[**Skip to content**](https://grabngoinfo.com/hyperparameter-tuning-for-bertopic-model-in-python/#content)

[](https://grabngoinfo.com/)

* [Home](https://grabngoinfo.com/)
* [Tutorials](https://grabngoinfo.com/tutorials/)
* [Contact](https://grabngoinfo.com/contact-me/)

Hyperparameter Tuning For BERTopic Model In Python

By [Amy](https://grabngoinfo.com/author/amy-grabngoinfogmail-com/)/ October 27, 2022

Hyperparameter tuning is an important optimization step for building a good topic model. BERTopic is a topic modeling python library that combines transformer embeddings and clustering model algorithms to identify topics in NLP (Natual Language Processing). In this tutorial, we will talk about the following:

* What are the hyperparameters for the BERTopic model?
* How to tune the hyperparameters for the topic model?
* How to compare the results with different hyperparameter values?

Please check out my previous tutorial [Topic Modeling with Deep Learning Using Python BERTopic](https://medium.com/grabngoinfo/topic-modeling-with-deep-learning-using-python-bertopic-cf91f5676504) for an introduction to BERTopic.

**Resources for this post:**

* Click [here](https://mailchi.mp/3820554f2923/q5xk85kkq0) for the Colab notebook.
* 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
* Video tutorial for this post on [YouTube](https://www.youtube.com/watch?v=PbX4nHEOe0o&list=PLVppujud2yJpx5r8GFeJ81fyek8dEDMX-&index=5)

Hyperparameter Tuning for BERTopic Model in Python – GrabNGoInfo.com

Let’s get started!

Step 1: Install And Import Python Libraries

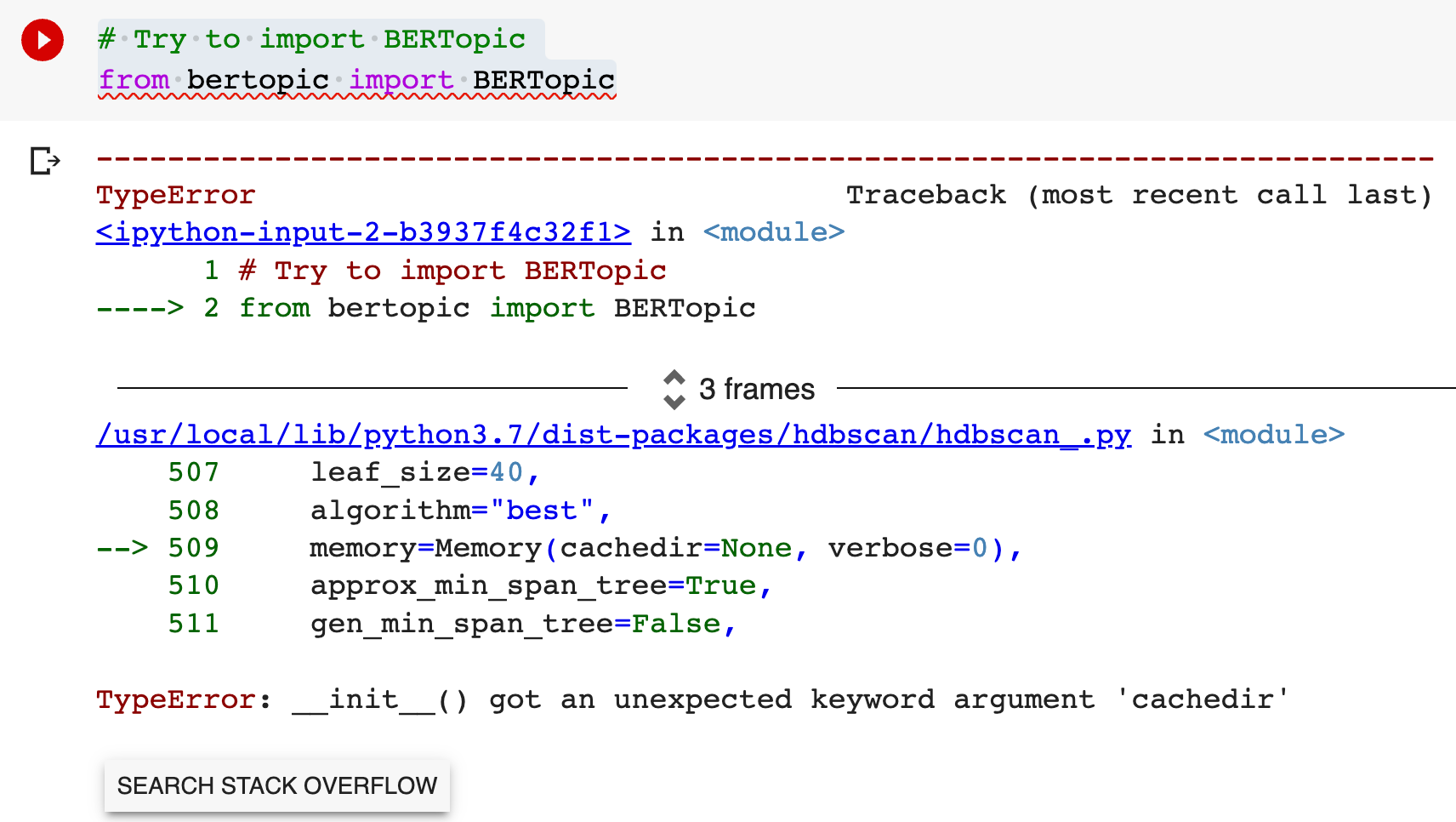
In step 1, we will install and import python libraries.

Firstly, let’s import bertopic and flair.

# Install bertopic  
!pip install bertopic flair

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

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.

Import BERTopic TypeError — GrabNGoInfo.com

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.
* UMAP and PCA are for dimension reduction.
* HDBSCAN and KMeans are for clustering models.
* CountVectorizer is for term vectorization.
* sentence\_transformers and flair are for pretrained document embeddings.
* BERTopic is for the topic modeling.

# Data processing  
import pandas as pd  
import numpy as np

# Dimension reduction  
from umap import UMAP  
from sklearn.decomposition import PCA

# Clustering  
from hdbscan import HDBSCAN  
from sklearn.cluster import KMeans

# Count vectorization  
from sklearn.feature\_extraction.text import CountVectorizer

# Sentence transformer  
from sentence\_transformers import SentenceTransformer

# Flair  
from transformers.pipelines import pipeline  
from flair.embeddings import TransformerDocumentEmbeddings, WordEmbeddings, DocumentPoolEmbeddings, StackedEmbeddings

# Topic model  
from bertopic import BERTopic

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.

* Go to: <https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences>
* Click “Data Folder”
* Download “sentiment labeled sentences.zip”
* Unzip “sentiment labeled sentences.zip”
* 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

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.

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

# Get the dataset information  
amz\_review.info()

Output:

<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: Hyperparameters For Dimensionality Reduction

In step 3, we will talk about the hyperparameters for dimensionality reduction in BERTopic.

Dimensionality reduction is necessary because the clustering model works better for low-dimension data than high-dimension data. The document embeddings usually have hundreds of dimensions, so we need to reduce the dimensionality before passing the embeddings to a clustering model.

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. The UMAP model accepts customized hyperparameters.

* n\_neighbors is the local neighborhood size for UMAP. This is the parameter that controls the local versus global structure in data.

1. A low value forces UMAP to focus more on the local structure and may lose insights into the big picture.
2. A high value pushes UMAP to look at the broader neighborhoods and may lose details on local structure. This usually results in a larger cluster size.
3. The default n\_neighbors value for UMAP is 15.

* n\_components indicates the output dimension for UMAP. 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 is the minimum distance between points in the low-dimensional space.

1. 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.
2. 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.

1. BERTopic model by default produces different results each time because of the stochasticity inherited from UMAP.
2. To get reproducible topics, we need to pass a value to the random\_state parameter in the UMAP method.

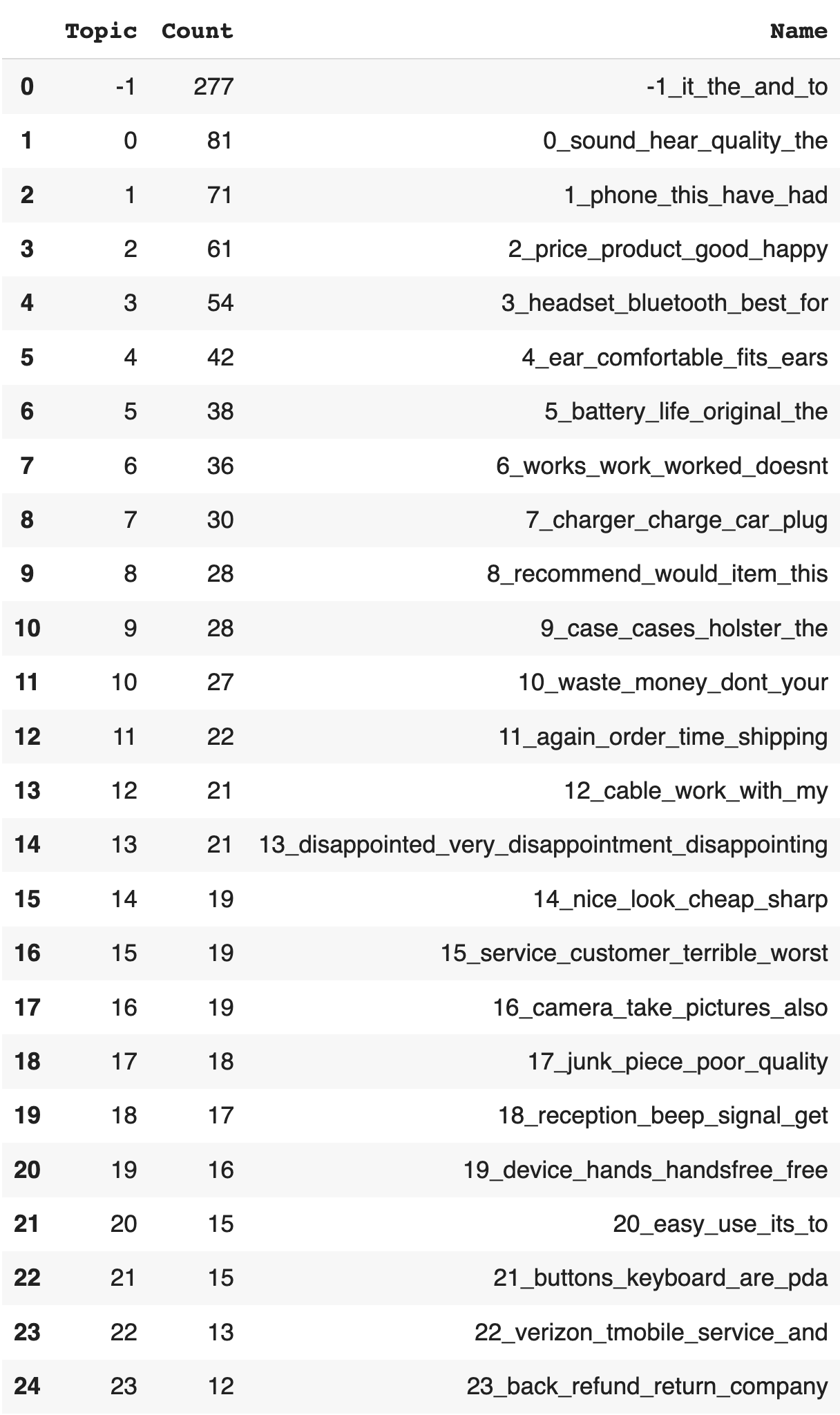
After initiating the UMAP model with the hyperparameters, we pass it to the BERTopic model, and run the model using the review data.

# 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)

# Run BERTopic model  
topics = topic\_model.fit\_transform(amz\_review['review'])

# Get the list of topics  
topic\_model.get\_topic\_info()

BERTopic model list of topics — GrabNGoInfo.com

Using the attribute get\_topic\_info() on the topic model gives us a list of topics. We can see that the output gives us 25 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 277, meaning that there are 277 outlier reviews that do not belong to any topic.
* Topic 0 to topic 23 are the 24 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, hear, quality, and the, indicating that it is a topic related to sound quality.

BERTopic provides the option of using other dimensionality reduction techniques by changing the umap\_model value in the BERTopic method.

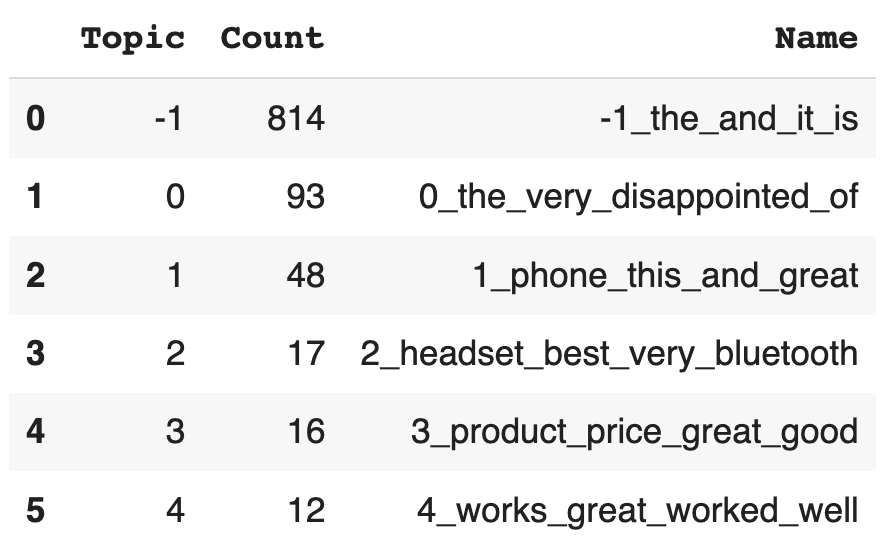
For example, we can use the widely used dimension reduction algorithm PCA to replace UMAP.

# PCA for dimensionality reduction  
pca\_model = PCA(n\_components=15)

# Initiate BERTopic  
topic\_model = BERTopic(umap\_model=pca\_model)

# Run BERTopic model  
topics = topic\_model.fit\_transform(amz\_review['review'])

# Get the list of topics  
topic\_model.get\_topic\_info()

BERTopic model list of topics — GrabNGoInfo.com

The output shows that 5 topics are created, which is much less than the 24 topics using UMAP.

Step 4: Hyperparameters For Clustering Model

In step 4, we will talk about the hyperparameters for the clustering model in BERTopic.

After the text documents have been transformed into embeddings, and the embeddings’ dimensionality has been reduced, the next step is to run a clustering model on the embedded documents.

The default algorithm for clustering is HDBSCAN. HDBSCAN is a density-based clustering model. It identifies the number of clusters automatically and does not require specifying the number of clusters beforehand like most of the clustering models.

HDBSCAN has a few important hyperparameters.

* min\_cluster\_size is the minimum number of documents in each cluster.

1. A larger minimum cluster size results in bigger clusters and fewer total clusters.
2. A smaller minimum cluster size results in smaller clusters and a larger number of total clusters.
3. A rule of thumb is to increase this threshold for a large dataset and keep it at the default value of 10 for a small dataset.

* min\_samples controls the number of outliers. It defaults to the same value as min\_cluster\_size. Reducing the value helps to reduce the noise in the topics.
* metric indicates the distance metric used for the clustering model such as euclidean.
* prediction\_data is for new documents topic predictions. We need to set it to False if there is no need for new document prediction.

After specifying the hyperparameters for the HDBSCAN model, we pass the model into the BERTopic method. Notice that when initiating the BERTopic model, the umap\_model from the previous step is passed in as well. This is because we would like to utilize the same random seed defined in the UMAP model to make the results comparable. We will include umap\_model for all the topic models going forward.

# Clustering model  
hdbscan\_model = HDBSCAN(min\_cluster\_size=10, min\_samples = 10, metric='euclidean', prediction\_data=True)

# Initiate BERTopic  
topic\_model = BERTopic(umap\_model=umap\_model, hdbscan\_model=hdbscan\_model)

# Run BERTopic model  
topics = topic\_model.fit\_transform(amz\_review['review'])

# Get the list of topics  
topic\_model.get\_topic\_info()

Besides the HDBSCAN clustering model, BERTopic supports other clustering algorithms.

There are two major differences between HDBSCAN and other clustering algorithms such as K-Means and GMM (Gaussian Mixture Model).

* HDBSCAN creates a separate cluster for outliers, but most other algorithms do not, so the clusters may contain more noise than HDBSCAN clusters.
* HDBSCAN automatically decides the number of clusters, while most other clustering algorithms need to have the number of clusters as input.

To learn more about the clustering model, please check out my previous tutorials [5 Ways for Deciding Number of Clusters](https://medium.com/grabngoinfo/5-ways-for-deciding-number-of-clusters-in-a-clustering-model-5db993ea5e09) and [4 Clustering Model Algorithms in Python](https://medium.com/grabngoinfo/4-clustering-model-algorithms-in-python-and-which-is-the-best-7f3431a6e624).

We provide the sample code to implement a K-Means clustering model with BERTopic, and other clustering model algorithms can follow the same process.

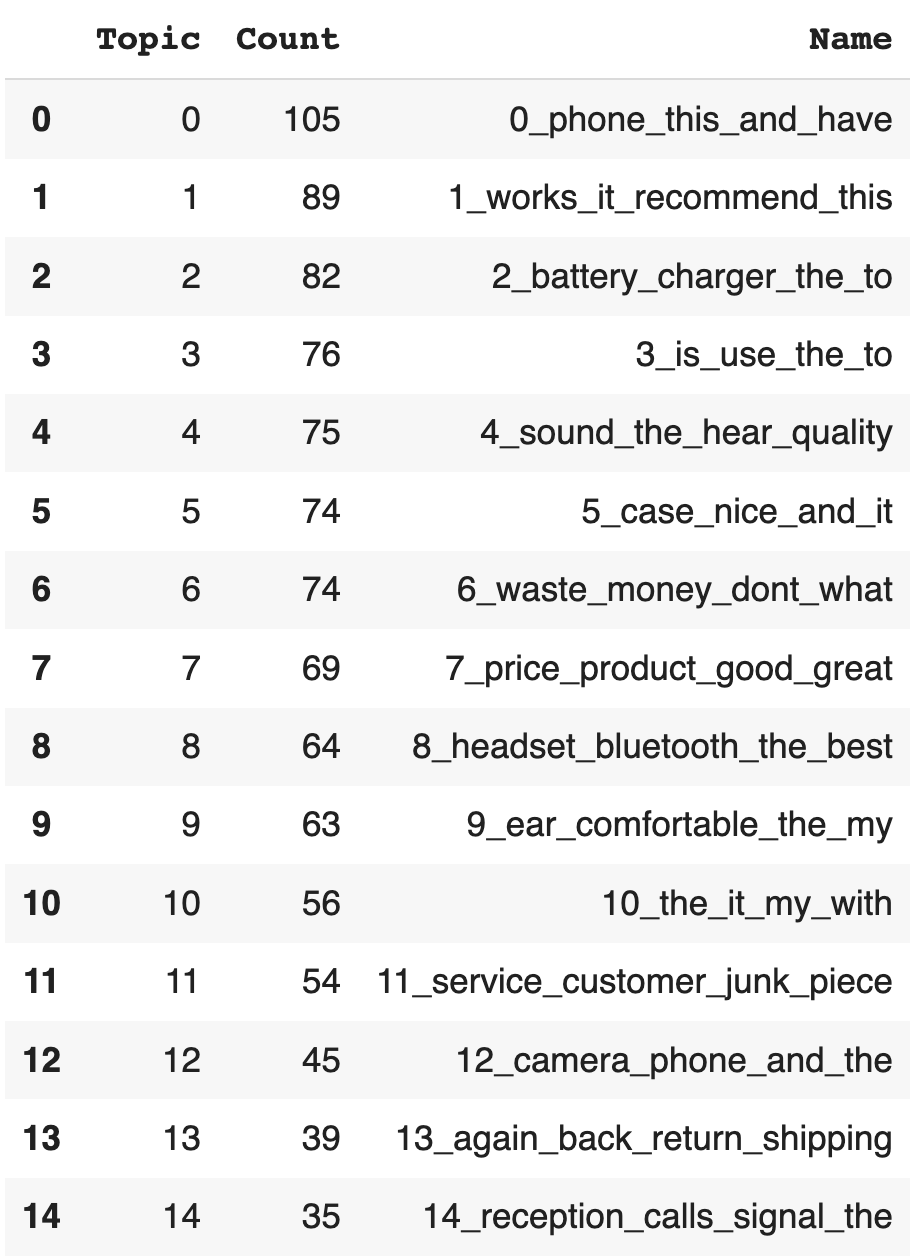
* Firstly, the K-Means model is initiated with the number of clusters.
* Then the initiated K-Means model is passed into the hdbscan\_model parameter in the BERTopic function.
* After that, the BERTopic model is fit and we get a list of 15 topics.

# Clustering model  
kmeans\_model = KMeans(n\_clusters=15)

# Initiate BERTopic  
topic\_model = BERTopic(umap\_model=umap\_model, hdbscan\_model=kmeans\_model)

# Run BERTopic model  
topics = topic\_model.fit\_transform(amz\_review['review'])

# Get the list of topics  
topic\_model.get\_topic\_info()

BERTopic model list of topics from k-means — GrabNGoInfo.com

Step 5: Hyperparameter Tuning For Language Embeddings

In step 5, we will talk about how to tune the language embeddings.

Embeddings are the vector representation of the documents. BERTopic uses the English version of the Sentence Transformer by default to get document embeddings.

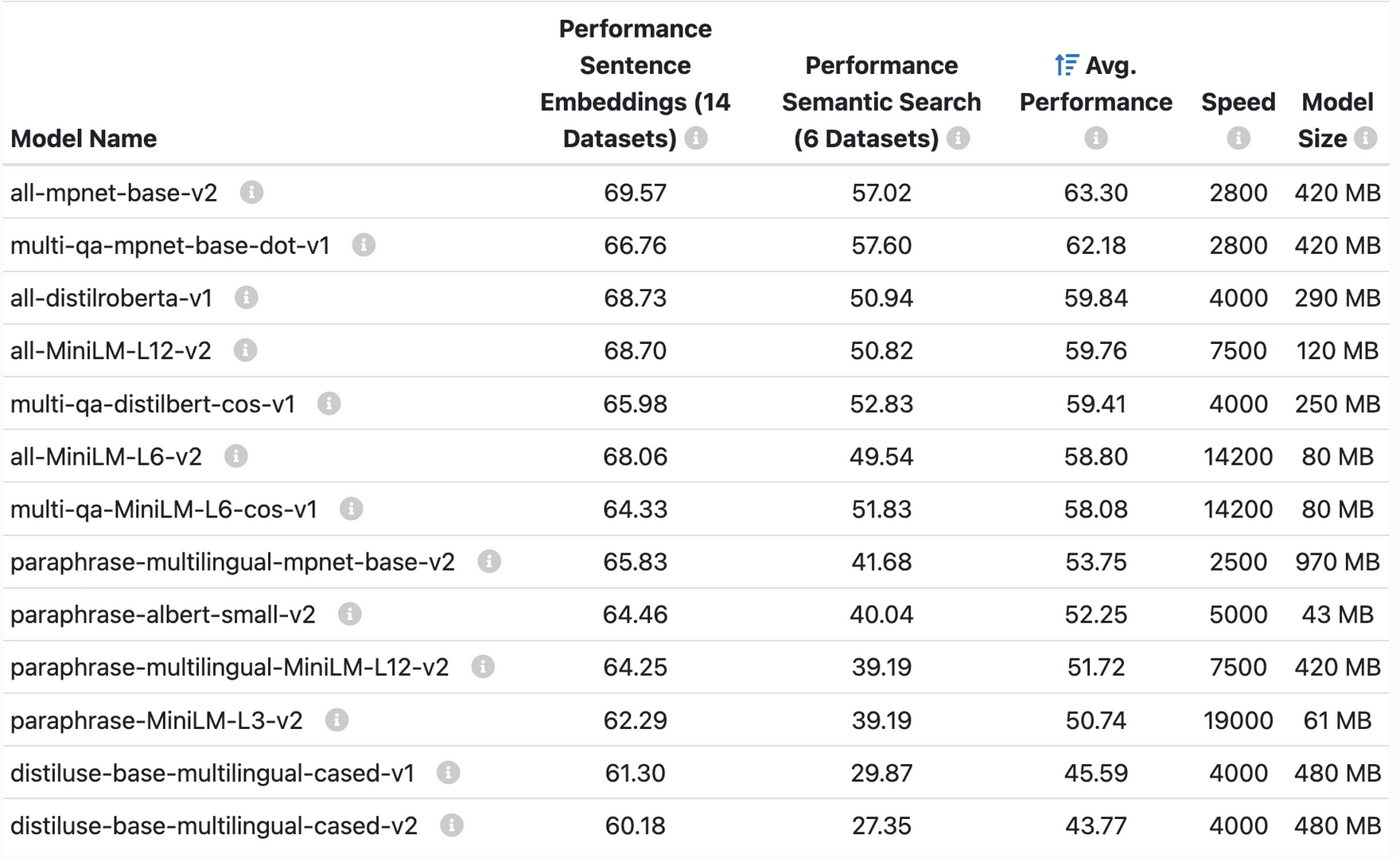
If there are multiple languages in the document, we can use BERTopic(language="multilingual") to support the topic modeling of over 50 languages.

# Initiate BERTopic  
topic\_model = BERTopic(umap\_model=umap\_model, language="multilingual")

On the backend, the language parameter in the BERTopic method uses the sentence-transformer model.

* When language="english", the sentence-transformer model all-MiniLM-L6-v2 is used.
* When language="multilingual", the sentence-transformer model paraphrase-multilingual-MiniLM-L12-v2 is used.

Sentence-transformer has different models with different sizes, speeds, and performances. We can go to the [sentence transformer website](https://www.sbert.net/docs/pretrained_models.html) for the latest list of pretrained models.

Sentence transformer model list — sbert.net

We can select any model from sentence transformers and pass it through BERTopic with the embedding\_model parameter.

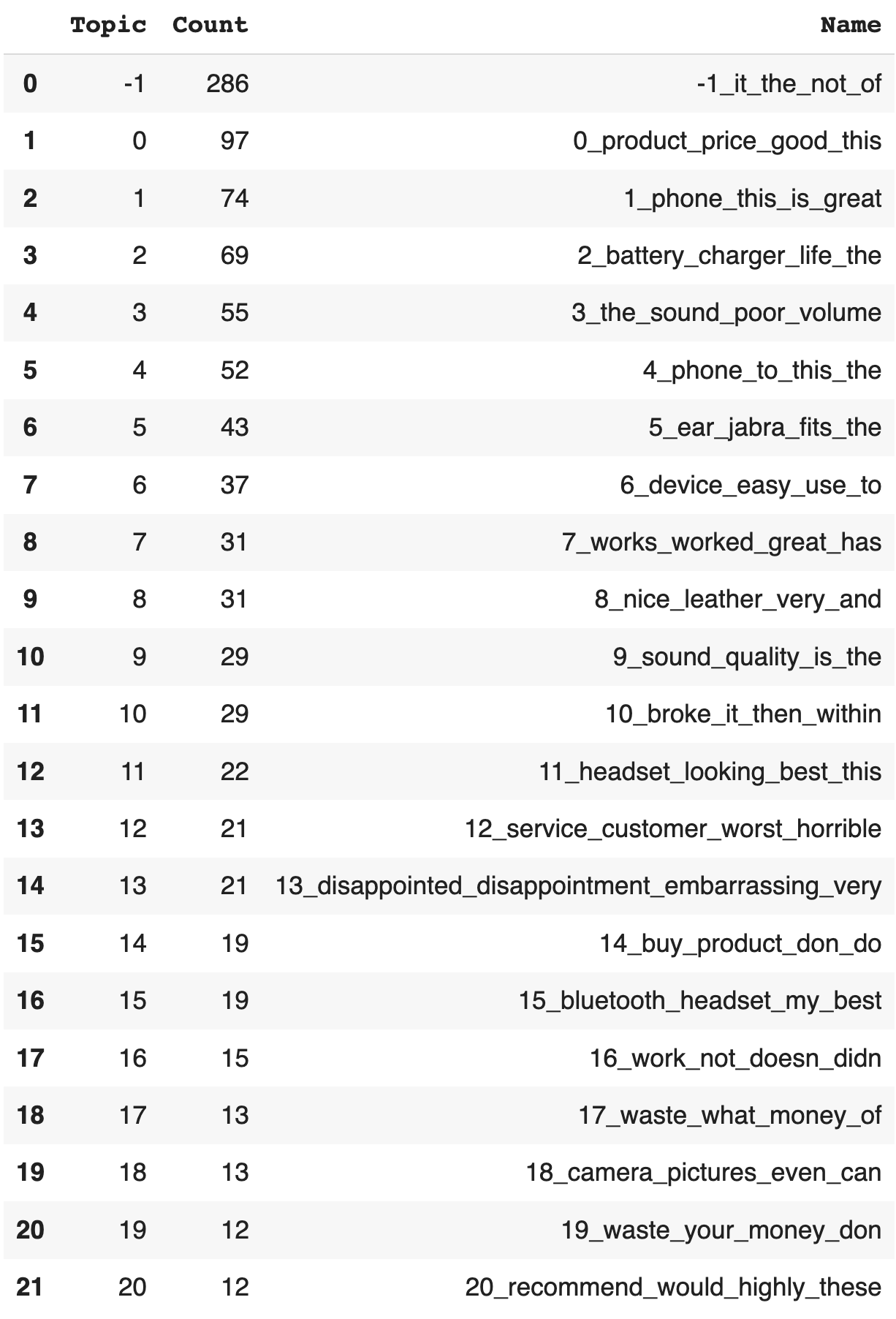
For example, to use the sentence transformer model paraphrase-albert-small-v2, we can pass it on to the SentenceTransformer, and set it as the embedding\_model.

# Initiate a sentence transformer model  
sentence\_model = SentenceTransformer("paraphrase-albert-small-v2")

# Initiate BERTopic  
topic\_model = BERTopic(umap\_model=umap\_model, embedding\_model=sentence\_model)

# Run BERTopic model  
topics = topic\_model.fit\_transform(amz\_review['review'])

# Get the list of topics  
topic\_model.get\_topic\_info()

Sentence transformer topic model output — GrabNGoInfo.com

The output gives us 21 topics, and the topics look similar to the topics from the default parameters.

Besides the sentence-transformer models, BERTopic supports the pre-trained models from other python packages such as hugging face and flair.

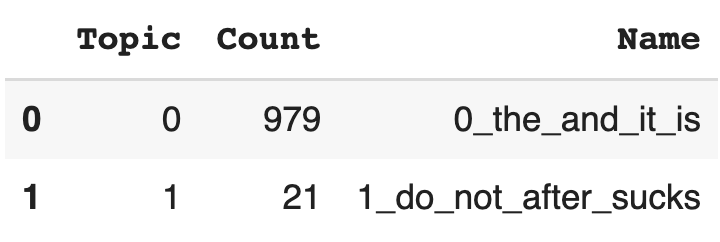
[Hugging Face model hub](https://huggingface.co/models) has thousands of pre-trained models. In this example, we used an English model called distilroberta-base, loaded it in a Hugging Face pipeline, and pass the pipeline to the parameter embedding\_model.

# Initiate a pretrained model  
hf\_model = pipeline("feature-extraction", model="distilroberta-base")

# Initiate BERTopic  
topic\_model = BERTopic(umap\_model=umap\_model, embedding\_model=hf\_model)

# Run BERTopic model  
topics = topic\_model.fit\_transform(amz\_review['review'])

# Get the list of topics  
topic\_model.get\_topic\_info()

Hugging face BERTopic model output — GrabNGoInfo.com

We can see that two topics are created, and the terms are not very meaningful, indicating that distilroberta-base is not a good choice for our review corpus.

Flair is an NLP (Natual Language Processing) library that allows us to choose almost any embedding models, or combine a few embedding models together.

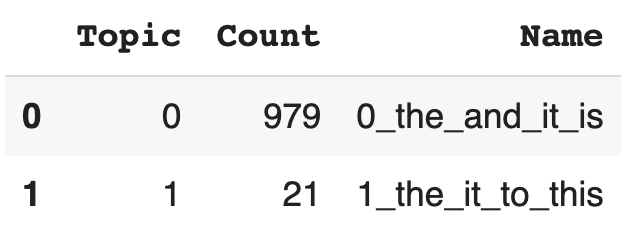
To use a single embedding model with Flair, we can pass the model name to TransformerDocumentEmbeddings, and use it as the input for the embedding\_model option in BERTopic.

# Initiate a pretrained embedding model  
roberta\_model = TransformerDocumentEmbeddings('roberta-base')

# Initiate BERTopic  
topic\_model = BERTopic(umap\_model=umap\_model, embedding\_model=roberta\_model)

# Run BERTopic model  
topics = topic\_model.fit\_transform(amz\_review['review'])

# Get the list of topics  
topic\_model.get\_topic\_info()

Flair BERTopic model output — GrabNGoInfo.com

We can see that two topics are created, and all the terms are stopwords without much meaning, indicating that this is not a good embedding model choice for our review corpus.

To use multiple embedding models with Flair, we first need to initiate different pretrained embedding models, then use the StackedEmbeddings function to stack the models, and finally pass the stacked embeddings to the BERTopic embedding\_model parameter.

# Initiate a pretrained embedding model  
roberta\_model = TransformerDocumentEmbeddings('roberta-base')

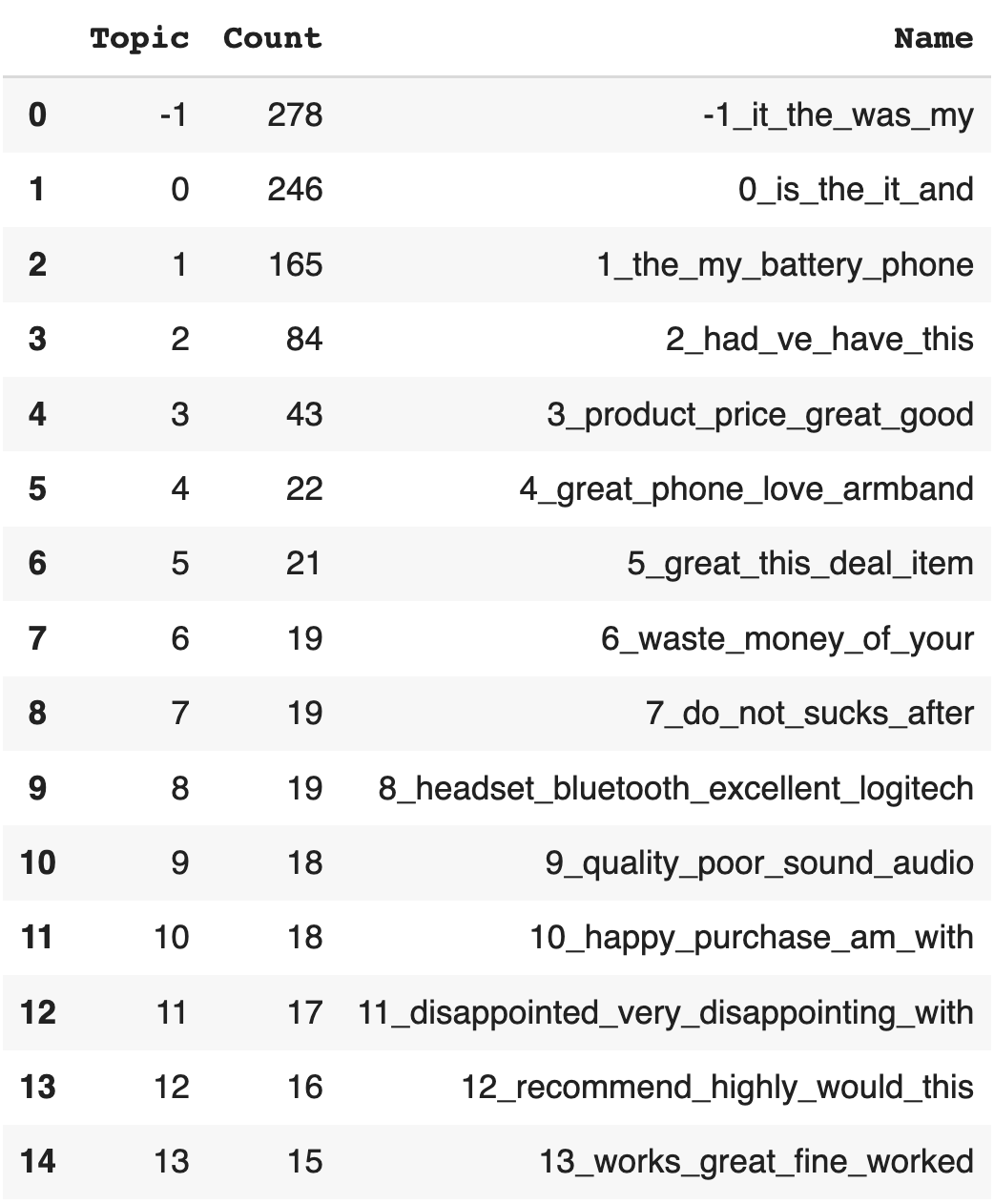
# Initiate another pretrained embedding model  
glove\_embedding = WordEmbeddings('crawl')  
document\_glove\_embeddings = DocumentPoolEmbeddings([glove\_embedding])

# Stack the two pretrained embedding models  
stacked\_embeddings = StackedEmbeddings(embeddings=[roberta\_model, document\_glove\_embeddings])

# Initiate BERTopic  
topic\_model = BERTopic(umap\_model=umap\_model, embedding\_model=stacked\_embeddings)

# Run BERTopic model  
topics = topic\_model.fit\_transform(amz\_review['review'])

# Get the list of topics  
topic\_model.get\_topic\_info()

Flair multiple pretrained embeddings BERTopic model output — GrabNGoInfo.com

We can see that 14 topics are created. The results look better than just using one model, but still not as good as the sentence transformer embedding models.

Step 6: Hyperparameter Tuning For Number Of Topics

In step 6, we will talk about how to change the number of topics for the topic model.

BERTopic uses the number of clusters created by the HDBSCAN model as the number of topics by default, but we can reduce the number of topics by changing the value of the nr\_topics parameter.

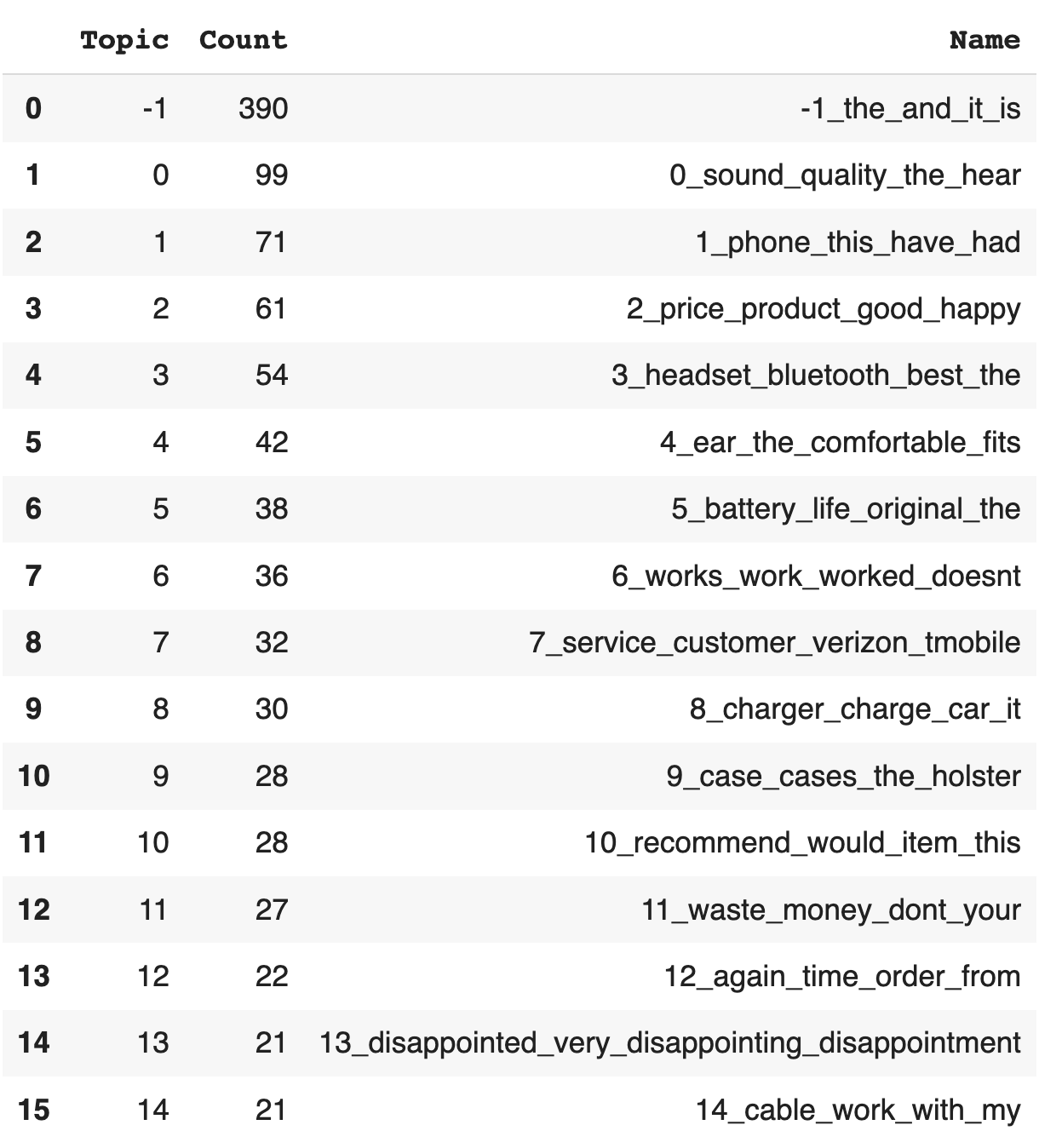
* nr\_topics=None indicates that there is no topic reduction.
* nr\_topics=auto indicates an automatic topic reduction of the HDBSCAN results by merging topics close to each other.
* nr\_topics=15 indicates that the target number of topics is 15.
* nr\_topics value should always be smaller than the number of topics created by nr\_topics=None.

On the backend, the topic reduction process is executed by merging similar topics based on the feature vector from c-TF-IDF. It starts with low-frequency topics and iteratively reduces the number of topics to the specified number.

# Initiate BERTopic  
topic\_model = BERTopic(umap\_model=umap\_model, nr\_topics=15)

# Run BERTopic model  
topics = topic\_model.fit\_transform(amz\_review['review'])

# Get the list of topics  
topic\_model.get\_topic\_info()

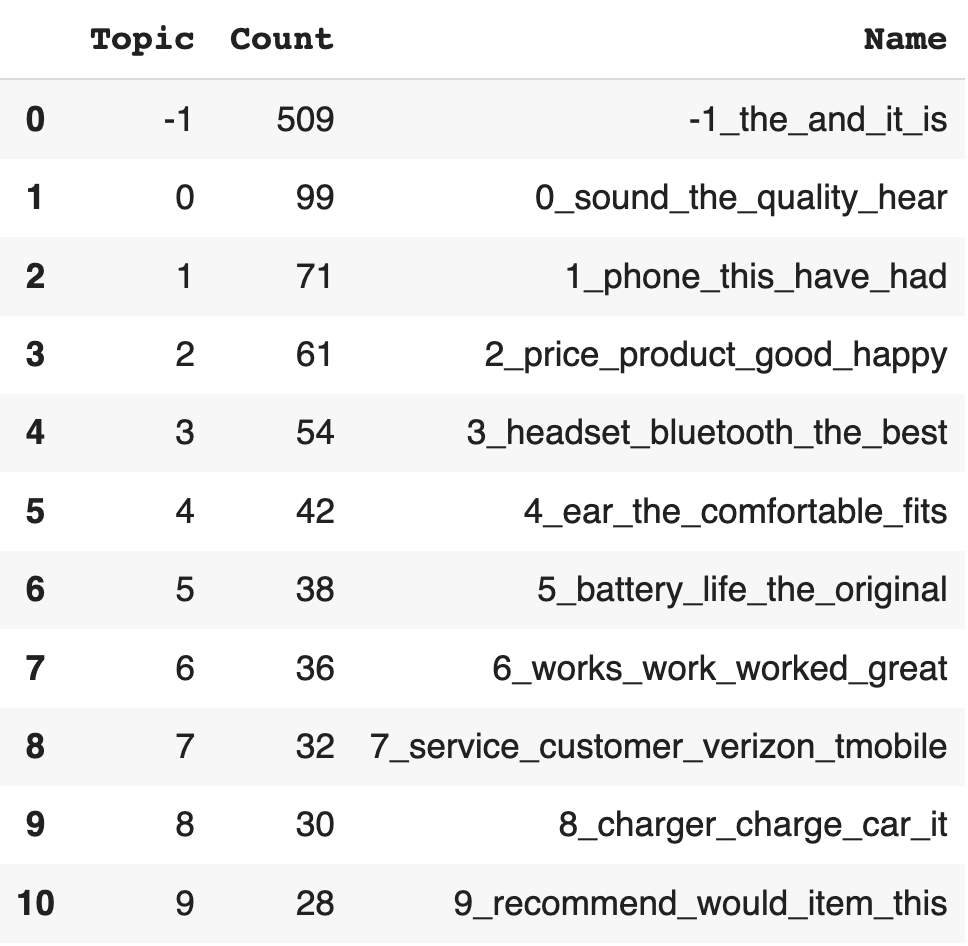
Number of topics for topic modeling — GrabNGoInfo.com

After setting nr\_topics=15, we can see that the BERTopic model produced 15 topics.

When the text corpus is large, training a BERTopic model can take a long time. Rerunning the model each time we change the number of topics can waste a lot of time and resources. The good news is that the BERTopic package has a reduce\_topics method that uses the existing model information to do a topic reduction.

# Further reduce topics  
topic\_model.reduce\_topics(amz\_review['review'], nr\_topics=10)

# Get the list of topics  
topic\_model.get\_topic\_info()

Reduce the number of topics for topic modeling — GrabNGoInfo.com

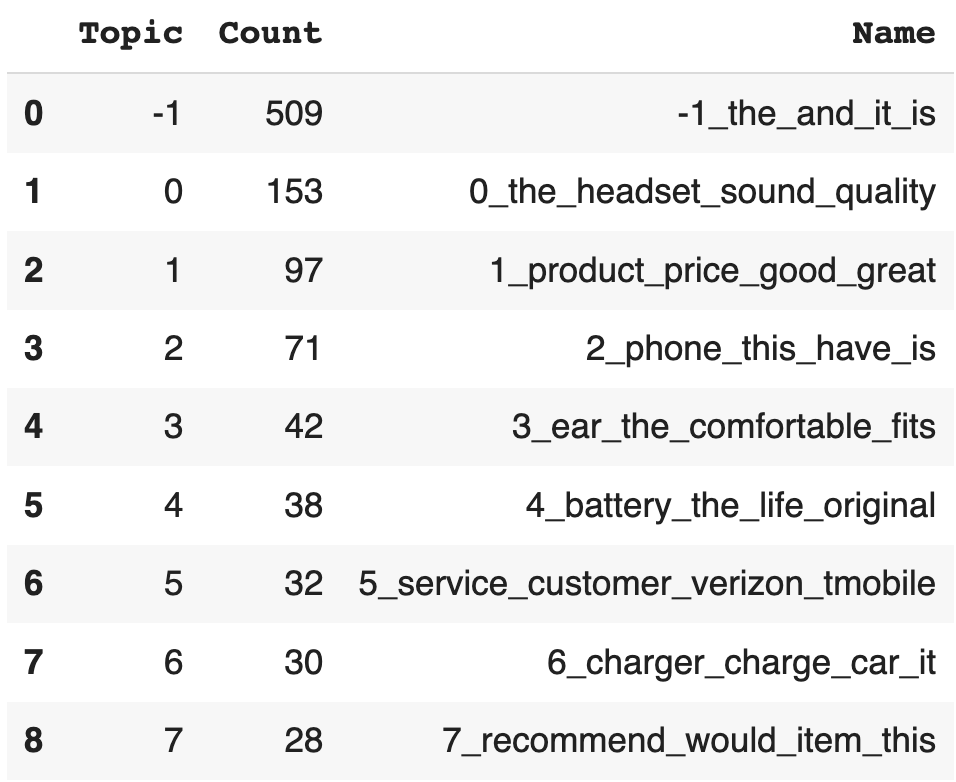
After passing in the review corpus and the number of topics, we can see that the number of topics is reduced to the specified number of 10.

If we would like to manually pick which topics to merge together based on domain knowledge, we can list the topic numbers and pass them into the merge\_topics function.

In this example, we merged topic 0 and topic 3 together because they both talk about headphone quality, and merged topic 2 and topic 6 together because they both talk about product satisfaction. The number of topics is reduced by two, and we have 8 topics now.

topics\_to\_merge = [[0, 3],  
 [2, 6]]  
topic\_model.merge\_topics(amz\_review['review'], topics\_to\_merge)

# Get the list of topics  
topic\_model.get\_topic\_info()

Reduce the number of topics for topic modeling — GrabNGoInfo.com

Another way of adjusting the number of topics is to control the minimum number of documents in a topic. We can set up this value by the parameter min\_topic\_size.

* A low value for min\_topic\_size allows fewer documents to form a topic, so the topic model produces more topics.
* A high value for min\_topic\_size requires a lot of documents to form a topic, so the topic model produces fewer topics.
* The default value for min\_topic\_size is 10. A general guideline for setting min\_topic\_size is to set up a low value for a smaller dataset, and a high value for a larger dataset.

Setting min\_topic\_size is the same as setting min\_cluster\_size in HDBSCAN.

# Initiate BERTopic  
topic\_model = BERTopic(umap\_model=umap\_model, min\_topic\_size=25)

# Run BERTopic model  
topics = topic\_model.fit\_transform(amz\_review['review'])

# Get the list of topics  
topic\_model.get\_topic\_info()

We can see that after setting the minimum topic size of 25, we get fewer topics, and each topic has more than 25 documents.

Step 7: Hyperparameter For Top Words

In step 7, we will tune the hyperparameter for the top words representing the topics. The parameters that we will talk about are n\_gram\_range and top\_n\_words.

* n\_gram\_range is used to specify the range of n-grams included in the topic model.
* top\_n\_words controls how many words are used to describe the topic.

Let’s take a look at n\_gram\_range first.

N-grams are a continuous sequence of words. Its value determines the words used in CountVectorizer, and the top words representing the topics.

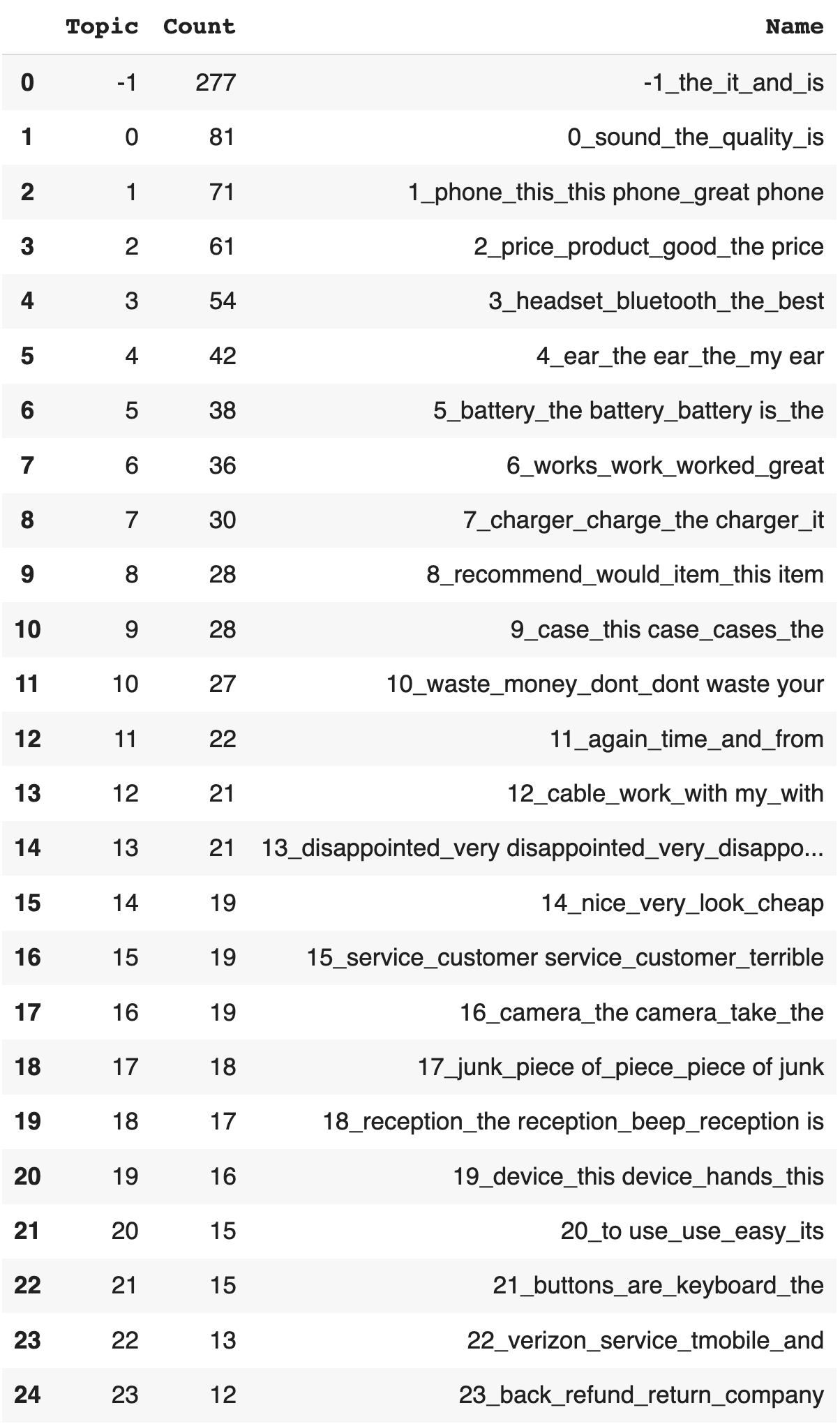
* Unigram refers to one word. Unigram is the default for BERTopic.
* Bigram refers to two consecutive words. For example, “ice cream” is considered as one word for bigram, but will be separated into two words, “ice” and “cream” for unigram.
* Trigram refers to three consecutive words.
* Cardinal numbers are used for more than three consecutive words, four-gram, five-gram, for example.

n\_gram\_range=(1, 3) means that unigrams, bigrams, and trigrams are included in the model.

# Initiate BERTopic  
topic\_model = BERTopic(umap\_model=umap\_model, n\_gram\_range=(1, 3))

# Run BERTopic model  
topics = topic\_model.fit\_transform(amz\_review['review'])

# Get the list of topics  
topic\_model.get\_topic\_info()

BERTopic ngram list of topics — GrabNGoInfo.com

From the output, we can see that the keywords representing the topics include both single words and multi-word phrases.

Next, let’s talk about top\_n\_words. top\_n\_words has the default value of 10, meaning that 10 top words will be used to represent each topic. If we change the value to 5, only the top five most representative words are included.

# Initiate BERTopic  
topic\_model = BERTopic(umap\_model=umap\_model, top\_n\_words=5)

# Run BERTopic model  
topics = topic\_model.fit\_transform(amz\_review['review'])

# Got top topic words  
topic\_model.get\_topic(1)

Output:

[('phone', 0.1329223535845747),  
 ('this', 0.06647473203777743),  
 ('have', 0.03391477256848962),  
 ('had', 0.033855540482838516),  
 ('great', 0.02960535516059868)]

Step 8: Hyperparameters For Words Universe

In step 8, we will talk about how to control the number of words for the topic model. Limiting the number of words helps to reduce the noise in the topics.

There are two ways to control how many words are used in CountVectorizer and c-TF-IDF.

* min\_df sets a threshold for the required word frequency. For example, min\_df=10 indicates that any words that appeared less than 10 times in the corpus will not be included in the c-TF-IDF calculation. A general guideline is to set a high min\_df value for a large corpus and a low value for a small corpus.
* max\_features indicates the maximum number of words to include for the c-TF-IDF calculation. max\_features=1\_000 means that the top 1000 words with the highest frequency in the corpus will be included.

Both min\_df and max\_features are the hyperparameters for the CountVectorizer.

To use min\_df, we set the value when initiating CountVectorizer, then pass it to the vectorizer\_model argument in the BERTopic method.

# Count vectorizer  
vectorizer\_model = CountVectorizer(min\_df=10)

# Initiate BERTopic  
topic\_model = BERTopic(umap\_model=umap\_model, vectorizer\_model=vectorizer\_model)

# Run BERTopic model  
topics = topic\_model.fit\_transform(amz\_review['review'])

# Get the list of topics  
topic\_model.get\_topic\_info()

To use max\_features, we set the value when initiating CountVectorizer, then pass it to the vectorizer\_model argument in the BERTopic method.

# Count vectorizer  
vectorizer\_model = CountVectorizer(max\_features=1\_000)

# Initiate BERTopic  
topic\_model = BERTopic(umap\_model=umap\_model, vectorizer\_model=vectorizer\_model)

# Run BERTopic model  
topics = topic\_model.fit\_transform(amz\_review['review'])

# Get the list of topics  
topic\_model.get\_topic\_info()

Step 9: Hyperparameter For Diversifying Topic Representation

In step 9, we will tune the hyperparameter to achieve a more diversified topic representation.

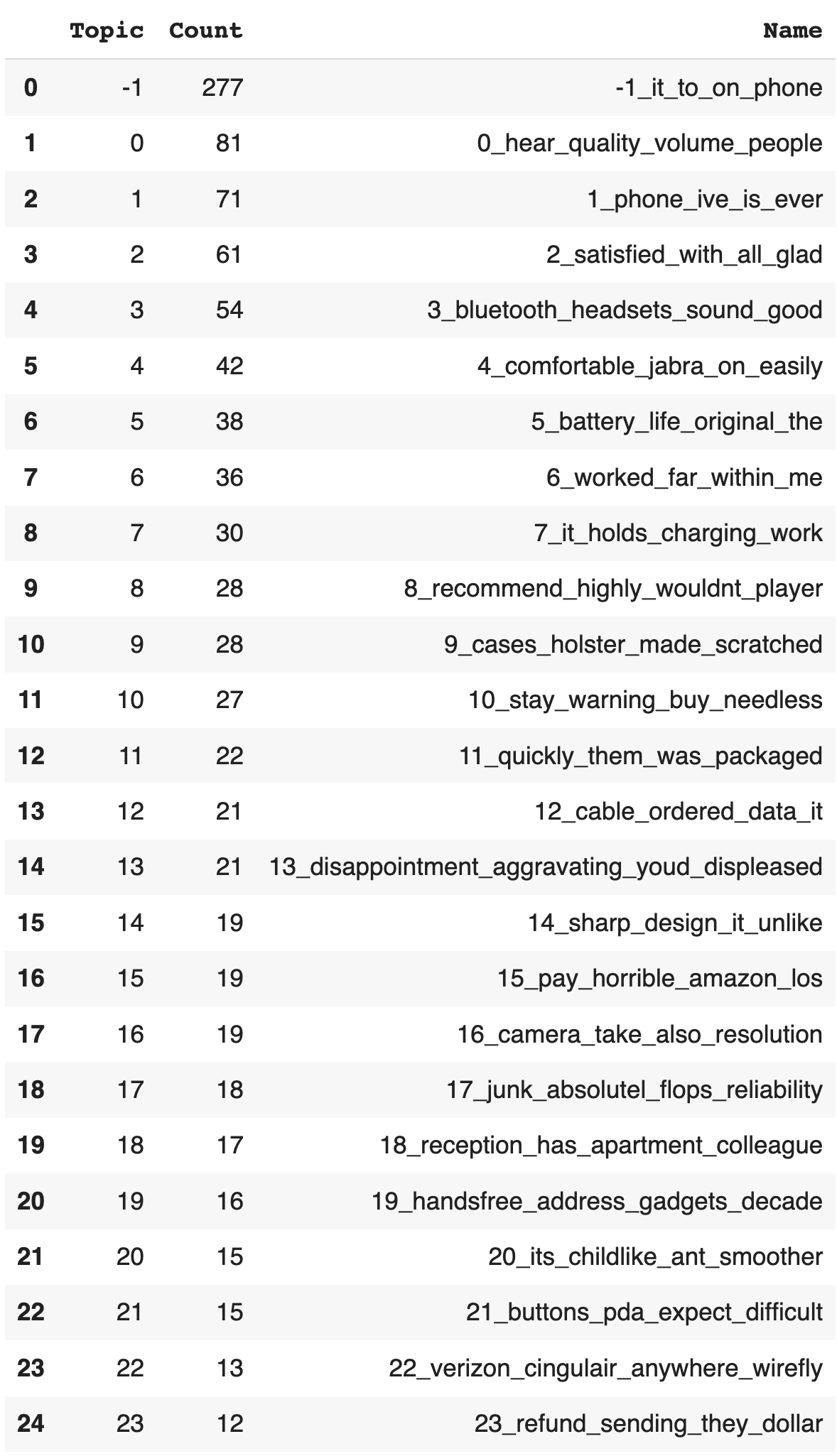
The top n words that represent the topic may include variations of the same word or words that are synonyms.

The hyperparameter diversity helps to remove the words with the same or similar meanings. It has a range of 0 to 1, where 0 means least diversity and 1 means most diversity.

# Initiate BERTopic  
topic\_model = BERTopic(umap\_model=umap\_model, diversity=0.8)

# Run BERTopic model  
topics = topic\_model.fit\_transform(amz\_review['review'])

# Get the list of topics  
topic\_model.get\_topic\_info()

Topic modeling diversified top words — GrabNGoInfo.com

After setting diversity=0.8, we can see that the top words describing the topics are more diversified. For example, topic 13 previously has the top words disappointed\_very\_disappointment\_disappointing, and has the top words disappointment\_aggravating\_youd\_displeased after diversification, which is much more diversified.

Step 10: Hyperparameter For Stopwords

In step 10, we will talk about how to remove the stopwords from the list of the top words.

After creating the topics, if the top words representing the topics contain stopwords, we can remove the stopwords using stop\_words="english" with CountVectorizer.

# Count vectorizer  
vectorizer\_model = CountVectorizer(stop\_words="english")

# Initiate BERTopic  
topic\_model = BERTopic(umap\_model=umap\_model, vectorizer\_model=vectorizer\_model)

# Run BERTopic model  
topics = topic\_model.fit\_transform(amz\_review['review'])

# Get the list of topics  
topic\_model.get\_topic\_info()

Topic modeling remove stopwords from top words — GrabNGoInfo.com

We can see that after using stop\_words="english", all the stopwords are removed from the top keywords.

Step 11: Hyperparameter For Topic Probability Output

In step 11, we will talk about the boolean parameter that decides whether or not to produce topic probability.

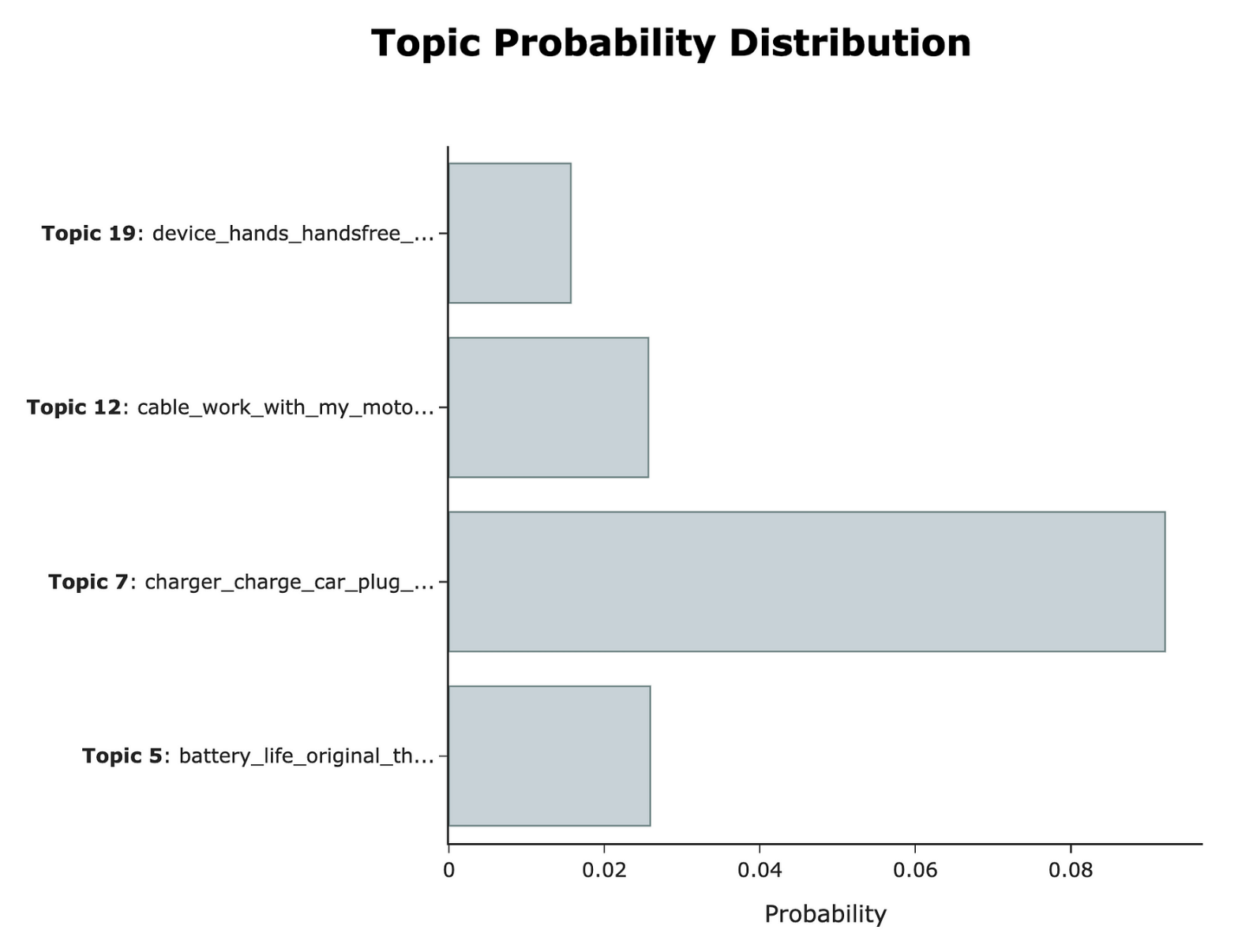
* When calculate\_probabilities = True, the probabilities of each document belonging to each topic are calculated. The topic with the highest probability is the predicted topic for a new document. This probability represents how confident we are about finding the topic in the document.
* When calculate\_probabilities = False, the probabilities of each document belonging to each topic are not calculated. This saves computation time and cost. If there is no new document to predict, we do not need to calculate the probabilities.

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.

# Initiate BERTopic  
topic\_model = BERTopic(umap\_model=umap\_model, calculate\_probabilities=True)

# Run BERTopic model  
topics, probablity = topic\_model.fit\_transform(amz\_review['review'])

# Visualize probability distribution  
topic\_model.visualize\_distribution(topic\_model.probabilities\_[0], min\_probability=0.015)

Topic model probability distribution — GrabNGoInfo.com

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]

Output:

array([0.0126781 , 0.00997078, 0.00806085, 0.01186496, 0.01055103,  
 0.02589788, 0.01059081, 0.09212873, 0.01008473, 0.00990964,  
 0.00826178, 0.00974247, 0.02565346, 0.00914894, 0.01153246,  
 0.0117639 , 0.01009115, 0.01135396, 0.01329444, 0.01567501,  
 0.01403915, 0.01352111, 0.01451199, 0.01091581])

We can see that there are 24 probability values, one for each topic. Index 7 has the highest value, indicating that topic 7 is the predicted topic.

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)
* [Topic Modeling with Deep Learning Using Python BERTopic](https://medium.com/grabngoinfo/topic-modeling-with-deep-learning-using-python-bertopic-cf91f5676504)
* [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)
* [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)
* [sbert pretrained model](https://www.sbert.net/docs/pretrained_models.html)
* [Hugging Face model hub](https://huggingface.co/models)
* [Flair GitHub](https://github.com/flairNLP/flair)