Chapter 5. Text Clustering and Topic Modeling

Although supervised techniques, such as classification, have reigned supreme over the last few years in the industry, the potential of unsupervised techniques such as text clustering cannot be understated.

Text clustering aims to group similar texts based on their semantic content, meaning, and relationships. As illustrated in Figure 5-1, the resulting clusters of semantically similar documents not only facilitate efficient categorization of large volumes of unstructured text but also allows for quick exploratory data analysis.

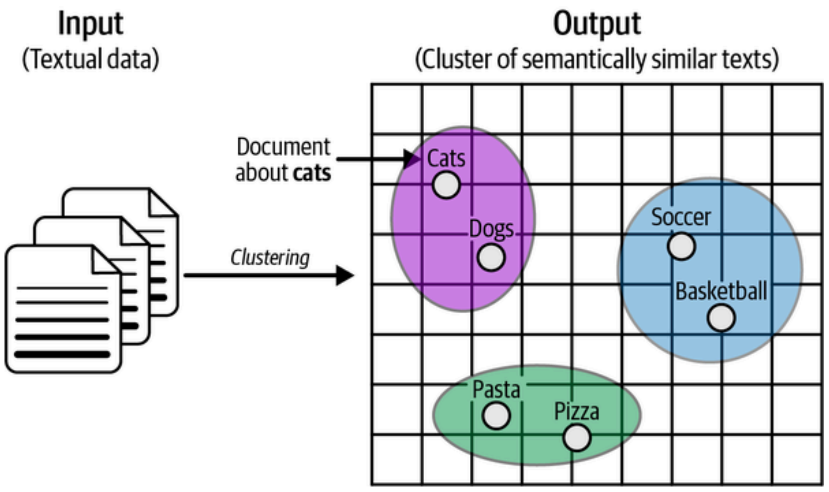


Figure 5-1. Clustering unstructured textual data.

The recent evolution of language models, which enable contextual and semantic representations of text, has enhanced the effectiveness of text clustering. Language is more than a bag of words, and recent language models have proved to be quite capable of capturing that notion. Text clustering, unbound by supervision, allows for creative solutions and diverse applications, such as finding outliers, speedup labeling, and finding incorrectly labeled data.

Text clustering has also found itself in the realm of topic modeling, where we want to discover (abstract) topics that appear in large collections of textual data. As shown in Figure 5-2, we generally describe a topic using keywords or keyphrases and, ideally, have a single overarching label.

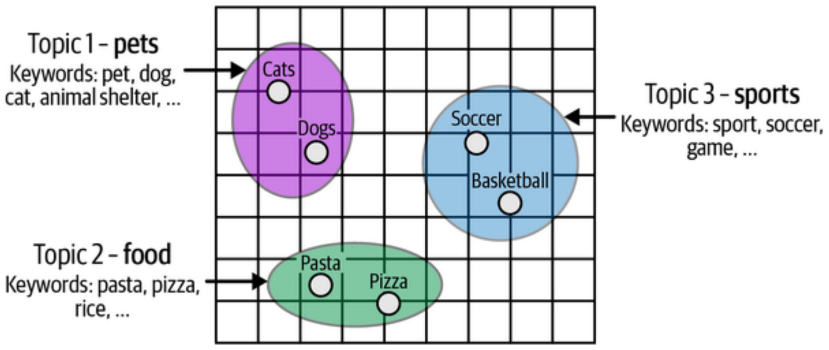

Topic 1 - pets
Keywords: pet, dog, cat, animal shelter, ...
Cats
Dogs
Soccer
Basketball
Topic 3 - sports
Keywords: sport, soccer, game, ...
Topic 2 - food
Keywords: pasta, pizza, rice, ...
Pasta
Pizza

Figure 5-2. Topic modeling is a way to give meaning to clusters of textual documents.

In this chapter, we will first explore how to perform clustering with embedding models and then transition to a text-clustering-inspired method of topic modeling, namely BERTopic.

Text clustering and topic modeling have an important role in this book as they explore creative ways to combine a variety of different language models. We will explore how combining encoder-only (embeddings), decoder-only (generative), and even classical methods (bag-of-words) can result in amazing new techniques and pipelines.

ArXiv's Articles: Computation and Language

Throughout this chapter, we will be running clustering and topic modeling algorithms on ArXiv articles. ArXiv (*https://arxiv.org/*) is an open-access platform for scholarly articles, mostly in the fields of computer science, mathematics, and physics. We will explore articles in the field of Computation and Language to keep with the theme of this book. The dataset contains 44,949 abstracts between 1991 and 2024 from ArXiv's cs.CL (Computation and Language) section.

We load the data and create separate variables for the abstracts, titles, and years of each article:

```
# Load data from Hugging Face
from datasets import load_dataset
dataset = load_dataset("maartengr/arxiv_nlp")["train"]

# Extract metadata
abstracts = dataset["Abstracts"]
titles = dataset["Titles"]
```

A Common Pipeline for Text Clustering

Text clustering allows for discovering patterns in data that you may or may not be familiar with. It allows for getting an intuitive understanding of the task, for example, a classification task, but also of its complexity. As a result, text clustering can become more than just a quick method for exploratory data analysis.

Although there are many methods for text clustering, from graph-based neural networks to centroid-based clustering techniques, a common pipeline that has gained popularity involves three steps and algorithms:

1. Convert the input documents to embeddings with an *embedding model*
2. Reduce the dimensionality of embeddings with a *dimensionality reduction model*
3. Find groups of semantically similar documents with a *cluster model*

1. Embedding Documents

The first step is to convert our textual data to embeddings, as illustrated in Figure 5-3. Recall from previous chapters that embeddings are numerical representations of text that attempt to capture its meaning.

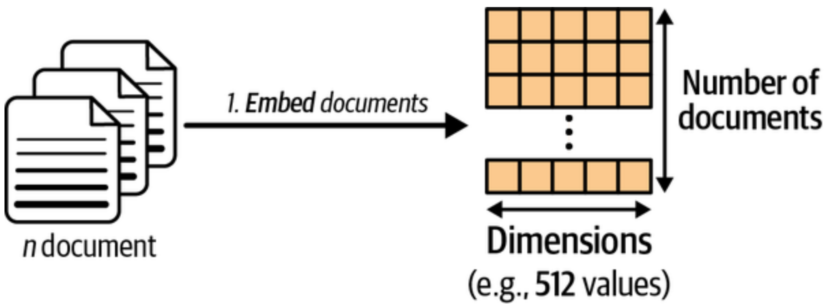


Figure 5-3. Step 1: We convert documents to embeddings using an embedding model.

Choosing embedding models optimized for semantic similarity tasks is especially important for clustering as we attempt to find groups of semantically similar documents. Fortunately, most embedding models at the time of writing focus on just that, semantic similarity.

As we did in the previous chapter, we will use the MTEB leaderboard to select an embedding model. We will need an embedding model that has a decent score on clustering tasks but also is small enough to run quickly. Instead of using the "sentence-transformers/all-mpnet-base-v2" we used in the previous chapter, we use the "thenlper/gte-small" model instead. It is a more recent model that outperforms the previous model on clustering tasks and due to its small size is even faster for inference. However, feel free to play around with newer models that have been released since!

```
from sentence_transformers import SentenceTransformer

# Create an embedding for each abstract
```

```
embedding_model = SentenceTransformer('thenlper/gte-small')
embeddings = embedding_model.encode(abstracts, show_progress_bar=True)
```

Let's check how many values each document embedding contains:

```
# Check the dimensions of the resulting embeddings
embeddings.shape
```

```
(44949, 384)
```

Each embedding has 384 values that together represent the semantic representation of the document. You can view these embeddings as the features that we want to cluster.

2. Reducing the Dimensionality of Embeddings

Before we cluster the embeddings, we will first need to take their high dimensionality into account. As the number of dimensions increases, there is an exponential growth in the number of possible values within each dimension. Finding all subspaces within each dimension becomes increasingly complex.

As a result, high-dimensional data can be troublesome for many clustering techniques as it gets more difficult to identify meaningful clusters. Instead, we can make use of dimensionality reduction. As illustrated in Figure 5-4, this technique allows us to reduce the size of the dimensional space and represent the same data with fewer dimensions. Dimensionality reduction techniques aim to preserve the global structure of high-dimensional data by finding low-dimensional representations.

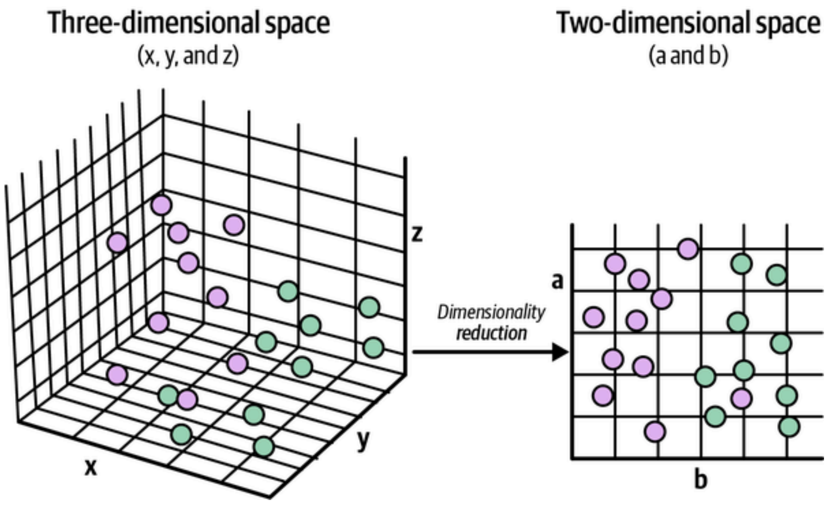


Figure 5-4. Dimensionality reduction allows data in high-dimensional space to be compressed to a lower-dimensional representation.

Note that this is a compression technique and that the underlying algorithm is not arbitrarily removing dimensions. To help the cluster model create meaningful clusters, the second step in our clustering pipeline is therefore dimensionality reduction, as shown in Figure 5-5.

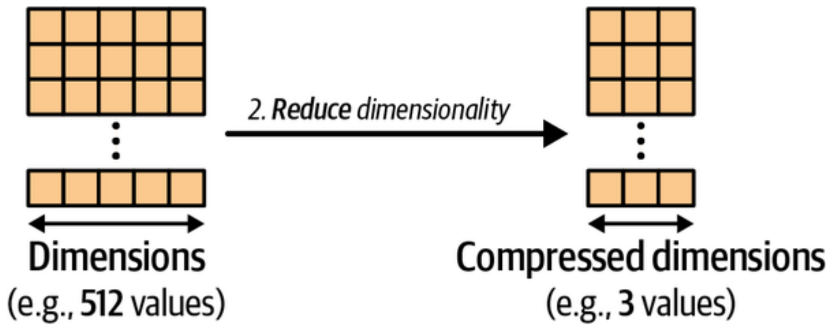


Figure 5-5. Step 2: The embeddings are reduced to a lower dimensional space using dimensionality reduction.

Well-known methods for dimensionality reduction are Principal Component Analysis (PCA)[1] and Uniform Manifold Approximation and Projection (UMAP).[2] For this pipeline, we are going with UMAP as it tends to handle non-linear relationships and structures a bit better than PCA.

NOTE

Dimensionality reduction techniques, however, are not flawless. They do not perfectly capture high-dimensional data in a lower-dimensional representation. Information will always be lost with this procedure. There is a balance between reducing dimensionality and keeping as much information as possible.

To perform dimensionality reduction, we need to instantiate our UMAP class and pass the generated embeddings to it:

```
from umap import UMAP

# We reduce the input embeddings from 384 dimensions to 5 dimenions
umap_model = UMAP(
    n_components=5, min_dist=0.0, metric='cosine', random_state=42
)
reduced_embeddings = umap_model.fit_transform(embeddings)
```

We can use the `n_components` parameter to decide the shape of the lower-dimensional space, namely 5 dimensions. Generally, values between 5 and 10 work well to capture high-dimensional global structures.

The `min_dist` parameter is the minimum distance between embedded points. We are setting this to 0 as that generally results in tighter clusters. We set `metric` to `'cosine'` as Euclidean-based methods have issues dealing with high-dimensional data.

Note that setting a `random_state` in UMAP will make the results reproducible across sessions but will disable parallelism and therefore slow down training.

3. Cluster the Reduced Embeddings

The third step is to cluster the reduced embeddings, as illustrated in Figure 5-6.

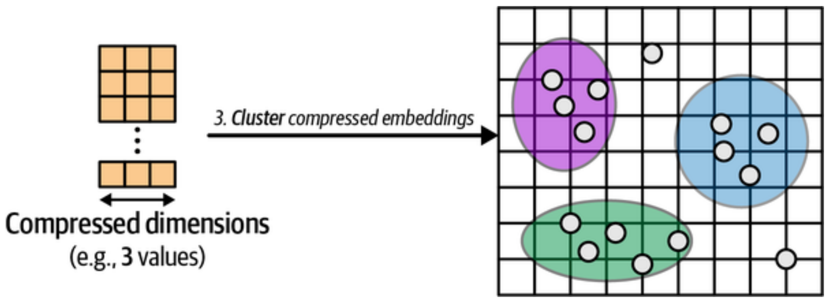


Figure 5-6. Step 3: We cluster the documents using the embeddings with reduced dimensionality.

Although a common choice is a centroid-based algorithm like k-means, which requires a set of clusters to be generated, we do not know the number of clusters beforehand. Instead, a density-based algorithm freely calculates the number of clusters and does not force all data points to be part of a cluster, as illustrated in Figure 5-7.

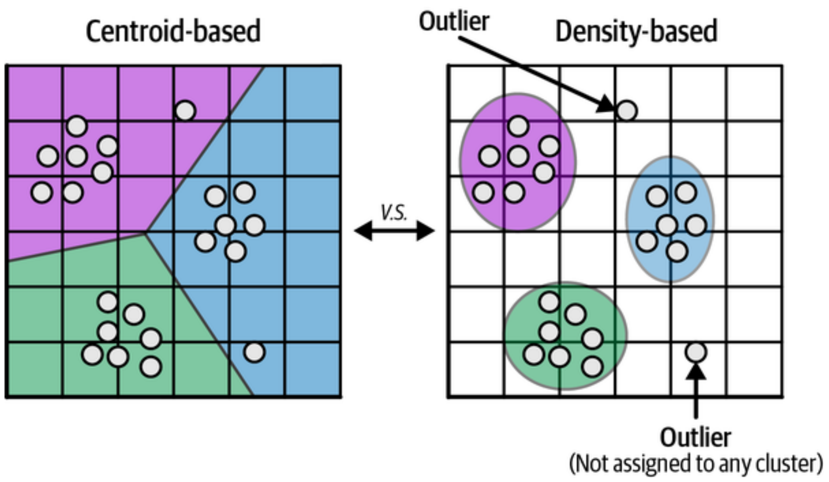


Figure 5-7. The clustering algorithm not only impacts how clusters are generated but also how they are viewed.

A common density-based model is Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN).[3] HDBSCAN is a hierarchical variation of a clustering algorithm called DBSCAN that allows for dense (micro)-clusters to be found without having to explicitly specify the number of clusters.[4] As a density-based method, HDBSCAN can also detect *outliers* in the data, which are data points that do not belong to any cluster. These outliers will not be assigned or forced to belong to any cluster. In other words, they are ignored. Since ArXiv articles might contain some niche papers, using a model that detects outliers could be helpful.

As with the previous packages, using HDBSCAN is straightforward. We only need to instantiate the model and pass our reduced embeddings to it:

```
from hdbscan import HDBSCAN

# We fit the model and extract the clusters
hdbscan_model = HDBSCAN(
    min_cluster_size=50, metric='euclidean', cluster_selection_method='eom'
).fit(reduced_embeddings)
clusters = hdbscan_model.labels_
```

```
# How many clusters did we generate?
len(set(clusters))
```

```
156
```

With HDBSCAN, we generated 156 clusters in our dataset. To create more clusters, we will need to reduce the value of `min_cluster_size` as it represents the minimum size that a cluster can take.

Inspecting the Clusters

Now that we have generated our clusters, we can inspect each cluster manually and explore the assigned documents to get an understanding of its content. For example, let us take a few random documents from cluster 0:

```
import numpy as np

# Print first three documents in cluster 0
cluster = 0
for index in np.where(clusters==cluster)[0][:3]:
    print(abstracts[index][:300] + "... \n")
```

```
 This works aims to design a statistical machine
translation from English text

 to American Sign Language (ASL). The system is based on
Moses tool with some

 modifications and the results are synthesized through a 3D
avatar for

 interpretation. First, we translate the input text to
gloss, a written fo...

 Researches on signed languages still strongly dissociate
lin- guistic issues

 related on phonological and phonetic aspects, and gesture
studies for

 recognition and synthesis purposes. This paper focuses on
the imbrication of

 motion and meaning for the analysis, synthesis and
evaluation of sign lang...

 Modern computational linguistic software cannot produce
important aspects of

 sign language translation. Using some researches we deduce
that the majority of
```

```
automatic sign language translation systems ignore many
aspects when they

generate animation; therefore the interpretation lost the
truth inf...
```

From these documents, it seems that this cluster contains documents mostly about translation from and to sign language, interesting!

We can take this one step further and attempt to visualize our results instead of going through all documents manually. To do so, we will need to reduce our document embeddings to two dimensions, as that allows us to plot the documents on an x/y plane:

```
import pandas as pd

# Reduce 384-dimensional embeddings to two dimensions for easier visualization
reduced_embeddings = UMAP(
    n_components=2, min_dist=0.0, metric='cosine', random_state=42
).fit_transform(embeddings)

# Create dataframe
df = pd.DataFrame(reduced_embeddings, columns=["x", "y"])
df["title"] = titles
df["cluster"] = [str(c) for c in clusters]

# Select outliers and non-outliers (clusters)
to_plot = df.loc[df.cluster != "-1", :]
outliers = df.loc[df.cluster == "-1", :]
```

We also created a dataframe for our clusters (`clusters_df`) and for the outliers (`outliers_df`) separately since we generally want to focus on the clusters and highlight those.

NOTE

Using any dimensionality reduction technique for visualization purposes creates information loss. It is merely an approximation of what our original embeddings look like. Although it is informative, it might push clusters together and drive them further apart than they actually are. Human evaluation, inspecting the clusters ourselves, is therefore a key component of cluster analysis!

To generate a static plot, we will use the well-known plotting library, `matplotlib`:

```
import matplotlib.pyplot as plt

# Plot outliers and non-outliers seperately
plt.scatter(outliers_df.x, outliers_df.y, alpha=0.05, s=2, c="grey")
plt.scatter(
    clusters_df.x, clusters_df.y, c=clusters_df.cluster.astype(int),
    alpha=0.6, s=2, cmap='tab20b'
)
plt.axis('off')
```

As we can see in Figure 5-8, it tends to capture major clusters quite well. Note how clusters of points are colored in the same color, indicating that HDBSCAN put them in a group together. Since we have a large number of clusters, the plotting library cycles the colors between clusters, so don't think that all green points are one cluster, for example.

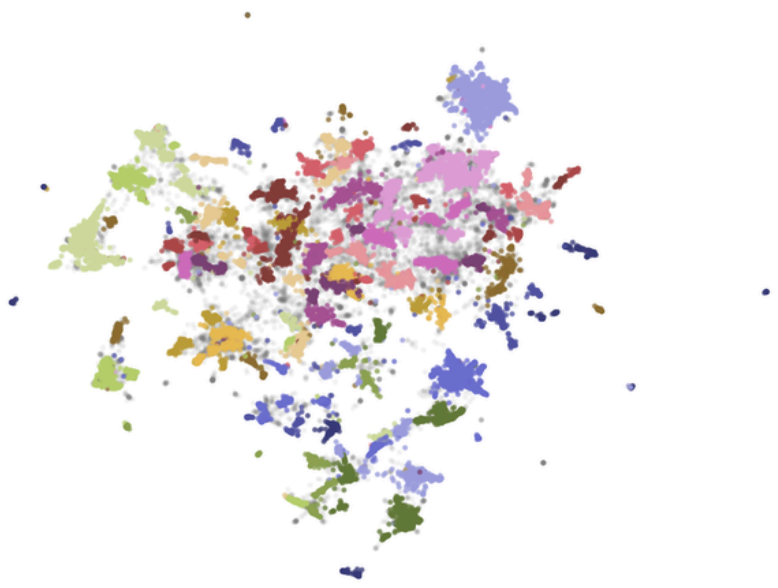


Figure 5-8. The generated clusters (colored) and outliers (grey) are represented as a 2D visualization.

This is visually appealing but does not yet allow us to see what is happening inside the clusters. Instead, we can extend this visualization by going from text clustering to topic modeling.

From Text Clustering to Topic Modeling

Text clustering is a powerful tool for finding structure amongst large collections of documents. In our previous example, we could manually inspect each cluster and identify them based on their collection of documents. For instance, we explored a cluster that contained documents about sign language. We could say that the *topic* of that cluster is "sign language."

This idea of finding themes or latent topics in a collection of textual data is often referred to as topic modeling. Traditionally, it involves finding a set of keywords or phrases that best represent and capture the meaning of the topic, as we illustrate in Figure 5-9.

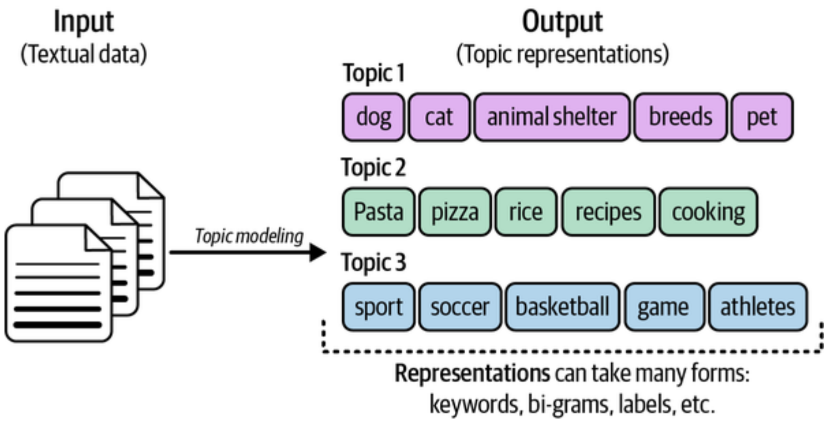


Figure 5-9. Traditionally, topics are represented by a number of keywords but can take other forms.

Instead of labeling a topic as "sign language," these techniques use keywords such as "sign," "language," and "translation" to describe the topic. As such, this does not give a single label to a topic and instead requires the user to understand the meaning of the topic through those keywords.

Classic approaches, like latent Dirichlet allocation assume that each topic is characterized by a probability distribution of words in a corpus's vocabulary.[5] Figure 5-10 demonstrates how each word in a vocab is scored against its relevance to each topic.

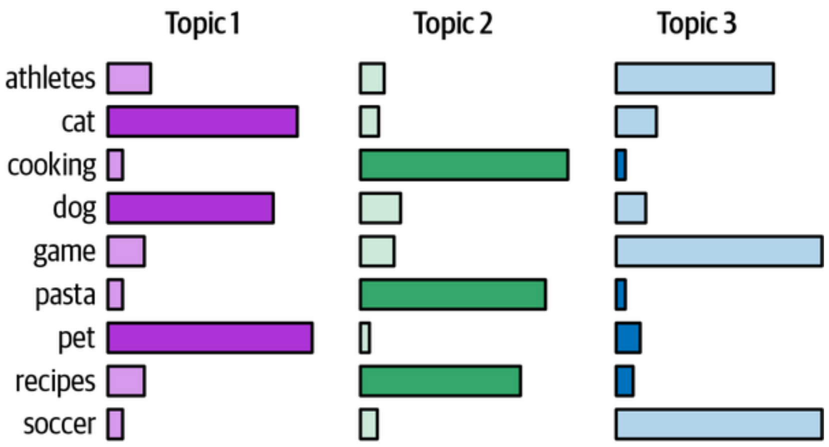


Figure 5-10. Keywords are extracted based on their distribution over a single topic.

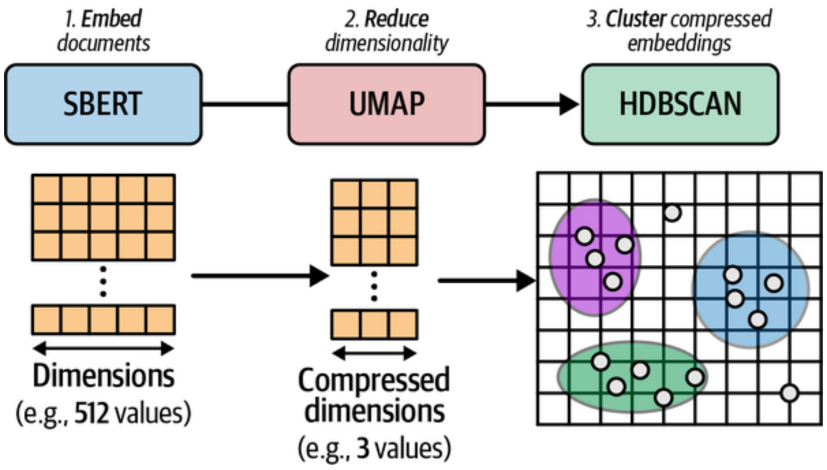
These approaches generally use a bag-of-words technique for the main features of the textual data, which does not take the context nor the meaning of words and phrases into account. In contrast, our text clustering example does take both into account as it relies on Transformer-based embeddings that are optimized for semantic similarity and contextual meaning through attention.

In this section, we will extend text clustering into the realm of topic modeling through a highly modular text clustering and topic modeling framework, namely BERTopic.

BERTopic: A Modular Topic Modeling Framework

BERTopic is a topic modeling technique that leverages clusters of semantically similar texts to extract various types of topic representations.[6] The underlying algorithm can be thought of in two steps.

First, as illustrated in Figure 5-11, we follow the same procedure as we did before in our text clustering example. We embed documents, reduce their dimensionality, and finally cluster the reduced embedding to create groups of semantically similar documents.



1. Embed documents
2. Reduce dimensionality
3. Cluster compressed embeddings
SBERT
UMAP
HDBSCAN
Dimensions (e.g., 512 values)
Compressed dimensions (e.g., 3 values)

Figure 5-11. The first part of BERTopic's pipeline is to create clusters of semantically similar documents.

Second, it models a distribution over words in the corpus's vocabulary by leveraging a classic method, namely bag-of-words. The bag-of-words, as we discussed briefly in Chapter 1 and illustrated in Figure 5-12, does exactly what its name implies, counting the number of times each word appears in a document. The resulting representation could be used to extract the most frequent words inside a document.

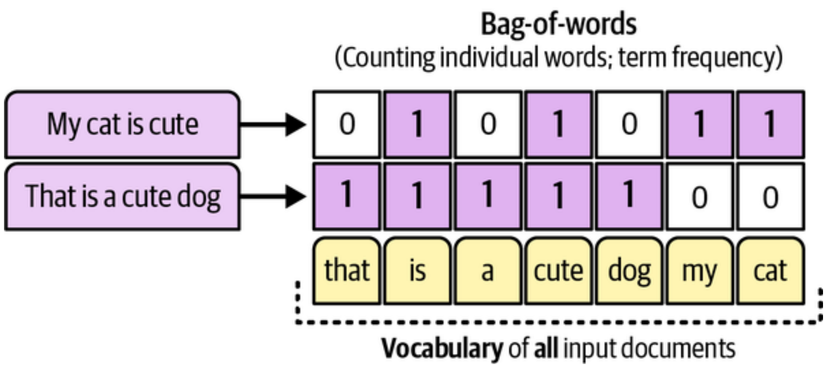


Figure 5-12. A bag-of-words counts the number of times each word appears inside a document.

There are two caveats, however. First, this is a representation on a document level and we are interested in a cluster-level perspective. To address this, the frequency of words is calculated within the entire cluster instead of only the document, as illustrated in Figure 5-13.

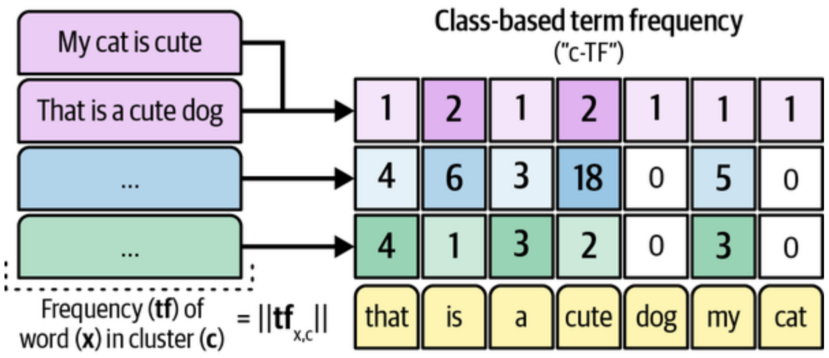

Class-based term frequency
("c-TF")
My cat is cute
That is a cute dog
...
...
1 2 1 2 1 1 1
4 6 3 18 0 5 0
4 1 3 2 0 3 0
that is a cute dog my cat
Frequency (tf) of word (x) in cluster (c) = ||tf x,c||

Figure 5-13. Generating c-TF by counting the frequency of words per cluster instead of per document.

Second, stop words like "the" and "I" tend to appear often in documents and provide little meaning to the actual documents. BERTopic uses a class-based variant of term frequency–nverse document frequency (c-TF-IDF) to put more weight on words that are more meaningful to a cluster and put less weight on words that are used across all clusters.

Each word in the bag-of-words, the c-TF in c-TF-IDF, is multiplied by the IDF value of each word. As shown in Figure 5-14, the IDF value is calculated by taking the logarithm of the average frequency of all words across all clusters divided by the total frequency of each word.

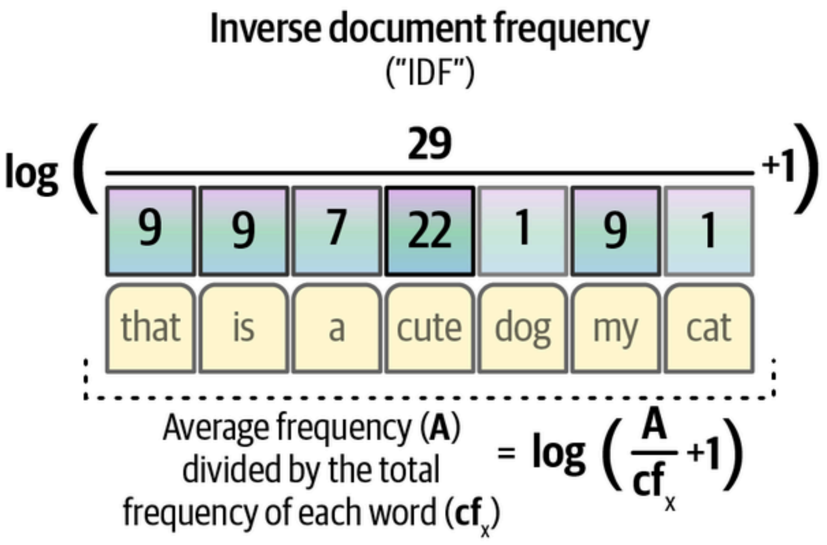

Inverse document frequency
("IDF")
log (29 / 9 9 7 22 1 9 1 +1)
that is a cute dog my cat
Average frequency (A) divided by the total frequency of each word (cf x) = log (A / cf x +1)

Figure 5-14. Creating a weighting scheme.

The result is a weight ("IDF") for each word that we can multiply with their frequency ("c-TF") to get the weighted values ("c-TF-IDF").

This second part of the procedure, as shown in Figure 5-15, allows for generating a distribution over words as we have seen before. We can use scikit-learn's `CountVectorizer` to generate the bag-of-words (or term frequency) representation. Here, each cluster is considered a topic that has a specific ranking of the corpus's vocabulary.

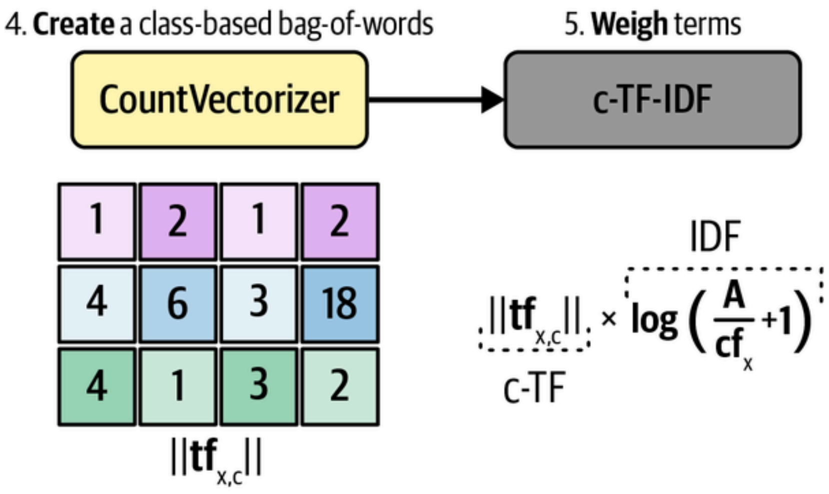

4. Create a class-based bag-of-words
5. Weigh terms
CountVectorizer
c-TF-IDF
1 2 1 2
4 6 3 18
4 1 3 2
$||tf_{x,c}||$
IDF
$||tf_{x,c}|| \times \log\left(\frac{A}{cf_x}+1\right)$
c-TF

Figure 5-15. The second part of BERTopic's pipeline is representing the topics: the calculation of the weight of term *x* in a class *c*.

Putting the two steps together, clustering and representing topics, results in the full pipeline of BERTopic, as illustrated in Figure 5-16. With this pipeline, we can cluster semantically similar documents and from the clusters generate topics represented by several keywords. The higher a word's weight in a topic, the more representative it is of that topic.

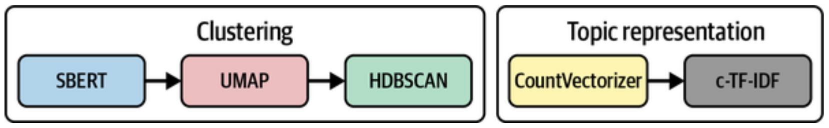


Figure 5-16. The full pipeline of BERTopic, roughly, consists of two steps, clustering and topic representation.

A major advantage of this pipeline is that the two steps, clustering and topic representation, are largely independent of one another. For instance, with c-TF-IDF, we are not dependent on the models used in clustering the documents. This allows for significant modularity throughout every component of the pipeline. And as we will explore later in this chapter, it is a great starting point to fine-tune the topic representations.

As illustrated in Figure 5-17, although `sentence-transformers` is used as the default embedding model, we can swap it with any other embedding technique. The same applies to all other steps. If you do not want outliers generated with HDBSCAN, you can use k-means instead.

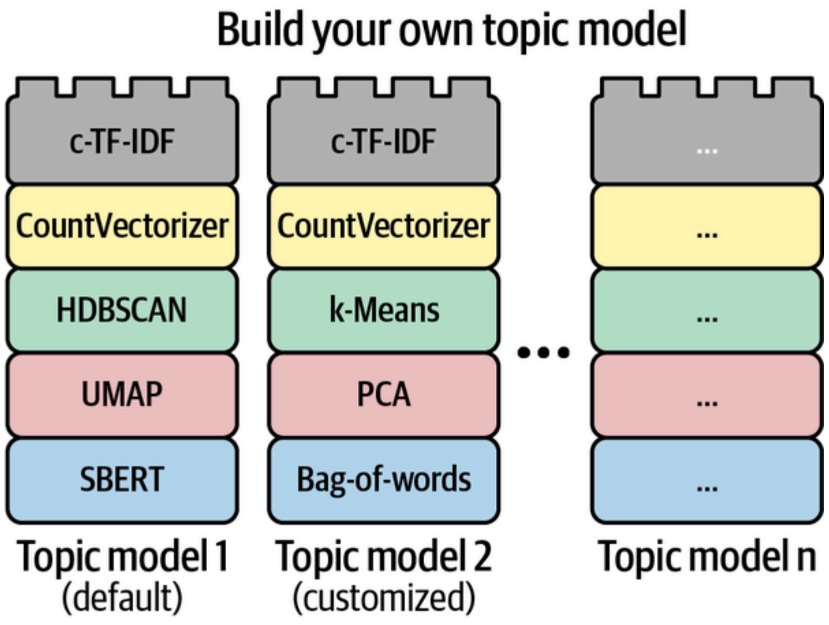


Figure 5-17. The modularity of BERTopic is a key component and allows you to build your own topic model whoever you want.

You can think of this modularity as building with Lego blocks; each part of the pipeline is completely replaceable with another, similar algorithm. Through this modularity, newly released models can be integrated within its architecture. As the field of Language AI grows, so does BERTopic!

TIP

The modularity of BERTopic has another advantage: it allows it to be used and adapted to different use cases using the same base model. For instance, BERTopic supports a wide variety of algorithmic variants:

- Guided topic modeling
- (Semi-)supervised topic modeling
- Hierarchical topic modeling
- Dynamic topic modeling
- Multimodal topic modeling
- Multi-aspect topic modeling
- Online and incremental topic modeling
- Zero-shot topic modeling
- Etc.

The modularity and algorithmic flexibility are the foundation of the author's aim to make BERTopic the one-stop-shop for topic modeling. You can find a full overview of its capabilities in the documentation or the repository.

To run BERTopic with our ArXiv dataset, we can use our previously defined models and embeddings (although it is not mandatory):

```
from bertopic import BERTopic

# Train our model with our previously defined models
topic_model = BERTopic(
    embedding_model=embedding_model,
    umap_model=umap_model,
    hdbscan_model=hdbscan_model,
    verbose=True
).fit(abstracts, embeddings)
```

Let us start by exploring the topics that were created. The `get_topic_info()` method is useful to get a quick description of the topics that we found:

```
    topic_model.get_topic_info()
```

Topic	Count	Name	Representation
-1	14520	-1_the_of_and_to	[the, of, and, to, in, we, that, language, for...
0	2290	0_speech_asr_recognition_end	[speech, asr, recognition, end, acoustic, spea...
1	1403	1_medical_clinical_biomedical_patient	[medical, clinical, biomedical, patient, healt...
2	1156	2_sentiment_aspect_analysis_reviews	[sentiment, aspect, analysis, reviews, opinion...
3	986	3_translation_nmt_machine_neural	[translation, nmt, machine, neural, bleu, engl...
...	...	...	...
150	54	150_coherence_discourse_paragraph_text	[coherence, discourse, paragraph, text, cohesi...
151	54	151_prompt_prompts_optimization_prompting	[prompt, prompts, optimization, prompting, llm...
152	53	152_sentence_sts_embeddings_similarity	[sentence, sts, embeddings, similarity, embedd...
153	53	153_counseling_mental_health_therapy	[counseling, mental, health, therapy, psychoth...
154	50	154_backdoor_attacks_attack_triggers	[backdoor, attacks, attack, triggers, poisoned...

Each of these topics is represented by several keywords, which are concatenated with a “_” in the `Name` column. This `Name` column allows us to quickly get a feeling of what the topic is about as it shows the four keywords that best represent it.

NOTE

You might also have noticed that the very first topic is labeled -1. That topic contains all documents that could not be fitted within a topic and are considered outliers. This is a result of the clustering algorithm, HDBSCAN, which does not force all points to be clustered. To remove outliers, we could either use a non-outlier algorithm like k-means or use BERTopic's `reduce_outliers()` function to reassign the outliers to topics.

We can inspect individual topics and explore which keywords best represent them with the `get_topic` function. For example, topic 0 contains the following keywords:

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

```
[('speech', 0.028177697715245358),

('asr', 0.018971184497453525),

('recognition', 0.013457745472471012),

('end', 0.00980445092749381),

('acoustic', 0.009452082794507863),

('speaker', 0.0068822647060204885),

('audio', 0.006807649923681604),

('the', 0.0063343444687017645),

('error', 0.006320144717019838),

('automatic', 0.006290216996043161)]
```

For example, topic 0 contains the keywords "speech," "asr," and "recognition"." Based on these keywords, it seems that the topic is about automatic speech recognition (ASR).

We can use the `find_topics()` function to search for specific topics based on a search term. Let's search for a topic about topic modeling:

```
topic_model.find_topics("topic modeling")
```

```
([22, -1, 1, 47, 32],

[0.95456535, 0.91173744, 0.9074769, 0.9067007,
0.90510106])
```

This returns that topic 22 has a relatively high similarity (0.95) with our search term. If we then inspect the topic, we can see that it is indeed a topic about topic modeling:

```
topic_model.get_topic(22)
```

```
[(’topic’, 0.06634619076655907),

(’topics’, 0.035308535091932707),

(’lda’, 0.016386314730705634),

(’latent’, 0.013372311924864435),

(’document’, 0.012973600191120576),

(’documents’, 0.012383715497143821),

(’modeling’, 0.011978375291037142),

(’dirichlet’, 0.010078277589545706),

(’word’, 0.008505619415413312),

(’allocation’, 0.007930890698168108)]
```

Although we know that this topic is about topic modeling, let’s see if the BERTopic abstract is also assigned to this topic:

```
topic_model.topics_[titles.index(’BERTopic: Neural topic modeling with a class-based TF-ID
```

```
22
```

It is! These functionalities allow us to find the topics that we are interested in quickly.

TIP

The modularity of BERTopic gives you a lot of choices, which can be overwhelming. For that purpose, the author created a best practices guide that goes through common practices to speed up training, improve representations, and more.

To make exploration of the topics a bit easier, we can look back at our text clustering example. There, we created a static visualization that allows us to see the general structure of the created topic. With BERTopic, we can create an interactive variant that allows us to quickly explore which topics exist and which documents they contain.

Doing so requires us to use the two-dimensional embeddings, `reduced_embeddings`, that we created with UMAP. Moreover, when we hover over documents, we will show the title instead of the abstract to quickly get an understanding of the documents in a topic:

```
# Visualize topics and documents
fig = topic_model.visualize_documents(
    titles,
    reduced_embeddings=reduced_embeddings,
    width=1200,
    hide_annotations=True
)

# Update fonts of legend for easier visualization
fig.update_layout(font=dict(size=16))
```

As we can see in Figure 5-18, this interactive plot quickly gives us a sense of the created topics. You can zoom in to view individual documents or double-click a topic on the righthand side to only view it.

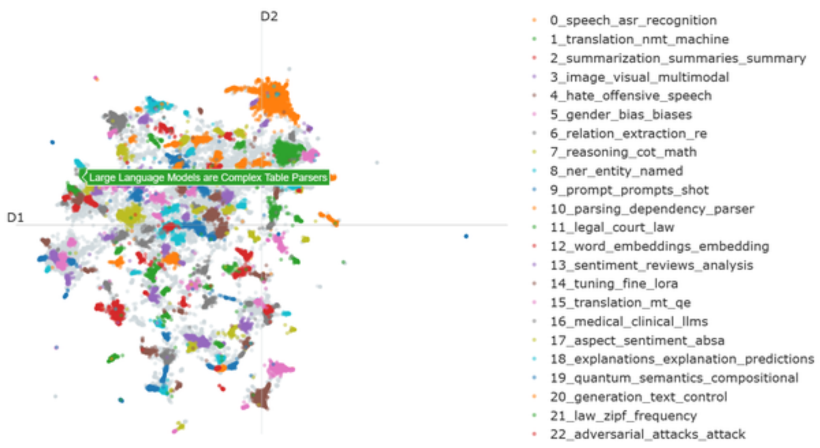


Figure 5-18. The output when we visualize documents and topics.

There is a wide range of visualization options in BERTopic. There are three that are worthwhile to explore to get an idea of the relationships between topics:

```
# Visualize barchart with ranked keywords
topic_model.visualize_barchart()

# Visualize relationships between topics
topic_model.visualize_heatmap(n_clusters=30)

# Visualize the potential hierarchical structure of topics
topic_model.visualize_hierarchy()
```

Adding a Special Lego Block

The pipeline in BERTopic that we have explored thus far, albeit fast and modular, has a disadvantage: it still represents a topic through a bag-of-words without taking into account semantic structures.

The solution is to leverage the strength of the bag-of-words representation, which is its speed to generate a meaningful representation. We can use this first meaningful representation and tweak it using more powerful but slower techniques, like embedding models. As shown in Figure 5-19, we can rerank the initial distribution of words to improve the resulting representation. Note that this idea of reranking an initial set of results is a main staple in neural search, a subject that we cover in Chapter 8.

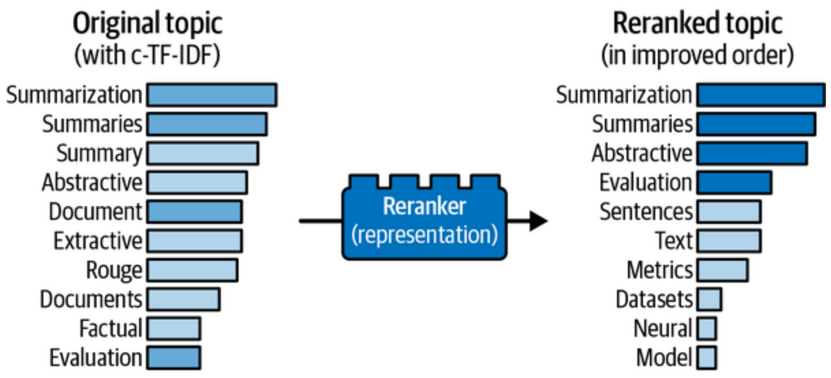


Figure 5-19. Fine-tune the topic representations by reranking the original c-TF-IDF distributions.

As a result, we can design a new Lego block, as shown in Figure 5-20, that takes in this first topic representation and spits out an improved representation. In BERTopic, these are called *representation models*.

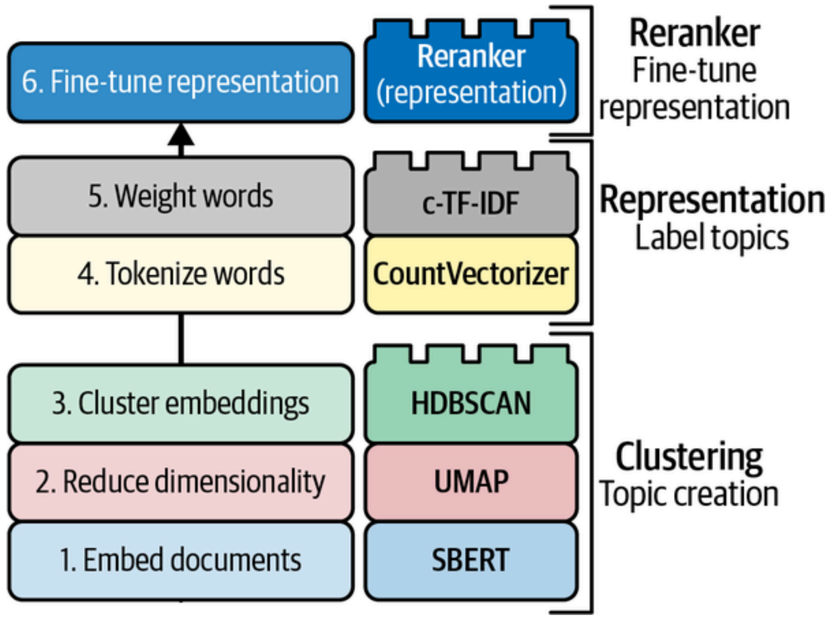


Figure 5-20. The reranker (representation) block sits on top of the c-TF-IDF representation.

In BERTopic, such reranker models are referred to as *representation models*. A major benefit of this approach is that the optimization of topic representations only needs to be done as many times as we have topics. For instance, if we were to have millions of documents and a hundred topics, the representation block only needs to be applied once for every topic instead of for every document.

As shown in Figure 5-21, a wide variety of representation blocks have been designed for BERTopic that allows you to fine-tune the representations. The representation block can even be stacked multiple times to fine-tune representations using different methodologies.

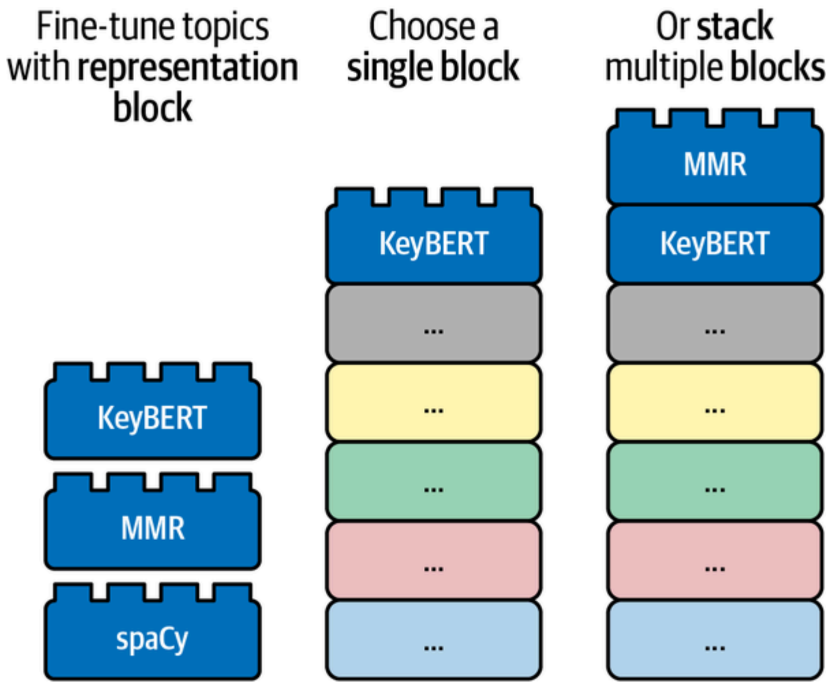


Figure 5-21. After applying the c-TF-IDF weighting, topics can be fine-tuned with a wide variety of representation models, many of which are large language models.

Before we explore how we can use these representation blocks, we first need to do two things. First, we are going to save our original topic representations so that it will be much easier to compare with and without representation models:

```
# Save original representations
from copy import deepcopy
original_topics = deepcopy(topic_model.topic_representations_)
```

Second, let's create a short wrapper that we can use to quickly visualize the differences in topic words to compare with and without representation models:

```
def topic_differences(model, original_topics, nr_topics=5):
    """Show the differences in topic representations between two models """
    df = pd.DataFrame(columns=["Topic", "Original", "Updated"])
    for topic in range(nr_topics):

        # Extract top 5 words per topic per model
        og_words = " | ".join(list(zip(*original_topics[topic]))[0][:5])
        new_words = " | ".join(list(zip(*model.get_topic(topic)))[0][:5])
        df.loc[len(df)] = [topic, og_words, new_words]

    return df
```

KeyBERTInspired

The first representation block that we are going to explore is KeyBERTInspired. KeyBERTInspired is, as you might have guessed, a method inspired by the keyword extraction package, KeyBERT.[7] KeyBERT extracts keywords from texts by comparing word and document embeddings through cosine similarity.

BERTopic uses a similar approach. KeyBERTInspired uses c-TF-IDF to extract the most representative documents per topic by calculating the similarity between a document's c-TF-IDF values and those of the topic they correspond to. As shown in Figure 5-22, the average document embedding per topic is calculated and compared to the embeddings of candidate keywords to rerank the keywords.

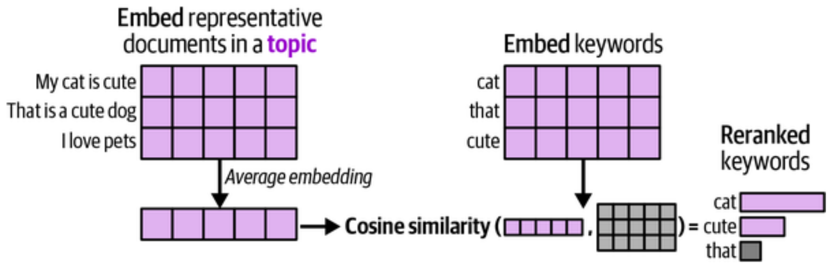


Figure 5-22. KeyBERTInspired representation model procedure.

Due to the modular nature of BERTopic, we can update our initial topic representations with KeyBERTInspired without needing to perform the dimensionality reduction and clustering steps:

```
from bertopic.representation import KeyBERTInspired

# Update our topic representations using KeyBERTInspired
representation_model = KeyBERTInspired()
topic_model.update_topics(abstracts, representation_model=representation_model)

# Show topic differences
topic_differences(topic_model, original_topics)
```

Topic	Original	Updated
0	speech | asr | recognition | end | acoustic	speech | encoder | phonetic | language | trans...
1	medical | clinical | biomedical | patient | he...	nlp | ehr | clinical | biomedical | language
2	sentiment | aspect | analysis | reviews | opinion	aspect | sentiment | aspects | sentiments | cl...
3	translation | nmt | machine | neural | bleu	translation | translating | translate | transl...
4	summarization | summaries | summary | abstract...	summarization | summarizers | summaries | summ...

The updated model shows that the topics are easier to read compared to the original model. It also demonstrates the downside of using embedding-based techniques. Words in the original model, like *nmt* (topic 3), which stands for neural machine translation, are removed as the model could not properly represent the entity. For domain experts, these abbreviations are highly informative.

Maximal marginal relevance

With c-TF-IDF and the previously shown KeyBERTInspired techniques, we still have significant redundancy in the resulting topic representations. For instance, having both the words “summaries” and “summary” in a topic representation introduces redundancy as they are quite similar.

We can use maximal marginal relevance (MMR) to diversify our topic representations. The algorithm attempts to find a set of keywords that are diverse from one another but still relate to the documents they are compared to. It does so by embedding a set of candidate keywords and iteratively calculating the next best keyword to add. Doing so requires setting a diversity parameter, which indicates how diverse keywords need to be.

In BERTopic, we use MMR to go from a set of initial keywords, let’s say 30, to a smaller but more diverse set of keywords, let’s say 10. It filters out redundant words and only keeps words that contribute something new to the topic representation.

Doing so is rather straightforward:

```python
from bertopic.representation import MaximalMarginalRelevance

# Update our topic representations to MaximalMarginalRelevance
representation_model = MaximalMarginalRelevance(diversity=0.2)
topic_model.update_topics(abstracts, representation_model=representation_model)

# Show topic differences
topic_differences(topic_model, original_topics)
```

Topic	Original	Updated
0	speech | asr | recognition | end | acoustic	speech | asr | error | model | training
1	medical | clinical | biomedical | patient | he...	clinical | biomedical | patient | healthcare |...
2	sentiment | aspect | analysis | reviews | opinion	sentiment | analysis | reviews | absa | polarity
3	translation | nmt | machine | neural | bleu	translation | nmt | bleu | parallel | multilin...
4	summarization | summaries | summary | abstract...	summarization | document | extractive | rouge ...

The resulting topics demonstrate more diversity in their representations. For instance, topic 4 only shows one “summary”-like word and instead adds other words that might contribute more to the overall representation.

TIP

Both KeyBERTInspired and MMR are amazing techniques for improving the first set of topic representations. KeyBERTInspired espcially tends to remove nearly all stopwords since it focuses on the semantic relationships between words and documents.

The Text Generation Lego Block

The representation block in BERTopic has been acting as a reranking block in our previous examples. However, as we already explored in the previous chapter, generative models have great potential for a wide variety of tasks.

We can use generative models in BERTopic quite efficiently by following a part of the reranking procedure. Instead of using a generative model to identify the topic of all documents, of which there can potentially be millions, we will use the model to generate a label for our topic. As illustrated in Figure 5-23, instead of generating or reranking keywords, we ask the model to generate a short label based on keywords that were previously generated and a small set of representative documents.

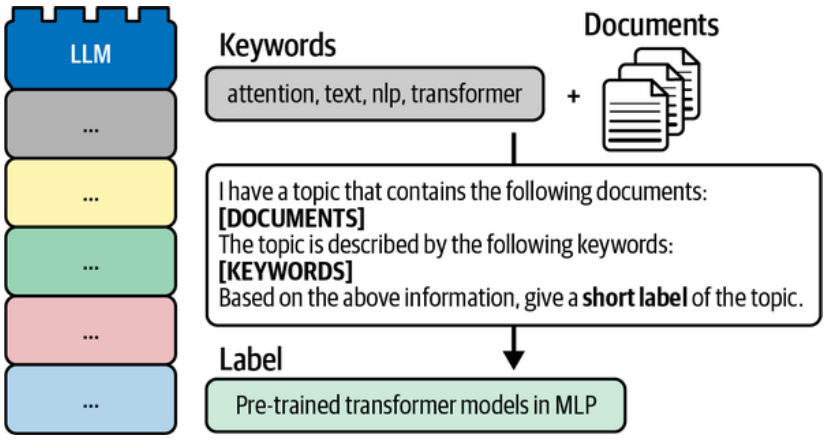

LLM
...
...
...
...
...
Keywords
attention, text, nlp, transformer
+
Documents
I have a topic that contains the following documents:
[DOCUMENTS]
The topic is described by the following keywords:
[KEYWORDS]
Based on the above information, give a short label of the topic.
Label
Pre-trained transformer models in MLP

Figure 5-23. Use text generative LLMs and prompt engineering to create labels for topics from keywords and documents related to each topic.

There are two components to the illustrated prompt. First, the documents that are inserted using the [DOCUMENTS] tag are a small subset of documents, typically four, that best represent the topic. The documents with the highest cosine similarity of their c-TF-IDF values with those of the topic are selected. Second, the keywords that make up a topic are also passed to the prompt and referenced using the [KEYWORDS] tag. The keywords could be generated by c-TF-IDF or any of the other representations we discussed thus far.

As a result, we only need to use the generative model once for every topic, of which there could be potentially hundreds, instead of once for each document, of which there could potentially be millions. There are many generative models that we can choose from, both open-source and proprietary. Let's start with a model that we have explored in the previous chapter, the Flan-T5 model.

We create a prompt that works well with the model and use it in BERTopic through the `representation_model` parameter:

```
from transformers import pipeline
from bertopic.representation import TextGeneration

prompt = """I have a topic that contains the following documents:
[DOCUMENTS]

The topic is described by the following keywords: '[KEYWORDS]'.

Based on the documents and keywords, what is this topic about?"""

# Update our topic representations using Flan-T5
generator = pipeline('text2text-generation', model='google/flan-t5-small')
representation_model = TextGeneration(
    generator, prompt=prompt, doc_length=50, tokenizer="whitespace"
)
topic_model.update_topics(abstracts, representation_model=representation_model)

# Show topic differences
topic_differences(topic_model, original_topics)
```

Topic	Original	Updated
0	speech | asr | recognition | end | acoustic	Speech-to-description
1	medical | clinical | biomedical | patient | he...	Science/Tech
2	sentiment | aspect | analysis | reviews | opinion	Review
3	translation | nmt | machine | neural | bleu	Attention-based neural machine translation
4	summarization | summaries | summary | abstract...	Summarization

Some of these labels, like “Summarization” seem to be logical when comparing them to the original representations. Others, however, like “Science/Tech,” seem quite broad and do not do the original topic justice. Let’s explore instead how OpenAI’s GPT-3.5 would perform considering the model is not only larger but expected to have more linguistic capabilities:

```
import openai
from bertopic.representation import OpenAI

prompt = """
I have a topic that contains the following documents:
[DOCUMENTS]

The topic is described by the following keywords: [KEYWORDS]
```

```
Based on the information above, extract a short topic label in the following format:
topic: <short topic label>
"""

# Update our topic representations using GPT-3.5
client = openai.OpenAI(api_key="YOUR_KEY_HERE")
representation_model = OpenAI(
    client, model="gpt-3.5-turbo", exponential_backoff=True, chat=True, prompt=prompt
)
topic_model.update_topics(abstracts, representation_model=representation_model)

# Show topic differences
topic_differences(topic_model, original_topics)
```

Topic	Original	Updated
0	speech | asr | recognition | end | acoustic	Leveraging External Data for Improving Low-Res...
1	medical | clinical | biomedical | patient | he...	Improved Representation Learning for Biomedica...
2	sentiment | aspect | analysis | reviews | opinion	Advancements in Aspect-Based Sentiment Analys...
3	translation | nmt | machine | neural | bleu	Neural Machine Translation Enhancements
4	summarization | summaries | summary | abstract...	Document Summarization Techniques

The resulting labels are quite impressive! We are not even using GPT-4 and the resulting labels seem to be more informative than our previous example. Note that BERTopic is not confined to only using OpenAI's offering but has local backends as well.

Although it seems like we do not need the keywords anymore, they are still representative of the input documents. No model is perfect and it is generally advised to generate multiple topic representations. BERTopic allows for all topics to be represented by different representations. You could, for example, use KeyBERTInspired, MMR, and GPT-3.5 side by side to get different perspectives on the same topic.

With these GPT-3.5 generated labels, we can create beautiful illustrations using the `datamapplot` package (Figure 5-24):

```
# Visualize topics and documents
fig = topic_model.visualize_document_datamap(
    titles,
    topics=list(range(20)),
    reduced_embeddings=reduced_embeddings,
    width=1200,
    label_font_size=11,
    label_wrap_width=20,
    use_medoids=True,
)
```

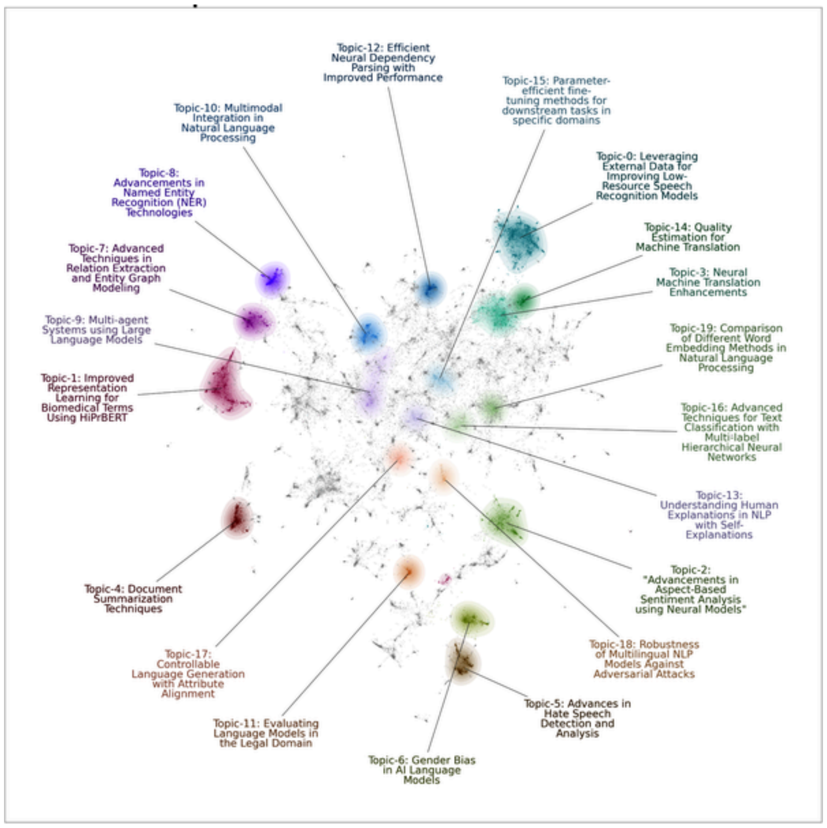


Figure 5-24. The top 20 topics visualized.

Summary

In this chapter, we explored how LLMs, both generative and representative, can be used in the domain of unsupervised learning. Despite supervised methods like classification being prevalent in recent years, unsupervised approaches such as text clustering hold immense potential due to their ability to group texts based on semantic content without prior labeling.

We covered a common pipeline for clustering textual documents that starts with converting input text into numerical representations, which we call embeddings. Then, dimensionality reduction is applied to these embeddings to simplify high-dimensional data for better clustering outcomes. Finally, a clustering algorithm on the dimensionality-reduced embeddings is applied to cluster the input text. Manually inspecting the clusters helped us understand which documents they contained and how to interpret these clusters.

To transition away from this manual inspection, we explored how BERTopic extends this text clustering pipeline with a method for automatically representing the clusters. This methodology is often referred to as topic modeling, which attempts to uncover themes within large amounts of documents. BERTopic generates these topic representations through a bag-of-words approach enhanced with c-TF-IDF, which weighs words based on their cluster relevance and frequency across all clusters.

A major benefit of BERTopic is its modular nature. In BERTopic, you can choose any model in the pipeline, which allows for additional representations of topics that create multiple perspectives of the same topic. We explored maximal marginal relevance and KeyBERTInspired as methodologies to fine-tune the topic representations generated with c-TF-IDF. Additionally, we used the same generative LLMs as in the previous chapter (Flan-T5 and GPT-3.5) to further improve the interpretability of topics by generating highly interpretable labels.

In the next chapter, we shift focus and explore a common method for improving the output of generative models, namely prompt engineering.

[1] Harold Hotelling. "Analysis of a complex of statistical variables into principal components." *Journal of Educational Psychology* 24.6 (1933): 417.

[2] Leland McInnes, John Healy, and James Melville. "UMAP: Uniform Manifold Approximation and Projection for dimension reduction." *arXiv preprint arXiv:1802.03426* (2018).

[3] Leland McInnes, John Healy, and Steve Astels. "hdbscan: Hierarchical density based clustering." *J. Open Source Softw.* 2.11 (2017): 205.

[4] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." *KDD'96,* Aug. 1996: 226-231.

[5] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. "Latent Dirichlet allocation." *Journal of Machine Learning Research* 3.Jan (2003): 993-1022.

[6] Maarten Grootendorst. "BERTopic: Neural topic modeling with a class-based TF-IDF procedure." *arXiv preprint arXiv:2203.05794* (2022).

[7] Maarten Grootendorst. "KeyBERT: Minimal keyword extraction with BERT." (2020).