of utterances per␣
↪batch)
gen = batcher.generate_feature_batches('mfcc', 512, splice_size=0, splice_step=1,␣
↪repeat_border_frames=True)

# now you can iterate over the batches
for batch in gen:

    # batch is now a list of features as np-arrays [from dataset_a, from dataset_b]
    print(len(batch)) # --> 2
```

CHAPTER 5

spych.data

class spych.data.**File**(*idx*, *path*)

class spych.data.**Utterance**(*idx*, *file_idx*, *speaker_idx=None*, *start=0*, *end=-1*)
An utterance defines an audio sample. Normally an audio file can contain multiple utterances. But every utterance is a part of a file.

class spych.data.**Speaker**(*idx*, *gender=None*)

class spych.data.**Segmentation**(*segments=[]*, *utterance_idx=None*, *key='text'*)
Represents a sequence (e.g. Transcription / Alignment).

first_segment
Return the first segment.

classmethod from_audacity(*path*)
Return the segmentation read from an audacity label file.

classmethod from_ctm(*path*)
Return a list of segmentations read from a ctm file.

classmethod from_text(*text*, *utterance_idx=None*, *key='text'*)
Create a segmentation from a string. It will be space separated into segments.

Parameters

- **text** – The string to be segmented.
- **utterance_idx** – Utt id this segmentation belongs to.
- **key** – A key which identifies this segmentation.

Returns Segmentation object

last_segment
Return the last segment.

to_audacity(*path*)
Write the segmentation to a audacity label file.

to_ctm(*path*)
Write the segmentation to a ctm file.

to_text()
Return segments concatenated as space separated string.

class spych.data.**FeatureContainer**(*path*)
This class defines a container for storing features (of a given type) of all utterances.

get_statistics()
Return basic stats for the features. Return min,max,mean,meanstdev.

CHAPTER 6

spych.data.dataset

6.1 Data structure

class spych.data.dataset.**Dataset**(*path=None*, *loader=None*)
Represents an audio dataset.

Notes on paths: All paths stored in the dataset object (audio files, features) are absolute.

Parameters

- **path** – Path to a directory on the filesystem, which acts as root folder for the dataset. If no path is given the dataset cannot be saved on disk.
- **loader** (spych.data.dataset.io.DatasetLoader) – This object is used to save the dataset. By default *spych.data.dataset.io.SpychDatasetLoader* is used.

add_features(*utterance_idx*, *feature_matrix*, *feature_container*)
Adds the given features to the dataset. Features are stored directly to the filesystem, so this dataset has to have a path set.

Parameters

- **utterance_idx** – Utterance to which the features correspond.
- **feature_matrix** – A numpy array containing the features.
- **feature_container** – Name of the container to store the features in.

add_file(*path*, *file_idx=None*, *copy_file=False*)
Adds a new file to the dataset.

Parameters

- **path** – Path of the file to add.
- **file_idx** – The id to associate the file with. If None or already exists, one is generated.

- **copy_file** – If True the file is copied to the dataset folder, otherwise the given path is used directly.

Returns File object

add_segmentation(*utterance_idx*, *segments=None*, *key=None*)
Adds a new segmentation.

Parameters

- **utterance_idx** – Utterance id the segmentation is associated with.
- **segments** – Segments can be a string (will be space separated into tokens) or a list of segments.
- **key** – A key this segmentation is assiciated with. (If None the default key is used.)

Returns Segmentation object

add_speaker(*speaker_idx=None*, *gender=None*)
Adds a new speaker to the dataset.

Parameters

- **speaker_idx** – The id to associate the speaker with. If None or already exists, one is generated.
- **gender** – Gender of the speaker.

Returns Speaker object

add_subview(*name*, *subview*)
Add the subview to this dataset.

add_utterance(*file_idx*, *utterance_idx=None*, *speaker_idx=None*, *start=0*, *end=-1*)
Adds a new utterance to the dataset.

Parameters

- **file_idx** – The file id the utterance is in.
- **utterance_idx** – The id to associate with the utterance. If None or already exists, one is generated.
- **speaker_idx** – The speaker id to associate with the utterance.
- **start** – Start of the utterance within the file [seconds].
- **end** – End of the utterance within the file [seconds]. -1 equals the end of the file.

Returns Utterance object

create_feature_container(*name*, *path=None*)
Create a new feature container

export_subview(*name*)
Return a subview as a standalone dataset.

generate_features(*feature_pipeline*, *target_feature_name*, *source_feature_name=None*)
Creates new feature container with features generated with the given pipeline. If source_feature_name is not given the pipeline needs an extraction stage.

import_dataset(*import_dataset*, *copy_files=False*)
Merges the given dataset into this dataset.

Parameters

- **import_dataset** – Dataset to merge
- **copy_files** – If True moves the wavs to this datasets folder.

import_file(*file*, *copy_file=False*)
Import a copy the given file and return the new file obj.

import_segmentation(*segmentation*)
Adds an existing segmentation to the dataset. Uses key and utterance-id from the segmentation object.

import_speaker(*speaker*)
Import a copy of the given speaker and return the new speaker.

import_utterance(*utterance*)
Import a copy of the given utterance and return the new utterance.

classmethod load(*path*, *loader=None*)
Loads the dataset from the given path, using the given loader. If no loader is given the spych loader is used.

name
Get the name of the dataset (Equals basename of the path, if not None.) :return: name

num_subviews
Return number of subviews.

remove_files(*file_ids*, *delete_files=False*)
Deletes the given wavs.

Parameters

- **file_ids** – List of file_idx's or fileobj's
- **delete_files** – Also delete the files

remove_utterances(*utterance_ids*)
Removes the given utterances by id.

Parameters **utterance_ids** – List of utterance ids

save()
Save this dataset at self.path.

save_at(*path*, *loader=None*, *copy_files=False*)
Save this dataset at the given path. If the path differs from the current path set, the path gets updated.

Parameters

- **path** – Path to save the dataset to.
- **loader** – If you want to use another loader (e.g. to export to another format). Otherwise it uses the loader associated with this dataset.
- **copy_files** – If true the files are also stored in the new path, if not already there.

subdivide_speakers(*target_number_of_speakers*)
Divide the available speakers in the dataset into different speakers so the number of speakers is target_number_of_speakers.

Parameters **target_number_of_speakers** – Target number of speakers

class spych.data.dataset.**Subview**(*filtered_utterances=set()*, *filtered_speakers=set()*, *dataset=None*)
A subview is a filtered view on a dataset. For example it only uses a subset of utterance-id's.

does_utterance_match(*utterance*)
Return True if the given utterance matches all filter criteria. Otherwise return False.

6.2 Validation

class spych.data.dataset.**Validator**(*metrics=[]*, *expected_file_format=<spych.audio.format.AudioFileFormat object>*)

Class to validate a dataset.

classmethod full_validator(*expected_file_format=<spych.audio.format.AudioFileFormat object>*)

Returns a validator, that checks all metrics.

static get_features_with_missing_file(*dataset*)

Return a dictionary (feature-container, list of missing utterance-ids) where there is no feature file.

static get_files_empty(*dataset*)

Return a list of file-idx's that contain no data.

static get_files_missing(*dataset*)

Return a list of file-idx's where the actual file is missing.

static get_files_with_wrong_format(*dataset*, *expected_format=<spych.audio.format.AudioFileFormat object>*)

Return a list of file-idx's that don't conform the given audio format.

static get_files_without_utterances(*dataset*)

Return a list of file-idx's that don't reference any utterances.

static get_speakers_without_gender(*dataset*)

Return a list of speaker-idx's, where the gender is not defined.

static get_utterances_with_invalid_start_end(*dataset*)

Check if there are any utterances that have invalid start/end time.

Must be:

- float
- can be empty –> 0 -1
- end >= start
- start >= 0
- end >= 0 or end = -1

Parameters **dataset** – Dataset to check.

Returns List of utterance-ids with invalid start/end.

static get_utterances_with_missing_file_idx(*dataset*)

Return a list of utterances that reference a wav-id that isn't existing.

static get_utterances_with_missing_speaker(*dataset*)

Return list without or invalid speaker.

static get_utterances_without_segmentation(*dataset*)

Return a dictionary of key/utterance-idx's where no segmentation is available.

validate(*dataset*)

Return the validation results for the selected validation metrics. (dictionary metric/results)

6.3 Rectification

class spych.data.dataset.**Rectifier**(*tasks=[]*, *expected_file_format=<spych.audio.format.AudioFileFormat object>*)

Provides functionality to fix invalid dataset parts.

static remove_empty_wavs(*dataset_to_fix*)

Delete all files that have no content.

static remove_files_missing(*dataset_to_fix*)

Delete all files references, where the referenced wav file doesn't exist.

static remove_files_with_wrong_format(*dataset_to_fix*, *expected_format=<spych.audio.format.AudioFileFormat object>*)

Delete all wav files with the wrong format (Sampling rate, sample width, ...).

static remove_utterances_with_missing_file(*dataset_to_fix*)

Remove all utterances where the referenced file doesn't exist.

6.4 Iteration

class spych.data.dataset.**BatchGenerator**(*datasets*)

Class that provides functions to generate batches from a single or multiple datasets. If multiple datasets only utterances are considered, that exist in all of the given datasets.

batches_with_features(*feature_name*, *batch_size*, *feature_pipeline=None*)

Return a generator which yields batches. One batch contains concatenated features from batch_size utterances.

Parameters

- **feature_name** – Name of the feature container in the dataset to use.
- **batch_size** – Number of utterances in one batch
- **feature_pipeline** – If not None processes the features with the pipeline.

Returns generator

batches_with_spliced_features(*feature_name*, *batch_size*, *splice_sizes=0*, *splice_step=1*, *repeat_border_frames=True*)

Return a generator which yields batches. One batch contains the concatenated features of batch_size utterances. Yields list with length equal to the number of datasets in this generator. The list contains ndarrays with the concatenated features.

Parameters

- **feature_name** – Name of the feature container in the dataset to use.
- **batch_size** – Number of utterances in one batch
- **splice_sizes** – Appends x previous and next features to the sample. (e.g. if splice_size is 4 the sample contains 9 features in total).

If a list of ints is given the different splice-sizes for the different datasets are used. :param splice_step: Number of features to move forward for the next sample. :param repeat_border_frames: If True repeat the first and last frame for splicing. Otherwise pad with zeroes. :returns: generator

batches_with_utterance_idx_and_features(*feature_name*, *batch_size*, *feature_pipeline=None*)

Return a generator which yields batches. One batch contains features from batch_size utterances. Yields a list of lists. Every sublist contains first the utterance id and following the feature arrays (ndarray) of all datasets.

e.g. If there are two source datasets:

[[utt_id, dataset_1_feature, dataset_2_feature], [utt_id2, dataset_1_feature2, dataset_2_feature2], ...

]

Parameters

- **feature_name** – Name of the feature container in the dataset to use.
- **batch_size** – Number of utterances in one batch
- **feature_pipeline** – If not None processes the features with the pipeline.

Returns generator

batches_with_utterance_idxs(*batch_size*)

Return a generator which yields batches. One batch is a list of utterance-ids of size batch-size.

CHAPTER 7

spych.data.dataset.io

class spych.data.dataset.io.**SpychDatasetLoader**(*main_features=None*)

class spych.data.dataset.io.**LegacySpychDatasetLoader**(*main_features=None*)

Loads dataset from the old spych format.

wavs.txt: [wav-id] [relative-wav-path]

utterances.txt : [utt-id] [wav-id] [start] [end]

transcriptions.txt : [utt-id] [transcription]

transcriptions_raw.txt : [utt-id] [transcription raw] Transcription with punctuation.

utt2spk.txt : [utt-id] [speaker-id]

speaker_info.json : {

"speaker_id" [{] "gender" : "m"/"f", ...

}

class spych.data.dataset.io.**KaldiDatasetLoader**(*main_features=None*)

static feature_scp_generator(*path*)

Return a generator over all feature matrices defined in a scp.

static read_float_matrix(*rx_specifier*)

Return float matrix as np array for the given rx specifier.

static write_float_matrices(*scp_path*, *ark_path*, *matrices*)

Write the given dict matrices (utt-id/float ndarray) to the given scp and ark files.

class spych.data.dataset.io.**TudaDatasetLoader**(*main_features=None*)

Loads german tuda corpus.

CHAPTER 8

spych.utils

8.1 array

spych.utils.array.**splice_features**(*features*, *splice_size=0*, *splice_step=1*, *repeat_border_frames=True*)

Splice features in a array. Splicing extends a single feature with right and left context features.

Parameters

- **features** – A 2D np-array with the shape (nr_of_features x feature_dimension).
- **splice_size** – Number of context features to use on the right and left. (splice_size = 4 –> resulting feature sizes = 9 * initial featdim)
- **splice_step** – Number of features to move for the next splice.
- **repeat_border_frames** – At the begin and end of the feature series the feature matrix has to be padded for splicing. If True it pads with the first/last feature otherwise with zeroes.

Returns A 2D np-array with the spliced features.

spych.utils.array.**unsplice_features**(*features*, *splice_size=0*, *splice_step=1*)

Take a array of spliced features and unsplice them. Uses the averaged values for a single feature.

Parameters

- **features** – A 2D np-array with the spliced features.
- **splice_size** – Number of right and left context features that were used for splicing.
- **splice_step** – Step that was used for splicing.

Returns A 2D np-array with the unspliced features.

8.2 math

spych.utils.math.**calculate_absolute_proportions**(*count*, *proportions={}*)

Given an int and a dictionary with name/float entries, which represent relative proportions. It calculates the absolute proportions.

e.g. input 20, {'a':0.5, 'b':0.5} –> output {'a':10, 'b':10}

If it doesn't come out even, the rest is appended to the different proportions randomly.

Parameters

- **count** – the total number to divide into proportions
- **proportions** – dict name/prop_value

Returns absolute proportions

spych.utils.math.**calculate_relative_proportions**(*proportions*)

Converts proportions so they sum to 1.0.

Parameters **proportions** – dict name/prop_value

Returns dict name/prop_value

spych.utils.math.**try_distribute_values_proportionally**(*values*, *proportions*)

e.g.

values = {'a': 160, 'b': 160, 'c': 20, 'd': 100, 'e': 50, 'f': 60} proportions = {'x': 0.6, 'y': 0.2, 'z': 0.2}

out: {'x' : [a,b], 'y' : [c,e]} ...

Parameters

- **values** – value_id/value
- **proportions** – prop_id/prop

Returns tuple (prop_id/list of value_ids) (resulting proportions)

8.3 text

spych.utils.text.**remove_punctuation**(*text*, *exceptions=[]*)

Removes the punctuation from a string.

Parameters

- **text** – Text
- **exceptions** – Symbols not to remove.

Returns Text without punctuation.

spych.utils.text.**starts_with_prefix_in_list**(*text*, *prefixes*)

Checks if the given string starts with one of the prefixes in the list.

Parameters

- **text** – Text
- **prefixes** – List of prefixes

Returns True/False

8.4 naming

spych.utils.naming.**generate_name**(*length=15*, *not_in=None*)
Generates a random string of lowercase letters with the given length.

Parameters

- **length** – Length of the string to output.
- **not_in** – Only return a string not in the given iterator.

Returns

spych.utils.naming.**index_name_if_in_list**(*name*, *name_list*, *suffix=''*, *prefix=''*)
Adds an index to the given name if it already exists in the given list.

Parameters

- **name** – Name
- **name_list** – List of names that the new name must differ from.
- **suffix** – The suffix to append after the index.
- **prefix** – The prefix to append in front of the index.

Returns Unique name

8.5 jsonfile

spych.utils.jsonfile.**read_json_file**(*path*)
Reads json file.

Parameters **path** – Path to read

Returns Data

spych.utils.jsonfile.**write_json_to_file**(*path*, *data*)
Writes data as json to file.

Parameters

- **path** – Path to write to
- **data** – Data

8.6 textfile

spych.utils.textfile.**read_key_value_lines**(*path*, *separator=' '*, *default_value=''*)
Reads lines of a text file with two columns as key/value dictionary.

Parameters

- **path** – Path to the file.
- **separator** – Separator that is used to split key and value.
- **default_value** – If no value is given this value is used.

Returns dict

spych.utils.textfile.**read_separated_lines**(*path*, *separator=' '*, *max_columns=-1*)

Reads a text file where each line represents a record with some separated columns.

Parameters

- **path** – Path to the file to read.
- **separator** – Separator that is used to split the columns.
- **max_columns** – Number of max columns (if the separator occurs within the last column).

Returns list

spych.utils.textfile.**read_separated_lines_generator**(*path*, *separator=' '*, *max_columns=-1*, *ignore_lines_starting_with=[]*)

Creates a generator through all lines of a file and returns the splitted line.

Parameters

- **path** – Path to the file.
- **separator** – Separator that is used to split the columns.
- **max_columns** – Number of max columns (if the separator occurs within the last column).
- **ignore_lines_starting_with** – Lines starting with a string in this list will be ignored.

Returns generator

spych.utils.textfile.**read_separated_lines_with_first_key**(*path*, *separator=' '*, *max_columns=-1*)

Reads the separated lines of a file and returns a dictionary with the first column as keys, value is a list with the rest of the columns.

Parameters

- **path** – Path to the file to read.
- **separator** – Separator that is used to split the columns.
- **max_columns** – Number of max columns (if the separator occurs within the last column).

Returns dict

spych.utils.textfile.**write_separated_lines**(*path*, *values*, *separator=' '*, *sort_by_column=0*)

Writes list or dict to file line by line. Dict can have list as value then they written separated on the line.

Parameters

- **path** – Path to write file to.
- **values** – Dict or list
- **separator** – Separator to use between columns.
- **sort_by_column** – if >= 0, sorts the list by the given index, if its 0 or 1 and its a dictionary it sorts it by either the key (0) or value (1). By default 0, meaning sorted by the first column or the key.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index