Python Documentation

version

May 13, 2020

Contents

Welcome to moseq2-model's documentation!

moseq2_model Package

CLI Module

moseq2-model

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

count-frames

```
moseq2-model count-frames [OPTIONS] INPUT_FILE
```

Options

`--var-name` `<var_name>`
Variable name in input file with PCs [default: scores]

Arguments

`INPUT_FILE`
Required argument

learn-model

```
moseq2-model learn-model [OPTIONS] INPUT_FILE DEST_FILE
```

Options

`-h, --hold-out`
Hold out one fold (set by nfolds) for computing heldout likelihood [default: False]

`--hold-out-seed` `<hold_out_seed>`
Random seed for holding out data (set for reproducibility) [default: -1]

`--nfolds` `<nfolds>`
Number of folds for split [default: 5]

`-c, --ncpus` `<ncpus>`
Number of cores to use for resampling [default: 0]

`-n, --num-iter` `<num_iter>`
Number of times to resample model [default: 100]

`--var-name` `<var_name>`
Variable name in input file with PCs [default: scores]

`--e-step`
Compute the expected state values for each animal [default: False]

`-s, --save-every` `<save_every>`
Increment to save labels and model object (-1 for just last) [default: -1]

`--save-model`
Save model object at the end of training [default: False]

`-m, --max-states` `<max_states>`
Maximum number of states [default: 100]

`--npcs` `<npcs>`
Number of PCs to use [default: 10]

`-w, --whiten` `<whiten>`

Whiten (e)each (a)ll or (n)o whitening [default: all]

-p, --progressbar `<progressbar>`
Show model progress [default: True]

--percent-split `<percent_split>`
Training-validation split percentage [default: 20]

-k, --kappa `<kappa>`
Kappa

-g, --gamma `<gamma>`
Gamma [default: 1000.0]

-a, --alpha `<alpha>`
Alpha [default: 5.7]

--noise-level `<noise_level>`
Additive white gaussian noise for regularization [default: 0]

--nlags `<nlags>`
Number of lags to use [default: 3]

--separate-trans
Use separate transition matrix per group [default: False]

--robust
Use tAR model [default: False]

--checkpoint-freq `<checkpoint_freq>`
checkpoint the training after N iterations [default: -1]

-i, --index `<index>`
Path to moseq2-index.yaml for group definitions (used only with the separate-trans flag) [default:]

--default-group `<default_group>`
Default group to use for separate-trans [default: n/a]

-v, --verbose
Print syllable log-likelihoods during training. [default: False]

Arguments

INPUT_FILE
Required argument

DEST_FILE
Required argument

GUI Module

`moseq2_model.gui.`**`learn_model_command`**`(input_file, dest_file, config_file, index, hold_out, nfolds, num_iter, max_states, npcs, kappa, separate_trans, robust, checkpoint_freq, percent_split=20, verbose=False, output_directory=None)`
Trains ARHMM from Jupyter notebook.

Parameters:

- **input_file (str)** (*pca scores file path.*)
- **dest_file (str)** (*path to save model to.*)
- **config_file (str)** (*configuration file path.*)
- **index (str)** (*index file path.*)
- **hold_out (bool)** (*indicate whether to hold out data or use train_test_split.*)
- **nfolds (int)** (*number of folds to hold out.*)
- **num_iter (int)** (*number of training iterations.*)
- **max_states (int)** (*maximum number of model states.*)
- **npcs (int)** (*number of PCs to include in analysis.*)
- **kappa (float)** (*probability prior distribution for syllable duration.*)
- **separate_trans (bool)** (*indicate whether to compute separate syllable transition matrices for each group.*)
- **robust (bool)** (*indicate whether to use a t-distributed syllable label distribution. (robust-ARHMM)*)
- **checkpoint_freq (int)** (*frequency at which to save model checkpoints*)
- **percent_split (int)** (*train-validation data split ratio percentage.*)
- **verbose (bool)** (*compute modeling summary (Warning current implementation is slow).*)
- **output_directory (str)** (*alternative output directory for GUI users*)

Returns:

Return type: None

General Utilities Module

`moseq2_model.util.`**`append_resample`**`(filename, label_dict: dict)`

Adds the labels from a resampling iteration to a pickle file.

Parameters:

- **filename (str)** (*file (containing modeling results) to append new label dict to.*)
- **label_dict (dict)** (*a dictionary with a single key/value pair, where the*) – key is the sampling iteration and the value contains a dict of: (labels, a log likelihood val, and expected states if the flag is set) from each mouse.

Returns:

Return type: None

`moseq2_model.util.`**`copy_model`**`(model_obj)`

Return a new copy of a model using deepcopy().

Parameters: **model_obj (ARHMM)** (*model to copy.*)

Returns: **cp (ARHMM)**

Return type: copy of the model

`moseq2_model.util.`**`flush_print`**`()`

print(value, ..., sep=' ', end='n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default. Optional keyword arguments: file: a file-like object (stream); defaults to the current sys.stdout. sep: string inserted between values, default a space. end: string appended after the last value, default a newline. flush: whether to forcibly flush the stream.

`moseq2_model.util.`**`get_parameters_from_model`**`(model, save_ar=True)`

Get parameter dictionary from model.

Parameters:
- **model (ARHMM)** (*model to get parameters from.*)
- **save_ar (bool)** (*save AR Matrices.*)

Returns: **parameters (dict)**

Return type: dictionary containing all modeling parameters

`moseq2_model.util.`**`h5_to_dict`** (`h5file`, `path: str`) → dict

Load h5 data to dictionary from a user specified path.

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*)

Returns: **out (dict)**

Return type: a dict with h5 file contents with the same path structure

`moseq2_model.util.`**`list_rank`** (`chk_list`)

`moseq2_model.util.`**`load_arhmm_checkpoint`** (`filename: str`, `train_data: dict`) → dict

Load an arhmm checkpoint and re-add data into the arhmm model checkpoint.

Parameters:
- **filename (str)** (*path that specifies the checkpoint.*)
- **train_data (OrderedDict)** (*an OrderedDict that contains the training data*)

Returns: **mdl_dict (dict)**

Return type: a dict containing the model with reloaded data, and associated training data

`moseq2_model.util.`**`load_cell_string_from_matlab`** (`filename`, `var_name='uuids'`)

Load cell strings from MATLAB file.

Parameters:
- **filename (str)** (*path to .mat file*)
- **var_name (str)** (*cell name to read*)

Returns: **return_list (list)**

Return type: list of selected loaded variables

`moseq2_model.util.`**`load_data_from_matlab`** (`filename`, `var_name='features'`, `npcs=10`)

Load PC Scores from a specified variable column in a MATLAB file.

Parameters:
- **filename (str)** (*path to MATLAB (.mat) file*)
- **var_name (str)** (*variable to load*)
- **npcs (int)** (*number of PCs to load.*)

Returns: **data_dict (OrderedDict)**

Return type: loaded dictionary of uuid and PC-score pairings.

`moseq2_model.util.`**`load_dict_from_hdf5`** (`filename`)

A convenience function to load the entire contents of an h5 file into a dictionary.

Parameters: **filename (str)** (*path to h5 file.*)

Returns: **(dict)**

Return type: dict containing all of the h5 file contents.

`moseq2_model.util.`**`load_pcs`** (`filename`, `var_name='features'`, `load_groups=False`, `npcs=10`, `h5_key_is_uuid=True`)

Load the Principal Component Scores for modeling.

Parameters:
- **filename (str)** (*path to the file that contains PC scores*)
- **var_name (str)** (key where the pc scores are stored within `filename`)
- **load_groups (bool)** (*Load metadata group variable*)
- **npcs (int)** (*Number of PCs to load*)
- **h5_key_is_uuid (bool)** (*use h5 key as uuid.*)

Returns: **data_dict (OrderedDict)** (*key-value pairs for keys being uuids and values being PC scores.*)
metadata (OrderedDict) (*dictionary containing lists of index-aligned uuids and groups.*)

`moseq2_model.util.`**`progressbar`**`(*args, **kwargs)`

Selects tqdm progress bar.

Parameters:
- **args (iterable)**
- **kwargs (tdqm args[1** (*]*))

Returns:

Return type: tqdm() iterating object.

`moseq2_model.util.`**`recursively_save_dict_contents_to_group`**`(h5file, export_dict, path='/')`

Recursively save dicts to h5 file groups. # https://codereview.stackexchange.com/questions/120802/recursively-save-python-dictionaries-to-hdf5-files-using-h5py

Parameters:
- **h5file (h5py.File)** (*opened h5py File object.*)
- **export_dict (dict)** (*dictionary to save*)
- **path (str)** (*path within h5 to save to.*)

Returns:

Return type: None

`moseq2_model.util.`**`save_arhmm_checkpoint`**`(filename: str, arhmm: dict)`

Save an arhmm checkpoint and strip out data used to train the model.

Parameters:
- **filename (str)** (*path that specifies the checkpoint*)
- **arhmm (dict)** (*a dictionary containing the model obj, training iteration number,*) – log-likelihoods of each training step, and labels for each step.

Returns:

Return type: None

`moseq2_model.util.`**`save_dict`**`(filename, obj_to_save=None)`

Save dictionary to file.

Parameters:
- **filename (str)** (*path to file where dict is being saved.*)
- **obj_to_save (dict)** (*dict to save.*)

Returns:

Return type: None

Subpackages

moseq2_model.helpers Package

Helpers - Data Module

`moseq2_model.helpers.data.`**`flush_print`**`()`

print(value, …, sep=' ', end='n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default. Optional keyword arguments: file: a file-like object (stream); defaults to the current sys.stdout. sep: string inserted between values, default a space. end: string appended after the last value, default a newline. flush: whether to forcibly flush the stream.

`moseq2_model.helpers.data.`**`get_heldout_data_splits`** (`all_keys`, `data_dict`, `train_list`, `hold_out_list`)

Split data based on held out keys.

Parameters:

- **all_keys (list)** (*list of all keys included in the model.*)
- **data_dict (OrderedDict)** (*dictionary of all PC scores included in the model*)
- **train_list (list)** (*list of keys included in the training data*)
- **hold_out_list (list)** (*list of keys included in the held out data*)

Returns: **train_list (list)** (*list of keys included in the training data.*) **train_data (OrderedDict)** (*dictionary of uuid to PC score key-value pairs for uuids in train_list*) **hold_out_list (list)** (*list of keys included in the held out data.*) **test_data (OrderedDict)** (*dictionary of uuids to PC score key-value pairs for uuids in hold_out_list.*) **nt_frames (list)** (*list of the number of frames in each session in train_data*)

`moseq2_model.helpers.data.`**`get_training_data_splits`** (`config_data`, `data_dict`)

Split data using sklearn train_test_split along all keys.

Parameters:

- **config_data (dict)** (*dictionary containing percentage split parameter. (autogenerated in GUI AND CLI)*)
- **data_dict (OrderedDict)** (*dict of uuid-PC Score key-value pairs for all data included in the model.*)

Returns: **train_list (list)** (*list of all the keys included in the model.*) **train_data (OrderedDict)** (*all the of the key-value pairs included in the model.*) **training_data (OrderedDict)** (*the split percentage of the training data.*) **hold_out_list (list)** (*None*) **validation_data (OrderedDict)** (*the split percentage of the validation data*) **nt_frames (list)** (*list of length of each session in the split training data.*)

`moseq2_model.helpers.data.`**`graph_modeling_loglikelihoods`** (`config_data`, `iter_lls`, `iter_holls`, `group_idx`, `dest_file`)

Graphs model training performance progress throughout modeling. Will only run if verbose == True

Parameters:

- **config_data (dict)** (*dictionary of model training parameters.*)
- **iter_lls (list)** (*list of training log-likelihoods over each iteration*)
- **iter_holls (list)** (*list of held out log-likelihoods over each iteration*)
- **group_idx (list)** (*list of groups included in the modeling.*)
- **dest_file (str)** (*path to the model.*)

Returns: **img_path (str)**

Return type: path to saved graph.

`moseq2_model.helpers.data.`**`prepare_model_metadata`** (`data_dict`, `data_metadata`, `config_data`, `nkeys`, `all_keys`)

Sets model training metadata parameters, whitens data, if hold_out is True, will split data and return list of heldout keys, and updates all dictionaries.

Parameters:

- **data_dict (OrderedDict)** (*loaded data dictionary.*)
- **data_metadata (OrderedDict)** (*loaded metadata dictionary.*)
- **config_data (dict)** (*dictionary containing all modeling parameters.*)
- **nkeys (int)** (*total amount of keys being modeled.*)
- **all_keys (list)** (*list of keys being modeled.*)

Returns: **config_data (dict)** (*updated dictionary containing all modeling parameters.*) **data_dict (OrderedDict)** (*update data dictionary.*) **model_parameters (dict)** (*dictionary of pre-selected model parameters*) **train_list (list)** (*list of keys included in training list.*) **hold_out_list (list)** (*heldout list of keys (if hold_out == True)*)

`moseq2_model.helpers.data.`**`process_indexfile`**`(index, config_data, data_metadata)`

Reads index file (if it exists) and returns dictionaries containing metadata in the index file. The data_metadata will also be updated with the information read from the index file

Parameters:

- **index (str)** (*path to index file.*)
- **config_data (dict)** (*dictionary containing all modeling parameters.*)
- **data_metadata (dict)** (*loaded metadata containing uuid and group information.*)

Returns: **index_data (dict)** (*dictionary containing data contained in the index file.*) **data_metadata (dict)** (*updated metadata dictionary.*)

`moseq2_model.helpers.data.`**`select_data_to_model`**`(index_data, gui=False)`

GUI: Prompts user to select data to model via the data uuids/groups and paths located in the index file. CLI: Selects all data from index file.

Parameters:

- **index_data (dict)** (*loaded dictionary from index file*)
- **gui (bool)** (*indicates prompting user input*)

Returns: **all_keys (list)** (*list of uuids to model*) **groups (list)** (*list of groups to model*)

Helpers - Wrappers Module

`moseq2_model.helpers.wrappers.`**`learn_model_wrapper`**`(input_file, dest_file, config_data, index=None, output_directory=None, gui=False)`

Wrapper function to train ARHMM, shared between CLI and GUI.

Parameters:

- **input_file (str)** (*path to pca scores file.*)
- **dest_file (str)** (*path to save model to.*)
- **config_data (dict)** (*dictionary containing necessary modeling parameters.*)
- **index (str)** (*path to index file.*)
- **output_directory (str)** (*path to alternative output directory.*)
- **gui (bool)** (*indicates whether Jupyter notebook is being used.*)

Returns:

Return type: None

moseq2_model.train Package

Train - Fit Module

Contains a model class that is compatible with scikit-learn's GridsearchCV api. This class extends other functionality, such as visually inspecting model statistics within a jupyter notebook

class `moseq2_model.train.fit.`**`MoseqModel`**`(max_iters=100, n_cpus=1, optimal_duration=0.4, scale_kappa_w_alpha=True, history=True, **model_params)`

Bases: **`object`**

`duration_score``()`

Computes score for assigned syllable duration

Returns: **(1D numpy array)**

Return type: scores of computed median syllable durations

fit (X, y=None)

Trains model given data.

Parameters:
- **X (OrderedDict)** (*data_dict used to train ARHMM*)
- **y (None)**

Returns:

Return type: None

get_median_duration ()

Calculates median duration.

Returns: **(pandas DataFrame)**

Return type: DataFrame of median syllable durations

get_params (deep=True)

Get model parameters.

Parameters: **deep (bool)** (*indicate whether to use deep copy*)

Returns: **params (dict)**

Return type: Model parameters

log_likelihood_score (X, reduction=None)

Compute Log-Likelihood Score of each session.

Parameters:
- **X (list or OrderedDict)** (*data to compute log-likelihood score from.*)
- **reduction (str)** (*indicates whether to use a reduction operation.*)

Returns: **_lls (1D numpy array)**

Return type: log-likelihood arrays.

partial_fit (X)

Not implemented.

Parameters: **X (OrderedDict)**

predict (X)

Get label predictions from input data.

Parameters: **X (list, or OrderedDict)** (*data to predict labels*)

Returns: **y_pred (list)**

Return type: list of label predictions

predict_proba ()

score ()

set_params (**model_params)

Update model parameters.

Parameters: **model_params (dict)** (*model parameter dictionary to update*)

Returns:

Return type: None

Train - Label Utilities Module

moseq2_model.train.label_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_model.train.label_util.`**`syll_id`**`(labels: numpy.ndarray)` → numpy.ndarray

Returns the syllable label at each 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_model.train.label_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_model.train.label_util.`**`to_df`**`(labels, uuid)` → pandas.core.frame.DataFrame

Convert labels numpy.ndarray to pandas.DataFrame

Parameters:

- **labels (np.ndarray)** (*array of syllable labels for a mouse.*)
- **uuid (list)** (*list of session uuids representing each series of labels.*)

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

Return type: DataFrame of syllables, durations, onsets, and session uuids.

Train - Model Module

`moseq2_model.train.models.`**`ARHMM`**`(data_dict, kappa=1000000.0, gamma=999, nlags=3, alpha=5.7, K_0_scale=10.0, S_0_scale=0.01, max_states=100, empirical_bayes=True, affine=True, model_hypparams={}, obs_hypparams={}, sticky_init=False, separate_trans=False, groups=None, robust=False, silent=False)`

Initializes ARHMM and adds data and groups to model.

Parameters:

- **data_dict (OrderedDict)** (*dictionary of data to add to model*)
- **kappa (float)** (*probability prior distribution for syllable duration*)
- **gamma (float)** (*probability prior distribution for PCs explaining syllable states*)
- **nlags (int)** (*number of lag frames to add to sessions*)
- **alpha (float)** (*probability prior distribution for syllable transition rate*)
- **K_0_scale (float)** (*Standard deviation of lagged data*)
- **S_0_scale (float)** (*Standard deviation of data*)
- **max_states (int)** (*Maximum number of model states*)
- **empirical_bayes (bool)** (*Use empirical bayes AR parameters*)
- **affine (bool)** (*Use affine transformation*)
- **model_hypparams (dict)** (*dictionary of model parameters*)
- **obs_hypparams (dict)** (*dictionary of observational parameters*)
- **sticky_init (bool)** (*Initialize the states with random projections.*)
- **separate_trans (bool)** (*use separate transition graphs for each unique group*)
- **groups (list)** (*list of groups to model*)
- **robust (bool)** (*use t-Distribution model*)
- **silent (bool)** (*print out model information.*)

Returns: **model (ARHMM)**

Return type: model object with data loaded, prepared for modeling.

`moseq2_model.train.models.`**`flush_print`** ()

print(value, …, sep=' ', end='n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default. Optional keyword arguments: file: a file-like object (stream); defaults to the current sys.stdout. sep: string inserted between values, default a space. end: string appended after the last value, default a newline. flush: whether to forcibly flush the stream.

Train - General Utilities Module

`moseq2_model.train.util.`**`get_crosslikes`** (`arhmm`, `frame_by_frame=False`)

Gets the cross-likelihoods, a measure of confidence in the model's segmentation, for each syllable a model learns.

Parameters:
- **arhmm** (*the ARHMM model object fit to your data*)
- **frame_by_frame (bool)** (*if True, the cross-likelihoods will be computed for each frame.*)

Returns: **All_CLs (list)** (*a dictionary containing cross-likelihoods for each syllable pair.*) if `frame_by_frame=True`, it will contain a value for each frame **CL (np.ndarray)** (*the average cross-likelihood for each syllable pair*)

`moseq2_model.train.util.`**`get_labels_from_model`** (`model`)

Grabs the model labels for each training dataset and places them in a list.

Parameters: **model (ARHMM)** (*trained ARHMM model*)

Returns: **cat_labels (list)**

Return type: Predicted syllable labels for all frames concatenated into a single list.

`moseq2_model.train.util.`**`get_model_summary`** (`model`, `groups`, `train_data`, `val_data`, `separate_trans`, `num_frames`, `iter_lls`, `iter_holls`)

Computes a summary of model performance after resampling steps. Is only run if verbose = True.

Parameters:
- **model (ARHMM)** (*model to compute lls.*)
- **groups (list)** (*list of session group names.*)
- **train_data (OrderedDict)** (*Ordered dict of training data*)
- **val_data** (*(OrderedDict): Ordered dict of validation/held-out data*)
- **separate_trans (bool) indicates whether to separate lls for each group.**
- **num_frames (int)** (*total number of frames included in modeling.*)
- **iter_lls (list)** (*list of log-likelihoods at an iteration level.*)
- **iter_holls (list)** (*list of held-out log-likelihoods at an iteration level.*)

Returns: **iter_lls (list)** (*updated list of log-likelihoods at an iteration level.*) **iter_holls (list)** (*updated list of held-out log-likelihoods at an iteration level.*)

`moseq2_model.train.util.`**`rleslices`** (`seq`)

Get changepoint index slices

Parameters: **seq (list)** (*list of labels*)

Returns: **(map generator)**

Return type: slices given syllable changepoint indices

`moseq2_model.train.util.`**`run_e_step`** (`arhmm`)

Computes the expected states for each training dataset and places them in a list.

Parameters: **arhmm (ARHMM)** (*model to compute expected states from.*)

Returns: **e_states (list)**

Return type: list of expected states `moseq2_model.train.util.`**`slices_from_indicators`**`(indseq)`

Given indices for seqences, return list sliced sublists.

Parameters: **indseq (list)** (*indices to create slices at.*)

Returns: **(list)**

Return type: list of slices from given indices.

`moseq2_model.train.util.`**`train_model`**`(model, num_iter=100, save_every=1, ncpus=1, checkpoint_freq=None, checkpoint_file=None, start=0, save_file=None, progress_kwargs={}, num_frames=[1], train_data=None, val_data=None, separate_trans=False, groups=None, verbose=False)`

ARHMM training: Resamples ARHMM for inputted number of iterations, and optionally computes loglikelihood scores for each iteration if verbose is True.

Parameters:

- **model (ARHMM)** (*model to train.*)
- **num_iter (int)** (*total number of resampling iterations.*)
- **save_every (int)** (*model parameter updating frequency.*)
- **ncpus (int)** (*number of cpus to resample model.*)
- **checkpoint_freq (int)** (*frequency of new checkpoint saves in iterations*)
- **checkpoint_file (str)** (*path to new checkpoint file*)
- **start (int)** (*starting iteration index (used to resume modeling, default is 0).*)
- **save_file (str)** (*path to file to save model checkpoint (only if checkpoint_freq > 0)*)
- **progress_kwargs (dict)** (*keyword arguments for progress bar*)
- **num_frames (int)** (*total number of frames included in modeling*)
- **train_data (OrderedDict)** (*dict of validation data (only if verbose = True)*)
- **val_data (OrderedDict)** (*dict of validation data (only if verbose = True)*)
- **separate_trans (bool)** (*using different transition matrices*)
- **groups (list)** (*list of groups included in modeling (only if verbose = True)*)
- **verbose (bool)** (*Compute model summary.*)

Returns: **model (ARHMM)** (*trained model.*) **model.log_likelihood() (list)** (*list of training Log-likelihoods per session after modeling.*) **get_labels_from_model(model) (list)** (*list of labels predicted post-modeling.*) **iter_lls (list)** (*list of log-likelihoods at an iteration level.*) **iter_holls (list)** (*list of held-out log-likelihoods at an iteration level.*) **group_idx (list)** (*list of group names per modeled session.*)

`moseq2_model.train.util.`**`whiten_all`**`(data_dict, center=True)`

Whitens all the PC Scores at once.

Parameters:

- **data_dict (OrderedDict)** (*Training dictionary*)
- **center (bool)** (*Indicates whether to center data.*)

Returns: **data_dict (OrderedDict)**

Return type: Whitened training data dictionary

`moseq2_model.train.util.`**`whiten_each`**`(data_dict, center=True)`

Whiten each group of PC scores separately

Parameters:

- **data_dict (OrderedDict)** (*Training dictionary*)
- **center (bool)** (*Indicates whether to normalize data.*)

Returns: **data_dict (OrderedDict)**

Return type: Whitened training data dictionary

`moseq2_model.train.util.`**`zscore_all`**`(data_dict, npcs=10, center=True)` z-score the PC Scores altogether.

Parameters:
- **data_dict (OrderedDict)** (*Training dictionary*)
- **npcs (int)** (*number of pcs included*)
- **center (bool)** (*Indicates whether to normalize data.*)

Returns: **data_dict (OrderedDict)**

Return type: z-scored training data dictionary

`moseq2_model.train.util.`**`zscore_each`**(`data_dict`, `center=True`)

z-score each set of PC Scores separately

Parameters:
- **data_dict (OrderedDict)** (*Training dictionary*)
- **center (bool)** (*Indicates whether to normalize data.*)

Returns: **data_dict (OrderedDict)**

Return type: z-scored training data dictionary

Index

- **`genindex`**

Index

Python Module Index