PySeqLab Documentation

Release 1.4.0

aa

Jun 01, 2017

Overview:

PySeqLab is an open source package for performing supervised learning in structured prediction tasks. It implements conditional random fields (CRFs) models from (1) first-order to higher-order linear-chain CRFs, and from (2) first-order to higher-order semi-Markov CRFs (semi-CRFs).

CHAPTER 1

Download

The latest release of PySeqLab could be obtained from Bitbucket.

Source code is also available on Bitbucket.

CHAPTER 2

Installation

To install the latest version of PySeqLab:

```
> pip install git+https://bitbucket.org/A_2/pyseqlab.git
```

Requirements

- Python 3.4.0
- NumPy 1.8.0
- SciPy 0.13 (only for using L-BFGS-B optimization method)

Anaconda

The easiest way to install and manage Python packages on various OS platforms is through Anaconda. Once installed, any package (even if not available on `Anaconda` channel) could be installed using `pip`. For more info about managing and installing packages using `Anaconda` refer to this link.

CHAPTER 3

Methods and Tutorials

In this section, we explain the different building blocks and concepts used in `PySeqLab` package. We provide a series of tutorials aimed at explaining the different aspects of the package in order to help users build their own applications.

1. Sequences, segments and input file format
2. Attributes, features, feature templates and features extraction
3. CRFs model building and training

CHAPTER 4

Applications

To illustrate the use of `PySeqLab` package, we give examples from multiple domains:

Natural language processing (NLP)

Chunker

We show how to build and train a `chunker` using CoNLL 2000 dataset in this tutorial. Moreover, the project repository is available on bitbucket.

Part-of-Speech tagger

Similarly, we demonstrate how to train a `part-of-speech tagger` using GENIA corpus 3.02 part of speech annotation in this tutorial. The project repository is available on bitbucket.

Bio-named entity recognizer

We conclude the NLP examples by this tutorial on training a biomedical named entity recognizer (`BioNER`) using BioNLP/NLPBA 2004 dataset. The project repository is available on bitbucket.

Eukaryotic splice-junction sequences predictor

We demonstrate how to train a `splice-junction classifier` using Molecular Biology (Splice-junction Gene Sequences) Data Set in this tutorial. Moreover, the project repository is available on bitbucket.

Human activity recognition (HAR) classifiers

We demonstrate how to train a `modes of locomotion classifier` and `gestures movements classifier` using the OPPORTUNITY activity recognition dataset in this tutorial. Moreover, the project repository is available on bitbucket.

CHAPTER 5

pyseqlab package

Submodules

pyseqlab.attributes_extraction module

@author: ahmed allam <ahmed.allam@yale.edu>

class pyseqlab.attributes_extraction.**AttributeScaler**(*scaling_info*, *method*)

Bases: object

attribute scalar class to scale/standardize continuous attributes/features

Parameters

- **scaling_info** – dictionary comprising the relevant info for performing standardization
- **method** – string defining the method of scaling {rescaling, standardization}

scaling_info

dictionary comprising the relevant info for performing standardization

method

string defining the method of scaling {rescaling, standardization}

Example:

```
in case of *standardization*:
    - scaling_info has the form: scaling_info[attr_name] = {'mean':value,'sd
↪':value}
in case of *rescaling*
    - scaling_info has the form: scaling_info[attr_name] = {'max':value, 'min
↪':value}
```

save(*folder_dir*)

save relevant info about the scaler on disk

Parameters **folder_dir** – string representing directory where files are pickled/dumped

scale_continuous_attributes(*seq*, *boundaries*)
scale continuous attributes of a sequence for a list of boundaries

Parameters

- **seq** – a sequence instance of `SequenceStruct`
- **boundaries** – list of boundaries `[(1,1), (2,2),...,]`

transform_scale(*x*, *xref_min*, *xref_max*)
transforms feature value to scale from [-1,1]

class pyseqlab.attributes_extraction.**GenericAttributeExtractor**(*attr_desc*)
Bases: `object`

Generic attribute extractor class implementing observation functions that generates attributes from tokens/observations

Parameters **attr_desc** – dictionary defining the atomic observation/attribute names including the encoding of such attribute (i.e. {continuous, categorical}}

attr_desc
dictionary defining the atomic observation/attribute names including the encoding of such attribute (i.e. {continuous, categorical}}

seg_attr
dictionary comprising the extracted attributes per each boundary of a sequence

determine_attr_encoding(*attr_desc*)

generate_attributes(*seq*, *boundaries*)

group_attributes()
function to group attributes based on the encoding type (i.e. continuous vs. categorical)

class pyseqlab.attributes_extraction.**NERSegmentAttributeExtractor**
Bases: *pyseqlab.attributes_extraction.GenericAttributeExtractor*

class implementing observation functions that generates attributes from word tokens/observations

Parameters **attr_desc** – dictionary defining the atomic observation/attribute names including the encoding of such attribute (i.e. {continuous, categorical}}

attr_desc
dictionary defining the atomic observation/attribute names including the encoding of such attribute (i.e. {continuous, categorical}}

seg_attr
dictionary comprising the extracted attributes per each boundary of a sequence

generate_attributes(*seq*, *boundaries*)
generate attributes of the sequence observations in a specified list of boundaries

Parameters

- **seq** – a sequence instance of `SequenceStruct`
- **boundaries** – list of boundaries [(1,1), (2,2),...,]

Note: the generated attributes are saved first in *seg_attr* and then passed to the **'seq.seg_attr'**. In other words, at the end `seg_att` is always cleared

---

generate_attributes_desc()
define attributes by including description and encoding of each observation or observation feature

get_degenerateshape(*boundary*)
get degenerate shape of segment

Parameters **boundary** – tuple (u,v) that marks beginning and end of a segment

get_num_chars(*boundary*, *filter_out=' '*)
get the number of characters of a segment

Parameters

- **boundary** – tuple (u,v) that marks beginning and end of a segment
- **filter_out** – string the default separator between attributes

get_seg_bagofattributes(*boundary*, *attr_names*, *sep=' '*)
implements the bag-of-attributes concept within a segment

Parameters

- **boundary** – tuple (u,v) representing current boundary
- **attr_names** – list of names of the atomic observations/attributes
- **sep** – separator (by default is the space)

Note: it can be used **only** with attributes that have binary_encoding type set equal True

---

get_seg_length(*boundary*)
get the length of a segment

Parameters **boundary** – tuple (u,v) that marks beginning and end of a segment

get_shape(*boundary*)
get shape of segment

Parameters **boundary** – tuple (u,v) that marks beginning and end of a segment

pyseqlab.crf_learning module

@author: Ahmed Allam <ahmed.allam@yale.edu>

class pyseqlab.crf_learning.**Evaluator**(*model_repr*)
Bases: object

Evaluator class to evaluate performance of the models

Parameters **model_repr** – the CRF model representation that has a suffix of *ModelRepresentation* such as HOCRFADModelRepresentation

model_repr
the CRF model representation that has a suffix of *ModelRepresentation* such as HOCRFADModelRepresentation

---

Note: this class is **EXPERIMENTAL/work in progress*** and does not support evaluation of segment learning. Use instead *SeqDecodingEvaluator* for evaluating models learned using **sequence** learning.

---

compute_model_performance(*Y_seqs_dict*, *metric*, *output_file*, *states_notation*)
compute the performance of the model

Parameters

- **Y_seqs_dict** – dictionary where each sequence has the reference label sequence and its corresponding predicted sequence. It has the following form `{seq_id:{'Y_ref':[reference_ylabels], 'Y_pred':[predicted_ylabels]}}`
- **metric** – evaluation metric that could take one of {'f1', 'precision', 'recall', 'accuracy'}
- **output_file** – file where to output the evaluation result
- **states_notation** – notation used to code the state (i.e. BIO)

compute_tags_confusionmatrix(*Y_ref*, *Y_pred*, *transformed_codebook_rev*, *M*)
compute confusion matrix on the level of the tag/state

Parameters

- **Y_ref** – list of reference label sequence (represented by the states code)
- **Y_pred** – list of predicted label sequence (represented by the states code)
- **transformed_codebook** – the transformed codebook of the new identified states
- **M** – number of states

map_states_to_num(*Y*, *state_mapper*, *transformed_codebook*, *M*)
map states to their code/number using the *Y_codebook*

Parameters

- **Y** – list representing label sequence
- **state_mapper** – mapper between the old and new generated states generated from `tranform_codebook()` method
- **trasformed_codebook** – the transformed codebook of the new identified states
- **M** – number of states

Note: we give one unique index for tags that did not occur in the training data such as len(Y_codebook)

---

transform_codebook(*Y_codebook*, *prefixes*)
map states coded in BIO notation to their original states value

Parameters

- **Y_codebook** – dictionary of states each assigned a unique integer
- **prefixes** – tuple of prefix notation used such as ("B-","I-") for BIO

class pyseqlab.crf_learning.**Learner**(*crf_model*)
Bases: object

learner used for training CRF models supporting search- and gradient-based learning methods

Parameters **crf_model** – an instance of CRF models such as HOCRFAD

Keyword Arguments

- **crf_model** – an instance of CRF models such as HOCRFAD

- **training_description** – dictionary that will include the training specification of the model

cleanup()
End of training – cleanup

train_model(*w0*, *seqs_id*, *optimization_options*, *working_dir*, *save_model=True*)
the **MAIN** method for training models using the various options available

Parameters

- **w0** – numpy vector representing initial weights for the parameters
- **seqs_id** – list of integers representing the sequence ids
- **optimization_options** – dictionary specifying the training method
- **working_dir** – string representing the directory where the model data and generated files will be saved

Keyword Arguments **save_model** – boolean specifying if to save the final model

Example

The available options for training are:

- *SGA* for stochastic gradient ascent
- *SGA-ADADELTA* for stochastic gradient ascent using ADADELTA approach
- *BFGS* or *L-BFGS-B* for optimization using second order information (hessian matrix)
- *SVRG* for stochastic variance reduced gradient method
- *COLLINS-PERCEPTRON* for structured perceptron
- *SAPO* for Search-based Probabilistic Online Learning Algorithm (SAPO) (an adapted version)

For example possible specification of the optimization options are:

```
 1) {'method': 'SGA-ADADELTA'
    'regularization_type': {'l1', 'l2'}
    'regularization_value': float
    'num_epochs': integer
    'tolerance': float
    'rho': float
    'epsilon': float
   }


2) {'method': 'SGA' or 'SVRG'
    'regularization_type': {'l1', 'l2'}
    'regularization_value': float
    'num_epochs': integer
    'tolerance': float
    'learning_rate_schedule': one of ("bottu", "exponential_decay", "t_
↪inverse", "constant")
    't0': float
    'alpha': float
    'eta0': float
   }
```

```
3) {'method': 'L-BFGS-B' or 'BFGS'
    'regularization_type': 'l2'
    'regularization_value': float
    'disp': False
    'maxls': 20,
    'iprint': -1,
    'gtol': 1e-05,
    'eps': 1e-08,
    'maxiter': 15000,
    'ftol': 2.220446049250313e-09,
    'maxcor': 10,
    'maxfun': 15000
    }


4) {'method': 'COLLINS-PERCEPTRON'
    'regularization_type': {'l1', 'l2'}
    'regularization_value': float
    'num_epochs': integer
    'update_type':{'early', 'max-fast', 'max-exhaustive', 'latest'}
    'shuffle_seq': boolean
    'beam_size': integer
    'avg_scheme': {'avg_error', 'avg_uniform'}
    'tolerance': float
   }


5) {'method': 'SAPO'
    'regularization_type': {'l2'}
    'regularization_value': float
    'num_epochs': integer
    'update_type':'early'
    'shuffle_seq': boolean
    'beam_size': integer
    'topK': integer
    'tolerance': float
   }
```

class pyseqlab.crf_learning.**SeqDecodingEvaluator**(*model_repr*)

Bases: object

Evaluator class to evaluate performance of the models

Parameters **model_repr** – the CRF model representation that has a suffix of *ModelRepresentation* such as HOCRFADModelRepresentation

model_repr

the CRF model representation that has a suffix of *ModelRepresentation* such as HOCRFADModelRepresentation

---

Note: this class does not support evaluation of segment learning (i.e. notations that include IOB2/BIO notation)

---

compute_states_confmatrix(*Y_seqs_dict*)

compute/generate the confusion matrix for each state

Parameters **Y_seqs_dict** – dictionary where each sequence has the ref-

erence label sequence and its corresponding predicted sequence. It has the following form `{seq_id:{'Y_ref':[reference_ylabels], 'Y_pred':[predicted_ylabels]}}`

get_performance_metric(*taglevel_performance*, *metric*, *exclude_states=[]*)
compute the performance of the model using a requested metric

Parameters

- **taglevel_performance** – *numpy* array with Mx2x2 dimension. For every state code a 2x2 confusion matrix is included. It is computed using `compute_model_performance()`
- **metric** – evaluation metric that could take one of `{'f1', 'precision', 'recall', 'accuracy'}`

Keyword Arguments **exclude_states** – list (default empty list) of states to exclude from the computation. Usually, in NER applications the non-entity symbol such as 'O' is excluded from the computation. Example: If `exclude_states = ['O']`, this will replicate the behavior of conlleval script

map_states_to_num(*Y*, *Y_codebook*, *M*)
map states to their code/number using the *Y_codebook*

Parameters

- **Y** – list representing label sequence
- **Y_codebook** – dictionary containing the states as keys and the assigned unique code as values
- **M** – number of states

Note: we give one unique index for tags that did not occur in the training data such as len(Y_codebook)

pyseqlab.features_extraction module

@author: ahmed allam <ahmed.allam@yale.edu>

class pyseqlab.features_extraction.**FOFeatureExtractor**(*templateX*, *templateY*, *attr_desc*, *start_state=True*)
Bases: *pyseqlab.features_extraction.FeatureExtractor*

Feature extractor class for first order CRF models

it supports the addition of start_state and **potentially** stop_state in the future release

Parameters

- **templateX** – dictionary specifying template to follow for observation features extraction. It has the form: `{attr_name: {x_offset:tuple(y_offsets)}}` e.g. `{'w': {(0,):((0,), (-1,0), (-2,-1,0))}}`
- **templateY** – dictionary specifying template to follow for y pattern features extraction. It has the form: `{Y: tuple(y_offsets)}` e.g. `{'Y': ((0,), (-1,0), (-2,-1,0))}`

- **attr_desc** – dictionary containing description and the encoding of the attributes/observations e.g. `attr_desc['w'] = {'description':'the word/token','encoding':'categorical'}`. For more details/info check the *attr_desc* of the `NERSegmentAttributeExtractor`
- **start_state** – boolean indicating if __START__ state is required in the model

templateX
dictionary specifying template to follow for observation features extraction. It has the form: `{attr_name: {x_offset:tuple(y_offsets)}}` e.g. `{'w': {(0,):((0,), (-1,0), (-2,-1,0))}}`

templateY
dictionary specifying template to follow for y pattern features extraction. It has the form: `{Y: tuple(y_offsets)}` e.g. `{'Y': ((0,), (-1,0), (-2,-1,0))}`

attr_desc
dictionary containing description and the encoding of the attributes/observations e.g. `attr_desc['w'] = {'description':'the word/token','encoding':'categorical'}`. For more details/info check the *attr_desc* of the `NERSegmentAttributeExtractor`

start_state
boolean indicating if __START__ state is required in the model

---

Note: The addition of this class is to add support for __START__ and potentially __STOP__ states

---

extract_features_Y(*seq*, *boundary*, *templateY*)
extract y pattern features for a given sequence and template Y

Parameters

- **seq** – a sequence instance of `SequenceStruct`
- **boundary** – tuple (u,v) representing current boundary
- **templateY** – dictionary specifying template to follow for extraction. It has the form: `{Y: tuple(y_offsets)}` e.g. `{'Y': ((0,), (-1,0)}`

class `pyseqlab.features_extraction.`**FeatureExtractor**(*templateX*, *templateY*, *attr_desc*)
Bases: `object`

Generic feature extractor class that contains feature functions/templates

Parameters

- **templateX** – dictionary specifying template to follow for observation features extraction. It has the form: `{attr_name: {x_offset:tuple(y_offsets)}}`. e.g. `{'w': {(0,):((0,), (-1,0), (-2,-1,0))}}`
- **templateY** – dictionary specifying template to follow for y pattern features extraction. It has the form: `{Y: tuple(y_offsets)}`. e.g. `{'Y': ((0,), (-1,0), (-2,-1,0))}`
- **attr_desc** – dictionary containing description and the encoding of the attributes/observations e.g. attr_desc['w'] = {'description':'the word/token','encoding':'categorical'} for more details/info check the *attr_desc* of the `NERSegmentAttributeExtractor`

template_X
dictionary specifying template to follow for observation features extraction. It has the form:

`{attr_name: {x_offset:tuple(y_offsets)}}` e.g. `{'w': {(0,):((0,), (-1, 0), (-2,-1,0))}}`

template_Y
dictionary specifying template to follow for y pattern features extraction. It has the form: `{Y: tuple(y_offsets)}` e.g. `{'Y': ((0,), (-1,0), (-2,-1,0))}`

attr_desc
dictionary containing description and the encoding of the attributes/observations. e.g. `attr_desc['w'] = {'description':'the word/token', 'encoding':'categorical'}`. For more details/info check the *attr_desc* of the `NERSegmentAttributeExtractor`

aggregate_seq_features(*features*, *boundaries*)
aggregate features across all boundaries

it is usually used to aggregate features in the dictionary obtained from *extract_seq_features_perboundary()* function

Parameters

- **features** – dictionary of sequence features per boundary
- **boundaries** – list of boundaries where detected features are aggregated

attr_represent_funcmapper()
assign a representation function based on the encoding (i.e. categorical or continuous) of each attribute name

extract_features_X(*seq*, *boundary*)
extract observation features for a given sequence at a specified boundary

Parameters

- **seq** – a sequence instance of `SequenceStruct`
- **boundary** – tuple (u,v) representing current boundary

extract_features_XY(*seq*, *boundary*, *seg_features=None*)
extract/join observation features with y pattern features as specified *template_X*

Parameters

- **seq** – a sequence instance of `SequenceStruct`
- **boundary** – tuple (u,v) representing current boundary

Keywords Arguments: seg_features: optional dictionary of observation features

Example:

```
template_X = {'w': {(0,):((0,), (-1,0), (-2,-1,0))}}
Using template_X the function will extract all unigram features of the observation 'w' (0, )
and join it with:
    - zero-order y pattern features (0,)
    - first-order y pattern features (-1, 0)
    - second-order y pattern features (-2, -1, 0)
template_Y = {'Y': ((0,), (-1,0), (-2,-1,0))}
```

extract_features_Y(*seq*, *boundary*, *templateY*)
extract y pattern features for a given sequence and template Y

Parameters

- **seq** – a sequence instance of `SequenceStruct`
- **boundary** – tuple (u,v) representing current boundary
- **templateY** – dictionary specifying template to follow for extraction. It has the form: {Y: tuple(y_offsets)} e.g. `{'Y': ((0,), (-1,0), (-2,-1,0))}`

extract_seq_features_perboundary(*seq*, *seg_features=None*)
extract features (observation and y pattern features) per boundary

Parameters **seq** – a sequence instance of `SequenceStruct`

Keywords Arguments: seg_features: optional dictionary of observation features

flatten_segfeatures(*seg_features*)
flatten observation features dictionary

Parameters **seg_features** – dictionary of observation features

lookup_features_X(*seq*, *boundary*)
lookup observation features for a given sequence using varying boundaries (i.e. g(X, u, v))

Parameters

- **seq** – a sequence instance of `SequenceStruct`
- **boundary** – tuple (u,v) representing current boundary

lookup_seq_modelactivefeatures(*seq*, *model*, *learning=False*)
lookup/search model active features for a given sequence using varying boundaries (i.e. g(X, u, v))

Parameters

- **seq** – a sequence instance of `SequenceStruct`
- **model** – a model representation instance of the CRF class (i.e. the class having *Model-Representation* suffix)

Keyword Arguments **learning** – optional boolean indicating if this function is used wile learning model parameters

save(*folder_dir*)
store the templates used – templateX and templateY

template_X

template_Y

class pyseqlab.features_extraction.**FeatureFilter**(*filter_info*)
Bases: `object`

class for applying filters by y pattern or feature counts

Parameters **filter_info** – dictionary that contains type of filter to be applied

filter_info
dictionary that contains type of filter to be applied

rel_func
dictionary of function map

Example:

```
filter_info dictionary has three keys:
    - `filter_type` to define the type of filter either {count or pattern}
    - `filter_val` to define either the y pattern or threshold count
    - `filter_relation` to define how the filter should be applied


*count filter*:
    - ``filter_info = {'filter_type': 'count', 'filter_val':5, 'filter_relation':'
↪<'}``
      this filter would delete all features that have count less than five

*pattern filter*:
    - ``filter_info = {'filter_type': 'pattern', 'filter_val': {"O|L", "L|L"},
↪'filter_relation':'in'}``
      this filter would delete all features that have associated y pattern ["O|L",
↪ "L|L"]
```

apply_filter (*featuresum_dict*)
apply define filter on model features dictionary

Parameters **featuresum_dict** – dictoinary that represents model features similar to *modelfeatures* attribute in one of model representation instances

class pyseqlab.features_extraction.**HOFeatureExtractor** (*templateX*, *templateY*, *attr_desc*)
Bases: *pyseqlab.features_extraction.FeatureExtractor*

Feature extractor class for higher order CRF models

class pyseqlab.features_extraction.**SeqsRepresenter** (*attr_extractor*, *fextractor*)
Bases: object

Sequence representer class that prepares, pre-process and transform sequences for learning/decoding tasks

Parameters

- **attr_extractor** – instance of attribute extractor class such as NERSegmentAttributeExtractor it is used to apply defined observation functions generating features for the observations
- **fextractor** – instance of feature extractor class such as *FeatureExtractor* it is used to extract features from the observations and generated observation features using the observation functions

attr_extractor
instance of attribute extractor class such as NERSegmentAttributeExtractor

fextractor
instance of feature extractor class such as *FeatureExtractor*

attr_scaler
instance of scaler class AttributeScaler it is used for scaling features that are continuous –not categorical (using standardization or rescaling)

aggregate_gfeatures (*gfeatures*, *boundaries*)
aggregate global features using specified list of boundaries

Parameters

- **gfeatures** – dictionary representing the extracted sequence features (i.e F(X, Y))
- **boundaries** – list of boundaries to use for aggregating global features

create_model(*seqs_id*, *seqs_info*, *model_repr_class*, *filter_obj=None*)
aggregate all identified features in the training sequences to build one model

Main task:

- use the sequences assigned in the training set to build the model
- takes the union of the detected global feature functions $F_j(X, Y)$ for each chosen parsed sequence from the training set to form the set of model features
- construct the tag set Y_set (i.e. possible tags assumed by y_t) using the chosen parsed sequences from the training data set
- determine the longest segment length (if applicable)
- apply feature filter (if applicable)

Parameters

- **seqs_id** – list of sequence ids to be processed
- **seqs_info** – dictionary comprising the the info about the prepared sequences
- **model_repr_class** – class name of model representation (i.e. class that has suffix *ModelRepresentation* such as `HOCRFModelRepresentation`)

Keyword Arguments **filter_obj** – optional instance of *FeatureFilter* class to apply filter

---

Note: it requires that the sequences have been already parsed and global features were generated using *extract_seqs_globalfeatures()*

---

extract_seqs_globalfeatures(*seqs_id*, *seqs_info*, *dump_gfeat_perboundary=False*)
extract globalfeatures (i.e. F(X,Y)) from every sequence

Main task:

- parses each sequence and generates global feature $F_j(X, Y) = \sum_{t=1}^{T} f_j(X, Y)$
- for each sequence we obtain a set of generated global feature functions where each $F_j(X, Y)$ represents the sum of the value of its corresponding low-level/local feature function $f_j(X, Y)$ (i.e. $F_j(X, Y) = \sum_{t=1}^{T+1} f_j(X, Y)$)
- saves all the results on disk

Parameters

- **seqs_id** – list of sequence ids to be processed
- **seqs_info** – dictionary comprising the the info about the prepared sequences

---

Note: it requires that the sequences have been already parsed and preprocessed (if applicable)

---

extract_seqs_modelactivefeatures(*seqs_id*, *seqs_info*, *model*, *output_foldername*, *learning=False*)
identify for every sequence model active states and features

Main task:

- generate attributes for all segments with length 1 to maximum length defined in the model it is an optional step and only applied in case of having segmentation problems
- generate segment features, potential activated states and a representation of segment features to be used potentially while learning
- dump all info on disk

Parameters

- **seqs_id** – list of sequence ids to be processed
- **seqs_info** – dictionary comprising the the info about the prepared sequences
- **model** – an instance of model representation class (i.e. class that has suffix *ModelRepresentation* such as `HOCRFModelRepresentation`)
- **output_foldername** – string representing the name of the root folder to be created for containing all saved info

Keyword Arguments **learning** – boolean indicating if this function used for the purpose of learning (model weights optimization)

Note: seqs_info dictionary will be updated by including the directory of the saved generatd info

---

feature_extractor

get_imposterseq_globalfeatures(*seq_id*, *seqs_info*, *y_imposter*, *seg_other_symbol=None*)
get an imposter decoded sequence

Main task:

- to be used for processing a sequence, generating global features and return back without storing/saving intermediary results on disk

Parameters

- **seqs_id** – list of sequence ids to be processed
- **seqs_info** – dictionary comprising the the info about the prepared sequences
- **y_imposter** – list of labels (y tags) decoded using a decoder

Keyword Arguments **seg_other_symbol** – in case of segmentation, this represents the non-entity symbol label used. Otherwise, it is None (default) which translates to be a sequence labeling problem.

get_seq_activatedstates(*seq_id*, *seqs_info*)
retrieve identified activated states that were saved on disk using *seqs_info*

Parameters

- **seqs_id** – list of sequence ids to be processed
- **seqs_info** – dictionary comprising the the info about the prepared sequences

---

Note: this data was generated using *extract_seqs_modelactivefeatures()*

---

get_seq_activefeatures(*seq_id*, *seqs_info*)
retrieve sequence model active features that are identified

Parameters

- **seqs_id** – list of sequence ids to be processed
- **seqs_info** – dictionary comprising the the info about the prepared sequences

get_seq_globalfeatures(*seq_id*, *seqs_info*, *per_boundary=True*)
retrieves the global features available for a given sequence (i.e. $F(X, Y)$ for all $j \in [1...J]$)

Parameters

- **seqs_id** – list of sequence ids to be processed
- **seqs_info** – dictionary comprising the the info about the prepared sequences

Keyword Arguments **per_boundary** – boolean specifying if the global features representation should be per boundary or aggregated across the whole sequence

get_seq_lsegfeatures(*seq_id*, *seqs_info*)
retrieve segment features that were extracted with a modified representation for the purpose of parameter learning

Parameters

- **seqs_id** – list of sequence ids to be processed
- **seqs_info** – dictionary comprising the the info about the prepared sequences

---

Note: this data was generated using *extract_seqs_modelactivefeatures()*

---

get_seq_segfeatures(*seq_id*, *seqs_info*)
retrieve segment features that were extracted and saved on disk using *seqs_info*

Parameters

- **seqs_id** – list of sequence ids to be processed
- **seqs_info** – dictionary comprising the the info about the prepared sequences

---

Note: this data was generated using *extract_seqs_modelactivefeatures()*

---

static load_seq(*seq_id*, *seqs_info*)
load dumped sequences on disk

Parameters **seqs_info** – dictionary comprising the the info about the prepared sequences

prepare_seqs(*seqs_dict*, *corpus_name*, *working_dir*, *unique_id=True*, *log_progress=True*)
prepare sequences to be used in the CRF models

Main task:

- generate attributes (i.e. apply observation functions) on the sequence
- create a directory for every sequence where we save the relevant data
- create and return seqs_info dictionary comprising info about the prepared sequences

Parameters

- **seqs_dict** – dictionary containing sequences and corresponding ids where each sequence is an instance of the `SequenceStruct` class

- **corpus_name** – string specifying the name of the corpus that will be used as corpus folder name
- **working_dir** – string representing the directory where the parsing and saving info on disk will occur
- **unique_id** – boolean indicating if the generated corpus folder will include a generated id

Returns dictionary comprising the the info about the prepared sequences

Return type *seqs_info* (dictionary)

Example:

```
seqs_info = {'seq_id':{'globalfeatures_dir':directory,
                       'T': length of sequence
                       'L': length of the longest segment
                      }
                     ....
             }
```

preprocess_attributes(*seqs_id*, *seqs_info*, *method='rescaling'*)

preprocess sequences by generating attributes for segments with L >1

Main task:

- generate attributes (i.e. apply observation functions) on segments (i.e. L>1)
- scale continuous attributes and building the relevant scaling info needed
- create a directory for every sequence where we save the relevant data

Parameters

- **seqs_id** – list of sequence ids to be processed
- **seqs_info** – dictionary comprising the the info about the prepared sequences

Keyword Arguments **method** – string determining the type of scaling (if applicable) it supports {standardization, rescaling}

represent_gfeatures(*gfeatures*, *model*, *boundaries=None*)

represent extracted sequence global features

two representation could be applied:

- 1. features identified by boundary (i.e. f(X,Y))
- 2. features identified and aggregated across all positions in the sequence (i.e. F(X, Y))

Parameters

- **gfeatures** – dictionary representing the extracted sequence features (i.e F(X, Y))
- **model** – an instance of model representation class (i.e. class that has suffix ModelRepresentation such as `HOCRFModelRepresentation`)

Keyword Arguments **boundaries** – if specified (i.e. list of boundaries), then the required representation is global features per boundary (i.e. option (1)) else (i.e. None or empty list), then the required representation is the aggregated global features (option(2))

save (*folder_dir*)
save essential info about feature extractor

Parameters **folder_dir** – string representing directory where files are pickled/dumped

scale_attributes (*seqs_id*, *seqs_info*)
scale continuous attributes

Parameters

- **seqs_id** – list of sequence ids to be processed
- **seqs_info** – dictionary comprising the the info about the prepared sequences

pyseqlab.features_extraction.**main**()

pyseqlab.fo_crf module

@author: ahmed allam <ahmed.allam@yale.edu>

class pyseqlab.fo_crf.**FirstOrderCRF**(*model*, *seqs_representer*, *seqs_info*, *load_info_fromdisk=5*)
Bases: *pyseqlab.linear_chain_crf.LCRF*

first-order CRF model

Parameters

- **model** – an instance of *FirstOrderCRFModelRepresentation* class
- **seqs_representer** – an instance of SeqsRepresenter class
- **seqs_info** – dictionary holding sequences info

Keyword Arguments **load_info_fromdisk** – integer from 0 to 5 specifying number of cached data to be kept in memory. 0 means keep everything while 5 means load everything from disk

model
an instance of *FirstOrderCRFModelRepresentation* class

weights
a numpy vector representing feature weights

seqs_representer
an instance of pyseqlab.feature_extraction.SeqsRepresenter class

seqs_info
dictionary holding sequences info

beam_size
determines the size of the beam for state pruning

fun_dict
a function map

def_cached_entities
a list of the names of cached entities sorted (descending) based on estimated space required in memory

cached_entitites (*load_info_fromdisk*)
construct list of names of cached entities in memory

compute_backward_vec (*w*, *seq_id*)
compute the backward matrix (beta matrix)

Parameters

- **w** – weight vector (numpy vector)
- **seq_id** – integer representing unique id assigned to the sequence

Note: potential matrix per boundary dictionary should be available in `seqs.info`

compute_feature_expectation(*seq_id*, *P_marginals*, *grad*)
compute the features expectations (i.e. expected count of the feature based on learned model)

Parameters

- **seq_id** – integer representing unique id assigned to the sequence
- **P_marginals** – probability matrix for y patterns at each position in time
- **grad** – numpy vector with dimension equal to the weight vector. It represents the gradient that will be computed using the feature expectation and the global features of the sequence

Note:

•activefeatures (per boundary) dictionary should be available in `seqs.info`

•P_marginal (marginal probability matrix) should be available in `seqs.info`

compute_forward_vec(*w*, *seq_id*)
compute the forward matrix (alpha matrix)

Parameters

- **w** – weight vector (numpy vector)
- **seq_id** – integer representing unique id assigned to the sequence

Note: activefeatures need to be loaded first in `seqs.info`

compute_marginals(*seq_id*)
compute the marginal (i.e. probability of each y pattern at each position)

Parameters **seq_id** – integer representing unique id assigned to the sequence

Note:

•potential matrix per boundary dictionary should be available in `seqs.info`

•alpha matrix should be available in `seqs.info`

•beta matrix should be available in `seqs.info`

•Z (i.e. P(x)) should be available in `seqs.info`

compute_potential(*w*, *active_features*)
compute the potential matrix of active features in a specified boundary

Parameters

- **w** – weight vector (numpy vector)
- **active_features** – dictionary of activated features in a specified boundary

perstate_posterior_decoding(*w*, *seq_id*)
decode sequences using posterior probability (per state) decoder

Parameters

- **w** – weight vector (numpy vector)
- **seq_id** – integer representing unique id assigned to the sequence

prune_states(*j*, *score_mat*, *beam_size*)
prune states that fall off the specified beam size

Parameters

- **j** – current position (integer) in the sequence
- **score_mat** – score matrix
- **beam_size** – specified size of the beam (integer)

validate_forward_backward_pass(*w*, *seq_id*)
check the validity of the forward backward pass

Parameters

- **w** – weight vector (numpy vector)
- **seq_id** – integer representing unique id assigned to the sequence

viterbi(*w*, *seq_id*, *beam_size*, *stop_off_beam=False*, *y_ref=[]*, *K=1*)
decode sequences using viterbi decoder

Parameters

- **w** – weight vector (numpy vector)
- **seq_id** – integer representing unique id assigned to the sequence
- **beam_size** – integer representing the size of the beam

Keyword Arguments

- **stop_off_beam** – boolean indicating if to stop when the reference state falls off the beam (used in perceptron/search based learning)
- **y_ref** – reference sequence list of labels (used while learning)
- **K** – integer indicating number of decoded sequences required (i.e. top-k list)

class pyseqlab.fo_crf.**FirstOrderCRFModelRepresentation**
Bases: *pyseqlab.linear_chain_crf.LCRFModelRepresentation*

Model representation that will hold data structures to be used in *FirstOrderCRF* class

it includes all attributes in the LCRFModelRepresentation parent class

Y_codebook_rev
reversed codebook (dictionary) of Y_codebook

startstate_flag
boolean indicating if to use an edge/boundary state (i.e. __START__ state)

generate_instance_properties()
generate instance properties that will be later used by *FirstOrderCRF* class

get_Y_codebook_reversed()
generate reversed codebook of Y_codebook

get_modelstates_codebook(*states*)
create states codebook by mapping each state to a unique code/number

Parameters **states** – set of tags identified in training sequences

Example:

```
states = {'B-PP', 'B-NP', ...}
```

setup_model(*modelfeatures*, *states*, *L*)
setup and create the model representation

Creates all maps and codebooks needed by the *FirstOrderCRF* class

Parameters

- **modelfeatures** – set of features defining the model
- **states** – set of states (i.e. tags)
- **L** – length of longest segment

pyseqlab.ho_crf module

@author: ahmed allam <ahmed.allam@yale.edu>

class pyseqlab.ho_crf.**HOCRF**(*model*, *seqs_representer*, *seqs_info*, *load_info_fromdisk=5*)
Bases: *pyseqlab.ho_crf_ad.HOCRFAD*

higher-order CRF model

•currently it supports *only* search-based training methods such as *COLLINS-PERCEPTRON* or *SAPO*

•it implements the model discussed in: https://papers.nips.cc/paper/3815-conditional-random-fields-with-high-order-features-for-sequence-labeling.pdf

compute_backward_vec(*w*, *seq_id*)

compute_bpotential(*w*, *active_features*)

compute_seq_gradient(*w*, *seq_id*, *grad*)
sequence gradient computation

> **Warning:** the *HOCRF* currently **does not support** gradient based training. Use search based training methods such as *COLLINS-PERCEPTRON* or *SAPO*
>
> this class is used for demonstration of the computation of the backward matrix using suffix relation as outlined in: https://papers.nips.cc/paper/3815-conditional-random-fields-with-high-order-features-for-sequence-labeling.pdf

validate_forward_backward_pass(*w*, *seq_id*)

class pyseqlab.ho_crf.**HOCRFModelRepresentation**
Bases: *pyseqlab.ho_crf_ad.HOCRFADModelRepresentation*

Model representation that will hold data structures to be used in *HOCRF* class

generate_instance_properties()
generate instance properties that will be later used by HOSemiCRFAD class

get_S_info()

get_backward_states()

get_backward_transitions()

get_si_ysk_map()

get_ysk_codebook()

map_z_ysk()

setup_model(*modelfeatures*, *states*, *L*)
setup and create the model representation

Creates all maps and codebooks needed by the HOSemiCRFAD class

Parameters

- **modelfeatures** – set of features defining the model
- **states** – set of states (i.e. tags)
- **L** – length of longest segment

pyseqlab.ho_crf_ad module

@author: ahmed allam <ahmed.allam@yale.edu>

class pyseqlab.ho_crf_ad.**HOCRFAD**(*model*, *seqs_representer*, *seqs_info*, *load_info_fromdisk=5*)
Bases: *pyseqlab.hosemi_crf_ad.HOSemiCRFAD*

higher-order CRF model that uses algorithmic differentiation in gradient computation

Parameters

- **model** – an instance of *HOCRFADModelRepresentation* class
- **seqs_representer** – an instance of SeqsRepresenter class
- **seqs_info** – dictionary holding sequences info

Keyword Arguments **load_info_fromdisk** – integer from 0 to 5 specifying number of cached data to be kept in memory. 0 means keep everything while 5 means load everything from disk

model
an instance of *HOCRFADModelRepresentation* class

weights
a numpy vector representing feature weights

seqs_representer
an instance of pyseqlab.feature_extraction.SeqsRepresenter class

seqs_info
dictionary holding sequences info

beam_size
determines the size of the beam for state pruning

fun_dict
a function map

def_cached_entities
a list of the names of cached entities sorted (descending) based on estimated space required in memory

compute_backward_vec(*w*, *seq_id*)
compute the backward matrix (beta matrix)

Parameters

- **w** – weight vector (numpy vector)
- **seq_id** – integer representing unique id assigned to the sequence

---

Note: fpotential per boundary dictionary should be available in `seqs.info`

---

compute_feature_expectation(*seq_id*, *P_marginals*, *grad*)
compute the features expectations (i.e. expected count of the feature based on learned model)

Parameters

- **seq_id** – integer representing unique id assigned to the sequence
- **P_marginals** – probability matrix for y patterns at each position in time
- **grad** – numpy vector with dimension equal to the weight vector. It represents the gradient that will be computed using the feature expectation and the global features of the sequence

---

Note:

•activefeatures (per boundary) dictionary should be available in `seqs.info`

•P_marginal (marginal probability matrix) should be available in `seqs.info`

---

compute_forward_vec(*w*, *seq_id*)
compute the forward matrix (alpha matrix)

Parameters

- **w** – weight vector (numpy vector)
- **seq_id** – integer representing unique id assigned to the sequence

---

Note: activefeatures need to be loaded first in `seqs.info`

---

compute_fpotential(*w*, *active_features*)
compute the potential of active features in a specified boundary

Parameters

- **w** – weight vector (numpy vector)
- **active_features** – dictionary of activated features in a specified boundary

compute_marginals(*seq_id*)
compute the marginal (i.e. probability of each y pattern at each position)

Parameters **seq_id** – integer representing unique id assigned to the sequence

---

Note:

•fpotential per boundary dictionary should be available in `seqs.info`

•alpha matrix should be available in `seqs.info`

•beta matrix should be available in `seqs.info`

•Z (i.e. P(x)) should be available in `seqs.info`

---

prune_states (*j*, *delta*, *beam_size*)
prune states that fall off the specified beam size

Parameters

- **j** – current position (integer) in the sequence
- **delta** – score matrix
- **beam_size** – specified size of the beam (integer)

viterbi (*w*, *seq_id*, *beam_size*, *stop_off_beam=False*, *y_ref=[]*, *K=1*)
decode sequences using viterbi decoder

Parameters

- **w** – weight vector (numpy vector)
- **seq_id** – integer representing unique id assigned to the sequence
- **beam_size** – integer representing the size of the beam

Keyword Arguments

- **stop_off_beam** – boolean indicating if to stop when the reference state falls off the beam (used in perceptron/search based learning)
- **y_ref** – reference sequence list of labels (used while learning)
- **K** – integer indicating number of decoded sequences required (i.e. top-k list)

class pyseqlab.ho_crf_ad.**HOCRFADModelRepresentation**
Bases: *pyseqlab.hosemi_crf_ad.HOSemiCRFADModelRepresentation*

Model representation that will hold data structures to be used in *HOCRFAD* class

it includes all attributes in the `HOSemiCRFADModelRepresentation` parent class

filter_activated_states (*activated_states*, *accum_active_states*, *boundary*)
filter/prune states and y features

Parameters

- **activaed_states** – dictionary containing possible active states/y features it has the form {patt_len:{patt_1, patt_2, ...}}
- **accum_active_states** – dictionary of only possible active states by position it has the form {pos_1:{state_1, state_2, ...}}
- **boundary** – tuple (u,v) representing the current boundary in the sequence

pyseqlab.hosemi_crf module

@author: ahmed allam <ahmed.allam@yale.edu>

class pyseqlab.hosemi_crf.**HOSemiCRF** (*model*, *seqs_representer*, *seqs_info*, *load_info_fromdisk=5*)
Bases: *pyseqlab.hosemi_crf_ad.HOSemiCRFAD*

higher-order semi-CRF model

it implements the model discussed in: http://www.jmlr.org/papers/volume15/cuong14a/cuong14a.pdf

compute_backward_vec(*w*, *seq_id*)

compute_bpotential(*w*, *active_features*)

compute_marginals(*seq_id*)

class pyseqlab.hosemi_crf.**HOSemiCRFModelRepresentation**

Bases: *pyseqlab.hosemi_crf_ad.HOSemiCRFADModelRepresentation*

Model representation that will hold data structures to be used in *HOSemiCRF* class

generate_instance_properties()

get_S_info()

get_backward_transitions()

get_si_siy_codebook()

get_siy_info()

map_pky_z()

generate a map between elements of the Z set and PY set

map_siy_z()

setup_model(*modelfeatures*, *states*, *L*)

pyseqlab.hosemi_crf_ad module

@author: ahmed allam <ahmed.allam@yale.edu>

class pyseqlab.hosemi_crf_ad.**HOSemiCRFAD**(*model*, *seqs_representer*, *seqs_info*, *load_info_fromdisk=5*)

Bases: *pyseqlab.linear_chain_crf.LCRF*

higher-order semi-CRF model that uses algorithmic differentiation in gradient computation

Parameters

- **model** – an instance of *HOSemiCRFADModelRepresentation* class
- **seqs_representer** – an instance of SeqsRepresenter class
- **seqs_info** – dictionary holding sequences info

Keyword Arguments **load_info_fromdisk** – integer from 0 to 5 specifying number of cached data to be kept in memory. 0 means keep everything while 5 means load everything from disk

model

an instance of *HOSemiCRFADModelRepresentation* class

weights

a numpy vector representing feature weights

seqs_representer

an instance of pyseqlab.feature_extraction.SeqsRepresenter class

seqs_info

dictionary holding sequences info

beam_size

determines the size of the beam for state pruning

fun_dict
a function map

def_cached_entities
a list of the names of cached entities sorted (descending) based on estimated space required in memory

cached_entitites(*load_info_fromdisk*)
construct list of names of cached entities in memory

compute_backward_vec(*w*, *seq_id*)
compute the backward matrix (beta matrix)

Parameters

- **w** – weight vector (numpy vector)
- **seq_id** – integer representing unique id assigned to the sequence

---

Note: fpotential per boundary dictionary should be available in `seqs.info`

---

compute_feature_expectation(*seq_id*, *P_marginals*, *grad*)
compute the features expectations (i.e. expected count of the feature based on learned model)

Parameters

- **seq_id** – integer representing unique id assigned to the sequence
- **P_marginals** – probability matrix for y patterns at each position in time
- **grad** – numpy vector with dimension equal to the weight vector. It represents the gradient that will be computed using the feature expectation and the global features of the sequence

---

Note:

•activefeatures (per boundary) dictionary should be available in `seqs.info`

•P_marginal (marginal probability matrix) should be available in `seqs.info`

---

compute_forward_vec(*w*, *seq_id*)
compute the forward matrix (alpha matrix)

Parameters

- **w** – weight vector (numpy vector)
- **seq_id** – integer representing unique id assigned to the sequence

---

Note: activefeatures need to be loaded first in `seqs.info`

---

compute_fpotential(*w*, *active_features*)
compute the potential of active features in a specified boundary

Parameters

- **w** – weight vector (numpy vector)
- **active_features** – dictionary of activated features in a specified boundary

compute_marginals(*seq_id*)
compute the marginal (i.e. probability of each y pattern at each position)

Parameters **seq_id** – integer representing unique id assigned to the sequence

---

Note:

•fpotential per boundary dictionary should be available in `seqs.info`

•alpha matrix should be available in `seqs.info`

•beta matrix should be available in `seqs.info`

•Z (i.e. P(x)) should be available in `seqs.info`

---

prune_states (*score_vec*, *beam_size*)
prune states that fall off the specified beam size

Parameters

- **score_vec** – score matrix
- **beam_size** – specified size of the beam (integer)

viterbi (*w*, *seq_id*, *beam_size*, *stop_off_beam=False*, *y_ref=[]*, *K=1*)
decode sequences using viterbi decoder

Parameters

- **w** – weight vector (numpy vector)
- **seq_id** – integer representing unique id assigned to the sequence
- **beam_size** – integer representing the size of the beam

Keyword Arguments

- **stop_off_beam** – boolean indicating if to stop when the reference state falls off the beam (used in perceptron/search based learning)
- **y_ref** – reference sequence list of labels (used while learning)
- **K** – integer indicating number of decoded sequences required (i.e. top-k list) A* searcher with viterbi will be used to generate k-decoded list

class pyseqlab.hosemi_crf_ad.**HOSemiCRFADModelRepresentation**
Bases: *pyseqlab.linear_chain_crf.LCRFModelRepresentation*

Model representation that will hold data structures to be used in `HOSemiCRF` class

P_codebook
set of proper prefixes of the elements in the set of patterns `Z_codebook` e.g. {'':0, 'P':1, 'L':2, 'O':3, 'L|O':4, ...}

P_codebook_rev
reversed codebook of *P_codebook* e.g. {0:'', 1:'P', 2:'L', 3:'O', 4:'L|O', ...}

P_len
dictionary comprising the length (i.e. number of elements) of elements in *P_codebook* e.g. {'':0, 'P':1, 'L':1, 'O':1, 'L|O':2, ...}

P_elems
dictionary comprising the composing elements of every prefix in *P_codebook* e.g. {'':('',), 'P':('P',), 'L':('L',), 'O':('O',), 'L|O':('L','O'), ...}

P_numchar
dictionary comprising the number of characters for every prefix in *P_codebook* e.g. {'':0, 'P':1, 'L':1, 'O':1, 'L|O':3, ...}

f_transition
a dictionary representing forward transition data structure having the form: {pi:{pky, (pk, y)}} where pi represents the longest prefix element in *P_codebook* for pky (representing the concatenation of elements in *P_codebook* and Y_codebook)

pky_codebook
generate a codebook for the elements of the set PY (the product of set P and Y)

pi_pky_map
a map between P elements and PY elements

z_pky_map
a map between elements of the Z set and PY set it has the form/template {ypattern:[pky_elements]}

z_pi_piy_map
a map between elements of the Z set and PY set it has the form/template {ypattern:(pk, pky, pi)}

filter_activated_states(*activated_states*, *accum_active_states*, *curr_boundary*)
filter/prune states and y features

Parameters

- **activaed_states** – dictionary containing possible active states/y features it has the form {patt_len:{patt_1, patt_2, ...}}
- **accum_active_states** – dictionary of only possible active states by position it has the form {pos_1:{state_1, state_2, ...}}
- **boundary** – tuple (u,v) representing the current boundary in the sequence

generate_instance_properties()
generate instance properties that will be later used by *HOSemiCRFAD* class

get_P_codebook_rev()
generate reversed codebook of *P_codebook*

get_P_info()
get the properties of P set (proper prefixes)

get_forward_states()
create set of forward states (referred to set P) and map each element to unique code

P is set of proper prefixes of the elements in Z_codebook set

get_forward_transition()
generate forward transition data structure

Main tasks:

- create a set PY from the product of P and Y sets
- for each element in PY, determine the longest suffix existing in set P
- include all this info in *f_transition* dictionary

get_pi_pky_map()
generate map between P elements and PY elements

Main tasks:

- for every element in PY, determine the longest suffix in P

- determine the two components in PY (i.e. p and y element)
- represent this info in a dictionary that will be used for forward/alpha matrix

get_pky_codebook()
generate a codebook for the elements of the set PY (the product of set P and Y)

map_pky_z()
generate a map between elements of the Z set and PY set

setup_model(*modelfeatures*, *states*, *L*)
setup and create the model representation

Creates all maps and codebooks needed by the *HOSemiCRFAD* class

Parameters

- **modelfeatures** – set of features defining the model
- **states** – set of states (i.e. tags)
- **L** – length of longest segment

pyseqlab.linear_chain_crf module

@author: ahmed allam <ahmed.allam@yale.edu>

class pyseqlab.linear_chain_crf.**LCRF**(*model*, *seqs_representer*, *seqs_info*, *load_info_fromdisk=5*)

Bases: object

linear chain CRF model

Parameters

- **model** – an instance of *LCRFModelRepresentation* class
- **seqs_representer** – an instance of SeqsRepresenter class
- **seqs_info** – dictionary holding sequences info

Keyword Arguments **load_info_fromdisk** – integer from 0 to 5 specifying number of cached data to be kept in memory. 0 means keep everything while 5 means load everything from disk

model
an instance of *LCRFModelRepresentation* class

weights
a numpy vector representing feature weights

seqs_representer
an instance of SeqsRepresenter class

seqs_info
dictionary holding sequences info

beam_size
determines the size of the beam for state pruning

fun_dict
a function map

def_cached_entities
a list of the names of cached entities sorted (descending) based on estimated space required in memory

cached_entitites(*load_info_fromdisk*)
construct list of names of cached entities in memory

check_cached_info(*seq_id*, *entity_names*)
check and load required data elements/entities for every computation step

Parameters

- **seq_id** – integer representing unique id assigned to the sequence
- **entity_name** – list of names of the data elements need to be loaded in `seqs.info` dictionary needed while performing computation

---

Note: order of elements in the entity_names list is **important**

---

check_gradient(*w*, *seq_id*)
implementation of finite difference method similar to `scipy.optimize.check_grad()`

Parameters

- **w** – weight vector (numpy vector)
- **seq_id** – integer representing unique id assigned to the sequence

clear_cached_info(*seqs_id*, *cached_entities=[]*)
clear/clean loaded data elements/entities in `seqs.info` dictionary

Parameters **seqs_id** – list of integers representing the unique ids of the training sequences

Keyword Arguments **cached_entities** – list of data entities to be cleared for the `seqs.info` dictionary

---

Note: order of elements in the entity_names list is **important**

---

compute_backward_vec(*w*, *seq_id*)
compute the backward matrix (beta matrix)

Parameters

- **w** – weight vector (numpy vector)
- **seq_id** – integer representing unique id assigned to the sequence

Warning: implementation of this method is in the child class

compute_feature_expectation(*seq_id*, *P_marginals*)
compute the features expectations (i.e. expected count of the feature based on learned model)

Parameters

- **seq_id** – integer representing unique id assigned to the sequence
- **P_marginals** – probability matrix for y patterns at each position in time

Warning: implementation of this method is in the child class

compute_forward_vec(*w*, *seq_id*)
compute the forward matrix (alpha matrix)

Parameters

- **w** – weight vector (numpy vector)
- **seq_id** – integer representing unique id assigned to the sequence

Warning: implementation of this method is in the child class

compute_marginals(*seq_id*)
compute the marginal (i.e. probability of each y pattern at each position)

Parameters **seq_id** – integer representing unique id assigned to the sequence

Warning: implementation of this method is in the child class

compute_seq_gradient(*w*, *seq_id*, *grad*)
compute the gradient of conditional log-likelihood with respect to the parameters vector w ($\frac{\partial p(Y|X;w)}{\partial w}$)

Parameters

- **w** – weight vector (numpy vector)
- **seq_id** – integer representing unique id assigned to the sequence

compute_seq_loglikelihood(*w*, *seq_id*)
computes the conditional log-likelihood of a sequence (i.e. $p(Y|X;w)$)

it is used as a cost function for the single sequence when trying to estimate parameters w

Parameters

- **w** – weight vector (numpy vector)
- **seq_id** – integer representing unique id assigned to the sequence

compute_seqs_gradient(*w*, *seqs_id*)
compute the gradient of conditional log-likelihood with respect to the parameters vector w

Parameters

- **w** – weight vector (numpy vector)
- **seqs_id** – list of integer representing unique ids of sequences used for training

compute_seqs_loglikelihood(*w*, *seqs_id*)
computes the conditional log-likelihood of training sequences

it is used as a cost/objective function for the whole training sequences when trying to estimate parameters w

Parameters

- **w** – weight vector (numpy vector)
- **seqs_id** – list of integer representing unique ids of sequences used for training

decode_seqs(*decoding_method*, *out_dir*, ***kwargs*)
decode sequences (i.e. infer labels of sequence of observations)

Parameters

- **decoding_method** – a string referring to type of decoding {viterbi, per_state_decoding}
- **out_dir** – string representing the working directory (path) where sequence processing will take place

Keyword Arguments

- **file_name** – the name of the file in case decoded sequences are required to be written
- **sep** – separator (default 't') between the columns when writing decoded sequences to file
- **procseqs_foldername** – string representing the folder name where intermediary data and parsing would take place
- **beam_size** – integer determining the size of the beam while decoding
- **seqs** – a list comprising of sequences that are instances of `SequenceStruct` class to be decoded (used for decoding test data or any new/unseen data – sequences)
- **seqs_info** – dictionary containing the info about the sequences to decode (used for decoding training sequences)
- **seqs_dict** – a dictionary comprising of sequence ids as keys and corresponding sequences that are instances of `SequenceStruct` class to be decoded as values

---

Note: for keyword arguments only one of {`seqs` , `seqs_info`, `seqs_dict`} option need to be specified

---

generate_activefeatures(*seq_id*)
construct a dictionary of model active features identified given a sequence

Main task:

- generate active features for every boundary of the sequence

Parameters **seq_id** – integer representing unique id assigned to the sequence

identify_activefeatures(*seq_id*, *boundary*, *accum_activestates*, *apply_filter=True*)
determine model active features for a given sequence at defined boundary

Main task:

- determine model active features in a given boundary
- update the accum_activestates dictionary

Parameters

- **seq_id** – integer representing unique id assigned to the sequence
- **boundary** – tuple (u,v) defining the boundary under consideration
- **accum_activestates** – dictionary of the form {(u,v):{state_1, state_2, ...}} it keeps track of the active states in each boundary

load_activatedstates(*seq_id*)
load sequence activated states in *seqs_info*

Parameters **seq_id** – integer representing unique id assigned to the sequence

load_activefeatures(*seq_id*)
load sequence model identified active features in *seqs_info*

Parameters **seq_id** – integer representing unique id assigned to the sequence

load_globalfeatures(*seq_id*, *per_boundary=True*)
load sequence global features in *seqs_info*

Parameters **seq_id** – integer representing unique id assigned to the sequence

Keyword Arguments **per_boundary** – boolean representing if the required global features dictionary is represented by boundary (i.e. True) or aggregated (i.e. False)

load_imposter_globalfeatures(*seq_id*, *y_imposter*, *seg_other_symbol*)
load imposter sequence global features in *seqs_info*

Parameters

- **seq_id** – integer representing unique id assigned to the sequence
- **y_imposter** – the imposter sequence generated using viterbi decoder
- **seg_other_sybmol** – If it is specified, then the task is a segmentation problem (in this case we need to specify the non-entity/other element) else if it is None (default), then it is considered as sequence labeling problem

load_segfeatures(*seq_id*)
load sequence observation features in *seqs_info*

Parameters **seq_id** – integer representing unique id assigned to the sequence

prune_states(*j*, *delta*, *beam_size*)
prune states that fall off the specified beam size

Parameters

- **j** – current position (integer) in the sequence
- **delta** – score matrix
- **beam_size** – specified size of the beam (integer)

> **Warning:** implementation of this method is in the child class

represent_globalfeature(*gfeatures*, *boundaries*)
represent extracted sequence global features

two representation could be applied:

- 1. features identified by boundary (i.e. f(X,Y))
- 2. features identified and aggregated across all positions in the sequence (i.e. F(X, Y))

Parameters

- **gfeatures** – dictionary representing the extracted sequence features (i.e F(X, Y))
- **boundaries** – if specified (i.e. list of boundaries), then the required representation is global features per boundary (i.e. option (1)) else (i.e. None or empty list), then the required representation is the aggregated global features (option(2))

save_model(*folder_dir*)
save model data structures

Parameters **folder_dir** – string representing directory where files are pickled/dumped

seqs_info

validate_expected_featuresum(*w*, *seqs_id*)
validate expected feature computation

Parameters

- **w** – weight vector (numpy vector)
- **seqs_id** – list of integers representing unique id assigned to the sequences

validate_forward_backward_pass(*w*, *seq_id*)
check the validity of the forward backward pass

Parameters

- **w** – weight vector (numpy vector)
- **seq_id** – integer representing unique id assigned to the sequence

validate_gradient(*w*, *seq_id*)

viterbi(*w*, *seq_id*, *beam_size*, *stop_off_beam=False*, *y_ref=[]*, *K=1*)
decode sequences using viterbi decoder

Parameters

- **w** – weight vector (numpy vector)
- **seq_id** – integer representing unique id assigned to the sequence
- **beam_size** – integer representing the size of the beam

Keyword Arguments

- **stop_off_beam** – boolean indicating if to stop when the reference state falls off the beam (used in perceptron/search based learning)
- **y_ref** – reference sequence list of labels (used while learning)
- **K** – integer indicating number of decoded sequences required (i.e. top-k list)

Warning: implementation of this method is in the child class

write_decoded_seqs(*ref_seqs*, *Y_pred_seqs*, *out_file*, *sep='\t'*)
write inferred sequences on file

Parameters

- **ref_seqs** – list of sequences that are instances of `SequenceStruct`
- **Y_pred_seqs** – list of list of tags decoded for every reference sequence
- **out_file** – string representing out file where data is written
- **sep** – separator used while writing on out file

class pyseqlab.linear_chain_crf.**LCRFModelRepresentation**
Bases: object

Model representation that will hold data structures to be used in *LCRF* class

modelfeatures
set of features defining the model

modelfeatures_codebook
dictionary mapping each features in *modelfeatures* to a unique code

Y_codebook
dictionary mapping the set of states (i.e. tags) to a unique code each

L
length of longest segment

Z_codebook
dictionary for the set Z, mapping each element to unique number/code

Z_len
dictionary comprising the length of each element in *Z_codebook*

Z_elems
dictionary comprising the composing elements of each member in the Z set (*Z_codebook*)

Z_numchar
dictionary comprising the number of characters of each member in the Z set (*Z_codebook*)

patts_len
set of lengths extracted from *Z_len* (i.e. set(Z_len.values()))

max_patts_len
maximum pattern length used in the model

modelfeatures_inverted
inverted model features (i.e inverting the *modelfeatures* dictionary)

ypatt_features
state features (i.e. y pattern features) that depend only on the states

ypatt_activestates
possible/potential activated y patterns/features using the observation features

num_features
total number of features in the model

num_states
total number of states in the model

accumulate_activefeatures(*activefeatures*, *accumfeatures*)

check_prefix(*token*, *ref_str*)

check_suffix(*token*, *ref_str*)

filter_activated_states(*activated_states*, *accum_active_states*, *boundary*)
filter/prune states and y features

Parameters

- **activaed_states** – dictionary containing possible active states/y features it has the form {patt_len:{patt_1, patt_2, ...}}
- **accum_active_states** – dictionary of only possible active states by position it has the form {pos_1:{state_1, state_2, ...}}
- **boundary** – tuple (u,v) representing the current boundary in the sequence

find_activated_states(*seg_features*, *allowed_z_len*)
identify possible activated y patterns/features using the observation features

Parameters

- **seg_features** – dictionary of the observation features. It has the form {featureA_name:value, featureB_name:value, ...}
- **allowed_z_len** – set of permissible order/length of y features {1,2,3} -> means up to third order y features are allowed

find_seg_activefeatures(*seg_features*, *allowed_z_len*)
finds active features based on the observation/segment features

Parameters

- **seg_features** –
- **allowed_z_len** –

find_ypatt_activefeatures(*allowed_z_len*)
finds the label and state transition features (if applicable – in case it is modeled)

Parameters **allowed_z_len** –

generate_instance_properties()
generate instance properties that will be later used by *LCRF* class

get_Z_info()
get the properties of Z set

get_Z_pattern()
create a codebook from set Z by mapping each element to unique number/code

Z is set of y patterns used in the model features

Example:

```
Z = {'O|B-VP|B-NP', 'O|B-VP', 'O', 'B-VP', 'B-NP', ...}
Z_codebook = {'O|B-VP|B-NP':1, 'O|B-VP':2, 'O':3, 'B-VP':5, 'B-NP':4, ...}
```

get_inverted_modelfeatures()
invert *modelfeatures* instance variable

Example:

```
modelfeatures_inverted =
{'w[0]=take': {1: {'I-VP'}, 2: {'I-VP|I-VP'}, 3: {'I-VP|I-VP|I-VP'}},
 'w[0]=the': {1: {'B-NP'},
              2: {'B-PP|B-NP', 'I-VP|B-NP'},
              3: {'B-NP|B-PP|B-NP', B-VP|I-VP|B-NP', ...}
             },
             ...
}

ypatt_features = {'B-NP', 'B-PP|B-NP', ..}
```

get_modelfeatures_codebook()
setup model features codebook

it flatten *modelfeatures* and map each element to a unique code *modelfeatures* are represented in a dictionary with this form:

```
{y_patt_1:{featureA:value, featureB:value, ...}
 y_patt_2:{featureA:value, featureC:value, ...}}
```

Example:

```
modelfeatures:
     {'B-PP': Counter({'w[0]=at': 1,
                       'w[0]=by': 1,
                       'w[0]=for': 4,
                       ...
                      }),
     'B-PP|B-NP': Counter({'w[0]=16': 1,
                           'w[0]=July': 1,
                           'w[0]=Nomura': 1,
                           ...
                           }),
                  ...
     }
modelfeatures_codebook:
    {('B-PP','w[0]=at'): 1,
     ('B-PP','w[0]=by'): 2,
     ('B-PP','w[0]=for'): 3,
     ...
    }
```

get_modelstates_codebook(*states*)
create states codebook by mapping each state to a unique code/number

Parameters **states** – set of tags identified in training sequences

Example:

```
states = {'B-PP', 'B-NP', ...}
states_codebook = {'B-PP':1, 'B-NP':2 ...}
```

get_num_features()
return total number of features in the model

get_num_states()
return total number of states identified by the model in the training set

join_segfeatures_filteredstates(*seg_features*, *filtered_states*)
represent detected active features while parsing sequences

Parameters

- **activestates** – dictionary of the form {'patt_len':{patt_1, patt_2, ...}}
- **seg_features** – dictionary of the observation features. It has the form {featureA_name:value, featureB_name:value, ...}

keep_longest_elems(*s*)
used to figure out longest suffix and prefix on sets

represent_activefeatures(*activefeatures*)

represent_globalfeatures(*seq_featuresum*)
represent features extracted from sequences using *modelfeatures_codebook*

Parameters **seq_featuresum** – dictionary of sequence global features representing F(X,Y)

represent_ypatt_filteredstates(*filtered_states*)
represent detected active features while parsing sequences

Parameters

- **activestates** – dictionary of the form {'patt_len':{patt_1, patt_2, ...}}

- **seg_features** – dictionary of the observation features. It has the form {featureA_name:value, featureB_name:value, ...}

save(*folder_dir*)
save main model data structures

setup_model(*modelfeatures*, *states*, *L*)
setup and create the model representation

Creates all maps and codebooks needed by the *LCRF* class

Parameters

- **modelfeatures** – set of features defining the model
- **states** – set of states (i.e. tags)
- **L** – length of longest segment

pyseqlab.utilities module

@author: ahmed allam <ahmed.allam@yale.edu>

class pyseqlab.utilities.**AStarAgenda**
Bases: object

class containing a heap where instances of *AStarNode* class will be pushed

the push operation will use the score matrix (built using viterbi algorithm) representing the unnormalized probability of the sequences ending at every position with the different available prefixes/states

qagenda
queue where instances of *AStarNode* are pushed

entry_count
counter that keeps track of the entries and associate each entry(node) with a unique number. It is useful for resolving nodes with equal costs

pop()
pop nodes with highest score from the heap

push(*astar_node*, *cost*)
push instance of *AStarNode* with its associated cost to the heap

Parameters

- **astar_node** – instance of *AStarNode* class
- **cost** – float representing the score/unnormalized probability of a sequence up to given position

class pyseqlab.utilities.**AStarNode**(*cost*, *position*, *pi_c*, *label*, *frwdlink*)
Bases: object

class representing A* node to be used with A* searcher and viterbi for generating k-decoded list

Parameters

- **cost** – float representing the score/unnormalized probability of a sequence up to given position
- **position** – integer representing the current position in the sequence
- **pi_c** – prefix or state code of the label

- **label** – label of the current position in a sequence
- **frwdlink** – a link to *AStarNode* node

cost
float representing the score/unnormalized probability of a sequence up to given position

position
integer representing the current position in the sequence

pi_c
prefix or state code of the label

label
label of the current position in a sequence

frwdlink
a link to *AStarNode* node

print_node()
print the info about a node

class pyseqlab.utilities.**BoundNode**(*parent*, *boundary*)
Bases: object

boundary entity class used when generating all possible partitions within specified constraint

Parameters

- **parent** – instance of *BoundNode*
- **boundary** – tuple (u,v) representing the current boundary

add_child(*child*)
add link to the child nodes

get_child()
retrieve child nodes

get_signature()
retrieve the id of the node

class pyseqlab.utilities.**DataFileParser**
Bases: object

class to parse a data file comprising the training/testing data

seqs
list comprising of sequences that are instances of *SequenceStruct* class

header
list of attribute names read from the file

parse_header(*x_arg*)
parse header

Parameters **x_arg** – tuple of attribute/observation names

parse_line(*x_arg*)
parse the read line

Parameters **x_arg** – tuple of observation columns

read_file(*file_path*, *header*, *y_ref=True*, *seg_other_symbol=None*, *column_sep=' '*)
read and parse a file the contains the sequences following a predefined format

the file should contain label and observation tracks each separated in a column

Note: label column is the **LAST** column in the file (i.e. X_a X_b Y)

Parameters

- **file_path** – string representing the file path to the data file
- **header** – specifies how the header is reported in the file containing the sequences options include: - 'main' -> one header in the beginning of the file - 'per_sequence' -> a header for every sequence - list of keywords as header (i.e. ['w', 'part_of_speech'])

Keyword Arguments

- **y_ref** – boolean specifying if the reference label column in the data file
- **seg_other_sybmol** – string or None(default), if specified then the task is a segmentation problem where *seg_other_symbol* represents the non-entity symbol. In this case semi-CRF models are used. Else (i.e. *seg_other_symbol* is not None) then it is considered as sequence labeling problem.
- **column_sep** – string, separator used between the columns in the file

update_X(*X*, *Y*)
update sequence observations

update_XY(*X*, *Y*)
update sequence observations and corresponding labels

class pyseqlab.utilities.**FO_AStarSearcher**(*Y_codebook_rev*)
Bases: object

A* star searcher associated with first-order CRF model such as FirstOrderCRF

Parameters **Y_codebook_rev** – a reversed version of dictionary comprising the set of states each assigned a unique code

Y_codebook_rev
a reversed version of dictionary comprising the set of states each assigned a unique code

infer_labels(*top_node*, *back_track*)
decode sequence by inferring labels

Parameters

- **top_node** – instance of *AStarNode* class
- **back_track** – dictionary containing back pointers built using dynamic programming algorithm

search(*alpha*, *back_track*, *T*, *K*)
A* star searcher uses the score matrix (built using viterbi algorithm) to decode top-K list of sequences

Parameters

- **alpha** – score matrix build using the viterbi algorithm
- **back_track** – back_pointers dictionary tracking the best paths to every state
- **T** – last decoded position of a sequence (in this context, it is the alpha.shape[0])
- **K** – number of top decoded sequences to be returned

Returns top-K list of decoded sequences

Return type topk_list

class pyseqlab.utilities.**HOSemi_AStarSearcher**(*P_codebook_rev*, *pi_elems*)
Bases: object

A* star searcher associated with higher-order CRF model such as HOSemiCRFAD

Parameters

- **P_codebook_rev** – reversed codebook of set of proper prefixes in the *P* set e.g. {0:'', 1:'P', 2:'L', 3:'O', 4:'L|O', ...}
- **P_elems** – dictionary comprising the composing elements of every prefix in the *P* set e.g. {'':('',), 'P':('P',), 'L':('L',), 'O':('O',), 'L|O':('L','O'), ...}

P_codebook_rev
reversed codebook of set of proper prefixes in the *P* set e.g. {0:'', 1:'P', 2:'L', 3:'O', 4:'L|O', ...}

P_elems
dictionary comprising the composing elements of every prefix in the *P* set e.g. {'':('',), 'P':('P',), 'L':('L',), 'O':('O',), 'L|O':('L','O'), ...}

get_node_label(*pi_code*)
get the the label/state given a prefix code

Parameters **pi_code** – prefix code which is an element of *P_codebook_rev*

infer_labels(*top_node*, *back_track*)
decode sequence by inferring labels

Parameters

- **top_node** – instance of *AStarNode* class
- **back_track** – dictionary containing back pointers tracking the best paths to every state

search(*alpha*, *back_track*, *T*, *K*)
A* star searcher uses the score matrix (built using viterbi algorithm) to decode top-K list of sequences

Parameters

- **alpha** – score matrix build using the viterbi algorithm
- **back_track** – back_pointers dictionary tracking the best paths to every state
- **T** – last decoded position of a sequence (in this context, it is the alpha.shape[0])
- **K** – number of top decoded sequences to be returned

Returns top-K list of decoded sequences

Return type topk_list

class pyseqlab.utilities.**HO_AStarSearcher**(*P_codebook_rev*, *P_elems*)
Bases: object

A* star searcher associated with higher-order CRF model such as HOCRFAD

Parameters

- **P_codebook_rev** – reversed codebook of set of proper prefixes in the *P* set e.g. {0:'', 1:'P', 2:'L', 3:'O', 4:'L|O', ...}

- **P_elems** – dictionary comprising the composing elements of every prefix in the *P* set e.g. `{'':('',), 'P':('P',), 'L':('L',), 'O':('O',), 'L|O':('L','O'), ...}`

P_codebook_rev
reversed codebook of set of proper prefixes in the *P* set e.g. `{0:'', 1:'P', 2:'L', 3:'O', 4:'L|O', ...}`

P_elems
dictionary comprising the composing elements of every prefix in the *P* set e.g. `{'':('',), 'P':('P',), 'L':('L',), 'O':('O',), 'L|O':('L','O'), ...}`

get_node_label(*pi_code*)
get the the label/state given a prefix code

Parameters **pi_code** – prefix code which is an element of *P_codebook_rev*

infer_labels(*top_node*, *back_track*)
decode sequence by inferring labels

Parameters

- **top_node** – instance of *AStarNode* class
- **back_track** – dictionary containing back pointers tracking the best paths to every state

search(*alpha*, *back_track*, *T*, *K*)
A* star searcher uses the score matrix (built using viterbi algorithm) to decode top-K list of sequences

Parameters

- **alpha** – score matrix build using the viterbi algorithm
- **back_track** – back_pointers dictionary tracking the best paths to every state
- **T** – last decoded position of a sequence (in this context, it is the alpha.shape[0])
- **K** – number of top decoded sequences to be returned

Returns top-K list of decoded sequences

Return type topk_list

class pyseqlab.utilities.**ReaderWriter**
Bases: object

class for dumping, reading and logging data

static dump_data(*data*, *file_name*, *mode='wb'*)
dump data by pickling

Parameters

- **data** – data to be pickled
- **file_name** – file path where data will be dumped
- **mode** – specify writing options i.e. binary or unicode

static log_progress(*line*, *outfile*, *mode='a'*)
write data to a file

Parameters

- **line** – string representing data to be written out
- **outfile** – file path where data will be written/logged

- **mode** – specify writing options i.e. append, write

static read_data(*file_name*, *mode='rb'*)
read dumped/pickled data

Parameters

- **file_name** – file path where data will be dumped
- **mode** – specify writing options i.e. binary or unicode

class pyseqlab.utilities.**SequenceStruct**(*X*, *Y*, *seg_other_symbol=None*)
Bases: object

class for representing each sequence/segment

Parameters

- **Y** – list containing the sequence of states/labels (i.e. ['P','O','O','L','L'])
- **X** – list containing dictionary elements of observation sequences and/or features of the input
- **seg_other_symbol** – string or None (default), if specified then the task is a segmentation problem where it represents the non-entity symbol else (None) then it is considered as sequence labeling problem

Y
list containing the sequence of states/labels (i.e. ['P','O','O','L','L'])

X
list containing dictionary elements of observation sequences and/or features of the input

seg_other_symbol
string or None(default), if specified then the task is a segmentation problem where it represents the non-entity symbol else (None) then it is considered as sequence labeling problem

T
int, length of a sequence (i.e. len(X))

seg_attr
dictionary comprising the extracted attributes per each boundary of a sequence

L
int, longest length of an identified segment in the sequence

flat_y
list of labels/tags

y_sboundaries
sorted list of boundaries of the *Y* of the sequence

y_range
range of the sequence

X

Y

flatten_y(*Y*)
flatten the *Y* attribute

Parameters **Y** – dictionary of this form `{(1, 1): 'P', (2,2): 'O', (3, 3): 'O', (4, 5): 'L'}`

Example

flattened y becomes `['P','O','O','L','L']`

get_x_boundaries()
return the boundaries of the observation sequence

get_y_boundaries()
return the sorted boundaries of the labels of the sequence

class pyseqlab.utilities.**TemplateGenerator**
Bases: object

template generator class for feature/function template generation

static generate_combinations(*n*)
generates all possible combinations based on the maximum number of ngrams n

Parameters **n** – integer specifying the maximum/greatest ngram option

static generate_ngram(*l*, *n*)
n-gram generator based on the length of the window and the ngram option

Parameters

- **l** – list of positions of the range representing the window size (i.e. list(wsize))
- **n** – integer representing the n-gram option (i.e. 1 for unigram, 2 for bigram, etc..)

generate_template_XY(*attr_name*, *x_spec*, *y_spec*, *template*)
generate template XY for the feature extraction

Parameters

- **attr_name** – string representing the attribute name of the atomic observations/tokens
- **x_spec** – tuple of the form (n-gram, range) that is we can specify the n-gram features required in a specific range/window for an observation token `attr_name`
- **y_spec** – string specifying how to join/combine the features on the X observation level with labels on the Y level.

  Example of passed options would be:

  - one state (i.e. current state) by passing `1-state` or
  - two states (i.e. current and previous state) by passing `2-states` or
  - one and two states (i.e. mix/combine observation features with one state model and two states models) by passing `1-state:2-states`. Higher order models support models with states > 2 such as `3-states` and above.
- **template** – dictionary that accumulates the generated feature template for all attributes

Example

suppose we have *word* attribute referenced by 'w' and we need to use the current word with the current label (i.e. unigram of words with the current label) in a range of (0,1)

```
templateXY = {}
generate_template_XY('w', ('1-gram', range(0, 1)), '1-state', templateXY)
```

we can also specify a two states/labels features at the Y level

```
generate_template_XY('w', ('1-gram', range(0, 1)), '1-state:2-states',␣
↪templateXY)
```

Note: this can be applied for every attribute name and accumulated in the *template* dictionary

generate_template_Y(*ngram_options*)
generate template on the Y labels level

Parameters **ngram_options** – string specifying the number of states to be use (i.e. `1-state`). It also supports multiple specification such as `1-state:2-states` where each is separated by a colon

pyseqlab.utilities.**aggregate_weightedsample**(*w_sample*)
represent the random picked sample for training/testing

Parameters **w_sample** – dictionary representing a random split of the grouped sequences by their length. it is obtained using *weighted_sample()* function

pyseqlab.utilities.**create_directory**(*folder_name*, *directory='current'*)
create directory/folder (if it does not exist) and returns the path of the directory

Parameters **folder_name** – string representing the name of the folder to be created

Keyword Arguments **directory** – string representing the directory where to create the folder if *current* then the folder will be created in the current directory

pyseqlab.utilities.**delete_directory**(*directory*)

pyseqlab.utilities.**delete_file**(*filepath*)

pyseqlab.utilities.**generate_datetime_str**()
generate string composed of the date and time

pyseqlab.utilities.**generate_partition_boundaries**(*depth_node_map*)
generate partitions of the boundaries generated in *generate_partitions()* function

Parameters **depth_node_map** – dictionary that arranges the generated nodes by their depth in the tree it is constructed using *generate_partitions()* function

pyseqlab.utilities.**generate_partitions**(*boundary*, *L*, *patt_len*, *bound_node_map*, *depth_node_map*, *parent_node*, *depth=1*)
generate all possible partitions within the range of segment length and model order

it transforms the partitions into a tree of nodes starting from the root node that uses *boundary* argument in its construction

Parameters

- **boundary** – tuple (u,v) representing the current boundary in a sequence
- **L** – integer representing the maximum length a segment could be constructed
- **patt_len** – integer representing the maximum model order
- **bound_node_map** – dictionary that keeps track of all possible partitions represented as instances of *BoundNode*
- **depth_node_map** – dictionary that arranges the generated nodes by their depth in the tree
- **parent_node** – instance of *BoundNode* or None in case of the root node

- **depth** – integer representing the maximum depth of the tree to be reached before stopping

pyseqlab.utilities.**generate_trained_model**(*modelparts_dir*, *aextractor_obj*)
regenerate trained CRF models using the saved trained model parts/components

Parameters

- **modelparts_dir** – string representing the directory where model parts are saved
- **aextractor_class** – name of the attribute extractor class such as `NERSegmentAttributeExtractor`

pyseqlab.utilities.**generate_updated_model**(*modelparts_dir*, *modelrepr_class*, *model_class*, *aextractor_obj*, *fextractor_class*, *seqrepresenter_class*, *ascaler_class=None*)
update/regenerate CRF models using the saved parts/components

Parameters

- **modelparts_dir** – string representing the directory where model parts are saved
- **modelrepr_class** – name of the model representation class to be used which has suffix *ModelRepresentation* such as `HOCRFADModelRepresentation`
- **model_class** – name of the CRF model class such as `HOCRFAD`
- **aextractor_class** – name of the attribute extractor class such as `NERSegmentAttributeExtractor`
- **fextractor_class** – name of the feature extractor class used such as `HOFeatureExtractor`
- **seqrepresenter_class** – name of the sequence representer class such as `SeqsRepresenter`
- **ascaler_class** – name of the attribute scaler class such as `AttributeScaler`

Note: This function is equivalent to *generate_trained_model()* function. However, this function uses explicit specification of the arguments (i.e. specifying explicitly the classes to be used)

---

pyseqlab.utilities.**get_conll00**()

pyseqlab.utilities.**group_seqs_by_length**(*seqs_info*)
group sequences by their length

Parameters **seqs_info** – dictionary comprsing info about the sequences it has this form {seq_id:{T:length of sequence}}

Note: sequences that are with unique sequence length are grouped together as singeltons

---

pyseqlab.utilities.**nested_cv**(*seqs_id*, *outer_kfold*, *inner_kfold*)
generate nested cross-validation division of sequence ids

pyseqlab.utilities.**split_data**(*seqs_id*, *options*)
utility function for splitting dataset (i.e. training/testing and cross validation)

Parameters

- **seqs_id** – list of processed sequence ids
- **options** – dictionary comprising of the options on how to split data

Example

To perform cross validation, we need to specify

- cross-validation for the *method*
- the number of folds for the *k_fold*

```
options = {'method':'cross_validation',
           'k_fold':number
          }
```

To perform random splitting, we need to specify

- random for the *method*
- number of splits for the *num_splits*
- size of the training set in percentage for the *trainset_size*

```
options = {'method':'random',
           'num_splits':number,
           'trainset_size':percentage
          }
```

pyseqlab.utilities.**vectorized_logsumexp**(*vec*)
vectorized version of log sum exponential operation

Parameters **vec** – numpy vector where entries are in the log domain

pyseqlab.utilities.**weighted_sample**(*grouped_seqs*, *trainset_size*)
get a random split of the grouped sequences

Parameters

- **grouped_seqs** – dictionary of the grouped sequences based on their length it is obtained using *group_seqs_by_length()* function
- **trainset_size** – integer representing the size of the training set in percentage

pyseqlab.workflow module

@author: ahmed allam <ahmed.allam@yale.edu>

class pyseqlab.workflow.**GenericTrainingWorkflow**(*aextractor_obj*, *fextractor_obj*, *feature_filter_obj*, *model_repr_class*, *model_class*, *root_dir*)

Bases: object

generic training workflow for building and training CRF models

Parameters

- **aextractor_obj** – initialized instance of GenericAttributeExtractor class/subclass
- **fextractor_obj** – initialized instance of FeatureExtractor class/subclass
- **feature_filter_obj** – None or an initialized instance of FeatureFilter class

- **model_repr_class** – a CRFs model representation class such as HOCRFADModelRepresentation
- **model_class** – a CRFs model class such as HOCRFAD
- **root_dir** – string representing the directory/path where working directory will be created

aextractor_obj
initialized instance of GenericAttributeExtractor class/subclass

fextractor_obj
initialized instance of FeatureExtractor class/subclass

feature_filter_obj
None or an initialized instance of FeatureFilter class

model_repr_class
a CRFs model representation class such as HOCRFADModelRepresentation

model_class
a CRFs model class such as HOCRFAD

root_dir
string representing the directory/path where working directory will be created

build_crf_model(*seqs_id*, *folder_name*, *load_info_fromdisk=10*, *full_parsing=True*)

build_seqsinfo_from_seqfile(*seq_file*, *data_parser_options*, *num_seqs=inf*)
prepares and process sequences to disk and return info dictionary about the parsed sequences

Parameters

- **seq_file** – string representing the path to the sequence file
- **data_parser_options** – dictionary containing options to be passed to read_file() method of DataFileParser class
- **num_seqs** – integer, maximum number of sequences to read from file (default numpy.inf – means read all file)

build_seqsinfo_from_seqs(*seqs*)
prepares and process sequences to disk and return info dictionary about the parsed sequences

Parameters **seqs** – list of sequences that are instances of SequenceStruct class

get_learned_crf(*savedmodel_dir*)
revive learned/trained model

static get_seqs_from_file(*seq_file*, *data_parser*, *data_parser_options*)
read sequences from a file

Parameters

- **seq_file** – string representing the path to the sequence file
- **data_parser** – instance of DataFileParser class
- **data_parser_options** – dictionary containing options to be passed to read_file() method of DataFileParser class

static map_pred_to_ref_seqs(*seqs_pred*)

seq_parsing_workflow(*split_options*, ***kwargs*)
preparing and parsing sequences to be later used in the learning framework

static split_dataset(*seqs_info*, *split_options*)
splits dataset for learning and testing

train_model(*trainseqs_id*, *crf_model*, *optimization_options*)
train a model and return the directory of the trained model

traineval_folds(*data_split*, ***kwargs*)
train and evaluate model on different dataset splits

use_model(*savedmodel_dir*, *options*)
use trained model for decoding and performance measure evaluation

verify_template()
verifying template – sanity check

class pyseqlab.workflow.**TrainingWorkflow**(*template_y*, *template_xy*, *model_repr_class*, *model_class*, *fextractor_class*, *aextractor_class*, *scaling_method*, *optimization_options*, *root_dir*, *filter_obj=None*)

Bases: object

general training workflow

Note: It is **highly recommended** to start using *GenericTrainingWorkflow* class

Warning: This class will be deprecated ...

eval_model(*savedmodel_info*, *eval_seqs*, *eval_filename*, *dec_seqs_filename*, *sep=' '*)

map_pred_to_ref_seqs(*seqs_pred*)

seq_parsing_workflow(*seqs*, *split_options*)
preparing sequences to be used in the learning framework

split_dataset(*seqs_info*, *split_options*)

train_model(*trainseqs_id*, *crf_model*)
training a model and return the directory of the trained model

traineval_folds(*data_split*, *meval=True*, *sep=' '*)
train and evaluate model on different dataset splits

verify_template()
verifying template – sanity check

class pyseqlab.workflow.**TrainingWorkflowIterative**(*template_y*, *template_xy*, *model_repr_class*, *model_class*, *fextractor_class*, *aextractor_class*, *scaling_method*, *ascaler_class*, *optimization_options*, *root_dir*, *data_parser_options*, *filter_obj=None*)

Bases: object

general training workflow that support reading/preparing **large** training sets

Note: It is **highly recommended** to start using *GenericTrainingWorkflow* class

> **Warning:** This class will be deprecated ...

build_seqsinfo(*seq_file*)

eval_model(*savedmodel_dir*, *options*)

get_learned_crf(*savedmodel_dir*)

get_seqs_from_file(*seq_file*)

map_pred_to_ref_seqs(*seqs_pred*)

seq_parsing_workflow(*seq_file*, *split_options*)
preparing sequences to be used in the learning framework

split_dataset(*seqs_info*, *split_options*)

train_model(*trainseqs_id*, *crf_model*)
training a model and return the directory of the trained model

traineval_folds(*data_split*, ***kwargs*)
train and evaluate model on different dataset splits

verify_template()
verifying template – sanity check

Module contents

CHAPTER 6

Contact

For any questions, please email Ahmed Allam or Michael Krauthammer.

For reporting bugs or issues, please refer to bitbucket Issue Tracker.

CHAPTER 7

Funding and Support

This work is supported by grant number P2TIP1-161635 awarded by the Swiss National Science Foundation.

CHAPTER 8

Citation

To cite PySeqLab in your work, please use:

Allam A, Krauthammer M. PySeqLab an open source Python package for sequence labeling and segmentation. *TBD*

CHAPTER 9

License

This work is open source and distributed under the MIT license. See the `license.txt` file accompanying the package.

CHAPTER 10

Indices

- genindex
- modindex
- search

Python Module Index

p

Index

D

E

F

G

H

I

J

K

L

M

N

P

Q

R

S

T

U

V

W

X

Y

Z