Bachelor Thesis Documentation

Release v1.0

Bachelor Thesis

Jul 31, 2016

Contents:

CHAPTER

ONE

SRC

1.1 src package

The `src` package contains all code used in this project. It is divided into multiple packages:

- `src.clustering` : Clustering of mapping vectors.
- `src.eval` : Evaluation of word embeddings
- `src.mapping` : Creation of mapping vectors from word embedding pairs.
- `src.misc` : Helper function and decorators.
- `src.prep` : Scripts used for different preparations stops for fundamental resources.
- `src.trans_e` : *TransE* related preparation scripts as well as *TransE* inspired Training with word embeddings.

1.1.1 Subpackages

src.clustering package

src.clustering.cluster_mappings module

Script to cluster mapping vectors created with *src.mapping.mapthreading* .

aggregate_cluster (*points*, *labels*)

Arranges all clusters in a list, where a sublist with all points at index i corresponds with the custer with label i.

Parameters

- **points** (*list*) – List of datapoints
- **labels** (*list*) – List of unique cluster labels

Returns list of lists of datapoints belonging to the i-th cluster

Return type list

cluster_mappings (*vector_inpath*, *do_pca=False*, *target_dim=100*, *indices_inpath=None*, *epsilon=2.625*, *min_s=20*)

Cluster mapping vectors created with *src.mapping.mapthreading* or `rc.mapping.map_vectors.py` . Because just reading about the number of clusters and their sizes, there's an option to resolve the indices of the vectors in the cluster to their original word pairs.

Parameters

- **vector_inpath** (*str*) – Path to vector file. File should have the following format (separated by spaces): <index of original vector #1> <index of original vector #2> <Dimension 1> ... <Dimension n>
- **do_pca** (*bool*) – Flag to indicate whether PCA should be executed before clustering to reduce amount of
- **computation.** –
- **target_dim** (*int*) – Number of dimensions vectors should be shrunk to in case PCA is performed.
- **indices_inpath** (*str*) – Path to file with the indices given to words. The file should have the following format: <index of word> <word> (separated by tab)
- **epsilon** (*float*) – Radius of circle DBSCAN uses to look for other data points.
- **min_s** (*int*) – Minimum number of points in radius epsilon DBSCAN needs to declare a point a core object.

get_cluster_size (*labels*)

Calculate the size of every cluster found by DBSCAN.

Parameters **labels** (*list*) – List of cluster IDs assigned to every data point.

Returns Dictionary of cluster sizes with cluster id as key and cluster size as value.

Return type defaultdict

init_argparser ( )

Initialize all possible arguments for the argument parser.

Returns ArgumentParser object with command line arguments for this script.

Return type `argparse.ArgumentParser`

load_indices (*indices_inpath*)

Load word indices from a file. The file should have the following format: <index of word> <word> (separated by tab)

Parameters **indices_inpath** (*str*) – Path to index file.

load_mappings_from_model (*mapping_inpath*)

Load mapping vectors from file.

Parameters **mapping_inpath** – Path mapping vector file.

Returns A tuple of a list of word index pairs and a dictionary (defaultdict) with index pair tuple as key and mapping vector (as numpy.array) as value.

Return type tuple

main ( )

This is the main function. It uses the parsed command line arguments to pick the right function to execute.

resolve_indices (*points*, *labels*, *indices_inpath*)

Resolves the indices of word pairs found in a cluster to their real names.

Parameters

- **points** (*list*) – List of datapoints
- **labels** (*list*) – List of unique cluster labels
- **indices_inpath** (*str*) – Path to file with the indices given to words. The file should have the following format: <index of word> <word> (separated by tab)

train_clustering_parameters (*vector_inpath*)
Functions that tries to figure out the optimal clustering parameters in regard to DBSCAN's epsilon, min_samples and p.

Parameters **vector_inpath** (*str*) – Path to vector file. File has to have the following format (separated by spaces): <index of original vector #1> <index of original vector #2> <Dimension 1> ... <Dimension n>

src.eval package

src.eval.analogy module

Module to evaluate word embeddings by the means of analogies like "W is to X like Y is to Z". Usually, the system uses the word embeddings of word W, X, Y and tries to find the vector of word Z that is most similar to X and Y and most dissimilar to W. Therefore, the CosMul method (Levy et al., 2015) is used.

The whole module is used in `src.eval.eval_vectors.py` .

analogy_eval (*vector_inpath*, *analogy_path*, *per_section=False*)
Perform analogy evaluation. Usually, the system uses the word embeddings of word W, X, Y and tries to find the vector of word Z that is most similar to X and Y and most dissimilar to W for an analogy like "W is to X like Y is to Z." Therefore, the CosMul method (Levy et al., 2015) is used.

Parameters

- **vector_inpath** (*str*) – Path to *word2vec* vector file.
- **analogy_path** (*str*) – Path to analogy file.
- **per_section** (*bool*) – Flag to indicate whether analogies test should be conducted section-wise or just all in one run.

read_analogies (*analogy_path*, *per_section=False*)
Reads a file with analogies.

Parameters

- **analogy_path** (*str*) – Path to analogy file.
- **per_section** (*bool*) – Flag to indicate whether analogies test should be conducted section-wise or just all in one run. In this function, the section will be put into a data structure accordingly.

Returns Dictionary with section header as key, list of analogy as 4-tuples as value.

Return type dict

src.eval.eval_vectors module

Main module used to evaluate word embeddings. It offers the following options:

1.) Analogy: The system tries to complete an analogy like "W is to X like Y is to...?" The percentage of correct answers is measured.

2.) Word similarity: The system assign word pairs a similarity score based on the cosine similarity of their word embeddings. Then, to correlation between those and human ratings is measured with Pearson's rho.

3.) Nearest neighbors: Find the nearest neighbors for a list of words based on their word embeddings. Good for a first look on the data, but not quantifiable.

4.) Visualize: Plot word embeddings in 2D or 3D. Fancy plots. Yay!

find_nearest_neighbors (*vector_inpath*, *max_n*, *wordlist*)
Find the nearest neighbors for a list of words based on their word embeddings.

Parameters

- **vector_inpath** (*str*) – Path to vector file. File has to have the following format (separated by spaces): <index of original vector #1> <index of original vector #2> <Dimension 1> ... <Dimension n>
- **max_n** (*int*) – Number of nearest neighbors that should be determined.
- **wordlist** (*list*) – List of words nearest neighbors should be found for.

init_argparser ( )
Initialize all possible arguments for the argument parser.

Returns ArgumentParser object with command line arguments for this script.

Return type argparse.ArgumentParser

main ( )
This is the main function. It uses the parsed command line arguments, especially *–mode*, to pick the right function to execute.

plot (*vector_inpath*, *max_n*, *target_dim*, *show_plot=False*, *display_names=False*)
Plot word embeddings in 2D or 3D. As a heuristic, word will only be plotted after the 50th most frequent words to avoid plotting boring stop words.

Parameters

- **vector_inpath** (*str*) – Path to vector file. File has to have the following format (separated by spaces): <index of original vector #1> <index of original vector #2> <Dimension 1> ... <Dimension n>
- **max_n** (*int*) – Maximum number of vectors to be plotted.
- **show_plot** (*bool*) – Flag to indicate whether a window with the (interactive) plot should pop up after executing the script.
- **display_names** (*bool*) – Flag to indicate whether the words should acutally be shown next to the data point in the plot. Can get very messy with higher *max_n*.

src.eval.word_similarity module

Module used to conduct the word similarity evaluation. The system assign word pairs a similarity score based on the cosine similarity of their word embeddings. Then, to correlation between those and human ratings is measured with Pearson's rho.

The whole module is used in src.eval.eval_vectors.py .

evaluate_wordpair_sims (*x*, *y*, *number_of_pairs*)
Evaluate results of the similarity score assignments, i.e. calculate pearson's rho and its significance.

Parameters

- **x** (*list*) – List of similarity scores assigned by humans.
- **y** (*list*) – List of similarity scores assigned by the system.
- **number_of_pairs** (*int*) – Number of word pairs evaluated.

Returns **rho** – Pearson's correlation coefficient. t (float): Student's t value. z (float): z value.

Return type float

read_wordpairs (*wordpair_path*, *format='google'*)
Read wordpair file with wordpairs and their similarity scores assigned by humans.

Parameters

- **wordpair_path** (*str*) – Path to word pair file.
- **format** (*str*) – Format of wor pair file {google|semrel}

Returns Tuple of a list of word pairs and a list of similarity scores for those same pair assigned by humans.

Return type tuple

remove_unknowns (*x*, *y*)
Remove word pairs from the results where one or two word embedding weren't found.

Parameters

- **x** (*list*) – List of similarity scores assigned by humans.
- **y** (*list*) – List of similarity scores assigned by the system.

Returns **x** – Purged list of similarity scores assigned by humans. y (list): Purged list of similarity scores assigned by the system.

Return type list

word_sim_eval (*vector_inpath*, *wordpair_path*, *format='google'*)
Function that let's the system assign word pairs a similarity score based on the cosine similarity of their word embeddings. Then, to correlation between those and human ratings is measured with Pearson's rho.

Parameters

- **vector_inpath** (*str*) – Path to vector file. File has to have the following format (separated by spaces): <index of original vector #1> <index of original vector #2> <Dimension 1> ... <Dimension n>
- **wordpair_path** (*str*) – Path to word pair file.
- **format** (*str*) – Format of word pair file {google|semrel}

src.mapping package

src.mapping.mapthreading module

Module used to map a pair of vectors into a new combined vector space. Those mappings will be created by multiple threads in a master-slave-pattern. To do so, the user can choose between different vector operations as offset, cosine similarity, euclidean distance and many more.

Warning: Because of $\Omega = \frac{n(n-1)}{2}$, it is recommended to use the co-occurrence constraint Λ, which limits the calculations to word embedding pairs which words occurred together in a corpus in at least *n* sentences (but it will still take quite a while).

class MappingMasterThread (*n*, *vector_inpath*, *vector_outpath*, *features*, *lambda_*, *ids_inpath*, *indices_inpath*)
Bases: threading.Thread

Master thread class. The master thread loads all necessary data into suitable data structures and distributes them among all worker threads.

prepare ( )
Loads The master thread loads all necessary data into suitable data structures. To be more specific, word embeddings, sentence IDs and word indices are processed.

read_ids_file (*ids_inpath*)
Read the sentence ID file.

Parameters **ids_inpath** (*str*) – Path to sentence IDs file. The file should be in the following *YAML*-format: - <word>:

- <sentence id>
- <sentence id>

...

Returns Dictionary with words as keys and the IDs of the sentences they occur in in a set as value.

Return type defaultdict

start_threads ( )
Starts all the threads (and ends them if they're all finished).

class **MappingWorkerThread** (*worker_id*, *vector_dict*, *vector_queue*, *vector_outpath*, *features*, *occurrences*, *indices*, *lambda_*)
Bases: threading.Thread

Worker thread class. The worker threads do all the dirty work after they receive all necessary data from the master thread an try to calculate every possible combinations of two word embeddings in a dataset.

All the word embeddings will be stores in a dictionary (*VectorDict* as well as a Queue). An idle thread picks a new vector from the queue and then starts to iterate over all the vectors in the *VectorDict* (this way, the queue gets shorter over time while the size of the dictionary stays fixed).

Before it starts calculations, it checks a) if the co-occurrence constraint is satisfied and b) if this combination of word embeddings has already been processed.

cosine_similarity (*v1*, *v2*)
Calculates the cosine similarity ($cos(\vec{v}_1, \vec{v}_2) \in [-1, -1]$) between two vectors.

Parameters

- **v1** (*numpy.array*) – First vector
- **v2** (*numpy.array*) – Second vector

Returns Cosine similarity between the two vectors.

Return type float

distance (*v1*, *v2*)
Return the vector offset of two vectors:

Parameters

- **v1** (*numpy.array*) – First vector
- **v2** (*numpy.array*) – Second vector

Returns numpy.array: Vector offset.

euclidean_distance1 (*v1*, *v2*)

Return the euclidean distance between two vectors.

$$eucl(\vec{a}, \vec{b}) = \sqrt{\sum_{i=1}^{n}(\vec{b}_i - \vec{a}_i)^2}$$

Parameters

- **v1** (*numpy.array*) – First vector
- **v2** (*numpy.array*) – Second vector

Returns Euclidean distance between the two vectors.

Return type float

euclidean_distance2 (*v1*, *v2*)

Returns the squared euclidean distance between two vectors.

$$eucl2(\vec{a}, \vec{b}) = \sum_{i=1}^{n}(\vec{b}_i - \vec{a}_i)^2$$

Parameters

- **v1** (*numpy.array*) – First vector
- **v2** (*numpy.array*) – Second vector

Returns Squared euclidean distance between the two vectors.

Return type float

hash_indices (*i1*, *i2*)

Combines two vector indices (the indices of the words' embeddings used in vector operations) into a hash s.t. threads can do an easy lookup if a mapping vector has already been calculated. To guarantee this, $h(i_1, i_2) = h(i_2, i_1)$ has to be the case.

Parameters

- **i1** (*int*) – Index of first word's embedding
- **i2** (*int*) – Index of second word's embedding

Returns Unique hash for index pair.

Return type int

manhattan_distance (*v1*, *v2*)

Returns the manhattan distance between two vectors.

$$manhattan(\vec{a}, \vec{b}) = \sum_{i=1}^{n}|\vec{b}_i - \vec{a}_i|$$

Parameters

- **v1** (*numpy.array*) – First vector
- **v2** (*numpy.array*) – Second vector

Returns Manhattan distance between the two vectors.

Return type float

run ( )

Starts a worker thread.

soft_cosine_similarity (*v1*, *v2*)

Calculates the soft cosine similarity between two vectors.

$$S = \begin{bmatrix} eucl(\vec{a}_1, \vec{b}_1) & \dots & eucl(\vec{a}_1, \vec{b}_n) \\ \vdots & \ddots & \vdots \\ eucl(\vec{a}_n, \vec{b}_1) & \dots & eucl(\vec{a}_n, \vec{b}_n) \end{bmatrix}$$

$$softcos(\vec{a}, \vec{b}) = \frac{\sum_{i,j}^{N} S_{ij} \vec{a}_i \vec{b}_j}{\sqrt{\sum_{i,j}^{N} S_{ij} \vec{a}_i \vec{a}_j} \sqrt{\sum_{i,j}^{N} S_{ij} \vec{b}_i \vec{b}_j}}$$

(It considers the similarity between pairs of features.)

Parameters

- **v1** (*numpy.array*) – First vector
- **v2** (*numpy.array*) – Second vector

Returns Soft cosine similarity between the two vectors.

Return type float

class VectorDict

Bases: object

VectorDict class that serves two functions:

1.) Storing word embeddings so they don't allocate memory for every worker thread

2.) Providing a set, where are processed vector pairs are stored so no redundant computations are made.

Locks are used for synchronization purposes.

add_skippable (*index_hash*)

Add the hash of an index pair to a set of already processed vector pairs.

Parameters **index_hash** (*int*) – Hash value of index pair. Produced with `hash_indices()`.

add_vector (*index*, *vector*)

Add a new word embedding.

Parameters

- **index** (*int*) – Index of the word the embedding belongs to.
- **vector** (*numpy.array*) – Word embedding corresponding to given index.

get_keys ( )

Get all the keys (word embedding IDs) of this dictionary.

Returns List of word embedding IDs.

Return type list

get_vector (*index*)

Get a word embedding given its word's index.

Parameters **index** (*int*) – Index of the word the embedding belongs to.

Returns Word embedding corresponding to given index.

Return type numpy.array

skippable (*index_hash*)

Checks whether a pair of vectors has already been processed.

Parameters **index_hash** (*int*) – Hash value of index pair. Produced with `hash_indices()`.

Returns Whether a pair of vectors has already been processed.

Return type bool

alt (*func*)
Prepends the local time to the output of a function.

Parameters **func** (*function*) – Function the local time should be prepended to.

init_argparse ( )
Initialize all possible arguments for the argument parser.

Returns ArgumentParser object with command line arguments for this script.

Return type `argparse.ArgumentParser`

main ( )
Main function that initializes the master thread with command line arguments and starts it.

Module contents

src.misc package

src.misc.decorators module

This module contains decorators that wrap around functions used in other modules.

log_time (*logpath='log.txt'*, *interval=5*)
This decorator is used to log the execution time of a function into a given logfile, following a constant interval.

Parameters

- **logpath** (*str*) – Path to logfile.
- **interval** (*int*) – Logging interval in seconds.

Returns Decorator with function.

Return type function

src.misc.helpers module

This module contains decorators that wrap aroud functions used in other modules.

alt (*func*)
Prepends the local time to the output of a function.

Parameters **func** (*function*) – Function the local time should be prepended to.

capitalize (*word*)
Capitalizes a string.

Parameters **word** (*str*) – Word to be capitalized.

Returns Capitalized word.

Return type str

contains_tag (*line*)
Checks whether the current line contains an xml tag.

Parameters **line** (*str*) – Current line

Returns Whether the current line contains an xml tag.

Return type bool

extract_sentence_id (*tag*)
Extract the sentence ID of current sentence.

Parameters **tag** (*str*) – Sentence tag

Returns sentence ID

Return type str

format_fbid (*fbid*)
Transform the format of the *Freebase* IDs from the format used in the dataset to the format used in requests.

Parameters **fbid** (*str*) – *Freebase* ID to be formatted.

Returns Formatted *Freebase* ID.

Return type str

load_vectors (*vector_inpath*)
Load word embeddings, gensim style.

Parameters **vector_inpath** (*str*) – Path to vector file.

Returns Word2Vec gensim model.

Return type gensim.models.Word2Vec

load_vectors_from_model (*vector_inpath*, *max_n=None*, *indices=False*)
Load word embeddings (or mapping vectors), my style.

Parameters

- **vector_inpath** (*str*) – Path to vector file.
- **max_n** (*int*) – Maximum number of vectors to load.
- **indices** (*bool*) – Flag to indicate the loading of mapping vectors.

Returns A list of words as well as a dictionary with the vectors as numpy.arrays as value and their corresponding words or index pairs as keys.

Return type Tuple

partitions_list (*l*, *prts*)
Partitions a list into three parts according to their percentages in regard to the length of the original list given in a tuple as floats.

Parameters

- **l** (*list*) – List to be partitioned.
- **prts** (*tuple*) – Tuple of float with new list sizes.

Returns Tuple of the three new lists.

Return type tuple

read_dataset (*inpath*)
Reads a generic dataset with rows separated by tabs into a list.

Parameters **inpath** (*str*) – Path to dataset.

Returns List of line contents as tuples.

Return type list

src.prep package

Subpackages

src.prep.corpus package

src.prep.corpus.convert_to_plain module

Convert the *DECOW14X* corpus into a plain text file. Is used as pre-processing step for the word2vec training. To make this this more feasible (decow is a **huge** corpus), python's `multiprocessing` is used, s.t. every part of the corpus in simultaneously processed. Afterwards, a bash command like `cat` can be used to merge into one single file.

convert_decow_to_plain (*decow_dir*, *out_dir*, *log_path*, *merge_nes*, *log_interval*)
Convert the whole corpus into plain text.

Parameters

- **decow_dir** (*str*) – Path to directory with decow corpus paths.
- **out_dir** (*str*) – Path where plain text parts should be written to.
- **log_path** (*str*) – Path where the log files should be written to.
- **merge_nes** (*bool*) – Flag to indicate whether multi-word expression should be merged with underscores.
- **log_interval** (*int*) – Interval to log current process state in seconds.

convert_part (*argstuple*)
Convert a corpus part into plain text without merging multiple word entries.

Parameters **argstuple** – Tuple of methods arguments (`inpath` (*str*): Path to this processes' corpus part / `dir_outpath` (*str*): Path to this processes' output / `log_path` (*str*): Path to this processes' log / `interval` (*int*): Logging interval in seconds)

convert_part_merging (*argstuple*)
Convert a corpus part into plain text and merging multiple word entries.

Parameters **argstuple** – Tuple of methods arguments (`inpath` (*str*): Path to this processes' corpus part / `dir_outpath` (*str*): Path to this processes' output / `log_path` (*str*): Path to this processes' log / `interval` (*int*): Logging interval in seconds)

extract_named_entity (*line*)
Extract named entity from current line.

Parameters **line** (*str*) – Current line

Returns Extracted named entity or None if no named entity is present.

Return type str or None

get_file_number (*filename*)
Get the number of the current decow corpus part.

Parameters **filename** (*str*) – Decow corpus part file name

Returns File number

Return type str

main ( )
Main function. Uses command lines to start corpus processing.

src.prep.corpus.extract_conll module

This script can be used to extract information out of a specific column of a file in the CoNLL-format.

extract_conll (*inpath*, *outpath*, *column*)
Extract information out of CoNLL files.

Parameters

- **inpath** (*str*) – Path to input file.
- **outpath** (*str*) – Path to output file.
- **column** (*int*) – The number (-1) of the column the information should be extracted from.

init_argparse ( )
Initialize all possible arguments for the argument parser.

Returns ArgumentParser object with command line arguments for this script.

Return type argparse.ArgumentParser

main ( )
The main function.

src.prep.corpus.mapper module

Mapper classed used to count frequencies of words in a corpus. Corpus has to be in plain text format. This class is used in a Map-Reduce-pattern, so you also need the `reducer.py` class.

Then, you can open your terminal and pipe them together:

```
> cat corpus.txt | ./mapper.py | sort | ./reducer.py
```

Also, you probably have to remove the `if __name__ == "__main__":` line and unindent the remaining code, this is only due to sphinx being picky and not documenting plain python scripts at all.

src.prep.corpus.reducer module

Reducer classed used to count frequencies of words in a corpus. Corpus has to be in plain text format. This class is used in a Map-Reduce-pattern, so you also need the `mapper.py` class.

Then, you can open your terminal and pipe them together:

```
> cat corpus.txt | ./mapper.py | sort | ./reducer.py
```

Also, you probably have to remove the `if __name__ == "__main__":` line and unindent the remaining code, this is only due to sphinx being picky and not documenting plain python scripts at all.

src.prep.nes package

src.prep.nes.extract_nes module

This script is used to find all named entities in a corpus, extract them and also store their frequencies as well as the IDs of the sentences they occur in.

extract_named_entity (*line*)
Extracts named entity from current line if present.

Parameters **line** (*str*) – Current line

Returns Named entity in this line and its NE tag

Return type tuple

main ( )
Main function.

process (*inpath*, *outpath*, *logpath*)
Starts extracting named entities and their corresponding sentence IDs.

Parameters

- **inpath** (*str*) – Path to input file. Input file is a gzipped xml file.
- **outpath** (*str*) – Path to output directory.
- **logpath** (*str*) – Path to log directory.

write_dict_into_file (*dictionary*, *out_path*)
Write a dictionary of named entities, their tags and their frequencies into a file.

Parameters

- **dictionary** (*dict*) – Dictionary with named entities as key and their frequencies as values.
- **out_path** (*str*) – Path the frequencies should written to.

write_ids_into_file (*dictionary*, *out_path*)
Write a dictionary of named entities,, their tags and IDs of the sentences they occur in into a file.

Parameters

- **dictionary** (*dict*) – Dictionary with named entities as key and their occurrences as a list as values.
- **out_path** (*str*) – Path the frequencies should written to.

src.prep.nes.merge module

This module is used to merge various output files created from `extract_nes.py` . Because they are only created for one corpus part at a time, you end up with multiple files that cannot simply by concatenated. Therefore, this module aims to merge them in a (relatively) memory-efficient manner.

freq_worker (*inpath*)
Reads the named entity frequencies from a file.

Parameters **inpath** (*str*) – Path to frequency file.

Returns Dictionary with named entities as keys and their frequencies as values.

Return type dict

id_worker (*inpath*)
Reads the named entity ids from a file.

Parameters **inpath** (*str*) – Path to frequency file.

Returns Dictionary with named entities as keys and their ids as values.

Return type dict

main ( )
Main function, handling command line arguments.

merge_dicts (*dicttuple*)
Merges two dictionary (efficiently).

Parameters **dicttuple** (*tuple*) – Tuple of two frequency dictionaries.

Returns New merged dictionary

Return type dict

merge_frequency_files (*infiles_path*, *outpath*, *logpath*)
Merge multiple named entitiy frequency files.

Parameters

- **infiles_path** (*str*) – Path to input file directory.
- **outpath** (*str*) – Path to output directory.
- **logpath** (*str*) – Path to logging directory.

merge_id_dicts (*dicttuple*)
Merges two id dictionary (efficiently).

Parameters **dicttuple** (*tuple*) – Tuple of two id dictionaries.

Returns New merged dictionary

Return type dict

merge_id_files (*infiles_path*, *outpath*, *logpath*, *yaml=False*)
Merge multiple named entitiy id files.

Parameters

- **infiles_path** (*str*) – Path to input file directory.
- **outpath** (*str*) – Path to output directory.
- **logpath** (*str*) – Path to logging directory.
- **yaml** (*bool*) – Flag to indicate whether merged files should be written in yaml format.

rl (*infile*)
Lazy function to read a line from a while and remove redundant whitespaces.

Parameters **infile** (*str*) – Path to input file.

Returns Stripped line

Return type str

src.prep.nes.statistics module

This script collects a few statistics about named entities extracted from the corpus and the percentage of their occurrence in the *Freebase* relation dataset. Requires a relation file in yaml format and a merged named entity frequency file, see `extract_nes.py`, `merge.py` and `relations.py`.

calculate_occurrences (*freqpath*, *relations_path*)
Calculate statistics about named entities extracted from the corpus and the percentage of their occurrence in the *Freebase* relation dataset.

Parameters

- **freqpath** (*str*) – Path to merged frequencies file.
- **relations_path** (*str*) – Path to relation yaml file.

main ( )
Main function

src.prep.relations package

src.prep.relations.relations module

This modules is about retrieving the names of entities and relations in the FB15k dataset. Because the entities are used with their (quite cryptic) *Freebase* ids, those have to be resolved.

Warning: Unfortunately, it isn't possible anymore to use this code (July 2016), because the *Freebase API* is now deprecated; the whole *Freebase* project has been integrated into *Wikidata*. However, this code is still included to show the process of thow freebase where transformed into real names using the API and the MQL query language.

exception MissingTranslationException
Bases: `exceptions.Exception`

Exception class to be thrown in cases where the API cannot find a translation for a *Freebase* API given the target language.

get_id ( )
Return the *Freebase* ID that triggered this exception.

Returns *Freebase* ID that triggered this exception.

Return type str

fetch_name (*fbid*, *lang='de'*)
Looks for the translation of a *Freebase* id in a target language.

Parameters

- **fbid** (*str*) – Freebase ID to be translated
- **lang** (*str*) – Target language of the translation process (default is "de" for german).

Raises *MissingTranslationException* – If no translation is found.

Returns Translation of *Freebase* ID

Return type str

fetch_relation_triples_of_file (*inpath*, *outpath*, *logpath*, *lang='de'*)
Start the translation of the *Freebase* IDs into real names.

Parameters

- **inpath** (*str*) – Path to *Freebase* relation file.
- **outpath** (*str*) – Path the translated triplets should be written to.
- **logpath** (*str*) – Path to log file.
- **lang** (*str*) – Target language of the translation process (default is "de" for german).

freebase_request (*query*, *api_key*, *service_url*)
Sends a request to the *Freebase* API.

Parameters

- **query** (*list*) – MQL query as a dictionary wrapped inside a list
- **api_key** (*str*) – API key
- **service_url** (*str*) – URI to API

Returns Response as a dictionary

Return type dict

init_optparser ( )
Initialize the option parser for this script.

Returns OptionParser object

Return type OptionParser

main ( )
Main function. Start translation of relation triplets based on command line arguments.

read_credentials ( )
Reads API credentials from a file.

Returns API key and API URI as strings

Return type tuple

rl (*infile*)
Lazy function to read a line from a while and remove redundant whitespaces.

Parameters **infile** (*str*) – Path to input file.

Returns Stripped line

Return type str

src.trans_e package

src.trans_e.add_inverse_relations module

This script is used to add inverse relations to a *Freebase* relations dataset, e.g. */location/location/contains* and */location/location/containedby*.

add_inverse_relations (*relations_inpath*, *relations_outpath*, *inverse_relations*)

init_argparse ( )
Initialize all possible arguments for the argument parser.

Returns ArgumentParser object with command line arguments for this script.

Return type `argparse.ArgumentParser`

main ( )
The main function. Uses command line arguments to start the script.

read_file_with_inverse_relations (*inverse_inpath*)
Read a *Freebase* information file where inverse relations are present and separated by a simple dot.

Parameters **inverse_inpath** (*str*) – Path to *Freebase* relation file.

Returns Dictionary with a relation as a key and its inverse as value.

Return type defaultdict

src.trans_e.contains_entities module

This script analyses entities in the *Freebase* `FB14k` relations datatset and the tql wikidata dump. This is handy because the *Freebase* API is deprecated nowadays. Also, this scripted was used to create the `GER14k` dataset.

contains_entities (*entities1*, *entities2*)
Prints stats about two sets of entities.

Parameters

- **entities1** (*set*) – First set of entities.
- **entities2** (*set*) – Second set of entities.

create_new_dataset (*entities1*, *dataset*, *outpath*)
Write a new dataset only with relations which entities appear in a specific set.

Parameters

- **entities1** (*set*) – Set entities in relations have to appear in.
- **dataset** (*list*) – Original dataset (a list of tuples).
- **outpath** (*str*) – Path to new dataset.

extract_entities_from_relation_dataset (*dataset_inpath*)
Extract all entities from the *Freebase* relations file.

Parameters **dataset_inpath** (*str*) – Path to the *Freebase* file.

Returns Set of entities in the *Freebase* relations file..

Return type set

extract_entities_from_tql_file (*tql_path*)
Extract all entities from the `tql` *Wikidata Freebase* dump.

Parameters **tql_path** (*str*) – Path to `tql` file.

Returns Set of entities in the `tql` dump.

Return type set

init_argparse ( )

main ( )
Main function.

src.trans_e.differentiate_datasets module

This script analyses entities of two relation datasets (e.g. `FB15k` and `GER14k` !).

compare_entities (*set1*, *set2*)
Compares unique entities of two relation datasets. Also determines the size of their intersection.

Parameters

- **set1** (*list*) – List of relation triples as tuples from dataset 1.
- **set2** (*list*) – List of relation trilpes as tuples from dataset 2.

init_argparse ( )
Initialize all possible arguments for the argument parser.

Returns ArgumentParser object with command line arguments for this script.

Return type `argparse.ArgumentParser`

main ( )
Main function

src.trans_e.partition_data module

This script partition the data of a relation dataset like `FB15k` into a training, validation and test set so it can be used by TransE. To make sure that no relation appears in the validation or test set that didn't appear in the training set, data will be partitioned relation-wise. To partition them intuitively is still an option, though.

check_data_integrity (*data_inpath*, *remove_clones*, *outpath*)
Check whether all triplets in the data are unique.

check_set_integrity (*indir*)
Checks the integrity of given training / validation / test sets (do triples with new relations appear in the validation or test, but not in the training set?).

Parameters **indir** (*str*) – Directory of the datasets.

get_stats (*data*)
Returns some statistics about the given data, i.e. the number of unique entities, relations and their sum.

Parameters **data** (*list*) – List of relation triples as tuples.

Returns #entities, #relations, #entities + #relations.

Return type tuple

init_argparse ( )
Initialize all possible arguments for the argument parser.

Returns ArgumentParser object with command line arguments for this script.

Return type `argparse.ArgumentParser`

main ( )
Main function.

partition_data (*data*, *prts*, *outdir*, *whole=True*)

partition_relation_wise (*data*, *prts*)
Partition data into training, validation and test set.

Parameters

- **data** (*list*) – List of relation triples as tuples.
- **prts** (*tuple*) – Tuple of floats with each number corresponding to the desired percentage of data distributed to the corresponding set (% train set / % validation set / % tets set)

Returns Tuple of the three data sets as lists of relation triples as tuples.

Return type tuples

partition_whole (*data*, *prts*)

read_only_relations_into_set (*inpath*)
Only read the relation of a given relation dataset into a set.

Parameters **inpath** (*str*) – Path to relation dataset.

Returns Set of dataset relation types.

Return type set

write_data_in_file (*data*, *outfile*)
Writes relation triples into a file.

Parameters

- **data** (*list*) – List of relation triples as tuples.
- **outfile** (*str*) – Path the triples should be written to.

src.trans_e.trans_we module

This module follows a modified approach from (Bordes et al., 2013). As so, noise-contrastive learning and corrupt triples are use. But whereas in this original paper, vector representations forn entities and relations are learned in a joint manner, in this case only the continuous representations for semantic relations will be learned and word embeddings used for the entities instead.

Warning: Because we use words embedding but still the *FB15k* dataset here, we can only use data samples where we have trained word embeddings for both entities. Those are only a few, which is one reason why this approach performs badly.

convert_data (*sets_path*, *tql_inpath*, *vector_inpath*)
Re-formats relation data sets to fit the training routine in this module. Also tests the coverage of word embedding model on all entities in the datasets.

Parameters

- **sets_path** (*str*) – Directory of the datasets.
- **tql_inpath** (*str*) – Path to *Wikidata Freebase* dump in *tql* format.
- **vector_inpath** (*str*) – Path to word embedding file.

create_corrupt_triples (*grouped_pairs*, *entities*)
Creates a set of corrupted training triplets group by their shared relation.

Parameters

- **grouped_pairs** (*dict*) – Test samples as dictionary with relation as key and a list of tuples
- **two entities each as value.** (*with*) –

- **entities** (*set*) – Set of unique entities.

Returns **grouped_train** – Corrupted training samples as dictionary with a relation as key and a list of tuples with two entities each as value.

Return type dict

dump_relation_vectors (*relation_vectors*, *outpath*)
Saves relation numpy vectors.

Parameters

- **relation_vectors** (*dict*) – Dictionary with index of a relation as key and the relations vector as a
- **as value.** (*numpy.array*) –
- **outpath** (*str*) – Path the vectors should be saved to.

evaluate (*model*, *grouped_test*, *relation_vectors*, *entities*)
Evaluate the relations vector the same way as in (Bordes et al., 2013). Therefore, for every relation triple in the testset, one entity will be removed and all entities will be inserted afterwards. Also they will be ranked by their loss (ascending) and assigned a rank. The evaluation metrics are the percentage of times the right entity is in the top ten highest ranked entities and mean rank of the correct entitiy.

Parameters

- **model** (*gensim.models.Word2Vec*) – Word embeddings as gensim model.
- **grouped_test** (*dict*) – Test samples as dictionary with relation as key and a list of tuples
- **two entities each as value.** (*with*) –
- **relation_vectors** (*dict*) – Dictionary with index of a relation as key and the relations vector as a numpy.array
- **value.** (*as*) –
- **entities** (*set*) – Set of unique entities.

extract_data_from_uri (*uri*)
Extracts data from an URI.

Parameters **uri** (*str*) – URI the data should be extracted from.

Returns Extracted data.

Return type str

get_rank (*target*, *ranks*)
Get rank of a target entity within all ranked entities.

Parameters

- **target** (*str*) – Target entity which rank should be determined.
- **ranks** (*list*) – List of tuples of an entity and its rank.

Returns Rank of entity or -1 if entity is not present in ranks.

Return type int

init_argparser ( )
Initialize all possible arguments for the argument parser.

Returns ArgumentParser object with command line arguments for this script.

Return type argparse.ArgumentParser

load_relation_vectors (*inpath*)
Loads relation numpy vectors.

Parameters **inpath** (*str*) – Path the numpy vectors should be loaded from.

Returns Dictionary with index of a relation as key and the relations vector as a numpy.array as value.

Return type dict

main ( )
Main function.

prepare_training (*sets_path*, *vector_inpath*)
Prepares the training step loading word embeddings and training sets.

Parameters

- **sets_path** (*str*) – Path to training set directory.
- **vector_inpath** (*str*) – Path to word embedding file.

Returns Tuple of results with **model** (*gensim.models.Word2Vec*): Word embeddings as gensim model **/ grouped_train** (*dict*): Training samples as dictionary with relation as key and a list of tuples with two entities each as value **/ grouped_valid** (*dict*): As **grouped_train / grouped_test** (*dict*): As **grouped_train / grouped_corrupted** (*dict*): As **grouped_train / relations_types** (*dict*): Dictionary with relations as key and the amount of triples with this relation as a key **/ entities** (*set*): Set of unique entities.

Return type tuple

rank_entities (*reference*, *solution*, *model*, *entities*)
Ranks entities against a reference vector.

Parameters

- **reference** (*numpy.array*) – Reference vector.
- **solution** (*str*) – The actual solution.
- **model** (*gensim.models.Word2Vec*) – Word embeddings as gensim model.
- **entities** (*set*) – Set of unique entities.

Returns Rank of solution as integer, flag if a Hit@10 has occurred as boolean.

Return type tuples

read_freebase_data (*sets_path*)
Reads all different datasets in a directory at once.

Parameters **sets_path** (*str*) – Directory of the datasets.

Returns Tuple of datasets as lists.

Return type tuple

read_tql_file (*tql_inpath*)
Reads a *Freebase* dump by wikidata. Must be in *tql* format. Available online here (July 2016).

Parameters **tql_inpath** (*str*) – Path to *Wikidata Freebase* dump in *tql* format.

Returns Dictionary with *Freebase* code as key and the corresponding real name of an entity as value.

Return type defaultdict

test_coverage (*triples*, *model*)

Test the coverage of a dataset consisting of freebase triples on word2vec word embeddings. For every triple (h, l, t), the entities h and t are taken and used for look up in the word2vec model.

Parameters

- **triples** (*list*) – List of relation triples as tuples.
- **model** (*gensim.models.Word2Vec*) – Word embeddings as gensim model.

Returns Set of entities in the model.

Return type set

train (*model*, *grouped_train*, *grouped_corrupted*, *lossf*, *relation_types*, *epochs=1000*, *learning_rate=0.01*, *margin=1.0*)

Train the relation vectors following the example of (Bordes et al., 2013), but use word embeddings for the entity vectors instead.

Parameters

- **model** (*gensim.models.Word2Vec*) – Word embeddings as gensim model.
- **grouped_train** (*dict*) – Training samples as dictionary with relation as key and a list of tuples
- **two entities each as value.** (*with*) –
- **grouped_corrupted** (*dict*) – As grouped_train.
- **lossf** (*func*) – Loss function for training.
- **relation_types** (*dict*) – Dictionary with relations as key and the amount of triples with this relation as a key.
- **epochs** (*int*) – Number of training epochs.
- **learning_rate** (*float*) – Learning rate for training.
- **margin** (*float*) – Margin γ for training.

Returns Dictionary with index of a relation as key and the relations vector as a numpy.array as value.

Return type dict

transform_triples (*triples*, *relation_types*, *entities*)

Groups a list of relations triples by their relations and returns a suitable data structure.

Parameters

- **triples** (*list*) – List of relation triples as tuples.
- **relation_types** (*dict*) – Dictionary with relations as key and the amount of triples with this relation as a key.
- **entities** (*set*) – Set of unique entities.

Returns

Dictionary with relation as key and a list of entity tuples as value and an augmented set of unique entities.

Return type tuple

write_data (*triples*, *found_entities*, *outpath*)

Writes relation triples into a file, but only those triples where both entities are also found in a designated set.

Parameters

- **triples** (*list*) – List of relation triples as tuples.
- **found_entities** (*set*) – Set of unique entities.
- **outpath** (*str*) – Path the data should be written to.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

S

H

I

L

M

P

R

S

T

V

W