Python Documentation

version

October 20, 2020

Contents

Welcome to moseq2-model's documentation!

moseq2_model Package

CLI Module

moseq2-model

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

Options

--version
Show the version and exit. [default: False]

count-frames

Counts number of frames in given h5 file (pca_scores)

```
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

kappa-scan

Batch fit multiple models scanning over different syllable length probability prior.

```
moseq2-model kappa-scan [OPTIONS] INPUT_FILE OUTPUT_DIR
```

Options

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

--out-script <out_script>
Path to output bash script file containing all model training commands. [default: /Users/aymanzeine/Desktop/moseq/moseq2-model/docs/train_out.sh]

--n-models <n_models>
Minimum kappa value to train model on. [default: 10]

--prefix <prefix>
Batch command string to prefix model training command. [default:]

--cluster-type <cluster_type>
Platform to train models on [default: local]

Options: local|slurm

--scan-scale <scan_scale>
Scale to scan kappa values at. [default: log]

Options: log|linear

--min-kappa <min_kappa>
Minimum kappa exponent to train model on.

--max-kappa <max_kappa>

Maximum kappa exponent to train model on.

-n, --ncpus <ncpus>
Number of CPUs [default: 4]

-m, --memory <memory>
RAM string [default: 5GB]

-w, --wall-time <wall_time>
Wall time [default: 3:00:00]

--partition <partition>
Partition name [default: short]

--get-cmd
Print scan command strings. [default: False]

--run-cmd
Run scan command strings. [default: False]

--check-every <check_every>
Increment to check whether the model training has converged. [default: 5]

--converge
Train model until loglikelihood converges. [default: False]

--robust
Use tAR model [default: False]

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

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

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

-a, --alpha <alpha>
Alpha; probability prior distribution for syllable transition rate. [default: 5.7]

-g, --gamma <gamma>
Gamma; probability prior distribution for PCs explaining syllable states. Smaller gamma = steeper PC_Scree plot. [default: 1000.0]

-h, --load-groups <load_groups>
Dictates in PC Scores should be loaded with their associated group. [default: True]

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

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

-w, --whiten <whiten>
Whiten (e)each (a)ll or (n)o whitening [default: all]

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

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

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

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

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

--var-name <var_name>

Variable name in input file with PCs [default: scores]

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

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

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

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

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

Arguments

INPUT_FILE
Required argument

OUTPUT_DIR
Required argument

learn-model

Trains ARHMM on PCA Scores with given training parameters

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

Options

--check-every <check_every>
Increment to check whether the model training has converged. [default: 5]

--converge
Train model until loglikelihood converges. [default: False]

--robust
Use tAR model [default: False]

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

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

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

-a, --alpha <alpha>
Alpha; probability prior distribution for syllable transition rate. [default: 5.7]

-g, --gamma <gamma>
Gamma; probability prior distribution for PCs explaining syllable states. Smaller gamma = steeper PC_Scree plot. [default: 1000.0]

-h, --load-groups <load_groups>
Dictates in PC Scores should be loaded with their associated group. [default: True]

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

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

-w, --whiten <whiten>
Whiten (e)each (a)ll or (n)o whitening [default: all]

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

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

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

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

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

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

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

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

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

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

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

-k, --kappa <kappa>
Kappa; probability prior distribution for syllable duration. Larger k = longer syllable lengths

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

--use-checkpoint
indicate whether to use previously saved checkpoint [default: False]

-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

GUI front-end function for training ARHMM.

moseq2_model.gui.**learn_model_command** (progress_paths, hold_out=False, nfolds=2, num_iter=100, max_states=100, npcs=10, scan_scale='log', kappa=None, min_kappa=None, max_kappa=None, n_models=5, alpha=5.7, gamma=1000.0, separate_trans=True, robust=True, checkpoint_freq=-1, use_checkpoint=False, converge=False, check_every=5, select_groups=False, percent_split=20, output_dir=None, out_script='train_out.sh', cluster_type='local', get_cmd=True, run_cmd=False, prefix='', memory='16GB', wall_time='3:00:00', partition='short', verbose=False)
Trains ARHMM from Jupyter notebook. Note that the configuration file parameters will be overriden with the inputted parameters from the jupyter notebook cell function call.

Parameters:

- **progress_paths (dict)** (*notebook progress dict that contains paths to the pca scores, config, and index files.*)
- **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. Larger kappa = longer syllable durations.*)
- **min_kappa (int)** (*Minimum kappa exponent to train model on. if min_kappa = 3; min(kappas) == 1e3*)
- **max_kappa (int)** (*Maximum kappa exponent to train model on. if min_kappa = 5; min(kappas) == 1e5*)
- **n_models (int)** (*Number of models to spawn to scan kappa values*)
- **scan_scale (str)** (*Scale factor to generate scanning kappa values. ['log', 'linear']*)
- **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*)
- **use_checkpoint (bool)** (*indicates to load a previously saved checkpoint*)
- **alpha (float)** (*probability prior distribution for syllable transition rate.*)
- **gamma (float)** (*probability prior distribution for PCs explaining syllable states. Smaller gamma = steeper PC_Scree plot.*)
- **select_groups (bool)** (*indicates to display all sessions and choose subset of groups to model alone.*)
- **check_every (int)** (*number of iterations between each training convergence check.*)
- **select_groups (bool)** (*indicates whether to interactively select data to model by group name.*)
- **get_cmd (bool)** (*indicates to print all the kappa scan learn-model command outputs.*)
- **run_cmd (bool)** (*indicates to run all the kappa scan learn-model commands.*)
- **percent_split (int)** (*train-validation data split ratio percentage.*)
- **output_dir (str)** (*directory to store multiple trained models via kappa-scan*)
- **out_script (str)** (*name of the script containing all the kappa scanning commands.*)
- **cluster_type (str)** (*name of cluster to run model training on; either ['local', 'slurm']*)
- **prefix (str)** (*slurm command prefix with job specification parameters.*)
- **memory (str)** (*amount of memory in GB to allocate to each training job.*)
- **wall_time (str)** (*maximum time for a slurm job to run.*)
- **partition (str)** (*slurm partition name to run training jobs on.*)
- **verbose (bool)** (*compute modeling summary (Warning current implementation is can slow down training).*)

Returns:

Return type: None

General Utilities Module

Utility functions for handling loading and saving models and their respective metadata.

`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.`**`count_frames`**(`data_dict=None`, `input_file=None`, `var_name='scores'`)
Counts the total number of frames loaded from the PCA scores file.

Parameters:
- **data_dict (OrderedDict)** (*Loaded PCA scores OrderedDict object.*)
- **input_file (str)** (*Path to PCA Scores file to load data_dict if not already data_dict == None*)
- **var_name (str)** (*Path within PCA h5 file to load scores from.*)

Returns: **total_frames (int)**

Return type: total number of counted frames.

`moseq2_model.util.`**`create_command_strings`**(`input_file`, `output_dir`, `config_data`, `kappas`, `model_name_format='model-{}-{}.p'`)

Creates the CLI learn-model N command strings with parameter flags based on the contents of the configuration

dict. Each model will a different kappa value within a given range (for N models to train).

Parameters:
- **input_file (str)** (*Path to PCA Scores*)
- **index_file (str)** (*Path to index file*)
- **output_dir (str)** (*Path to directory to save models in.*)
- **config_data (dict)** (*Configuration parameters dict.*)
- **kappas (list)** (*List of kappa values to assign to model training commands.*)
- **model_name_format (str)** (*Filename string format string.*)

Returns: **command_string (str)**

Return type: CLI learn-model command strings with the requested parameters separated by newline characters

`moseq2_model.util.`**`dict_to_h5`**(`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.**get_current_model**(use_checkpoint, all_checkpoints, train_data, model_parameters)

Checks to see whether user is loading a checkpointed model, if so, loads the latest iteration. Otherwise, will instantiate a new model.

Parameters:
- **use_checkpoint (bool)** (*CLI input parameter indicating user is loading a checkpointed model*)
- **all_checkpoints (list)** (*list of all found checkpoint paths*)
- **train_data (OrderedDict)** (*dictionary of uuid-PC score key-value pairs*)
- **model_parameters (dict)** (*dictionary of required modeling hyperparameters.*)

Returns: **arhmm (ARHMM)** (*instantiated model object including loaded data*) **itr (int)** (*starting iteration number for the model to begin training from.*)

moseq2_model.util.**get_loglikelihoods**(arhmm, data, groups, separate_trans)

Computes the log-likelihoods of the trained ARHMM states.

Parameters:
- **arhmm (ARHMM)** (*Trained ARHMM model.*)
- **data (dict)** (*dict object containing training data keyed by their corresponding UUIDs*)
- **groups (list)** (*list of assigned groups for all corresponding session uuids. (Only used if*) – separate_trans == True.
- **separate_trans (bool)** (*boolean that determines whether to compute separate log-likelihoods*)
- **for each modeled group.**

Returns: **ll (list)**

Return type: list of log-likelihoods for the trained model, len(ll) > 1 if separate_trans==True

moseq2_model.util.**get_parameter_strings**(config_data)

Creates the CLI learn-model parameters string using the given config_data dict contents.

Function checks for the following paramters: [npcs, num_iter, separate_trans, robust, e_step, hold_out, max_states, converge, tolerance].

Parameters:
- **index_file (str)** (*Path to index file.*)
- **config_data (dict)** (*Configuration parameters dict.*)

Returns: **parameters (str)** (*String containing all the requested CLI command parameter flags.*) **prefix (str)** (*Prefix string for the learn-model command, used for Slurm functionality.*)

moseq2_model.util.**get_parameters_from_model**(model)

Get parameter dictionary from model.

Parameters: **model (ARHMM)** (*model to get parameters from.*)

Returns: **parameters (dict)**

Return type: dictionary containing all modeling parameters

moseq2_model.util.**get_scan_range_kappas**(data_dict, config_data)

Helper function that returns the kappa values to train models on based on the user's selected scanning scale range. Different default range values will be selected if min/max_kappa are None. Otherwise, min_kappa and max_kappa represent exponent ranges to get kappa values within.

For example, scan_scale = 'log'; nframes = 1800; min_kappa = 3; max_kappa = 5; n_models = 10; min(kappas) == 1e3; max(kappas) == 1e5; kappas = [1000, 1668, 2782, 4641, 7742, 12915, 21544, 35938, 59948, 100000]

Another Exmaple: nframes = 1800 'scan_scale': 'linear', 'min_kappa': 2, 'max_kappa': 4, 'n_models': 10 min(kappas) == 18 max(kappas) == 18000000 kappas == [18, 2000016, 4000014, 6000012, 8000010, 10000008, 12000006, 14000004, 16000002, 18000000]

Parameters:
- **data_dict (OrderedDict)** (*Loaded PCA score dictionary.*)
- **config_data (dict)** (*Configuration parameters dict.*)

Returns: **kappas (list)**

Return type: list of ints corresponding to the kappa value for each model.

`moseq2_model.util.`**`get_session_groupings`** (`data_metadata`, `all_keys`, `hold_out_list`)
Creates a list or tuple of assigned groups for training and (optionally) held out data.

Parameters:
- **data_metadata (dict)** (*dict containing session group information*)
- **groups (list)** (*list of all session groups*)
- **all_keys (list)** (*list of all corresponding included session UUIDs*)
- **hold_out_list (list)** (*list of held-out uuids*)

Returns: **groupings (list or tuple)** (*1/2-tuple containing lists of train groups*) *and held-out groups (if held_out_list exists)*

`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.`**`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_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.`**`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

Helper functions for reading data from index files, and preparing metadata prior to training.

`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_session_metadata`**(`index`)

Reads index file verbose session metadata to display in case user prompts for interactive group selection.

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

Returns: **index_data (dict)** (*dict object of loaded index file*) **metadata (dict)** (*dict of lists corresponding to session metadata to display*)

`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: **training_data (OrderedDict)** (*the split percentage of the training data.*) **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_helper`**(`groups`, `lls`, `legend`, `iterations`, `ll_type='train'`)

Helper function to plot the training and validation log-likelihoods

over the each model training iteration.

Parameters:

- **groups (list)** (*list of group names that the model was trained on.*)
- **lls (list)** (*list of log-likelihoods over each iteration.*)
- **legend (list)** (*list of legend labels for each group's log-likelihoods curve.*)
- **iterations (list)** (*range() generated list indicated x-axis length.*)
- **ll_type (str)** (*string to indicate (in the legend) whether plotting training or validation curves.*)
- **sep_trans (bool)** (*indicates whether there is more than one set on log-likelihoods.*)

Returns:

Return type: None

`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`, `select_groups=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

Wrapper functions for all functionality included in MoSeq2-Model 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_model.helpers.wrappers.`**`kappa_scan_fit_models_wrapper`**(`input_file`, `config_data`, `output_dir`)

Wrapper function that spools multiple model training commands for different kappa values within a

given range. (Either n models with kappa values equally spaced between a min and max value, or choosing n kappa values ranging in factors of 10 starting from nframes/100 for n=number of models).

Parameters:
- **input_file (str)** (*Path to PCA Scores*)
- **config_data (dict)** (*Dict containing model training parameters*)
- **output_dir (str)** (*Path to output directory to save trained models*)

Returns: **command_string (str)** – (or parallel in case of cluster-type=='slurm') model training commands.

Return type: CLI command string to sequential

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

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.*)

Returns:

Return type: None

moseq2_model.train Package

Train - Fit Module

Train - Label Utilities Module

Train - Model Module

ARHMM model initialization utilities.

`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

ARHMM utility functions

`moseq2_model.train.util.`**`check_convergence`**(`iter_lls`)

Checks whether the model log-likelihood increase is below the given tolerance threshold, signalling that the

modeling has converged.

Reference for Maximum Likelihood Estimation: https://medium.com/@rrfd/what-is-maximum-likelihood-estimation-examples-in-python-791153818030

Parameters: **iter_lls (list)** (*List of computed log-likelihoods from previous iterations*)

Returns: **converged (bool)**

Return type: Boolean to decide whether to stop model training if log-likelihoods have converged

`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, ncpus=1, checkpoint_freq=None, checkpoint_file=None, start=0, progress_kwargs={}, num_frames=[1], train_data=None, val_data=None, separate_trans=False, groups=None, converge=False, verbose=False, check_every=2)`

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)** (*iteration frequency where model predictions are saved to a file*)
- **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 is not None)*)
- **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)*)
- **converge (bool)** (*Train model until the log-likelihoods converge*)
- **verbose (bool)** (*Compute model summary.*)
- **check_every (int)** (*iteration frequency to check whether the model log-likelihoods have converged*)

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.`**`training_checkpoint`**`(model, itr, checkpoint_file, checkpoint_freq)`

Formats the model checkpoint filename and saves the model checkpoint

Parameters:

- **model (ARHMM)** (*Model being trained.*)
- **itr (itr)** (*Current modeling iteration.*)
- **checkpoint_file (str)** (*Model checkpoint file name.*)
- **checkpoint_freq (int)** (*Model checkpointing iteration frequency*)

`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

Symbols

A

C

D

F

O

P

R

S

T

W

Z

Python Module Index

m