acres Documentation

Release 0.1

Stefan Schulz

May 22, 2021

CONTENTS

acres is an acronym expansion module based on word embeddings and filtering rules.

We provided here auto-generated module by module documentation only.

CHAPTER

ONE

MODULE DOCUMENTATION

1.1 acres package

Root package.

1.1.1 Subpackages

acres.evaluation package

Package containing evaluation modules.

Submodules

acres.evaluation.evaluation module

Benchmark code. It's the main entry point for comparing strategies using evaluation metrics such as precision, recall, and F1-score.

class acres.evaluation.evaluation.**Level**(*value*)

Bases: enum.Enum

Enum that holds acronym-solving levels.

TOKEN = 1

TYPE = 2

acres.evaluation.evaluation.**analyze**(*contextualized_acronym*, *true_expansions*, *strategy*, *max_tries*)

Analyze a given row of the gold standard.

Parameters

- **contextualized_acronym** (*Acronym*) –
- **true_expansions** (Set[str]) –
- **strategy** (*Strategy*) –
- **max_tries** (int) –

Return type Dict[str, bool]

Returns A dictionary with keys {'found', 'correct', and 'ignored'} pointing to boolean.

acres.evaluation.evaluation.**do_analysis**(*topics_file*, *detection_file*, *expansion_file*, *strategy*, *level*, *max_tries*, *lenient*)

Analyze a given expansion standard.

Parameters

- **topics_file** (str) –
- **detection_file** (str) –
- **expansion_file** (str) –
- **strategy** (*Strategy*) –
- **level** (*Level*) –
- **max_tries** (int) –
- **lenient** (bool) –

Return type Tuple[List[*Acronym*], List[*Acronym*], List[*Acronym*]]

Returns A tuple with lists containing correct, found, and valid contextualized acronyms

acres.evaluation.evaluation.**evaluate**(*topics*, *valid_standard*, *standard*, *strategy*, *level*, *max_tries*, *lenient*)

Analyze a gold standard with text excerpts centered on an acronym, followed by *n* valid expansions.

Parameters

- **topics** (List[*Acronym*]) –
- **valid_standard** (Set[str]) –
- **standard** (Dict[str, Dict[str, int]]) –
- **strategy** (*Strategy*) –
- **level** (*Level*) –
- **max_tries** (int) –
- **lenient** (bool) – Whether to consider partial matches (1) as a valid sense.

Return type Tuple[List[*Acronym*], List[*Acronym*], List[*Acronym*]]

Returns A tuple with lists containing correct, found, and valid contextualized acronyms

acres.evaluation.evaluation.**plot_data**(*topics_file*, *detection_file*, *expansion_file*)

Run all strategies using different ranks and lenient approaches and generate a TSV file to be used as input for the *plots.R* script.

Parameters

- **topics_file** (str) –
- **detection_file** (str) –
- **expansion_file** (str) –

Returns

acres.evaluation.evaluation.**summary**(*topics_file*, *detection_file*, *expansion_file*, *level*, *max_tries*, *lenient*)

Save a summary table in TSV format that can be used to run statistical tests (e.g. McNemar Test)

Parameters

- **topics_file** (str) –
- **detection_file** (str) –

- **expansion_file** (str) –
- **level** (*Level*) –
- **max_tries** (int) –
- **lenient** (bool) –

Returns

acres.evaluation.evaluation.**test_input**(*true_expansions*, *possible_expansions*, *max_tries=10*)
Test an acronym + context strings against the model.

Parameters

- **true_expansions** (Set[str]) –
- **possible_expansions** (List[str]) – An ordered list of possible expansions.
- **max_tries** (int) – Maxinum number of tries

Return type bool

Returns

acres.evaluation.metrics module

Helper functions to calculate evaluation metrics.

acres.evaluation.metrics.**calculate_f1**(*precision*, *recall*)
Calculates the F1-score.

Parameters

- **precision** (float) –
- **recall** (float) –

Return type float

Returns

acres.evaluation.metrics.**calculate_precision**(*total_correct*, *total_found*)
Calculate precision as the ratio of correct acronyms to the found acronyms.

Parameters

- **total_correct** (int) –
- **total_found** (int) –

Return type float

Returns

acres.evaluation.metrics.**calculate_recall**(*total_correct*, *total_acronyms*)
Calculate reall as the ratio of correct acronyms to all acronyms.

Parameters

- **total_correct** (int) –
- **total_acronyms** (int) –

Return type float

Returns

acres.fastngram package

Package containing a full in-memory implementation of n-gram matching.

Submodules

acres.fastngram.fastngram module

A faster version of n-gram matching that uses dictionaries for speed-up.

class acres.fastngram.fastngram.**CenterMap**
Bases: object

A map of center words to contexts.

add(*center*, *left_context*, *right_context*, *freq*)
Add a center n-gram with a context.

Parameters

- **center** (str) –
- **left_context** (str) –
- **right_context** (str) –
- **freq** (int) –

Return type None

Returns

contexts(*center*)
Find contexts for a given center word.

Parameters **center** –

Returns

class acres.fastngram.fastngram.**ContextMap**
Bases: object

A map of contexts to center words.

add(*center*, *left_context*, *right_context*, *freq*)
Add a center n-gram with a context.

Parameters

- **center** (str) –
- **left_context** (str) –
- **right_context** (str) –
- **freq** (int) –

Return type None

Returns

centers(*left_context*, *right_context*)
Find center n-grams that happen on a given context.

Parameters

- **left_context** –
- **right_context** –

Returns

acres.fastngram.fastngram.**baseline**(*acronym*, *left_context=''*, *right_context=''*)

A baseline method that expands only with unigrams.

Parameters

- **acronym** (str) –
- **left_context** (str) –
- **right_context** (str) –

Return type Iterator[str]

Returns

acres.fastngram.fastngram.**create_map**(*ngrams*, *model*, *partition=0*)

Create a search-optimized represenation of an ngram-list.

Parameters

- **ngrams** (Dict[str, int]) –
- **model** (Union[*ContextMap*, *CenterMap*]) –
- **partition** (int) –

Return type Union[*ContextMap*, *CenterMap*]

Returns

acres.fastngram.fastngram.**fastngram**(*acronym*, *left_context=''*, *right_context=''*, *min_freq=2*, *max_rank=100000*)

Find an unlimited set of expansion candidates for an acronym given its left and right context. Note that no filtering is done here, except from the acronym initial partioning.

Parameters

- **acronym** (str) –
- **left_context** (str) –
- **right_context** (str) –
- **min_freq** (int) –
- **max_rank** (int) –

Return type Iterator[str]

Returns

acres.fastngram.fastngram.**fasttype**(*acronym*, *left_context=''*, *right_context=''*, *min_freq=2*, *max_rank=100000*)

Find an unlimited set of expansion candidates given the training contexts of the acronym. Note that no filtering is done here, except from the acronym initial partioning.

Parameters

- **acronym** (str) –
- **left_context** (str) – Not used.
- **right_context** (str) – Not used.

- **min_freq** (int) –
- **max_rank** (int) –

Return type Iterator[str]

Returns

acres.model package

Package containing domain models (from the MVC design pattern).

Submodules

acres.model.detection_standard module

Model class that represents a detection standard. A detection standard works like a allow/block list to filter out inputs from the topic list that are not proper acronyms (e.g. *BEFUND*, *III*). Such inputs are then not considered for evaluation purposes.

It is designed as an append-only list (i.e., entries do not need to be updated with variable inputs).

acres.model.detection_standard.**filter_valid**(*standard*)
Filter out invalid entries from a gold standard. Invalid entries are not proper acronyms or repeated types.

Parameters **standard** (Dict[str, bool]) –

Return type Set[str]

Returns

acres.model.detection_standard.**parse**(*filename*)
Parses a .tsv-formatted detection standard into a dictionary.

Parameters **filename** (str) –

Return type Dict[str, bool]

Returns

acres.model.detection_standard.**parse_valid**(*filename*)
Wrapper method for both *parse* and *filter_valid*.

Parameters **filename** (str) –

Return type Set[str]

Returns

acres.model.detection_standard.**update**(*previous*, *acronyms*)
Update a previous detection standard with new acronyms from a topic list, preserving order.

Parameters

- **previous** (Dict[str, bool]) –
- **acronyms** (List[*Acronym*]) –

Return type Dict[str, bool]

Returns

acres.model.detection_standard.**write**(*filename*, *standard*)
Write a detection standard into a file.

Parameters

- **filename** (str) –
- **standard** (Dict[str, bool]) –

Return type None

Returns

acres.model.expansion_standard module

Model class that represents an expansion standard. An expansion standard is the main reference standard containing acronyms-expansion pairs and their evaluation following the TREC standard (2/1/0).

It is designed as an append-only list (i.e., entries do not need to be updated with variable inputs).

acres.model.expansion_standard.**parse**(*filename*)
Parse a TSV-separated expansion standard into a dictionary.

Parameters **filename** (str) –

Return type Dict[str, Dict[str, int]]

Returns A dictionary with acronyms pointing to expansions and an assessment value.

acres.model.expansion_standard.**write**(*filename*, *previous*, *valid*, *topics*)
Write results in the TREC format, one candidate expansion per line.

Parameters

- **filename** (str) –
- **previous** (Dict[str, Dict[str, int]]) – A dictionary of acronyms mapped to their senses and assesments (if any).
- **valid** (Set[str]) – A set of valid acronyms, normally parsed from a detection standard.
- **topics** (List[*Acronym*]) – A topic list.

Return type None

Returns

acres.model.ngrams module

Module to handle n-gram lists.

class acres.model.ngrams.**FilteredNGramStat**(*ngram_size*)
Bases: object

Filtered NGramStat generator

This generator generates ngrams of a given size out of a ngramstat.txt file, while respecting each ngram frequency.

@todo ngramstat itself should be a generator

PRINT_INTERVAL = 1000000

TOKEN_SEPARATOR = ' '

acres.model.ngrams.**filter_acronym_contexts**(*ngrams*)
Filter an iterable of tokens by the ones containing an acronym in the middle and convert them to Acronym tuples.

Parameters **ngrams** (Iterator[List[str]]) –

Return type Iterator[*Acronym*]

Returns

acres.model.topic_list module

Model class that represents a topic list. A topic list is used as main input (a la TREC) and thus can control which acronyms (together with their contexts) are to be considered for evaluation. A topic list can be used, e.g., to quickly switch between different evaluation scenarios such as acronyms collected from either the training or test dataset.

acres.model.topic_list.**create**(*filename*, *chance*, *ngram_size=7*)
Create a topic list out of random n-grams with a given chance and size.

Parameters

- **filename** (str) –
- **chance** (float) –
- **ngram_size** (int) –

Returns

acres.model.topic_list.**parse**(*filename*)
Parses a TSV-formatted topic list into a list of acronyms (with context).

Parameters **filename** (str) –

Return type List[*Acronym*]

Returns

acres.model.topic_list.**unique_types**(*topics*)
Extract types from a topic list.

Parameters **topics** (List[*Acronym*]) –

Return type Set[str]

Returns

acres.preprocess package

Package containing modules for pre-processing the corpus and a resource factory to easily access pre-processed files.

Submodules

acres.preprocess.dumps module

Module to process the corpus training data and create data structures for speed-up retrieval.

acres.preprocess.dumps.**create_corpus_ngramstat_dump**(*corpus_path*, *min_freq*, *min_length=1*, *max_length=7*)
Takes a corpus consisting of text files in a single directory Substitutes digits and line breaks It requires that all documents are in UTF-8 text. It can perform substitutions of digits.

Parameters

- **corpus_path** (str) –
- **min_freq** (int) –
- **min_length** (int) –
- **max_length** (int) –

Return type Dict[str, int]

Returns

acres.preprocess.dumps.**create_indexed_ngrams**(*ngrams*)
Create an indexed version of a ngram list. This basically adds an unique identifier to every (str, int) tuple.

Parameters **ngrams** (Dict[str, int]) –

Return type Dict[int, Tuple[int, str]]

Returns

acres.preprocess.resource_factory module

Resource factory. This module provides methods for lazily loading resources.

acres.preprocess.resource_factory.**get_center_map**(*partition=0*)
Lazy load the fast n-gram center map model.

Return type *CenterMap*

Returns

acres.preprocess.resource_factory.**get_context_map**(*partition=0*)
Lazy load the fast n-gram context map model.

Return type *ContextMap*

Returns

acres.preprocess.resource_factory.**get_dictionary**()
Lazy load the sense inventory.

Return type Dict[str, List[str]]

Returns

acres.preprocess.resource_factory.**get_ngramstat**()
Lazy load an indexed representation of ngrams.

Loading order is as follows: 1. Variable; 2. Pickle file; 3. Generation.

Return type Dict[int, Tuple[int, str]]

Returns A dictionary of identifiers mapped to ngrams. Ngrams are tuples with the frequency and the corresponding ngram.

acres.preprocess.resource_factory.**get_nn_model**(*ngram_size=3*, *min_count=1*, *net_size=100*, *alpha=0.025*, *sg=0*, *hs=0*, *negative=5*)
Lazy load a word2vec model.

Parameters

- **ngram_size** (int) –

- **min_count** (int) –
- **net_size** (int) –
- **alpha** (float) –
- **sg** (int) –
- **hs** (int) –
- **negative** (int) –

Return type Word2Vec

Returns

acres.preprocess.resource_factory.**get_word_ngrams**()
Lazy load a not-indexed representation of ngrams.

Loading order is as follows: 1. Variable; 2. Pickle file; 3. Generation.

Return type Dict[str, int]

Returns

acres.preprocess.resource_factory.**reset**()
Resets global variables to force model recreation.

Return type None

Returns

acres.preprocess.resource_factory.**warmup_cache**()
Warms up the cache of pickle and txt files by calling all the methods.

Return type None

Returns

acres.preprocess.resource_factory.**write_txt**(*resource*, *filename*)
Writes a tab-separated represenation of a dictionary into a file specified by filename.

Parameters

- **resource** (Dict[str, int]) –
- **filename** (str) –

Return type int

Returns

acres.rater package

Package with rating modules. Rating modules are used to filter out candidate expansions provided by expansion strategies.

Submodules

acres.rater.expansion module

Rating submodule for expansion (acronym + full form) checks.

acres.rater.expansion.**is_expansion_valid**(*acro*, *full*)
Check whether an expansion is valid for a given acronym.

Parameters

- **acro** (str) –
- **full** (str) –

Return type bool

Returns

acres.rater.full module

Rating submodule for full form checks.

acres.rater.full.**is_full_valid**(*full*)
Check whether the full form is valid.

Parameters **full** (str) –

Return type bool

Returns

acres.rater.rater module

Rating main module.

acres.rater.rater.**get_acronym_score**(*acro*, *full*)
Scores acronym/resolution pairs according to a series of well-formedness criteria.

This scoring function should be used only for cleaned and normalized full forms.

For forms that may contain acronym-definition pairs, see *get_acronym_definition_pair_score*. For forms that should be checked for variants, see *get_acronym_score_variants*.

TODO Consider again morphosaurus checks.

TODO Full form should not be an acronym itself.

Parameters

- **acro** (str) – Acronym to be expanded.
- **full** (str) – Long form to be checked whether it qualifies as an acronym expansion.

Return type float

Returns score that rates the likelihood that the full form is a valid expansion of the acronym.

acres.resolution package

Package with a facade to the several expansion strategies.

Submodules

acres.resolution.resolver module

Facade to the several expansion strategies.

class acres.resolution.resolver.**Strategy**(*value*)
Bases: enum.IntEnum

Enum that holds acronym-solving strategies.

BASELINE = 5

DICTIONARY = 3

FASTNGRAM = 4

FASTTYPE = 6

WORD2VEC = 2

acres.resolution.resolver.**filtered_resolve**(*acronym*, *left_context*, *right_context*, *strategy*)
Resolve a given acronym + context using the provided Strategy and filter out invalid expansions.

Parameters

- **acronym** (str) –
- **left_context** (str) –
- **right_context** (str) –
- **strategy** (*Strategy*) –

Return type Iterator[str]

Returns

acres.resolution.resolver.**resolve**(*acronym*, *left_context*, *right_context*, *strategy*)
Resolve a given acronym + context using the provided Strategy.

Parameters

- **acronym** (str) –
- **left_context** (str) –
- **right_context** (str) –
- **strategy** (*Strategy*) –

Return type List[str]

Returns

acres.stats package

Package with modules to collect statistics from the gold-standard (*senses*), the training corpus (*stats*), and a fixed sense inventory (*dictionary*).

Submodules

acres.stats.dictionary module

Module to collect metrics from a sense inventory. This module can be used to debug the sense inventory e.g. by detecting extreme expansions. It can also be used to debug methods that relies on real data.

acres.stats.dictionary.**analyze_file**(*filename*)
Analyzes a given dictionary file for extreme cases.

Parameters **filename** (str) –

Return type None

Returns

acres.stats.dictionary.**edit_distance_generated_acro**(*acro*, *full*)
Calculates the edit distance between the original acronym and the generated acronym out of the full form.

Parameters

- **acro** (str) –
- **full** (str) –

Return type Optional[Tuple]

Returns

acres.stats.dictionary.**expand**(*acronym*, *left_context=''*, *right_context=''*)

Parameters

- **acronym** (str) –
- **left_context** (str) –
- **right_context** (str) –

Return type List[str]

Returns

acres.stats.dictionary.**parse**(*filename*)
Parse a tab-separated sense inventory as a Python dictionary.

Parameters **filename** (str) –

Return type Dict[str, List[str]]

Returns

acres.stats.dictionary.**ratio_acro_words**(*acro*, *full*)
Calculates the ratio of acronym lenfth to the number of words in the full form.

Parameters

- **acro** (str) –

- **full** (str) –

Return type Tuple

Returns

acres.stats.dictionary.**show_extremes**(*txt*, *lst*, *lowest_n=10*, *highest_n=10*)

Parameters

- **txt** (str) –
- **lst** (List) –
- **lowest_n** (int) –
- **highest_n** (int) –

Return type None

Returns

acres.stats.senses module

Module to estimate acronym ambiguity. It can be used to collect common acronym statistics, such as senses/acronym.

acres.stats.senses.**bucketize**(*acronyms*)
Reduce: calculate the number of different acronyms for each degree of ambiguity.

Parameters **acronyms** (Dict[str, Set[str]]) –

Return type Dict[int, int]

Returns

acres.stats.senses.**get_sense_buckets**(*filename*)
Parses a reference standard and get a map of senses per acronym.

Parameters **filename** (str) –

Return type Dict[str, Set[str]]

Returns

acres.stats.senses.**map_senses_acronym**(*standard*, *lenient=False*)
Map: collect senses for each acronym.

Parameters

- **standard** (Dict[str, Dict[str, int]]) –
- **lenient** (bool) – Whether to consider partial matches (1) as a valid sense.

Return type Dict[str, Set[str]]

Returns

acres.stats.senses.**print_ambiguous**(*filename*)
Print ambiguous acronyms, the ones with more than one sense according to the reference standard.

Parameters **filename** (str) –

Return type None

Returns

acres.stats.senses.**print_senses**(*filename*)
Print the distribution of senses per acronym.

Parameters **filename** (str) –

Return type None

Returns

acres.stats.senses.**print_undefined**(*filename*)
Print undefined acronyms, the ones with no valid sense according to the reference standard.

Parameters **filename** (str) –

Return type None

Returns

acres.stats.stats module

Module for calculating corpus statistics. It is used to measure the training/test dataset according to, e.g., number of tokens.

class acres.stats.stats.**Stats**
Bases: object

Class that generates and holds stats about a given text.

calc_stats(*text*)
Calculates statistics for a given text string and sets the results as variables.

Parameters **text** (str) –

Return type None

Returns

static count_acronyms(*text*)
Count the number of acronyms in a string.

Acronyms are as defined by the *acronym.is_acronym()* function.

Parameters **text** (str) –

Return type int

Returns

static count_acronyms_types(*text*)
Count the number of unique acronyms in a string.

Acronyms are as defined by the *acronym.is_acronym()* function.

Parameters **text** (str) –

Return type int

Returns

static count_chars(*text*)
Count the number of non-whitespace chars in a string.

Parameters **text** (str) –

Return type int

Returns

static count_sentences(*text*)
Count the number of sentences in a string.

Sentences are any string separated by *line_separator*.

Parameters **text** (str) –

Return type int

Returns

static count_tokens(*text*)
Count the number of all tokens in a string.

Parameters **text** (str) –

Return type int

Returns

static count_types(*text*)
Count the number of unique tokens (types) in a string.

Parameters **text** (str) –

Return type int

Returns

source_line_separator = '\n'

acres.stats.stats.**get_stats**(*corpus_path*)
Generates all statistics from a given corpus directory.

Parameters **corpus_path** (str) –

Return type List[*Stats*]

Returns A list of statistics objects, one for each file found in the corpus dir, plus an extra one for the full corpus.

acres.stats.stats.**print_stats**()
Generates and print statistics from the default corpus set in config.

Return type None

Returns None

acres.util package

Package with general utilities modules.

Submodules

acres.util.acronym module

Utility functions related to acronyms.

class acres.util.acronym.**Acronym**(*acronym*, *left_context*, *right_context*)
Bases: tuple

property **acronym**
Alias for field number 0

property **left_context**
Alias for field number 1

property **right_context**
Alias for field number 2

acres.util.acronym.**create_german_acronym**(*full*)
Creates an acronym out of a given multi-word expression.

@todo Use is_stopword?

Parameters **full** (str) – A full form containing whitespaces.

Return type str

Returns

acres.util.acronym.**is_acronym**(*str_probe*, *max_length=7*)
Identifies Acronyms, restricted by absolute length XXX look for "authoritative" definitions for acronyms

Parameters

- **str_probe** (str) –
- **max_length** (int) –

Return type bool

Returns

acres.util.acronym.**trim_plural**(*acronym*)
Trim the german plural form out of an acronym.

@todo rewrite as regex

Parameters **acronym** (str) –

Return type str

Returns

acres.util.functions module

Module with general functions.

acres.util.functions.**create_ngram_statistics**(*input_string*, *n_min*, *n_max*)
Creates a dictionary that counts each nGram in an input string. Delimiters are spaces.

Example: bigrams and trigrams nMin = 2 , nMax = 3 PROBE: # print(WordNgramStat('a ab aa a a a ba ddd', 1, 4))

Parameters

- **input_string** (str) –
- **n_min** (int) –
- **n_max** (int) –

Return type Dict[str, int]

Returns

acres.util.functions.**import_conf**(*key*)

Parameters **key** (str) –

Return type Optional[str]

Returns

acres.util.functions.**is_stopword**(*str_in*)
Tests whether word is stopword, according to list.

For German, source http://snowball.tartarus.org/algorithms/german/stop.txt

Parameters **str_in** (str) –

Return type bool

Returns

acres.util.functions.**partition**(*word*, *partitions*)
Find a bucket for a given word.

Parameters

- **word** (str) –
- **partitions** (int) –

Return type int

Returns

acres.util.functions.**robust_text_import_from_dir**(*path*)
Read the content of valid text files from a path into a list of strings.

Parameters **path** (str) – The path to look for documents.

Return type List[str]

Returns A list of strings containing the content of each valid file.

acres.util.functions.**sample**(*iterable*, *chance*)
Randomly sample items from an iterable with a given chance.

Parameters

- **iterable** (Iterable) –
- **chance** (float) –

Return type Iterable

Returns

acres.util.text module

Utility functions related to text processing.

acres.util.text.**clean**(*text*, *preserve_linebreaks=False*)
Clean a given text to preserve only alphabetic characters, spaces, and, optionally, line breaks.

Parameters

- **text** (str) –
- **preserve_linebreaks** (bool) –

Return type str

Returns

acres.util.text.**clean_whitespaces**(*whitespaced*)
Clean up an input string of repeating and trailing whitespaces.

Parameters **whitespaced** (str) –

Return type str

Returns

acres.util.text.**clear_digits**(*str_in*, *substitute_char*)
Substitutes all digits by a character (or string)

Example: ClearDigits("Vitamin B12", "°"):

TODO rewrite as regex

Parameters

- **str_in** (str) –
- **substitute_char** (str) –

Return type str

acres.util.text.**reduce_repeated_chars**(*str_in*, *char*, *remaining_chars*)

Parameters

- **str_in** (str) – text to be cleaned
- **char** (str) – character that should not occur more than remaining_chars times in sequence
- **remaining_chars** (int) – remaining_chars

Return type str

Returns

acres.util.text.**remove_duplicated_whitespaces**(*whitespaced*)
Clean up an input string out of any number of repeated whitespaces.

Parameters **whitespaced** (str) –

Return type str

Returns

acres.word2vec package

Package grouping modules related to the word2vec expansion strategy.

Submodules

acres.word2vec.test module

Module to apply/*test* a given word2vec model.

acres.word2vec.test.**find_candidates**(*acronym*, *left_context=''*, *right_context=''*, *min_distance=0.0*, *max_rank=500*)

Similar to robust_find_embeddings, this finds possible expansions of a given acronym.

Parameters

- **acronym** (str) –
- **left_context** (str) –
- **right_context** (str) –
- **min_distance** (float) –
- **max_rank** (int) –

Return type Iterator[str]

Returns

acres.word2vec.train module

Trainer for word2vec embeddings based on an idea originally proposed by Johannes Hellrich (https://github.com/JULIELab/hellrich_dh2016).

acres.word2vec.train.**train**(*ngram_size=6*, *min_count=1*, *net_size=100*, *alpha=0.025*, *sg=1*, *hs=0*, *negative=5*)

Lazy load a word2vec model.

Parameters

- **ngram_size** (int) –
- **min_count** (int) –
- **net_size** (int) –
- **alpha** (float) –
- **sg** (int) –
- **hs** (int) –
- **negative** (int) –

Return type `Word2Vec`

Returns

1.1.2 Submodules

1.1.3 acres.constants module

Module with global constants.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

A

B

C

D

E

F

G

I

L

M

P

R

S

T

U

W