MoSeq2-Viz Documentation

version

Datta Lab

August 23, 2021

Contents

Welcome to moseq2-viz's documentation!

moseq2_viz package

CLI Module

moseq2-viz

```
moseq2-viz [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

Default: False

add-group

Change group name in index file given a key-value pair

```
moseq2-viz add-group [OPTIONS] INDEX_FILE
```

Options

-k, --key <key>
Key to search for value

Default: SubjectName

-v, --value <value>
Value to search for

Default: Mouse

-g, --group <group>
Group name to map to

Default: Group1

-e, --exact
Exact match only

Default: False

--lowercase
Lowercase text filter

Default: False

-n, --negative
Negative match (everything that does not match is included)

Default: False

Arguments

INDEX_FILE
Required argument

copy-h5-metadata-to-yaml

Copies metadata within an h5 file to a yaml file.

```
moseq2-viz copy-h5-metadata-to-yaml [OPTIONS]
```

Options

-i, --input-dir <input_dir>
Directory to find h5 files

Default: /Users/aymanzeine/Desktop/moseq/moseq2-viz/docs

get-best-model

Returns the model with the closest median duration to the PC Changepoints, given a directory containing multiple models

```
moseq2-viz get-best-model [OPTIONS] MODEL_DIR CP_PATH OUTPUT_FILE
```

Options

--plot-all
Plot all included model results

Default: False

--ext <ext>
Model extensions found in input directory

Default: p

--fps <fps>
Frames per second

Default: 30

Arguments

MODEL_DIR
Required argument

CP_PATH
Required argument

OUTPUT_FILE
Required argument

make-crowd-movies

Writes movies of overlaid examples of the rodent perform a given syllable

```
moseq2-viz make-crowd-movies [OPTIONS] INDEX_FILE MODEL_PATH
```

Options

--max-syllable <max_syllable>
Index of max syllable to render

Default: 40

-m, --max-examples <max_examples>
Number of examples to show

Default: 40

--processes <processes>
Number of processes to use for rendering crowd movies. Default None uses every process

--separate-by <separate_by>
Generate crowd movies from individual group sources.

Default: default
Options: default|groups|sessions|subjects

--specific-syllable <specific_syllable>

Index of max syllable to render

-s, --session-names <session_names>
SessionNames to create crowd movies from

Default:

--sort <sort>
Sort syllables by usage

Default: True

--count <count>
How to quantify syllable usage

Default: usage
Options: usage|frames

-o, --output-dir <output_dir>
Path to store files

Default: /Users/aymanzeine/Desktop/moseq/moseq2-viz/docs/crowd_movies

--gaussfilter-space <gaussfilter_space>
Spatial filter for data (Gaussian)

Default: 0, 0

--medfilter-space <medfilter_space>
Median spatial filter

Default: 0

--min-height <min_height>
Minimum height for scaling videos

Default: 5

--max-height <max_height>
Minimum height for scaling videos

Default: 80

--raw-size <raw_size>
Size of original videos

Default: 512, 424

--scale <scale>
Scaling from pixel units to mm

Default: 1

--cmap <cmap>
Name of valid Matplotlib colormap for false-coloring images

Default: jet

--max-dur <max_dur>
Exclude syllables longer than this number of frames (None for no limit)

Default: 60

--min-dur <min_dur>
Exclude syllables shorter than this number of frames

Default: 0

--legacy-jitter-fix <legacy_jitter_fix>
Set to true if you notice jitter in your crowd movies

Default: False

--frame-path <frame_path>
Path to depth frames in h5 file

Default: frames

-p, --progress-bar
Show verbose progress bars.

Default: False

--pad <pad>
Pad crowd movie videos with this many frames.

Default: 30

Arguments

INDEX_FILE
Required argument

MODEL_PATH
Required argument

plot-group-position-heatmaps

Plots position heatmaps for each group in the index file

```
moseq2-viz plot-group-position-heatmaps [OPTIONS] INDEX_FILE
```

Options

--output-file <output_file>

Default: /Users/aymanzeine/Desktop/moseq/moseq2-viz/docs/scalars

Arguments

INDEX_FILE
Required argument

plot-scalar-summary

Plots a scalar summary of the index file data.

```
moseq2-viz plot-scalar-summary [OPTIONS] INDEX_FILE
```

Options

--output-file <output_file>

Default: /Users/aymanzeine/Desktop/moseq/moseq2-viz/docs/scalars

-c, --colors <colors>
Colors to plot groups with.

Arguments

INDEX_FILE
Required argument

plot-stats

Plots syllable usages with different sorting,coloring and grouping capabilities

```
moseq2-viz plot-stats [OPTIONS] INDEX_FILE MODEL_FIT
```

Options

--stat <stat>

Statistic to plot.

Default: usage

--output-file <output_file>
Filename to store plot

Default: /Users/aymanzeine/Desktop/moseq/moseq2-viz/docs/syll_stat

--sort <sort>
Sort syllables by usage

Default: True

--figsize <figsize>
Size in inches (w x h) of the plotted figure.

Default: 10, 5

--count <count>
How to relabel syllables

Default: usage
Options: usage|frames

--max-syllable <max_syllable>
Index of max syllable to render

Default: 40

-g, --group <group>
Name of group(s) to show

-o, --ordering <ordering>
How to order syllables in plot

Default: stat

--ctrl-group <ctrl_group>
Name of control group. Only if ordering = 'diff'

--exp-group <exp_group>
Name of experimental group. Only if ordering = 'diff'

-c, --colors <colors>
Colors to plot groups with.

Arguments

INDEX_FILE
Required argument

MODEL_FIT
Required argument

plot-transition-graph

Plots the transition graph depicting the transition probabilities between syllables.

```
moseq2-viz plot-transition-graph [OPTIONS] INDEX_FILE MODEL_FIT
```

Options

--max-syllable <max_syllable>
Index of max syllable to render

Default: 40

-g, --group <group>
Name of group(s) to show

--output-file <output_file>
Filename to store plot

Default: /Users/aymanzeine/Desktop/moseq/moseq2-viz/docs/transitions

--normalize <normalize>
How to normalize transition probabilities

Default: bigram
Options: bigram|rows|columns

--edge-threshold <edge_threshold>
Threshold for edges to show

Default: 0.001

--usage-threshold <usage_threshold>
Threshold for nodes to show

Default: 0

--layout <layout>
Default networkx layout algorithm

Default: spring

-k, --keep-orphans
Show orphaned nodes

Default: False

--orphan-weight <orphan_weight>
Weight for non-existent connections

Default: 0

--arrows
Show arrows

Default: False

--sort <sort>
Sort syllables by usage

Default: True

--count <count>
How to quantify syllable usage

Default: usage
Options: usage|frames

--edge-scaling <edge_scaling>
Scale factor from transition probabilities to edge width

Default: 250

--node-scaling <node_scaling>
Scale factor for nodes by usage

Default: 10000.0

--scale-node-by-usage <scale_node_by_usage>
Scale node sizes by usages probabilities

Default: True

--width-per-group <width_per_group>
Width (in inches) for figure canvas per group

Default: 8

Arguments

INDEX_FILE
Required argument

MODEL_FIT
Required argument

plot-verbose-position-heatmaps

Plots a position heatmap for each session in the index file.

```
moseq2-viz plot-verbose-position-heatmaps [OPTIONS] INDEX_FILE
```

Options

--output-file <output_file>

Default: /Users/aymanzeine/Desktop/moseq/moseq2-viz/docs/scalars

Arguments

INDEX_FILE
Required argument

GUI Module

GUI front-end operations. This module contains all the functionality and configurable parameters users can alter to most accurately process their data.

Note: These functions perform jupyter notebook specific pre-processing, loads in corresponding parameters from the CLI functions, then call the corresponding wrapper function with the given input parameters.

moseq2_viz.gui.**add_group** (index_file, by='SessionName', value='default', group='default', exact=False, lowercase=False, negative=False)
Updates index file SubjectName group names with user defined group names.

Parameters:

- **index_file (str)** (*path to index file*)
- **value (str or list)** (*SessionName value(s) to search for and update with the corresponding group(s)*)
- **group (str or list)** (*Respective group name(s) to set corresponding sessions as.*)
- **exact (bool)** (*indicate whether to search for exact match.*)
- **lowercase (bool)** (*indicate whether to convert all searched for names to lowercase.*)
- **negative (bool)** (*whether to update the inverse of the found selection.*)

moseq2_viz.gui.**get_best_fit_model** (progress_paths, output_file=None, plot_all=False, fps=30, ext='p', objective='duration')
Given a directory containing multiple models, and the path to the pca scores they were trained on, this function returns the path to the model that has the closest median syllable duration to that of the PC Scores.

Parameters:
- **progress_paths (dict)** (*Dict containing paths the to model directory and pca scores file*)
- **output_file (str)** (*Optional path to save the comparison plot*)
- **plot_all (bool)** (*Indicates whether to plot all the models' changepoint distributions with the PCs, highlighting*) – the best model curve.
- **fps (int)** (*Frames per second.*)
- **ext (str)** (*File extension to search for models with*)
- **objective (str)** (*can be either duration or jsd. The objective finds the best model*) – based on either median changepoint durations or the jensen-shannon divergence beteween changepoint duration distributions

Returns: **best_fit_model (str)**

Return type: Path tp best fit model

`moseq2_viz.gui.`**`get_groups_command`** (index_file)
Jupyter Notebook to print index file current metadata groupings.

Parameters: **index_file (str)** (*path to index file*)

Returns: **(int)**

Return type: number of unique groups

Utilities Module

General utility functions to facilitate loading and organizing data.

`moseq2_viz.util.`**`assert_model_and_index_uuids_match`** (model, index)
Asserts that both the model and index file contain the same set of UUIDs.

Parameters:
- **model (str or dict)** (*if str, must be a path to the model. If dict, it contains the*) – model data after parsing the model results
- **index (str or dict)** (*if str, must be a path to the index file. If dict, it contains*) – the parsed and sorted index.

`moseq2_viz.util.`**`camel_to_snake`** (s)
Converts CamelCase to snake_case

Parameters: **s (str)** (*string to convert to snake case*)

Returns: **(str)**

Return type: snake_case string

`moseq2_viz.util.`**`clean_dict`** (dct)
Casts numpy array values into lists and *np.generic* data into scalar values.

Parameters: **dct (dict)** (*dictionary with values to clean.*)

Returns: **(dict)**

Return type: dictionary with standardized value types.

`moseq2_viz.util.`**`get_index_hits`** (config_data, metadata, key, v)
Searches for matching keys in given index file metadata dict. Returns list of booleans indicating that a session was found.

Parameters:
- **config_data (dict)** (*dictionary containing boolean search filters [lowercase, negative]*)
- **metadata (list)** (*list of session metadata dict objects*)
- **key (str)** (*metadata key being searched for*)
- **v (str)** (*value of the corresponding key to be found*)

Returns: **hits (list)**

Return type: list of booleans indicating the found sessions to be updated in add_group_wrapper()

moseq2_viz.util.**get_metadata_path** (h5file)
Return path within h5 file that contains the kinect extraction metadata.

Parameters: **h5file (str)** (*path to h5 file.*)
Returns: **(str)**
Return type: path to acquistion metadata within h5 file.

moseq2_viz.util.**get_sorted_index** (index_file: str) → dict
Just return the sorted index from an index_file path.

Parameters: **index_file (str)** (*path to index file.*)
Returns: **sorted_ind (dict)**
Return type: dictionary of loaded sorted index file contents

moseq2_viz.util.**get_timestamps_from_h5** (h5file: str) → numpy.ndarray
Returns dict of timestamps from h5file.

Parameters: **h5file (str)** (*path to h5 file.*)
Returns: **(np.ndarray)**
Return type: timestamps from extraction within the h5file.

moseq2_viz.util.**h5_filepath_from_sorted** (sorted_index_entry: dict) → str
Gets the h5 extraction file path from a sorted index entry

Parameters: **sorted_index_entry (dict)** (*get filepath from sorted index.*)
Returns: **(str)**
Return type: a str containing the extraction filepath

moseq2_viz.util.**h5_to_dict** (h5file, path: str = '/') → dict
Load h5 dict contents to a dict variable.

Parameters:
- **h5file (str or h5py.File)** (*file path to the given h5 file or the h5 file handle*)
- **path (str)** (*path to the base dataset within the h5 file. Default: /*)

Returns: **out (dict)**
Return type: dictionary of all h5 contents

moseq2_viz.util.**load_changepoint_distribution** (cpfile)
Loads changepoint durations from given changepoints file *cpfile*.

Parameters: **cpfile (str)** (*Path to changepoints h5 file.*)
Returns: **(1d numpy array)**
Return type: Array of changepoint durations.

moseq2_viz.util.**load_timestamps** (timestamp_file, col=0)
Read timestamps from space delimited text file.

Parameters:
- **timestamp_file (str)** (*path to timestamp file*)
- **col (int)** (*column to load.*)

Returns: **ts (np.ndarray)**
Return type: loaded array of timestamps

moseq2_viz.util.**parse_index** (index: Union[str, dict]) → tuple
Load an index file, and use extraction UUIDs as entries in a sorted index.

Parameters: **index (str or dict)** (*if str, must be a path to the index file. If dict,*) – must be the unsorted index.
Returns: **index (dict)** (*loaded index file contents in a dictionary*) **sorted_index (dict)** (*index where the files have been sorted by UUID and pca_score path.*)

moseq2_viz.util.**read_yaml** (yaml_path: str)
Reads a given yaml file path into a dict object.

Parameters: **yaml_path (str)** (*path to yaml file to read.*)
Returns: **loaded (dict)**
Return type: loaded yaml file contents.

moseq2_viz.util.**recursive_find_h5s** (root_dir='/Users/aymanzeine/Desktop/moseq/moseq2-viz/docs', ext='.h5', yaml_string='{}.yaml')
Recursively find h5 files, along with yaml files with the same basename.

Parameters:
- **root_dir (str)** (*path to directory containing h5*)
- **ext (str)** (*extension to search for.*)
- **yaml_string (str)** (*yaml file format name.*)

Returns: **h5s (list)** (*list of paths to h5 files*) **dicts (list)** (*list of paths to metadata files*) **yamls (list)** (*list of paths to yaml files*)

moseq2_viz.util.**star**
Apply a function to a tuple of args, by expanding the tuple into each of the function's parameters. It is curried, which allows one to specify one argument at a time.

Parameters:
- **f (function)** (*a function that takes multiple arguments*)
- **args (tuple)** (: a tuple to expand into `f`)

Returns:
Return type: the output of `f`

moseq2_viz.util.**strided_app** (a, L, S)
Taking subarrays from numpy array given stride

Parameters:
- **a (np.array)** (*array to get subarrays from.*)
- **L (int)** (*window length.*)
- **S (int)** (*stride size.*)

Returns: **(np.ndarray)**
Return type: sliced subarrays

Visualization Module

Visualization model containing all plotting functions and some dependent data pre-processing helper functions.

moseq2_viz.viz.**clean_frames** (frames, medfilter_space=None, gaussfilter_space=None, tail_filter=None, tail_threshold=5)
Filters frames using spatial filters such as Median or Gaussian filters.

Parameters:
- **frames (3D numpy array)** (*frames to filter.*)
- **medfilter_space (list)** (*list of len()==1, must be odd. Median space filter kernel size.*)
- **gaussfilter_space (list)** (*list of len()==2. Gaussian space filter kernel size.*)
- **tail_filter (cv2.getStructuringElement)** (*structuringElement to filter out mouse tails.*)
- **tail_threshold (int)** (*filtering threshold value*)

Returns: **out (3D numpy array)**
Return type: filtered numpy array.

moseq2_viz.viz.**make_crowd_matrix** (slices, nexamples=50, pad=30, raw_size=512, 424, frame_path='frames', crop_size=80, 80, max_dur=60, min_dur=0, offset=50, 50, scale=1, center=False, rotate=False, min_height=10, legacy_jitter_fix=False, **kwargs)
Creates crowd movie video numpy array.

Parameters:
- **slices (np.ndarray)** (*video slices of specific syllable label*)
- **nexamples (int)** (*maximum number of mice to include in crowd_matrix video*)
- **pad (int)** (*number of frame padding in video*)
- **raw_size (tuple)** (*video dimensions.*)
- **frame_path (str)** (*variable to access frames in h5 file*)
- **crop_size (tuple)** (*mouse crop size*)
- **max_dur (int or None)** (*maximum syllable duration.*)
- **min_dur (int)** (*minimum syllable duration.*)
- **offset (tuple)** (*centroid offsets from cropped videos*)
- **scale (int)** (*mouse size scaling factor.*)
- **center (bool)** (*indicate whether mice are centered.*)
- **rotate (bool)** (*rotate mice to orient them.*)
- **min_height (int)** (*minimum max height from floor to use.*)
- **legacy_jitter_fix (bool)** (*whether to apply jitter fix for K1 camera.*)
- **kwargs (dict)** (*extra keyword arguments*)

Returns: **crowd_matrix (np.ndarray)**

Return type: crowd movie for a specific syllable.

`moseq2_viz.viz.`**`plot_cp_comparison`** (model_results, pc_cps, plot_all=False, best_model=None, bw_adjust=0.4)

Plot the duration distributions for model labels and principal component changepoints.

Parameters:
- **model_cps (dict)** (*Multiple parsed model results aggregated into a single dict.*)
- **pc_cps (1D np.array)** (*Computed PC changepoints*)
- **plot_all (bool)** (*Plot all model changepoints for all keys included in model_cps dict.*)
- **best_model (str)** (*key name to the model with the closest median syllable duration*)
- **bw_adjust (float)** (*fraction to modify bandwith of kernel density estimate. (lower = higher definition)*)

Returns: **fig (pyplot figure)** (*syllable usage ordered by frequency, 90% usage marked*) **ax (pyplot axis)** (*plotted scalar axis*)

`moseq2_viz.viz.`**`plot_mean_group_heatmap`** (`pdfs`, `groups`, `normalize=True`, `norm_color=<matplotlib.colors.LogNorm object>`)

Computes the overall group mean of the computed PDFs and plots them.

Parameters:
- **pdfs (list)** (*list of 2d probability density functions (heatmaps) describing mouse position.*)
- **groups (list)** (*list of groups to compute means and plot*)
- **normalize (bool)** (*flag to normalize the pdfs between 0-1*)
- **norm_color (mpl.colors Color Scheme or None)** (*indicates a color scheme to use when plotting heatmaps.*)

Returns: **fig (pyplot figure)**

Return type: plotted scalar scatter plot

`moseq2_viz.viz.`**`plot_syll_stats_with_sem`** (scalar_df, syll_info=None, sig_sylls=None, stat='usage', ordering='stat', max_sylls=40, groups=None, ctrl_group=None, exp_group=None, colors=None, join=False, figsize=10, 5)

Plots a line and/or point-plot of a given pre-computed syllable statistic (usage, duration, or speed), with a SEM error bar with respect to the group. This function is decorated with the check types function that will ensure that the inputted data configurations are safe to plot in matplotlib.

Parameters:

- **scalar_df (pd.DataFrame)** (*dataframe containing the statistical information about syllable data [usages, durs, etc.]*)
- **syll_info (dict)** (*dictionary of syllable numbers mapped to dict containing the label, description and crowd movie path.*) – If provided, will add x-tick markers with the labels for each syllable.
- **sig_sylls (1d list)** (*List of syllable numbers that are statistically significant to optionally mark in the graph.*)
- **stat (str)** (*choice of statistic to plot: either usage, duration, or speed*)
- **ordering (str, list, None)** ("stat" for sorting syllables by their average *stat*. "diff" for sorting syllables by) – the difference in *stat* between *exp_group* and *ctrl_group*. If a list, the user should supply the order of syllable labels to plot. If None, the original syllable IDs are used.
- **max_sylls (int)** (*maximum number of syllable to include in plot. default: 40*)
- **groups (list)** (*list of groups to include in plot. If groups=None, all groups will be plotted.*)
- **ctrl_group (str)** (*name of control group to base mutation sorting on.*)
- **exp_group (str)** (*name of experimental group to base mutation sorting on.*)
- **colors (list)** (*list of user-selected colors to represent the data*)
- **join (bool)** (*flag to connect points of pointplot*)
- **figsize (tuple)** (*tuple value of length = 2, representing (columns x rows) of the plotted figure dimensions*)

Returns: **fig (pyplot figure)** (*plotted scalar scatter plot*) **legend (pyplot legend)** (*figure legend*)

`moseq2_viz.viz.`**`plot_verbose_heatmap`** (`pdfs`, `sessions`, `groups`, `subjectNames`, `normalize=False`, `norm_color=<matplotlib.colors.LogNorm object>`)

Plots the PDF position heatmap for each session, titled with the group and subjectName.

Parameters:

- **pdfs (list)** (*list of 2d probability density functions (heatmaps) describing mouse position.*)
- **sessions (list)** (*list of sessions corresponding to the pdfs indices*)
- **groups (list)** (*list of groups corresponding to the pdfs indices*)
- **subjectNames (list)** (*list of subjectNames corresponding to the pdfs indices*)
- **normalize (bool)** (*flag to normalize the pdfs between 0-1*)
- **norm_color (mpl.colors Color Scheme or None)** (*indicates a color scheme to use when plotting heatmaps.*)

Returns: **fig (pyplot figure)**

Return type: plotted scalar scatter plot

`moseq2_viz.viz.`**`position_plot`** (scalar_df, centroid_vars=['centroid_x_mm', 'centroid_y_mm'], sort_vars=['SubjectName', 'uuid'], group_var='group', plt_kwargs={'linewidth': 1})

Creates a position summary graph that shows all the mice's centroid path throughout the respective sessions.

Parameters:
- **scalar_df (pandas DataFrame)** (*dataframe containing all scalar data*)
- **centroid_vars (list)** (*list of scalar variables to track mouse position*)
- **sort_vars (list)** (*list of variables to sort the dataframe by.*)
- **group_var (str)** (*groups df column to graph position plots for.*)
- **plt_kwargs (dict)** (*extra keyword arguments for plt.plot*)

Returns: **fig (pyplot figure)** (*matplotlib figure object*) **ax (pyplot axis)** (*matplotlib axis object*) **g (sns.FacetGrid)** (*FacetGrid object the data was plotted with*)

`moseq2_viz.viz.`**`save_fig`** (fig, output_file, suffix=None, **kwargs)

Convenience function for saving created/open matplotlib figures to PNG and PDF formats.

Parameters:
- **fig (pyplot.Figure)** (*open figure to save*)
- **output_file (str)** (*path to save figure to (without extension)*)
- **suffix (str)** (*string to append to the end of output_file*)
- **kwargs (dict)** (*dictionary containing additional figure saving parameters. (check plot-stats in wrappers.py)*)

`moseq2_viz.viz.`**`scalar_plot`** (scalar_df, sort_vars=['group', 'uuid'], group_var='group', show_scalars=['velocity_2d_mm', 'velocity_3d_mm', 'height_ave_mm', 'width_mm', 'length_mm'], headless=False, colors=None, plt_kwargs={'aspect': 0.8, 'height': 2})

Creates scatter plot of given scalar variables representing extraction results.

Parameters:
- **scalar_df (pandas DataFrame)** (*dataframe containing scalar data.*)
- **sort_vars (list)** (*list of variables to sort the dataframe by.*)
- **group_var (str)** (*groups scalar plots into separate distributions.*)
- **show_scalars (list)** (*list of scalar variables to plot.*)
- **headless (bool)** (*exclude head of dataframe from plot.*)
- **colors (list)** (*list of color strings to indicate groups*)
- **plt_kwargs (dict)** (*extra arguments for the swarmplot*)

Returns: **fig (pyplot figure)** (*plotted scalar scatter plot*) **ax (pyplot axis)** (*plotted scalar axis*)

Subpackages

moseq2_viz.helpers package

Helpers - Wrappers Module

Wrapper functions for all functionality included in MoSeq2-Viz that is accessible via CLI or GUI. Each wrapper function executes the functionality from end-to-end given it's dependency parameters are inputted. (See CLI Click parameters)

`moseq2_viz.helpers.wrappers.`**`add_group_wrapper`** (index_file, config_data)

Given a pre-specified key and value, the index file will be updated with the respective found keys and values.

Parameters:
- **index_file (str)** (*path to index file*)
- **config_data (dict)** (*dictionary containing the user specified keys and values*)

`moseq2_viz.helpers.wrappers.`**`copy_h5_metadata_to_yaml_wrapper`** (input_dir)

Copy h5 metadata dictionary contents into the respective file's yaml file.

Parameters: **input_dir (str)** (*path to directory that contains h5 files.*)

moseq2_viz.helpers.wrappers.**get_best_fit_model_wrapper** (model_dir, cp_file, output_file, plot_all=False, ext='p', fps=30, objective='duration')

Given a directory containing multiple models trained on different kappa values, finds the model with the closest median syllable duration to the PC changepoints. Function also graphs the distributions of the best fit model and PC changepoints

Parameters:

- **model_dir (str)** (*Path to directory containing multiple models.*)
- **cp_file (str)** (*Path to PCA changepoints*)
- **output_file (str)** (*Path to file to save figure to.*)
- **plot_all (bool)** (*Plot all model changepoint distributions.*)
- **ext (str)** (*File extension to search for models with*)
- **fps (int)** (*Frames per second*)
- **objective (str)** (*can be either duration or jsd. The objective finds the best model*) – based on either median changepoint durations or the jensen-shannon divergence beteween changepoint duration distributions

Returns: **best_model_info (dict)** (*Dict containing the best model info with respect to given objective.*) **fig (pyplot figure)** (*syllable usage ordered by frequency, 90% usage marked*)

moseq2_viz.helpers.wrappers.**init_wrapper_function** (index_file=None, output_dir=None, output_file=None)

Helper function that will optionally load the index file and a trained model given their respective paths. The function will also create any output directories given path to the output file or directory.

Parameters:

- **index_file (str)** (*path to index file to load.*)
- **output_dir (str)** (*path to directory to save plots in.*)
- **output_file (str)** (*path to saved figures.*)

Returns: **index (dict)** (*loaded index file dictionary*) **sorted_index (dict)** (*OrderedDict object representing a sorted version of index*)

moseq2_viz.helpers.wrappers.**make_crowd_movies_wrapper** (index_file, model_path, output_dir, config_data)

Wrapper function to create crowd movie videos and write them to individual files depicting respective syllable labels.

Parameters:

- **index_file (str)** (*path to index file*)
- **model_path (str)** (*path to trained model.*)
- **output_dir (str)** (*directory to store crowd movies in.*)
- **config_data (dict)** (*dictionary conataining all the necessary parameters to generate the crowd movies.*) –

  E.g.: max_syllable: Maximum number of syllables to generate crowd movies for.

  max_example: Maximum number of mouse examples to include in each crowd movie specific_syllable: Set to a syllable number to only generate crowd movies of that syllable.

  if value is None, command will generate crowd movies for all syllables with # <= max_syllable.

  separate_by: ['default', 'groups', 'sessions', 'subjects']; If separate_by != 'default', the command

  will generate a separate crowd movie for each selected grouping per syllable. Resulting in (ngroups * max_syllable) movies.

Returns: **cm_paths (dict)**

Return type: Dictionary of syllables and their generated crowd movie paths

`moseq2_viz.helpers.wrappers.`**`plot_mean_group_position_pdf_wrapper`**`(index_file, output_file, normalize=False, norm_color=<matplotlib.colors.LogNorm object>)`

Computes the position PDF for each session, averages the PDFs within each group, and plots the averaged PDFs.

Parameters:
- **index_file (str)** (*path to index file.*)
- **output_file (str)** (*filename for the group heatmap graph.*)
- **normalize (bool)** (*normalize the PDF so that min and max values range from 0-1.*)
- **norm_color (mpl.colors Color Scheme or None)** (*indicates a color scheme to use when plotting heatmaps.*)

Returns: **fig (pyplot figure)**

Return type: figure to graph in Jupyter Notebook.

`moseq2_viz.helpers.wrappers.`**`plot_scalar_summary_wrapper`**`(index_file, output_file, groupby='group', colors=None, show_scalars=['velocity_2d_mm', 'velocity_3d_mm', 'height_ave_mm', 'width_mm', 'length_mm'])`

Creates a scalar summary graph.

Parameters:
- **index_file (str)** (*path to index file.*)
- **output_file (str)** (*path to save graphs*)
- **groupby (str)** (*scalar_df column to group sessions by when graphing scalar and position summaries*)
- **colors (list)** (*list of colors to serve as the palette in the scalar summary*)
- **show_scalars (list)** (*list of scalar variables to plot; variable names must equal columns in the scalar_df DataFrame.*)

Returns: **scalar_df (pandas DataFrame)**

Return type: df containing scalar data per session uuid.

`moseq2_viz.helpers.wrappers.`**`plot_syllable_stat_wrapper`**`(model_fit, index_file, output_file, stat='usage', sort=True, count='usage', group=None, max_syllable=40, ordering=None, ctrl_group=None, exp_group=None, colors=None, figsize=10, 5)`

Graph given syllable statistic from a trained AR-HMM model.

Parameters:
- **model_fit (str)** (*path to trained model file.*)
- **index_file (str)** (*path to index file.*)
- **output_file (str)** (*filename for syllable usage graph.*)
- **stat (str)** (*syllable statistic to plot. It can be any of the scalars (i.e., velocity_2d_mm) in addition*) – to: 'usage' and 'duration'
- **sort (bool)** (sort syllables by *count*.)
- **count (str)** (*method to sort syllables: 'usage' or 'frames'.*)
- **group (tuple, list, None)** (*tuple or list of groups to include in usage plot. (None to graph all groups)*)
- **max_syllable (int)** (*maximum number of syllables to plot.*)
- **ordering (list, range, str, None)** (*order to list syllables. Default is None to graph syllables [0-max_syllable)*) – in numerical order. Setting ordering to "diff" will sort syllables based the difference in "stat" between ctrl_group and exp_group. "stat" to order syllables by stat values.
- **ctrl_group (str)** (*Control group to graph.*)
- **exp_group (str)** (*Experimental group to compare with control group.*)
- **colors (list)** (*list of colors to serve as the sns palette in the scalar summary. If None, default colors are used.*)
- **figsize (tuple)** (*tuple value of length = 2, representing (columns x rows) of the plotted figure dimensions*)

Returns: **fig (pyplot figure)**

Return type: figure to show in Jupyter Notebook.

`moseq2_viz.helpers.wrappers.`**`plot_transition_graph_wrapper`** (index_file, model_fit, output_file, config_data)

Wrapper function to plot transition graphs.

Parameters:
- **index_file (str)** (*path to index file*)
- **model_fit (str)** (*path to trained model.*)
- **output_file (str)** (*filename for syllable usage graph.*)
- **config_data (dict)** (*dictionary containing the user specified keys and values*)

Returns: **plt (pyplot figure)**

Return type: graph to show in Jupyter Notebook.

`moseq2_viz.helpers.wrappers.`**`plot_verbose_pdfs_wrapper`** (`index_file`, `output_file`, `normalize=False`, `norm_color=<matplotlib.colors.LogNorm object>`)

Wrapper function that computes the PDF for the mouse position for each session in the index file. Will plot each session's heatmap with a "SessionName: Group"-like title.

Parameters:
- **index_file (str)** (*path to index file.*)
- **output_file (str)** (*filename for the verbose heatmap graph.*)
- **normalize (bool)** (*normalize the PDF so that min and max values range from 0-1.*)
- **norm_color (mpl.colors Color Scheme or None)** (*indicates a color scheme to use when plotting heatmaps.*)

Returns: **fig (pyplot figure)**

Return type: figure to graph in Jupyter Notebook.

moseq2_viz.info package

Info - Utilities Module

Utility functions for computing syllable usage entropy, and syllable transition entropy rate. These can be used for measuring modeling model performance and group separability.

`moseq2_viz.info.util.`**`entropy`** (labels, truncate_syllable=40, smoothing=1.0, relabel_by='usage')

Computes syllable usage entropy, base 2.

Parameters:
- **labels (list of np.ndarray)** (*list of predicted syllable label arrays from a group of sessions*)
- **truncate_syllable (int)** (*truncate list of relabeled syllables*)
- **smoothing (float)** (*a constant added to label usages before normalization*)
- **relabel_by (str)** (*mode to relabel predicted labels. Either 'usage', 'frames', or None.*)

Returns: **ent (list)**

Return type: list of entropies for each session.

`moseq2_viz.info.util.`**`entropy_rate`** (labels, truncate_syllable=40, normalize='bigram', smoothing=1.0, tm_smoothing=1.0, relabel_by='usage')

Computes entropy rate, base 2 using provided syllable labels. If syllable labels have not been re-labeled by usage, this function will do so.

Parameters:
- **labels (list or np.ndarray)** (*a list of label arrays, where each entry in the list*) – is an array of labels for one subject.
- **truncate_syllable (int)** (*maximum number of labels to keep for this calculation.*)
- **normalize (str)** (*the type of transition matrix normalization to perform. Options*) – are: 'bigram', 'rows', or 'columns'.
- **smoothing (float)** (*a constant added to label usages before normalization*)
- **tm_smoothing (float)** (*a constant added to label transtition counts before normalization.*)
- **relabel_by (str)** (*how to re-order labels. Options are: 'usage', 'frames', or None.*)

Returns: **ent (list)**

Return type: list of entropy rates per syllable label

`moseq2_viz.info.util.`**`transition_entropy`** (labels, tm_smoothing=0, truncate_syllable=40, transition_type='incoming', relabel_by='usage')

Computes directional syllable transition entropy. Based on whether the given transition_type is 'incoming' or

or 'outgoing', the function will compute the respective transition entropy.

Parameters:
- **labels (list or np.ndarray)** (*a list of label arrays, where each entry in the list*) – is an array of labels for one subject.
- **tm_smoothing (float)** (*a constant added to label transtition counts before normalization.*)
- **truncate_syllable (int)** (*maximum number of labels to keep for this calculation.*)
- **transition_type (str)** (*can be either "incoming" or "outgoing" to compute the entropy of each*) – incoming or outgoing syllable transition.
- **relabel_by (str)** (*how to re-order labels. Options are: 'usage', 'frames', or None.*)

Returns: **entropies (list of np.ndarra)** – each mouse and syllable.

Return type: a list of transition entropies (either incoming or outgoing) for

moseq2_viz.io package

IO - Video Module

Helper functions for handling crowd movie file writing and video metadata maintenance.

`moseq2_viz.io.video.`**`check_video_parameters`** (index: dict) → dict

Iterates through each extraction parameter file to verify extraction parameters were the same. If they weren't this function raises a RuntimeError.

Parameters: **index (dict)** (a *sorted_index* dictionary of extraction parameters.)

Returns: **vid_parameters (dict)**

Return type: a dictionary with a subset of the used extraction parameters.

`moseq2_viz.io.video.`**`write_crowd_movie_info_file`** (model_path, model_fit, index_file, output_dir)

Creates an info.yaml file in the crowd movie directory that holds model training parameters. This file helps identify the conditions from which the crowd movies were generated.

Parameters:

- **model_path (str)** (*path to model used to generate movies*)
- **model_fit (dict)** (*loaded ARHMM dict*)
- **index_file (str)** (*path to index file used with model*)
- **output_dir (str)** (*path to crowd movies directory to store file in.*)

`moseq2_viz.io.video.`**`write_crowd_movies`** (sorted_index, config_data, ordering, labels, label_uuids, output_dir)

Creates syllable slices for crowd movies and writes them to files.

Parameters:

- **sorted_index (dict)** (*dictionary of sorted index data.*)
- **config_data (dict)** (*dictionary of visualization parameters.*)
- **filename_format (str)** (*string format that denotes the saved crowd movie file names.*)
- **ordering (list)** (*ordering for the new mapping of the relabeled syllable usages.*)
- **labels (numpy ndarray)** (*list of syllable usages*)
- **label_uuids (list)** (*list of session uuids each series of labels belongs to.*)
- **output_dir (str)** (*path directory where all the movies are written.*)

`moseq2_viz.io.video.`**`write_frames_preview`** (filename, frames=array([], dtype=float64), threads=6, fps=30, pixel_format='rgb24', codec='h264', slices=24, slicecrc=1, frame_size=None, depth_min=0, depth_max=80, get_cmd=False, cmap='jet', text=None, text_scale=1, text_thickness=2, pipe=None, close_pipe=True, progress_bar=True, **kwargs)

Writes out a false-colored mp4 video. [Duplicate from moseq2-extract]

Parameters:

- **filename (str)** (*path to write output crowd movie file*)
- **frames (3D numpy array)** (*num_frames * r * c*)
- **threads (int)** (*number of threads to write file*)
- **fps (int)** (*frames per second*)
- **pixel_format (str)** (*ffmpeg image formatting flag.*)
- **codec (str)** (*ffmpeg image encoding flag.*)
- **slices (int)** (*number of slices per thread.*)
- **slicecrc (int)** (*check integrity of slices.*)
- **frame_size (tuple)** (*image dimensions*)
- **depth_min (int)** (*minimum mouse distance from bucket floor*)
- **depth_max (int)** (*maximum mouse distance from bucket floor*)
- **get_cmd (bool)** (*return ffmpeg command instead of executing the command in python.*)
- **cmap (str)** (*color map selection.*)
- **text (range(num_frames)** (*display frame number in output video.*)
- **text_scale (int)** (*text size.*)
- **text_thickness (int)** (*text thickness.*)
- **pipe (subProcess.Pipe object)** (*if not None, indicates that there are more frames to be written.*)
- **close_pipe (bool)** (*indicates whether video is done writing, and to close pipe to file-stream.*)
- **progress_bar (bool)** (*display progress bar.*)
- **kwargs (dict)** (*extra keyword arguments*)

Returns: **pipe (subProcess.Pipe object)**

Return type: if there are more slices/chunks to write to, otherwise the path to the movie.

moseq2_viz.model package

Model - Dist Module

Utility functions for estimating "behavioral distance" AKA model state similarity analysis.

`moseq2_viz.model.dist.`**`get_behavioral_distance`** (index, model_file, whiten='all', distances=['ar[init]', 'scalars'], max_syllable=None, resample_idx=- 1, dist_options={}, sort_labels_by_usage=True, count='usage')

Computes the behavioral distance (square) matrices with respect to a predefined set of variables.

Parameters:
- **index (str)** (*Path to index file*)
- **model_file (str)** (*Path to trained model*)
- **whiten (str)** (*Indicates whether to whiten all PCs at once or each one at a time. Options = ['all', 'each']*)
- **distances (list or str)** (*type of distance(s) to compute.*) – Available options = ['scalars', 'ar[init]', 'ar[dtw]', 'pca[dtw]', 'combined']
- **max_syllable (int)** (*Maximum number of syllables/AR matrices to include in analysis*)
- **resample_idx (int)** (*Indicates the parsing method according to the shape of the labels array.*)
- **dist_options (dict)** (*Dictionary holding each distance operations configurable parameters*)
- **sort_labels_by_usage (bool)** (*Indicates whether to relabel syllables by count ordering*)
- **count (str)** (*Indicates what ordering to relabel syllables by. Options = ['usage', 'frames']*)

Returns: **dist_dict (dict)**

Return type: Dictionary containing all computed behavioral square distance matrices

`moseq2_viz.model.dist.`**`get_behavioral_distance_ar`** (ar_mat, init_point=None, sim_points=10, max_syllable=40, dist='correlation', parallel=False)

Computes behavioral distance with respect to the model's AutoRegressive matrices. Affords either AR trajectory correlation distance, or computing dynamically time-warped trajectory distances.

Parameters:
- **ar_mat (3D numpy array)** (*Trained model AutoRegressive matrices; shape=(max_syllable, npcs, npcs*nlags+1)*)
- **init_point (list)** (*Initial values as a reference point for distance estimation*)
- **sim_points (int)** (*Number of AR trajectories to simulate*)
- **max_syllable (int)** (*Max number of syllables included in the analysis. Should be equal to ar_mat.shape[0]*)
- **dist (str)** (*Distance operation to compute. Either 'correlation' or 'dtw'.*)
- **parallel (bool)** (*Use multiprocessing to compute dtw distances.*)

Returns: **ar_dist (2D numpy array)** (*Computed AR trajectory distances for each AR matrix/model state.*) *shape=(max_syllable, max_syllable)*

`moseq2_viz.model.dist.`**`get_init_points`** (pca_scores, model_labels, max_syllable=40, nlags=3, npcs=10)

Compute initial AR trajectories based on a cumulative average of lagged-PC Scores over nlags.

Parameters:
- **pca_scores (2D numpy array)** (*Loaded PCA Scores. Shape=(npcs, nsamples)*)
- **model_labels (2D list)** (*list of 1D numpy arrays of relabeled/sorted syllable labels*)
- **max_syllable (int)** (*Maximum number of syllables to include.*)
- **nlags (int)** (*Number of lagged frames.*)
- **npcs (int)** (*Number of PCs to use in computation.*)

Returns: **syll_average (list)** – PC scores array. Shape = (max_syllables, nlags*2 +1, npcs)

Return type: List containing 2D np arrays of average syllable trajectories over a nlag-strided

`moseq2_viz.model.dist.`**`reformat_dtw_distances`** (full_mat, nsyllables, rescale=True)

Reduce full (max states) dynamically time-warped PC Score distance matrices to only include dimensions for a total of nsyllables. Formatting the 3D matrix (full_mat) to 2D to show the correlation distances from each state pair.

Parameters:
- **full_mat (3D np.ndarray)** (*DTW distance matrices for all model states/syllables.*)
- **nsyllables (int)** (*Number of syllables to include in truncated DTW distance matrix.*)
- **rescale (bool)** (*Rescale truncated dtw-distance matrices to match output distribution.*)

Returns: **rmat (2D np array)**

Return type: Reformatted-Truncated DTW Distance Matrix; shape = (nsyllables, nsyllables)

Model - Label Utilities Module

Model - Utilities Module

Utility functions specifically responsible for handling model data during pre and post processing.

`moseq2_viz.model.util.`**`add_duration_column`** (scalar_df)
Adds syllable duration column to scalar dataframe if it also contains syllable labels.

Parameters: **scalar_df (pd.DataFrame)** (*Merged Syllable Stat + Scalar DataFrame object.*)

Returns: **scalar_df (pd.DataFrame)**

Return type: Same DataFrame with a new column titled "duration".

`moseq2_viz.model.util.`**`compute_behavioral_statistics`** (scalar_df, groupby=['group', 'uuid'], count='usage', fps=30, usage_normalization=True, syllable_key='labels (usage sort)')
Computes syllable statistics merged with the inputted scalar features.

Parameters:
- **scalar_df (pd.DataFrame)** (*Scalar measuresments for full dataset, including metadata for all the sessions.*) – Outputted via scalars_to_dataframe().
- **groupby (list of strings)** (*list of columns to run the pandas groupby() on the scalar_df.*)
- **count (str)** (*indicates how to determine mean usage calculation. either 'usage' (default), or 'frames'*)
- **fps (int)** (*frames per second that the data was acquired in.*)
- **usage_normalization (bool)** (*indicates whether to normalize syllable usages by the value counts.*)
- **syllable_key (str)** (*column to rename to "syllable" for convenient referencing later on.*)

Returns: **features (pd.DataFrame)**

Return type: full feature Dataframe with scalars, metadata, and syllable statistics.

`moseq2_viz.model.util.`**`compute_syllable_explained_variance`** (model, n_explained=99)
Computes the maximum number of syllables to include that explain *n_explained* percent of all frames in the dataset.

Parameters:
- **model (dict)** (*ARHMM results dict*)
- **n_explained (int)** (*explained variance percentage threshold*)

Returns: **max_sylls (int)**

Return type: Number of syllables that explain the given percentage of the variance

`moseq2_viz.model.util.`**`compute_syllable_onset`** (labels)
Computes the onset index of the each syllable label in a Series.

Parameters: **labels (list or dict)** (*label sequences loaded from a model fit*)

Returns: **onsets (2D np.array)**

Return type: onset indices for each syllable for the given sessions.

`moseq2_viz.model.util.`**`get_Xy_values`** (stat_means, unique_groups, stat='usage')

Computes the syllable or scalar mean statistics for each session, stored in X. Computes and corresponding

mapped group name value for each of the sessions to be tracked when plotting the values in the embedding steps.

Parameters:
- **stat_means (pd DataFrame)** (*Dataframe of syllable or session-scalar mean statistics*)
- **unique_groups (1D list)** (*list of unique groups in the syll_means dataframe.*)
- **stat (str or list)** (*statistic column(s) to read from the syll_means df.*)

Returns: **X (2D np.array)** (*mean syllable or scalar statistics for each session. (nsessions x nsyllables)*) **y (1D list)** (*list of group names corresponding to each row in X.*) **mapping (dict)** (*dictionary conataining mappings from group string to integer for later embedding.*) **rev_mapping (dict)** (*inverse mapping dict to retrieve the group names given their mapped integer value.*)

`moseq2_viz.model.util.`**`get_best_fit`** (cp_path, model_results)

Returns the model with the closest median syllable duration and closest duration distribution to the PCA changepoints.

Parameters:
- **cp_path (str)** (*Path to PCA Changepoints h5 file.*)
- **model_results (dict)** (*dict of pairs of model names paired with dict containing their respective changepoints.*)

Returns: **info (dict)** (*information about the best-fit models.*) **pca_cps (1D array)** (*pc score changepoint durations.*)

`moseq2_viz.model.util.`**`get_mouse_syllable_slices`** (syllable: int, labels: numpy.ndarray) → Iterator[slice]

Return a list containing slices of *syllable* indices for a mouse.

Parameters:
- **syllable (list)** (*list of syllables to get slices from.*)
- **labels (np.ndarrary)** (*list of label predictions for each session.*)

Returns: **slices (list)**

Return type: list of syllable label slices; e.g. [slice(3, 6, None), slice(9, 12, None)]

`moseq2_viz.model.util.`**`get_normalized_syllable_usages`** (model_data, max_syllable=100, count='usage')

Computes syllable usages and normalizes to sum to 1. Returns a 1D array of their corresponding usage values.

Parameters:
- **model_data (dict)** (*dict object of modeling results*)
- **max_syllable (int)** (*number of syllables to compute the mean usage for.*)
- **count (str)** (*option for whether to count syllable usages; by 'frames', or 'usage'.*)

Returns: **syllable_usages (1D np array)**

Return type: array of sorted syllable usages for all syllables in model

`moseq2_viz.model.util.`**`get_syllable_slices`**

Get the indices that correspond to a specific syllable for each animal in a modeling run.

Parameters:
- **syllable (int)** (*syllable number to get slices of.*)
- **labels (np.ndarrary)** (*list of label predictions for each session.*)
- **label_uuids (list)** (*list of uuid keys corresponding to each session.*)
- **index (dict)** (*index file contents contained in a dict.*)
- **trim_nans (bool)** (*flag to use the pca scores file for removing time points that contain NaNs.*) – Only use if you have not already trimmed NaNs previously and need to.

Returns: **syllable_slices (list)** (a list of indices for *syllable* in the *labels* array. Each item in the list) *is a tuple of (slice, uuid, h5_file).*

`moseq2_viz.model.util.`**`get_syllable_statistics`** (data, fill_value=- 5, max_syllable=100, count='usage')

Compute the usage and duration statistics from a set of model labels

Parameters:
- **data (list of np.array of ints)** (*labels loaded from a model fit.*)
- **fill_value (int)** (*lagged label values in the labels array to remove.*)
- **max_syllable (int)** (*maximum syllable to consider.*)
- **count (str)** (*how to count syllable usage, either by number of emissions (usage), or number of frames (frames).*)

Returns: **usages (OrderedDict)** (*default dictionary of usages*) **durations (OrderedDict)** (*default dictionary of durations*)

`moseq2_viz.model.util.`**`get_syllable_usages`** (data, max_syllable=100, count='usage')

Computes syllable usages for relabeled syllable labels.

Parameters:
- **data (2d list)** (*list of syllable frame-labels for each session.*)
- **max_syllable (int)** (*maximum number of syllables to compute usages for.*)
- **count (str)** (*method to relabel the syllable usages. Either by 'usage' or by 'frames'*)

Returns: **usages (dict)**

Return type: dict object that contains usage frequency information.

`moseq2_viz.model.util.`**`labels_to_changepoints`** (labels, fs=30)

Compute syllable durations and combine into a "changepoint" distribution.

Parameters:
- **labels (list of np.ndarray of ints)** (*labels loaded from a model fit.*)
- **fs (float)** (*sampling rate of camera.*)

Returns: **cp_dist (np.ndarray of floats)**

Return type: list of block durations per element in labels list.

`moseq2_viz.model.util.`**`make_separate_crowd_movies`** (config_data, sorted_index, group_keys, label_dict, output_dir, ordering, sessions=False)

Helper function that writes syllable crowd movies for each given grouping found in group_keys, and returns

a dictionary with session/group name keys paired with paths to their respective generated crowd movies.

Parameters:
- **config_data (dict)** (*Loaded crowd movie writing configuration parameters.*)
- **sorted_index (dict)** (*Loaded index file and sorted files in list.*)
- **group_keys (dict)** (*Dict of group/session name keys paired with UUIDS to match with labels.*)
- **label_dict (dict)** (*dict of corresponding session UUIDs for all sessions included in labels.*)
- **output_dir (str)** (*Path to output directory to save crowd movies in.*)
- **ordering (list)** (*ordering for the new mapping of the relabeled syllable usages.*)
- **sessions (bool)** (*indicates whether session crowd movies are being generated.*)

Returns: **cm_paths (dict)**

Return type: group/session name keys paired with paths to their respectively generated syllable crowd movies.

`moseq2_viz.model.util.`**`merge_models`** (model_dir, ext='p', count='usage', force_merge=False, cost_function='ar_norm')

WARNING: THIS IS EXPERIMENTAL. USE AT YOUR OWN RISK. Merges model states by using the Hungarian Algorithm: a minimum distance state matching algorithm. User inputs a directory containing models to merge, (and the name of the latest-trained model) to match other model states to.

Parameters:
- **model_dir (str)** (*path to directory containing all the models to merge.*)
- **ext (str)** (*model extension to search for.*)
- **count (str)** (*method to compute usages 'usage' or 'frames'.*)
- **force_merge (bool)** (*whether or not to force a merge. Keeping this false will*) – protect you from merging models trained with different kappa values.
- **cost_function (str)** (*either ar_norm or label - if ar_norm, uses the ar matrices*) – to find the most similar syllables. If label, finds the syllable labels that are most overlapping

Returns: **model_data (dict)**

Return type: a dictionary containing all the new keys and state-matched labels.

moseq2_viz.model.util.**normalize_pcs** (pca_scores: dict, method: str = 'zscore') → dict

Normalize PC scores. Options are: demean, zscore, ind-zscore. zscore: standardize pc scores using all data demean: subtract the mean from each score. ind-zscore: zscore each session independently

Parameters:
- **pca_scores (dict)** (*dict of uuid to PC-scores key-value pairs.*)
- **method (str)** (*the type of normalization to perform (demean, zscore, ind-zscore)*)

Returns: **norm_scores (dict)**

Return type: a dictionary of normalized PC scores.

moseq2_viz.model.util.**normalize_usages** (usage_dict)

Normalizes syllable usages to frequency values from [0,1] instead of total counts.

Parameters: **usage_dict (dict)** (*dictionary containing syllable label keys pointing to total counts.*)

Returns: **usage_dict (dict)**

Return type: dictionary containing syllable label keys pointing to usage frequencies.

moseq2_viz.model.util.**parse_batch_modeling** (filename)

Reads model parameter scan training results into a single dictionary.

Parameters: **filename (str)** (*path to h5 manifest file containing all the model results.*)

Returns: **results_dict (dict)** – concatenated into a single list. Maintaining the original structure as though it was a single model's results.

Return type: dictionary containing each model's training results,

moseq2_viz.model.util.**parse_model_results** (model_obj, restart_idx=0, resample_idx=- 1, map_uuid_to_keys: bool = False, sort_labels_by_usage: bool = False, count: str = 'usage') → dict

Reads model file and returns dictionary containing modeled results and some metadata.

Parameters:
- **model_obj (str or results returned from joblib.load)** (*path to the model fit or a loaded model fit*)
- **restart_idx (int)** (*Select which model restart to load. (Only change for models with multiple restarts used)*)
- **resample_idx (int)** (*parameter used to select labels from a specific sampling iteration. Default is the last iteration (-1)*)
- **map_uuid_to_keys (bool)** (*flag to create a label dictionary where each key->value pair*) – contains the uuid and the labels for that session.
- **sort_labels_by_usage (bool)** (*sort and re-assign labels by their usages.*)
- **count (str)** (*how to count syllable usage, either by number of emissions (usage),*) – or number of frames (frames).

Returns: **output_dict (dict)**

Return type: dictionary with labels and model parameters

moseq2_viz.model.util.**prepare_model_dataframe** (model_path, pca_path)

Creates a dataframe from syllable labels to be aligned with scalars.

Parameters:
- **model_path (str)** (*path to model to load label arrays from*)
- **pca_path (str)** (*path to pca_scores.h5 file.*)

Returns: **_df (pd.DataFrame)**

Return type: DataFrame object of timestamp aligned syllable label information.

`moseq2_viz.model.util.`**`relabel_by_usage`** (labels, fill_value=- 5, count='usage')

Resort model labels by their usages.

Parameters:
- **labels (list or dict)** (*label sequences loaded from a model fit*)
- **fill_value (int)** (*value prepended to modeling results to account for nlags*)
- **count (str)** (*how to count syllable usage, either by number of emissions (usage), or number of frames (frames)*)

Returns: **labels (list or dict)** (*label sequences sorted by usage*) **sorting (list)** (*the new label sorting. The index corresponds to the new label,*) – while the value corresponds to the old label.

`moseq2_viz.model.util.`**`retrieve_pcs_from_slices`** (slices, pca_scores, max_dur=60, min_dur=3, max_samples=100, npcs=10, subsampling=None, remove_offset=False, **kwargs)

Subsample Principal components from syllable slices

Parameters:
- **slices (np.ndarray)** (*syllable slices or subarrays*)
- **pca_scores (np.ndarray)** (*PC scores for respective session.*)
- **max_dur (int)** (*maximum syllable length.*)
- **min_dur (int)** (*minimum syllable length.*)
- **max_samples (int)** (*maximum number of samples to retrieve.*)
- **npcs (int)** (*number of pcs to use.*)
- **subsampling (int)** (*number of syllable subsamples (defined through KMeans clustering).*)
- **remove_offset (bool)** (*indicate whether to remove initial offset from each PC score.*)
- **kwargs (dict)** (*used to capture certain arguments in other parts of the codebase.*)

Returns: **syllable_matrix (np.ndarray)**

Return type: 3D matrix of subsampled PC projected syllable slices.

`moseq2_viz.model.util.`**`simulate_ar_trajectory`** (ar_mat, init_points=None, sim_points=100)

Simulate auto-regressive trajectory matrices from a set of initalized points.

Parameters:
- **ar_mat (3D np.ndarray)** (*numpy array representing the autoregressive matrix of each model state.*)
- **init_points (2D np.ndarray)** (*pre-initialzed array (npcs x nlags) in shape*)
- **sim_points (int)** (*number of time points to simulate.*)

Returns: **sim_mat[nlags**

Return type:] simulated AR trajectories excluding lagged values.

`moseq2_viz.model.util.`**`sort_batch_results`** (data, averaging=True, filenames=None, **kwargs)

Sort modeling results from batch/parameter scan.

Parameters:
- **data (np.ndarray)** (*model AR-matrices.*)
- **averaging (bool)** (*return an average of all the model AR-matrices.*)
- **filenames (list)** (*list of paths to fit models.*)
- **kwargs (dict)** (*dict of extra keyword arguments.*)

Returns: **new_matrix (np.ndarray)** (*either average of all AR-matrices, or top sorted matrix*) **param_dict (dict)** (*model parameter dict*) **filename_index (list)** (*list of filenames associated with each model.*)

`moseq2_viz.model.util.`**`sort_syllables_by_stat`** (complete_df, stat='usage', max_sylls=None)

Computes the sorted ordering of the given DataFrame with respect to the chosen stat.

Parameters:

- **complete_df (pd.DataFrame)** (*DataFrame containing the statistical information about syllable data [usages, durs, etc.]*)
- **stat (str)** (*choice of statistic to order syllables by: {usage, duration, speed}.*)
- **max_sylls (int or None)** (*maximum number of syllables to include in ordering*)

Returns: **ordering (list)** (*list of sorted syllables by stat.*) **relabel_mapping (dict)** (*a dict with key-value pairs {old_ordering: new_ordering}.*)

`moseq2_viz.model.util.`**`sort_syllables_by_stat_difference`** (complete_df, ctrl_group, exp_group, max_sylls=None, stat='usage')

Computes the syllable ordering for the difference of the inputted groups (exp - ctrl). The sorted result will yield an array will indices depicting the largest positive (upregulated) difference between exp and ctrl groups on the left, and vice versa on the right.

Parameters:

- **complete_df (pd.DataFrame)** (*dataframe containing the statistical information about syllable data [usages, durs, etc.]*)
- **ctrl_group (str)** (*Control group.*)
- **exp_group (str)** (*Experimental group.*)
- **max_sylls (int)** (*maximum number of syllables to include in ordering.*)
- **stat (str)** (*choice of statistic to order mutations by: {usage, duration, speed}.*)

Returns: **ordering (list)**

Return type: list of array indices for the new label mapping.

`moseq2_viz.model.util.`**`syll_duration`** (labels: numpy.ndarray) → numpy.ndarray

Computes the duration of each syllable.

Parameters: **labels (np.ndarray)** (*array of syllable labels for a mouse.*)

Returns: **durations (np.ndarray)**

Return type: array of syllable durations.

`moseq2_viz.model.util.`**`syll_id`** (labels: numpy.ndarray) → numpy.ndarray

Returns the syllable label at each onset of a syllable transition.

Parameters: **labels (np.ndarray)** (*array of syllable labels for a mouse.*)

Returns: **labels[onsets] (np.ndarray)**

Return type: an array of compressed labels.

`moseq2_viz.model.util.`**`syll_onset`** (labels: numpy.ndarray) → numpy.ndarray

Finds indices of syllable onsets.

Parameters: **labels (np.ndarray)** (*array of syllable labels for a mouse.*)

Returns: **indices (np.ndarray)**

Return type: an array of indices denoting the beginning of each syllables.

`moseq2_viz.model.util.`**`syllable_slices_from_dict`**

Reads dictionary of syllable labels, and returning a dict of syllable slices.

Parameters:
- **syllable (list)** (*list of syllables to get slices from.*)
- **labels (np.ndarrary)** (*list of label predictions for each session.*)
- **index (dict)** (*index file contents contained in a dict.*)
- **filter_nans (bool)** (*replace NaN values with 0.*)

Returns: **vals (dict)**

Return type: key-value pairs of syllable slices per session uuid.

`moseq2_viz.model.util.`**`whiten_pcs`** (pca_scores, method='all', center=True)

Whiten PC scores using Cholesky whitening.

Parameters:
- **pca_scores (dict)** (*dictionary where values are pca_scores (2d np arrays)*)
- **method (str)** (*'all' to whiten using the covariance estimated from all keys, or 'each' to whiten each separately*)
- **center (bool)** (*whether or not to center the data*)

Returns: **whitened_scores (dict)**

Return type: dictionary of whitened pc scores

moseq2_viz.scalars package

Scalars - Utilities Module

Utility functions responsible for handling all scalar data-related operations.

`moseq2_viz.scalars.util.`**`compute_all_pdf_data`** (scalar_df, normalize=False, centroid_vars=['centroid_x_mm', 'centroid_y_mm'], key='SubjectName', bins=20)

Computes a position PDF for all sessions and returns the pdfs with corresponding lists of

groups, session uuids, and subjectNames.

Parameters:
- **scalar_df (pd.DataFrame)** (*DataFrame containing all scalar data + uuid columns for all stacked sessions*)
- **normalize (bool)** (*Indicates whether normalize the pdfs.*)
- **centroid_vars (list)** (*list of strings for column values to use when computing mouse position.*)
- **key (str)** (*metadata column to return info from.*)

Returns: **pdfs (list)** (*list of 2d np.arrays of PDFs for each session.*) **groups (list)** (*list of strings of groups corresponding to pdfs index.*) **sessions (list)** (*list of strings of session uuids corresponding to pdfs index.*) **subjectNames (list)** (*list of strings of subjectNames corresponding to pdfs index.*)

`moseq2_viz.scalars.util.`**`compute_mean_syll_scalar`** (scalar_df, scalar='velocity_3d_mm', max_sylls=40, syllable_key='labels (usage sort)')

Computes the mean syllable scalar-value based on the time-series scalar dataframe and the selected scalar. Finds the frame indices with corresponding each of the label values (up to max syllables) and looks up the scalar values in the dataframe.

Parameters:
- **scalar_df (pd.DataFrame)** (*DataFrame containing all scalar data + uuid and syllable columns for all stacked sessions*)
- **scalar (str or list)** (*Selected scalar column(s) to compute mean value for syllables*)
- **max_sylls (int)** (*maximum amount of syllables to include in output.*)
- **syllable_key (str)** (*column in scalar_df that points to the syllable labels to use.*)

Returns: **mean_df (pd.DataFrame)**

Return type: updated input DataFrame with a speed value for each syllable merge in as a new column.

moseq2_viz.scalars.util.**compute_mouse_dist_to_center** (roi, centroid_x_px, centroid_y_px)

Given the session's ROI shape and the frame-by-frame (x,y) pixel centroid location

to compute the mouse's relative distance to the center of the bucket.

Parameters:
- **roi (tuple)** (*Tuple of session's arena dimensions.*)
- **centroid_x_px (1D np.array)** (*x-coordinate of the mouse centroid throughout the recording*)
- **centroid_y_px (1D np.array)** (*y-coordinate of the mouse centroid throughout the recording*)

Returns: **dist_to_center (1D np.array)**

Return type: array of distance to the arena center in pixels.

moseq2_viz.scalars.util.**compute_syllable_position_heatmaps** (scalar_df, syllable_key='labels (usage sort)', syllables=range(0, 40), centroid_keys=['centroid_x_mm', 'centroid_y_mm'], normalize=False, bins=20)

Computes position heatmaps for each syllable on a session-by-session basis

Parameters:
- **scalar_df (pd.DataFrame)** (*DataFrame containing scalar data & labels for all sessions*)
- **syllable_key (str)** (*dataframe column to access syllable labels*)
- **syllables (list)** (*List of syllables to compute heatmaps for.*)
- **centroid_keys (list)** (*list of column names containing the centroid values used to compute mouse position.*)
- **normalize (bool)** (*If True, normalizes the histogram to be a probability density*)
- **bins (int)** (*number of bins to cut the position data into*)

Returns: **complete_df (pd.DataFrame)** – new PDF column corresponding to each session-syllable pair.

Return type: Inputted model results dataframe with a

moseq2_viz.scalars.util.**convert_legacy_scalars** (old_features, force: bool = False, true_depth: float = 673.1) → dict

Converts scalars in the legacy format to the new format, with explicit units.

Parameters:
- **old_features (str, h5 group, or dictionary of scalars)** (*filename, h5 group,*) – or dictionary of scalar values.
- **force (bool)** (*force the conversion of centroid_[xy]_px into mm.*)
- **true_depth (float)** (*true depth of the floor relative to the camera (673.1 mm by default)*)

Returns: **features (dict)**

Return type: dictionary of scalar values

moseq2_viz.scalars.util.**convert_pxs_to_mm** (coords, resolution=512, 424, field_of_view=70.6, 60, true_depth=673.1)

Converts x, y coordinates in pixel space to mm # http://stackoverflow.com/questions/17832238/kinect-intrinsic-parameters-from-field-of-view/18199938#18199938 # http://www.imaginativeuniversal.com/blog/post/2014/03/05/quick-reference-kinect-1-vs-kinect-2.aspx # http://smeenk.com/kinect-field-of-view-comparison/

Parameters:
- **coords (list)** (*list of [x,y] pixel coordinate lists.*)
- **resolution (tuple)** (*video frame size.*)
- **field_of_view (tuple)** (*camera focal lengths.*)
- **true_depth (float)** (*detected distance between depth camera and bucket floor.*)

Returns: **new_coords (list)**

Return type: list of same [x,y] coordinates in millimeters.

moseq2_viz.scalars.util.**generate_empty_feature_dict** (nframes) → dict
Generates a dict of numpy array of zeros of length nframes for each feature parameter.

Parameters: **nframes (int)** (*length of video*)
Returns: **(dict)**
Return type: dictionary feature to numpy 0 arrays of length nframes key-value pairs.

moseq2_viz.scalars.util.**get_scalar_map** (index, fill_nans=True, force_conversion=False)
Returns a dictionary of scalar values loaded from an index dictionary.

Parameters:
- **index (dict)** (*dictionary of index file contents.*)
- **fill_nans (bool)** (*indicate whether to replace NaN values with 0.*)
- **force_conversion (bool)** (*force the conversion of centroid_[xy]_px into mm.*)

Returns: **scalar_map (dict)**
Return type: dictionary of all the scalar values acquired after extraction.

moseq2_viz.scalars.util.**get_scalar_triggered_average** (scalar_map, model_labels, max_syllable=40, nlags=20, include_keys=['velocity_2d_mm', 'velocity_3d_mm', 'width_mm', 'length_mm', 'height_ave_mm', 'angle'], zscore=False)
Get averages of selected scalar keys for each syllable.

Parameters:
- **scalar_map (dict)** (*dictionary of all the scalar values acquired after extraction.*)
- **model_labels (dict)** (*dictionary of uuid to syllable label array pairs.*)
- **max_syllable (int)** (*maximum number of syllables to use.*)
- **nlags (int)** (*number of lags to use when averaging over a series of PCs.*)
- **include_keys (list)** (*list of scalar values to load averages of.*)
- **zscore (bool)** (*indicate whether to z-score loaded values.*)

Returns: **syll_average (dict)**
Return type: dictionary of scalars for each syllable sequence.

moseq2_viz.scalars.util.**get_syllable_pdfs** (pdf_df, normalize=True, syllables=range(0, 40), groupby='group', syllable_key='labels (usage sort)')
Computes the mean syllable position PDF/Heatmap for the given groupings. Either mean of modeling groups: groupby='group', or a verbose list of all the session's syllable PDFs groupby='SessionName'

Parameters:
- **pdf_df (pd.DataFrame)** (*model results dataframe including a position PDF column containing 2D numpy arrays.*)
- **normalize (bool)** (*Indicates whether normalize the pdf scales.*)
- **syllables (list)** (*list of syllables to get a grouping of.*)
- **groupby (str)** (*column name to group the df keys by. (either group, or SessionName)*)
- **syllable_key (str)** (*name of the column that contains the requested syllable label sequences.*)

Returns: **group_syll_pdfs (list)** (*2D list of computed pdfs of shape ngroups x nsyllables*) **groups (list)** (*list of corresponding names to each row in the group_syll_pdfs list*)

moseq2_viz.scalars.util.**is_legacy** (features: dict)
Checks a dictionary of features to see if they correspond with an older version of moseq.

Parameters: **features (dict)** (*dict of scalar_df column names.*)
Returns: **(bool)**
Return type: true if the dict is from an old dataset

moseq2_viz.scalars.util.**nanzscore** (data)
Z-score numpy array that may contain NaN values.

Parameters: **data (np.ndarray)** (*array of scalar values.*)
Returns: **data (np.ndarray)**
Return type: z-scored data.

`moseq2_viz.scalars.util.`**`process_scalars`** (scalar_map: dict, include_keys: list, zscore: bool = False) → dict
Fill NaNs and possibly zscore scalar values.

Parameters:
- **scalar_map (dict)** (*dictionary of all the scalar values acquired after extraction.*)
- **include_keys (list)** (*scalar keys to process.*)
- **zscore (bool)** (*indicate whether to z-score loaded values.*)

Returns: **scalar_map (dict)**
Return type: dict that contains the updated NaN-filled values.

`moseq2_viz.scalars.util.`**`scalars_to_dataframe`** (index: dict, include_keys: list = ['SessionName', 'SubjectName', 'StartTime'], disable_output=False, force_conversion=True, model_path=None)
Generates a dataframe containing scalar values over the course of a recording session. If a model string is included, then return only animals that were included in the model Called to sort scalar metadata information when graphing in plot-scalar-summary.

Parameters:
- **index (dict)** (a sorted_index generated by *parse_index* or *get_sorted_index*)
- **include_keys (list)** (*a list of other moseq related keys to include in the dataframe*)
- **disable_output (bool)** (*indicate whether to show tqdm output.*)
- **force_conversion (bool)** (*force the conversion of centroid_[xy]_px into mm.*)
- **model_path (str)** (*path to model object to pull labels from and include in the dataframe*)

Returns: **scalar_df (pandas DataFrame)**
Return type: DataFrame of loaded scalar values with their selected metadata.

`moseq2_viz.scalars.util.`**`star_valmap`** (func, d)

Index

- **genindex**

Index

Symbols

A

C

E

G

-e

-g

-k

-n

-v

INDEX_FILE

moseq2-viz-copy-h5-metadata-to-yaml command line option

--input-dir <input_dir>

-i

moseq2-viz-get-best-model command line option

--ext <ext>

--fps <fps>

--plot-all

CP_PATH

MODEL_DIR

OUTPUT_FILE

moseq2-viz-make-crowd-movies command line option

--cmap <cmap>

--count <count>

--frame-path <frame_path>

--gaussfilter-space <gaussfilter_space>

--legacy-jitter-fix <legacy_jitter_fix>

--max-dur <max_dur>

--max-examples <max_examples>

--max-height <max_height>

--max-syllable <max_syllable>

--medfilter-space <medfilter_space>

--min-dur <min_dur>

--min-height <min_height>

--output-dir <output_dir>

--pad <pad>

--processes <processes>

--progress-bar

--raw-size <raw_size>

--scale <scale>

--separate-by <separate_by>

--session-names <session_names>

--sort <sort>

--specific-syllable <specific_syllable>

-m

-o

-p

-s

INDEX_FILE

MODEL_PATH

moseq2-viz-plot-group-position-heatmaps command line option

--output-file <output_file>

INDEX_FILE

moseq2-viz-plot-scalar-summary command line option

--colors <colors>

--output-file <output_file>

-c

INDEX_FILE

moseq2-viz-plot-stats command line option

--colors <colors>

--count <count>

--ctrl-group <ctrl_group>

--exp-group <exp_group>

--figsize <figsize>

--group <group>

--max-syllable <max_syllable>

--ordering <ordering>

--output-file <output_file>

--sort <sort>

--stat <stat>

-c

-g

-o

INDEX_FILE

MODEL_FIT

moseq2-viz-plot-transition-graph command line option

--arrows

--count <count>

--edge-scaling <edge_scaling>

--edge-threshold <edge_threshold>

--group <group>

--keep-orphans

--layout <layout>

--max-syllable <max_syllable>

--node-scaling <node_scaling>

--normalize <normalize>

--orphan-weight <orphan_weight>

N

O

P

R

S

T

W

Python Module Index

m