BERTopic

BERTopic is a topic modeling technique that leverages 🤗 transformers and c-TF-IDF to create dense clusters allowing for easily interpretable topics whilst keeping important words in the topic descriptions.

BERTopic supports all kinds of topic modeling techniques:

Guided	Supervised	Semi-supervised
Manual	Multi-topic distributions	Hierarchical
Class-based	Dynamic	Online/Incremental
Multimodal	Multi-aspect	Text Generation/LLM
Zero-shot **(new!)**	Merge Models **(new!)**	Seed Words **(new!)**

Corresponding medium posts can be found here, here and here. For a more detailed overview, you can read the paper or see a brief overview.

Installation

Installation, with sentence-transformers, can be done using pypi:

```
pip install bertopic
```

You may want to install more depending on the transformers and language backends that you will be using. The possible installations are:

```
# Choose an embedding backend
pip install bertopic[flair, gensim, spacy, use]

# Topic modeling with images
pip install bertopic[vision]
```

Quick Start

We start by extracting topics from the well-known 20 newsgroups dataset containing English documents:

```
from bertopic import BERTopic
from sklearn.datasets import fetch_20newsgroups

docs = fetch_20newsgroups(subset='all',  remove=('headers', 'footers', 'quotes'))
['data']

topic_model = BERTopic()
topics, probs = topic_model.fit_transform(docs)
```

After generating topics and their probabilities, we can access the frequent topics that were generated:

```
>>> topic_model.get_topic_info()

Topic   Count   Name
-1      4630    -1_can_your_will_any
0       693     49_windows_drive_dos_file
1       466     32_jesus_bible_christian_faith
2       441     2_space_launch_orbit_lunar
3       381     22_key_encryption_keys_encrypted
```

-1 refers to all outliers and should typically be ignored. Next, let's take a look at the most frequent topic that was generated, topic 0:

```
>>> topic_model.get_topic(0)

[('windows', 0.006152228076250982),
 ('drive', 0.004982897610645755),
 ('dos', 0.004845038866360651),
 ('file', 0.004140142872194834),
 ('disk', 0.004131678774810884),
 ('mac', 0.003624848635985097),
 ('memory', 0.0034840976976789903),
 ('software', 0.0034415334250699077),
 ('email', 0.0034239554442333257),
 ('pc', 0.003047105930670237)]
```

Using `.get_document_info`, we can also extract information on a document level, such as their corresponding topics, probabilities, whether they are representative documents for a topic, etc.:

```
>>> topic_model.get_document_info(docs)
```

```
Document                                      Topic    Name                        Top_n_words
Probability    ...
I am sure some bashers of Pens...              0        0_game_team_games_season    game - team
- games...            0.200010        ...
My brother is in the market for...             -1      -1_can_your_will_any         can - your
- will...              0.420668        ...
Finally you said what you dream...             -1      -1_can_your_will_any         can - your
- will...              0.807259        ...
Think! It is the SCSI card doing...            49      49_windows_drive_dos_file    windows -
drive - docs...          0.071746        ...
1) I have an old Jasmine drive...              49      49_windows_drive_dos_file    windows -
drive - docs...          0.038983        ...
```

> **Multilingual**
>
> Use `BERTopic(language="multilingual")` to select a model that supports 50+ languages.

Fine-tune Topic Representations

In BERTopic, there are a number of different topic representations that we can choose from. They are all quite different from one another and give interesting perspectives and variations of topic representations. A great start is `KeyBERTInspired`, which for many users increases the coherence and reduces stopwords from the resulting topic representations:

```
from bertopic.representation import KeyBERTInspired

# Fine-tune your topic representations
representation_model = KeyBERTInspired()
topic_model = BERTopic(representation_model=representation_model)
```

However, you might want to use something more powerful to describe your clusters. You can even use ChatGPT or other models from OpenAI to generate labels, summaries, phrases, keywords, and more:

```
import openai
from bertopic.representation import OpenAI

# Fine-tune topic representations with GPT
client = openai.OpenAI(api_key="sk-...")
representation_model = OpenAI(client, model="gpt-4o-mini", chat=True)
topic_model = BERTopic(representation_model=representation_model)
```

Multi-aspect Topic Modeling

Instead of iterating over all of these different topic representations, you can model them simultaneously with multi-aspect topic representations in BERTopic.

Modularity

By default, the main steps for topic modeling with BERTopic are sentence-transformers, UMAP, HDBSCAN, and c-TF-IDF run in sequence. However, it assumes some independence between these steps which makes BERTopic quite modular. In other words, BERTopic not only allows you to build your own topic model but to explore several topic modeling techniques on top of your customized topic model:

You can swap out any of these models or even remove them entirely. The following steps are completely modular:

1. Embedding documents
2. Reducing dimensionality of embeddings
3. Clustering reduced embeddings into topics
4. Tokenization of topics
5. Weight tokens
6. Represent topics with one or multiple representations

To find more about the underlying algorithm and assumptions here.

Overview

BERTopic has many functions that quickly can become overwhelming. To alleviate this issue, you will find an overview of all methods and a short description of its purpose.

Common

Below, you will find an overview of common functions in BERTopic.

Method	Code
Fit the model	`.fit(docs)`
Fit the model and predict documents	`.fit_transform(docs)`
Predict new documents	`.transform([new_doc])`
Access single topic	`.get_topic(topic=12)`
Access all topics	`.get_topics()`
Get topic freq	`.get_topic_freq()`
Get all topic information	`.get_topic_info()`
Get all document information	`.get_document_info(docs)`

Get representative docs per topic	`.get_representative_docs()`
Update topic representation	`.update_topics(docs, n_gram_range=(1, 3))`
Generate topic labels	`.generate_topic_labels()`
Set topic labels	`.set_topic_labels(my_custom_labels)`
Merge topics	`.merge_topics(docs, topics_to_merge)`
Reduce nr of topics	`.reduce_topics(docs, nr_topics=30)`
Reduce outliers	`.reduce_outliers(docs, topics)`
Find topics	`.find_topics("vehicle")`
Save model	`.save("my_model", serialization="safetensors")`
Load model	`BERTopic.load("my_model")`
Get parameters	`.get_params()`

Attributes

After having trained your BERTopic model, several are saved within your model. These attributes, in part, refer to how model information is stored on an estimator during fitting. The attributes that you see below all end in `_` and are public attributes that can be used to access model information.

Attribute	**Description**
`.topics_`	The topics that are generated for each document after training or updating the topic model.
`.probabilities_`	The probabilities that are generated for each document if HDBSCAN is used.

`.topic_sizes_`	The size of each topic
`.topic_mapper_`	A class for tracking topics and their mappings anytime they are merged/reduced.
`.topic_representations_`	The top *n* terms per topic and their respective c-TF-IDF values.
`.c_tf_idf_`	The topic-term matrix as calculated through c-TF-IDF.
`.topic_aspects_`	The different aspects, or representations, of each topic.
`.topic_labels_`	The default labels for each topic.
`.custom_labels_`	Custom labels for each topic as generated through `.set_topic_labels`.
`.topic_embeddings_`	The embeddings for each topic if `embedding_model` was used.
`.representative_docs_`	The representative documents for each topic if HDBSCAN is used.

Variations

There are many different use cases in which topic modeling can be used. As such, several variations of BERTopic have been developed such that one package can be used across many use cases.

Method	**Code**
Topic Distribution Approximation	`.approximate_distribution(docs)`
Online Topic Modeling	`.partial_fit(doc)`
Semi-supervised Topic Modeling	`.fit(docs, y=y)`

Supervised Topic Modeling	`.fit(docs, y=y)`
Manual Topic Modeling	`.fit(docs, y=y)`
Multimodal Topic Modeling	`.fit(docs, images=images)`
Topic Modeling per Class	`.topics_per_class(docs, classes)`
Dynamic Topic Modeling	`.topics_over_time(docs, timestamps)`
Hierarchical Topic Modeling	`.hierarchical_topics(docs)`
Guided Topic Modeling	`BERTopic(seed_topic_list=seed_topic_list)`
Zero-shot Topic Modeling	`BERTopic(zeroshot_topic_list=zeroshot_topic_list)`
Merge Multiple Models	`BERTopic.merge_models([topic_model_1, topic_model_2])`

Visualizations

Evaluating topic models can be rather difficult due to the somewhat subjective nature of evaluation. Visualizing different aspects of the topic model helps in understanding the model and makes it easier to tweak the model to your liking.

Method	**Code**
Visualize Topics	`.visualize_topics()`
Visualize Documents	`.visualize_documents()`
Visualize Document with DataMapPlot	`.visualize_document_datamap()`
Visualize Document Hierarchy	`.visualize_hierarchical_documents()`
Visualize Topic Hierarchy	`.visualize_hierarchy()`

Visualize Topic Tree	`.get_topic_tree(hierarchical_topics)`
Visualize Topic Terms	`.visualize_barchart()`
Visualize Topic Similarity	`.visualize_heatmap()`
Visualize Term Score Decline	`.visualize_term_rank()`
Visualize Topic Probability Distribution	`.visualize_distribution(probs[0])`
Visualize Topics over Time	`.visualize_topics_over_time(topics_over_time)`
Visualize Topics per Class	`.visualize_topics_per_class(topics_per_class)`

Citation

To cite the BERTopic paper, please use the following bibtex reference:

```
@article{grootendorst2022bertopic,
  title={BERTopic: Neural topic modeling with a class-based TF-IDF procedure},
  author={Grootendorst, Maarten},
  journal={arXiv preprint arXiv:2203.05794},
  year={2022}
}
```