[Topic Modeling With Deep Learning Using Python BERTopic](https://grabngoinfo.com/topic-modeling-with-deep-learning-using-python-bertopic/)

By [Amy](https://grabngoinfo.com/author/amy-grabngoinfogmail-com/)/ October 21, 2022

BERTopic is a topic modeling python library that uses the combination of transformer embeddings and clustering model algorithms to identify topics in NLP (Natual Language Processing). In this tutorial, we will talk about:

* How transformers, c-TF-IDF, and clustering models are used behind the BERTopic?
* How to extract and interpret topics from the topic modeling results?
* How to make predictions using topic modeling?
* How to save and load a BERTopic topic model?

This is an introduction to the BERTopic model. To learn how to optimize the BERTopic model, please check out [Hyperparameter Tuning for BERTopic Model in Python](https://medium.com/grabngoinfo/hyperparameter-tuning-for-bertopic-model-in-python-104445778347).

* **Resources for this post:** Video tutorial for this post on [YouTube](https://www.youtube.com/watch?v=Q5g0OnoyC5U&list=PLVppujud2yJpx5r8GFeJ81fyek8dEDMX-&index=4" \t "_blank)
* Click [here](https://mailchi.mp/0410c9c60986/0l89ak45no) for the Colab notebook.: https://colab.research.google.com/drive/1jhP3-UV0DCCh0JvFYv\_8Pwsz1PQ-TkFo?usp=sharing
* More video tutorials on [NLP](https://www.youtube.com/playlist?list=PLVppujud2yJpx5r8GFeJ81fyek8dEDMX-)
* More blog posts on [NLP](https://medium.com/@AmyGrabNGoInfo/list/nlp-49340193610f)
* If you are not a Medium member and want to support me to keep providing free content ( Buy me a cup of coffee ), join Medium membership through [this link](https://medium.com/@AmyGrabNGoInfo/membership). You will get full access to posts on Medium for $5 per month, and I will receive a portion of it. Thank you for your support
* Give me a [tip](https://www.paypal.com/donate/?hosted_button_id=4PZAFYA8GU8JW) to show your appreciation and help me keep providing free content. Thank you for your generosity
* Video tutorial for this post on [YouTube](https://www.youtube.com/watch?v=Q5g0OnoyC5U&list=PLVppujud2yJpx5r8GFeJ81fyek8dEDMX-&index=4)

Topic Modeling with Deep Learning Using Python BERTopic – GrabNGoInfo.com

Let’s get started!

Step 0: BERTopic Model Algorithms

In step 0, we will talk about the algorithms behind the BERTopic model.

* **Documents Embedding**: Firstly, we need to get the embeddings for all the documents. Embeddings are the vector representation of the documents.
  + BERTopic uses the English version of the sentence\_transformers by default to get document embeddings.
  + If there are multiple languages in the document, we can use BERTopic(language=”multilingual”) to support the topic modeling of over 50 languages.
  + BERTopic also supports the pre-trained models from other python packages such as hugging face and flair.
* **Documents Clustering**: After the text documents have been transformed into embeddings, the next step is to run a clustering model on the embedded documents. Because the embedding vectors usually have very high dimensions, dimension reduction techniques are used to reduce the dimensionalities.
  + The default algorithm for dimension reduction is UMAP (Uniform Manifold Approximation & Projection). Compared with other dimension reduction techniques such as PCA (Principle Component Analysis), UMAP maintains the data’s local and global structure when reducing the dimensionality, which is important for representing the semantics of the text data. BERTopic provides the option of using other dimensionality reduction techniques by changing the umap\_model value in the BERTopic method.
  + The default algorithm for clustering is HDBSCAN. HDBSCAN is a density-based clustering model. It identifies the number of clustering automatically, and does not require specifying the number of clusters beforehand like most of the clustering models.
* **Topic Representation**: After assigning each document in the corpus into a cluster, the next step is to get the topic representation using a class-based TF-IDF called c-TF-IDF. The top words with the highest c-TF-IDF scores are selected to represent each topic.
  + c-TF-IDF is similar to TF-IDF in that it measures the term importance by term frequencies while taking into account the whole corpus (all the text data for the analysis).
  + c-TF-IDF is different from TF-IDF in that the term frequency level is different. In the regular TF-IDF, TF measures the term frequency in each document. While in the c-TF-IDF, TF measures the term frequency in each cluster, and each cluster includes many documents.
* **Maximal Marginal Relevance (MMR)** (optional): After extracting the most important terms describing each cluster, there is an optional step to optimize the terms using Maximal Marginal Relevance (MMR). Maximal Marginal Relevance (MMR) has two benefits:
  + The first benefit is to increase the coherence among the terms for the same topic and remove irrelevant terms.
  + The second benefit is to increase the topic representation by removing synonyms and variations of the same words.

Step 1: Install And Import Python Libraries

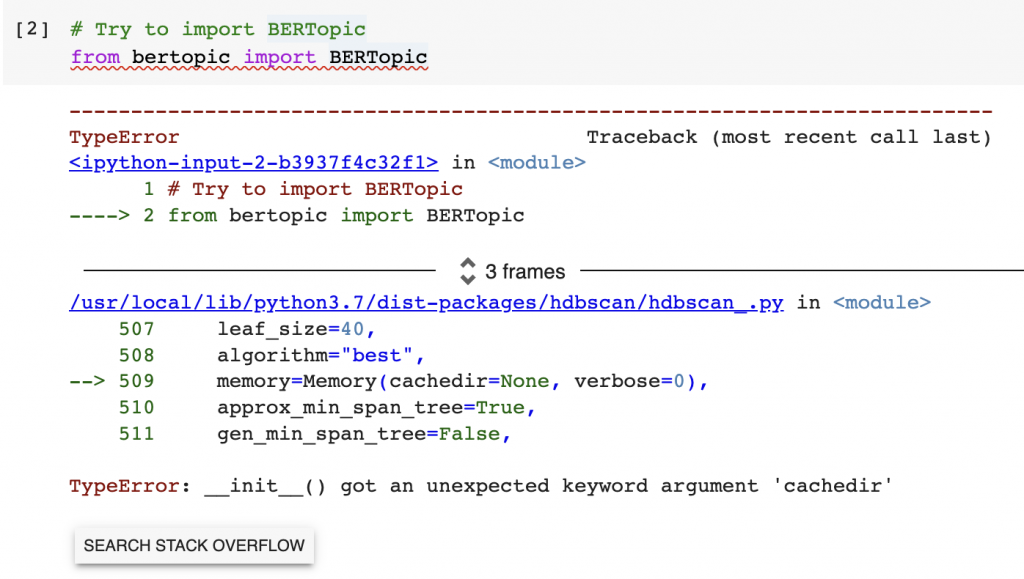
In step 1, we will install and import python libraries.

Firstly, let’s import bertopic.

# Install bertopic

!pip install bertopic

After installing bertopic, when we tried to import the BERTopic method, a type error about an unexpected keyword argument cachedir came up.

BERTopic TypeError on cashedir – GrabNGoInfo.com

This TypeError is caused by the incompatibility between joblib and HDBSCAN. At the time this tutorial was created, joblib has a new release that is not supported by HDBSCAN. HDBSCAN does have a fix for it but has not been rolled out. So if you are watching this tutorial on YouTube or reading this tutorial on Medium.com at a later time, you may not encounter this error message.

Before the incompatibility issue between joblib and HDBSCAN is fixed, we can solve this issue by installing an old version of joblib. In this example, we used joblib version 1.1.0. After installing joblib, we need to restart the runtime.

# Install older version of joblib

!pip install --upgrade joblib==1.1.0

After installing the python packages, we will import the python libraries.

* pandas and numpy are imported for data processing.
* nltk is imported for text preprocessing. We downloaded the information for removing stopwords and lemmatization from nltk.
* BERTopic is imported for the topic modeling.
* UMAP is for dimension reduction.

# Data processing

import pandas as pd

import numpy as np

# Text preprocessiong

import nltk

nltk.download**(**'stopwords'**)**

nltk.download**(**'omw-1.4'**)**

nltk.download**(**'wordnet'**)**

wn = nltk.WordNetLemmatizer**()**

# Topic model

from bertopic import BERTopic

# Dimension reduction

from umap import UMAP

Step 2: Download And Read Data

The second step is to download and read the dataset.

The UCI Machine Learning Repository has the review data from three websites: imdb.com, amazon.com, and yelp.com. We will use the review data from amazon.com for this tutorial. Please follow these steps to download the data.

1. Go to: <https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences>
2. Click “Data Folder”
3. Download “sentiment labeled sentences.zip”
4. Unzip “sentiment labeled sentences.zip”
5. Copy the file “amazon\_cells\_labelled.txt” to your project folder

Those who are using Google Colab for this analysis need to mount Google Drive to read the dataset. You can ignore the code below if you are not using Google Colab.

* drive.mount is used to mount to the Google drive so the colab notebook can access the data on the Google drive.
* os.chdir is used to change the default directory on Google drive. I set the default directory to the folder where the review dataset is saved.
* !pwd is used to print the current working directory.

Please check out [Google Colab Tutorial for Beginners](https://medium.com/towards-artificial-intelligence/google-colab-tutorial-for-beginners-834595494d44) for details about using Google Colab for data science projects.

# Mount Google Drive

from google.colab import drive

drive.mount**(**'/content/drive'**)**

# Change directory

import os

os.chdir**(**"drive/My Drive/contents/nlp"**)**

# Print out the current directory

!pwd

Output:

Mounted at /content/drive

/content/drive/My Drive/contents/nlp

Now let’s read the data into a pandas dataframe and see what the dataset looks like.

The dataset has two columns. One column contains the reviews and the other column contains the sentiment label for the review. Since this tutorial is for topic modeling, we will not use the sentiment label column, so we removed it from the dataset.

# Read in data

amz\_review = pd.read\_csv**(**'sentiment labelled sentences/amazon\_cells\_labelled.txt', sep='\t', names=**[**'review', 'label'**])**

# Drop te label

amz\_review = amz\_review.drop**(**'label', axis=1**)**;

# Take a look at the data

amz\_review.head**()**

.info helps us to get information about the dataset.

# Get the dataset information

amz\_review.info**()**

From the output, we can see that this data set has 1000 records, and no missing data. The ‘review’ column is the object type.

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1000 entries, 0 to 999

Data columns (total 1 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 review 1000 non-null object

dtypes: object(1)

memory usage: 7.9+ KB

Step 3: Text Data Preprocessing (Optional)

In step 3, we included some sample code for text data preprocessing.

Generally speaking, there is no need to preprocess the text data when using the python BERTopic model. However, since our dataset is a simple dataset, a lot of stopwords are picked to represent the topics.

Therefore, we removed stopwords and did lemmatization as data preprocessing. But please ignore this step if this is not an issue for you.

# Remove stopwords

stopwords = nltk.corpus.stopwords.words**(**'english'**)**

print**(**f'There are {len(stopwords)} default stopwords. They are {stopwords}'**)**

There are 179 default stopwords in the nltk library. We printed all the stopwords out to see what words are considered to be stopwords. nltk provides the option to add customized stopwords to the list.

There are 179 default stopwords. They are ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]

Lemmatization refers to changing words to their base form.

After removing stopwords and lemmatizing the words we can see that the stopwords like to and for are removed, and the word like conversations is converted to conversation.

# Remove stopwords

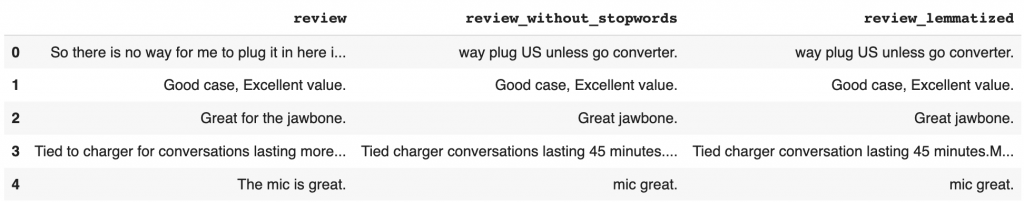
amz\_review**[**'review\_without\_stopwords'**]** = amz\_review**[**'review'**]**.apply**(**lambda x: ' '.join**([**w for w in x.split**()** if w.lower**()** not in stopwords**]))**

# Lemmatization

amz\_review**[**'review\_lemmatized'**]** = amz\_review**[**'review\_without\_stopwords'**]**.apply**(**lambda x: ' '.join**([**wn.lemmatize**(**w**)** for w in x.split**()** if w not in stopwords**]))**

# Take a look at the data

amz\_review.head**()**

Text data pre-processing – GrabNGoInfo.com

Step 4: Topic Modeling Using BERTopic

In step 4, we will build the topic model using BERTopic.

BERTopic model by default produces different results each time because of the stochasticity inherited from UMAP.

To get reproducible topics, we need to pass a value to the random\_state parameter in the UMAP method.

* n\_neighbors=15 means that the local neighborhood size for UMAP is 15. This is the parameter that controls the local versus global structure in data.
  + A low value forces UMAP to focus more on local structure, and may lose insights into the big picture.
  + A high value pushes UMAP to look at the broader neighborhood, and may lose details on local structure.
  + The default n\_neighbors values for UMAP is 15.
* n\_components=5 indicates that the target dimension from UMAP is 5. This is the dimension of data that will be passed into the clustering model.
* min\_dist controls how tightly UMAP is allowed to pack points together. It’s the minimum distance between points in the low dimensional space.
  + Small values of min\_dist result in clumpier embeddings, which is good for clustering. Since our goal of dimension reduction is to build clustering models, we set min\_dist to 0.
  + Large values of min\_dist prevent UMAP from packing points together and preserves the broad structure of data.
* metric='cosine' indicates that we will use cosine to measure the distance.
* random\_state sets a random seed to make the UMAP results reproducible.

After initiating the UMAP model, we pass it to the BERTopic model, set the language to English, and set the calculate\_probabilities parameter to True.

Finally, we pass the processed review documents to the topic model and saved the results for topics and topic probabilities.

* The values in topics represents the topic each document is assigned to.
* The values in probabilities represents the probability of a document belongs to each of the topics.

# Initiate UMAP

umap\_model = UMAP**(**n\_neighbors=15,

n\_components=5,

min\_dist=0.0,

metric='cosine',

random\_state=100**)**

# Initiate BERTopic

topic\_model = BERTopic**(**umap\_model=umap\_model, language="english", calculate\_probabilities=True**)**

# Run BERTopic model

topics, probabilities = topic\_model.fit\_transform**(**amz\_review**[**'review\_lemmatized'**])**

Step 5: Extract Topics From Topic Modeling

In step 5, we will extract topics from the BERTopic modeling results.

Using the attribute get\_topic\_info() on the topic model gives us the list of topics. We can see that the output gives us 31 rows in total.

* Topic -1 should be ignored. It indicates that the reviews are not assigned to any specific topic. The count for topic -1 is 322, meaning that there are 322 reviews as outliers that do not belong to any topic.
* Topic 0 to topic 29 are the 30 topics created for the reviews. It was ordered by the number of reviews in each topic, so topic 0 has the highest number of reviews.
* The Name column lists the top terms for each topic. For example, the top 4 terms for Topic 0 are sound, quality, volume, and audio, indicating that it is a topic related to sound quality.

# Get the list of topics

topic\_model.get\_topic\_info**()**

Topic Modeling results from BERTopic – GrabNGoInfo.com

If more than 4 terms are needed for a topic, we can use get\_topic and pass in the topic number. For example, get\_topic(0) gives us the top 10 terms for topic 0 and their relative importance.

# Get top 10 terms for a topic

topic\_model.get\_topic**(**0**)**

Output:

[('sound', 0.1060322237741523),

('quality', 0.06904479135165552),

('volume', 0.05915482066025614),

('audio', 0.046799811254827524),

('poor', 0.04253208080983699),

('loud', 0.04078174318539755),

('hear', 0.03943654710683742),

('talk', 0.03746480989684604),

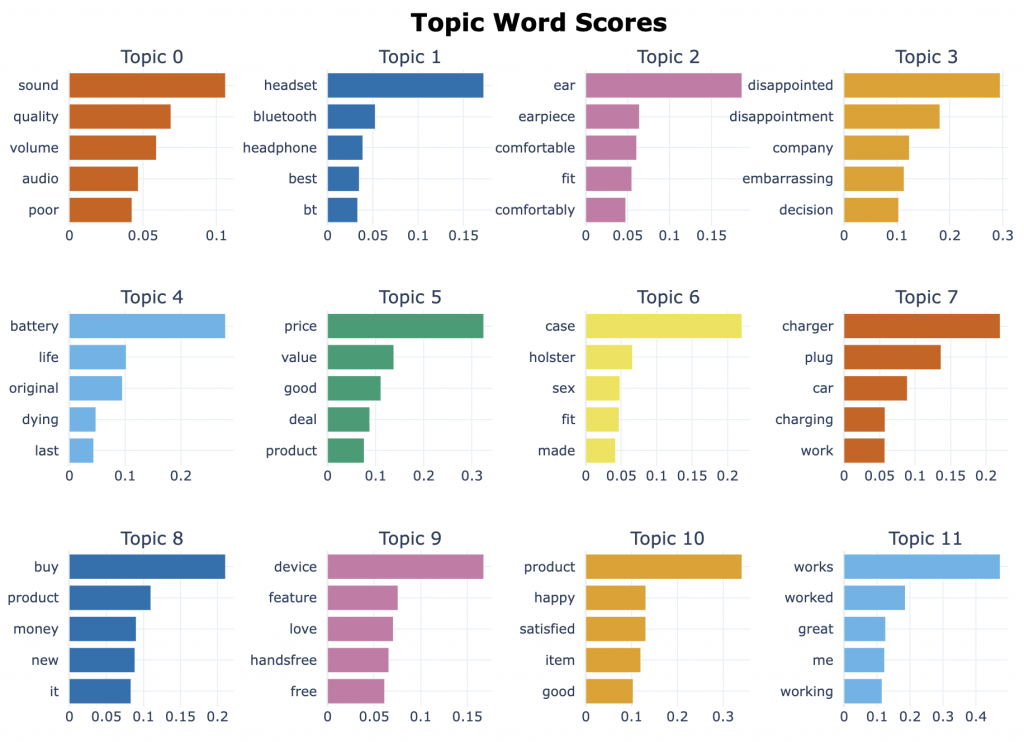
('low', 0.036118244310613924),

('clear', 0.03286378925569785)]

We can visualize the top keywords using a bar chart. top\_n\_topics=12 means that we will create bar charts for the top 12 topics. The length of the bar represents the score of the keyword. A longer bar means higher importance for the topic.

# Visualize top topic keywords

topic\_model.visualize\_barchart**(**top\_n\_topics=12**)**

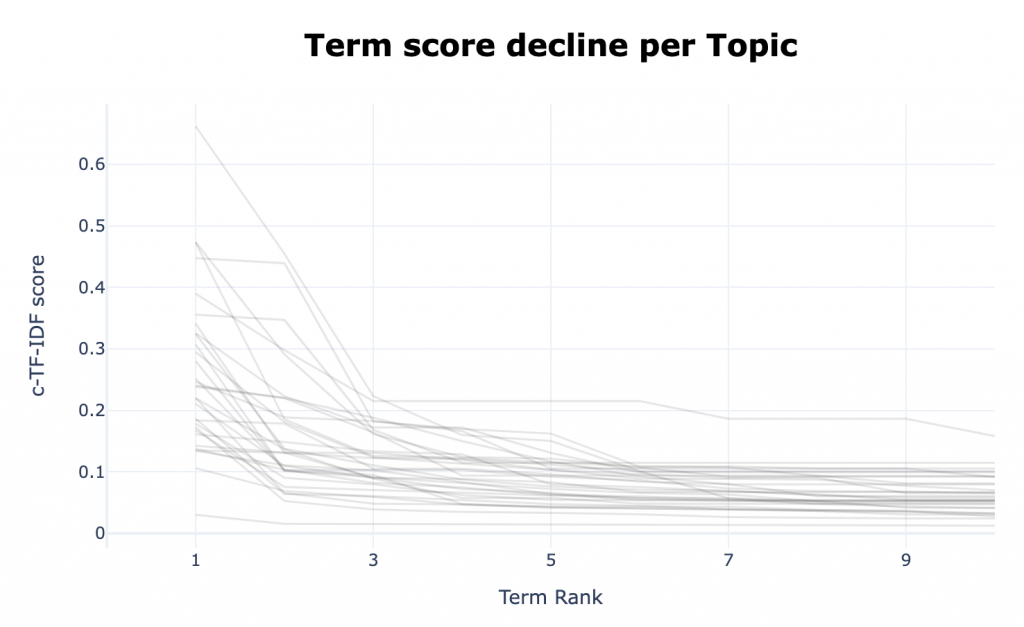
Topic model top keywords visualization – GrabNGoInfo.com

Another view for keyword importance is the “Term score decline per topic” chart. It’s a line chart with the term rank being the x-axis and the c-TF-IDF score on the y-axis.

There are a total of 31 lines, one line for each topic. Hovering over the line shows the term score information.

# Visualize term rank decrease

topic\_model.visualize\_term\_rank**()**

Topic model term score decline per topic – GrabNGoInfo.com

Step 6: Topic Similarities

In step 6, we will analyze the relationship between the topics generated by the topic model.

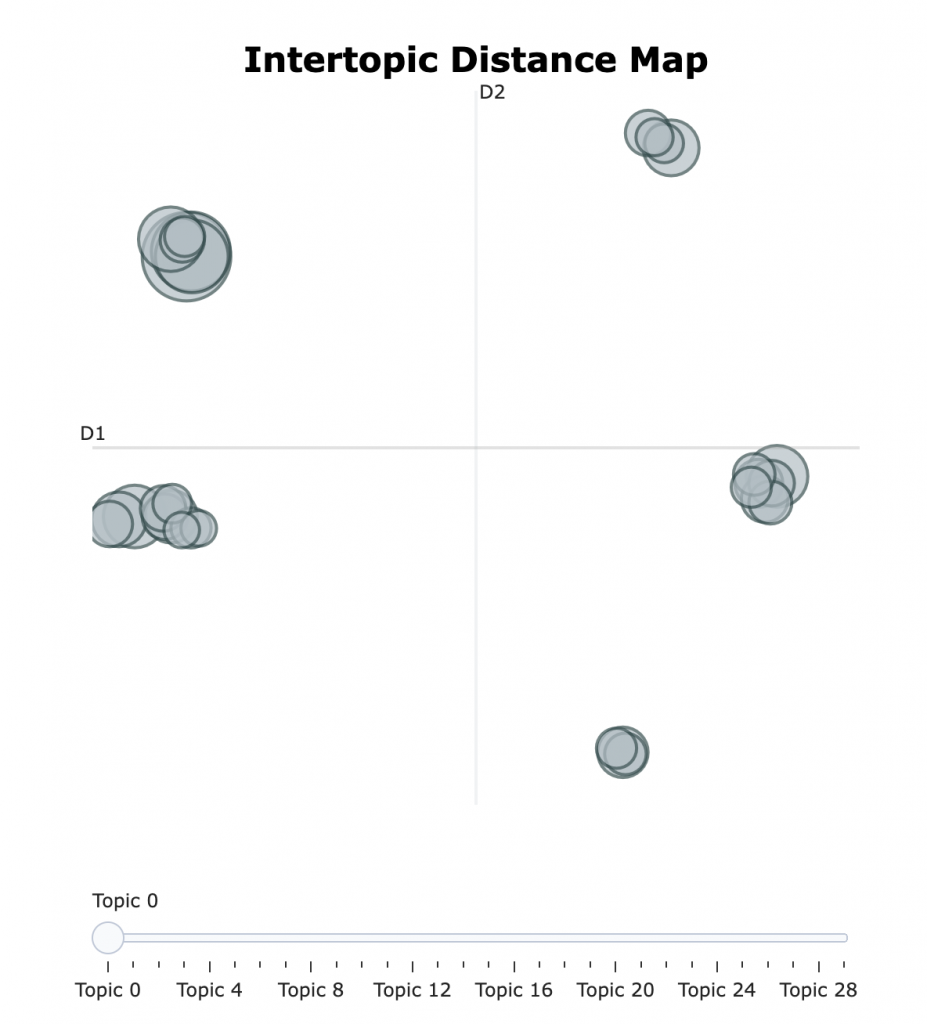
We will use three visualizations to study how the topics are related to each other. The three methods are intertopic distance map, the hierarchical clustering of topics, and the topic similarity matrix.

Intertopic distance map measures the distance between topics. Similar topics are closer to each other, and very different topics are far from each other. From the visualization, we can see that there are five topic groups for all the topics. Topics with similar semantic meanings are in the same topic group.

The size of the circle represents the number of documents in the topics, and larger circles mean that more reviews belong to the topic.

# Visualize intertopic distance

topic\_model.visualize\_topics**()**

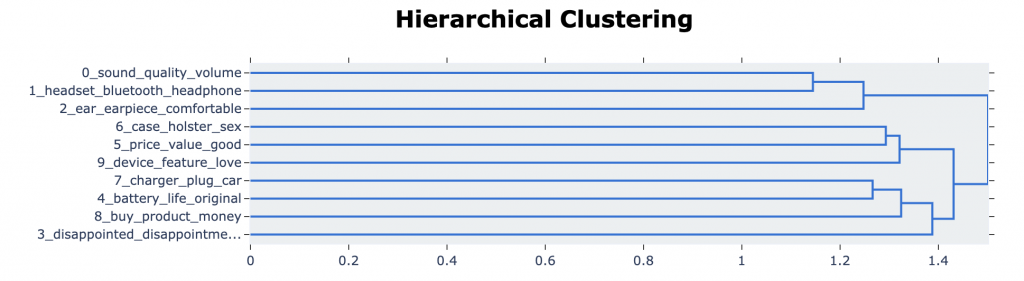
Topic modeling visualization about topic distance – GrabNGoInfo.com

Another way to see how the topics are connected is through a hierarchical clustering graph. We can control the number of topics in the graph by the top\_n\_topics parameter.

In this example, the top 10 topics are included in the hierarchical graph. We can see that the sound quality topic is closely connected to the headset topic, and both of them are connected to the earpiece comfortable topic.

# Visualize connections between topics using hierachical clustering

topic\_model.visualize\_hierarchy**(**top\_n\_topics=10**)**

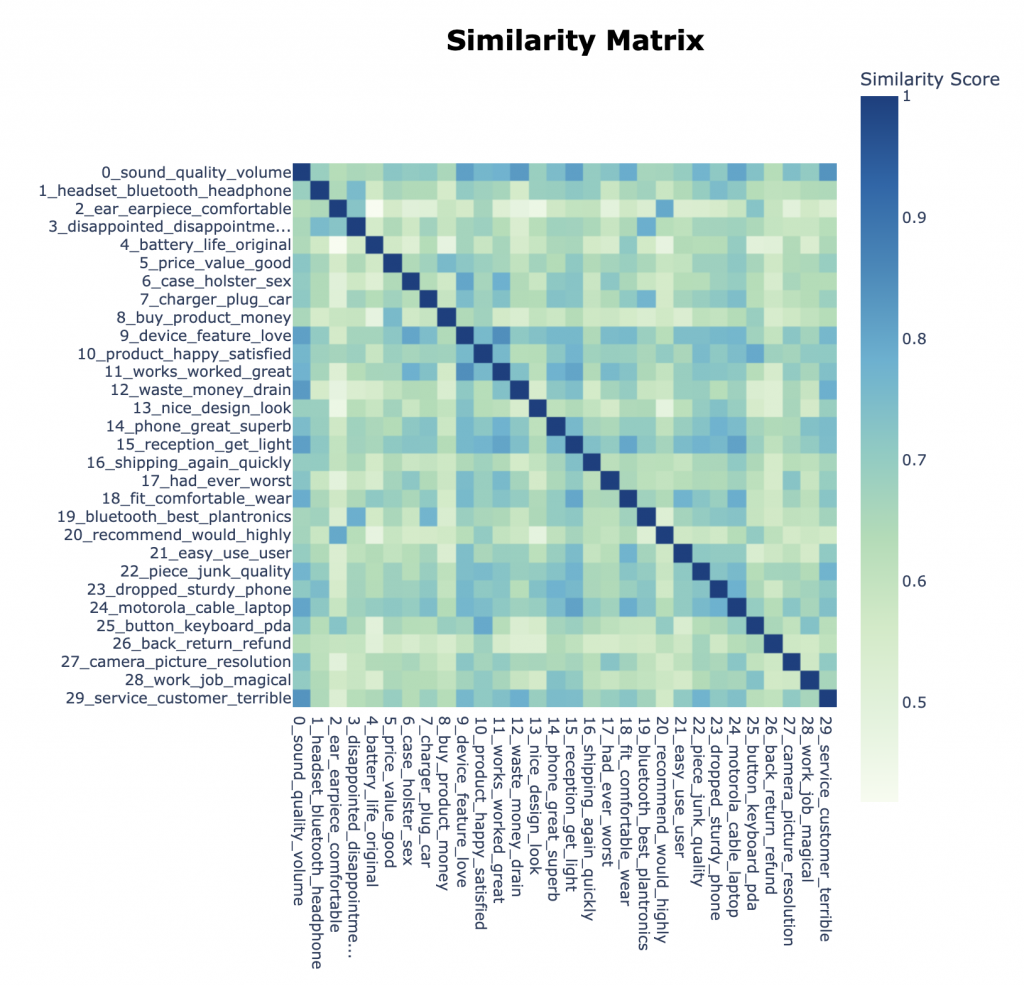


Topic modeling hierarchical clustering – GrabNGoInfo.com

Heatmap can also be used to analyze the similarities between topics. The similarity score ranges from 0 to 1. A value close to 1 represents a higher similarity between the two topics, which is represented by darker blue color.

# Visualize similarity using heatmap

topic\_model.visualize\_heatmap**()**



Topic model topic similarity heatmap – GrabNGoInfo.com

Step 7: Topic Model Predicted Probabilities

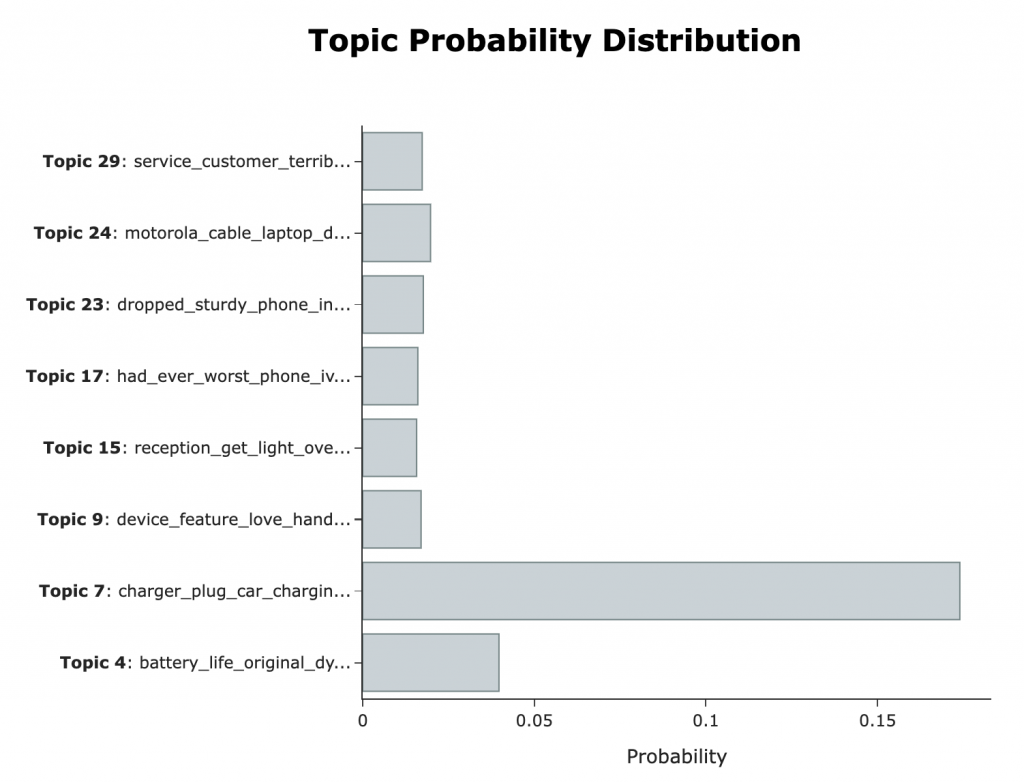
In step 7, we will talk about how to use BERTopic model to get predicted probabilities.

The topic prediction for a document is based on the predicted probabilities of the document belonging to each topic. The topic with the highest probability is the predicted topic. This probability represents how confident we are about finding the topic in the document.

We can visualize the probabilities using visualize\_distribution, and pass in the document index. visualize\_distribution has the default probability threshold of 0.015, so only the topic with a probability greater than 0.015 will be included.

# Visualize probability distribution

topic\_model.visualize\_distribution**(**topic\_model.probabilities\_**[**0**]**, min\_probability=0. 015**)**

Topic modeling topic probability distribution – GrabNGoInfo.com

If you would like to save the visualization as a separate html file, we can save the chart into a variable and use write\_html to write the chart into a file.

# Save the chart to a variable

chart = topic\_model.visualize\_distribution**(**topic\_model.probabilities\_**[**0**])**

# Write the chart as a html file

chart.write\_html**(**"amz\_review\_topic\_probability\_distribution.html"**)**

The topic probability distribution for the first review in the dataset shows that topic 7 has the highest probability, so topic 7 is the predicted topic.

The first review is “So there is no way for me to plug it in here in the US unless I go by a converter.”, and the topic of plugging a charger is pretty relevant.

# Check the content for the first review

amz\_review**[**'review'**][**0**]**

Output:

So there is no way for me to plug it in here in the US unless I go by a converter.

We can also get the predicted probability for all topics using the code below.

# Get probabilities for all topics

topic\_model.probabilities\_**[**0**]**

We can see that there are 30 probability values, one for each topic. The index 7 has the highest value, indicating that topic 7 is the predicted topic.

array([0.01334882, 0.01113644, 0.00977354, 0.0112337 , 0.03971048,

0.00935348, 0.01189061, 0.17397613, 0.012551 , 0.01700506,

0.00939692, 0.0093238 , 0.0096451 , 0.01237214, 0.01308384,

0.01570936, 0.01239052, 0.01606719, 0.01109378, 0.01482075,

0.01097928, 0.01400667, 0.01405934, 0.01766433, 0.01976593,

0.01361063, 0.01399875, 0.01216231, 0.01027492, 0.01733308])

Step 8: Topic Model In-Sample Predictions

In step 8, we will talk about how to make in-sample predictions using the topic model.

BERTopic model can output the predicted topic for each review in the dataset.

Using .topics\_, we save the predicted topics in a list and then save it as a column in the review dataset.

# Get the topic predictions

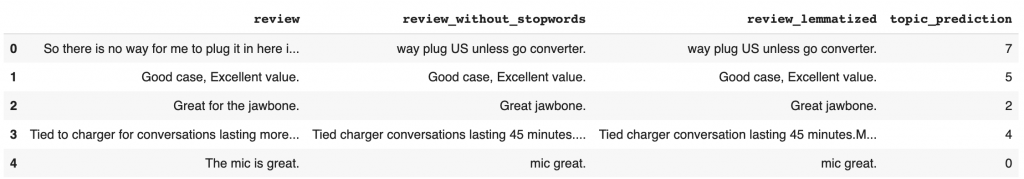
topic\_prediction = topic\_model.topics\_**[**:**]**

# Save the predictions in the dataframe

amz\_review**[**'topic\_prediction'**]** = topic\_prediction

# Take a look at the data

amz\_review.head**()**

Topic modeling topic predictions using BERTopic – GrabNGoInfo.com

Step 9: Topic Model Predictions On New Data

In step 9, we will talk about how to use the BERTopic model to make predictions on new reviews.

Let’s say there is a new review “I like the new headphone. Its sound quality is great.”, and we would like to automatically predict the topic for this review.

* Firstly, let’s decide the number of topics to include in the prediction.
  + If we would like to assign only one topic to the document, then the number of topics should be 1.
  + If we would like to assign multiple topics to the document, then the number of topics should be greater than 1. Here we are getting the top 3 topics that are most relevant to the new review.
* After that, we pass the new review and the number of topics to the find\_topics method. This gives us the topic number and the similarity value.
* Finally, the results are printed. The top 3 similar topics for the new review are topic 1, topic 0, and topic 2. The similarities are 0.43, 0.34, and 0.30.

# New data for the review

new\_review = "I like the new headphone. Its sound quality is great."

# Find topics

num\_of\_topics = 3

similar\_topics, similarity = topic\_model.find\_topics**(**new\_review, top\_n=num\_of\_topics**)**;

# Print results

print**(**f'The top {num\_of\_topics} similar topics are {similar\_topics}, and the similarities are {np.round(similarity,2)}'**)**

Output:

The top 3 similar topics are [1, 0, 2], and the similarities are [0.43 0.34 0.3 ]

To verify if the assigned topics are a good fit for the new review, let’s get the top keywords for the top 3 topics using the get\_topic method.

# Print the top keywords for the top similar topics

for i in range**(**num\_of\_topics**)**:

print**(**f'The top keywords for topic {similar\_topics[i]} are:'**)**

print**(**topic\_model.get\_topic**(**similar\_topics**[**i**]))**

We can see that topic 1 is about headsets and topic 0 is about sound quality. Both topics are a good fit for the new review. Topic 2 is about the earpiece, which is similar to the headset. From this example, we can see that the BERTopic model made good predictions on the new document.

The top keywords for topic 1 are:

[('headset', 0.17221714572056337), ('bluetooth', 0.052588262695443776), ('headphone', 0.03889073797181146), ('best', 0.03501700292623107), ('bt', 0.03324079437966619), ('sound', 0.03115016718968907), ('looking', 0.026515653931063962), ('ever', 0.025198382714548315), ('logitech', 0.02433335225259895), ('wired', 0.02433335225259895)]

The top keywords for topic 0 are:

[('sound', 0.1060322237741523), ('quality', 0.06904479135165552), ('volume', 0.05915482066025614), ('audio', 0.046799811254827524), ('poor', 0.04253208080983699), ('loud', 0.04078174318539755), ('hear', 0.03943654710683742), ('talk', 0.03746480989684604), ('low', 0.036118244310613924), ('clear', 0.03286378925569785)]

The top keywords for topic 2 are:

[('ear', 0.1862713736216058), ('earpiece', 0.0639723645453697), ('comfortable', 0.060666083611044856), ('fit', 0.05484820578989847), ('comfortably', 0.04759234093005887), ('easily', 0.04486455038754426), ('jabra', 0.039179500255761245), ('one', 0.036361948996750736), ('sound', 0.03080141923637862), ('stay', 0.030588591368750903)]

Step 10: Save And Load Topic Models

In step 10, we will talk about how to save and load BERTopic models.

The trained BERTopic model and its settings can be saved using .save. UMAP and HDBSCAN are saved, but the documents and embeddings are not saved.

We can use .load to load the saved BERTopic model.

# Save the topic model

topic\_model.save**(**"amz\_review\_topic\_model"**)**

# Load the topic model

my\_model = BERTopic.load**(**"amz\_review\_topic\_model"**)**

For more information about data science and machine learning, please check out my [YouTube channel](https://youtube.com/@grabngoinfo) and [Medium Page](https://medium.com/@AmyGrabNGoInfo) or follow me on [LinkedIn](https://www.linkedin.com/company/grabngoinfo/).

Recommended Tutorials

* [GrabNGoInfo Machine Learning Tutorials Inventory](https://medium.com/grabngoinfo/grabngoinfo-machine-learning-tutorials-inventory-9b9d78ebdd67)
* [Hyperparameter Tuning for BERTopic Model in Python](https://medium.com/grabngoinfo/hyperparameter-tuning-for-bertopic-model-in-python-104445778347)
* [Hierarchical Topic Model for Airbnb Reviews](https://medium.com/@AmyGrabNGoInfo/hierarchical-topic-model-for-airbnb-reviews-f772eaa30434)
* [5 Ways for Deciding Number of Clusters](https://medium.com/grabngoinfo/5-ways-for-deciding-number-of-clusters-in-a-clustering-model-5db993ea5e09)
* [4 Clustering Model Algorithms in Python](https://medium.com/grabngoinfo/4-clustering-model-algorithms-in-python-and-which-is-the-best-7f3431a6e624)
* [Google Colab Tutorial for Beginners](https://medium.com/towards-artificial-intelligence/google-colab-tutorial-for-beginners-834595494d44)
* [Hyperparameter Tuning For XGBoost](https://medium.com/p/hyperparameter-tuning-for-xgboost-91449869c57e)
* [Time Series Causal Impact Analysis in Python](https://medium.com/grabngoinfo/time-series-causal-impact-analysis-in-python-63eacb1df5cc)
* [3 Ways for Multiple Time Series Forecasting Using Prophet in Python](https://medium.com/p/3-ways-for-multiple-time-series-forecasting-using-prophet-in-python-7a0709a117f9)
* [Four Oversampling And Under-Sampling Methods For Imbalanced Classification Using Python](https://medium.com/p/four-oversampling-and-under-sampling-methods-for-imbalanced-classification-using-python-7304aedf9037)
* [Time Series Anomaly Detection Using Prophet in Python](https://medium.com/grabngoinfo/time-series-anomaly-detection-using-prophet-in-python-877d2b7b14b4)
* [Five Ways To Create Tables In Databricks](https://medium.com/grabngoinfo/five-ways-to-create-tables-in-databricks-cd3847cfc3aa)
* [Explainable S-Learner Uplift Model Using Python Package CausalML](https://medium.com/grabngoinfo/explainable-s-learner-uplift-model-using-python-package-causalml-a3c2bed3497c)
* [One-Class SVM For Anomaly Detection](https://medium.com/p/one-class-svm-for-anomaly-detection-6c97fdd6d8af)
* [Recommendation System: Item-Based Collaborative Filtering](https://medium.com/grabngoinfo/recommendation-system-item-based-collaborative-filtering-f5078504996a)
* [Hyperparameter Tuning for Time Series Causal Impact Analysis in Python](https://medium.com/grabngoinfo/hyperparameter-tuning-for-time-series-causal-impact-analysis-in-python-c8f7246c4d22)
* [Hyperparameter Tuning and Regularization for Time Series Model Using Prophet in Python](https://medium.com/grabngoinfo/hyperparameter-tuning-and-regularization-for-time-series-model-using-prophet-in-python-9791370a07dc)
* [Multivariate Time Series Forecasting with Seasonality and Holiday Effect Using Prophet in Python](https://medium.com/p/multivariate-time-series-forecasting-with-seasonality-and-holiday-effect-using-prophet-in-python-d5d4150eeb57)
* [LASSO (L1) Vs Ridge (L2) Vs Elastic Net Regularization For Classification Model](https://medium.com/towards-artificial-intelligence/lasso-l1-vs-ridge-l2-vs-elastic-net-regularization-for-classification-model-409c3d86f6e9)
* [S Learner Uplift Model for Individual Treatment Effect and Customer Segmentation in Python](https://medium.com/grabngoinfo/s-learner-uplift-model-for-individual-treatment-effect-and-customer-segmentation-in-python-9d410746e122)
* [How to Use R with Google Colab Notebook](https://medium.com/p/how-to-use-r-with-google-colab-notebook-610c3a2f0eab)

References

* [BERTopic GitHub](https://github.com/MaartenGr/BERTopic)
* [Documentation on BERTopic algorithms](https://maartengr.github.io/BERTopic/algorithm/algorithm.html#visual-overview)
* [UMAP documentation](https://umap-learn.readthedocs.io/en/latest/parameters.html)