Eric Kleppen Follow

Feb 12 · 11 min read · ✨ · Listen

Topic Modeling For Beginners Using BERTopic and Python

How to make sense of your text data by reducing it to topics

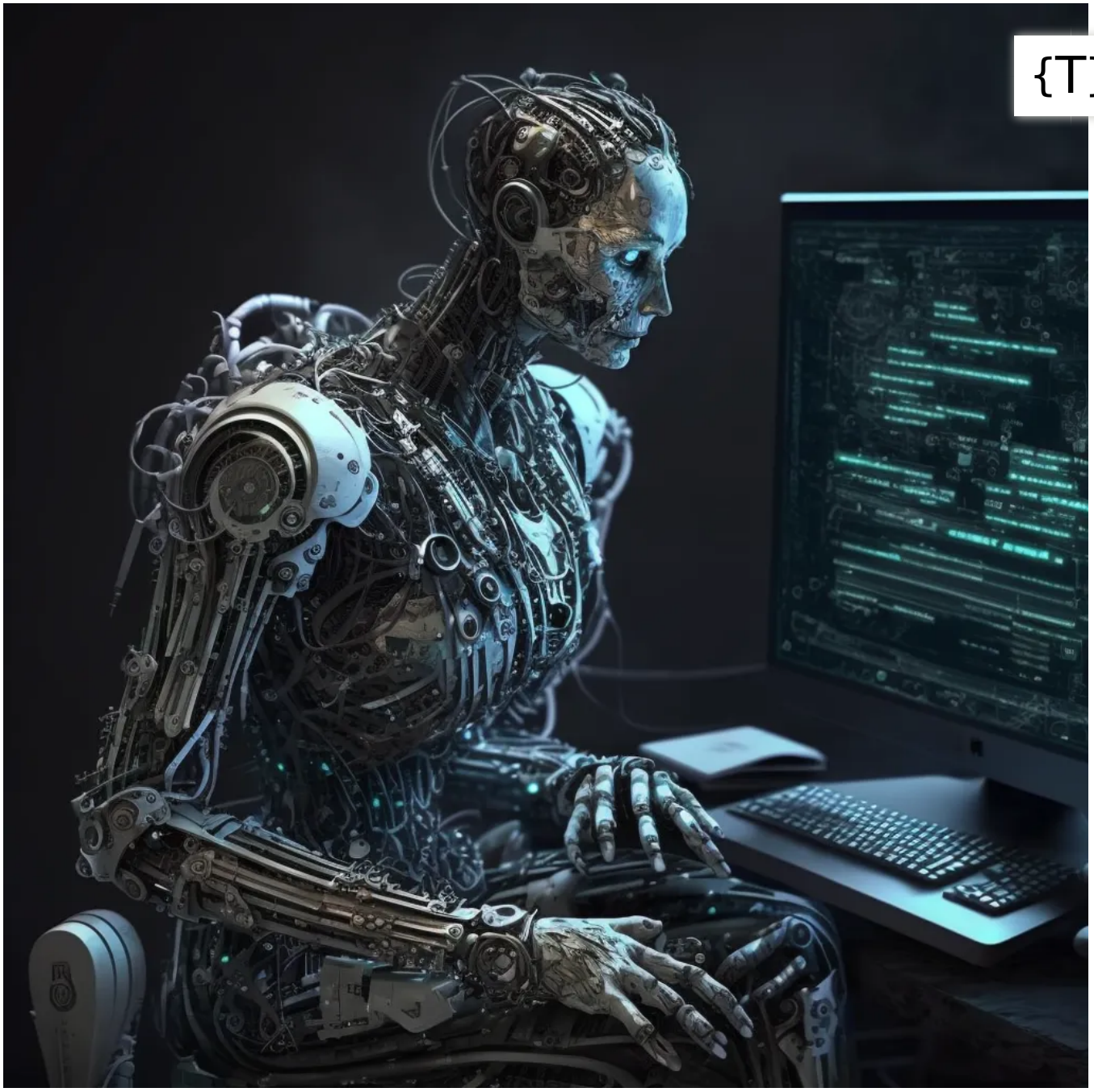


Image by Eric Kleppen

Working with text

I recently got a job working for a company that develops messaging software used in the hospitality industry. Since starting, one of my main focuses has been designing queries and features that help our customers and our customer success team make sense of unstructured messaging data. One thing our stakeholders want to understand is the topics being discussed between agents and guests.

In this article, we will explore how to use the BERTopic Python library to uncover

the topics hidden within thousands of Cabernet Sauvignon wine reviews. Click here to download the data from kaggle.com to follow the examples. As always, you c find the full example code at the bottom of the article.

We'll cover the following questions and walk through code examples:

- What is topic modeling?
- What is BERTopic?
- How does BERTopic work?
- How to prepare our data for topic modeling using BERTopic
- How to create a topic model using BERTopic
- How to explore and visualize a topic model using BERTopic

What is topic modeling?

In the field of Natural Language Processing (NLP), topic modeling is typically done as an unsupervised learning task in which algorithms apply statistics to figure out what words are similar so they can be clustered into groups. The clusters represent the topics and are composed of the words that fit within the topic. For example, if we're performing topic modeling on customer reviews of a smartphone, we might see text grouped into topics like camera, screen size, and battery. When we explore the camera topic, we'll probably find words like HD, Megapixel, low-light, and blur. Essentially, the algorithm tries to figure out what words are similar and then groups them, forming a topic.

Topic modeling is an art as well as a science, and the results can be very subjective. In most cases, an analyst needs to review the words within a topic to understand what the topic is really about and validate it. Although large language models have made topic modeling simpler, fine-tuning a language model to better interpret domain-specific jargon might be necessary to create a good topic model.

What is BERTopic?

BERTopic is a topic modeling technique that utilizes language models, like Google's BERT, to transform text into numerical representations called embeddings. After creating the embeddings, the data are passed through various algorithms to produce dense clusters of words, resulting in easily interpretable topics.

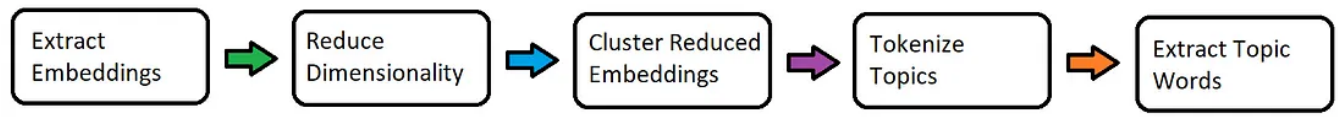
Install BERTopic using pip:

```
pip install bertopic
```

How does BERTopic work?

Although it's designed to produce good results "out-of-the-box," BERTopic was built to be modular and utilizes a sequence of steps to create the topic model. The BERTopic algorithm has five primary steps:

1. Extract embeddings
2. Reduce dimensionality
3. Cluster reduced embeddings
4. Tokenize topics
5. Extract topic words

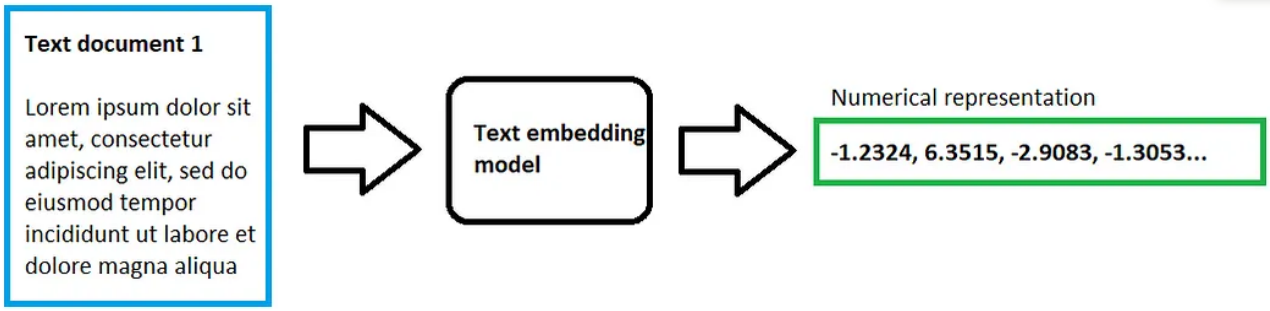


The BERTopic Algorithm workflow (image by author).

1. Extract embeddings

BERTopic is machine learning. Machine learning uses math, and math works with numbers, so we need to transform our text into numbers before passing the data into the clustering algorithm. To transform the text into numerical representations, BERTopic uses the sentence-transformers library natively since it provides several

pre-trained language models that have been fine-tuned for NLP tasks.

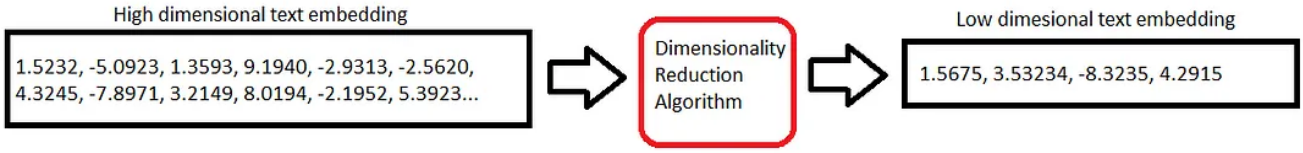


Extracting text embeddings (image by author)

The BERTopic default settings use the language model `"all-MiniLM-L6-v2"` for English embeddings or `"paraphrase-multilingual-MiniLM-L12-v2"` for multilingual embeddings. Although these are the default language models, you can choose any embedding model or even pass in embeddings directly.

2. Reduce dimensionality

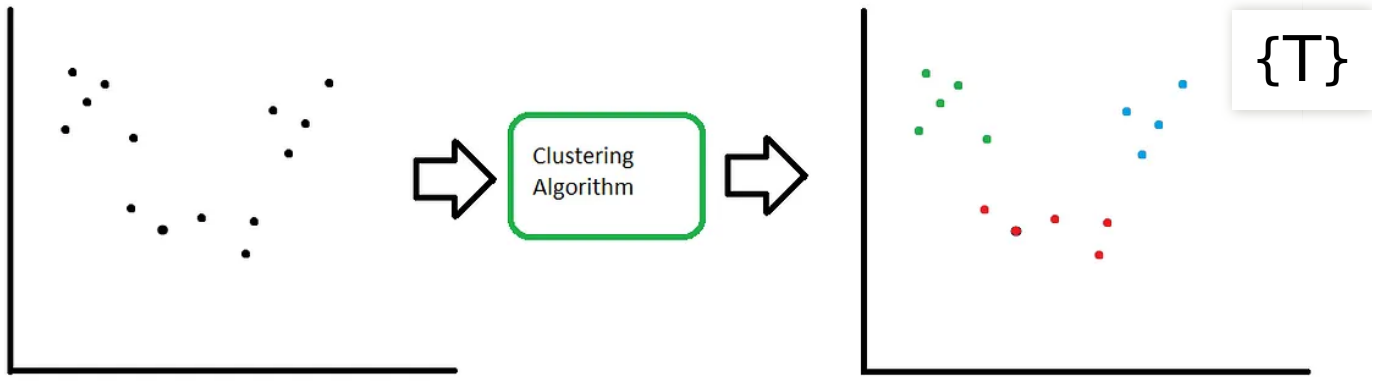
After transforming text into embeddings, each document is represented by a high-dimensional vector which is essentially a long list of numbers. Clustering techniques aren't always good at handling high-dimensional data, so BERTopic performs dimensionality reduction using the UMAP algorithm by default. UMAP is used because it keeps some of the data's structure which is important for clustering text based on similarity.



Reducing the dimensionality of the text embedding (image by author)

3. Cluster reduced embeddings

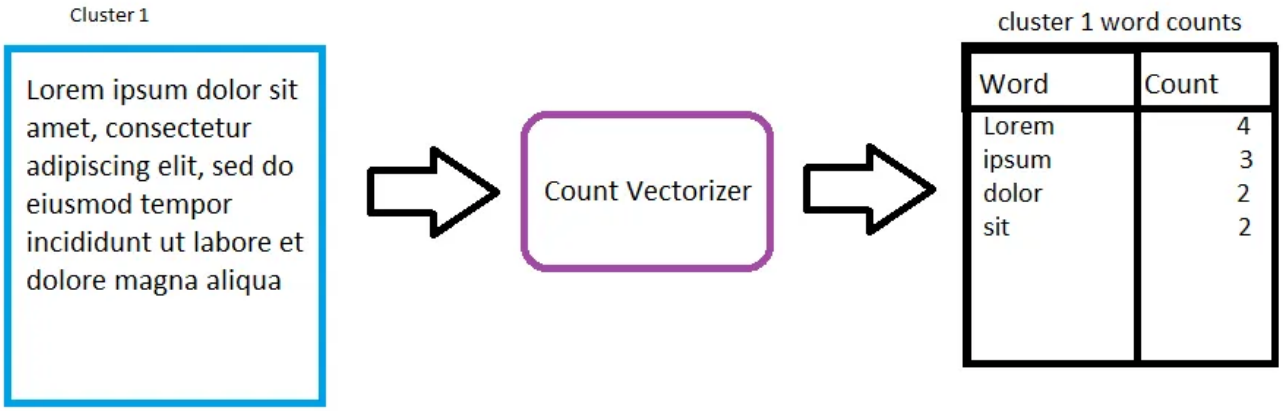
Once the dimensionality has been reduced, BERTopic clusters the data using the HDBSCAN because it can form clusters of different shapes and can identify outliers. Since it doesn't force text into clusters, the resulting topic representations are less noisy.



Clustering the data (image by author)

4. Tokenize topics

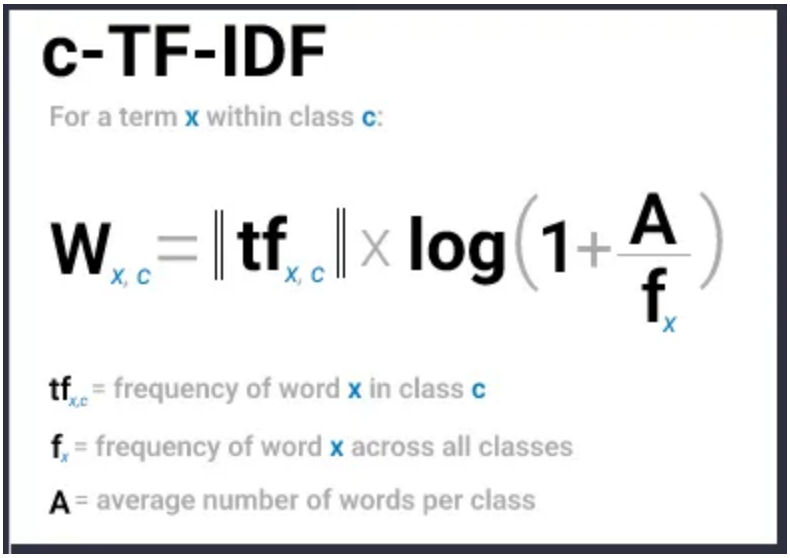
When BERTopic clusters the data using HDBSCAN, the resulting clusters can have varying degrees of density and different shapes. To generate topics without making assumptions about the expected structure of the clusters, BERTopic uses a bag-of-words approach by counting how often each word appears in each cluster.



Tokenizing the topics using bag of words (image by author)

5. Extract topic words

After generating the bag of words representation, BERTopic uses c-TF-IDF (class-based TF-IDF) to determine what makes one cluster different from another. For example, it calculates which words are typical for cluster 1 and not typical in the other clusters by comparing the importance of words within clusters.



The C-TF-IDF calculation [1] (https://maartengr.github.io/BERTopic/algorithm/algorithm.html#4-bag-of-words)

Understanding how BERTopic works is important because it is designed to be modular so that all of the parameters can be tuned to suit the use case. Each of these steps is customizable making BERTopic a very dynamic topic modeling tool. Now that we know how the algorithm works, let's explore the code.

How to prepare our data for topic modeling using BERTopic

With the wine reviews CSV file downloaded from Kaggle and BERTopic installed in the Python environment, let's begin by importing the dependencies and then opening the CSV file into a pandas dataframe object.

```
#import dependencies
import pandas as pd
from bertopic import BERTopic
from sentence_transformers import SentenceTransformer, util
from umap import UMAP
from hdbscan import HDBSCAN
from sklearn.feature_extraction.text import CountVectorizer
from bertopic.vectorizers import ClassTfidfTransformer
import numpy as np


#create a pandas dataframe
df = pd.read_csv('winemag-data-130k-v2.csv')
```

```
#check the first 5 dataframe rows
df.head()
```

	Unnamed: 0	country	description	designation	points	price	province	region_1	region_2	taster_name	taster_twitter_handle	title	variety	winery
0	0	Italy	Aromas include tropical fruit, broom, brimston...	Vulkà Bianco	87	NaN	Sicily & Sardinia	Etna	NaN	Kerin O'Keefe	@kerinokeefe	Nicosia 2013 Vulkà Bianco (Etna)	White Blend	Nicosia
1	1	Portugal	This is ripe and fruity, a wine that is smooth...	Avidagos	87	15.0	Douro	NaN	NaN	Roger Voss	@vossroger	Quinta dos Avidagos 2011 Avidagos Red (Douro)	Portuguese Red	Quinta dos Avidagos
2	2	US	Tart and snappy, the flavors of lime flesh and...	NaN	87	14.0	Oregon	Willamette Valley	Willamette Valley	Paul Gregutt	@paulgwine	Rainstorm 2013 Pinot Gris (Willamette Valley)	Pinot Gris	Rainstorm

df.head() (image by author)

Processing over 100,000 records will take a good amount of time, so to reduce the amount of time and computer resources required, let's reduce the data to only the Cabernet Sauvignon reviews using the **dataframe.loc[]** method.

```
#reduce the data to Cabernet Sauvignon reviews
df = df.loc[df.variety == 'Cabernet Sauvignon']

#generate summary statistics of the numerical columns
df.describe()
```

After reducing the dataframe, we're left with 9472 rows of data to explore.

From here, store the array of wine descriptions in a variable named docs so we can pass it into BERTopic.

```
#create an array of descriptions
docs = df.description.values
```

Dealing with stopwords

Stopwords are commonly used words in any language, like "*and*," "*the*," and "*is*." In older methods of topic modeling, it was important to remove stopwords because they are noise and don't contribute any information to the interpretation of topics. However, generating text embeddings using transformer-based models like BERT need the stop words within the text in order to model the context and create accurate embeddings. Therefore, it is best to leave stopwords in the text when using BERTopic and utilize the count vectorizer model when tokenizing topics (step 4) to handle stopwords appearing in topics.

How to create a topic model using BERTopic

In only a few lines of code, we can instantiate BERTopic, fit our data, and generate our topic model. Before looking at how we can customize each step in the BERTopic algorithm, let's start by using the default settings to get a benchmark of topics.

```
#instantiate BERTopic
topic_model = BERTopic(language="english", calculate_probabilities=True, verbos

#generate the topics
topics, probs = topic_model.fit_transform(docs)
```

If the dataset is large, you can set **calculate_probabilities** to *False* to save on compute resources since it is a resource-intensive calculation. On my machine, it takes about 2.5 minutes to generate the topic model for the ~9.5k records.

Once the topic model is complete, we can explore the topics using the **get_topic_info()** method to output a dataframe containing the topic group, the count of words, and the topic name:

```
topic_model.get_topic_info()
```

	Topic	Count	Name
0	-1	3172	-1_and_the_of_with
1	0	657	0_cabernet_wine_the_its
2	1	583	1_sauvignon_cabernet_of_and
3	2	391	2_blackberry_flavors_now_but
4	3	357	3_wine_texture_aromas_flavors
...	...	...	...
85	84	11	84_napa_valley_sauvignon_cabernet
86	85	10	85_tannic_quibble_broadly_etched
87	86	10	86_sugary_dessert_drinkers_sweet
88	87	10	87_steak_melts_texture_winner
89	88	10	88_clove_cinnamon_leatherbound_perceptible

topic_model.get_topic_info() output (image by author)

Notice we generated 90 topics. The first topic is **-1** and contains the most records. This is the *outliers topic* and should typically be ignored during analysis.

Customizing the BERTopic algorithm

We can see that stop words appear in some of the topic names. If we want to customize the five steps of the BERTopic algorithm to do things like remove the stop words, we can instantiate each of the five components and set the hyper-parameters individually. Then we can pass the custom models as parameters into BERTopic.

```
# Step 1 - Extract embeddings
embedding_model = SentenceTransformer("all-MiniLM-L6-v2")

# Step 2 - Reduce dimensionality
umap_model = UMAP(n_neighbors=15, n_components=5, min_dist=0.0, metric='cosine'

# Step 3 - Cluster reduced embeddings
hdbscan_model = HDBSCAN(min_cluster_size=15, metric='euclidean', cluster_select

# Step 4 - Tokenize topics
vectorizer_model = CountVectorizer(stop_words="english")

# Step 5 - Create topic representation
ctfidf_model = ClassTfidfTransformer()
```

```
# All steps together
topic_model = BERTopic(
  embedding_model=embedding_model,    # Step 1 - Extract embeddings
  umap_model=umap_model,              # Step 2 - Reduce dimensionality
  hdbscan_model=hdbscan_model,        # Step 3 - Cluster reduced embeddings
  vectorizer_model=vectorizer_model,  # Step 4 - Tokenize topics
  ctfidf_model=ctfidf_model,          # Step 5 - Extract topic words
  diversity=0.5,                      # Diversify topic words
  calculate_probabilities=True,
  verbose=True
)

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

	Topic	Count	Name
0	-1	2083	-1_cabernet_flavors_aromas_tannins
1	0	3075	0_wine_cherry_flavors_oak
2	1	541	1_cabernet_tannins_flavors_winery
3	2	398	2_cabernet_drink_flavors_blackberry
4	3	393	3_sauvignon_cabernet_tannins_flavors
5	4	383	4_blackberry_flavors_dry_blackberries
6	5	260	5_cab_flavors_blackberries_ripe
7	6	260	6_napa_tannins_currant_wine
8	7	200	7_nose_palate_herbal_aromas

topic_model.get_topic_info() output (image by author)

Notice we only get 43 topics, and no stop words appear in any of the names. That's because we set the count vectorizer model to remove English stopwords from being counted.

We also set the **diversity** value to 0.5. To reduce the number of synonyms, we can use this parameter to increase the diversity among words to find the most coherent words without leaving a lot of overlap between the words themselves.

Using custom word embeddings

If we've customized our own language model and want to use the embeddings from that, we can pass them into the model directly. In this example, we'll generate

embeddings using Google's Universal Sentence Encoder, available on tensorflow hub.

```
import tensorflow
import tensorflow_hub as hub

#load the universal sentence encoder model
use4 = hub.load("https://tfhub.dev/google/universal-sentence-encoder/4")

#generate embeddings
use4_embeddings = use4(df['description'])
use= np.array(use4_embeddings)

#create list from np arrays to store the embeddings in the dataframe
df['use4'] = use.tolist()

#pass the embeddings into BERTopic
topic_model.fit_transform(docs, use)

#get topic info
topic_model.get_topic_info()
```

	Topic	Count	Name
0	-1	5490	-1_wine_flavors_tannins_oak
1	0	808	0_cabernet_tannins_blackberries_cellar
2	1	734	1_blackberry_flavors_drink_oak
3	2	686	2_herbal_aromas_plum_flavors
4	3	350	3_aromas_spice_lead_flavors
5	4	244	4_bottling_palate_purple_oregano
6	5	225	5_merlot_verdot_cabernet_malbec
7	6	156	6_wine_flavors_soft_tannins

topic_model.get_topic_info() using Universal Sentence Encoder embeddings (image by author)

Overall, the flexibility of the BERTopic model makes it incredibly powerful and easy to use. In just a few lines of code, we went from text documents to topics, and even the out-of-the-box results are good. To maximize the power of BERTopic, review their documentation and play around with the intricacies of the model and

submodels:

Quickstart

Installation, with sentence-transformers, can be done using pypi: You may want to install more depending on the...

maartengr.github.io

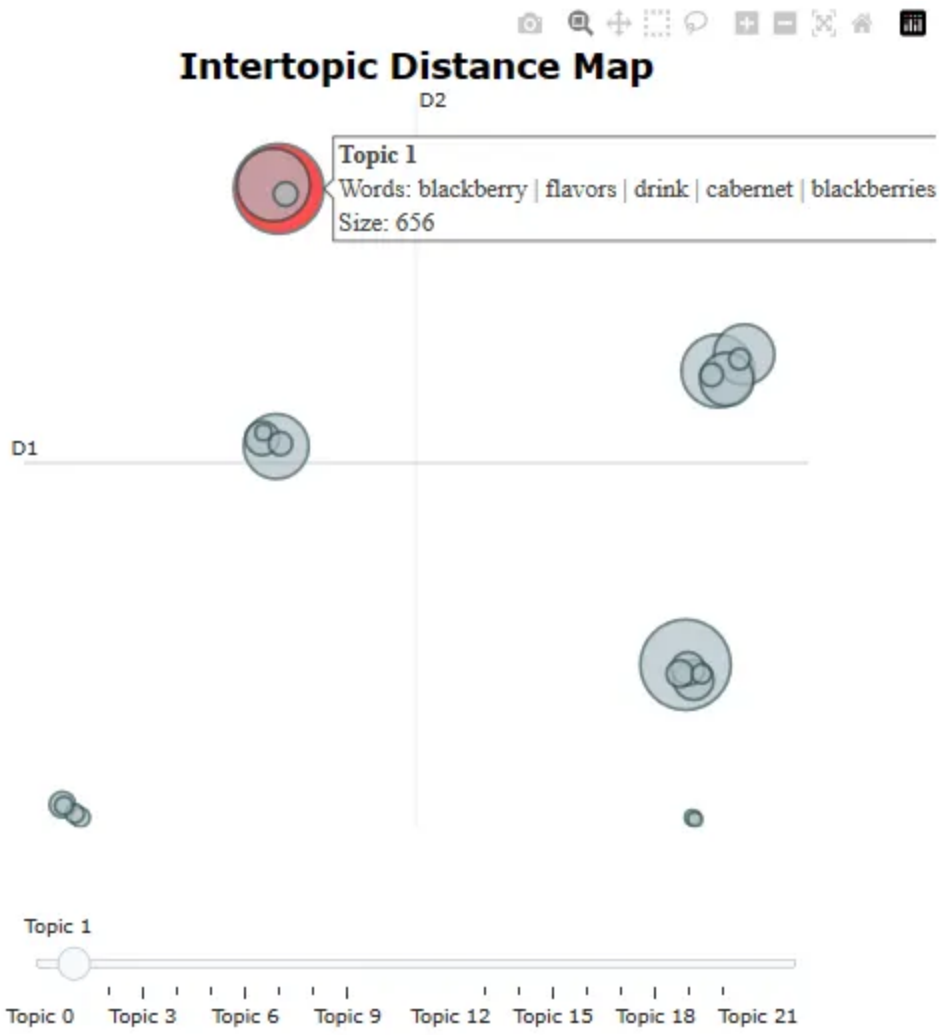
How to explore and visualize a topic model using BERTopic

Now that we have our topics, BERTopic comes with functionality to visualize and explore them. Topic modeling is as much an art as it is a science, which means the model's output needs to be validated to make sure the topics and the words within the topics are a good fit. BERTopic uses Plotly under the hood to output interactive visualizations.

Visualizing topics

Let's start by using the visualize_topics() method to visualize the two-dimensional representations of our topics, making them easier to explore.

```
topic_model.visualize_topics()
```



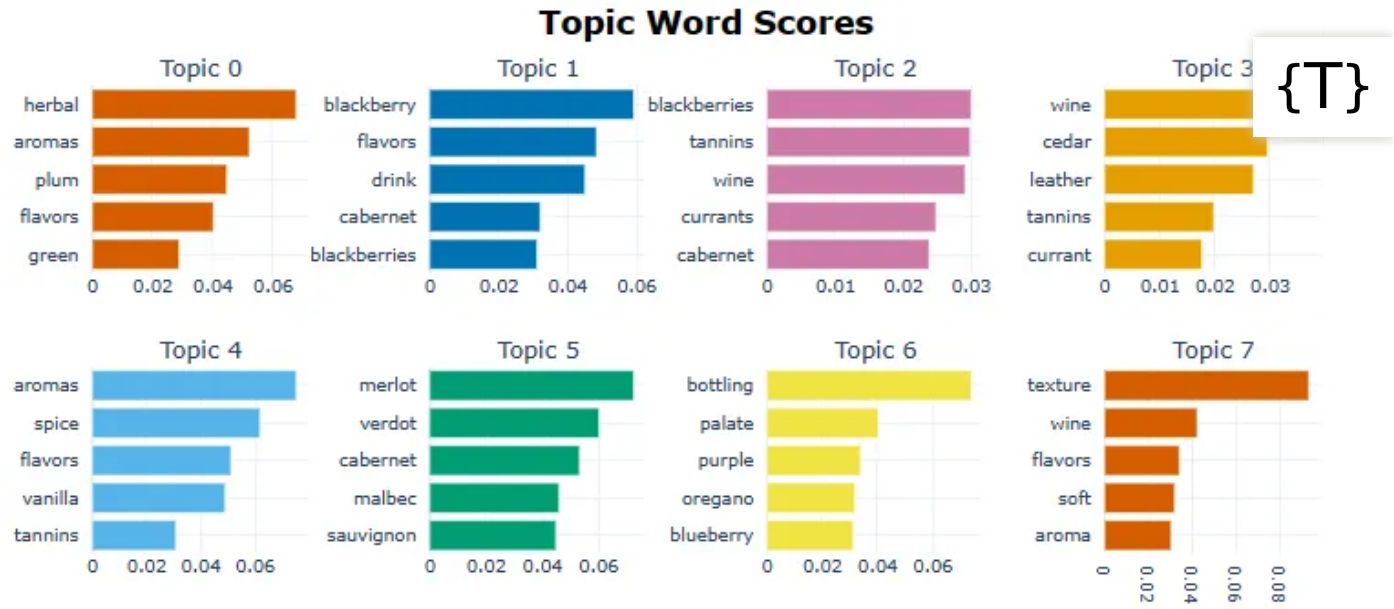
topic_model.visualize_topics() output (image by author)

We can use the slider to select the topic, which will appear in red. Hovering over a topic displays general information about it, including the size of the topic and its corresponding words.

Visualizing word frequency

We can visualize the top terms topics by creating bar charts out of the c-TF-IDF scores. This makes it easier to compare what terms are within each topic so you can easily compare topic representations to each other.

```
topic_model.visualize_barchart(top_n_topics=8)
```

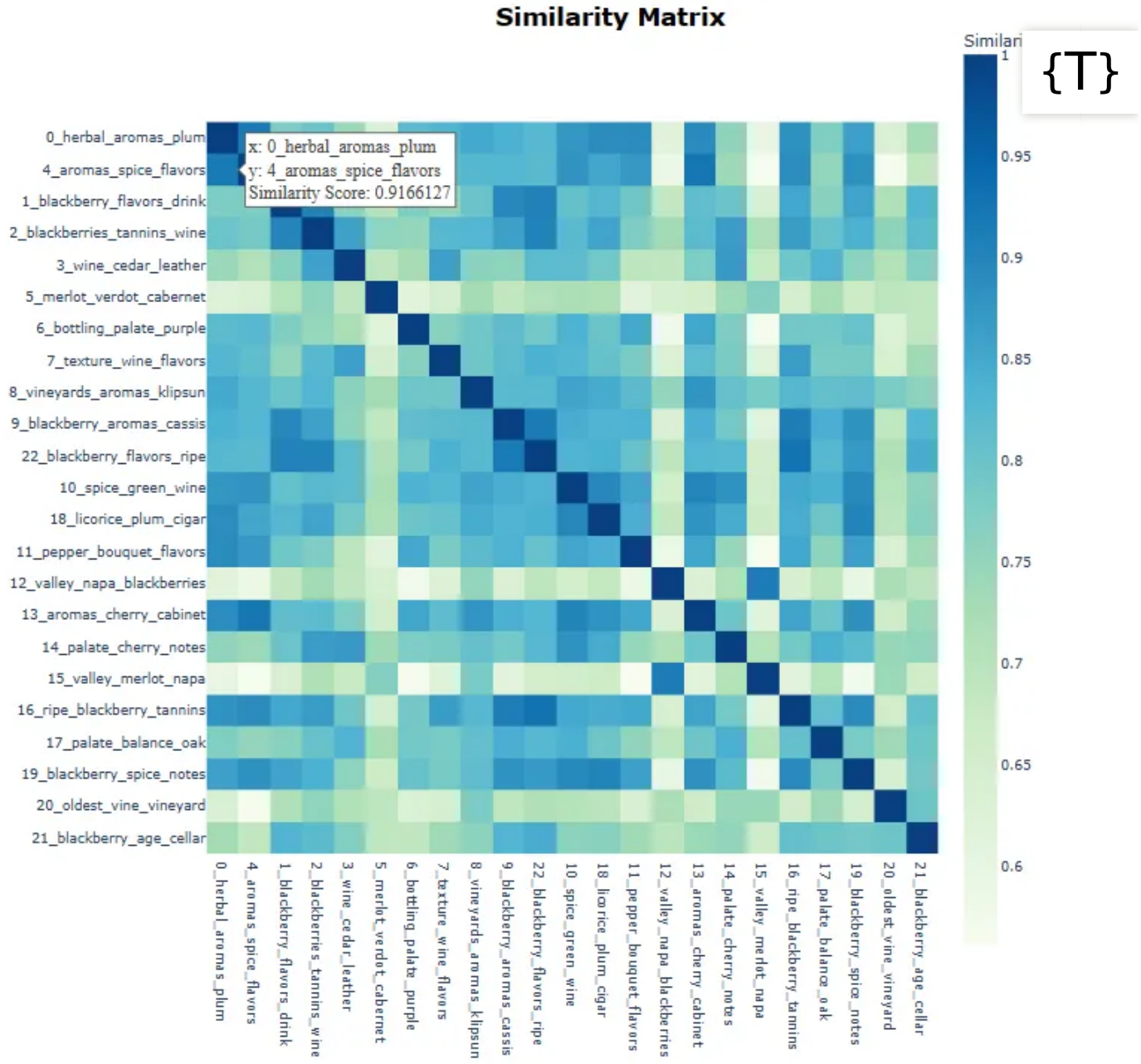


topic_model.visualize_barchart() output (image by author)

Exploring the words, we can begin to see themes within the topics. Notice how many topics share similar words indicating topics might be very similar to each other. One way to validate this suspicion is to use a similarity heatmap.

Visualize topic similarity

We can create a similarity heatmap by simply applying cosine similarities through those topic embeddings. The result will be a matrix indicating how similar certain topics are to each other.



Overall, topics are fairly similar, which could indicate we can reduce the number of topics. We can do this by using the **update_topics()** or the **reduce_topics()** methods.

```
topic_model.update_topics(docs, n_gram_range=(1, 2))
topic_model.reduce_topics(docs, nr_topics=6)
topic_model.visualize_heatmap(n_clusters=5, width=1000, height=1000)
```

When instantiating the BERTopic model, we can pass *n_gram_range* and *nr_topics* as

parameters if we already have an idea as to how many topics we want to produce.

```
topics_model = BERTopic(
    n_gram_range=(1, 2),
    nr_topics=10,
    min_topic_size=15,
    diversity=0.5,
    calculate_probabilities=True)
```

Final Thoughts

BERTopic is one of the cutting-edge NLP tools that lets users transform text into embeddings and extract topics in just a few lines of code. Through topic modeling, we're able to leverage the power of machine learning to automatically parse large volumes of text into topics so we can gain insight into things like online product reviews, customer support issues, medical research, news, and financial reports. As we explored in this tutorial, BERTopic was designed with flexibility and functionality in mind, so it can be used for many different use cases and languages.

Citations

1. Grootendorst, M. (2022). BERTopic: Neural topic modeling with a class-based TF-IDF procedure. *ArXiv*. https://doi.org/10.48550/arXiv.2203.05794

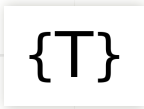
Thank You!

- *If you enjoyed my work, follow me on Medium for more!*
- *Get FULL ACCESS and help support my content by subscribing*!
- *Let's connect on LinkedIn*
- *Analyze Data using Python? Check out my website*!

— Eric Kleppen

Complete code

Find the complete code, along with several other NLP projects, on my GitHub.



GitHub - bendgame/intro_to_bertopic

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or...

github.com

```
import pandas as pd
import numpy as np
from sentence_transformers import SentenceTransformer
from umap import UMAP
from hdbscan import HDBSCAN
from sklearn.feature_extraction.text import CountVectorizer
from bertopic.vectorizers import ClassTfidfTransformer
from bertopic import BERTopic

df = pd.read_csv('winemag-data-130k-v2.csv')
df.head()

df = df.loc[df.variety == 'Cabernet Sauvignon']
df.describe()

docs = df.description.values

#generate topic model using english defaults
topic_model = BERTopic(language="english", calculate_probabilities=True, verbos
topics, probs = topic_model.fit_transform(docs)
topic_model.get_topic_info()

#cusomizing the topic model
# Step 1 - Extract embeddings
embedding_model = SentenceTransformer("all-MiniLM-L6-v2")

# Step 2 - Reduce dimensionality
umap_model = UMAP(n_neighbors=15, n_components=5, min_dist=0.0, metric='cosine'

# Step 3 - Cluster reduced embeddings
hdbscan_model = HDBSCAN(min_cluster_size=15, metric='euclidean', cluster_select

# Step 4 - Tokenize topics
vectorizer_model = CountVectorizer(stop_words="english")
```

```
ctfidf_model = ClassTfidfTransformer()

# All steps together
topic_model = BERTopic(
  embedding_model=embedding_model,    # Step 1 - Extract embeddings
  umap_model=umap_model,              # Step 2 - Reduce dimensionality
  hdbscan_model=hdbscan_model,        # Step 3 - Cluster reduced embeddings
  vectorizer_model=vectorizer_model,  # Step 4 - Tokenize topics
  ctfidf_model=ctfidf_model,          # Step 5 - Extract topic words
  diversity=0.5,                      # Diversify topic words
  calculate_probabilities=True,
  verbose=True
)

#using custom embeddings
import tensorflow
import tensorflow_hub as hub
use4 = hub.load("https://tfhub.dev/google/universal-sentence-encoder/4")

#generate embeddings
use4_embeddings = use4(df['description'])
use = np.array(use4_embeddings)

#create list from np arrays
df['use4'] = use.tolist()

topic_model.fit_transform(docs, use)

#visualize topics
vis
vis                                    top_n_topics=8)
topic_model.visualize_heatmap(n_clusters=20, width=1000, height=1000)

#update the topic model and reduce the topics
topic_model.update_topics(docs, n_gram_range=(1, 2))
topic_model.reduce_topics(docs, nr_topics=6)
```


Thank You!

- *If you enjoyed my work, follow me on Medium for more!*
- *Get FULL ACCESS and help support my content by subscribing*!

- *Let's connect on LinkedIn*
- *Analyze Data using Python? Check out my website!*

— Eric Kleppen

*More content at **PlainEnglish.io**.*

*Sign up for our **free weekly newsletter**. Follow us on **Twitter, LinkedIn, YouTube, and Discord**.*

Build awareness and adoption for your tech startup with Circuit.