Top2Vec

Release 1.0.27

Apr 03, 2022

User Guide / Tutorial:

Updates:

- New pre-trained transformer models available
- Ability to use any embedding model by passing callable to `embedding_model`
- Document chunking options for long documents
- Phrases in topics by setting `ngram_vocab=True`

CHAPTER 1

Top2Vec

Top2Vec is an algorithm for **topic modeling** and **semantic search**. It automatically detects topics present in text and generates jointly embedded topic, document and word vectors. Once you train the Top2Vec model you can:

- Get number of detected topics.
- Get topics.
- Get topic sizes.
- Get hierarchichal topics.
- Search topics by keywords.
- Search documents by topic.
- Search documents by keywords.
- Find similar words.
- Find similar documents.
- Expose model with RESTful-Top2Vec

See the paper for more details on how it works.

1.1 Benefits

1. Automatically finds number of topics.
2. No stop word lists required.
3. No need for stemming/lemmatization.
4. Works on short text.
5. Creates jointly embedded topic, document, and word vectors.
6. Has search functions built in.

1.2 How does it work?

The assumption the algorithm makes is that many semantically similar documents are indicative of an underlying topic. The first step is to create a joint embedding of document and word vectors. Once documents and words are embedded in a vector space the goal of the algorithm is to find dense clusters of documents, then identify which words attracted those documents together. Each dense area is a topic and the words that attracted the documents to the dense area are the topic words.

1.2.1 The Algorithm:

1. Create jointly embedded document and word vectors using Doc2Vec or Universal Sentence Encoder or BERT Sentence Transformer.

Documents will be placed close to other similar documents and close to the most distinguishing words.

2. Create lower dimensional embedding of document vectors using UMAP.

Document vectors in high dimensional space are very sparse, dimension reduction helps for finding dense areas. Each point is a document vector.

3. Find dense areas of documents using HDBSCAN.

The colored areas are the dense areas of documents. Red points are outliers that do not belong to a specific cluster.

4. For each dense area calculate the centroid of document vectors in original dimension, this is the topic vector.

The red points are outlier documents and do not get used for calculating the topic vector. The purple points are the document vectors that belong to a dense area, from which the topic vector is calculated.

5. Find n-closest word vectors to the resulting topic vector.

The closest word vectors in order of proximity become the topic words.

1.3 Installation

The easy way to install Top2Vec is:

```
pip install top2vec
```

To install pre-trained universal sentence encoder options:

```
pip install top2vec[sentence_encoders]
```

To install pre-trained BERT sentence transformer options:

```
pip install top2vec[sentence_transformers]
```

To install indexing options:

```
pip install top2vec[indexing]
```

1.4 Usage

```
from top2vec import Top2Vec

model = Top2Vec(documents)
```

Important parameters:

- `documents`: Input corpus, should be a list of strings.
- `speed`: This parameter will determine how fast the model takes to train. The 'fast-learn' option is the fastest and will generate the lowest quality vectors. The 'learn' option will learn better quality vectors but take a longer time to train. The 'deep-learn' option will learn the best quality vectors but will take significant time to train.
- `workers`: The amount of worker threads to be used in training the model. Larger amount will lead to faster training.

  Trained models can be saved and loaded.

```
model.save("filename")
model = Top2Vec.load("filename")
```

For more information view the API guide.

1.5 Pretrained Embedding Models

Doc2Vec will be used by default to generate the joint word and document embeddings. However there are also pretrained `embedding_model` options for generating joint word and document embeddings:

- `universal-sentence-encoder`
- `universal-sentence-encoder-multilingual`
- `distiluse-base-multilingual-cased`

```
from top2vec import Top2Vec

model = Top2Vec(documents, embedding_model='universal-sentence-encoder')
```

For large data sets and data sets with very unique vocabulary doc2vec could produce better results. This will train a doc2vec model from scratch. This method is language agnostic. However multiple languages will not be aligned.

Using the universal sentence encoder options will be much faster since those are pre-trained and efficient models. The universal sentence encoder options are suggested for smaller data sets. They are also good options for large data sets that are in English or in languages covered by the multilingual model. It is also suggested for data sets that are multilingual.

The distiluse-base-multilingual-cased pre-trained sentence transformer is suggested for multilingual datasets and languages that are not covered by the multilingual universal sentence encoder. The transformer is significantly slower than the universal sentence encoder options.

More information on universal-sentence-encoder, universal-sentence-encoder-multilingual, and distiluse-base-multilingual-cased.

1.6 Citation

If you would like to cite Top2Vec in your work this is the current reference:

```
@article{angelov2020top2vec,
      title={Top2Vec: Distributed Representations of Topics},
      author={Dimo Angelov},
      year={2020},
      eprint={2008.09470},
      archivePrefix={arXiv},
      primaryClass={cs.CL}
}
```

1.7 Example

1.7.1 Train Model

Train a Top2Vec model on the 20newsgroups dataset.

```
from top2vec import Top2Vec
from sklearn.datasets import fetch_20newsgroups

newsgroups = fetch_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'))

model = Top2Vec(documents=newsgroups.data, speed="learn", workers=8)
```

1.7.2 Get Number of Topics

This will return the number of topics that Top2Vec has found in the data.

```
>>> model.get_num_topics()
77
```

1.7.3 Get Topic Sizes

This will return the number of documents most similar to each topic. Topics are in decreasing order of size.

```
topic_sizes, topic_nums = model.get_topic_sizes()
```

Returns:

- `topic_sizes`: The number of documents most similar to each topic.

- `topic_nums`: The unique index of every topic will be returned.

1.7.4 Get Topics

This will return the topics in decreasing size.

```
topic_words, word_scores, topic_nums = model.get_topics(77)
```

Returns:

- `topic_words`: For each topic the top 50 words are returned, in order of semantic similarity to topic.
- `word_scores`: For each topic the cosine similarity scores of the top 50 words to the topic are returned.
- `topic_nums`: The unique index of every topic will be returned.

1.7.5 Search Topics

We are going to search for topics most similar to **medicine**.

```

topic_words, word_scores, topic_scores, topic_nums = model.search_topics(keywords=[
↪"medicine"], num_topics=5)
```

Returns:

- `topic_words`: For each topic the top 50 words are returned, in order of semantic similarity to topic.
- `word_scores`: For each topic the cosine similarity scores of the top 50 words to the topic are returned.
- `topic_scores`: For each topic the cosine similarity to the search keywords will be returned.
- `topic_nums`: The unique index of every topic will be returned.

```

>>> topic_nums
[21, 29, 9, 61, 48]

>>> topic_scores
[0.4468, 0.381, 0.2779, 0.2566, 0.2515]
```

Topic 21 was the most similar topic to “medicine” with a cosine similarity of 0.4468. (Values can be from least similar 0, to most similar 1)

1.7.6 Generate Word Clouds

Using a topic number you can generate a word cloud. We are going to generate word clouds for the top 5 most similar topics to our **medicine** topic search from above.

```
topic_words, word_scores, topic_scores, topic_nums = model.search_topics(keywords=[
↪"medicine"], num_topics=5)
for topic in topic_nums:
    model.generate_topic_wordcloud(topic)
```

1.7.7 Search Documents by Topic

We are going to search by **topic 48**, a topic that appears to be about **science**.

```
documents, document_scores, document_ids = model.search_documents_by_topic(topic_
↪num=48, num_docs=5)
```

Returns:

- `documents`: The documents in a list, the most similar are first.
- `doc_scores`: Semantic similarity of document to topic. The cosine similarity of the document and topic vector.
- `doc_ids`: Unique ids of documents. If ids were not given, the index of document in the original corpus.

For each of the returned documents we are going to print its content, score and document number.

```
documents, document_scores, document_ids = model.search_documents_by_topic(topic_
↪num=48, num_docs=5)
for doc, score, doc_id in zip(documents, document_scores, document_ids):
    print(f"Document: {doc_id}, Score: {score}")
    print("-----------")
    print(doc)
    print("-----------")
    print()
```

```
Document: 15227, Score: 0.6322
-----------
  Evolution is both fact and theory.  The THEORY of evolution represents the
scientific attempt to explain the FACT of evolution.  The theory of evolution
does not provide facts; it explains facts.  It can be safely assumed that ALL
scientific theories neither provide nor become facts but rather EXPLAIN facts.
I recommend that you do some appropriate reading in general science.  A good
starting point with regard to evolution for the layman would be "Evolution as
Fact and Theory" in "Hen's Teeth and Horse's Toes" [pp 253-262] by Stephen Jay
Gould.  There is a great deal of other useful information in this publication.
-----------

Document: 14515, Score: 0.6186
-----------
Just what are these "scientific facts"?  I have never heard of such a thing.
Science never proves or disproves any theory - history does.

-Tim
-----------

Document: 9433, Score: 0.5997
-----------
The same way that any theory is proven false.  You examine the predicitions
that the theory makes, and try to observe them.  If you don't, or if you
observe things that the theory predicts wouldn't happen, then you have some
evidence against the theory.  If the theory can't be modified to
incorporate the new observations, then you say that it is false.

For example, people used to believe that the earth had been created
10,000 years ago.  But, as evidence showed that predictions from this
theory were not true, it was abandoned.
-----------
```

(continues on next page)

(continued from previous page)

```
Document: 11917, Score: 0.5845
-----------
The point about its being real or not is that one does not waste time with
what reality might be when one wants predictions. The questions if the
atoms are there or if something else is there making measurements indicate
atoms is not necessary in such a system.

And one does not have to write a new theory of existence everytime new
models are used in Physics.
-----------

...
```

1.7.8 Semantic Search Documents by Keywords

Search documents for content semantically similar to **cryptography** and **privacy**.

```
documents, document_scores, document_ids = model.search_documents_by_
↪keywords(keywords=["cryptography", "privacy"], num_docs=5)
for doc, score, doc_id in zip(documents, document_scores, document_ids):
    print(f"Document: {doc_id}, Score: {score}")
    print("-----------")
    print(doc)
    print("-----------")
    print()
```

```
Document: 16837, Score: 0.6112
-----------
...
Email and account privacy, anonymity, file encryption,  academic
computer policies, relevant legislation and references, EFF, and
other privacy and rights issues associated with use of the Internet
and global networks in general.
...

Document: 16254, Score: 0.5722
-----------
...
The President today announced a new initiative that will bring
the Federal Government together with industry in a voluntary
program to improve the security and privacy of telephone
communications while meeting the legitimate needs of law
enforcement.
...
-----------
...
```

1.7.9 Similar Keywords

Search for similar words to **space**.

```
words, word_scores = model.similar_words(keywords=["space"], keywords_neg=[], num_
↪words=20)
for word, score in zip(words, word_scores):
    print(f"{word} {score}")
```

```
space 1.0
nasa 0.6589
shuttle 0.5976
exploration 0.5448
planetary 0.5391
missions 0.5069
launch 0.4941
telescope 0.4821
astro 0.4696
jsc 0.4549
ames 0.4515
satellite 0.446
station 0.4445
orbital 0.4438
solar 0.4386
astronomy 0.4378
observatory 0.4355
facility 0.4325
propulsion 0.4251
aerospace 0.4226
```

CHAPTER 2

Top2Vec API Guide

class top2vec.Top2Vec.**Top2Vec**(*documents*, *min_count=50*, *ngram_vocab=False*, *ngram_vocab_args=None*, *embedding_model='doc2vec'*, *embedding_model_path=None*, *embedding_batch_size=32*, *split_documents=False*, *document_chunker='sequential'*, *chunk_length=100*, *max_num_chunks=None*, *chunk_overlap_ratio=0.5*, *chunk_len_coverage_ratio=1.0*, *sentencizer=None*, *speed='learn'*, *use_corpus_file=False*, *document_ids=None*, *keep_documents=True*, *workers=None*, *tokenizer=None*, *use_embedding_model_tokenizer=False*, *umap_args=None*, *hdbscan_args=None*, *verbose=True*)

Creates jointly embedded topic, document and word vectors.

Parameters

- **documents** (*List of str*) – Input corpus, should be a list of strings.
- **min_count** (*int (Optional, default 50)*) – Ignores all words with total frequency lower than this. For smaller corpora a smaller min_count will be necessary.
- **ngram_vocab** (*bool (Optional, default False)*) – Add phrases to topic descriptions.

  Uses gensim phrases to find common phrases in the corpus and adds them to the vocabulary.

  For more information visit: https://radimrehurek.com/gensim/models/phrases.html
- **ngram_vocab_args** (*dict (Optional, default None)*) – Pass custom arguments to gensim phrases.

  For more information visit: https://radimrehurek.com/gensim/models/phrases.html
- **embedding_model** (*string or callable*) – This will determine which model is used to generate the document and word embeddings. The valid string options are:
  - doc2vec
  - universal-sentence-encoder

  - universal-sentence-encoder-large
  - universal-sentence-encoder-multilingual
  - universal-sentence-encoder-multilingual-large
  - distiluse-base-multilingual-cased
  - all-MiniLM-L6-v2
  - paraphrase-multilingual-MiniLM-L12-v2

  For large data sets and data sets with very unique vocabulary doc2vec could produce better results. This will train a doc2vec model from scratch. This method is language agnostic. However multiple languages will not be aligned.

  Using the universal sentence encoder options will be much faster since those are pre-trained and efficient models. The universal sentence encoder options are suggested for smaller data sets. They are also good options for large data sets that are in English or in languages covered by the multilingual model. It is also suggested for data sets that are multilingual.

  For more information on universal-sentence-encoder options visit: https://tfhub.dev/google/collections/universal-sentence-encoder/1

  The SBERT pre-trained sentence transformer options are distiluse-base-multilingual-cased, paraphrase-multilingual-MiniLM-L12-v2, and all-MiniLM-L6-v2.

  The distiluse-base-multilingual-cased and paraphrase-multilingual-MiniLM-L12-v2 are suggested for multilingual datasets and languages that are not covered by the multilingual universal sentence encoder. The transformer is significantly slower than the universal sentence encoder options(except for the large options).

  For more information on SBERT options visit: https://www.sbert.net/docs/pretrained_models.html

  If passing a callable embedding_model note that it will not be saved when saving a top2vec model. After loading such a saved top2vec model the set_embedding_model method will need to be called and the same embedding_model callable used during training must be passed to it.

- **embedding_model_path** (*string (Optional)*) – Pre-trained embedding models will be downloaded automatically by default. However they can also be uploaded from a file that is in the location of embedding_model_path.

  Warning: the model at embedding_model_path must match the embedding_model parameter type.

- **embedding_batch_size** (*int (default=32)*) – Batch size for documents being embedded.

- **split_documents** (*bool (default False)*) – If set to True, documents will be split into parts before embedding. After embedding the multiple document part embeddings will be averaged to create a single embedding per document. This is useful when documents are very large or when the embedding model has a token limit.

  Document chunking or a senticizer can be used for document splitting.

- **document_chunker** (*string or callable (default 'sequential')*) – This will break the document into chunks. The valid string options are:

  - sequential
  - random

The sequential chunker will split the document into chunks of specified length and ratio of overlap. This is the recommended method.

The random chunking option will take random chunks of specified length from the document. These can overlap and should be thought of as sampling chunks with replacement from the document.

If a callable is passed it must take as input a list of tokens of a document and return a list of strings representing the resulting document chunks.

Only one of document_chunker or sentincizer should be used.

- **chunk_length** (*int (default 100)*) – The number of tokens per document chunk if using the document chunker string options.
- **max_num_chunks** (*int (Optional)*) – The maximum number of chunks generated per document if using the document chunker string options.
- **chunk_overlap_ratio** (*float (default 0.5)*) – Only applies to the 'sequential' document chunker.

  Fraction of overlapping tokens between sequential chunks. A value of 0 will result i no overlap, where as 0.5 will overlap half of the previous chunk.
- **chunk_len_coverage_ratio** (*float (default 1.0)*) – Only applies to the 'random' document chunker option.

  Proportion of token length that will be covered by chunks. Default value of 1.0 means chunk lengths will add up to number of tokens of the document. This does not mean all tokens will be covered since chunks can be overlapping.
- **sentencizer** (*callable (Optional)*) – A sentincizer callable can be passed. The input should be a string representing the document and the output should be a list of strings representing the document sentence chunks.

  Only one of document_chunker or sentincizer should be used.
- **speed** (*string (Optional, default 'learn')*) – This parameter is only used when using doc2vec as embedding_model.

  It will determine how fast the model takes to train. The fast-learn option is the fastest and will generate the lowest quality vectors. The learn option will learn better quality vectors but take a longer time to train. The deep-learn option will learn the best quality vectors but will take significant time to train. The valid string speed options are:
  - fast-learn
  - learn
  - deep-learn
- **use_corpus_file** (*bool (Optional, default False)*) – This parameter is only used when using doc2vec as embedding_model.

  Setting use_corpus_file to True can sometimes provide speedup for large datasets when multiple worker threads are available. Documents are still passed to the model as a list of str, the model will create a temporary corpus file for training.
- **document_ids** (*List of str, int (Optional)*) – A unique value per document that will be used for referring to documents in search results. If ids are not given to the model, the index of each document in the original corpus will become the id.
- **keep_documents** (*bool (Optional, default True)*) – If set to False documents will only be used for training and not saved as part of the model. This will reduce

model size. When using search functions only document ids will be returned, not the actual documents.

- **workers** (*int (Optional)*) – The amount of worker threads to be used in training the model. Larger amount will lead to faster training.
- **tokenizer** (*callable (Optional, default None)*) – Override the default tokenization method. If None then gensim.utils.simple_preprocess will be used.

  Tokenizer must take a document and return a list of tokens.
- **use_embedding_model_tokenizer** (*bool (Optional, default False)*) – If using an embedding model other than doc2vec, use the model's tokenizer for document embedding. If set to True the tokenizer, either default or passed callable will be used to tokenize the text to extract the vocabulary for word embedding.
- **umap_args** (*dict (Optional, default None)*) – Pass custom arguments to UMAP.
- **hdbscan_args** (*dict (Optional, default None)*) – Pass custom arguments to HDBSCAN.
- **verbose** (*bool (Optional, default True)*) – Whether to print status data during training.

add_documents (*documents*, *doc_ids=None*, *tokenizer=None*, *use_embedding_model_tokenizer=False*, *embedding_batch_size=32*)

Update the model with new documents.

The documents will be added to the current model without changing existing document, word and topic vectors. Topic sizes will be updated.

If adding a large quantity of documents relative to the current model size, or documents containing a largely new vocabulary, a new model should be trained for best results.

Parameters

- **documents** (*List of str*) –
- **doc_ids** (*List of str, int (Optional)*) – Only required when doc_ids were given to the original model.

  A unique value per document that will be used for referring to documents in search results.
- **tokenizer** (*callable (Optional, default None)*) – Override the default tokenization method. If None then gensim.utils.simple_preprocess will be used.
- **use_embedding_model_tokenizer** (*bool (Optional, default False)*) – If using an embedding model other than doc2vec, use the model's tokenizer for document embedding.
- **embedding_batch_size** (*int (default=32)*) – Batch size for documents being embedded.

change_to_download_embedding_model ()

Use automatic download to load embedding model used for training. Top2Vec will no longer try and load the embedding model from a file if a embedding_model path was previously added.

delete_documents (*doc_ids*)

Delete documents from current model.

Warning: If document ids were not used in original model, deleting documents will change the indexes and therefore doc_ids.

The documents will be deleted from the current model without changing existing document, word and topic vectors. Topic sizes will be updated.

If deleting a large quantity of documents relative to the current model size a new model should be trained for best results.

Parameters **`doc_ids`** (*`List of str, int`*) – A unique value per document that is used for referring to documents in search results.

`generate_topic_wordcloud`(*topic_num*, *background_color='black'*, *reduced=False*)

Create a word cloud for a topic.

A word cloud will be generated and displayed. The most semantically similar words to the topic will have the largest size, less similar words will be smaller. The size is determined using the cosine distance of the word vectors from the topic vector.

Parameters

- **`topic_num`** (*`int`*) – The topic number to search.
- **`background_color`** (*`str (Optional, default='white')`*) –

  Background color for the word cloud image. Suggested options are:

  - white
  - black
- **`reduced`** (*`bool (Optional, default False)`*) – Original topics are used by default. If True the reduced topics will be used.

Returns

- *A matplotlib plot of the word cloud with the topic number will be*
- *displayed.*

`get_documents_topics`(*doc_ids*, *reduced=False*, *num_topics=1*)

Get document topics.

The topic of each document will be returned.

The corresponding original topics are returned unless reduced=True, in which case the reduced topics will be returned.

Parameters

- **`doc_ids`** (*`List of str, int`*) – A unique value per document that is used for referring to documents in search results. If ids were not given to the model, the index of each document in the model is the id.
- **`reduced`** (*`bool (Optional, default False)`*) – Original topics are returned by default. If True the reduced topics will be returned.
- **`num_topics`** (*`int (Optional, default 1)`*) – The number of topics to return per document.

Returns

- **topic_nums** (*array of int, shape(len(doc_ids), num_topics)*) – The topic number(s) of the document corresponding to each doc_id.
- **topic_score** (*array of float, shape(len(doc_ids), num_topics)*) – Semantic similarity of document to topic(s). The cosine similarity of the document and topic vector.

- **topics_words** (*array of shape(len(doc_ids), num_topics, 50)*) – For each topic the top 50 words are returned, in order of semantic similarity to topic.

  Example: [['data', 'deep', 'learning' … 'artificial'], <Topic 4> ['environment', 'warming', 'climate … 'temperature'] <Topic 21> …]

- **word_scores** (*array of shape(num_topics, 50)*) – For each topic the cosine similarity scores of the top 50 words to the topic are returned.

  Example: [[0.7132, 0.6473, 0.5700 … 0.3455], <Topic 4> [0.7818', 0.7671, 0.7603 … 0.6769] <Topic 21> …]

`get_num_topics`(*reduced=False*)

Get number of topics.

This is the number of topics Top2Vec has found in the data by default. If reduced is True, the number of reduced topics is returned.

Parameters **`reduced`** (`bool (Optional, default False)`) – The number of original topics will be returned by default. If True will return the number of reduced topics, if hierarchical topic reduction has been performed.

Returns **num_topics**

Return type int

`get_topic_hierarchy`()

Get the hierarchy of reduced topics. The mapping of each original topic to the reduced topics is returned.

Hierarchical topic reduction must be performed before calling this method.

Returns

hierarchy – Each index of the hierarchy corresponds to the topic number of a reduced topic. For each reduced topic the topic numbers of the original topics that were merged to create it are listed.

Example: [[3] <Reduced Topic 0> contains original Topic 3 [2,4] <Reduced Topic 1> contains original Topics 2 and 4 [0,1] <Reduced Topic 3> contains original Topics 0 and 1 …]

Return type list of ints

`get_topic_sizes`(*reduced=False*)

Get topic sizes.

The number of documents most similar to each topic. Topics are in increasing order of size.

The sizes of the original topics is returned unless reduced=True, in which case the sizes of the reduced topics will be returned.

Parameters **`reduced`** (`bool (Optional, default False)`) – Original topic sizes are returned by default. If True the reduced topic sizes will be returned.

Returns

- **topic_sizes** (*array of int, shape(num_topics)*) – The number of documents most similar to the topic.
- **topic_nums** (*array of int, shape(num_topics)*) – The unique number of every topic will be returned.

`get_topics`(*num_topics=None*, *reduced=False*)

Get topics, ordered by decreasing size. All topics are returned if num_topics is not specified.

The original topics found are returned unless reduced=True, in which case reduced topics will be returned.

Each topic will consist of the top 50 semantically similar words to the topic. These are the 50 words closest to topic vector along with cosine similarity of each word from vector. The higher the score the more relevant the word is to the topic.

Parameters

- **`num_topics`** (*`int, (Optional)`*) – Number of topics to return.
- **`reduced`** (*`bool (Optional, default False)`*) – Original topics are returned by default. If True the reduced topics will be returned.

Returns

- **topics_words** (*array of shape(num_topics, 50)*) – For each topic the top 50 words are returned, in order of semantic similarity to topic.

  Example: [['data', 'deep', 'learning' … 'artificial'], <Topic 0> ['environment', 'warming', 'climate … 'temperature'] <Topic 1> …]
- **word_scores** (*array of shape(num_topics, 50)*) – For each topic the cosine similarity scores of the top 50 words to the topic are returned.

  Example: [[0.7132, 0.6473, 0.5700 … 0.3455], <Topic 0> [0.7818', 0.7671, 0.7603 … 0.6769] <Topic 1> …]
- **topic_nums** (*array of int, shape(num_topics)*) – The unique number of every topic will be returned.

`hierarchical_topic_reduction`(*num_topics*)

Reduce the number of topics discovered by Top2Vec.

The most representative topics of the corpus will be found, by iteratively merging each smallest topic to the most similar topic until num_topics is reached.

Parameters **`num_topics`** (*`int`*) – The number of topics to reduce to.

Returns

hierarchy – Each index of hierarchy corresponds to the reduced topics, for each reduced topic the indexes of the original topics that were merged to create it are listed.

Example: [[3] <Reduced Topic 0> contains original Topic 3 [2,4] <Reduced Topic 1> contains original Topics 2 and 4 [0,1] <Reduced Topic 3> contains original Topics 0 and 1 …]

Return type list of ints

`index_document_vectors`(*ef_construction=200*, *M=64*)

Creates an index of the document vectors using hnswlib. This will lead to faster search times for models with a large number of documents.

For more information on hnswlib see: https://github.com/nmslib/hnswlib

Parameters

- **`ef_construction`** (*`int (Optional default 200)`*) – This parameter controls the trade-off between index construction time and index accuracy. Larger values will lead to greater accuracy but will take longer to construct.
- **`M`** (*`int (Optional default 64)`*) – This parameter controls the trade-off between both index size as well as construction time and accuracy. Larger values will lead to greater accuracy but will result in a larger index as well as longer construction time.

For more information on the parameters see: https://github.com/nmslib/hnswlib/blob/master/ALGO_PARAMS.md

index_word_vectors(*ef_construction=200*, *M=64*)

Creates an index of the word vectors using hnswlib. This will lead to faster search times for models with a large number of words.

For more information on hnswlib see: https://github.com/nmslib/hnswlib

Parameters

- **ef_construction** (*int (Optional default 200)*) – This parameter controls the trade-off between index construction time and index accuracy. Larger values will lead to greater accuracy but will take longer to construct.
- **M** (*int (Optional default 64)*) – This parameter controls the trade-off between both index size as well as construction time and accuracy. Larger values will lead to greater accuracy but will result in a larger index as well as longer construction time.

  For more information on the parameters see: https://github.com/nmslib/hnswlib/blob/master/ALGO_PARAMS.md

classmethod load(*file*)

Load a pre-trained model from the specified file.

Parameters **file** (*str*) – File where model will be loaded from.

query_documents(*query*, *num_docs*, *return_documents=True*, *use_index=False*, *ef=None*, *tokenizer=None*)

Semantic search of documents using a text query.

The most semantically similar documents to the query will be returned.

Parameters

- **query** (*string*) – Any sequence of text. This could be an actual question, a sentence, a paragraph or a document.
- **num_docs** (*int*) – Number of documents to return.
- **return_documents** (*bool (Optional default True)*) – Determines if the documents will be returned. If they were not saved in the model they will not be returned.
- **use_index** (*bool (Optional default False)*) – If index_documents method has been called, setting this to True will speed up search for models with large number of documents.
- **ef** (*int (Optional default None)*) – Higher ef leads to more accurate but slower search. This value must be higher than num_docs.

  For more information see: https://github.com/nmslib/hnswlib/blob/master/ALGO_PARAMS.md
- **tokenizer** (*callable (Optional, default None)*) – ** For doc2vec embedding model only **

  Override the default tokenization method. If None then gensim.utils.simple_preprocess will be used.

Returns

- **documents** (*(Optional) array of str, shape(num_docs)*) – The documents in a list, the most similar are first.

  Will only be returned if the documents were saved and if return_documents is set to True.
- **doc_scores** (*array of float, shape(num_docs)*) – Semantic similarity of document to vector. The cosine similarity of the document and vector.
- **doc_ids** (*array of int, shape(num_docs)*) – Unique ids of documents. If ids were not given to the model, the index of the document in the model will be returned.

query_topics (*query*, *num_topics*, *reduced=False*, *tokenizer=None*)

Semantic search of topics using text query.

These are the topics closest to the vector. Topics are ordered by proximity to the vector. Successive topics in the list are less semantically similar to the vector.

Parameters

- **query** (*string*) – Any sequence of text. This could be an actual question, a sentence, a paragraph or a document.
- **num_topics** (*int*) – Number of documents to return.
- **reduced** (*bool (Optional, default False)*) – Original topics are searched by default. If True the reduced topics will be searched.
- **tokenizer** (*callable (Optional, default None)*) – ** For doc2vec embedding model only **

  Override the default tokenization method. If None then gensim.utils.simple_preprocess will be used.

Returns

- **topics_words** (*array of shape (num_topics, 50)*) – For each topic the top 50 words are returned, in order of semantic similarity to topic.

  Example: [['data', 'deep', 'learning' … 'artificial'], <Topic 0> ['environment', 'warming', 'climate … 'temperature'] <Topic 1> …]
- **word_scores** (*array of shape (num_topics, 50)*) – For each topic the cosine similarity scores of the top 50 words to the topic are returned.

  Example: [[0.7132, 0.6473, 0.5700 … 0.3455], <Topic 0> [0.7818', 0.7671, 0.7603 … 0.6769] <Topic 1> …]
- **topic_scores** (*array of float, shape(num_topics)*) – For each topic the cosine similarity to the search keywords will be returned.
- **topic_nums** (*array of int, shape(num_topics)*) – The unique number of every topic will be returned.

save (*file*)

Saves the current model to the specified file.

Parameters **file** (*str*) – File where model will be saved.

search_documents_by_documents (*doc_ids*, *num_docs*, *doc_ids_neg=None*, *return_documents=True*, *use_index=False*, *ef=None*)

Semantic similarity search of documents.

The most semantically similar documents to the semantic combination of document ids provided will be returned. If negative document ids are provided, the documents will be semantically dissimilar to those document ids. Documents will be ordered by decreasing similarity. This method finds the closest document vectors to the provided documents averaged.

Parameters

- **`doc_ids`** (*`List of int, str`*) – Unique ids of document. If ids were not given, the index of document in the original corpus.
- **`doc_ids_neg`** (*`(Optional) List of int, str`*) – Unique ids of document. If ids were not given, the index of document in the original corpus.
- **`num_docs`** (*`int`*) – Number of documents to return.
- **`return_documents`** (*`bool (Optional default True)`*) – Determines if the documents will be returned. If they were not saved in the model they will also not be returned.
- **`use_index`** (*`bool (Optional default False)`*) – If index_documents method has been called, setting this to True will speed up search for models with large number of documents.
- **`ef`** (*`int (Optional default None)`*) – Higher ef leads to more accurate but slower search. This value must be higher than num_docs.

  For more information see: https://github.com/nmslib/hnswlib/blob/master/ALGO_PARAMS.md

Returns

- **documents** (*(Optional) array of str, shape(num_docs)*) – The documents in a list, the most similar are first.

  Will only be returned if the documents were saved and if return_documents is set to True.
- **doc_scores** (*array of float, shape(num_docs)*) – Semantic similarity of document to keywords. The cosine similarity of the document and average of keyword vectors.
- **doc_ids** (*array of int, shape(num_docs)*) – Unique ids of documents. If ids were not given to the model, the index of the document in the model will be returned.

`search_documents_by_keywords`(*keywords*, *num_docs*, *keywords_neg=None*, *return_documents=True*, *use_index=False*, *ef=None*)

Semantic search of documents using keywords.

The most semantically similar documents to the combination of the keywords will be returned. If negative keywords are provided, the documents will be semantically dissimilar to those words. Too many keywords or certain combinations of words may give strange results. This method finds an average vector(negative keywords are subtracted) of all the keyword vectors and returns the documents closest to the resulting vector.

Parameters

- **`keywords`** (*`List of str`*) – List of positive keywords being used for search of semantically similar documents.
- **`keywords_neg`** (*`List of str (Optional)`*) – List of negative keywords being used for search of semantically dissimilar documents.
- **`num_docs`** (*`int`*) – Number of documents to return.
- **`return_documents`** (*`bool (Optional default True)`*) – Determines if the documents will be returned. If they were not saved in the model they will also not be returned.

- **`use_index`** (*`bool (Optional default False)`*) – If index_documents method has been called, setting this to True will speed up search for models with large number of documents.
- **`ef`** (*`int (Optional default None)`*) – Higher ef leads to more accurate but slower search. This value must be higher than num_docs.

  For more information see: https://github.com/nmslib/hnswlib/blob/master/ALGO_PARAMS.md

Returns

- **documents** (*(Optional) array of str, shape(num_docs)*) – The documents in a list, the most similar are first.

  Will only be returned if the documents were saved and if return_documents is set to True.
- **doc_scores** (*array of float, shape(num_docs)*) – Semantic similarity of document to keywords. The cosine similarity of the document and average of keyword vectors.
- **doc_ids** (*array of int, shape(num_docs)*) – Unique ids of documents. If ids were not given to the model, the index of the document in the model will be returned.

`search_documents_by_topic`(*topic_num*, *num_docs*, *return_documents=True*, *reduced=False*)

Get the most semantically similar documents to the topic.

These are the documents closest to the topic vector. Documents are ordered by proximity to the topic vector. Successive documents in the list are less semantically similar to the topic.

Parameters

- **`topic_num`** (*`int`*) – The topic number to search.
- **`num_docs`** (*`int`*) – Number of documents to return.
- **`return_documents`** (*`bool (Optional default True)`*) – Determines if the documents will be returned. If they were not saved in the model they will not be returned.
- **`reduced`** (*`bool (Optional, default False)`*) – Original topics are used to search by default. If True the reduced topics will be used.

Returns

- **documents** (*(Optional) array of str, shape(num_docs)*) – The documents in a list, the most similar are first.

  Will only be returned if the documents were saved and if return_documents is set to True.
- **doc_scores** (*array of float, shape(num_docs)*) – Semantic similarity of document to topic. The cosine similarity of the document and topic vector.
- **doc_ids** (*array of int, shape(num_docs)*) – Unique ids of documents. If ids were not given to the model, the index of the document in the model will be returned.

`search_documents_by_vector`(*vector*, *num_docs*, *return_documents=True*, *use_index=False*, *ef=None*)

Semantic search of documents using a vector.

These are the documents closest to the vector. Documents are ordered by proximity to the vector. Successive documents in the list are less semantically similar to the vector.

Parameters

- **`vector`** (*`array of shape(vector dimension, 1)`*) – The vector dimension should be the same as the vectors in the topic_vectors variable. (i.e. model.topic_vectors.shape[1])
- **`num_docs`** (*`int`*) – Number of documents to return.
- **`return_documents`** (*`bool (Optional default True)`*) – Determines if the documents will be returned. If they were not saved in the model they will not be returned.
- **`use_index`** (*`bool (Optional default False)`*) – If index_documents method has been called, setting this to True will speed up search for models with large number of documents.
- **`ef`** (*`int (Optional default None)`*) – Higher ef leads to more accurate but slower search. This value must be higher than num_docs.

  For more information see: https://github.com/nmslib/hnswlib/blob/master/ALGO_PARAMS.md

Returns

- **documents** (*(Optional) array of str, shape(num_docs)*) – The documents in a list, the most similar are first.

  Will only be returned if the documents were saved and if return_documents is set to True.
- **doc_scores** (*array of float, shape(num_docs)*) – Semantic similarity of document to vector. The cosine similarity of the document and vector.
- **doc_ids** (*array of int, shape(num_docs)*) – Unique ids of documents. If ids were not given to the model, the index of the document in the model will be returned.

`search_topics`(*keywords*, *num_topics*, *keywords_neg=None*, *reduced=False*)
Semantic search of topics using keywords.

The most semantically similar topics to the combination of the keywords will be returned. If negative keywords are provided, the topics will be semantically dissimilar to those words. Topics will be ordered by decreasing similarity to the keywords. Too many keywords or certain combinations of words may give strange results. This method finds an average vector(negative keywords are subtracted) of all the keyword vectors and returns the topics closest to the resulting vector.

Parameters

- **`keywords`** (*`List of str`*) – List of positive keywords being used for search of semantically similar documents.
- **`keywords_neg`** (*`(Optional) List of str`*) – List of negative keywords being used for search of semantically dissimilar documents.
- **`num_topics`** (*`int`*) – Number of documents to return.
- **`reduced`** (*`bool (Optional, default False)`*) – Original topics are searched by default. If True the reduced topics will be searched.

Returns

- **topics_words** (*array of shape (num_topics, 50)*) – For each topic the top 50 words are returned, in order of semantic similarity to topic.

  Example: [['data', 'deep', 'learning' … 'artificial'], <Topic 0> ['environment', 'warming', 'climate … 'temperature'] <Topic 1> …]

- **word_scores** (*array of shape (num_topics, 50)*) – For each topic the cosine similarity scores of the top 50 words to the topic are returned.

  Example: [[0.7132, 0.6473, 0.5700 … 0.3455], <Topic 0> [0.7818', 0.7671, 0.7603 … 0.6769] <Topic 1> …]
- **topic_scores** (*array of float, shape(num_topics)*) – For each topic the cosine similarity to the search keywords will be returned.
- **topic_nums** (*array of int, shape(num_topics)*) – The unique number of every topic will be returned.

`search_topics_by_vector`(*vector*, *num_topics*, *reduced=False*)

Semantic search of topics using keywords.

These are the topics closest to the vector. Topics are ordered by proximity to the vector. Successive topics in the list are less semantically similar to the vector.

Parameters

- **vector** (*array of shape(vector dimension, 1)*) – The vector dimension should be the same as the vectors in the topic_vectors variable. (i.e. model.topic_vectors.shape[1])
- **num_topics** (*int*) – Number of documents to return.
- **reduced** (*bool (Optional, default False)*) – Original topics are searched by default. If True the reduced topics will be searched.

Returns

- **topics_words** (*array of shape (num_topics, 50)*) – For each topic the top 50 words are returned, in order of semantic similarity to topic.

  Example: [['data', 'deep', 'learning' … 'artificial'], <Topic 0> ['environment', 'warming', 'climate … 'temperature'] <Topic 1> …]
- **word_scores** (*array of shape (num_topics, 50)*) – For each topic the cosine similarity scores of the top 50 words to the topic are returned.

  Example: [[0.7132, 0.6473, 0.5700 … 0.3455], <Topic 0> [0.7818', 0.7671, 0.7603 … 0.6769] <Topic 1> …]
- **topic_scores** (*array of float, shape(num_topics)*) – For each topic the cosine similarity to the search keywords will be returned.
- **topic_nums** (*array of int, shape(num_topics)*) – The unique number of every topic will be returned.

`search_words_by_vector`(*vector*, *num_words*, *use_index=False*, *ef=None*)

Semantic search of words using a vector.

These are the words closest to the vector. Words are ordered by proximity to the vector. Successive words in the list are less semantically similar to the vector.

Parameters

- **vector** (*array of shape(vector dimension, 1)*) – The vector dimension should be the same as the vectors in the topic_vectors variable. (i.e. model.topic_vectors.shape[1])
- **num_words** (*int*) – Number of words to return.

- **use_index** (*bool (Optional default False)*) – If index_words method has been called, setting this to True will speed up search for models with large number of words.
- **ef** (*int (Optional default None)*) – Higher ef leads to more accurate but slower search. This value must be higher than num_docs.

  For more information see: https://github.com/nmslib/hnswlib/blob/master/ALGO_PARAMS.md

Returns

- **words** (*array of str, shape(num_words)*) – The words in a list, the most similar are first.
- **word_scores** (*array of float, shape(num_words)*) – Semantic similarity of word to vector. The cosine similarity of the word and vector.

set_embedding_model(*embedding_model*)

Set the embedding model. This is called after loading a saved Top2Vec model which was trained with a passed callable embedding_model.

Parameters **embedding_model** (*callable*) – This must be the same embedding model used during training.

similar_words(*keywords*, *num_words*, *keywords_neg=None*, *use_index=False*, *ef=None*)

Semantic similarity search of words.

The most semantically similar word to the combination of the keywords will be returned. If negative keywords are provided, the words will be semantically dissimilar to those words. Too many keywords or certain combinations of words may give strange results. This method finds an average vector(negative keywords are subtracted) of all the keyword vectors and returns the words closest to the resulting vector.

Parameters

- **keywords** (*List of str*) – List of positive keywords being used for search of semantically similar words.
- **keywords_neg** (*List of str*) – List of negative keywords being used for search of semantically dissimilar words.
- **num_words** (*int*) – Number of words to return.
- **use_index** (*bool (Optional default False)*) – If index_words method has been called, setting this to True will speed up search for models with large number of words.
- **ef** (*int (Optional default None)*) – Higher ef leads to more accurate but slower search. This value must be higher than num_docs.

  For more information see: https://github.com/nmslib/hnswlib/blob/master/ALGO_PARAMS.md

Returns

- **words** (*array of str, shape(num_words)*) – The words in a list, the most similar are first.
- **word_scores** (*array of float, shape(num_words)*) – Semantic similarity of word to keywords. The cosine similarity of the word and average of keyword vectors.

update_embedding_model_path(*embedding_model_path*)

Update the path of the embedding model to be loaded. The model will no longer be downloaded but loaded from the path location.

Warning: the model at embedding_model_path must match the embedding_model parameter type.

Parameters **`embedding_model_path`** (*`Str`*) – Path to downloaded embedding model.

`top2vec.Top2Vec.`**`default_tokenizer`**(*document*)

Tokenize a document for training and remove too long/short words

Parameters **`document`** (*`List of str`*) – Input document.

Returns **tokenized_document** – List of tokens.

Return type List of str

`top2vec.Top2Vec.`**`get_chunks`**(*tokens*, *chunk_length*, *max_num_chunks*, *chunk_overlap_ratio*)

Split a document into sequential chunks

Parameters

- **`tokens`** (*`List of str`*) – Input document tokens.
- **`chunk_length`** (*`int`*) – Length of each document chunk.
- **`max_num_chunks`** (*`int (Optional, default None)`*) – Limit the number of document chunks
- **`chunk_overlap_ratio`** (*`float`*) – Fraction of overlapping tokens between sequential chunks.

Returns **chunked_document** – List of document chunks.

Return type List of str

`top2vec.Top2Vec.`**`get_random_chunks`**(*tokens*, *chunk_length*, *chunk_len_coverage_ratio*, *max_num_chunks*)

Split a document into chunks starting at random positions

Parameters

- **`tokens`** (*`List of str`*) – Input document tokens.
- **`chunk_length`** (*`int`*) – Length of each document chunk.
- **`chunk_len_coverage_ratio`** (*`float`*) – Proportion of token length that will be covered by chunks. Default value of 1.0 means chunk lengths will add up to number of tokens. This does not mean all tokens will be covered.
- **`max_num_chunks`** (*`int (Optional, default None)`*) – Limit the number of document chunks

Returns **chunked_document** – List of document chunks.

Return type List of str

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index