# TECHNICAL UNIVERSITY OF MUNICH

## TUM Data Innovation Lab

# Deep Learning in Natural Language Processing for Analysis of Document Similarity


| | |
|---|---|
| Authors | Pia Baronetzky, Noir Nigmatov, Theodor Stoican, and Peter Karl Weinberger |
| Mentor | M.Sc. Olena Schüssler<br>Steering Lab by Horváth and Partners GmbH |
| Co-Mentor | M.Sc. Cristina Cipriani |
| Project Lead | Dr. Ricardo Acevedo Cabra (Department of Mathematics) |
| Supervisor | Prof. Dr. Massimo Fornasier (Department of Mathematics) |


Jul 2021

# Abstract

Full-text search based on keyword indexing has been a strongly popular, tried and true approach to search for information in large amounts of textual data. There are plenty of specialized search engines supporting this type of syntactic indexing, e.g. Elasticsearch. However, whenever language in documents is highly context-dependent, full-text search falls apart. Natural human language is rich – words in different contexts have different meanings, and users can often find themselves sifting through a lot of irrelevant results. Moreover, semantically similar texts can consist out of a variety of different words, i.e. synonyms, in such scenario a keyword search is not working properly. It then becomes the user's responsibility to devise complex search queries and techniques to process large amounts of irrelevant data.

In this project our goal is to be able to index a large number of documents and issue simple text queries similarly to the full-text search but have them be contextually as well as semantically aware. Thus, the search should also return documents that are apparently different, but semantically very close to the query – a functionality not available in the traditional full-text search.

Recent advances in natural language processing (NLP) research, namely the evolution of transformer-based models, have set up the state-of-the-art for various downstream tasks by a large margin. In our work – in order to be able to quickly search indexed documents – the cosine similarity metric is used to compare the documents within the high-dimensional embedding space. There are two ways of generating the embeddings. By using Sentence-BERT models, which are specifically pre-trained for semantic textual similarity tasks [20], we are able to create two meaningful embeddings exploiting DistilRoberta and Multilingual Universal Sentence Encoder models. Another approach consists of first performing unsupervised topic modeling and then computing vectors of topic probabilities for each indexed document. We use Contextualized Topic Models (CTM) [2] and BERTopic [12] to produce the vectors. Further on, we integrate Elastic's full-text search with the combined weighted semantic approach, building on both topic modeling and contextual document embeddings. By using keyword indexing, it is possible to include or exclude specific terms in the query. The weighted linear combination of topic modeling and Sentence-BERT embeddings allows to see different levels of semantic similarity, from very general topic-based to the finer more narrow search results. Additionally, we also applied a re-ranking of the search results using a heavy and more accurate Cross-Encoder model "ms-marco-MiniLM-L-6-v2". A re-ranker based on a cross-encoder can substantially improve the final results for the user.

Furthermore, the implementation can be easily packaged and delivered to apply to a real world scenario as it was developed in a modular way prioritizing the computational feasibility of the proposed semantic search approach.

# Contents

# 1 Introduction

Search engines have undoubtedly changed the world. The ability to find documents that contain *The US president* or *German chancellor*, i.e. by keyword indexing, has been a strongly popular, tried and true approach to search for information in large amounts of textual data. There are plenty of specialized search engines supporting this type of syntactic indexing. The commercially successful Elasticsearch engine is one of them (Figure 1).

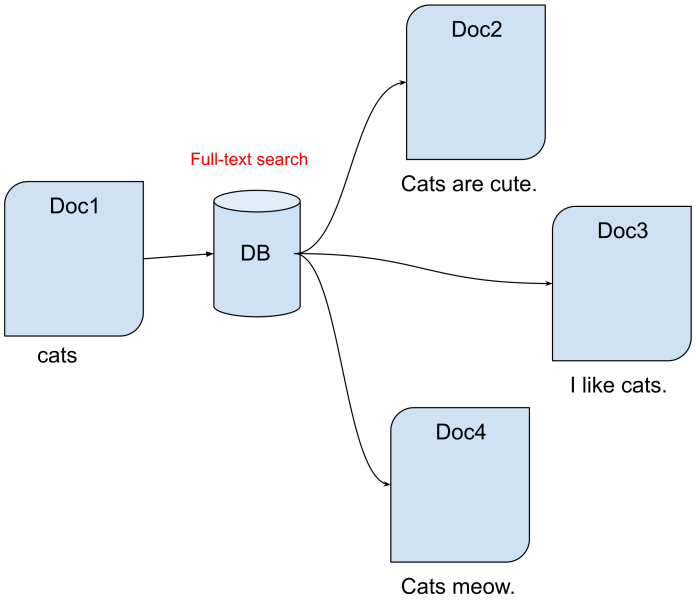



Figure 1: Full-text search with a database engine

However, whenever language in those documents is highly context-dependent, full-text search falls apart. Natural human language is rich – words in different contexts have different meanings, and users can often find themselves sifting through a lot of irrelevant results. Moreover, semantically similar texts often take use of different words, e.g. synonyms, and hence, in full-text search, these are just left out. For instance, let us assume that someone searches for "*painkillers for dogs*". In that case, they might want to find out either particular instances or more general instances (semantically speaking) of that query, such as "*ibuprofen for pets*". Or maybe they would be curious to find out about side effects of painkillers for pets. Standard full-text search would be unable to find such results which are technically close to the main query in the semantic space.

## 1.1 Problem Definition and Goals

With the advent of Deep Learning, however, such capabilities – as described in the last section – become imperative. The state-of-the-art NLP models (based on *transformers* primarily) are making many of the classically unsolvable NLP problems perfectly feasible. Semantic search is naturally one of them. In this work, we develop a semantic search engine (for news articles) based on transformers and Elasticsearch. The final goal of the

project is to be able to index a large number of documents and issue simple text queries similarly to the full-text search but have them be context- and semantically aware. Thus the search should also return documents that are apparently different but semantically very close to the query (Figure 2) – a functionality not available in the traditional full-text search.

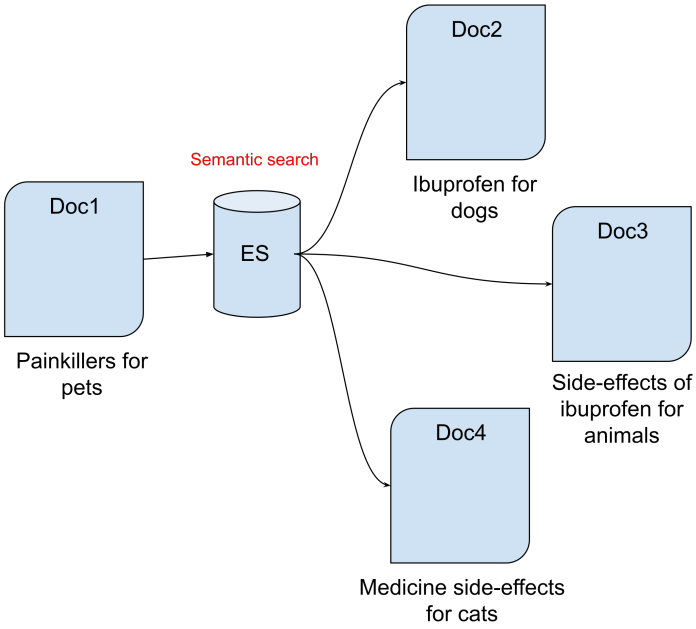



Figure 2: Semantic search with a database engine

Additionally, this search is to be extended to multiple input documents (each document would serve as an input query). Furthermore, this idea of finding documents similar to a set of input documents goes very much in the direction of finding a common theme or topic among these documents. Hence, the next objective of the project is topic modeling and topic-based search, i.e. fetching documents which are similar (in terms of the topic) to the input documents (Figure 3).

In the end, we bundle all software resources to be able to present a proof of concept application, which shows our final implementation and findings. The final proof of concept will combine the semantic, context-aware search built by the searching in the high-dimensional vector space with a classical keyword search. Moreover, this proof of concept will be also capable to demonstrate the power of semantic search driven by topic modeling. Finally, we are able to combine both semantic search engines to allow the user to decide how he or she wants to apply the search.

## 1.2 Report Structure

In Section 2, we present the overall high-level idea of our project outcome and the system architecture we are using. Then in Section 3, we investigate the provided data and describe the preprocessing steps, which are necessary in order to apply deep learning models on the data. Next, we express in detail our implementation in Section 4. Finally, we show

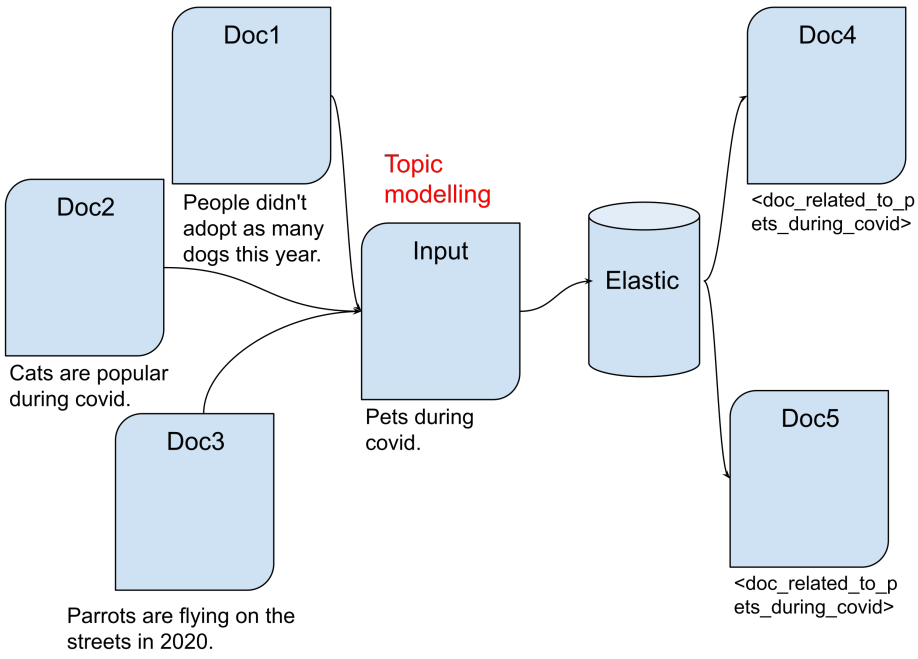



Figure 3: Topic-based search with a database engine

our end-to-end prototype in Section 5, and provide a conclusion as well as an outlook in Section 6 and Section 7 respectively.

# 2 General Approach

Within this section, we address the high-level intuition behind our implementation. Therefore, in Section 2.1, the process of taking deep learning models for natural language processing into account is described in more detail. In Section 2.2, we illustrate the underlying architecture of our software solution.

## 2.1 Model

In order to be able to quickly search indexed documents, the cosine similarity metric is used to compare the high-dimensional embedding vectors, which are compact representations of the query and the document indexed in Elasticsearch. The search results are then ordered based on the score from the highest to the lowest. There are two different ways to create such embeddings. By using Sentence-BERT models, which are specifically pre-trained for semantic textual similarity tasks [20], we are able to create two meaningful embeddings exploiting DistilRoberta and Multilingual Universal Sentence Encoder models with dimensionalities of 768 and 512 respectively. Another approach consists of first performing unsupervised topic modeling on a subset of documents and then computing vectors of topic probabilities for each indexed document. We use Contextualized Topic Models [2] and BERTopic [12] to produce vectors with pre-computed 10 and 50 topics for

each model, with the Contextualized Topic Models for 50 topics producing the arguably most accurate topics. Further on, we integrate Elastic's full-text search with the combined weighted semantic approach. By using keyword indexing, it is possible to include or exclude specific terms in the query. A weighted linear combination of topic modeling and Sentence-BERT embeddings allows to see different levels of semantic similarity, from very general topic-based to the finer more narrow search results.

### 2.2 Architecture and Hardware

Elasticsearch is used as a back-end storage for the indexed documents. Besides providing a scalable full-text search engine with near real-time performance, it allows to store dense vectors (dim. up to 2048) and perform document scoring (ordering from the highest to lowest) based on cosine similarity. Moreover, Elasticsearch provides a simple extensive REST API interface so that new documents can be easily added to the index. One apparent disadvantage is that the basic Elasticsearch does not yet support a particularly relevant optimization – approximate nearest neighbors (ANN) – necessary for a billion-scale low-latency performance. This disadvantage in particular is still an open issue of vanilla Elasticsearch [15]. Another viable solution in such case is to use the proprietary GSI's Elasticsearch k-NN Plugin [13]. Amazon Elasticsearch Service also provides ANN optimization as a built-in functionality [1].

We used the LRZ Compute Cloud to create an instance with 10 CPUs, up to 45 GBs of RAM, and 200 GBs of disk space (the maximum limit). We deployed the latest Elasticsearch (v. 7.12) through Docker. Huggingface Transformers [29] and Sentence-BERT [19] pre-trained models were installed via pip and Python was used for development. The user interface to perform actual search is built as a lightweight web application using Streamlit [27]. Communication with Elasticsearch is ensured over the REST API.

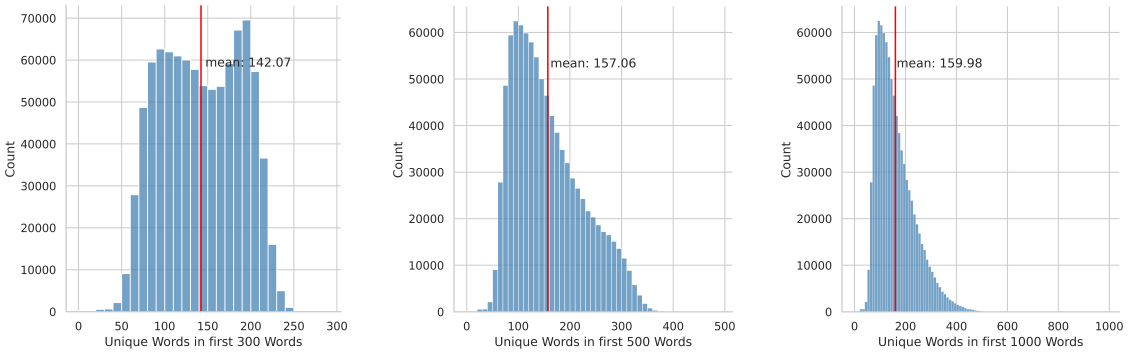
Due to the sheer amount of computations needed to apply large cutting-edge NLP models, we secured access to the LRZ AI infrastructure featuring latest NVIDIA P100 and V100 GPUs. We computed transformer embeddings and performed training of topic models solely on GPUs. The overall GPU compute time is about 250 hours.

## 3 Data

The most important part of any machine learning project is data. The industry partner of this project supplied approximately 20 million news articles, which enabled our work with cutting-edge transformer models. Due to the small volume of disk space on the LRZ Compute Cloud, and more importantly the sheer massiveness of the data, we reduced the operational size of the data to about 2 million news articles, or 8 GB of storage. This limitation was necessary in order to enable us to apply inference with large transformer-based models. However, before we can apply any model, we have to explore, analyze, and preprocess the data. All those steps are explained in more detail in Section 3.1 and Section 3.2. Furthermore, we also experimented with summarization techniques as an additional preprocessing step – described in Section 3.3.

## 3.1 Data Exploration

Before applying machine learning models to data, one has to fully understand what the data looks like. To gain a better understanding of the provided data, we conducted preliminary analysis and visualization.




(a) Distribution of the unique first 300 words.

(b) Distribution of the unique first 500 words.

(c) Distribution of the unique first 1000 words.

Figure 4: Illustration of the distributions of the first unique 300, 500, and 1000 words of 1 million news articles.

In Figure 4, we analyzed the count of unique words within the first 300, 500, and 1000 words of each article of a sample of 1 million documents, which is $\frac{1}{20}$ of the total data and $\frac{1}{2}$ of the data used. The models which we are applying to the data have only a certain number of words, which they can take into account for one sequence. Therefore, we had to ensure that we do not lose information while we are only taking the first words till the word limit into account. From our analysis, we see that we do not omit information even if we take only the first 300 words into account, because the average unique words observed is really close to the average unique words observed within the first 500, or the first 1000 words.

The 1 million sample articles are crawled in the time frame from January 2019 to May 2019. They are uniformly distributed over time and also the word count per article is uniformly distributed over the time frame. We also analyzed if there is a heavy bias within the news articles, i.e. if there are dominating topics which would lead to a biased search engine. From our perspective, the corpora obviously includes many articles about then U.S. President Donald J. Trump, in addition to various other topics, but given the historical context of the time period from which the articles originated and the nature of journalism, we feel that we should leave the corpus as it is.

Furthermore, we wanted to analyze where the news articles are crawled from. From our analysis, we can report that the articles are crawled from numerous news pages around the world, e.g. “reuters.com”. However, for the majority of the articles (roughly $800,000$ articles), the field which includes the information of the source is not filled. So we can not tell precisely from which parts of the world the data is crawled, but we can say for sure that the corpus includes mainly news articles.

## 3.2 Data Preprocessing

Another necessary step before we can apply deep learning models for natural language processing is to preprocess the data in order to ensure that the models can operate properly. In detail, we had to apply initial preprocessing and deduplication.

### Initial Data Preprocessing

To ensure that the data has been crawled from reliable sources, articles containing "twitter" in their URL were removed from the data set. Moreover, articles were filtered to have a word count between 200 and 2000. Articles in any other language than English were removed. This resulted in a data reduction of 25%. Some articles contained HTML entities that were also removed.

### Deduplication

Because the original data was regularly crawled from news websites, we encountered many duplicates of the very same article. Some articles had up to 10-15 copies, though with minor changes. We decided to use a classic NLP method – the Jaccard similarity index on sets of words (1-grams) to filter syntactically similar articles due to its simplicity and speed:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{1}$$

However, some further simplifications were made: splitting of data into chunks of 3 GBs each, taking only the first paragraph (about 100 words), and comparing an article with next 1000 articles. Otherwise it would have been computationally infeasible. With such simplifications it took 12-15 hours to process a 3 GB chunk on the cloud instance. In general, this resulted in 10-15 % reduction of size.

Overall, after the preprocessing, the original 3 GB chunks got reduced to 2 GBs chunks, each having around 500k documents. We proceeded further with 4 chunks of data, hence 8 GBs of data.

## 3.3 Summarization

Apparently, the transformer models described in Section 4 that are used to compute text embeddings can take the maximum of 512 tokens as input. However, some documents exceed this limit in length. In such a case, and as some blog posts point out, it is sufficient to use the first 512 tokens of the input, and truncate the rest. The natural logic behind this is that practically all news articles explain the gist of it in the first one or rarely two paragraphs. Furthermore, we investigated the number of unique words in a sample of 1 million articles – as described in Section 3.1. Our analysis shows that we do not lose much information in this regards if we take the first e.g. 300 words (which yields in approx. 512 tokens) into account. Nevertheless, we decided to compare it with extractive summaries produced for long texts on a small subset of data. We intentionally did not

use abstractive summarization, which would introduce new words and thus, might change the meaning. We tested and timed three models:

1. BERT Extractive Summarizer [18]
2. Latent Semantic Analysis (LSA) [11]
3. LexRank [10]

All three models produced valid extractive summaries, the difference being in speed. The slowest proved to be the BERT summarizer, taking about 0.5 seconds for one document on a V100 GPU. LSA takes on average 0.2 seconds, whereas LexRank is 10 times faster. Overall, producing extractive summaries using LexRank would require additional 11 hours of preprocessing time for the 2 million documents. Further on, we computed text embeddings on the summaries and performed semantic search. The results actually proved to be worse than taking just the first 512 tokens of the text into account. Since LexRank uses Sentence Centrality and Centroid-based summarization techniques, the search results are distributed across different topics identified in the summaries. Thus, we dropped the idea of using extractive summarization.

# 4 Implementation

The transformer models BERT [9] and RoBERTa [17] have set a state-of-the-art performance on semantic textual similarity tasks. They compare sentences pairwise to obtain a similarity score, which causes a large computational overhead. Comparing 10,000 sentences requires about 50 million inference computations (65 hours on a modern V100 GPU). The construction of BERT makes it unsuitable for semantic similarity search inside a large collection of documents. Having a fixed query text and running pairwise inferences with all the documents is quite cumbersome and infeasible in terms of performance. Our goal is rather to move the bulk of computations away from the user, and have cheap fast comparisons during the retrieval, similar to full-text search utilizing keyword indexing. We decided to first produce embeddings for each document that compactly represent it as a high-dimensional vector of numbers. The semantically similar documents in this case should then have vectors that are close in that embedding space. Thus, during the retrieval step, the dot product of vectors produces the cosine similarity, which is not an expensive operation. Elasticsearch is used to store and index the collection of documents to be searched, as it provides the specially designed "dense_vector" type and the scoring mechanism based on the cosine similarity between the query vector and the vector of each item in the index.

## 4.1 Transformer-based bare models

Since transformer-based models perform well on sentence pair regression tasks, our first naïve approach was to use these models as they are, and extract the embeddings from the hidden states of the encoder. The logic behind this was that theoretically the hidden states should represent the contextual encodings of the input text because the models are trained to perform language modeling. We experimented with four models on a small subset of documents:

1. **DistilBERT** [28]: The last hidden state (dim. 768) is used as the embedding. The distilled version was chosen due its speed compared to the original BERT – it is 60% faster and preserves 95% of BERT's accuracy as measured on the GLUE language understanding benchmark.

2. **GPT-2** [31]: The last hidden state (dim. 768) is used as the embedding. The speed is rather slow, as GPT-2 contains 1.5 billion parameters. At the time of its release in 2019, the model was the best in text generation domain.

3. **Pegasus** [32]: This model is used for abstractive summarization. At the time of its release in 2019, Pegasus achieved state-of-the-art summarization performance. The structure consists of both an encoder and decoder. The last hidden state of the encoder (dim. 1024) is used as embedding. The latency is also big, due to the fact that the input goes through the entire model to produce an abstractive summary.

4. **T5** [33]: This model achieves state-of-the-art results at semantic similarity tasks when comparing sentences pairwise. Creating embeddings by using the last hidden state (dim. 768) is rather slow.

For all four models, we used mean pooling of the hidden state vectors representing each input token to produce a single vector. To speed up the prototyping and testing phase, we used FAISS [22] – a library for efficient in-memory similarity search and clustering of dense vectors. However, the expectations were not met – the search results were not even semantically similar to the query. That is, the produced embeddings were not meaningful in the embedding space. The point initially overlooked by us was that these models were not created and trained with the aim of producing meaningful embeddings.

## 4.2 Sentence-BERT

After doing an extensive research, we discovered that sentence embeddings were a well studied area with a number of proposed methods. Skip-Thought [16] trains an encoder-decoder architecture to predict the surrounding sentences. InferSent [8] uses labeled data of the Stanford Natural Language Inference (SNLI) data set [5] and the Multi-Genre NLI (MNLI) data set [34] to train a siamese BiLSTM network with max-pooling over the output. Conneau et al. showed, that InferSent consistently outperforms unsupervised methods like Skip-Thought. Universal Sentence Encoder [6] trains a transformer network and augments unsupervised learning with training on SNLI. Hill et al. [14] showed, that the task on which sentence embeddings are trained significantly impacts their quality. Previous work (Conneau et al., 2017 [8]; Cer et al., 2018 [6]) found that the SNLI data sets are suitable for training sentence embeddings. Thus, in the first place we should have used models that were specifically additionally fine-tuned for the semantic textual similarity tasks using the SNLI data sets. Further research revealed Sentence-BERT (SBERT) by Reimers et al. [20], a modification of the BERT/RoBERTa network that is able to derive semantically meaningful sentence embeddings. The architecture of SBERT intentionally uses siamese and triplet networks [23] so that the sole aim of the model is to produce meaningful embeddings that can be further compared with cosine similarity. Additionally, SBERT is trained on the combination of the SNLI [5], the Multi-Genre NLI [34], and

| Model | STS12 | STS13 | STS14 | STS15 | STS16 | STSb | SICK-R | Avg. |
|---|---|---|---|---|---|---|---|---|
| Avg. GloVe embeddings | 55.14 | 70.66 | 59.73 | 68.25 | 63.66 | 58.02 | 53.76 | 61.32 |
| Avg. BERT embeddings | 38.78 | 57.98 | 57.98 | 63.15 | 61.06 | 46.35 | 58.40 | 54.81 |
| InferSent - GloVe | 52.86 | 66.75 | 66.75 | 72.77 | 66.87 | 68.03 | 65.65 | 65.01 |
| Universal Sentence Encoder | 64.49 | 67.80 | 67.80 | 76.83 | 73.18 | 74.92 | 76.69 | 71.22 |
| SBERT-NLI-base | 70.97 | 76.53 | 73.19 | 79.09 | 74.30 | 77.03 | 72.91 | 74.89 |
| SBERT-NLI-large | 72.27 | 78.48 | 74.90 | 80.99 | 76.25 | 79.23 | 73.75 | 76.55 |
| SBERT-STSb-base | - | - | - | - | - | 84.67 | - | 84.67 |
| SBERT-STSb-large | - | - | - | - | - | 84.45 | - | 84.45 |
| SRoBERTa-NLI-base | 71.54 | 72.49 | 70.80 | 78.74 | 73.69 | 77.77 | 74.46 | 74.21 |
| SRoBERTa-NLI-large | 74.53 | 77.00 | 73.18 | 81.85 | 76.82 | 79.10 | 74.29 | 76.68 |
| SRoBERTa-STSb-base | - | - | - | - | - | 84.92 | - | 84.92 |
| SRoBERTa-STSb-large | - | - | - | - | - | 85.02 | - | 85.02 |

Table 1: Spearman-rank correlation $\rho$ between the cosine similarity of sentence representations and the gold labels for various semantic textual similarity (STS) tasks. Performance is reported by convention as $\rho \times 100$. STS12-STS16: SemEval 2012-2016, STSb: STSbenchmark, SICK-R: SICK relatedness data set.

the STS benchmark (STSb) [7] data set. The SNLI is a collection of 570,000 sentence pairs annotated with the labels contradiction, entailment, and neutral. Multi-Genre NLI contains 430,000 sentence pairs and covers a range of genres of spoken and written text. The STSb provides a popular data set to evaluate supervised STS systems. The data includes 8,628 sentence pairs from the three categories `captions`, `news`, and `forums`. It is divided into train (5,749), dev (1,500), and test (1,379).

As a result, SBERT embeddings significantly outperform other state-of-the-art sentence embedding methods like InferSent [8] and Universal Sentence Encoder [6]. Moreover, the results of experiments by Reimers et al. show that directly using the output of BERT leads to rather poor performances. The results are depicted in Table 1.

Ultimately, we concluded that SBERT fits perfectly into our project to achieve the goal, namely produce textual embeddings that can be used for scalable semantic search. Further research showed that there existed already an open-source implementation of SBERT called SentenceTransformers [24]. It is a Python framework based on PyTorch and HuggingFace's Transformers [29] and offers a large collection of pre-trained models [19] tuned for various tasks. Table 2 lists some of the already available pre-trained models, ranked in descending order of the Spearman-rank correlation on the STSb test set.

Consequently, for the second round of experiments we chose three models:

1. stsb-mpnet-base-v2: the base model is MPNET (Microsoft) [25]. Training data: NLI + STSb. It has the highest Spearman-rank correlation.

2. stsb-distilroberta-base-v2: the base model is DistilRoBERTa-base. Training data: NLI + STSb. The rank is 2 points less, but the inference speed is almost 1.5 times higher than the speed of mpnet.

3. distiluse-base-multilingual-cased-v1: distilled version of the Multilingual Universal

| Model Name | STSb | Speed |
|---|---|---|
| stsb-mpnet-base-v2 | 88.57 | 2800 |
| stsb-roberta-base-v2 | 87.21 | 2300 |
| paraphrase-mpnet-base-v2 | 86.99 | 2800 |
| paraphrase-multilingual-mpnet-base-v2 | 86.82 | 2500 |
| nli-mpnet-base-v2 | 86.53 | 2800 |
| stsb-distilroberta-base-v2 | 86.41 | 4000 |
| nli-roberta-base-v2 | 85.54 | 2300 |
| paraphrase-distilroberta-base-v2 | 85.37 | 4000 |

Table 2: Top 8 models on the STSb test set (in descending order). Speed is described as encoding speed (sentences per second) on a V100 GPU. [19]

> Sentence Encoder for 15 languages: Arabic, Chinese, Dutch, English, French, German, Italian, Korean, Polish, Portuguese, Russian, Spanish, Turkish. The rank is 80.62 and the inference speed is 4000 sentences/sec. This model was chosen because it is multilingual and the inference speed is the highest among other multilingual models though with slightly higher ranks.

We computed new embeddings on 50,000 documents randomly taken from the data. This time, the expectations were actually met. The search returned semantically similar documents, ranking the semantically similar articles – which covered the same context – the highest. Additionally, as already mentioned in Section 3.3, we produced extractive summaries using LexRank algorithm and computed mpnet and distilroberta embeddings on the extracts. However, this approach produced worse results, as the output had a mix of articles based on different topics identified by LexRank. For instance, a query article about measles outbreak in Vancouver airport because of two returning unvaccinated passengers from the Philippines produced different independent articles on measles outbreak, Vancouver airport, vaccination, and the Philippines.

After doing an extensive human evaluation by our project team, we selected DistilRoBERTa because it prfoduced the best results, and the multilingual Distiluse which also performed quite well. Both models yield the fastest inference speed among other models, 4000 sentences/sec on a V100 GPU. Further, we computed two embeddings for each document of our working subset, approx. 2 million articles. This operation took approximately 8 hours on a single V100 GPU of DGX-1 node. Then the documents and its corresponding embeddings were indexed in the Elasticsearch instance.

## 4.3 Topic Modeling

Topic models are entirely about uncovering the hidden topical patterns which pervade a collection of texts. Those patterns are called topics [4]. In topic modeling, each document consists of a certain number of topics derived from the text collection, and each topic is composed of a certain group of words. After applying a topic model to a text collection it is possible to annotate the documents according to the assigned topics and then use those annotations to organize, summarize, and search the hitherto unlabeled texts. An example

algorithm of this kind of natural language processing models is displayed in Figure 5 (to illustrate the pure character of a topic model).

Because of our goal of designing a cutting-edge semantic search engine for documents, we go one step further and use each documents' topic distribution as a vector space, which we exploit for searching similar documents. To be more precise, we experimented with two cutting-edge models: Contextualized Topic Models and BERTopic. Both models take advantage of embeddings created by transformer models and will be explained in more detail in Section 4.3.1 and Section 4.3.2.

### 4.3.1 Contextualized Topic Models

Contextualized Topic Models [3] (CTM) is a topic modeling technique which builds on top of ProdLDA. ProdLDA is the yielded model by Autoencoding Variational Inference for Topic Models (AVITM) [26] which is, in the end, an altered version of plain LDA. They simply replace the word-level multinomial distribution assumption with a weighted product of experts. This means that they replaced line 5 of the algorithm shown in Figure 5 with $w_n|\beta, \theta \sim Multinomial(1, \sigma(\beta\theta))$. The function $\sigma$ is indeed the AVITM blackbox inference method, which returns $\mu$ and $\Sigma$ of the multinomial distribution after learning the hidden structures within the data via applying a neural autoencoder with variational inference targeting the reconstruction of the input. They claim that the most significant advantages of ProdLDA over LDA are the better topic coherence, and the computational efficiency. Latter is due the fact, that on unseen data, AVITM does only require to pass through the forward pass of a neural network.

1: **for** document $d_d$ in corpus $D$ **do**
2: Choose $\theta_d \sim$ Dirichlet($\alpha$)
3: **for** position $w$ in $d_d$ **do**
4: Choose a topic $z_w \sim$ Multinomial($\theta_d$)
5: Choose a word $w_w$ from $p(w_w|z_w, \beta)$, a multinomial distribution over words conditioned on the topic and the prior $\beta$.
6: **end for**
7: **end for**

Figure 5: Essential latent dirichlet allocation algorithm (LDA) derived by [4]. $\alpha$ and $\beta$ are a priori intialized parameters of the algorithm. ProdLDA and therefore also CTM is based on LDA.

The authors of [3] extended the bag of words approach of ProdLDA by adding contextualized information to the input. This minor change of ProdLDA then yields even more coherent topics.

There are several reasons why we have chosen CTM as a topic model. The authors of the model claim that this model outperforms other state-of-the-art topic models. Furthermore, we can utilize arbitrary sentence BERT embeddings. That means that one can reuse the embeddings created for the semantic search for the training of a topic model,

which enables an efficient machine learning pipeline.

We conducted experiments with various parameters and variable number of documents taken into account. In the end, we decided to include two models in the final proof of concept. Both models are trained on 100,000 documents. The documents are embedded with "stsb-distilroberta-base-v2", which is the same model used for the embeddings. Furthermore, both models are trained with 100 epochs with a batch size of 256. The batch size is restricted by the amount of VRAM available on the GPU of the LRZ AI cluster which is 15 GB. The only difference between the two models is the number of topics. One model is trained with 10 and the other with 50. In early evaluations, we saw that the model with 50 topics yields more meaningful results and should be a model to consider for a real product, while the model with 10 topics is more suitable for a demo showcase.

For inference without any training of any part of the model – which is necessary in an productive environment – we had to make some changes within the source code of CTM. For example, the developers of CTM did not consider that while the model can be trained on a GPU, there would be no GPU available for inference. After the submission of this project, we will open pull requests accordingly on the open source platform GitHub to contribute and share our improvements with the community.

### 4.3.2 BERTopic

Traditional topic modeling has been done by considering the bag-of-words assumption for underlying documents. With the advent of transformers though, smarter ways to encode documents have been developed. One such encoding method is BERT. BERT is meant to capture contextualized information from the underlying document in a better way that classic methods (e.g. bag-of-words) could ever do. As its name also suggests, BERTopic leverages BERT for codifying (or embedding) a document as well as possible and uses those embeddings in further steps (Figure 6).

#### Intuition

BERTopic is essentially a clustering algorithm. In a nutshell, what it does is a sort of embedding-based clustering. The documents whose embeddings are clustered together share a common topic.

#### Dimensionality Reduction

The clustering algorithm used as part of BERTopic is HDBSCAN. However, HDBSCAN is highly sensitive to the course of dimensionality, requiring large amounts of data for large-sized embeddings. BERT spits out rather some large-dimensional vectors, so, for that reason, the UMAP algorithm is used in order to reduce the dimensionalities of those embeddings before actually feeding them into HDBSCAN.

[1] ©https://maartengr.github.io/BERTopic/tutorial/algorithm/algorithm.html

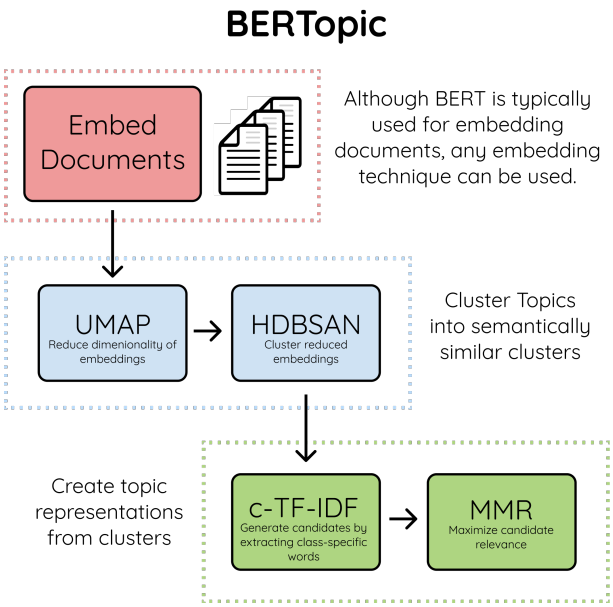



Figure 6: BERTopic – under the hood [1]

### Clustering

After generating contextualized embeddings using BERT and subsequently reducing their dimensionality, we have extracted some meaningful vectors which fully characterize the documents. At this point, HDBSCAN is used to cluster this documents using a custom distance function, such as the euclidean distance, and find similar documents.

### Finding Topics

At this point, all the semantically similar documents are clustered together. That is, all the documents which belong to one cluster talk about a common thing in one way or another. The question that arises now is: *How can this common thing be identified?* The answer is *TF-IDF*. Or more precisely, a cluster-based version of TF-IDF. In more detail, what is done as part of this algorithm, is an aggregation of documents which belong to the same cluster. That means that all documents which belong to the same cluster are considered as one large document. Consequently, the *common thing* we spoke about before is identified by computing the most significant terms within these clusters via TF-IDF. The $k$ most significant terms ($k$ being configurable) are then used for characterizing a topic.

### Training

Like most clustering algorithms, BERTopic is also highly configurable. Starting with choosing the number of clusters to changing the underlying model which generates the embeddings, everything can be customized. In the current set-up, the following configuration has been used:

- stop words removal

- a pre-trained model to generate embeddings: *paraphrase-MiniLM-L6-v2*
- $k = 10$ minimum documents required to create a cluster
- 100k documents as the training data set

Normal configuration with BERTopic produces a multitude of outliers. At first, 47% of the data was classified as outliers. By empirical investigation, we figured out that $k = 10$ minimum documents (outlined by the 2 parameters *min_samples* and *min_cluster_size* in Table 3) for producing a cluster yields very good results, by reducing the number of clustering outliers to 17%. Last but not least, BERTopic was left to produce as many topics as possible, by letting the corresponding parameter free. Subsequent reduction to 10 and 50 topics respectively was performed, ending up with 2 models (as in the case of CTM).

| | **min_samples** | **min_cluster_size** | **Outliers** |
|---|---|---|---|
| **BERTopic** | 1 | 2 | 17.4% |
| | 1 | 10 | 17% |
| | 1 | 50 | 28% |

Table 3: BERTopic – outliers experiments

## 4.4 Retrieve and Re-Rank

As pointed out by Reimers et al. [20], SBERT embeddings serve the purpose of performing scalable semantic search. The reader might imagine a situation where the user's query suggested a question for which the results should return documents containing valid answers first rather than the general pool of semantically similar documents. Correspondingly, Reimers et al. also suggest that for complex search tasks, e.g. for question answering retrieval, the search can significantly be improved by using additional re-ranking after the initial retrieval step.

In a retrieve and re-rank pipeline such as in Figure 7, a set of documents as well as an input document are passed to the retrieving model first. The top $k$ articles returned from the retrieving model are then passed to the re-ranker, which compares them to the input query and re-ranks them.

As a re-ranking model, Reimers et al. suggest to use a Cross-Encoder. The input text and a document from the data set are passed simultaneously to the transformer network, which then outputs a score indicating how relevant the document is for the given input. The advantage of Cross-Encoders is the higher performance, as they perform attention across the input and the document [21]. However, they do not produce embeddings, thus cannot be used for scalable semantic retrieval.

Among different variations and models, we chose the "ms-marco-MiniLM-L-6-v2" [30] model as a re-ranking model in our search engine. This decision is based on a ranking

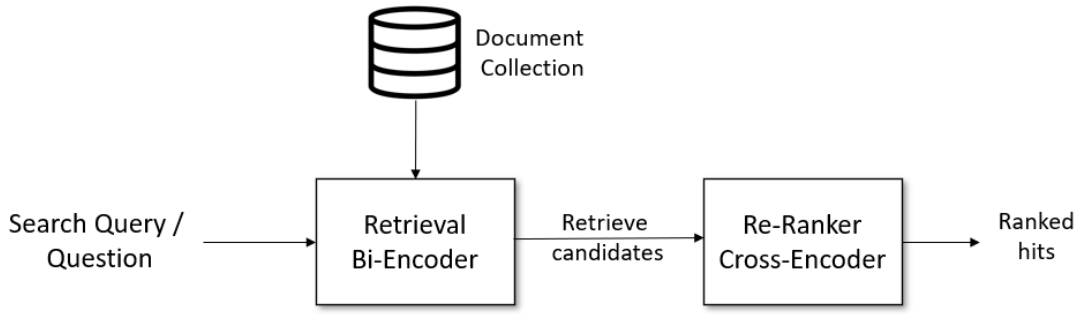



Figure 7: Retrieve and Re-Rank Pipeline [21].

of models trained on the MS MARCO Passage Ranking data set [30]. MS MARCO is a large scale information retrieval corpus that was created based on real user search queries using the Bing search engine. The provided models can be used for semantic search using given keywords or a search phrase. As a result, they will find passages that are relevant for the search query.

Thus, our Cross-Encoder re-ranker does not only pay attention to the semantic similarity of the results, it also pays attention to the relevance of the results to the user. The re-ranker is included as an option in our search engine.

# 5 Prototype

Our search engine uses a combination of Topic Modeling and Sentence-BERT embeddings by default. Our application supports the following functionality:

- A slider on the left indicates to how much percent the search should be based on Topic Modeling.
- Keywords to be included or excluded in the search can be provided.
- The re-ranker can be chosen as an option.
- For multilingual use cases, the transformer model will be changed from the Distil-RoBERTa to Distiluse.
- For a search based on several articles, up to 3 articles can be added in text fields or a list of articles can be uploaded in an Excel file. For semantic search, the embeddings of several input files will be averaged apriori to the search.

The search results go through one more deduplication step as not all duplicates could be deleted while preprocessing the data in small portions.

## 5.1 Sentence Embeddings

Searching for documents semantically similar to an article about a “looting” of shops in Johannesburg, South Africa, the search engine based only on DistilRoBERTa embeddings

returns closely related documents. The first result depicted in Figure 8 is about "sporadic attacks" on shops by protesters in Johannesburg, the second one about riots in Pretoria, South Africa.

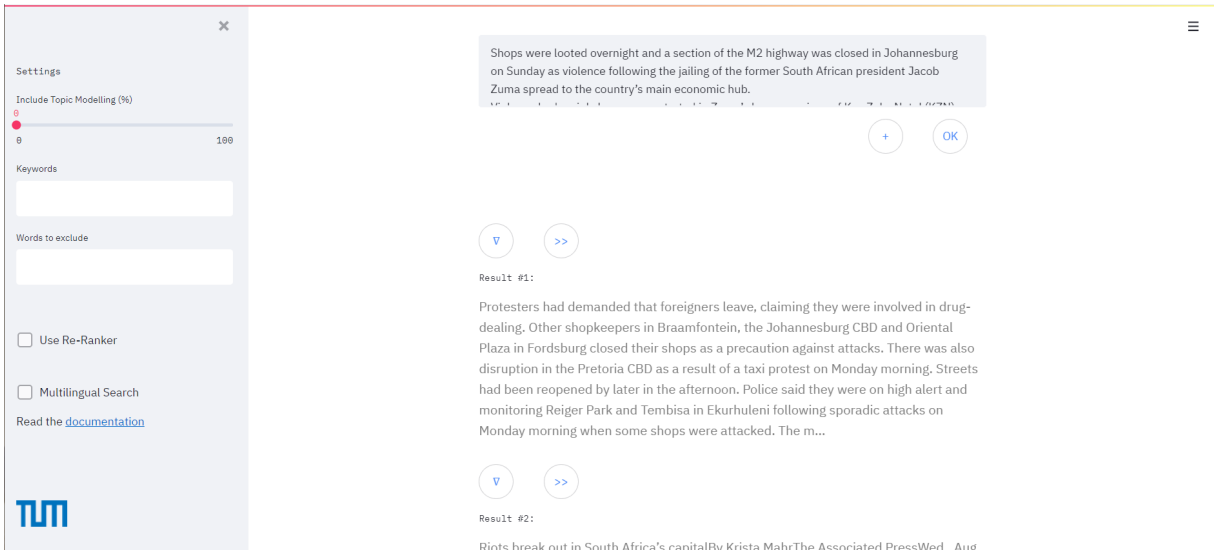



Figure 8: Screenshot of the semantic search user interface.

All of the top ten results contain news about crime or violence in Johannesburg or other South African cities. Another article very relevant to the input article – also reporting on looting of stores in Johannesburg and Pretoria – is ranked only 10th in the semantic search with DistilRoBERTa embeddings. Applying the re-ranking method will change that. The "ms-marco-MiniLM-L-6-v2" model recognizes the article to be more similar as well as more relevant to the search and puts it to the 1st rank.

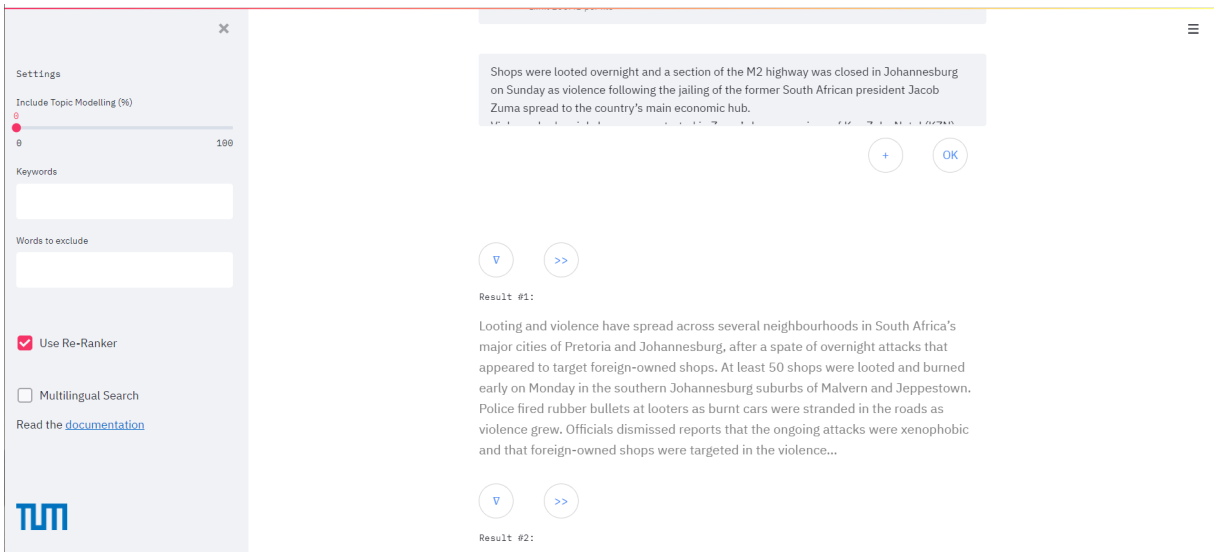



Figure 9: Re-ranked search results.

To our knowledge, there has not been a large-scale evaluation about the results of re-

ranking methods applied on Semantic Similarity tasks. We observed the re-ranked results to be more relevant in many cases, this is, however, subjective. In our experiments, re-ranking a lower amount of the top $k$ documents ($15 \leq k \leq 25$) performed better than re-ranking a larger amount of documents ($50 \leq k \leq 100$).

The results returned by Distiluse are semantically similar, yet some of them tend to exhibit more topic-based general similarity among documents semantically very close to the input article. For example, an article on measles outbreak returned another article about norovirus outbreak among the top 10 results. Nevertheless, the application of Distiluse is to add an opportunity of multilingual search once non-English data is added to the data set.

## 5.2 Topic Modeling

For topic modeling, we have included 4 models into the final proof of concept:

- Contextualized topic model with 10 topics
- Contextualized topic model with 50 topics
- BERTopic with 10 topics
- BERTopic with 50 topics

Topics for all the 2 million documents have been generated and pushed into Elasticsearch. Due to the lack of effective formal evaluation methods in the scientific literature, naked-eye comparisons of the models were made. On a set of 10 randomly chosen documents, the following results were obtained:

| | **CTM10** | **CTM50** | **BERTopic10** | **BERTopic50** |
|---|---|---|---|---|
| **Accuracy** | 70% | 90% | 40% | 70% |

Table 4: Naked-eye comparison of the 4 topic models.

As one can see from Table 4, the most precise model is **CTM50**. There is one caveat to the table though. BERTopic, unlike CTM, does not perform any sort of hard assignment. Instead, at inference time, BERTopic assigns a document to the outlier class, provided it is too far away from any of the other clusters which correspond to real topics. In particular, for the tested documents, **BERTopic10** has detected **3 outliers** (among the 60% incorrectly classified), whereas **BERTopic50** has detected **1 outlier**.

**Further Observations**

- **CTM10** is more general than **CTM50**, and, generally, a smaller amount of topics specified within the model leads to more general topics. However, **CTM50** seems to be in a sort of sweet spot, since further decrease in the number of topics leads to poorer performance while other experiments showed that more topics lead to very fine granular results, which are already covered by the context-aware embeddings.

- BERTopic may be very descriptive in terms of the correctly assigned documents. However, working with it may be problematic due to the existence of the so-called *Outlier* topic, which makes it impossible to do any further topic-based search.

**Topic-based Search**

The topic assignment for one particular document is an array of 10 probability values (in the case of 10 topics-based models) and an array of 50 probability values (in the case of 50 topics-based models). Each of these *values* correspond to the *probability of the document having the topic at that particular index*. Each topic, in turn, at one particular index is described by a multitude of words (Figure 10). For the sake of outlining how the search works, the reader may assume that we have a 10 topics model. Hence, each document is assigned a 10-value topic array. In order to identify documents with similar topics, the cosine similarity measure is computed (between this document and all the others in the database) – the intuition being that documents which have a similarly assigned distribution should "talk" roughly about the same story.

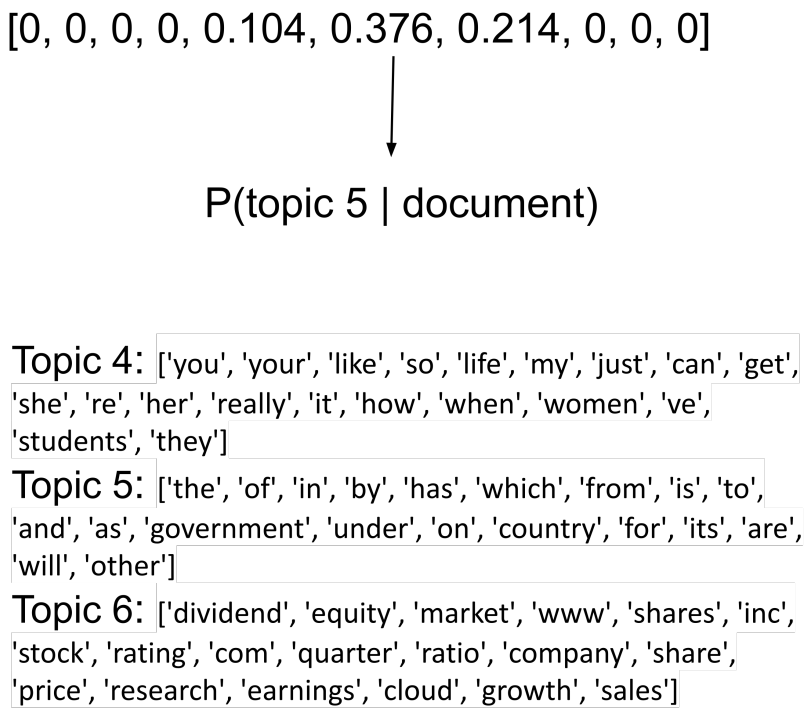



Figure 10: Example of topic assignment for one document within the data base.

**Topic-based Search Results**

For the input article about lootings in Johannesburg, the Contextualized Topic Models approach finds a topic described by violence-related words such as "explosion", "pistols" or "infantry" to fit the article the most, assigning it a probability of 17.66%. This is reflected in the search results with a search based solely on Contextualized Topic Models,

depicted in Figure 11. The top articles returned are all related to violence, the first being about “attacks”, the second one about “night raids”.

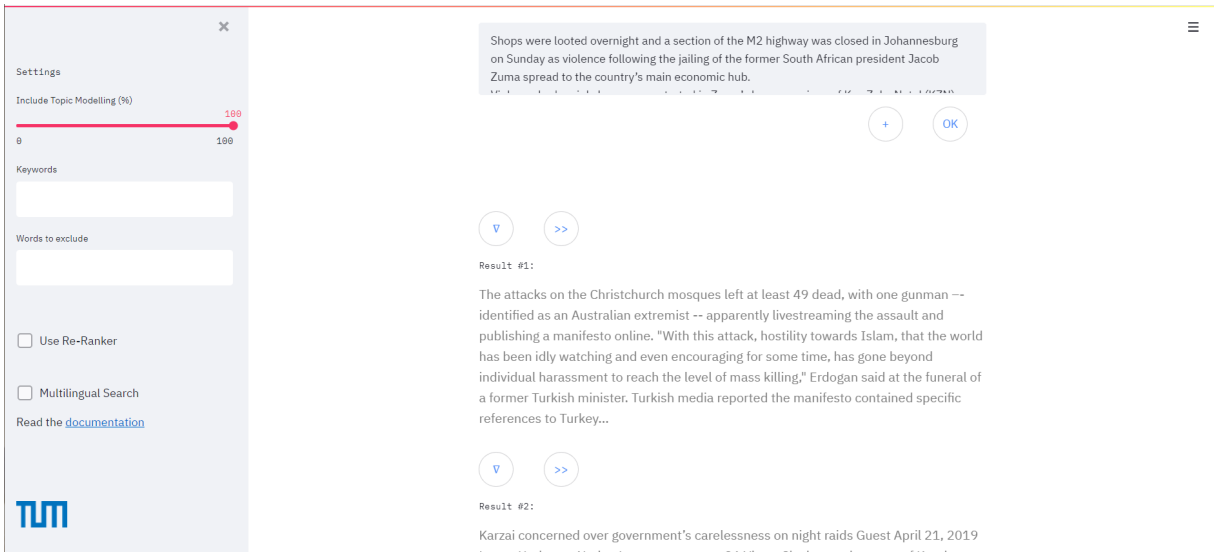



Figure 11: Semantic search fully based on Contextualized Topic Model.

## 5.3 Combined Search

When setting the slider for topic modelling to 50%, the results will contain news about rather general violence-related topics, but still have a close link to the input article. The top result in an example displayed in Figure 12 is still about “attacks”, but they are taking place in “Johannesburg”.

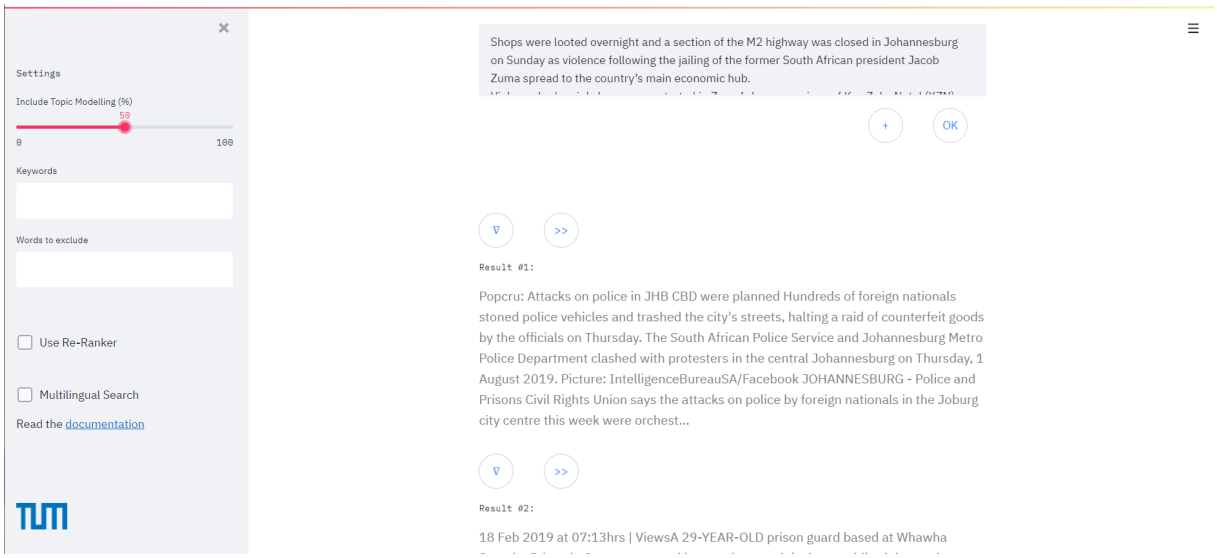



Figure 12: Example of a combined search output. Utilizing both topic modeling and context-aware embeddings.

# 6 Conclusion

We have developed a full end-to-end proof of concept semantic search engine, which is capable to understand the contextual differences between texts while utilizing cutting-edge deep learning models. All this success would not have been possible without the interdisciplinary characteristic of our team, the supportive mentorship of both sides (TUM and the industry partner), and the Leibniz Rechenzentrum which offered us access to top-notch AI infrastructure. We can conclude that this project enabled the project team to acquire hands-on experience on the latest deep learning models for natural language processing.

# 7 Outlook

To be able to apply transformer-based models in the short time frame, and because of storage limitations on the Leibniz Rechenzentrum compute cloud, we had to restrict our amount of data to 2 million articles. This amount of data is still a lot by all means, but future students could exploit even more articles having more storage capacity to be able to gain even better search results.

Furthermore, we had to find a compromise between the power of the models, i.e. accuracy and the applicability in terms of time the models need to compute embeddings of a document. With even more compute power one could have use even bigger, and therefore also more powerful models.

Additionally to the implemented functionality to include or exclude special keywords, one could also think of supplying the user a keyword extension to his or her search. Such a keyword extension could be done by finding synonyms for a certain keyword, or by exploiting thesauri. For this, we implemented one approach for finding synonyms in the lexical database WordNet. We also implemented another approach finding synonyms as well as paraphrases of a named entity by searching Wikipedia for links leading to the article about the respective named entity. It gives a list of the captions of all links leading to the article. Thus, the keyword search for “Elisabeth II” will be extended with paraphrases such as “The Queen”. But we did not include this software to our final prototype. Due to the complexity, more time would be needed to actually test the keyword extension.

Since the formal evaluation of the developed semantic search engine is not yet feasible, future work could try to find methods and approaches of how to evaluate such a context-aware search engine. However, such an evaluation might be driven by subjective decisions. Developing an objective benchmark is a clear challenge.

# References


[1] *Amazon Elasticsearch Service*. Online; accesed 6/12/2021. URL: https://aws.amazon.com/elasticsearch-service/.

[2] Federico Bianchi, Silvia Terragni, and Dirk Hovy. *Contextualized Topic Models*. Online; accessed 7/12/2021. URL: https://github.com/MilaNLProc/contextualized-topic-models.

[3] Federico Bianchi, Silvia Terragni, and Dirk Hovy. "Pre-training is a hot topic: Contextualized document embeddings improve topic coherence". In: *arXiv preprint arXiv:2004.03974* (2020).

[4] David M Blei, Andrew Y Ng, and Michael I Jordan. "Latent dirichlet allocation". In: *Journal of machine Learning research* 3.Jan (2003), pp. 993–1022.

[5] Samuel R. Bowman et al. "A large annotated corpus for learning natural language inference". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 632–642. DOI: 10.18653/v1/D15-1075. URL: https://aclanthology.org/D15-1075.

[6] Daniel Cer et al. "Universal Sentence Encoder". In: *CoRR* abs/1803.11175 (2018). arXiv: 1803.11175. URL: http://arxiv.org/abs/1803.11175.

[7] Daniel M. Cer et al. "SemEval-2017 Task 1: Semantic Textual Similarity - Multilingual and Cross-lingual Focused Evaluation". In: *CoRR* abs/1708.00055 (2017).

[8] Alexis Conneau et al. "Supervised Learning of Universal Sentence Representations from Natural Language Inference Data". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 670–680. DOI: 10.18653/v1/D17-1070. URL: https://aclanthology.org/D17-1070.

[9] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[10] Günes Erkan and Dragomir R. Radev. "LexRank: Graph-based Lexical Centrality as Salience in Text Summarization". In: *CoRR* abs/1109.2128 (2011). arXiv: 1109.2128. URL: http://arxiv.org/abs/1109.2128.

[11] Yihong Gong and Xin Liu. "Generic Text Summarization Using Relevance Measure and Latent Semantic Analysis". In: *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '01. New Orleans, Louisiana, USA: Association for Computing Machinery, 2001, 19–25. ISBN: 1581133316. DOI: 10.1145/383952.383955. URL: https://doi.org/10.1145/383952.383955.

[12] Maarten Grootendorst. *BERTopic: Leveraging BERT and c-TF-IDF to create easily interpretable topics.* Version v0.7.0. 2020. DOI: 10.5281/zenodo.4381785. URL: https://doi.org/10.5281/zenodo.4381785.

[13] *GSI's Elasticsearch k-NN Plugin*. Online; accesed 6/21/2021. URL: https://www.gsitechnology.com/sites/default/files/AppNotes/GSIT-Elasticsearch-Plugin-AppBrief.pdf.

[14] Felix Hill, Kyunghyun Cho, and Anna Korhonen. "Learning Distributed Representations of Sentences from Unlabelled Data". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, June 2016, pp. 1367–1377. DOI: 10.18653/v1/N16-1162. URL: https://aclanthology.org/N16-1162.

[15] *Investigate various implementations of ann search for vector fields*. Online; accesed 7/3/2021. URL: https://github.com/elastic/elasticsearch/issues/42326.

[16] Ryan Kiros et al. "Skip-Thought Vectors". In: *CoRR* abs/1506.06726 (2015). arXiv: 1506.06726. URL: http://arxiv.org/abs/1506.06726.

[17] Yinhan Liu et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: *CoRR* abs/1907.11692 (2019). arXiv: 1907.11692. URL: http://arxiv.org/abs/1907.11692.

[18] Derek Miller. "Leveraging BERT for Extractive Text Summarization on Lectures". In: *CoRR* abs/1906.04165 (2019). arXiv: 1906.04165. URL: http://arxiv.org/abs/1906.04165.

[19] *Pretrained Models*. Online; accesed 6/12/2021. URL: https://www.sbert.net/docs/pretrained_models.html.

[20] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *arXiv:1908.10084* (2019).

[21] Reimers, Nils and Gurevych, Iryna. *Cross-Encoder Re-Ranking*. Online; accessed 7/5/2021. URL: https://www.sbert.net/examples/applications/retrieve_rerank/README.html.

[22] Facebook AI Research. *FAISS library*. Online; accessed 7/5/2021. URL: https://faiss.ai/.

[23] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "FaceNet: A Unified Embedding for Face Recognition and Clustering". In: *CoRR* abs/1503.03832 (2015).

[24] *SentenceTransformers Documentation*. Online; accessed 7/12/2021. URL: https://www.sbert.net/.

[25] Kaitao Song et al. "MPNet: Masked and Permuted Pre-training for Language Understanding". In: *CoRR* abs/2004.09297 (2020).

[26] Akash Srivastava and Charles Sutton. "Autoencoding Variational Inference for Topic Models". English. In: *Proceedings for the 5th International Conference on Learning Representations*. 5th International Conference on Learning Representations, ICLR 2017 ; Conference date: 24-04-2017 Through 26-04-2017. Apr. 2017. URL: https://iclr.cc/archive/www/2017.html.

[27] *Streamlit*. Online; accessed 7/13/2021. URL: https://streamlit.io/.

[28] The Hugging Face Team. *DistilBERT*. Online; accessed 7/5/2021. URL: https://huggingface.co/transformers/model_doc/distilbert.html.

[29] The Hugging Face Team. *Huggingface Models*. Online; accesed 6/6/2021. URL: https://huggingface.co/models.

[30] The Hugging Face Team. *ms-marco-MiniLM-L-6-v2*. Online; accessed 7/13/2021. URL: https://huggingface.co/cross-encoder/ms-marco-MiniLM-L-6-v2.

[31] The Hugging Face Team. *OpenAI GPT2*. Online; accessed 7/5/2021. URL: https://huggingface.co/transformers/model_doc/gpt2.html.

[32] The Hugging Face Team. *Pegasus*. Online; accessed 7/5/2021. URL: https://huggingface.co/transformers/model_doc/pegasus.html.

[33] The Hugging Face Team. *T5*. Online; accessed 7/5/2021. URL: https://huggingface.co/transformers/model_doc/t5.html.

[34] Adina Williams, Nikita Nangia, and Samuel Bowman. "A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 1112–1122. DOI: 10.18653/v1/N18-1101. URL: https://aclanthology.org/N18-1101.