SynSetMine Documentation

Release 0.1.0

Jiaming Shen

Nov 26, 2021

Contents:

This project presents a distantly-supervised synonym set discovery tool.

Details about SynSetMine can be accessed here, and the implementation is based on PyTorch 0.4.1.

CHAPTER 1

data loader

1.1 dataloader.element_set module

class dataloader.element_set.**ElementSet**(*name*, *data_format*, *options*, *raw_data_strings=None*)

Bases: object

Dataset Object

Parameters

- **name** (*str*) – dataset name
- **data_format** (*str*) – dataset format, either "set" or "sip"
- **options** (*dict*) – dataset parameters, including two dicts mapping element to element index
- **raw_data_strings** (*list*) – a list of strings representing an element set.
  - If data_format is "set", each string is of format "c0 {'d93', 'd377', 'd141', 'd63', 'd166'}".
  - If data_format is "sip", each string is of format "{'d93', 'd377'} d141 0".

_convert_set_format_to_sip_format(*raw_sets*, *pos_strategy*, *neg_strategy*, *neg_sample_size=10*, *subset_size=5*, *max_set_size=50*)

Generate <set, instance> pairs (sip) from a collection of sets

Parameters

- **raw_sets** (*list*) – a list of sets
- **pos_strategy** (*str*) – name of positive sampling method
- **neg_strategy** (*str*) – name of negative sampling method
- **neg_sample_size** (*int*) – negative sampling ratio

- **subset_size** (*int*) – size of "set" in <set, instance> pairs, used only in "fix_size_repeat_set" pos_strategy
- **max_set_size** (*int*) – maximum size of "set" in <set, instance> pairs, used only in "vary_size_enumerate" pos_strategy

Returns len(raw_sets) * (1 + neg_sample_size) sips, among which len(raw_sets) sips are positive and len(raw_sets) * neg_sample_size sips are negative

Return type list

Notes

- if pos_strategy is "sample_size_repeat_set", for each original set, we sample the size of "set" in sip, repeat this generated set neg_sample_size times, and pair them with each negative instance. This is the strategy to original AAAI submission.
- if pos_strategy is "sample_size_random_set", for each original set, we sample one size of "set" in sip, and generate one set for each negative instance.
- if pos_strategy is "fix_size_repeat_set", for each original set, we use pre-determined subset size to generate one "set" in sip, repeat this generated set neg_sample_size times, and pair them with each negative instance. This is the one used in cold-start training.
- if pos_strategy is "vary_size_enumerate", for each original set and for each subset size less than max_set_size, we enumerate the original set and generate all possible sips. This is the one used for converting test_set in "set" format to "sip" format.
- if pos_strategy is "vary_size_enumerate_with_full_set", it's basically same as the "vary_size_enumerate" strategy, except that it will also generate full set with only negative instances
- if pos_strategy is "vary_size_enumerate_with_full_set_plus_group_id", it's basically same as the "vary_size_enumerate_with_full_set" strategy, expect that it will also return the group id of each sip the group id is this sip's corresponding raw set index
- if pos_strategy is "enumerate", this is the one used for pre-generating sip triplets

_convert_sip_format_to_tensor(*max_set_size*, *batch_set*, *batch_inst*, *labels*)

Generate tensors for <set, instance> pairs

Parameters

- **max_set_size** (*int*) – maximum size of "set" in <set, instance> pairs
- **batch_set** (*list*) – a list of "sets" in <set, instance> pairs
- **batch_inst** (*list*) – a list of "instances" in <set, instance> pairs
- **labels** (*list*) – a list of labels for each above <set, instance> pair

Returns a dict of pytorch tensors representing <set, instance> pairs with their corresponding labels

Return type dict

_generate_negative_samples_within_pool(*positive_sets*, *neg_sample_size*, *remove_pos=True*)

Generate negative samples from vocabulary

Parameters

- **positive_sets** (*list*) – a list of positive sets

- **neg_sample_size** (*int*) – negative sampling ratio
- **remove_pos** (*bool*) – whether to remove instances in positive sets from the vocabulary

Returns a list of negative sets

Return type list

_initialize_set_format(*raw_set_strings*)
Initialize dataset from a collection of strings representing element sets

Parameters **raw_set_strings** (*list*) – a list of strings representing element sets

Returns None

Return type None

_initialize_sip_format(*raw_set_instance_strings*)
Initialize dataset from a collection of strings representing <set, instance> pairs

Parameters **raw_set_instance_strings** (*list*) – a list of strings representing <set instance> pairs

Returns None

Return type None

_shuffle()
Shuffle dataset

Returns None

Return type None

get_test_batch(*max_set_size=5*, *batch_size=32*)
Generate one testing batch of <set, instance> pairs

Parameters

- **max_set_size** (*int*) – maximum size of set S
- **batch_size** (*int*) – number of **<set, instance> pairs** in one batch

Returns a testing batch containing "batch_size" <set, instance> pairs

Return type dict

get_train_batch(*max_set_size=100*, *pos_sample_method='sample_size_random_set'*, *neg_sample_size=1*, *neg_sample_method='complete_random'*, *batch_size=32*)
Generate one training batch of <set, instance> pairs

Parameters

- **max_set_size** (*int*) – maximum size of set S
- **pos_sample_method** (*str*) – name of positive sampling method
- **neg_sample_size** (*int*) – number of negative samples for each set
- **neg_sample_method** (*str*) – name of negative sampling method
- **batch_size** (*int*) – number of **sets** in one batch

Returns a training batch containing "batch_size * (1+neg_sample_size)" <set, instance> pairs

Return type dict

CHAPTER 2

model

class model.**SSPM**(*params*)

Bases: sphinx.ext.autodoc.importer._MockObject

Synonym Set Prediction Model (SSPM), namely SynSetMine

Parameters **params** (*dict*) – a dictionary containing all model specifications

_get_test_sip_batch_size(*x*)

_set_scorer(*set_tensor*)

Return the quality score of a batch of sets

Parameters **set_tensor** (*tensor*) – sets to be scored, size: (batch_size, max_set_size)

Returns scores of all sets, size: (batch_size, 1)

Return type tensor

initialize(*params*)

Initialize model components

Parameters **params** (*dict*) – a dictionary containing all model specifications

Returns None

Return type None

predict(*batch_set_tensor*, *batch_inst_tensor*)

Make set instance pair prediction

Parameters

- **batch_set_tensor** (*tensor*) – packed sets in a collection of <set, instance> pairs, size: (batch_size, max_set_size)
- **batch_inst_tensor** (*tensor*) – packed instances in a collection of <set, instance> pairs, size: (batch_size, 1)

Returns

- scores of packed sets, (batch_size, 1)

- scores of packed sets union with corresponding instances, (batch_size, 1)
- the probability of adding the instance into the corresponding set, (batch_size, 1)

Return type tuple

train_step(*train_batch*)
Train the model on the given train_batch

Parameters **train_batch** (*dict*) – a dictionary containing training batch in <set, instance> pair format

Returns batch_loss, true_positive_num, false_positive_num, false_negative_num, true_positive_num

Return type tuple

model.**initialize_weights**(*moduleList*, *itype='xavier'*)
Initialize a list of modules

Parameters

- **moduleList** (*list*) – a list of nn.modules
- **itype** (*str*) – name of initialization method

Returns None

Return type None

CHAPTER 3

cluster predictor

cluster_predict.**multiple_set_single_instance_prediction**(*model*, *sets*, *instance*, *size_optimized=False*)

Apply the given model to predict the probabilities of adding that one instance into each of the given sets

Parameters

- **model** (SSPM) – a trained SynSetMine model
- **sets** (*list*) – a list of sets, each contain the element index
- **instance** (*int*) – a single instance, represented by the element index
- **size_optimized** (*bool*) – whether to optimize the multiple-set-single-instance prediction process. If the size of each set in the given 'sets' varies a lot and there exists a single huge set in the given 'sets', set this parameter to be True

Returns

- scores of given sets, (batch_size, 1)
- scores of given sets union with the instance, (batch_size, 1)
- the probability of adding the instance into the corresponding set, (batch_size, 1)

Return type tuple

cluster_predict.**set_generation**(*model*, *vocab*, *threshold=0.5*, *eid2ename=None*, *size_opt_clus=False*, *max_K=None*, *verbose=False*)

Set Generation Algorithm

Parameters

- **model** (SSPM) – a trained set-instance classifier
- **vocab** (*list*) – a list of elements to be clustered, each element is represented by its index
- **threshold** (*float*) – the probability threshold for determine whether to create new singleton cluster

- **eid2ename** (*dict*) – a dictionary mapping element index to its corresponding (human-readable) name
- **size_opt_clus** (*bool*) – a flag indicating whether to optimize the multiple-set-single-instance prediction process
- **max_K** (*int*) – maximum number of clusters, If None, we will infer this number automatically
- **verbose** (*bool*) – whether to print out all intermediate results

Returns a list of detected clusters

Return type list

CHAPTER 4

evaluator

evaluator.**calculate_km_matching_score**(*weight_nm*)
Calculate maximum weighted matching score

Parameters **weight_nm** (*list*) – a similarity matrix

Returns weighted matching score

Return type float

evaluator.**calculate_precision_recall_f1**(*tp*, *fp*, *fn*)
Calculate precision, recall, and f1 score

Parameters

- **tp** (*int*) – true positive number
- **fp** (*int*) – false positive number
- **fn** (*int*) – false negative number

Returns (precision, recall, f1 score)

Return type tuple

evaluator.**end2end_evaluation_matching**(*groundtruth*, *result*)
Evaluate the maximum weighted jaccard matching of groundtruth clustering and predicted clustering

Parameters

- **groundtruth** (*list*) – a list of element lists representing the ground truth clustering
- **result** (*list*) – a list of element lists representing the model predicted clustering

Returns best matching score

Return type float

evaluator.**evaluate_clustering**(*cls_pred*, *cls_true*)
Evaluate clustering results

Parameters

- **cls_pred** (*list*) – a list of element lists representing model predicted clustering
- **cls_true** (*list*) – a list of element lists representing the ground truth clustering

Returns a dictionary of clustering evaluation metrics

Return type dict

evaluator.**evaluate_set_instance_prediction**(*model*, *dataset*)

Evaluate model on the given dataset for set-instance pair prediction task

Parameters

- **model** (SSPM) – a trained set-instance classifier
- **dataset** (ElementSet) – an ElementSet dataset with

Returns a dictionary of set-instance pair prediction metrics

Return type dict

CHAPTER 5

utils

class utils.**Metrics**
Bases: object

A metric class wrapping all metrics

add(*metric_name*, *metric_value*)
Add metric value for the given metric name

Parameters

- **metric_name** (*str*) – metric name
- **metric_value** – metric value

Returns None

Return type None

class utils.**Results**(*filename*)
Bases: object

A result class for saving results to file

Parameters **filename** (*str*) – name of result saving file

save_metrics(*hyperparams*, *metrics*)
Save model hyper-parameters and evaluation results to the file

Parameters

- **hyperparams** (*dict*) – a dictionary of model hyper-parameters, keyed with the hyper-parameter names
- **metrics** (Metrics) – a Metrics object containg all model evaluation results

Returns None

Return type None

utils.**check_model_consistency**(*args*)
Check whether the model architecture is consistent with the loss function used

Parameters **args** (*dict*) – a dictionary containing all model specifications

Returns a flag indicating whether the model architecture is consistent with the loss function, if not, also return the error message

Return type a tuple of (bool, str)

utils.**get_num_lines**(*file_path*)

Return the number of lines in the file without actually reading them into the memory. Used together with tqdm for tracking file reading progress.

Usage:

```
with open(inputFile, "r") as fin:
    for line in tqdm(fin, total=get\_num\_lines(inputFile)):
        ...
```

Parameters **file_path** (*str*) – path of input file

Returns number of lines in the file

Return type int

utils.**load_checkpoint**(*model*, *optimizer*, *load_dir*, *load_prefix*, *step*)

Load model checkpoint (including trained model, training epoch, and optimizer) from a file

Notes

- The loaded model and optimizer are initially on CPU and need to be explicitly moved to GPU c.f. https://discuss.pytorch.org/t/loading-a-saved-model-for-continue-training/17244/3.
- You need to first initialize a model which has the same architecture/size of the model to be loaded.

Parameters

- **model** (*torch.nn*) – a model which has the same architecture of the model to be loaded
- **optimizer** (*torch.optim*) – a pytorch optimizer
- **load_dir** (*str*) – model save directory
- **load_prefix** (*str*) – model snapshot prefix
- **step** (*int*) – model training epoch

Returns None

Return type None

utils.**load_embedding**(*fi*, *embed_name='word2vec'*)

Load pre-trained embedding from file

Parameters

- **fi** (*str*) – embedding file name
- **embed_name** (*str*) – embedding format, currently only supports "word2vec" format embedding. c.f.: https://radimrehurek.com/gensim/models/keyedvectors.html

Returns

- embedding : embedding file

- index2word: map from element index to element
- word2index: map from element to element index
- vocab_size: size of element pool
- embed_dim: embedding dimension

Return type (gensim.KeyedVectors, list, dict, int, int)

utils.**load_model**(*model*, *load_dir*, *load_prefix*, *steps*)
load model from file

Note: You need to first initialize a model which has the same architecture/size of the model to be loaded.

Parameters

- **model** (*torch.nn*) – a model which has the same architecture of the model to be loaded
- **load_dir** (*str*) – model save directory
- **load_prefix** (*str*) – model snapshot prefix
- **steps** (*int*) – model training epoch

Returns None

Return type None

utils.**load_raw_data**(*fi*)
Load raw data from file

Parameters **fi** (*str*) – data file name

Returns a list of raw data from file

Return type list

utils.**my_logger**(*name=''*, *log_path='./'*)
Create a python logger

Parameters

- **name** (*str*) – logger name
- **log_path** (*str*) – path for saving logs

Returns a logger for logging messages

Return type python logger

utils.**print_args**(*args*, *interested_args='all'*)
Print arguments in command line

Parameters

- **args** (*Namespace*) – parsed command line argument
- **interested_args** (*list*) – a list of interested argument names

Returns None

Return type None

utils.**save_checkpoint**(*model*, *optimizer*, *save_dir*, *save_prefix*, *step*)
Save model checkpoint (including trained model, training epoch, and optimizer) to a file

Parameters

- **model** (*torch.nn*) – a trained model

- **optimizer** (*torch.optim*) – a pytorch optimizer
- **save_dir** (*str*) – model save directory
- **save_prefix** (*str*) – model snapshot prefix
- **step** (*int*) – model training epoch

Returns None

Return type None

utils.**save_model**(*model*, *save_dir*, *save_prefix*, *steps*)
Save model to file

Parameters

- **model** (*torch.nn*) – a trained model
- **save_dir** (*str*) – model save directory
- **save_prefix** (*str*) – model snapshot prefix
- **steps** (*int*) – model training epoch

Returns None

Return type None

utils.**to_gpu**(*optimizer*, *device*)
Move optimizer from CPU to GPU

Parameters

- **optimizer** (*torch.optim*) – a pytorch optimizer
- **device** (*torch.device*) – a pytorch device, CPU or GPU

Returns None

Return type None

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index

Symbols

A

C

D

E

G

I

L

M

P

R

S

T

U